REAL-TIME 3D RECONSTRUCTION OF DYNAMIC SCENES WITH MULTIPLE KINECT V2 SENSORS

Kai Zhou[1], Li Song[1], Jingchuan Hu[1], Shuai Guo[1], Yu Dong[1]

[2] Yanying Sun, Yesheng Xu

[1] Institute of Image Communication and Network Engineering, Shanghai Jiao Tong University, China
[2] Intel Corporation, China

ABSTRACT

3D reconstruction of dynamic scenes is an important task in the field of multimedia which arouses widespread interest in recent years. Although several novel 3D applications have been unlocked with the increasing popularity of commercial RGB-D cameras like Microsoft Kinect, Azure, and Intel RealSense, real-time performance is still a big problem. In this paper, an open-solution, low-latency, real-time 3D reconstruction system is presented. The proposed system runs in a simple architecture where multiple Kinect v2 sensors are connected to a single computer. It consists of a series of modules including capture, alignment, and post-processing. The data stream of all sensors will pass through these modules in turn, generate point clouds, and finally fusion to a single 3D model. To improve the real-time performance, this system adopts a pipeline design that uses multiple threads to execute all modules concurrently. Besides, the first-in-first-out feature of the queue is utilized for data transmission and thread separation. At last, experimental results are presented to verify the system's effectiveness concerning the 3D reconstruction visual quality and real-time performance.

INTRODUCTION

Real-time 3D reconstruction has emerged as one of the active research topics in the field of multimedia and computer graphics. It can be applied to many novel applications such as free view video[1], human motion recognition, 3D game interaction, and augmented or immersive reality. Most of these applications have high requirements for real-time performance. However, although real-time reconstruction techniques are mature in the field of static object reconstruction, it is still a big challenge to reconstruct dynamic scenes limited by huge amounts of data and the complexity of algorithms. Obtaining the depth information of the target scene is the essential for dynamic scene reconstruction, which has made it widely studied in recent years.

According to the way of depth information acquisition, previous researches can be classified into two categories, active methods and passive methods. Passive methods generally use disparity information[2] and surrounding environment[3] to calculate depth map by stereo-matching algorithms. Passive methods do not interact with the reconstructed object in the reconstruction process so there is no interference between multiple sensors. In contrast, active methods acquire a depth map by emitting lasers or infrared light to the target object. However, considering that the accuracy and performance of the depth estimation algorithm in passive methods cannot meet the requirement of real-time systems, most researchers turn to active depth sensors such as Time-of-flight(TOF) sensors[4,5,6].

The existing 3D reconstruction systems based on active depth cameras have many drawbacks. First, a multi-sensor system is cumbersome to build due to numerous devices and complex system architecture. Next, as for real-time performance, the frame rate of the system decreases when the number of sensors increases and the scene becomes larger. In terms of practicality, most real-time reconstruction systems are not open-solution. These problems are satisfactorily addressed in this paper.

In this paper, we present an open-solution system based on multiple Kinect v2 sensors. The proposed system targets real-time reconstruction of dynamic scenes with the ability to acquire full 3D models of moving objects and generate realistic 3D models of dynamic scenes through pipelining multiple modules. With the integration of many optimizations such as multi-threading and GPU acceleration, each module achieves a balance between complexity and quality, making it realistic to reconstruct moving objects.

The main contributions of our work are as follows:

- A simple 3D reconstruction system with multiple Kinect v2 sensors that only uses one single convenient computer, which can reconstruct dynamic scenes in real-time.
- Combines multiple optimization methods including multi-threading and GPU parallelism thus achieves the sensors' inherent frame rate.
- The overall system solution will be released and kept updated on https://github.com/sjtu-medialab/Kinect3D

The remaining parts of the paper are organized as follows. First, the existing related works are discussed. Next, we describe the hardware architecture of the system. Then the pipeline of the system and the corresponding algorithms are described. Finally, the quality and real-time performance of the system are presented.

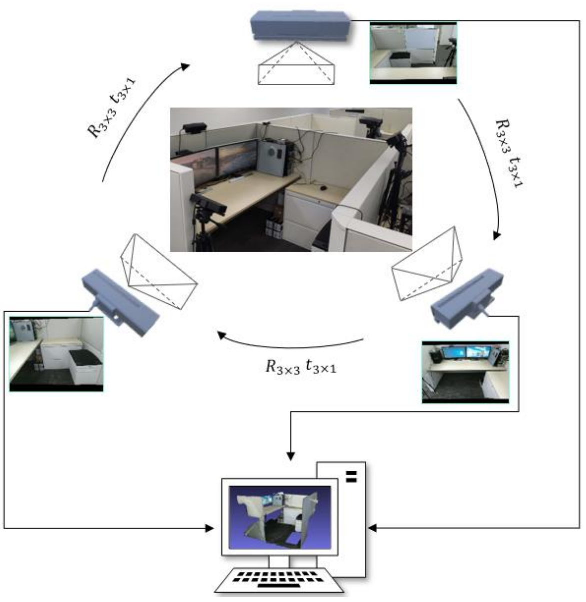
RELATED WORK

3D reconstruction has attracted a lot of attention in recent years. Due to the high complexity of stereo matching algorithms used for depth estimation and the rapid emergence of low-cost RGB-D cameras in recent years, RGB-D sensors are commonly used as data acquisition devices in real-time systems. Many of the relevant real-time reconstruction systems such as [4,5,6,7,8,9,10] refer to Microsoft Kinect sensors as data acquisition devices, since it provides acceptable resolution with affordable cost.

Microsoft has released three types of depth sensors: Kinect v1, Kinect v2, and azure Kinect. Their parameters are examined in [11]. Although the azure Kinect is the successor of the previous two generations of sensors, Kinect v2 sensor is still the most widely used sensor in research since the Azure Kinect was just released in March 2020. Based on Kinect sensors, many real-time systems are proposed.

LiveScan3D[5] presents an open-source 3D reconstruction system using multiple Kinect v2 sensors which can generate high-quality point clouds of target scenes with real-time performance. It adopts a server-client distributed system where the clients capture raw 3D point clouds and transmit them to the server. As a result, the frame rate gradually decreases when the size of point cloud data increases. The experimental results show that when the scene size is 2m*2m*2m, the frame rate will be reduced to below 10 fps.

In [4], a similar multiple depth stream-based system is proposed with a better real-time performance by employing an intra-frame compression scheme(JPEG for the RGB and LZ4 entropy compression for the depth maps). It also introduces a novel framework to evaluate the quality of real-time 3D reconstruction systems in the absence of ground truth. However, both [4] and [5] use a slave-master architecture which uses multiple computers to distribute the computational load. This makes system deployment more complicated and cumbersome.


$R_{3\times3}\ t_{3\times1}$
$R_{3\times3}\ t_{3\times1}$
$R_{3\times3}\ t_{3\times1}$
Figure 1 – System hardware architecture

Andrej[6,9] proposes a much simpler system in which several Kinect v2 cameras are connected to a single computer. This system achieves improved performance by integrating GPU parallel computing. Experiment results show that it processes each camera in an average of 14.18ms. The disadvantage of such a system is obvious, when the number of sensors is more than two, the frame rate cannot meet the real-time requirement.

The system proposed by this paper is designed concerning previous works, integrating the advantages of each system, adopting the simplest system architecture, and conducting a lot of optimizations for real-time performance. As a result, achieving the best performance at acceptable quality.

OVERVIEW OF THE SYSTEM

As demonstrated in Fig.1, the capturing system is composed of multiple Kinect v2 sensors. The Kinect v2 sensors are positioned on a circle of radius $r \in [0.5m, 4m]$ and oriented toward the center. Each sensor consists of a regular RGB camera and a depth scanner, which can generate color and depth maps simultaneously. The RGB camera is used for the acquisition of color maps with a 1920x1080 pixels resolution and the depth scanner captures depth maps with a resolution of 512x424[12].

In terms of the software development kit(SDK), the official SDK provided by Microsoft limits its usage to one sensor per computer, which inevitably leads to a complex structure of multi-sensor systems. To simplify the system, the system uses an open-source third-party driver libfreenect2[13], which has been reported to support up to 5 devices through USB3.0.

To capture the RGB-D stream simultaneously, we allocate a separate thread for each sensor. Since multiple infrared projectors work simultaneously, the infrared rays emitted by different sensors may interfere. However, as confirmed by [14], it has been proved that interference is not as strong as expected. Moreover, according to the experimental results of [15], interference is mainly manifested in the absence of a few depth pixels at object boundaries. This has little impact on the system because the overlap areas between adjacent sensors can compensate for missing values.

Fig.2 presents the whole pipeline of the system, which consists of multiple stages that start with capturing RGB-D streams and end with an interactive 3D rendering window created by OpenGL.

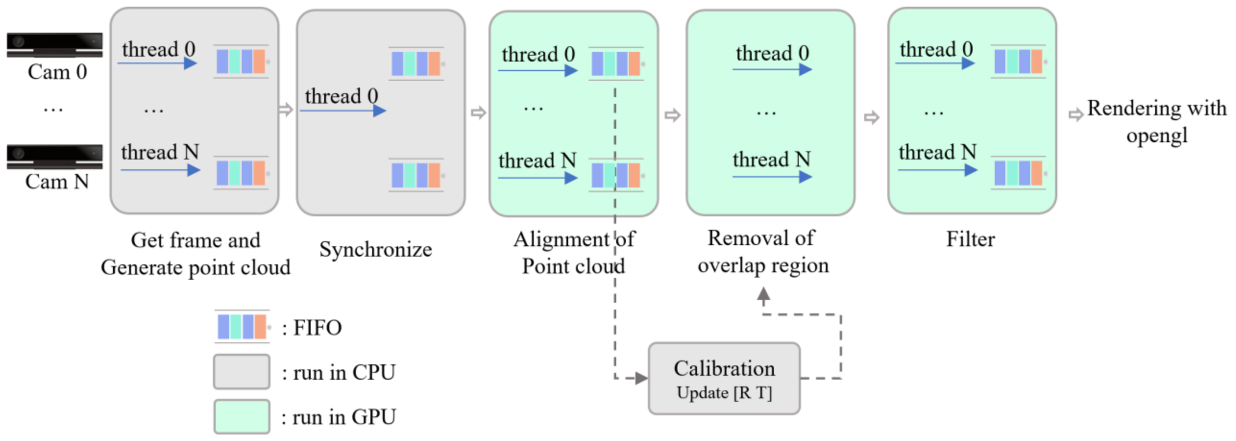


Figure 2 – Pipeline design of the system

THE REAL-TIME 3D RECONSTRUCTION ALGORITHM

In this section, we describe the pipeline and implementation details of the proposed system. According to the direction of data flow transmission and processing, this section mainly describes four modules of the system in order, which are synchronization, extrinsic calibration, removal of overlap regions, and post-processing.

Synchronization

Although the official document gives a frame rate of 30 fps, it is observed that the frame rate can fluctuate in practice. Therefore, the system allocates a thread dedicated to synchronizing all sensors. To synchronize the frames captured by different sensors, we employed a post-synchronize procedure which can keep the frame interval within 16 ms and the procedure is as follows.

Considering the depth and color pairs are generated with timestamps, we denote the timestamps of RGB-D frames as $t^i, i = 1, \dots, N$, where N is the total number of sensors. When all cameras are started, records the current timestamp as t_0. Then calculate timestamp interval for each sensor:

$$t^i_{int} = t^i - t_0, i = 1, \dots, N$$

For all timestamp intervals, if exist two sensors i and j satisfy:

$$t^i_{int} - t^j_{int} < threshold$$

then drop the corresponding overrunning frame. Here the value of $threshold$ is set to 16 ms, which is half a frame interval.

Extrinsic Calibration

Calibration refers to the process of registering point cloud data from the camera coordinate system to the world coordinate system. The transformation relationship can be described by a rotation matrix R_{3x3} and a translation vector t_{3x1}, called extrinsic matrix. For each sensor, an extrinsic matrix [R t] has to be calculated.

During the calibration stage, we use a 2D visual marker system proposed in [16] which is also used in [5]. When all sensors are oriented to the same center, which is assumed in this system, only one marker is needed. And for convenience, this system establishes the origin of the world coordinate system at the center of the marker. The marker is required to appear in all of the sensors. Each sensor will detect five corner points in the field of view as feature points, once the 3D location of the marker is known, the transform matrix that transforms any point x_s from the sensor's coordinate system to a point x_m in the world coordinate system can be calculated:

$$x_w = R_s(x_s - t_s)$$

where t_s is the translation matrix and R_s is the rotation matrix.

The disadvantage of marker calibration is that it will be affected by the detection accuracy of key points on the plane. In order to reduce the matching error, this system use iterative closest points(ICP)[17] to refine the initial estimate. Let R_r be the rotation matrix and t_r the translation matrix, the two steps transform relationship can be combined into:

$$x_w = R_r(R_s(x_s - t_s) + t_r)$$

In the refining stage, an offline thread is deployed due to the high complexity of the ICP algorithm. As long as the refine button is pressed, the pipeline outputs a synchronous frameset for ICP. After the refining stage is finished, the current frame will be discarded and the extrinsic parameters will be updated. The offline mode ensures that the frame rate of the system is not affected during the refining process.

Removal of Overlapping Regions

The point clouds generated from different sensors inevitably have a large portion of overlap regions, especially between two adjacent cameras, resulting in many redundant and mismatched data in overlapping areas. In this system, the removal of overlap regions is performed between every two adjacent sensors. The methodology applied in this system is similar to [15,18], which maps the points of one sensor to another through the external parameter matrix. In [15], removal of overlap regions is the bottle-neck part due to its iterative strategy, normally takes up to several hundreds of milliseconds, which forces them to adopt a coarse to fine strategy. This system abandons the iterative features

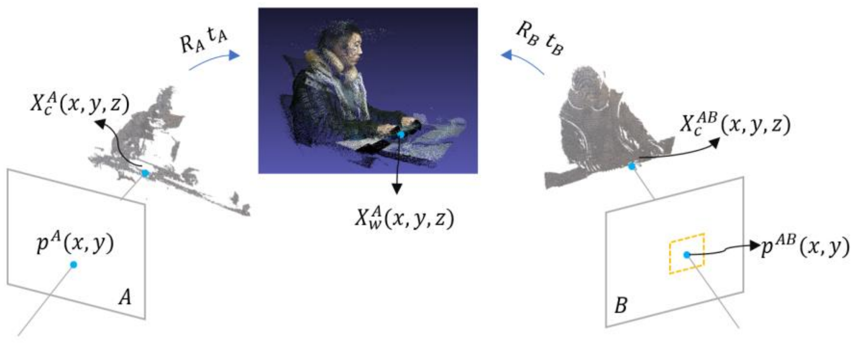


Figure 3 – Removal of Overlap Point cloud Regions

and implements the algorithm in GPU considering that the mapping of each pixel is independent, thus reduces execution time dramatically, just taking a few milliseconds.

Let A and B be two adjacent sensors, the algorithm implemented in this system is illustrated as Fig.3, which will remove the area that overlaps with B in A. This algorithm can be summarized as follows:

1) **Forward mapping**: For each pixel $p^A(x,y)$ in A's depth map, there is a corresponding 3D point $X_c^A(x,y,z)$ in the camera coordinate system. To establish mapping relationship with camera B, $X_c^A(x,y,z)$ must first be forward mapped to the world coordinate system:

$$X_w^A(x,y,z) = R_A(X_c^A(x,y,z) + t_A)$$

where R_A and t_A are sensor A's external matrix.

2) **Backward mapping**: Backward mapping has two steps, the first step is to project $X_w^A(x,y)$ to camera coordinate system of B, the projected points from A to B are

$$X_c^{AB}(x,y,z) = R_B^{-1}X_w^A(x,y,z) - t_B$$

The second step is to project $X_c^{AB}(x,y,z)$ to the 2D depth image. The 2D projection points are:

$$p^{AB}(x,y) = \Pi\{K_B, X_c^{AB}(x,y,z)\}$$

where Π represents the 3d to 2D projection and K_B is the internal parameters of camera B.

3) **Overlap determination**: Considering a patch centered on $p^{AB}(x,y)$, when the depth of the points in the patch and $p^{AB}(x,y)$ is less than a threshold T_r, it means that the current point overlaps with B, then remove $p^A(x,y)$ from A. The threshold T_r used in this system is 30mm.

Post processing

Due to the limitations of the TOF camera, there are many missing and unstable pixels in the Kinect v2 depth map, so the reconstructed model contains a lot of noise. Considering the high complexity of the filtering algorithm for point clouds, it is one of the major difficulties in achieving real-time frame rates.

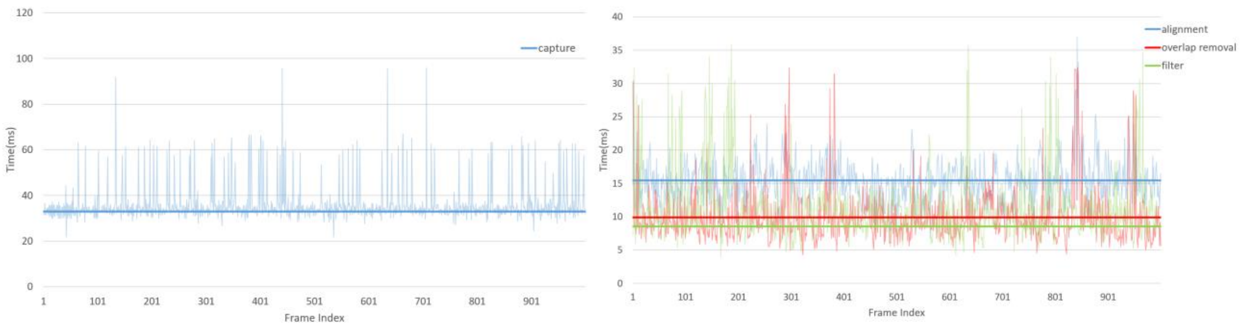
Inspired by the reconstruction algorithm "Step Discontinuity Constrained (SDC) triangulation" proposed in [19], this system applies the step discontinuity constrain in filter and implements an algorithm named “SDC filter”. Considering the pixels of the depth map and the 3D points are in a one-to-one correspondence, and the neighborhood relationship between depth pixels represents the topology of the 3D space points. The algorithm can be described as follows:

For each pixel $p = I_z(x,y)$, considering its adjacent pixels at the top, down, left and right, $p_t = I_z(x,y-1)$, $p_d = I_z(x,y+1)$, $p_l = I_z(x-1,y)$ and $p_r = I_z(x+1,y)$. These five points can generate four potential triangles. Take $S_1 = \{p, p_t, p_l\}$ as an example, it will be generated if it satisfies $|p - p_t| < t$ and $|p - p_l| < t$ and$|p_l - p_t| < t$, where t is a threshold, and the value of t is 0.015m in this paper. If none of these four potential triangles are generated, then the depth pixel $p = I_z(x,y)$ will be removed and so is the corresponding 3D point.

EXPERIMENT

In this section, we present the experimental results of the proposed system. The system is written in C++ and CUDA C with the operating system Linux Manjaro 64bit. The whole system runs in a single computer with an i9-10900K processor, 32-GB RAM, and a CUDA-enabled NVIDIA RTX 2080Ti GPU.

Real-time Performance



(a) (b)

Figure 4 – Performance test of each module. (a) Acquisition RGB-D from sensors. (b) Alignment, overlap removal and filter modules

The real-time performance of the proposed system is tested by measuring the processing time of each module in the system. The experimental condition is 3 sensors and the scene size is set to 1.5m*1.5m*1.5m. To make the results more reliable, we capture 1000 consecutive frames and track the latency of each frame. The result is demonstrated in Fig.4. Then we calculate the average latency of each module in Tab.1. Due to the pipeline design pattern, all the modules in this system are highly parallel, so the frame rate is limited only by the most time-consuming module. As shown in Tab.1, the camera acquisition module takes the longest time which is slightly over 33ms and the final average frame rate is calculated to be 25-27 fps. However, in most cases, the system works at 30 fps. The lower average frame rate is the result of frame loss by some unknown agnostic problem, which can be seen in Fig.4(a). Under normal circumstances, the interval between two frames captured by the sensor is 33ms, but in some occasional cases, the interval between two frames is 66ms or even 99ms. This is a problem caused by the hardware, so we can still state that this system achieves the inherent sensor frame rate of 30 fps in terms of real-time performance.

Table 1. Performance results of the system pipeline

Sensors	Scene size(m^3)	Execution time(ms)						Frame rate
		capture	synchronize	alignment	overlap removal	filter	render	
1	1.0*1.0*1.0	37.447	—	4.350	—	1.995	17.381	26.674
	1.5*1.5*1.5	37.748	—	4.286	—	1.905	17.691	26.460
	2.0*2.0*2.0	37.748	—	4.283	—	1.862	17.613	26.505
2	1.0*1.0*1.0	37.951	0.068	7.360	5.826	5.258	18.518	25.897
	1.5*1.5*1.5	37.083	0.099	7.158	5.496	5.872	17.308	25.897
	2.0*2.0*2.0	38.449	0.001	7.699	5.418	5.507	18.893	25.637
3	1.0*1.0*1.0	36.582	0.030	14.495	9.724	8.510	17.006	26.935
	1.5*1.5*1.5	36.182	0.105	15.471	9.911	8.510	16.876	27.002
	2.0*2.0*2.0	37.216	0.999	13.828	9.911	9.756	17.401	26.645

Reconstruction Quality

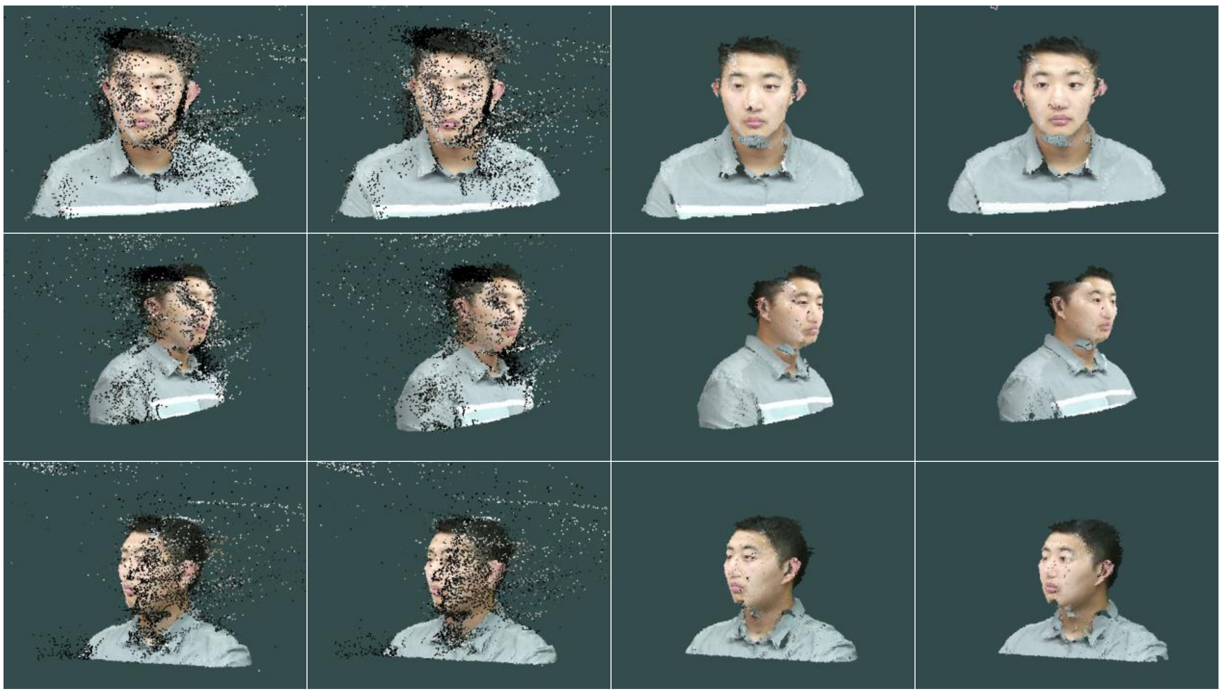
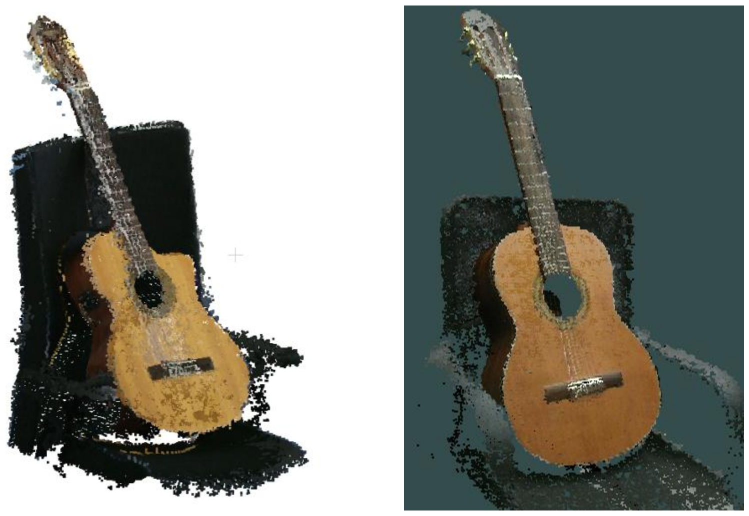


Figure 5 – Reconstruction quality. (a) Initial point cloud captured from kinect v2 sensors. (b) Result after overlapping regions removal. (c) Result after post-processing. (d) Final result combining both overlapping regions removal and post-processing.

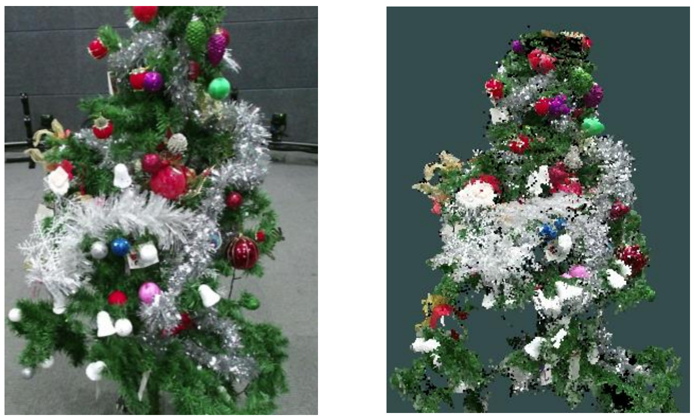
The proposed system is a combination of multiple techniques. Fig.5 shows how each technique improves the reconstruction quality. As Fig.5(a) shows, where we simply acquire RGB-D streams from each sensor and register to world coordinate. The reconstructed model obtained has a lot of discrete noisy points, especially at the edges of the object and in the regions where the depth distance is far resulting in missing depth pixels. Fig.5(b) shows the result of removing the redundant data in the overlap area of the sensor. Next in Fig.5(c), the quality can be greatly improved by using the SDC filter algorithm. Fig.5(d) is the final result combining all methods and achieves the best visual quality.

Besides, to make the results in this paper comparable, we build almost the same scene as in LiveScan3D[5] and give the reconstructed results in Figure 6. As can be seen, the reconstruction results in this paper have less noise and smoother surfaces, mainly benefited from the removal of overlap regions and the step discontinuity constrained filtering algorithm. Also, this system gives favorable results in complex environments, as shown in Fig.7.



(a) (b)

Figure 6 – Comparison of reconstruction results with LiveScan3D (a) the 3D reconstruction results given by LiveScan3D (b) the results given in this paper



(a) (b)

Figure 7 – A scene featuring a Christmas tree (a) the color image (b) the reconstruction result from one viewpoint

CONCLUSION

In this paper, we have implemented a low-cost system that supports multiple Kinect v2 sensors connected to a single computer. The advantages of this system are high frame rate and concise architecture. This is mainly ensured by two parts. At the implementation level, a pipeline design pattern is used, with modules parallelized and separated by FIFO, and integrated with CPU multi-threading and GPU parallel acceleration. At the algorithm level, lightweight algorithms are used such as point cloud filtering based on depth continuity. Finally, as the result shows, the entire system can achieve the camera's intrinsic frame rate, and no other system to date has been able to achieve that to the best of this paper's knowledge.

In the future, we plan to integrate more sophisticated algorithms to improve the quality of the point cloud and explore the feasibility of real-time surface reconstruction.

REFERENCES

[1] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sulli-van. High-quality streamable free-viewpoint video. ACM Transactions on Graphics (ToG), 34(4):1–13, 2015.

[2] Sudipta N. Sinha. Multiview Stereo, pages 516–522. Springer US, Boston, MA, 2014.

[3] Berthold KP Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. 1970

[4] Dimitrios S Alexiadis, Anargyros Chatzitofis, Nikolaos Zioulis, Olga Zoidi, Georgios Louizis, Dimitrios Zarpalas, and Petros Daras. An integrated platform for live 3d human reconstruction and motion capturing. IEEE Transactions on Circuits and Systems for Video Technology,27(4):798–813, 2016

[5] Marek Kowalski, Jacek Naruniec, and Michal Daniluk. Livescan3d: A fast and inexpensive 3d data acquisition system for multiple kinect v2 sensors. In 2015 international conference on 3D vision, pages 318–325.IEEE, 2015.

[6] Andrej Satnik, Ebroul Izquierdo, and Richard Orjesek. Multiview 3dsensing and analysis for high quality point cloud reconstruction. In Tenth International Conference on Machine Vision (ICMV 2017), volume10696, page 106962K. International Society for Optics and Photonics,2018.

[7] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, 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, pages 559–568, 2011

[8] Xiongfeng Peng, Liaoyuan Zeng, Wenyi Wang, Zhili Liu, Yifeng Yang, Zhen Zeng, and Jianwen Chen. A robust and real-time full 3d reconstruction method based on multiple kinect. In International Conference in Communications, Signal Processing, and Systems, pages 1420–1428. Springer, 2017

[9] Andrej Satnik and Ebroul Izquierdo. Real-time multi-view volumetric reconstruction of dynamic scenes using kinect v2. In 2018-3DTV-Conference: The True Vision-Capture, Transmission and Display of 3DVideo (3DTV-CON), pages 1–4. IEEE, 2018

[10] Andrej Satnik, Robin Ribback, Krishna Chandramouli, GiacomoInches, Mark Wheatley, and Ebroul Izquierdo. Comparative analysis of real-time multi-view reconstruction of a sign language interpreter

[11] Michal Tölgyessy, Martin Dekan, L'uboš Chovanec, and Peter Hubin-skỳ. Evaluation of the azure kinect and its comparison to kinect v1 and kinect v2. Sensors, 21(2):413, 2021

[12] E Lachat, H Macher, MA Mittet, T Landes, and P Grussenmeyer. First experiences with kinect v2 sensor for close range 3d modelling.The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 40(5):93, 2015

[13] Lingzhu Xiang, Florian Echtler, Christian Kerl, Thiemo Wiedemeyer, Lars, hanyazou, Ryan Gordon, Francisco Facioni, laborer2008, RichWareham, Matthias Goldhoorn, alberth, gaborpapp, Steffen Fuchs, jmtatsch, Joshua Blake, Federico, Henning Jungkurth, Yuan Mingze, vi-nouz, Dave Coleman, Brendan Burns, Rahul Rawat, Serguei Mokhov, Paul Reynolds, P.E. Viau, Matthieu Fraissinet-Tachet, Ludique, James Billingham, and Alistair. libfreenect2: Release 0.2, April 2016.

[14] Andrew Maimone and Henry Fuchs. Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras. In 2011 10th IEEE International Symposium on Mixed and Augmented Reality, pages 137–146. IEEE, 2011

[15] Dimitrios S Alexiadis, Dimitrios Zarpalas, and Petros Daras. Real-time, full 3-d reconstruction of moving foreground objects from multiple consumer depth cameras. IEEE Transactions on Multimedia,15(2):339–358, 2012.

[16] Tomasz Rybus, T Barciński, J Lisowski, J Nicolau-Kukliński, K Sew-eryn, M Ciesielska, K Grassmann, J Grygorczuk, M Karczewski, M Kowalski, et al. New planar air-bearing microgravity simulator for verification of space robotics numerical simulations and control algorithms. In proceedings of 12th Symposium on Advanced Space Technologies in Robotics and Automation, 2013

[17] Paul J Besl and Neil D McKay. Method for registration of 3d shapes. In Sensor fusion IV: control paradigms and data structures, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992

[18] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 311–318, 1994

[19] Adrian Hilton, Andrew J Stoddart, John Illingworth, and Terry Windeatt. Reliable surface reconstruction from multiple range images. In European conference on computer vision, pages 117–126. Springer,1996.