Real-Time 3D Reconstruction of Thin Surface Based on Laser Line Scanner

Yuan He*, Shunyi Zheng*, Fengbo Zhu and Xia Huang

School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430079, China; zhu_fb@whu.edu.cn (F.Z.); yuxiuhx@whu.edu.cn (X.H.)
* Correspondence: yuanhe@whu.edu.cn (Y.H.); syzheng@whu.edu.cn (S.Z.)

Received: 24 November 2019; Accepted: 13 January 2020; Published: 18 January 2020

Abstract: The truncated signed distance field (TSDF) has been applied as a fast, accurate, and flexible geometric fusion method in 3D reconstruction of industrial products based on a hand-held laser line scanner. However, this method has some problems for the surface reconstruction of thin products. The surface mesh will collapse to the interior of the model, resulting in some topological errors, such as overlap, intersections, or gaps. Meanwhile, the existing TSDF method ensures real-time performance through significant graphics processing unit (GPU) memory usage, which limits the scale of reconstruction scene. In this work, we propose three improvements to the existing TSDF methods, including: (i) a thin surface attribution judgment method in real-time processing that solves the problem of interference between the opposite sides of the thin surface; we distinguish measurements originating from different parts of a thin surface by the angle between the surface normal and the observation line of sight; (ii) a post-processing method to automatically detect and repair the topological errors in some areas where misjudgment of thin-surface attribution may occur; (iii) a framework that integrates the central processing unit (CPU) and GPU resources to implement our 3D reconstruction approach, which ensures real-time performance and reduces GPU memory usage. The proposed results show that this method can provide more accurate 3D reconstruction of a thin surface, which is similar to the state-of-the-art laser line scanners with 0.02 mm accuracy. In terms of performance, the algorithm can guarantee a frame rate of more than 60 frames per second (FPS) with the GPU memory footprint under 500 MB. In total, the proposed method can achieve a real-time and high-precision 3D reconstruction of a thin surface.

Keywords: real-time 3D reconstruction; laser line scanner; thin surface; parallel computing; GPU

1. Introduction

Recent years have witnessed the release of a large number of fast, accurate depth sensors. With the opening up of graphics processors to general computing, 3D reconstruction by the depth sensors has attracted new attention in the field of computer vision [1]. It has a wide range of applications in reverse engineering, 3D printing, industrial inspection, robotics, augmented reality (AR), autonomous driving, and other fields [2–4]. Traditional depth sensors, such as the Structured Light Camera, Time of Flight (ToF) Camera, and Terrestrial Laser Scanner (TLS), provide depth data as depth images or sets of unorganized points. These sensors have particular tradeoffs in terms of speed, accuracy, flexibility, and price [5]. The hand-held laser line scanner projects a laser pattern onto the surface of the object, and densely measures the surface profiles (3D curves) based on the optical triangulation method. Given its advantages in speed, accuracy, and flexibility [6], this kind of scanner, such as HandySCAN [7], HSCAN [8], and FreeScan [9] have been widely used in high-precision 3D measurement applications in industrial manufacturing.
The scanning process of 3D scanners can be divided into two parts: tracking and mapping [10]. Tracking solves the problem of multiple views registration. These scanners usually use external positioning devices or stable features in the observed scene to estimate the sensor’s position and orientation, which are usually represented by a 6-degree of freedom (DOF) transformation matrix [10]. In the mapping part, we need to align the depth data obtained from different viewpoints into a common reference frame by using the transformation matrix, and then use the depth data to build a model with a given resolution [11]; the mapping process needs to remove redundant data to the resolution of the final model and refine the observed surface continuously, since it is almost impossible to build a complete model without observing some areas twice or more, and the position estimation of the sensor is always inaccurate to some degree [6]. In other words, real-time 3D reconstruction requires incremental fusion of the depth data from different viewpoints into a single 3D representation that is continuously reparametrized [12]. This process is usually referred to as geometric fusion [6].

Fusion approaches based on different principles have been proposed. Point-based fusion methods use simple unstructured representations that closely map to the input depth data through the measured ray to merge overlapping points [13–15]. However, these methods cannot generate continuous surfaces directly. Height-map based methods use a more compact 2.5D continuous surface representation [16,17]. Although these methods effectively compress the surface, they fail to reconstruct the complex 3D structure. Implicit volumetric methods have demonstrated convincing results for incremental 3D reconstruction [18–20]. The volumetric representation is also referred to as the truncated signed distance field (TSDF), which represents the reconstructed surface by samples of a distance function continuously stored in volumetric grids (voxels) [21]. The reconstruction result can be obtained by extracting the isosurface from the zero-level set of the signed distance function [22,23]. This approach is topologically flexible and has linear complexity. With the help of massively parallel processors, it has become very popular in real-time 3D reconstruction [1,4,19,20,24,25].

In the voxel-based fusion method, Bylow et al. [26] improve the weight function to reduce the impact of unreliable measurements. Dong et al. [27] model the internal probability distribution of SDF values to eliminate outlier observations. Other methods use RGB image to improve the level of detail of surface representation, but they cannot achieve real-time performance [28–30].

Some high-precision sensor-tracking methods [26,31,32] using the features of volumetric representation are also proposed. Furthermore, some other research has focused on introducing global adjustment in real-time reconstruction to reduce the tracking drift accumulation caused by frame-to-frame registration, as well as the corresponding voxel data fast update strategy [19,33–36]. In addition to the static scene reconstruction, voxel-based methods can also be used in the 3D reconstruction of some dynamic scenes, such as reconstructing the human body surface undergoing free non-rigid motion [25,37–41].

Recently, the application of machine-learning methods in voxel-based 3D reconstruction, especially in the form of deep neural networks, has become a new trend [10]. Deep learning has achieved impressive results in processing sensor noise [42], completing scans of scenes [43], and 3D scene semantic segmentation [44–49].

Most of the fusion methods require the depth maps or sets of unorganized points as the input. Little research has been based on 3D laser curves. Tubić et al. first proposed a method of constructing TSDF using intersecting curves and a method of the pose refinement of curves by TSDF [6]. This method has been successfully exploited in the hand-held laser line 3D scanning system.

It was mentioned by [21] that TSDF is problematic for reconstructing thin surfaces, but did not provide corresponding solutions. Moreover, no relevant research has been found to address this problem. The influence of the observed surface extends behind their estimated position and interferes with the TSDF from the opposite side of the thin surface. This will cause the surface to shrink towards the interior of the model, and make some topological errors appear on the surface, such as overlap, intersections, or gaps. This effect can be mitigated by reducing truncation, but there is still a set limit for the object’s thickness. The first contribution of this paper is to propose a thin-surface attribution
judgment method to distinguish opposite surfaces and store them separately, which effectively avoids the interference of observations of thin surfaces. In some exceptional cases, real-time thin-surface attribution judgment may fail, so we have developed a post-processing method to detect and repair regions that have topological problems.

The efficiency of the underlying data structure is another critical issue in real-time 3D reconstruction. Many advanced real-time reconstruction systems have implemented different underlying data structures on graphics processors, but they have made tradeoffs in terms of speed, quality, and scale. The kinect fusion [18] achieves a real-time frame rate but has a large memory footprint since it uses the volume to represent the whole space instead of only reconstructing densely near the observed surface. The moving volume method uses the motion of a camera to stream data from the graphics processing unit (GPU) to central processing unit (CPU) to expand the space [30]. However, the current active volume structure is the same as the Kinect Fusion, and a lot of memory is required to represent the empty space. In recent years, sparse volumetric representations have been developed to compress space [20]. The main idea is to reconstruct densely only near the observed surface and discard empty space. To index the allocated data, octree [51–53], and hash table [1,19,20,24,27,34] have been successfully applied. An octree is unsuitable for graphics processors because of its low degree of parallelism. Nießner et al. proposed a voxel-hashing method for storing TSDF. The complexity is $O(1)$ for accessing, inserting, and deleting data in hash tables; this method both effectively reduces the memory footprint and has a high-speed advantage [20].

Hashing-based fusion methods can even run on mobile platforms with relatively low computing capability, such as tablets and mobile phones [24,31,35,54]. Based on the work of Nießner et al., Kahler et al. improved the storage structure of the hash table to reduce the impact of hash conflict [24]. Moreover, they have developed a hierarchical structure based on hash with few levels to improve storage efficiency further [1]. A method called structured skip list also has been proposed to increase the number of voxels that a hash table can accommodate [55]. Although these improved hashing methods alleviate the memory pressure to some degree by compressing data, they do not solve the problem from the root. GPU memory still limits the spatial extent and resolution of the voxel grid. Nießner et al. alleviate the memory pressure of GPU through bidirectional GPU–CPU streaming, but the GPU memory may still be exhausted when reconstruction scenarios are very dense. Our second contribution is to improve the data structure of voxel hashing. We only store hash tables on the GPU, and all voxels are stored on the CPU; thus, the limitation of GPU memory capacity is lifted on the scale of the reconstructed scene.

In this work, aiming at the problem of thin-surface reconstruction in the existing TSDF fusion method, we add a step of thin-surface attribute judgment in the real-time process to eliminate the mutual interference between the opposite sides of the thin surface; the angle between the surface and the observed ray is used to distinguish the TSDF observations from different parts of the thin surface. For the regions where surface attribution misjudgment may occur, we exploit a method of weak topological regions repairing to fix the possible topological errors in these regions. To solve the problem that the underlying data structure of the existing TSDF method takes up too much memory of the GPU, we design a new memory management method to reduce the memory utilization of GPU. By moving all voxel data to CPU, the memory pressure of GPU is alleviated, and the scale of reconstruction scene is further increased.

The remainder of this paper is structured as follows: Section 2.1. outlines the TSDF implementation based on 3D curves. Section 2.2. describes our improvements to thin-surface reconstruction in real-time processing; Section 2.3. explains a post-processing method of repairing weak topological regions, Section 2.4. exposes the underlying data structure for storing TSDF. The results and discussion follow in Section 3. Finally, we conclude and outline some ideas for future work in Section 4.
2. Methodology

2.1. Truncated Signed Distance Field (TSDF) Computation from 3D Curves

Our real-time 3D reconstruction approach is implemented with a highly integrated hand-held laser scanner, as shown in Figure 1. The scanner measures the 3D curves (a set of 3D points) of the object’s surface using laser triangulation [56]. A laser pattern is projected onto the measured object, and the left and right cameras capture the 2D images of the object at the same time; from the 2D images, we extract the 2D surface curves of the projection pattern reflected on the object; a set of 3D curves is computed in the sensor coordinate system using the 2D surface curves. Meanwhile, the 3D curves measured from different frames are aligned to the global coordinate system by the sensor’s position and orientation, which are continuously computed from the circular mark-points pasted on the object surface. Our 3D reconstruction algorithm integrates these arbitrary intersecting 3D curves into a single volumetric representation and refines the reconstructed surface continuously based on [6].

Figure 1. Schematic of the 3D laser scanning system; (a) components of the 3D scanner; (b) laser patterns; there are two kinds of projection patterns shown as red and blue respectively; we alternately project them onto the measured object; once one of the patterns is extracted, it will switch to the other immediately; the reflection of these two patterns will form multiple intersecting curves on the object.

TSDF constructed from 3D curves is a vector field, which is different from the classical signed distance field [21]. As shown in Figure 2a, each voxel \( v \) in a field stores the distance vector to its closest point on the reconstructed surface. The vector’s direction is parallel to the normal of the closest point on the surface. The sign of the distance indicates whether a voxel \( v \) is inside or outside the surface. If \( v \) is outside of the surface, its sign is positive, and if \( v \) is inside the surface, its sign is negative. Since voxels that are too far from the surface have little effect on the surface construction, the distance field is truncated at distance \( |d| \leq T \). Truncation \( T \) affects not only the effect of surface reconstruction but also the runtime performance and memory consumption of the algorithm. On the one hand, if \( T \) is set too large, the surface details will be smoothed out. At the same time, increasing \( T \) means processing more voxels, thus slowing down the algorithm and consuming more memory. On the other hand, if \( T \) is smaller than the level of noise, it will lead to the appearance of isolated patches around the reconstructed surface. Therefore, it is generally appropriate to set \( T \) to the minimum value larger than the noise level of the apparatus, which is 0.9 mm for the scanner used in our experiment. Truncation \( T \) is also the leading cause of thin-surface problems in TSDF. How \( T \) affects thin-surface reconstruction will be mentioned in the next section.

Reconstruction of surfaces is based on such a fundamental principle that two intersecting lines can determine a plane. Then the local reconstructed surface patch can be approximated in the neighborhood of the intersection points of all the curves on the surface. Figure 2b illustrates the principle of reconstruction in a simple case where two curves intersect. Assuming that \( p_1 \) and \( p_2 \) are the closest points to voxel \( v \) on curves \( c_1 \) and \( c_2 \), respectively, \( v_1 \) and \( v_2 \) are two vectors from the voxel center point \( p_v \) to \( p_1 \) and \( p_2 \), respectively. The tangents of the curves at \( p_1 \) and \( p_2 \) have been computed...
in advance and are expressed as $t_1$ and $t_2$, respectively. Since the estimated normal $n$ of the tangent plane to the surface should be perpendicular to $t_1$ and $t_2$, $n$ should be the cross-product of $t_1$ and $t_2$. The distance $d$ from $p_v$ to the closest surface can be approximated as the average of the projections of $v_1$ and $v_2$ on $n$. Therefore, the value $f$ of the distance vector field at $v$ is $dn$. A full description of how to locate the closest points on a curve and how to calculate the tangent at the closest point is beyond the scope of this paper. Interested readers can find relevant content in [6].

![Figure 2](image)

**Figure 2.** Schematic of truncated signed distance field (TSDF) constructed from intersecting curves; (a) a 2D slice of the TSDF computed on a regular voxel grid; each arrow indicates the distance vector of a voxel; gradients of color encode the signs and values of the distances; (b) a simple example of computing the value $f$ of a vector field.

More formally, let $n$ curves of $c_1, \ldots, c_n$ pass through the local surface patch; $p_1, \ldots, p_n$ are their respective closest points to voxel $v$ and $t_1, \ldots, t_n$ are the tangents of $p_1, \ldots, p_n$ respectively. Then the distance from the central point $p_v$ of voxel $v$ to the surface $\hat{S}$ can be obtained as the average value of the projected distance vectors on the estimated normal, i.e.,:

$$d(v, \hat{S}) = \frac{1}{n} \sum_{i=1}^{n} (p_i - p_v, n)$$

where $p_i$ is the closest point to $p_v$ on the curve $c_i$, and $n$ is the estimated normal of the surface.

Since the normal $n$ is perpendicular to all tangents $t_1, \ldots, t_n$, the following equation can be obtained:

$$\frac{1}{n} \sum_{i=1}^{n} t_i t_i^T n = Cn = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$

$C$ is the covariance matrix of tangents, and the eigenvector associated with the minimum eigenvalue of $C$ is the solution to $n$. Then, the value of vector field $f: \mathbb{R}^3 \to \mathbb{R}^3$ at $v$ is:

$$f(v) = d(v, \hat{S})n$$

Therefore, the closest surface point $p_c$ of voxel $v$ can be obtained by:

$$p_c = p_v + f(v), p_c \in \hat{S}$$

A curve can be considered as a linear interpolation of measure points $\{p_1, \ldots, p_n\}$ represented by a set of line segments $c = \{l_1, \ldots, l_{n-1}\}$, where $l_i = \frac{p_i - p_{i+1}}{p_i - p_{i+1}}$. In online scanning, the measured surface curves are used to update the covariance matrix $C$ and distance $d$ of all neighboring voxels incrementally. Each curve only affects voxels within a certain vicinity. As shown in Figure 3, the voxels affected by each line segment $p_i p_{i+1}$ on a curve can form an approximate cylinder (called a fundamental cell) whose
radius is determined by the truncation distance, which is called the envelope size in [6]. The envelope here serves the same purpose as the truncation mentioned earlier. Curves only affect voxels in the envelope and do not affect voxels beyond it. The following low-pass weight function prevents the discontinuity of the field over the envelope boundary:

$$\omega(d) = e^{-\frac{d^2}{\sigma^2}}$$

$$\sigma$$ stands for the size of the envelope. Increasing the envelope size can make the reconstructed surface smoother, but it will also lose any sharp features. Conversely, reducing the envelope size makes the surface noisy. Tubić et al. recommended $\frac{\sqrt{3}}{2} \cdot \delta$ as the envelope size as in [6], and $\delta$ as the voxel size.

After stopping scanning, a marching cubes method can be used to generate mesh models. The reconstructed surface can be obtained by extracting the isosurface from the zero-level set of Sensors 2020 is 0.9 mm. Tubić’s output has a large area of data missing, and its distance (one-sided Hausdorff size) of the model is 0.5 mm, and the size of the truncation (also known as the fundamental envelope) cannot reconstruct the surface of very thin objects. We use a laser line scanner to measure the 3D curves from the surface of a sheet metal part with a thickness of 0.8 mm and use method of Tubić et al. to reconstruct the surface mesh model of this object, which is shown in Figure 4a. The resolution (voxel size) of the model is 0.5 mm, and the size of the truncation (also known as the fundamental envelope) is 0.9 mm. Tubić’s output has a large area of data missing, and its distance (one-sided Hausdorff) to the

$$\omega = \sum_{i=1}^{N} \omega_i$$

The average distance-vector $\bar{v}$ is projected onto the normal $n$ calculated from the covariance matrix $C$ to obtain the value of TSDF at $v$:

$$f(v) = \frac{\langle \bar{v}, n \rangle}{\|n\|^2}$$

$f(v)$ is only calculated via real-time rendering and post-processing to generate surface meshes. The reconstructed surface can be obtained by extracting the isosurface from the zero-level set of $f(v)$. In the online process, raycasting is used to render the reconstructed surface from the current view. After stopping scanning, a marching cubes method can be used to generate mesh models.

The above content briefly introduces method of Tubić et al. [6] of generating a surface model from a set of intersecting 3D curves. This method is topologically flexible and has linear complexity, but it cannot reconstruct the surface of very thin objects. We use a laser line scanner to measure the 3D curves from the surface of a sheet metal part with a thickness of 0.8 mm and use method of Tubić et al. to reconstruct the surface mesh model of this object, which is shown in Figure 4a. The resolution (voxel size) of the model is 0.5 mm, and the size of the truncation (also known as the fundamental envelope) is 0.9 mm. Tubić’s output has a large area of data missing, and its distance (one-sided Hausdorff) to the
computer-aided design (CAD) model is greater than 0.2mm (Figure 4b). We optimize the robustness of Tubić’s method to thin objects. With the same resolution and truncation, the optimized results not only guarantee the completeness of data (Figure 4c) but also guarantee its maximum distance (one-sided Hausdorff) to the CAD model within 0.2 mm (Figure 4d). The proposed thin-surface optimization of Tubić’s method is described in Sections 2.2 and 2.3. Figure 5 shows the whole pipeline of the proposed algorithm. The algorithm employs a hash table to organize the underlying voxel data. The content related to the underlying data structure can be found in Section 2.4. The whole procedure of the proposed 3D reconstruction algorithm can be summarized as the following seven steps:

1. Voxels Allocation: after receiving a frame of 3D curves data, we compute the \( p_v \) of all voxels that fall within the fundamental envelopes of the curves and find the corresponding voxel block of \( p_v \) in the hash table. If no block is found, we allocate a new block for it on the heap;

2. TSDF Computation: compute the TSDF observations of these voxels (including the closest point \( p_c \) on the curve and tangent \( t_c \) on \( p_c \));

3. Voxels Indexing: use \( p_v \) to index the address of voxel \( v \) in hash table;

4. Voxel Integration: before integrating the new TSDF, it is necessary to determine whether the new one is on the same side as the surface already encoded by \( v \). If so, the new TSDF is integrated to voxel \( v \) by Equations (6)–(8) (updating weight \( \omega \), matrix \( C \) and distance vector \( \tilde{v} \)). Conversely, we find the inverse voxel \( v' \) on the other side of the thin surface and update \( v' \). The method of determining which side of the thin surface the current observation belongs to will be elaborated in the next section;

5. Normal filtering: the surface normal plays a vital role in the judgment of thin-surface attribution, and also affects the real-time display quality. Therefore, we need to denoise it to make it reliable. Firstly, the surface point \( p_c \) (Equation (4)) and normal \( n \) (through a singular value decomposition (SVD) of matrix \( C \)) encoded by voxel \( v \) are computed. Then the neighborhood points of \( v \) are found in the hash table, and \( n \) is recomputed with the surface points and normals of \( v \)'s neighbors. The related content of normal filtering will also be described in the next section.

6. Weak Topological Regions Repair: although we have distinguished the opposite sides of the observed thin surface when integrating voxels, it is difficult to distinguish them in some areas with complex topology. When the user stops scanning, we adopt a post-processing method to detect and repair the topological errors in these areas automatically.

7. Isosurface Extraction: after repairing the topological errors in TSDF, we use the marching cubes algorithm to extract the isosurface and generate a mesh model.

### 2.2. TSDF for Thin Surface

As mentioned earlier, TSDF has a disadvantage for reconstructing thin surfaces. The distance field extends a band, i.e., truncation, both inside and outside the observed surface. When the surface’s thickness is less than the width of the truncation band, the values of distance fields that encode opposite sides of the surface will interfere with each other, leading to the effect that the rear surface cannot be recovered from the fields. Figure 6a shows a 2D slice of a corrupted thin surface. Black solid lines represent the real surface of a thin object, and each solid arrow indicates a value in the distance field. The truncation distance of the field is two times of voxel size, and the interfered voxels are colored gray; the darker their color, the higher the impact. Unaffected lattices are white because they are too far from the other side of the surface. These arrows of interfered voxels do not point to the real surface, but to multiple false surfaces that are represented by red dotted lines. Therefore, the correct method is to store two vectors (dotted arrows) in a gray lattice, as shown in Figure 6b. That is to say, each voxel in the overlapping region should encode two tangent planes that are on different real surfaces. It is first necessary to determine which side of the thin surface each TSDF observation originates from during online scanning; we call this process thin-surfaces attribution judgment.
When one side of a thin surface is observed, the opposite side can be occluded. The view direction of the current observations implies that the current observations belong to one side of the thin surface. Moreover, to obtain the most accurate observations, the direction of the sensor’s view usually needs to be approximately perpendicular to the object’s surface. Therefore, the correlation between the direction of the sensor’s view and the attribution of the thin surface is stronger. Thus, we store the line of sight of each voxel to be approximately perpendicular to the object’s surface. Therefore, the correlation between the direction of the sensor’s view and the attribution of the thin surface is stronger. Thus, we store the line of sight of each voxel as follows:

\[
\text{sight}(v) = \begin{cases} 
  v - v_p & \text{if } \text{sight}(v) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\
  \text{sight}(v) & \text{if } \langle v - v_p, n_v \rangle \leq \| \text{sight}(v), n_v \| \\
  v - v_p & \text{if } \langle v - v_p, n_v \rangle > \| \text{sight}(v), n_v \| 
\end{cases}
\]  

(10)
here, \( n_v \) denotes the normal of the surface encoded by voxel \( v \), and \( v_p \) represents the current viewpoint. Each normal \( n_v \) computed from the covariance matrix \( C \) in Equation (6) must be oriented consistently towards the viewpoint \( v_p \) (so that the angle between \( n_v \) and the line of sight is less than \( \frac{\pi}{2} \)). Equation (10) guarantees that the angle between the line of sight and the surface normal is the smallest so that the sign of the normal \( n_v \), determined by the line of sight is reliable.

\[
\text{sign}(v) = \begin{cases} 
1 & \text{if } \langle n_{\text{dis}}, v \rangle \leq 0 \\
-1 & \text{if } \langle n_{\text{dis}}, v \rangle > 0
\end{cases}
\]

(12)

Figure 6. Corrupted thin surface vs. the correct thin surface; (a) interfered TSDF and corrupted surfaces; (b) correct TSDF.

In the online scanning process, there are two cases of the viewpoint orientation:

1. One Side Visible: when the line of sight is oriented toward one side of the thin surface, the sensor can only measure the current side and cannot measure the opposite side because the dominant side occludes it.
2. Both Sides Visible: when the line of sight is approximately parallel to a thin surface and perpendicular to the edge of a thin surface, the thin surface on the opposite sides near the edge can be observed.

The first case (One Side Visible) must be addressed to prevent the thin surface from being totally corrupted during online scanning. The second case (Both Sides Visible) will only affect the reconstruction quality near the edge of the thin surface. Moreover, it requires a more complex mechanism, and its computation cost is very high. Therefore, we put this in the post-processing process, whose specific details are described in Section 2.3.

The signs of TSDF indicate whether a voxel is inside or outside an object. We use the following procedure to determine the signs of TSDF. Firstly, the orientation of the normal \( n_v \) is adjusted to be consistent with the distance vector \( \tilde{v} \) (Equation (7)) to get the vector \( n_{\text{dis}} \), that is, \( n_{\text{dis}} \) needs to satisfy the equation:

\[ n_{\text{dis}} \cdot \tilde{v} > 0 \]

(11)

Then the sign \( S_v \) at voxel \( v \) is determined by the dot product of \( n_{\text{dis}} \) and the line of sight \( V_v \) stored in \( v \):

\[
\text{sign}(v) = \begin{cases} 
1 & \text{if } \langle n_{\text{dis}}, V_v \rangle \leq 0 \\
-1 & \text{if } \langle n_{\text{dis}}, V_v \rangle > 0
\end{cases}
\]

(12)

With the change of signs, we can quickly judge the surface attribution online. When scanning from one side of a thin surface to the other, the current direction of view \( V_v \) will deviate significantly from the previous \( V_v \) stored in voxel \( v \). If the normal \( n_v \) is oriented to the current viewpoint, the sign
Perform the approach proposed in [6] to estimate the first normal \( n_f \) and compute the reliability measure \( \lambda_r \) of \( n_f \):

1. For each voxel \( \mathbf{v} \) in the field, estimate the normal \( \mathbf{n}_s \) based on an anisotropic neighborhood;
2. Update the normals by combining the first normal \( n_f \) with \( n_s \) under the influence of coefficient \( \lambda_r \) until either convergence or a maximum number of iterations is reached.

**Figure 7.** The plot of the sign \( S \) varying with the angle \( \alpha \) between vector \( \mathbf{n}_{\text{dis}} \) and line of sight \( \mathbf{V}_v \). As shown in Figure 7, near the critical point \( (\alpha = \frac{\pi}{2}) \), a slight change in the angle will cause the sign \( S \) to be flipped. Small perturbations of data (related to the accuracy of apparatus) will lead to a certain degree of deviation in normal estimation. When the angle between the line of sight and the surface normal \( \alpha \approx \frac{\pi}{2} \), the error of normal estimation will lead to an inaccurate sign of TSDF, which will make the misjudgment of the thin-surface attribution. Meanwhile, the reliability of the measures obtained in this case is also relatively low. Therefore, a current measurement reliability judgment is added to the process to determine whether the thin-surface attribution judgment can be turned on. When the line of sight and the surface normal are almost perpendicular \( |(\mathbf{n}_{\text{dis}}, \mathbf{V}_v)| \leq 0.05 \), the current measurement is judged to be unreliable and discarded. Conversely, the thin-surface attribution judgment is turned on, and the current observation is integrated into the corresponding voxel determined by the thin-surface attribution judgment.

\( S'_{\mathbf{v}} \) determined by \( \mathbf{n}'_{\mathbf{v}} \) and \( \mathbf{V}'_{\mathbf{v}} \) is contrary to the sign \( S_{\mathbf{v}} \) previously stored in \( \mathbf{v} \). Meanwhile, if the signs \( S'_{\mathbf{v}} \) and \( S_{\mathbf{v}} \) are the same, the current measurement is on the same side as the previous cumulative measurement. If the current observation is judged to belong to the other side of the thin surface, then a new voxel \( \mathbf{v}' \) is allocated on the heap, the observed value is integrated into \( \mathbf{v}' \), and the address of the new voxel \( \mathbf{v}' \) is stored in the original voxel \( \mathbf{v} \). When querying the value of voxels in TSDF, we need to know not only the coordinates of the query point but also the line-of-sight at this moment to determine whether the voxels we are looking for are \( \mathbf{v} \) or \( \mathbf{v}' \). Section 2.4 provides more details about the data structure.

Figure 7 depicts the plot of the sign \( S \) varying with the angle \( \alpha \) between vector \( \mathbf{n}_{\text{dis}} \) and line of sight \( \mathbf{V}_v \). As shown in Figure 7, near the critical point \( (\alpha = \frac{\pi}{2}) \), a slight change in the angle will cause the sign \( S \) to be flipped. Small perturbations of data (related to the accuracy of apparatus) will lead to a certain degree of deviation in normal estimation. When the angle between the line of sight and the surface normal \( \alpha \approx \frac{\pi}{2} \), the error of normal estimation will lead to an inaccurate sign of TSDF, which will make the misjudgment of the thin-surface attribution. Meanwhile, the reliability of the measures obtained in this case is also relatively low. Therefore, a current measurement reliability judgment is added to the process to determine whether the thin-surface attribution judgment can be turned on. When the line of sight and the surface normal are almost perpendicular \( |(\mathbf{n}_{\text{dis}}, \mathbf{V}_v)| \leq 0.05 \), the current measurement is judged to be unreliable and discarded. Conversely, the thin-surface attribution judgment is turned on, and the current observation is integrated into the corresponding voxel determined by the thin-surface attribution judgment.
When estimating the first normal \( \mathbf{n}_i \) of voxel \( \mathbf{v} \), the three eigenvalues \( \lambda_1 > \lambda_2 > \lambda_3 \) of the covariance matrix \( \mathbf{C} \) in Equation (6) are used to evaluate the reliability coefficient \( \lambda \) of \( \mathbf{n}_i \). Let

\[
\lambda_i = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3},
\]

Pauly et al. \cite{Pauly2005} defined this coefficient as the surface variation. This measures the ratio of variance in the normal direction to the total variance and can be used to express the reliability of normal estimation. This quantity ranges between 0 (flat case) and 1/3 (anisotropic case). Add the following weight function to the surface. The weights are more significant when there are more observations, or the distance is closer, whereas the weights are smaller otherwise.

\[
\lambda_i(\lambda) = \begin{cases} 
1 & \text{if } \lambda_i < \rho \\
-\varphi \ln(\lambda_i) - \psi & \text{if } \lambda_i \geq \rho, \text{ and } -\varphi \ln(\lambda_i) - \psi \geq 0 \\
0 & \text{if } -\varphi \ln(\lambda_i) - \psi < 0
\end{cases}
\]

The parameters \( \rho, \varphi, \) and \( \psi \) can be determined experimentally. Here, we use \( \rho = 0.01, \varphi = 0.1, \psi = 0.15 \). The purpose of introducing coefficient \( \lambda_i \) is to preserve the features of the first normals as much as possible and to prevent over-smoothing.

Let \( \mathbf{v} \) be the voxel at which we need to re-estimate the normal; \( \mathbf{p} \) represents the point closest to \( \mathbf{v} \) on the surface, and \( \mathbf{n} \) represents the first normal. \( \mathbf{p}_i \) is the surface point at voxel \( \mathbf{v}_i \) (\( \mathbf{v}_i \in \mathcal{N}_v \)) in the neighborhood of voxel \( \mathbf{v} \), and \( \mathbf{n}_i \) is first normal at \( \mathbf{v}_i \). \( \mathcal{N}_v \) is under a neighborhood size \( \sigma_p \) specified by users. A covariance matrix \( \mathbf{C}_n \) can be constructed from all normals \( \mathbf{n}_i \) in the neighborhood of voxel \( \mathbf{v} \) as follows:

\[
\mathbf{C}_n = \frac{1}{N} \sum_{i=1}^{N} \omega_i \left( \mathbf{n}_i - \mathbf{n}_m \right) \left( \mathbf{n}_i - \mathbf{n}_m \right)^T
\]

Figure 8. The comparison between the normal computed by Tubić et al. \cite{6} and that filtered by our method; (a) normal before filtering; (b) normal after filtering.
where $\sigma_p$ is the size of the neighborhood, and the angle parameter $\sigma_n$ measures the similarity of the neighborhood normals. $p_m$ and $n_m$ are the average of $p_i$ and $n_i$, respectively. The eigenvector associated with the minimum eigenvalue of $C_n$ is the new estimate $n_s$ at voxel $v$, since it corresponds to the direction of the smallest variance of the neighborhood normal.

The final normal estimate is updated iteratively by the following formula:

$$n \propto \lambda r_n f + (1 - \lambda_r)n_s$$  \hspace{1cm} (16)

where $\propto$ denotes the normalization of the length such that $n$ is a unit vector. Whenever voxel $v$ gets a new $n_f$, a new normal estimate $n_i$ is obtained by the above method until the deviation between $n_i$ and $n_{i-1}$ is less than a certain threshold, or the number of updated times reaches a limit. As a local filtering method, our method has a low computational cost. Its time-consuming is related to the neighborhood’s size; a large neighborhood may increase the time consumption but will also lead to over-smoothing. The $3 \times 3 \times 3$ neighborhood proved sufficient.

### 2.3. Repairing Weak Topological Regions

As mentioned in the previous section, when the line-of-sight is almost parallel to the thin surface, the thin-surface attribution judgment may fail, so the distance vector in the vector field cannot point to the correct position. The reconstructed surface mesh data from the vector field will deviate significantly from the object’s actual surface and will result in topological errors such as overlap, intersections, or gaps. Without prior knowledge, it is impossible to get the desired surface shape by merely repairing the topological errors in the surface mesh. Therefore, the distance vectors in TSDF must be repaired to restore the locus of the real surface. We need to analyze TSDF to find areas that may have topological problems and then repair the values of these areas.

#### 2.3.1. Detecting Weak Topological Regions

Our method of detecting weak topological regions in TSDF is inspired by Sharf’s method [59]. Sharf’s approach for repairing weak topological regions is to find the critical point of the signed distance function (that is the point where the gradient disappears), and then specify the locus of the correct surface at the critical point by user interaction. Sharf finds discrete critical points, not contiguous regions. Many points with topological errors may still be undetected near these critical points, but Sharf compensates for this problem through specific user interface interactions, which violates our original intention of automatically detecting and repairing weak topological areas. Here, we make some adjustments to Sharf’s method so that the weak areas we find are continuous and complete. Specifically, the methods of detecting weak regions can be summarized as follows:

1. Find the critical points: find all voxels $v_i \in N_v$ in a neighborhood with $v$ as the center. $v_i$ is divided into two sets, one $f(v_i) > f(v)$ is defined as positive, and one $f(v_i) \leq f(v)$ is defined as negative. Within each set, adjacent voxels are joined together to obtain several connected groups $\Gamma_i^\pm$, as shown in Figure 10. If the number of connected groups of $v$ is 2, then $v$ is not the critical point. If the number of connection groups of $v$ is 1 or $\geq 2$, then $v$ is a critical point;

2. Generate weak topological regions: take each critical point $P_c$ as a seed point, find connected component $P_g$ (a region of adjacent voxels) of $P_c$ within a specified range (such as two times of the voxel size), and the dot product of the normal less than a certain threshold $T_{dot}$. Figure 9 shows weak regions detected by our method. This model has many stiffeners with a thickness of about 1 mm. When the truncation is also set to 1 mm, the surface model is prone to generate topological errors in the location of these stiffeners. The proposed method can detect these weak topological regions (the red areas in the figure), and the method of repairing topology errors will be described in the next section.
Therefore, the task of repairing weak areas can be transformed into the process of finding reliable points and inferring the value of unreliable points from reliable points. Values of voxels with positive signs can be used as input to infer the values of voxels with negative signs. Conversely, for concave surfaces, the TSDF values of positive voxels can be inferred from negative voxels. The number of positive and negative connection groups in weak regions can be used to judge whether a surface is convex or concave. When the number of positive connection groups is larger than that of the negative connection groups, the surface is convex, whereas the surface is concave otherwise.

2.3.2. Repairing Weak Topological Regions

Figure 9b shows that some points in the weak region are unaffected by the thin-surface effect and can fall on the actual surface. By a reasonable parameter configuration (including voxel size and truncation width), we can ensure that there will always be some reliable points in the weak region. Therefore, the task of repairing weak areas can be transformed into the process of finding reliable points and inferring the value of unreliable points from reliable points.

As shown in Figure 11, voxels inside convex surfaces are easily affected by observations on both sides of the thin wall, while voxels outside are unaffected. Therefore, for convex surfaces, the TSDF values of voxels with positive signs can be used as input to infer the values of voxels with negative signs. Conversely, for concave surfaces, the TSDF values of positive voxels can be inferred from negative voxels. The number of positive and negative connection groups in weak regions can be used to judge whether a surface is convex or concave. When the number of positive connection groups is larger than that of the negative connection groups, the surface is convex, whereas the surface is concave otherwise.
Figure 11. The distribution of reliable and unreliable points in weak regions of convex and concave surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.

After identifying reliable voxels, we need to infer the actual surface from them. \( p_i \) and \( n_i \) are the surface points and normals of reliable voxels, respectively. Our goal is to construct a directed distance surfaces; reliable voxel centers are blue, unreliable voxel centers are red; (a) distribution of weak regions for convex surfaces; (b) distribution of weak regions for concave surfaces.
Therefore, the equation system $Ax^T = b^T$ can be written as:

$$
\begin{pmatrix}
\alpha_1 & \beta_1 \\
\vdots & \vdots \\
\alpha_N & \beta_N
\end{pmatrix} =
\begin{pmatrix}
0 \\
\vdots \\
0
\end{pmatrix}
$$

(20)

A single matrix block $(d + 1) \times (d + 1)$ of coefficient $A$ can be written as:

$$
\begin{pmatrix}
\phi'(\|x - p_i\|) f_i(x_1) & \ldots & \phi'(\|x - p_i\|) f_i(x_N) \\
\vdots & \ddots & \vdots \\
\phi'(\|x - p_i\|) f_i(x_1) & \ldots & \phi'(\|x - p_i\|) f_i(x_N)
\end{pmatrix}
\begin{pmatrix}
\frac{\phi'(\|x - p_i\|)}{\|x - p_i\|} (x_{j,1} - p_{j,1}) \\
\vdots \\
\frac{\phi'(\|x - p_i\|)}{\|x - p_i\|} (x_{j,d} - p_{j,d})
\end{pmatrix}
= \begin{pmatrix}
\phi'(\|x - p_i\|) f_i(x_1) \\
\vdots \\
\phi'(\|x - p_i\|) f_i(x_N)
\end{pmatrix}
= \begin{pmatrix}
\alpha_i \\
\vdots \\
\alpha_i
\end{pmatrix}
$$

(21)

After calculating the weights of $\alpha_i$ and $\beta_i$, the distance from any spatial position $x$ to the implicit surface can be calculated using Formula (17), and the gradient can be calculated by Formula (18). The normal of the implicit surface encoded by $x$ can be obtained by normalizing the gradient. Therefore, the distance vector and normal of every unreliable voxel in the weak region of topology can be recomputed from the HRBF.

### 2.4. Data Structure

We defined the following structure to store the TSDF properties at each voxel:

```c
struct Voxel {
    float sight [3];
    float C [6];
    float distance [3];
    float weight;
    unsigned int pointer;
    unsigned int address;
};
```

Each voxel stores the sight, the covariance matrix $C$ defined in Equation (6) (Since matrix $C$ is a symmetric matrix, only six independent elements in the corresponding upper triangular matrix are stored), the distance defined in Equation (7), the accumulated weight defined in Equation (8), the pointer to the voxel belongs to the other side of the thin surface and the address of the surface point. The normal is stored in the surface point, which can be used to judge the surface attribution and interactive visualization. A certain number of voxels (the default is $8^3$) are merged to store, which is called a block. All voxels in a block are stored continuously in the order of $x$-$y$-$z$, and the coordinates of each voxel center can be obtained from the coordinates of the block.

All voxel blocks are organized by a hash table implemented on GPU [20]. The hash table stores hash entities, each of which points to an allocated voxel block. Their corresponding relationship is established by converting the spatial coordinates into hash values through a hash function. Nießner provided a complete method for searching, inserting, and deleting data in hash tables and a mechanism for resolving hash collisions. Since the hash table is implemented on the GPU, it can index all allocated data in parallel and thus has a considerable speed advantage.

Unlike Nießner, which stores some voxel blocks on the GPU and others on the CPU, we directly store all voxel blocks on the CPU to reduce the memory footprint of the CPU. Only hash tables are stored on the GPU, and each entity in the hash table contains a pointer to the voxel blocks allocated on the CPU. Once we have inserted a new entry into the hash table, we allocate a voxel block to the heap.
memory on the CPU. The heap is a dynamic 2D array that supports dynamic scaling. It is divided into contiguous blocks (mapped to the size of voxel blocks). Once allocated heap memory is exhausted, a fixed-length array is appended to the back of the 2D array. Since the array is accessed in parallel, synchronization is necessary, by incrementing the pointer of the array using atomic operation.

Before integrating the new TSDFs, all voxel blocks must be allocated that fall on the input 3D curves, and are also within the fundamental envelope of the curves. For each line segment $\overline{p_i,p_{i+1}}$ on the curve, we first calculate the bounding box determined by $\overline{p_i,p_{i+1}}$, and then for all the voxels in the bounding box, we determine whether it falls into the fundamental envelope of $\overline{p_i,p_{i+1}}$. If it is so, we find the corresponding entity of it in the hash table. If no entity is found, we insert an entity in the hash table and allocate a voxel block on the heap.

In addition to the voxel block array, the CPU also stores an inverse voxel block array and an array of neighborhood lookup tables, which are also maintained by dynamic 2D arrays. Inverse voxel blocks are used to store the opposite sides of a thin surface. We also store each element in the inverse voxel blocks as a Voxel struct. Before integrating each new TSDF, we need to determine whether it belongs to the same side of the surface that has been encoded in the non-inverse voxel $v$. If it is so, we integrate the new value to $v$, and vice versa integrate it to the inverted voxel $v'$ pointed by $v$.pointer. If the value of $v$.pointer is empty, a new voxel $v'$ is allocated on the heap of the inverse voxel blocks, and the address of $v'$ is assigned to $v$.pointer. Then we compute the new TSDF (the closest point $p_c$ on the curve and tangent $t$ at $p_i$) and integrate the new value to its corresponding voxel. When assigning a new inverse voxel $v'$, we lock voxel $v$ atomically to prevent other threads from writing to it at the same time.

### Algorithm 1. Computation of the TSDF from a set of curves.

**Input:** Set of curves $c_i = [p_{i,1}, \ldots, p_{i,n_i}]$, $i\in[1,N]$; tangent $t_{i,j}$ at each point $p_{i,j}$, $j\in[1,n]$.

**Output:** TSDF: $\mathbb{R}^3 \rightarrow \mathbb{R}^3$.

1. Initialize matrix $C$, distance vector $\bar{v}$ and weight $\omega$ in all voxels to zero.
2. for each curve $i\in[1,N]$ (number of curves) do
3.     for each point $j\in[1,n_i]$ (number of points on the curves) do
4.         Compute the bounding box of the line segment $\overline{p_{i,j},p_{i,j+1}}$.
5.     for all voxels $p_v$ (the central point of the voxel) in the bounding box do
6.         if $p_v$ is in the fundamental envelope associated with $\overline{p_{i,j},p_{i,j+1}}$ then
7.             Compute the closest point $p_c$ to $p_v$ on the curve;
8.             Compute the tangent $t_c$ on the point $p_c$;
9.             Find the voxel $v$ associated $p_v$ in the hash table;
10.            Find 26 neighborhoods of the voxel block where $v$ resides;
11.            Update the elements corresponding to $v$’s block in the neighborhood lookup table;
12.            if current TSDF observations $(p_c$ and $t_c$) belong to the surface encoded by $v$ then
13.                Update matrix $C$ at voxel $v$ (Equation (6));
14.                Update the sum of distance vectors $\bar{v}$ with $p_{c,v}$ (Equation (7));
15.                Update the sum of weight $\omega$ at voxel $v$ (Equation (8));
16.                Compute the surface point $p_s$ (Equation (4)) and normal $n$ (by SVD decomposition of matrix $C$);
17.                Find the neighborhoods of $v$;
18.                Recompute $n$ (Equation (16)) with the normals of the neighborhood points;
19.            else
20.                Find inverse voxel $v'$;
21.                Update matrix $C$ at voxel $v'$ (Equation (6));
22.                Update the sum of distance vectors $\bar{v}$ with $p_{c,v'}$ (Equation (7));
23.                Update the sum of weight $\omega$ at voxel $v'$ (Equation (8));
24.                Compute the surface point $p_s$ (Equation (4)) and normal $n'$ (by SVD decomposition of matrix $C$);
25.                Find the neighborhoods of $v'$;
26.                Recompute $n'$ (Equation (16)) with the normals of the neighborhood points;
To search voxels neighborhood information conveniently and avoid frequent data transmission between CPU and GPU, we maintain a neighborhood lookup table of all blocks on CPU. Neighborhood lookup table plays a vital role in normal filtering operation. It stores the pointers of 26 neighborhood voxel blocks for all voxel blocks. For each voxel \( v \), if it is located at the center of the voxel block \( B \), the 26 neighboring voxels adjacent to it can be found in the block \( B \); its neighbors can be found by simple subscript addition and subtraction since the elements in a voxel block are stored continuously. If it is on the edge of the voxel block \( B \), the neighboring voxels need to be found in the voxel blocks adjacent to \( B \). Neighborhood lookup tables are divided into contiguous buckets, each consisting of 26 elements that store pointers to the neighborhood voxel blocks. The storage order of these buckets is the same as that of the voxel blocks. Therefore, all neighbors of each voxel block can be found through its pointer. Neighborhood lookup tables are also updated in real time during scanning. First, we find the voxel blocks in the fundamental envelopes of the current input frame, and then find 26 neighbors for each of them by the hash table to update the corresponding elements of the lookup table. The pseudocode Algorithm 1 shows the whole procedure of computing TSDF based on the above data structure. It is assumed that the tangent of each point on the curve has been calculated before field computation. Figure 12 shows our entire data structure.

**Figure 12.** The whole data structure; first, the world coordinates of voxel center are mapped to the hash table to get the address of the voxel block on the central processing unit (CPU), and a thin-surface attribution judgment is then made to store the new TSDFs in either the non-inverse block or the inverse block; the subscripts of the neighborhood lookup table correspond to the individual blocks and each element of the lookup table stores the corresponding neighborhoods of the blocks; only the hash table is stored on the graphics processing unit (GPU), and all other data is stored on the CPU.

3. Results

Here, we verify the performance of our proposed algorithm through a series of experiments. The experimental data are all registered 3D curves collected by the laser scanner shown in Figure 1. Figure 13 shows an example of the input data of our algorithm. We use the proposed method to integrate all 3D curves of real objects into TSDF, extract the isosurface, and construct the mesh model.

To verify the effectiveness of the proposed thin-surface improvement, we compare the reconstruction results of the proposed method with Tubić et al. [6] for industrial products of different thicknesses. The accuracy of the proposed method is evaluated by comparing it with two existing hand-held 3D laser line scanners: Creaform HandySCAN700 [7] and Scantech HSCAN771 [8]. As shown in Table 1, these two kinds of scanner can achieve a volumetric accuracy of 0.02 mm (part size within 1 m) and have been widely used in the field of industrial 3D scanning. We use cloud to mesh (C2M) distance metric to quantify the accuracy of surface reconstruction [61]. The distance from each vertex in...
the source model to the closest triangle in the reference model is recorded, and two standard statistics are computed for all distances: Mean and Standard Deviation (Std.) [61]. In our experiments, the CAD model is used as a reference model. We do statistical hypothesis tests (Welch’s t-test [62] and F-test [63]) on these two statistics to evaluate the significance of the improvement. All the statistical tests are implemented by R programming language version 3.0.0 [64].

Figure 13. An example of input data of our algorithm; curves are displayed in different colors, and histograms of different color intervals are displayed on the right side.

Table 1. Accuracy of the scanners involved in the comparison.

| Scanner                    | Volumetric Accuracy $^1$ (Based on Part Size) |
|---------------------------|-----------------------------------------------|
| Creaform HandySCAN700 [7] | $0.020 \text{ mm} + 0.100 \text{ mm/m}$      |
| Scantech HSCAN771 [8]     | $0.020 \text{ mm} + 0.060 \text{ mm/m}$      |

$^1$ ISO 17025 accredited. Visit the manufacturer’s website for more details [7,8].

In terms of computational efficiency, we test the speed of the proposed algorithm on three hardware devices with different configurations shown in Table 2. The performance of the devices we used is equivalent to the hardware configuration recommended by mainstream industrial 3D scanner manufacturers (Creaform) [7]. Furthermore, we verify the effectiveness of the proposed method to improve the underlying data structure by comparing its GPU memory usage with VoxelHashing [20].

Table 2. Personal computer (PC) configuration for performance test.

| PC             | Configuration                                                                 |
|----------------|-------------------------------------------------------------------------------|
| Configuration A | Intel Core i7-7700HQ (2.8 GHz), 32 GB RAM, NVIDIA GeForce GTX1050 (4 GB)    |
| Configuration B | Intel Core i7-7700 (3.6 GHz), 32 GB RAM, NVIDIA GeForce GTX960 (2 GB)        |
| Configuration C | Intel Core i7-7820X (3.6 GHz), 128 GB RAM, NVIDIA GeForce GTX1060 (4 GB)     |

3.1. Experimental Object and Parameter Setting

Figure 14 shows the 3D model of six different objects scanned by the proposed method in our experiment. Figure 14a–e are five thin industrial products with different thicknesses, which are used to verify the effectiveness of the proposed thin-surface optimization. The thickness of these objects is indicated in each subfigure. All of the 3D models in the thin-surface verification experiment are generated with the same configuration parameters: voxel size $= 0.5 \text{ mm}$, truncation width $= 0.9 \text{ mm}$. For comparison purposes, we also set the same parameters for Tubić et al. [6]. When using the scanners of Creaform [7] and Scannech [8], we also set the resolution of scan to 0.5 mm.

The “relief” (f) is $2.4 \text{ m} \times 1.6 \text{ m}$ in size. The total number of curves acquired from the “relief” is 472,612, and the total number of 3D surface points is 3,365,470,052. Since this object has large size and low self-occlusion, we use it to test the algorithm time and memory efficiency. This object can ensure that around 7000 laser points are obtained at each frame, basically reaching the upper limit of
the single-frame data of the apparatus we used. In the performance test, we compare the speed and memory footprint of the algorithm in different resolutions. The truncation width is set to $\sqrt{\frac{3}{2}} \cdot \delta$ ($\delta$ as the voxel size), which is recommended by Tubić et al. [6].

![Image](image_url)

Figure 14. Six experimental objects: (a) metal bar; (b) automobile hub; (c) sheet metal part; (d) plastic shell; (e) metal panel; and (f) relief; the metal panel is composed of five parts with different thickness.

3.2. Qualitative Results

Figure 15 compares the surface reconstruction results of Tubić et al. [6] with ours for the same thin objects. The front and back views of each model are shown in these figures, and the front and back sides of the surface model are rendered in different colors (the front is given blue, the back is made yellow). The apparent improvements in the figures are marked with red ellipses. From these figures, we can see that many parts of results of Tubić et al. [6] have been lost, while our results are complete and smooth.

We also used Creaform HandySCAN700 [7] to scan the “sheet metal part” with a resolution of 0.5 mm, as shown in Figure 16. There are some small spikes and holes in Creaform’s [7] model, which are not found in the scanning process. The same defects were not found on the surface of our model. However, compared with Creaform’s [7] results, our surface is not smooth enough on the edges. Alias artifacts appear at sharp features on the extracted surfaces since TSDF is sampled on a regular grid with a fixed step length. In contrast, Creaform [7] uses higher resolution at sharp features.

The “plastic shell” is a more extreme example with a very complex topology. When scanning this object, it is almost impossible to ensure that the line of sight can be perpendicular to the sheets, so almost all of the sheets are corrupted (Figure 17a). In Figure 17b, Creaform’s [7] scanning results show similar problems, and the model surface at the edge of the sheet is messy and incomplete. By repairing the topological structure of these weak topological regions, the shape of these sheets is well restored (the missing parts are caused by occlusion, not by our method).
Figure 15. Comparison of scan models from thin products; (a) Tubić et al. [6] (left) and our (right) results of the metal bar; (b) Tubić et al. [6] (left) and our (right) results of the automobile hub; (c) Tubić et al. [6] (left) and our (right) results of the sheet metal part; (d) Tubić et al. [6] (top) and our (bottom) results of the metal panel.

Figure 16. Comparison of our scan results (top row) with the Creaform scanner [7] (bottom row); (a) a comparison of zoomed details at the same location; (b) the front and back views; (c) the details of another region; (d) triangles of the mesh models.
Figure 17. Comparison of the plastic shell model; (a) the interference of TSDF brings holes and spikes on the stiffener of the model; (b) Creaform’s [7] results show similar problems; (c) the proposed post-processing approach repaired the surface of the stiffener.

3.3. Quantitative Results

In the accuracy evaluation, we choose two typical thin objects: sheet metal part and plastic shell. The topological structure of the sheet metal part is simple, so it is used to verify the effectiveness of thin-surface improvement (thin-surface attribution judgment) in real-time reconstruction. The plastic shell has a complex topology, which makes it easy to misjudge the attribution of a thin surface in
the scanning process; so we use it to verify the effectiveness of the post-processing method for weak topological region repairing.

The accuracy of surface reconstruction is evaluated by the C2M distances [61] between reconstructions and CAD models. Figure 18 plots color maps of C2M distances for the two thin objects. In each subfigure, we visualize the C2M errors through a jet color map and a histogram. Distances greater than 0.2 mm are depicted with deeper colors, and the red/blue gradations indicate positive/negative distances, respectively. We evaluate the C2M errors quantitatively for each method with two statistics, as shown in Table 3. Our real-time processing approach is more accurate than those of Tubić et al. [6], but is similar to the existing scanners (Creaform, Levi, QC, Canada [7] and Scannech [8]). After the proposed post-processing, we get the minimum values on both Mean and Std.

**Figure 18.** Cloud to mesh (C2M) error colormaps for surface reconstructions of two thin products; the left column is the reconstruction of the sheet metal part, and the right column is the reconstruction of the plastic shell; The compared method are (a) Tubić et al. [6]; (b) Creaform [7]; (c) Scantech [8]; (d) our real-time processing approach; and (e) our post-processing approach. In each subfigure, frequencies with C2M errors in the interval of [−0.02 mm, 0.02 mm] are also indicated.
We randomly selected 1000 samples from the C2M distance data population of each method and carried out statistical tests on the samples to evaluate the significance of the proposed improvement. The mean and Std. of each group of sample data are listed in Table 4. Normal distribution is the basic hypothesis of a statistical test, so we use the Pearson’s \( \chi^2 \) test [65] to judge whether the samples are well-modeled by a normal distribution. As shown in Table 5, the \( p \)-value of all samples is higher than 0.05, so we conclude that the samples are consistent with the normal distribution at the significance level of \( \alpha = 0.05 \).

### Table 4. C2M error statistics (mean ± Std.) for the random samples of different methods.

| Object           | Tubić et al. [6] | Creaform [7] | Scantech [8] | Our Real-Time Approach | Our Post-Processing Approach |
|------------------|------------------|--------------|--------------|------------------------|-----------------------------|
| sheet metal part | 0.1599 ± 0.2592  | 0.0010 ± 0.0700 | 0.0024 ± 0.0408 | −0.0011 ± 0.0227       | −0.0005 ± 0.0212            |
| plastic shell    | 0.0093 ± 0.2412  | 0.0015 ± 0.0913 | −0.0053 ± 0.0554 | −0.0045 ± 0.0768       | −0.0008 ± 0.0579            |

\(^1\) The vertices number of the model.

### Table 5. Pearson’s \( \chi^2 \) test for samples of different methods.

| Object           | Tubić et al. [6] | Creaform [7] | Scantech [8] | Our Real-Time Approach | Our Post-Processing Approach |
|------------------|------------------|--------------|--------------|------------------------|-----------------------------|
| sheet metal part | 0.8562           | 0.3485       | 0.5587       | 0.7404                 | 0.6724                      |
| plastic shell    | (3.2966, 7) \(^1\) | (7.8224, 7) | (5.9662, 7) | (5.1856, 8)            | (5.7747, 8)                 |

\(^1\) Meaning of values in each cell: \( p \)-value (\( \chi^2 \) value, degrees of freedom \( \nu \)).

Here we apply Welch’s \( t \)-test (unequal variances \( t \)-test) [62] to compare the significance of the differences in the mean values between groups of samples, since the Std. between groups is different. Tables 6 and 7 show the \( t \)-test results of compared methods for two thin objects. \( \mu \) be the mean of the population corresponding to a column in the table, and \( \mu_i \) be the mean of the population corresponding to a row. We applied a one-tailed test [66] to each grid in the table at a significance level of 0.05. That is, \( H_0: \mu = \mu_i, H_1: \mu_i > \mu \). If \( p < 0.05 \), we reject the null hypothesis and conclude that \( \mu_i \) and \( \mu \) have a significant difference. For the two thin objects we selected, all the methods involved in the comparison have smaller mean values than Tubić et al. [6], but the differences between the other methods are not significant.

### Table 6. Welch’s \( t \)-test for the sheet metal part.

|                  | Tubić et al. [6] | Creaform [7] | Scantech [8] | Our Real-Time Approach |
|------------------|------------------|--------------|--------------|------------------------|
|                  | 0.0 *            |              | 0.7816       |                        |
| Creaform [7]     | (18.0343, 1131) \(^1\) |              |              |                        |
| Scantech [8]     | 0.0 *            |              | 0.5343 (−0.0862, 1194) | (1.2105, 1514) |
| Our real-time    | (18.6413, 1014) |              |              | 0.4487                 |
| Our post-processing | 0.0 *            |              | 0.5131 (−0.0300, 1175) | (1.3072, 1471) |
|                  | (18.6413, 1012) |              |              | (0.1288, 1992)         |

\(^1\) Meaning of values in each cell: \( p \)-value (\( t \) value, degrees of freedom \( \nu \)); * significant difference (\( p < 0.05 \)).

For the difference of Std. between each group of samples, we conduct the \( F \)-test of equality of variances [63], as shown in Tables 8 and 9. Let \( \sigma_1^2 \) be the variance of the population corresponding to a column in the table, and \( \sigma_2^2 \) be the variance of the population corresponding to a row. For each grid in the table, \( H_0: \sigma_1^2 = \sigma_2^2, H_1: \sigma_1^2 > \sigma_2^2. \) If \( p < 0.05 \), we reject the null hypothesis and conclude that \( \sigma_1^2 \) and \( \sigma_2^2 \) have a significant difference. For the sheet metal part, the variance of each group can be sorted as follows: Tubić et al. [6] > Creaform [7] > Scantech [8] > Our approach; there is no significant
difference between our real-time approach and post-processing approach at the significance level of 0.05. For the plastic shell, Tubić et al. [6] > Creaform [7] > Our real-time approach > Scantech [8] > Our post-processing approach.

### Table 7. Welch’s t-test for the plastic shell.

|                | Tubić et al. [6] | Creaform [7] | Scantech [8] | Our Real-Time Approach |
|----------------|------------------|--------------|--------------|------------------------|
| Creaform [7]   | 0.0 * (7.2831, 1305) | 0.0 * (7.5697, 1126) | 0.0 * (7.4946, 1224) | 0.0 * (8.0481, 1049) |
| Scantech [8]   | 0.0 * (6.1483, 1949) | 0.0 * (7.3644, 1309) | 0.0 * (6.1483, 1949) | 0.0 * (7.3644, 1309) |
| Our Real-time Approach | 0.0 * (1.3735) | 0.0 * (1.3735) | 0.0 * (1.3735) | 0.0 * (1.3735) |
| Our Post-processing Approach | 0.0 * (1.3735) | 0.0 * (1.3735) | 0.0 * (1.3735) | 0.0 * (1.3735) |

1 Meaning of values in each cell: p-value (t value, degrees of freedom ν); * significant difference (p < 0.05).

### Table 8. F-test for the sheet metal part.

|                | Tubić et al. [6] | Creaform [7] | Scantech [8] | Our Real-Time Approach |
|----------------|------------------|--------------|--------------|------------------------|
| Creaform [7]   | 0.0 * (13.0255) | 0.0 * (36.6872) | 0.0 * (132.0038) | 0.0 * (146.0340) |
| Scantech [8]   | 0.0 * (2.8165) | 0.0 * (10.1342) | 0.0 * (3.9805) | 0.0553 |
| Our Real-time Approach | 0.0 * (109.1113) | 0.0 * (3.9805) | 0.0 * (1.0162) | 0.0553 |

1 Meaning of values in each cell: p-value (F value); * the degree of freedom of each term is 999; * significant difference (p < 0.05).

### Table 9. F-test for the plastic shell.

|                | Tubić et al. [6] | Creaform [7] | Scantech [8] | Our Real-Time Approach |
|----------------|------------------|--------------|--------------|------------------------|
| Creaform [7]   | 0.0 * (6.3688) | 0.0 * (15.5504) | 0.0 * (8.7480) | 0.0 * (39.9106) |
| Scantech [8]   | 0.0 * (2.4424) | 0.0 * (1.3735) | 0.0 * (2.84e-7) | 0.0 * (39.9106) |
| Our Real-time Approach | 0.0 * (2.84e-7) | 0.0 * (1.3735) | 0.0 * (2.84e-7) | 0.0 * (39.9106) |
| Our Post-processing Approach | 0.0 * (2.84e-7) | 0.0 * (1.3735) | 0.0 * (2.84e-7) | 0.0 * (39.9106) |

1 Meaning of values in each cell: p-value (F value); * the degree of freedom of each term is 999; * significant difference (p < 0.05).

### 3.4. Performance

We have implemented various Single Instruction Multiple Data (SIMD) [67] instruction set versions of the proposed algorithm in C++ language, and the program was compiled with Visual Studio 2015. Figure 19a compares the frame rates of different instruction set versions for the online proposed reconstruction method on different hardware devices shown in Table 2. The voxel resolution of scanning is set to 0.5 mm, and about 40,000 voxels are inserted in every single frame. Although on different devices, the algorithm uses CPU multithreading and GPU parallelism to speed up, the use of SIMD instruction set still leads to a frame rate increase. Our algorithm can achieve a real-time frame rate above 60 FPS since most of the CPUs support instruction sets above SSE now. Figure 19b compares the time per frame for our method with Tubić et al. Although our method has additional operations such as thin-surface attribution judgment and normal filtering, the time-consumption increase is not obvious due to the superior performance of the underlying data structure we adopted. The comparison was made on a device with Configuration A.

Figure 20 compares the CPU memory usage and GPU memory usage of our proposed method and VoxelHashing [20] method. The comparison was made on a device with Configuration A. When the voxel resolution was increased to 2 mm, VoxelHashing [20] had already filled up all of the 4 GB GPU memory and made the program crashed, which significantly limited the algorithm’s practicability since most of the GPU hardware devices on the market only supported 4 GB memory. In contrast,
the GPU memory footprint of our proposed method was insignificant compared to voxel hashing [20]. Although the CPU memory footprint had increased, it was acceptable (up to 6 GB). After all, most computer devices support CPU memory expansion.

![Figure 19](image1.png)

Figure 19. Time efficiency comparison; (a) frame rate comparison of real-time reconstruction algorithms on three different hardware configurations in Table 2; the blue bars represent the algorithm implemented with Advanced Vector Extension Instruction Set (AVX) [68], red bars represent Streaming SIMD Extensions Set (SSE) [69], and the green bars represent the version with no instruction sets; (b) performance comparison with Tubić et al. [6].

![Figure 20](image2.png)

Figure 20. Memory footprint comparisons; (a) GPU memory footprint comparisons; (b) CPU memory footprint comparisons.

3.5. Discussion

From the qualitative results shown in Figure 15, we can see that our proposed modifications have visible progress in the reconstruction of thin surfaces compared with the original method of Tubić et al. [6]. The surface reconstructed by our method is complete and smooth, and it is difficult to see the interference between the opposite sides of the thin surface. However, Tubić et al. [6] cannot achieve the desired results. Both methods use the same input data and configuration parameters (voxel size and truncation width). It can be seen from Figure 17 that the proposed post-processing method is useful and necessary for thin objects with complex topology.

Quantitative analysis results can confirm the conclusion of qualitative evaluation. Whether it is a thin object with simple topological structure (sheet metal part) or complex topological structure...
(plastic shell), the deviation between the reconstruction of our method and CAD model is the smallest (the minimum mean and Std.). Taking the mean as the evaluation metric, our method significantly improves the accuracy of the original method of Tubić et al. [6], but it shows similar accuracy to the other two existing scanners (Creaform [7] and Scantech [8]). If Std. is used as the evaluation metric, our method is significantly different from other existing methods (Tubić et al. [6], Creaform [7], and Scantech [8]). Our method is more stable than the existing approaches. Moreover, for the thin object with complex topological structure, it can significantly improve the reconstruction accuracy by the proposed post-processing method.

As can be seen from Figure 19, the proposed algorithm can achieve a frame rate of more than 60 FPS on the hardware devices used in the experiment. These devices are not expensive, and their performance is the same as the hardware configuration recommended by mainstream industrial 3D scanner manufacturers. In terms of efficiency, the proposed algorithm can meet the requirements of industrial real-time 3D reconstruction. As can be seen from Figure 20, the proposed method can effectively reduce the GPU memory footprint of voxel hashing [20]. Although the memory footprint of the CPU has increased, it is within a reasonable range. After all, most computer devices support CPU memory expansion, while GPU cannot expand memory capacity. Therefore, our algorithm has high practicability.

Although the proposed algorithm improves the accuracy of thin-surface reconstruction and can meet the scanning requirements of most thin industrial products, the post-processing method of thin-surface topology repair still has some limitations on parameter setting. When the voxel size or truncation width is too large relative to the thickness of the object, the topology-repairing method will lose its effect. It is because our method of topological repair is based on the assumption that some voxels in the distance field are not interfered by the overlap of truncation bands on thin surfaces. The topology repair method uses these unaffected voxels as input to fit the surface and infer the values of the voxels in the affected area. If the voxel size or the truncation width are so large that the truncation bands on opposite sides of the thin surface overlap entirely, then the surface-fitting method loses reliable input, which leads to the inability to reconstruct the accurate surface. Therefore, for thin objects with complex topology, we recommend that users use such parameter configurations as voxel size ≤ 0.5 mm and truncation width ≤ 1 mm. There is no such limitation for thin objects with simple topology, such as a thin plate.

We use hash tables to compress the space and only reconstruct densely near the observed surface to eliminate the memory footprint of the empty space. However, it is not necessary to use regular grids of fixed voxel size to represent all parts of the scene. For example, a desktop needs less detail than a highly structured keyboard on it. Selecting different resolutions to represent different parts of the scene can reduce memory consumption and improve the reconstruction speed at the same time. It is also advantageous to use small voxel sizes to represent areas with complex topology or sharp features. Therefore, in the future, we will study how to build a 3D model with different resolutions adaptively according to the shape of the surface.

4. Conclusions

An improved 3D reconstruction method for a thin surface based on a hand-held laser line scanner is proposed. Aiming at the problems existing in the TSDF method for the 3D reconstruction of thin object surfaces, we propose two main improvements: first, in the real-time scanning process, we use the angle between the surface normal and the line of sight to determine the attribution of the thin surface, avoiding the mutual interference between the observation surfaces belonging to different positions; second, we exploit a post-processing method for automatically detecting and repairing topological errors in some areas where misjudgment of thin-surface attribution may occur. Besides, we also improve the underlying data structure of the existing TSDF method to reduce the GPU memory usage. Qualitative and quantitative experimental results provide evidence that the proposed 3D reconstruction approach to a thin surface is valid and can significantly improve the accuracy of
In comparison with the state-of-the-art laser line scanners, our method shows a similar (C2M distance mean [61]) or even better accuracy (C2M distance Std. [61]). In terms of computing efficiency, the proposed method effectively reduces the GPU memory consumption, which can guarantee a frame rate of more than 60 FPS with the GPU memory footprint under 500 MB.

Although the proposed method can reconstruct a thin surface accurately in most cases, there are still some limitations in parameter setting when reconstructing thin objects with complex topology. If the voxel size or truncation width is too large, mutual interference between the opposite sides of the thin surfaces may still occur. Meanwhile, it is a waste of memory to represent all parts of the scene with a regular grid. In the future, we plan to further improve the robustness and memory efficiency of complex thin-surface reconstruction by adaptively setting different resolutions to the reconstructed model according to the shape of the surface.

Author Contributions: Conceptualization, Y.H. and S.Z.; Methodology, Y.H. and S.Z.; Software, Y.H., S.Z. and F.Z.; Validation, Y.H. and F.Z.; Formal Analysis, Y.H., X.H. and F.Z.; Investigation, Y.H., S.Z., X.H. and F.Z.; Resources, S.Z.; Data Curation, Y.H.; Writing, Y.H., S.Z., X.H. and F.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Natural Sciences Foundation of China (41671452 and 41701532).

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

References
1. Kahler, O.; Prisacariu, V.; Valentin, J.; Murray, D. Hierarchical Voxel Block Hashing for Efficient Integration of Depth Images. *IEEE Robot. Autom. Lett.* **2016**, *1*, 192–197. [CrossRef]
2. Bourke, P. Automatic 3D reconstruction: An exploration of the state of the art. *GSTF J. Comput.* **2018**, *2*, 71–75.
3. Henry, P.; Krainin, M.; Herbst, E.; Ren, X.; Fox, D. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *Experimental Robotics*; Springer: Marrakech, Morocco, 2014; pp. 477–491.
4. Klingensmith, M.; Dryanovski, I.; Srinivasa, S.; Xiao, J. Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields. In *Proceedings of the Robotics: Science and Systems (RSS)*, Rome, Italy, 13–17 July 2015.
5. Lachat, E.; Macher, H.; Landes, T.; Grussenmeyer, P. Assessment and calibration of a RGB-D camera (Kinect v2 Sensor) towards a potential use for close-range 3D modeling. *Remote Sens.-Basel* **2015**, *7*, 13070–13097. [CrossRef]
6. Tubić, D.; Hébert, P.; Laurendeau, D. 3D surface modeling from curves. *Image Vis. Comput.* **2004**, *22*, 719–734. [CrossRef]
7. HandySCAN. Available online: https://www.creaform3d.com/en/portable-3d-scanner-handyscan-3d (accessed on 25 October 2019).
8. HSCAN. Available online: https://www.3d-scantech.com/product/hscan771-3d-scanner/ (accessed on 25 October 2019).
9. FreeScan. Available online: https://www.shining3d.com/solutions/freescan-x7 (accessed on 25 October 2019).
10. Zollhöfer, M.; Stotko, P.; Görß, A.; Theobalt, C.; Nießner, M.; Klein, R.; Kolb, A. State of the Art on 3D Reconstruction with RGB-D Cameras. *Comput. Graphics Forum* **2018**, *37*, 625–652. [CrossRef]
11. Hinzmann, T.; Schönberger, J.L.; Pollefeys, M.; Siegwart, R. Mapping on the Fly: Real-Time 3D Dense Reconstruction, Digital Surface map and Incremental Orthomosaic Generation for Unmanned Aerial Vehicles. In *Field and Service Robotics*; Springer: Zurich, Switzerland, 2018; pp. 383–396.
12. Cao, Y.; Kobelt, L.; Hu, S. Real-time High-accuracy Three-Dimensional Reconstruction with Consumer RGB-D Cameras. *ACM Trans. Graph.* **2018**, *37*, 171. [CrossRef]
13. Rusinkiewicz, S.; Levoy, M. Efficient Variants of the ICP Algorithm. In Proceedings of the Third International Conference on 3-D Digital Imaging and Modeling, Quebec, QC, Canada, 28 May–1 June 2001; pp. 145–152.
14. Stücker, J.; Behnke, S. Integrating Depth and Color Cues for Dense Multi-Resolution Scene Mapping Using RGB-d Cameras. In Proceedings of the 2012 IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), Hamburg, Germany, 13–15 September 2012; pp. 162–167.
15. Weise, T.; Wismer, T.; Leibe, B.; Van Gool, L. In-Hand Scanning with Online Loop Closure. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), Kyoto, Japan, 29 September–2 October 2009; pp. 1630–1637.

16. Gallup, D.; Pollefeys, M.; Frahm, J. 3D Reconstruction Using an n-Layer Heightmap. In Joint Pattern Recognition Symposium; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–10.

17. Pollefeys, M.; Nistér, D.; Frahm, J.; Akbarzadeh, A.; Mordohai, P.; Clipp, B.; Engels, C.; Gallup, D.; Kim, S.; Merrell, P. Detailed real-time urban 3D reconstruction from video. Int. J. Comput. Vis. 2008, 78, 143–167. [CrossRef]

18. 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 24th Annual ACM Symposium on User Interface Software and Technology, Minneapolis, MN, USA, 20–23 October 2011; pp. 559–568.

19. Kähler, O.; Prisacariu, V.A.; Murray, D.W. Real-Time Large-Scale Dense 3D Reconstruction with Loop Closure. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 October 2016; pp. 500–516.

20. Nießner, M.; Zollhöfer, M.; Izadi, S.; Stamminger, M. Real-time 3D reconstruction at scale using voxel hashing. ACM Trans. Graph. 2013, 32, 169. [CrossRef]

21. Curless, B.; Levoy, M. A Volumetric Method for Building Complex Models from Range Images. In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New Orleans, LA, USA, 4–9 August 1996; pp. 303–312.

22. Fedkiw, S.O.R.; Osher, S. Level set methods and dynamic implicit surfaces. Surfaces 2002, 44, 77.

23. Lorensen, W.E.; Cline, H.E. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In Proceedings of the 14th annual Conference on Computer graphics and Interactive Techniques, Anaheim, CA, USA, 27–31 July 1987; pp. 163–169.

24. Kähler, O.; Prisacariu, V.A.; Ren, C.Y.; Sun, X.; Torr, P.; Murray, D. Very high frame rate volumetric integration of depth images on mobile devices. IEEE Trans. Vis. Comput. Graph. 2015, 21, 1241–1250. [CrossRef]

25. 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 2013; pp. 343–352.

26. Bylow, E.; Sturm, J.; Kerl, C.; Kahl, F.; Cremers, D. Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions. In Proceedings of the Robotics: Science and Systems (RSS), Berlin, Germany, 24–28 June 2013.

27. Dong, W.; Wang, Q.; Wang, X.; Zha, H. PSDF Fusion: Probabilistic Signed Distance Function for On-the-fly 3D Data Fusion and Scene Reconstruction. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 701–717.

28. Choe, G.; Park, J.; Tai, Y.; Kweon, I.S. Refining geometry from depth sensors using IR shading images. Int. J. Comput. Vis. 2017, 122, 1–16. [CrossRef]

29. Maier, R.; Kim, K.; Cremers, D.; Kautz, J.; Nießner, M. Intrinsics3d: High-Quality 3D Reconstruction by Joint Appearance and Geometry Optimization with Spatially-Varying Lighting. In Proceedings of the IEEE International Conference on Computer Vision, Honolulu, HI, USA, 21–26 July 2017; pp. 3114–3122.

30. Zollhöfer, M.; Dai, A.; Innmann, M.; Wu, C.; Stamminger, M.; Theobalt, C.; Nießner, M. Shading-based refinement on volumetric signed distance functions. ACM Trans. Graph. 2015, 34, 96. [CrossRef]

31. Liu, Y.; Gao, W.; Hu, Z. Geometrically stable tracking for depth images based 3D reconstruction on mobile devices. ISPRS J. Photogramm. 2018, 143, 222–232. [CrossRef]

32. Slavcheva, M.; Kehl, W.; Navab, N.; Ilie, S. SDF-2-SDF Registration for Real-Time 3D Reconstruction from RGB-D Data. Int. J. Comput. Vis. 2018, 126, 615–636. [CrossRef]

33. Choi, S.; Zhou, Q.; Koltun, V. Robust Reconstruction of Indoor Scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 5556–5565.

34. Dai, A.; Nießner, M.; Zollhöfer, M.; Izadi, S.; Theobalt, C. Bundlefusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegartion. ACM Trans. Graph. 2017, 36, 76a. [CrossRef]

35. Han, L.; Fang, L. FlashFusion: Real-time Globally Consistent Dense 3D Reconstruction Using CPU Computing. In Proceedings of the Robotics: Science and Systems (RSS), Pittsburgh, PA, USA, 26–30 June 2018.
36. Maier, R.; Schaller, R.; Cremers, D. Efficient online surface correction for real-time large-scale 3D reconstruction. *arXiv* 2017, arXiv:1709.03763.

37. Guo, K.; Taylor, J.; Fanello, S.; Tagliasacchi, A.; Dou, M.; Davidson, P.; Kowdle, A.; Izadi, S. TwinFusion: High FrameRate Non-Rigid Fusion Through Fast Correspondence Tracking. In Proceedings of the 2018 International Conference on 3D Vision (3DV), Verona, Italy, 5–8 September 2018; pp. 596–605.

38. Palazzolo, E.; Behley, J.; Lottes, P.; Giguere, P.; Stachniss, C. ReFusion: 3D Reconstruction in Dynamic Environments for RGB-D Cameras Exploiting Residuals. *arXiv* 2019, arXiv:1905.02082.

39. Slavcheva, M.; Baust, M.; Cremers, D.; Ilic, S. Killingfusion: Non-rigid 3D Reconstruction without Correspondences. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1386–1395.

40. Slavcheva, M.; Baust, M.; Ilic, S. SobolevFusion: 3D Reconstruction of Scenes Undergoing Free Non-Rigid Motion. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Verona, Italy, 5–8 September 2018; pp. 2646–2655.

41. Zheng, Z.; Yu, T.; Li, H.; Guo, K.; Dai, Q.; Fang, L.; Liu, Y. Hybrid Fusion: Real-Time Performance Capture Using a Single Depth Sensor and Sparse IMUs. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 384–400.

42. Riegler, G.; Ulusoy, A.O.; Bischof, H.; Geiger, A. Octnetfusion: Learning Depth Fusion from Data. In Proceedings of the 2017 International Conference on 3D Vision (3DV), Qingdao, China, 10–12 October 2017; pp. 57–66.

43. Dai, A.; Ritchie, D.; Bokeloh, M.; Reed, S.; Sturm, J.; Nießner, M. Scancomplete: Large-scale scene completion and semantic segmentation for 3D scans. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Verona, Italy, 5–8 September 2018; pp. 4578–4587.

44. Chang, A.; Dai, A.; Funkhouser, T.; Halber, M.; Niessner, M.; Savva, M.; Song, S.; Zeng, A.; Zhang, Y. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv* 2017, arXiv:1709.06158.

45. Hou, J.; Dai, A.; Nießner, M. 3D-sis: 3D Semantic Instance Segmentation of rgb-d Scans. In Proceedings of the IEEE International Conference on 3D Vision (3DV), Qingdao, China, 10–12 October 2017; pp. 5828–5839.

46. McCormac, J.; Clark, R.; Bloesch, M.; Davison, A.; Leutenegger, S. Fusion++: Volumetric Object-Level Slam. In Proceedings of the 2018 International Conference on 3D Vision (3DV), Verona, Italy, 5–8 September 2018; pp. 32–41.

47. Pham, Q.; Hua, B.; Nguyen, T.; Yeung, S. Real-Time Progressive 3D Semantic Segmentation for Indoor Scenes. In Proceedings of the 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), Waikoloa Village, HI, USA, 7–11 January 2019; pp. 1089–1098.

48. Vineet, V.; Miksik, O.; Lidegård, M.; Niessner, M.; Golodetz, S.; Prisacariu, V.A.; Kähler, O.; Murray, D.W.; Izadi, S.; Pérez, P. Incremental Dense Semantic Stereo Fusion for Large-Scale Semantic Scene Reconstruction. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 75–82.

49. Whelan, T.; Kaess, M.; Johannsson, H.; Fallon, M.; Leonard, J.J.; McDonald, J. Real-time large-scale dense RGB-D SLAM with volumetric fusion. *Int. J. Robot. Res.* 2015, 34, 598–626. [CrossRef]

50. Chen, J.; Bautembach, D.; Izadi, S. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.* 2013, 32, 113. [CrossRef]

51. Steinbrücker, F.; Sturm, J.; Cremers, D. Volumetric 3D Mapping in Real-Time on a CPU. In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–7 June 2014; pp. 2021–2028.

52. Zeng, M.; Zhao, F.; Zheng, J.; Liu, X. Octree-Based Fusion for Realtime 3D Reconstruction. *Graph. Models* 2013, 75, 126–136. [CrossRef]

53. Dryanovski, I.; Klingsensmith, M.; Srinivasa, S.S.; Xiao, J. Large-scale, real-time 3D scene reconstruction on a mobile device. *Auton. Robot.* 2017, 41, 1423–1445. [CrossRef]
55. Li, S.; Cheng, M.; Liu, Y.; Lu, S.; Wang, Y.; Prisacariu, V.A. Structured Skip List: A Compact Data Structure for 3D Reconstruction. In Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 1–5 October 2018; pp. 1–7.

56. França, J.G.D.; Gazziro, M.A.; Ide, A.N.; Saito, J.H. A 3D Scanning System Based on Laser Triangulation and Variable Field of View. In Proceedings of the IEEE International Conference on Image Processing 2005, Genova, Italy, 11–14 September 2005; p. 425.

57. Pauly, M.; Gross, M.; Kobbelt, L. Efficient simplification of point-sampled surfaces. In Proceedings of the the conference on Visualization’02, Washington, DC, USA, 2 October 2002; pp. 163–170.

58. Lu, X.; Wu, S.; Chen, H.; Yeung, S.; Chen, W.; Zwicker, M. GPF: GMM-inspired feature-preserving point set filtering. *IEEE Trans. Vis. Comput. Graph.* 2018, 24, 2315–2326. [PubMed]

59. Sharf, A.; Lewiner, T.; Shklarski, G.; Toledo, S.; Cohen-Or, D. Interactive Topology-Aware Surface Reconstruction. *ACM Trans. Graphics (TOG)* 2007, 26, 43. [CrossRef]

60. Vaillant, R. Recipe for Implicit Surface Reconstruction with HRBF-Rodolphe Vaillant’s Homepage. Available online: http://rodolphe-vaillant.fr/?e=12 (accessed on 30 August 2019).

61. Handa, A.; Whelan, T.; McDonald, J.; Davison, A.J. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–5 June 2014; pp. 1524–1531.

62. Welch’s t-Test. Available online: https://en.wikipedia.org/wiki/Welch%27s_t-test (accessed on 8 January 2020).

63. F-Test of Equality of Variances. Available online: https://en.wikipedia.org/wiki/F-test_of_equality_of_variances (accessed on 8 January 2020).

64. R Programming Language. Available online: https://www.r-project.org/ (accessed on 8 January 2020).

65. Pearson’s Chi-Squared Test. Available online: https://en.wikipedia.org/wiki/Pearson%27s_chi-squared_test (accessed on 8 January 2020).

66. One-and Two-Tailed Tests. Available online: https://en.wikipedia.org/wiki/One-_and_two-tailed_tests (accessed on 8 January 2020).

67. SIMD. Available online: https://en.wikipedia.org/wiki/SIMD (accessed on 8 January 2020).

68. Advanced Vector Extensions. Available online: https://en.wikipedia.org/wiki/Advanced_Vector_Extensions (accessed on 8 January 2020).

69. Streaming SIMD Extensions. Available online: https://en.wikipedia.org/wiki/Streaming_SIMD_Extensions (accessed on 8 January 2020).