Structured Skip List: A Compact Data Structure for 3D Reconstruction

Shi-Jie Li[1], Ming-Ming Cheng[1], Yun Liu[1], Shao-Ping Lu[1], YaHui Wang[1], Victor Adrian Prisacariu[2]

Abstract— **The model produced by 3D reconstruction algorithm is usually represented by voxels. The management of these voxels is usually divided into two categories: ordered and unordered methods. The ordered method holds too many empty voxels to maintain data order which leads to a low storage efficiency. On the contrary, the unordered method keeps massive index data to only store nonempty voxels. In this paper, we design a new data management method for real-time indoor 3D reconstruction, called Structured Skip List (SSL). The SSL can be treated as a semi-ordered method, because the advantages of both the ordered and unordered methods are taken into account: 1) it only holds nonempty voxels similar to the unordered method; 2) the structured information is introduced to reduce the storage space of index data. By these designs, the SSL has a better performance on storage efficiency. To handle the data collision in voxel allocation, a hash allocation list (HAL) is proposed. The length of each Skip List is kept balanced by fusing the IMU (Inertial Measurement Unit) information for a high operation efficiency . The storage efficiency analysis of different data management methods is shown in this paper. What's more, exhaustive investigation is carried out on several datasets with these methods. The experimental result demonstrates that our design can achieve a high storage efficiency with little time loss compared to the state-of-the-art methods.**

I. INTRODUCTION

3D reconstruction [1], [2] is an attractive research topic, and it has been widely used in various applications, such as robot navigation, environmental perception and 3D modeling. To further boost these applications, efficient 3D reconstruction is of vital importance. Fortunately, 3D reconstruction at real-time has gone through a rapid progress in last several years, due to significant advances on both 3D sensors and computational capacity. With the development of the low-cost RGB-D cameras (such as Kinect), the acquirement of 3D data is much easier than before. The computational load is the main reason to limit the running efficiency of algorithms in the past. With the progress of computing resource, the speed of 3D reconstruction gets great improvement.

Existing 3D reconstruction algorithms can usually divide the whole 3D space into uniform cubes, called voxels, and the 3D data is stored in each voxel. The management of voxels is usually based on one of the following two styles: ordered [3] or unordered [4] methods. The ordered method organizes the voxels in order. Hence there is no need to store extra index data. However, this method needs to hold plenty of empty voxels to keep this order due to the sparse distribution of nonempty voxels. Therefore, the storage efficiency of the ordered method is poor. To address this issue, the unordered data method is proposed. This method only stores the voxels near the reconstructed surface. The management of voxels is implemented through a well-designed hash table. By reducing the storage of empty voxels, the unordered method gets a better performance on storage efficiency than the ordered method. However, the index data usually takes up a lot of space in whole storage which is not desired.

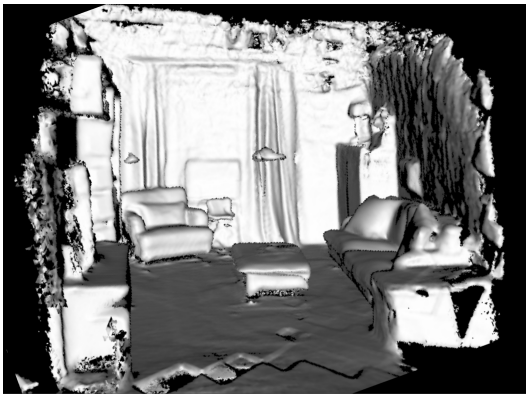


Fig. 1: A 3D model produced by our algorithm.

The data distribution in the real world is usually along a principal direction, which means the data along the principal direction distributes more uniformly than other directions. The horizontal direction is usually the principal direction in most situations. Combining the advantages of above data management methods and taking the characteristic of data distribution into consideration, we propose a new data structure, called Structured Skip List (SSL). Similar to the unordered method, only those voxels near the reconstructed surface are stored for the consideration of storage efficiency. The voxels along the vertical direction of principal direction are linked one by one and form as an unordered Skip List. Along the principal direction, an ordered index is used to manage all Skip Lists. Because the structured information is utilized, the needed index data is reduced similar to ordered method. What's more, there exists less empty terms in index data because index data distributes along principal direction. By this design, SSL can get good performance on storage efficiency. There are two factors existing in our algorithm which are harmful to operation efficiency, the data collision in voxel allocation and the unbalanced lengths of Skip Lists. For the data collision in voxel allocation, a Hash Allocation List (HAL) is used to solve intensive allocation collision when new voxels are allocated. The IMU information is introduced into our system to keep the balance of Skip List length. Hence our system can also run in a high frame rate.

In the following sections, we first review some recent

[1] College of Computer Science, Nankai University, Tianjin, China
[2] Department of Engineering, Oxford University, UK

progress on 3D reconstruction in Sec. II. Then, an overview of the system pipeline is described in Sec. III. In Sec. IV, the design of SSL will be discussed in details. An exhaustive analysis of storage efficiency is shown in Sec. V. The evaluation results of our method with other state-of-the-art methods will be given in Sec. VI.

II. RELATED WORK

In this section, we briefly review the development of real-time 3D reconstruction in recent years. Real-time 3D reconstruction algorithms can be divided into two streams according to the data structures they used. The mainstream data structure that reconstruction algorithms usually use is the *voxel* which is a small cube in 3D space [1]–[10]. Another data structure coming into use recently is *surfel* which represents a small area data on the reconstructed surface [11], [12].

The data in voxel-based reconstruction algorithms is usually represented as Truncated Signed Distance Function (TSDF) [13], [14] which can easily fuse the measurement from different frames. The Kinect Fusion algorithm [1], [3] is the first successful real-time 3D reconstruction system. In this algorithm, the whole space is divided into small voxels and stored on the GPU memory. The storage efficiency of Kinect Fusion algorithm is low because of the large number of empty voxels. Hence the Kinect Fusion only aims at small-scale 3D reconstruction task. To handle this problem, Kintinuous [7]–[10] use the cloud slice technique to keep the used GPU memory fixed. By this method, the Kintinuous algorithm can reconstruct larger space. A better design, voxel hashing [4], [5], is proposed to utilize the sparsity of data distribution in 3D space. With this method, the storage efficiency becomes higher.

Although the pipeline of 3D reconstruction is roughly similar, recent research mainly focuses on more specific aspects. DynamicFusion [15] aims to handle the reconstruction of non-rigid scenes. GravityFusion [16], [17] fuses the information from different sensors to boost the mapping accuracy. To improve the reconstruction accuracy, InfiniTAM V3 [6] uses *submap* strategy to adjust the relative locations of adjacent *submaps*. Whelan *et al.* [9] used a deformation-based map to adjust the reconstructed scenes. Other algorithms [18]–[21] utilize the geometry information to improve the performance. As for plane information, surfel-based reconstruction system is more popular due to the consistency of data structure and plane, such as ElasticFusion [11].

III. ALGORITHM PIPELINE

The pipeline of our algorithm is shown in Fig. 2, which is similar to [5]. The main modification is the modules related to the new data structure. The whole pipeline can be divided into four parts: tracking, mapping, rendering and coordinate adjustment.

A. Tracking

Similar to [8], the tracking is processed both on depth images and color images. For the input depth image, a

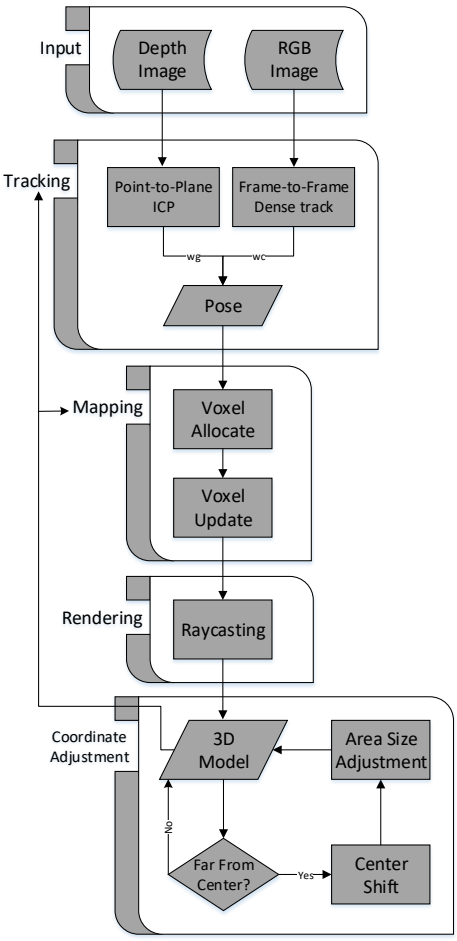


Fig. 2: The pipeline of the whole system.

point-to-plane ICP (Iterative Closest Point) algorithm [22], [23] is applied. In other words, the point-to-plane distance (geometry error) between points from the depth image and the reconstructed 3D model is minimized:

$$\min_{R,t} \sum_{p} ((Rp + t - \mathcal{V}(\bar{p}))^T \cdot \mathcal{N}(\bar{p}))^2, \tag{1}$$

where p is the 3D points observed in the depth image I_D. R and t are the rotation and translation of the camera. $\mathcal{V}$ and $\mathcal{N}$ are the maps of surface points and normals, which can be produced by the rendering stage. $\bar{p}$ is the projection of p in the reconstructed 3D model.

To improve the tracking robustness, a color based direct image alignment is carried out between the input color image and the rendered color image. In this situation, the photometric error for all pixels are minimized by

$$\min_{R,t} \sum_{p} ||I_C(\pi(Rp + t) - C(p)||_2, \tag{2}$$

where p and $C(p)$ represent a 3D point and its color, as extracted in the raycasting stage. I_C is the current color image. These two error functions are minimized using Gauss-Newton approach, and the two optimized results are linearly

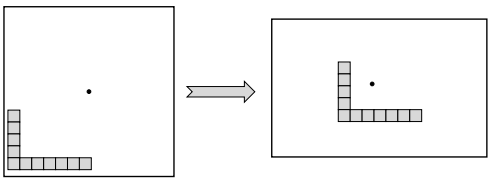


Fig. 3: The procedure of coordinate adjustment. The grey squares are occupied voxel. The black circle is the world center.

combined to get the current camera pose:

$$pose = w_g \cdot pose_g + w_c \cdot pose_c, \tag{3}$$

where w_g and w_c are corresponding weights.

B. Mapping

The world is represented by a truncated signed distance function (TSDF) D [13], [14], which maps each 3D point to a distance from the nearest surface. To reduce the index number and data collision, voxels are grouped as voxel blocks, each of which includes $8 \times 8 \times 8$ voxels. Considering the storage efficiency, only the voxel block near the reconstructed surface is allocated. When a new frame is coming, the new voxel blocks are allocated first. Then the data of all visible voxel blocks are updated.

Voxel Allocation: When a new frame is coming, the unallocated voxel blocks will be allocated. The detail of this procedure will be presented in the next section.

Voxel Update: The voxel data update is the same as the original KinectFusion algorithm [1], [3]. For each pixel in the depth image, the voxel update is performed if the depth value is in the valid range. The corresponding voxel is found using the camera pose that is obtained from the tracking stage.

C. Rendering

The rendering pipeline is similar to [5]. The output of this procedure is a point cloud map and a normal map which will be used in the following procedure. Because the new data structure is adopted, the processes related to data access are modified to fit the new data structure. Similar to voxel update, only the visible voxels are processed.

D. Coordinate Adjustment

To avoid crossing the border and improve the reconstructed scale that same index data can hold, we design an operation called coordinate adjustment. Benefiting from complete separation of index data and voxel data, the adjustment is performed only on index data. The coordinate adjustment includes two parts: center shift and area adjustment.

Center Shift: The center shift keeps the reconstructed 3D model at the center of coordinate. The new center is the geometric center of the reconstructed 3D model. When the new center is selected, only the index data is adjusted and the voxel data remains unchanged.

Area Adjustment: The area adjustment will change the length-width ratio of x-y coordinates to make the coordinates fit the reconstructed 3D model better. Similar to center shift, only the index is adjusted.

By these dynamic adjustments, the same storage can accommodate more data than fixed coordinate and avoid crossing the border of coordinate. Fig. 3 shows this procedure.

IV. Data Structure Design

In this section, we will give an exhaustive description of our new data structure: Structure Skip List (SSL). A diagram of SSL is shown in Fig. 4. As we all know, the data usually distributes along the principal direction. For the clear description, we define the world coordinate system in the following section as follow: the ground is represented by the x-y plane and the z axis is upward. To fit the character of data distribution, we design the SSL as a semi-order data structure: the index data along x-y coordinates is ordered whereas the voxel data along z axis is unordered.

A. Data Organization

The SSL consists of two lists: an ordered index list and an unordered data list. The voxel blocks with the same x-y values are called Skip List and stored in unordered data list. Each voxel block in Skip List is linked by a offset. The ordered index list manages all Skip Lists. The entries in index list store the pointers that point to the start node of the corresponding Skip List. The relevant data structures are list in the following:

```
// ordered index list
int ptr[INDEX_NUM];

// unordered data list
struct Voxel
{
  float SDF_value;
  uchar fusion_num;
};

struct VoxelBlock
{
  Voxel voxels[512];
  Vector3s pos;
  short offset;
};

VoxelBlock *VBA;

// HAL
Vector3s allocatePos[ALLOCATE_NUM];
```

B. Retrieval

The search for specified voxel is carried out as follows:

1) Find the corresponding entry in ordered index list using x-y values.

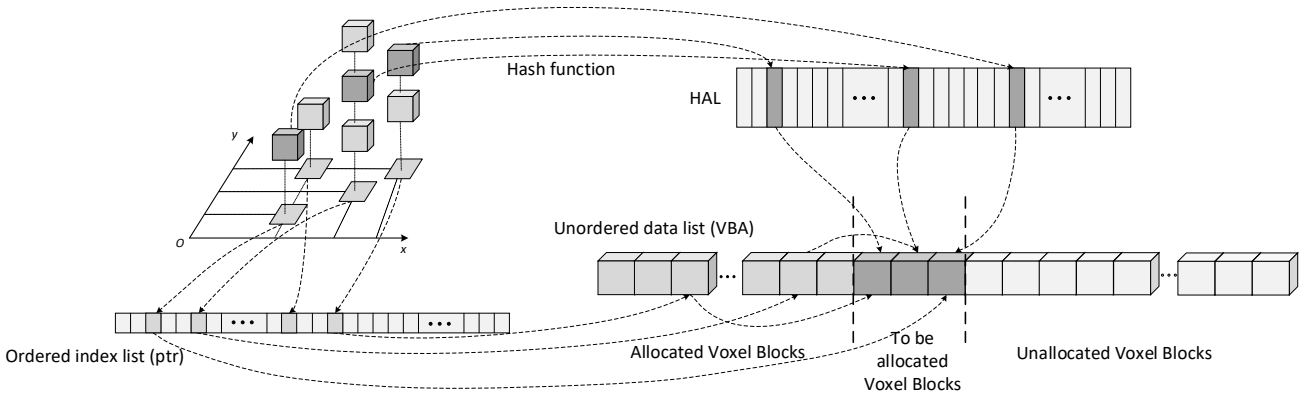


Fig. 4: The overviews of SSL and HAL. The new allocated voxel blocks are represented by dark grey cubes. The positions of allocated voxel blocks are inserted into HAL first. The inserted positions in VBA are found using ordered index list.

2) Get the start position of corresponding Skip List using above results.
3) Traverse the Skip List to get the position of corresponding voxel block.
4) Search in the voxel block to obtain the target voxel.

If any of the above procedures fails, the voxel is not allocated up to now.

C. Allocation

When a new frame is coming, the unallocated voxel blocks will be allocated first. Because the check for allocation position is parallel, the allocation procedure is with heavy data collision. We use Hash Allocation List (HAL) to handle this problem. When the voxel block containing checked point does not exist in VBA, the position of voxel block will be inserted into HAL. All checked voxel blocks with same position will be mapped to the same entry in HAL using a hash function. As we know, in a Simultaneous Localization And Mapping (SLAM) system, the contents of the continuous frames are similar. Hence the voxel in the current frame can be usually observed in the following frames. The voxel block position mapping to an occupied entry HAL will not be taken into consideration in the allocation of current frame and will be allocated in the following frames. In addition, in each allocation procedure, only one voxel block is allocated for each Skip List to simplify the allocation procedure. The rest of other voxel blocks will also be allocated in the following frames. The burden of voxel allocation for different frames is usually different. By this design, the heavy burden is distributes among the continuous frame which is beneficial to system efficiency.

The inserted position (x, y, z) is mapped to HAL by a hash function:

$$h = ((x * P_1) \bigoplus (y * P_2)) \ mod \ ALLOCATE\_NUM \quad (4)$$

where $P1$ and $P2$ are 653 and 541, respectively. With our design, the HAL is efficient with a limited length. The voxel allocation procedure is shown in Fig. 4.

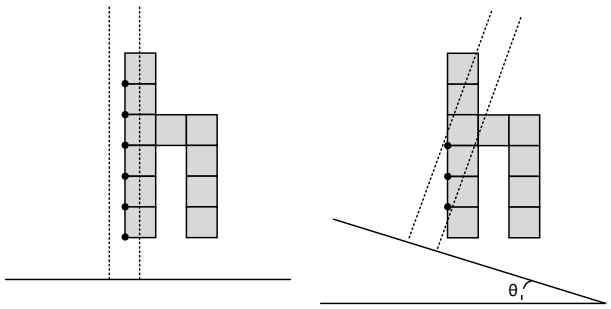


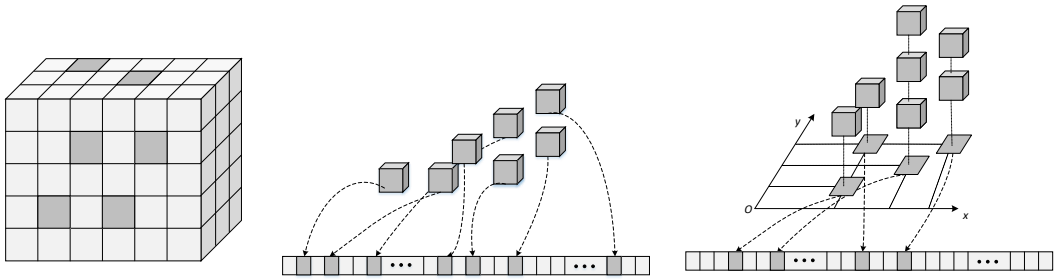
Fig. 5: The length of Skip List is varying with principal direction changing. The black circles represent the positions of voxel blocks. We can see that the length of vertical Skip List reduces from six to three due to the rotation of principal direction.

D. IMU Information Fusion

Because the time consumption for parallel algorithm is determined by the maximum time of one thread in all parallel threads. It is harmful for operation efficiency when the lengths of Skip Lists are unbalanced. As we can see, the data usually distributes along principal direction. To keep balance among the lengths of Skip Lists, the IMU information is utilized to determine the principal direction. By the aid of IMU, the world coordinate system is set as follows: the ground is on x-y plane and the z axis is opposite to gravity direction. Although this setting is fine for most situations, in some situations this problem is still not been solved. For example, some vertical structures should be taken into account such as the walls. In Fig. 5, when the z axis rotates a little angle we can see that the length of SKip List is reduced. Thus the IMU information can be used to reduce the influence of unbalance length.

V. Storage Efficiency Analysis

In this section, we give a detailed analysis of three data management methods: ordered method, unordered method,



(a) (b) (c)

Fig. 6: Diagrammatic sketch of three methods: (a) ordered, (b) unordered, (c) semi-ordered. The dark cubes represent the voxels with meaningful data, whereas the light grey ones represent empty voxels.

and semi-ordered method. In these methods, the data is divided into two parts, index data and voxel data. We define the notations in the following analysis as follows:

- N_I: The item number in index data.
- S_I: The storage of index item.
- N_V: Total voxel number.
- N_{NV}: The number of nonempty voxel.
- S_V: The storage of a voxel.

In the algorithm design, the storage efficiency E can be formulated as

$$E = \frac{N_{NV} S_V}{N_I S_I + N_V S_V}, \tag{5}$$

in which E is expected to be high enough. In the following, we assume the voxel number N_V is fixed in each method.

A. Ordered Method

There is no index data existing in the ordered method, which means $N_I S_I = 0$. At the expense of this, all voxels in space must be hold to keep the data in order no matter whether there is data in them or not. Only by this way, the voxels can be accessed. As mentioned above, the data distribution in real world is extremely sparse which leads to low utilization rate of voxels. In other words, the N_{NV} is far less than N_V, *i.e.* $N_{NV} << N_V$. Hence the storage efficiency is in a low level. Thus the data in ordered method can be treated as a dense 3D matrix with many zero in it (as shown in Fig. 6 (a)). The final storage efficiency is

$$E_{ordered} = \frac{N_{NV}}{N_V} \approx 0 \tag{6}$$

B. Unordered Method

Different from ordered method, unordered method only stores the voxels near the reconstructed surface. Almost all stored voxels are nonempty, which means $N_{NV} \approx N_V$. To manage unordered voxel data, index data (hash table) is introduced. In most situations, the number of index terms is greater than voxel terms due to some management operation such as swap operation [4]. Hence the N_I is greater than N_V, *i.e.* $N_I >> N_V$. What's worse, some voxel data (voxel position) is stored in index data, and it will occupy a lot of memory due to different lengths of index data and voxel data. Due to the sparsity of the data distribution, unordered method has a better performance on storage efficiency than ordered method. However, it is still possible to compress the cost space further. From above discussion, we can see that the unordered method is similar to a sparse 3D matrix. The voxel data is the element in this matrix and the hash table is used for data access (as shown in Fig. 6 (b)).

C. Semi-ordered Method

Similar to unordered method, only the voxels near the reconstructed surface are hold for the consideration of storage efficiency ($N_{NV} \approx N_V$). The data in each index term is minimized (a pointer) and all voxel-related data is stored in voxel ($S_I < S_V$). Because the data distribution along the principal direction is usually uniform, almost all indexes hold the valid data. We have $N_I < N_V$ because one index term may point to several voxels. In a word, the index data is organized in order, and is similar to a 2D dense matrix. The voxel data is separated in each index term in an unordered method which is similar to a link list. Because of the combination of the ordered method and unordered method, this method is treated as semi-ordered method. The mechanism of semi-ordered method is shown in Fig. 6 (c).

TABLE I: The experimental results on TUM RGB-D dataset (storage efficiency (%) and average runtime (ms) per frame).

sequence	SSL	hash	hhash	ordered
fr1_desk	**99.988**	96.124	93.19	0.043
fr3_cabinet	**99.982**	99.516	99.47	0.036
fr3_str_notex_far	**99.986**	99.727	99.694	0.043
fr3_str_notex_near	**99.982**	99.031	99.553	0.011
fr3_str_tex_far	**99.988**	99.375	99.704	0.021
fr3_str_tex_near	**99.983**	92.963	98.99	0.173
average time (ms)	4.99	**3.33**	4.77	14.81

VI. EXPERIMENTAL EVALUATION

In this section, we evaluate various competitors on two datasets: TUM RGB-D dataset [24] and ScanNet dataset [25]. All algorithms in this section are evaluated on the same platform with Intel i7-4790K (4GHz) CPU and NVIDIA GTX 1080 GPU. We choose a state-of-the-art 3D reconstruction system, InfiniTAM [5], [6], [26], [27], as the baseline

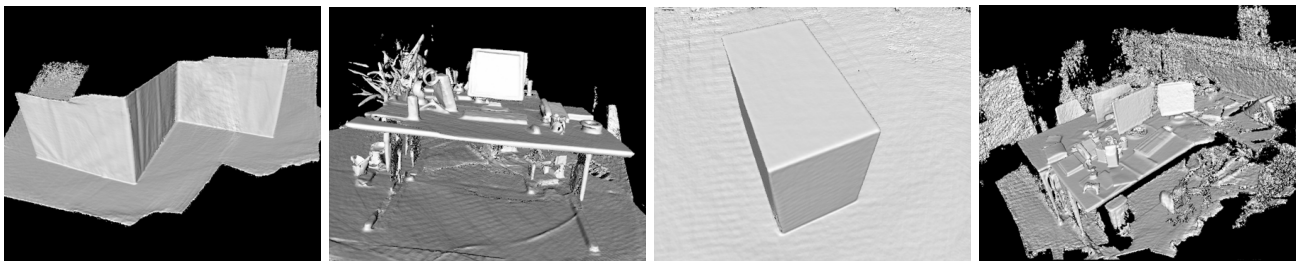


Fig. 7: Reconstruction results on some TUM RGB-D Dataset sequences.

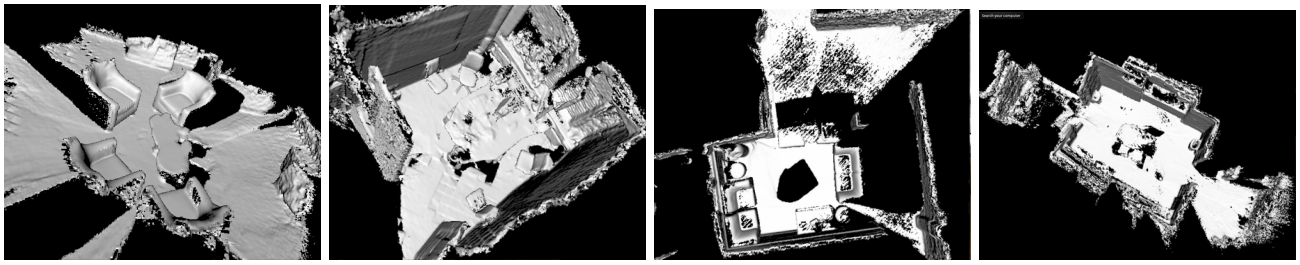


Fig. 8: Reconstruction results on some Scannet Dataset sequences.

TABLE II: The experimental results on ScanNet dataset (storage efficiency (%) and average runtime (ms) per frame).

sequence	SSL	hash	hhash	ordered
0013_00	**99.981**	99.052	98.045	0.003
0032_00	**99.986**	98.848	93.857	-
0036_00	**99.981**	98.55	96.218	-
0047_00	**99.988**	97.514	93.484	0.005
0049_00	**99.985**	98.299	98.291	-
average time (ms)	3.92	3.4	**2.83**	11.75

method. Apart from the highly efficient implementation, it also includes several mainstream data structure used for 3D reconstruction, such as *voxel hashing* (hash) [5], a variant of voxel hashing called *hierarchical hashing* (hhash) [26], and the traditional ordered method (ordered). In our experiments, the reconstructed size of ordered is set as $512 \times 512 \times 512$ voxels. For a fair comparison, SSL is also implemented based on this system. Thus the different results are mainly caused by different data management methods.

A. TUM RGB-D Dataset [24]

The TUM RGB-D dataset contains several sequences acquired from a Kinect in different scenes. Each sequence contains RGB images, depth images, and accelerator data. Some reconstructed models of SSL are shown in Fig. 7.

The numeric results are shown in TABLE. I. Among all methods, our proposed SSL gets the best performance on storage efficiency. What's more, the index data is kept in a small percentage on all sequence (less than 0.02%). As for other methods, the storage efficiency varies with different sequences, which means our SSL is more robust than these methods on storage efficiency. In the experiments, we run each sequence for three times and the average running time is used as the final speed. The average runtime per frame for different methods is also shown in TABLE. I. We can see that the hash achieves the best performance among all competitors. Our method (SSL) has the similar performance with hhash which is far enough for real-time applications. In addition, the ordered method is much slower than our method.

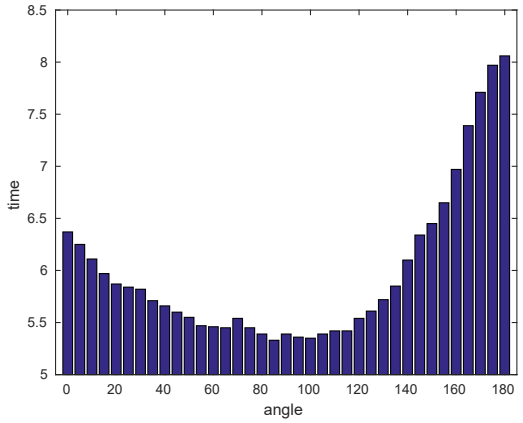


Fig. 9: The influence of principal direction setting for the running time. The interval of two adjacent principal directions is 5°. This experiment is performed on the sequence fr1_desk of TUM RGB-D dataset.

We test the influence of principal direction on sequence fr1_desk. The results are shown in Fig. 9. As we can see, the average runtime per frame varies with the principal direction obviously. Hence the setting of principal direction is of vital importance for algorithm efficiency.

B. ScanNet Dataset [25]

The ScanNet dataset consists of many sequences acquired from different indoor scenes. For each sequence, the same data is provided as in TUM RGB-D dataset. Some reconstructed models are shown in Fig. 8. We can come to the similar conclusion from TABLE. II. Specifically, SSL achieves the best storage efficiency on all sequences, and it is more robust than other competitors with only a little more

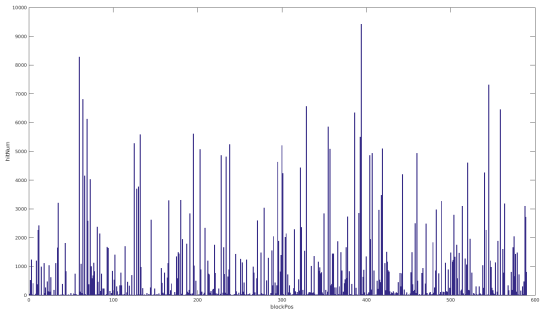


Fig. 10: The data collision in voxel allocation at the first frame in scene0013_00. The number of allocated voxel is 590. However, the number of check times for these voxels are 441187.

time consumption. The speed is already enough for real-time application. It is worth noting that the ordered fails on some sequences due to its limited reconstruction scales.

We test the data collision in voxel allocation on scene0013_00. This test is performed at the first frame of scene0013_00. We show the results in Fig. 10. From the figure, we can see that the check number for each block position is extremely high. If all checked voxel blocks are allocated, it is a huge burden for the system. What's worse, the VBA will be exhausted in a minute. This validates the necessity of HAL.

ACKNOWLEDGMENTS

This research was supported by NSFC (NO. 61620106008, 61572264), the national youth talent support program, Tianjin Natural Science Foundation for Distinguished Young Scholars (NO. 17JCJQJC43700), Huawei Innovation Research Program.

REFERENCES

[1] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, *et al.*, "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera," in *ACM Symposium on User Interface Software and Technology*. ACM, 2011, pp. 559–568.

[2] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, p. 24, 2017.

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

[4] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 169, 2013.

[5] O. Kähler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. Murray, "Very high frame rate volumetric integration of depth images on mobile devices," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 21, no. 11, pp. 1241–1250, 2015.

[6] O. Kähler, V. A. Prisacariu, and D. W. Murray, "Real-time large-scale dense 3D reconstruction with loop closure," in *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 500–516.

[7] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended kinectfusion," 2012.

[8] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense RGB-D mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 5724–5731.

[9] T. Whelan, M. Kaess, J. J. Leonard, and J. McDonald, "Deformation-based loop closure for large scale dense RGB-D SLAM," in *IEEE\RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 548–555.

[10] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense RGB-D SLAM with volumetric fusion," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015.

[11] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM without a pose graph." Robotics: Science and Systems, 2015.

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

[13] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *ACM SIGGRAPH Annual Conference (SIGGRAPH)*. ACM, 1996, pp. 303–312.

[14] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt, "Reliable surface reconstruction from multiple range images," in *European Conference on Computer Vision (ECCV)*. Springer, 1996, pp. 117–126.

[15] R. A. Newcombe, D. Fox, and S. M. Seitz, "DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 343–352.

[16] P. Puri, D. Jia, and M. Kaess, "GravityFusion: Real-time dense mapping without pose graph using deformation and orientation."

[17] T. Laidlow, M. Bloesch, W. Li, and S. Leutenegger, "Dense RGB-D-Inertial SLAM with map deformations," in *IEEE\RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6741–6748.

[18] T. Whelan, L. Ma, E. Bondarev, J. McDonald, *et al.*, "Incremental and batch planar simplification of dense point cloud maps," *Robotics and Autonomous Systems*, vol. 69, pp. 3–14, 2015.

[19] L. Ma, C. Kerl, J. Stückler, and D. Cremers, "CPA-SLAM: Consistent plane-model alignment for direct RGB-D SLAM," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1285–1291.

[20] P. F. Proença and Y. Gao, "Probabilistic RGB-D odometry based on points, lines and planes under depth uncertainty," *arXiv preprint arXiv:1706.04034*, 2017.

[21] J. Wang, J. Song, L. Zhao, and S. Huang, "A submap joining based RGB-D SLAM algorithm using planes as features," in *Field and Service Robotics*. Springer, 2018, pp. 367–382.

[22] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–607.

[23] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*. IEEE, 2001, pp. 145–152.

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

[25] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3D reconstructions of indoor scenes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 2017.

[26] O. Kähler, V. Prisacariu, J. Valentin, and D. Murray, "Hierarchical voxel block hashing for efficient integration of depth images," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 192–197, 2016.

[27] V. A. Prisacariu, O. Kähler, S. Golodetz, M. Sapienza, T. Cavallari, P. H. Torr, and D. W. Murray, "InfiniTAM v3: A framework for large-scale 3D reconstruction with loop closure," *arXiv preprint arXiv:1708.00783*, 2017.