Real-Time 360° Body Scanning System for Virtual Reality Research Applications

Louis ALBERT[2,■], Florian LANCE[1,■], Margaux DUBESSY[2],
Bruno HERBELIN[*1], Gilles REYMOND[2], Olaf BLANKE[1]
[1] Laboratory of Cognitive Neuroscience, Ecole Polytechnique Fédérale de Lausanne, Campus Biotech, Geneva, Switzerland;
[2] VR Facility, Foundation Campus Biotech Geneva, Geneva, Switzerland

https://doi.org/10.15221/19.150

Abstract

Here, we present a low-cost solution to perform the online and realistic representation of users using an array of depth cameras. The system is composed of a cluster of 10 Microsoft Kinect 2 cameras, each one associated to a compact NUC PC to stream live depth & color images to a master PC which reconstructs live the point cloud of the scene and can in particular show the body of users standing in the capture area. A custom geometric calibration procedure allows accurate reconstruction of the different 3D data streams. Despite the inherent limitations of depth cameras, in particular sensor noise, the system provides a convincing representation of the user's body, is not limited by changes in clothing (also during immersion), can capture complex poses and even interactions between two persons or with physical objects. The advantage of using depth cameras over conventional cameras is that little processing is required for dynamic reconstruction of unknown shapes, thus allowing true interactive applications. The resulting live 3D model can be inserted in any virtual environment (e.g. Unity 3D software integration plugin), and can be subject to all usual 3D manipulation and transformations.

Keywords: 360° body scanning, virtual reality, volumetric capture, depth camera, real-time

1. Introduction

A major challenge in immersive virtual reality is the online and realistic representation of the user(s). A widespread solution is to animate synthetic characters with live motion capture of participants, thus facing limitations in the level of realism of the underlying 3D model and requiring a tedious design work for making look-alike avatars. Animations in real-time motion capture also often leads to animation artefacts due to inverse kinematic problems, or occlusions that could happen during capture. Pre-created characters are also mostly not responsive to physiological changes of the real person (ex: when breathing), nor adapted to changes in the person's appearance (ex: clothes manipulation). This is notably problematic for VR scenarios requiring users to see their own body immersed in the scene, or to interact with other participants. Recently, the release of affordable 3D sensing technologies has opened a new field of research for capturing volumetric scenes and rendering scenes from the point clouds of captured data. Point clouds can be defined as a set of data points in three dimensions representing the external surfaces of all visible objects of a scanned area [1]. It has recently became a viable solution for real-time rendering, and provides a practical way for capturing, transferring and rendering. Here, we present a low-cost solution, based on an array of depth cameras, to create a 360° volumetric capture of one or two persons in real-time.

2. State of the art

Last decade has been marked by the introduction of new and inexpensive depth cameras for 3D acquisition and reconstruction, such as Microsoft Kinect™ (v1 and v2), Creative Senz3D™, or Softkinetic DepthSense™. Such systems allow real-time data acquisition and are now widely adopted by many researchers working in the field [2]. There are two widespread types of depth cameras. The first one can be found on the Microsoft Kinect v1 and is based on structured light. It consists in projecting a known pattern onto a three-dimensionally shaped surface, which gets distorted when seen from other perspectives than of the projector. The depth information is extracted by analyzing the disparity between the original projected pattern and the observed pattern. The second type can be found on the Microsoft Kinect v2 and is based on time-of-flight principle [3]. It consists in sending a light pulse for a very short time onto a three-dimensionally shaped surface; the light pulse illuminates the surface and is reflected onto the sensor. The depth information can be extracted by measuring the time difference between the emission of the light pulse and its return to the sensor. For a system with multiple camera, Structured

■ Both authors contributed equally to this work

Light is not appropriate because of the interferences between the overlapping structured light projected by each camera. Mechanical fix using vibrator motors put on each camera during data acquisition have been used to prevent the Microsoft Kinect v1 interferences when using multiple devices at the same time [4], but this method deteriorated the data (as sensor is vibrating) and have not been tested with a large number of devices. Microsoft Kinect v2 cameras are currently the most affordable and performant time-of-flight depth cameras, are not subject to this type of interference, and provides also better data than its predecessor—see Sarbolandi et Al. [5] for a detailed comparison between Microsoft Kinect v1 and Microsoft Kinect v2. Time-of-flight cameras are however subject to another type of interference, called multipath interference. By principle, rays of light are sent out from the cameras for each pixel, but as light can be reflected of surfaces in numerous ways, a particular pixel may receive light originally sent out from other pixels [6]. This effect increases when using multiple devices at the same time. By firmware implementation, to limit this effect, Microsoft Kinect v2 randomly cycles through different modulation frequencies. There is, however, no way to access such low-level functions allowing the user to select or modify the modulation frequency on the Microsoft Kinect v2 from the original firmware. A special firmware fixing the multipath interference problem during the use of multiple Microsoft Kinect v2 has been developed and reported by Microsoft [7], but has never been released to the public.

3D data acquisition system using one or more RGB-D cameras (depth sensing device that work in association with RGB color camera) have previously been reported [8,9,10,11]. A more complete and recent review on 3D Reconstruction with RGB-D cameras has also been reported [12], and provides a complete state of the art of the multiple possible approaches, with technical details of existing systems. Work from Microsoft Research has also described an end-to-end system for augmented reality and virtual reality telepresence, providing real-time and high-quality 3D reconstruction of a scene with depth cameras [13]. These works do not however explain how to overcome the technical limitations for connecting multiple sensors, with e.g. Microsoft drivers limiting the support of a single Kinect v2 at a time. Although some custom drivers [14] make it possible to use multiple Kinect v2 devices on a single computer, strong hardware limitations remain (notably on the PCI-E bus), affecting the maximum number of devices that can be linked to a single client computer and limiting the capture transfer rate. Kowalski et al. [6] have therefore designed a free and open source system allowing 3D reconstruction in real time in the form of a colored point cloud from information captured by multiple Kinect v2 simultaneously. However, their system lacks some useful features that we introduce.

3. System description

3.1. Architecture

Considering the limitations found in previous implementations [6], our system is designed to be scalable and versatile. In particular the system should work with any kind of RGB-D cameras, which can be placed in any physical configuration, and can be upgraded in number without limitations. Our solution for 360 volumetric capture integrates several RGB-D cameras in a local network of dedicated computers, allowing the integration of point clouds in 3D on a central graphical computer (Fig.1). This server computer is independent from the clients, not only for practical maintenance reasons, but also to allow connecting different servers and for distributing equally the computational load.

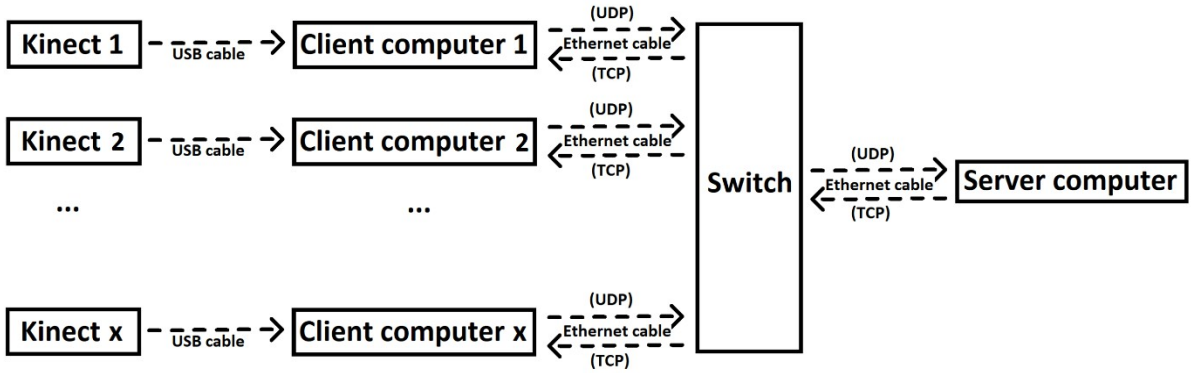

Figure. 1. Architecture diagram of the system

First, we choose a distributed architecture with one computer that acts as a server and several others that acts as client. This facilitates the creation of a similar set-up and the addition of new client computers. Indeed, all clients computers can have identical hardware and system in order to facilitate

the maintenance of the whole system. The server is a completely independent computer, with powerful graphics capabilities, and is dedicated to the rendering of the combination of all cloud points acquired by the clients. To the opposite, Kowalski et al. [6] require the computer that acts as main computer to also act as a client, making it harder to maintain and requiring the server computer to manage a Kinect v2. The server application they provide only allows changing and setting the same parameters for all client Kinects v2, which in practice is problematic depending the location of the cameras and the desired point of focus. A completely distributed architecture should provide the ability to configure each client Kinect v2 sensor independently from the server application, and should allow displaying the different clients Kinects v2 sensors streams (color, depth, infrared), all together or independently, from the server application. This way, once the clients are configured and the client application launched, no more access (physical or remote) is needed. In addition, this gives the possibility to easily replace the server computer, as it just needs to run the server application and does not require specific drivers or software.

Second, the server application from Kowalski et al. [6] synchronizes frame capture by signaling all client computers and waiting for their confirmation (that a frame has been captured) before sending the next capture signal. This distributed locking mechanism uselessly limits the framerate on the server side (at best to the 30FPS of Kinect v2). With an asynchronous sending of frames by the clients, the server rendering speed is independent from and can be higher than the Kinect v2 framerate, thus merging in the server the latest frames recorded by the clients even if they were captured at different time points. Such asynchrony of streams, combined with parallelization and multi-threading, allows the server application to perform a real-time acquisition and to display captured data sent over Ethernet in real-time at a higher frequency.

In our approach, the client application does all the processing needed to transform the data stream from the connected Kinect v2 sensors into an array of point clouds. The server application then retrieves the point clouds from all clients and merges them, thus focusing its workload on the rendering. In the implementation presented here, it is possible to place several RGB-D sensors in any physical configuration and with no constraint between cameras. Currently only the Kinect v2 is supported, but it would be straightforward to integrate any other RGB-D camera using its associated API and retrieving the correct sensor streams. The server application is agnostic to the hardware used, making it flexible and easily upgradable.

3.2. Data flow (preprocessing & streaming)

In our design, client computers running the client application are separated from the server computer running the server application.

For each frame captured by a Kinect v2 on a client computer, the client application transforms the acquired streams into a point cloud. In order to speed up this data processing, to remove useless and artefacts data, and to minimize the size of the data that are sent to the server, the client application applies some preprocessing on the raw Kinect v2 sensors' data. This preprocessing starts with removing pixels that are located outside of the depth range of interest, as specified in the settings. It lightens the data for the following processing, also deleting the artefacts outliers. An erosion is then made on the outcome of depth data to remove noise and artifacts, common with time-of-flight sensors which often give abrupt changes of depth. Two further processing can optionally be performed on the outcome of depth data. The first one consists in a k-neighbor filtering procedure, removing the points that have not enough neighbors. This can be useful to clean some noise induced by the depth sensor. The second one is a temporal filter that can be used to remove remaining depth sensors artifacts that are not consistent during continuous frames. It can also be used to smooth the visual feedback by increasing the frame rate of data send by each Kinect v2 thanks to quaternion interpolation of previous frames, but as a consequence, adding some delay to the visual feedback. After preprocessing, remaining depth pixels are mapped in 3D coordinates, and associated to a single color based on average colors of corresponding mapped pixels on the color stream (depth sensors have a lower resolution than color sensors).

To acquire a new frame on the server computer, the server application sends an appropriate message to all clients simultaneously. Upon reception of that message, all clients' applications send their latest computed point cloud without waiting to acquire a new frame. Data is formatted as 8 bytes point structure containing depth and color information. Common point structure is held on 16 bytes. 4 bytes are used for each 3D coordinate (X, Y, Z) which are stored as floats on 32 bits, and 1 byte is used for each color component (R, G, B) which are stored as integers on 8 bits. We make a lossy compression to transform this 16-bytes point structure into 8-bytes point structure by reducing the precision of calculated 3D coordinates. Such reduction of precision is not visible on global point cloud, and is far

below the Kinect v2 depth sensor inaccuracy. Technically, 5 bytes are used to store all 3D coordinate (14 bits for X, 14 bits for Y, and 12 bits for Z), and 1 byte is still use for each color component (R, G, B) which are stored as integers on 8bits. Finally, when the server receives all the data for from all the clients for last recorded frame, it applies appropriate 3D integration to merge them.

3.3. 3D integration (calibration, rendering)

In order to display the combined point cloud of all sensors in the same coordinate system, a calibration is necessary. The calibration takes place on the server, and the result correspond to a specific transform for each client point cloud that is applied each frame on the server before rendering.
Kowalski et al. [6] use a pattern-based calibration independently for each client. A pre-defined calibration pattern picture is first placed in a way to be visible by all sensors, and an Iterative Closest Point (ICP) [15] refinement is then executed and stored independently for each client. This approach does not allow important optimizations that can be obtained by combining information from multiple viewpoints. Here we propose a single ICP centralized calibration procedure that uses data from all clients, and where calibration settings are managed and stored on the server application. During calibration, data is acquired for a short period so as to capture a dozen of frames. During this acquisition, a colored circular marker must be moved inside the tracking zone while remaining visible by all Kinect v2. For our calibration, we use a red marker, and apply a color thresholding to make it the only visible object seeable by Kinect v2 color sensors. After this acquisition, for each frame and for each Kinect v2 color sensor frame containing the extracted colored marker, the server creates a 3D point cloud (thus containing only the colored marker). The first client Kinect v2 3D point cloud of the colored marker is set as the reference. For each other client, one at a time, its 3D point cloud is transformed to best match the reference. Doing this on multiple different frames gives the mathematical transformation of one client point cloud coordinate system compared to the first client point cloud coordinate system. The server can then apply corresponding geometrical transformations to each client's point clouds, and align all coordinate systems to the first client reference. Pairwise calibration between two Kinect v2 enables fine-tuning of the geometrical transformations.

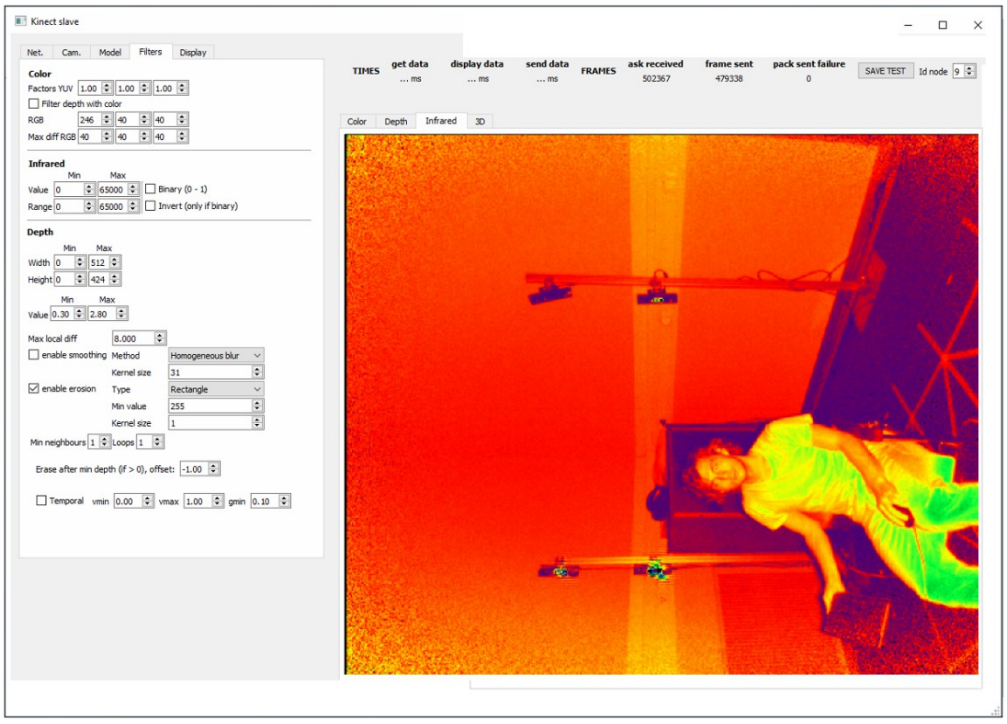

Figure. 2. Example view of the client application. Left pane shows the configurable filters parameters.

3.4. Implementation

The software consists in a client application for the clients connected to a Kinect v2, and in a server application executed on a server graphics computer.

3.4.1. Client application

The client application controls one Kinect v2 device, and allows configuring capture parameters such as setting threshold for depth limitation or selecting filtering algorithms (see Fig. 2). For practical reasons, it can also display the data acquired by the Kinect v2 sensors (infrared, depth, raw) and the reconstructed point cloud in real time. However this application is mostly a remote client that waits for the instructions of the server.

3.4.2. Server application

The server application manages all clients found in its local network (Fig. 1). It allows configuring each client parameters independently though TCP packets. Parameters can be sent to all clients simultaneously, or to specific clients only, thus enabling the finetuning of each Kinect v2 filters depending on individual physical constraints and configurations.
The server application is able to display Kinect v2 sensors stream of all clients, as well as all the computed point clouds (Fig.3). It performs the coordinate system calibration and can show a real-time view of all clients point clouds, combined with the same landmarks. The multi-threaded implementation of the server application allows receiving data send by clients asynchronously, and displaying the reconstructed scene at a high frame rate with one vertex per point. As all clients Kinect v2 are not time dependent between them, and thanks to some additional temporal-filtering, the server is able to render a smoothed real-time reconstructed scene at over 90FPS.

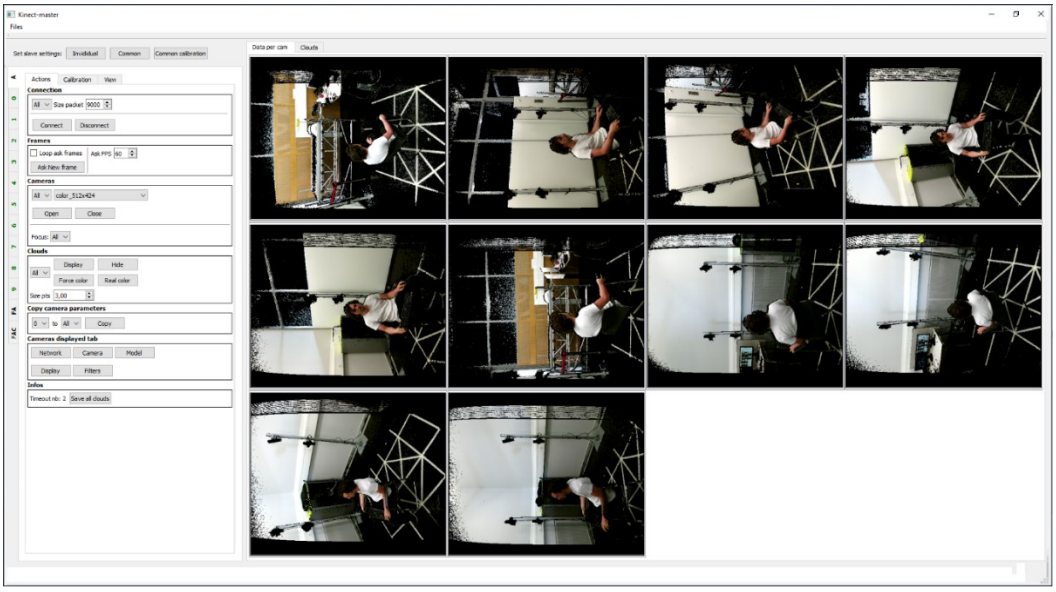

Figure. 3. Example view of the server application. Right Pane shows all clients Kinect v2 color streams in real-time.

3.4.3. Unity 3D plugin

A Unity 3D plugin has been developed to facilitate the use of the system. It is a specific implementation of the server application, and offers the same functionalities (without the graphical user interface). Based on a prepared configuration, it is used to display the reconstructed scene in real-time and at high framerate inside a Unity 3D scene (Fig 4).

Figure. 4. Reconstructed scene displayed inside Unity 3D in real time

4. Tests and Results

Four our needs, we designed a system composed of 10 Microsoft Kinect v2 placed in a facing inward configuration (Fig. 5). Each Microsoft Kinect v2 is linked to a unique computer (DELL OptiPlex 7050 Micro – Intel core i5 7500T, 8Go ram), running the client application. Another computer (Intel Xeon Silver 4110, NVIDIA GeForce GTX 980, 64Go RAM) is running the master application. All these computers are on the same local network, being linked together with Ethernet cables (1Gb/s) and a switch (CISCO SG100-24, max bandwidth 1Gb/s per port). In order to avoid saturating the bandwidth and to distribute the traffic of client computers data, our master computer is equipped with a 4 Ethernet port PCI-E card (max bandwidth 10Gb/s per port), each Ethernet port receiving the data of maximum 3-clients simultaneously.

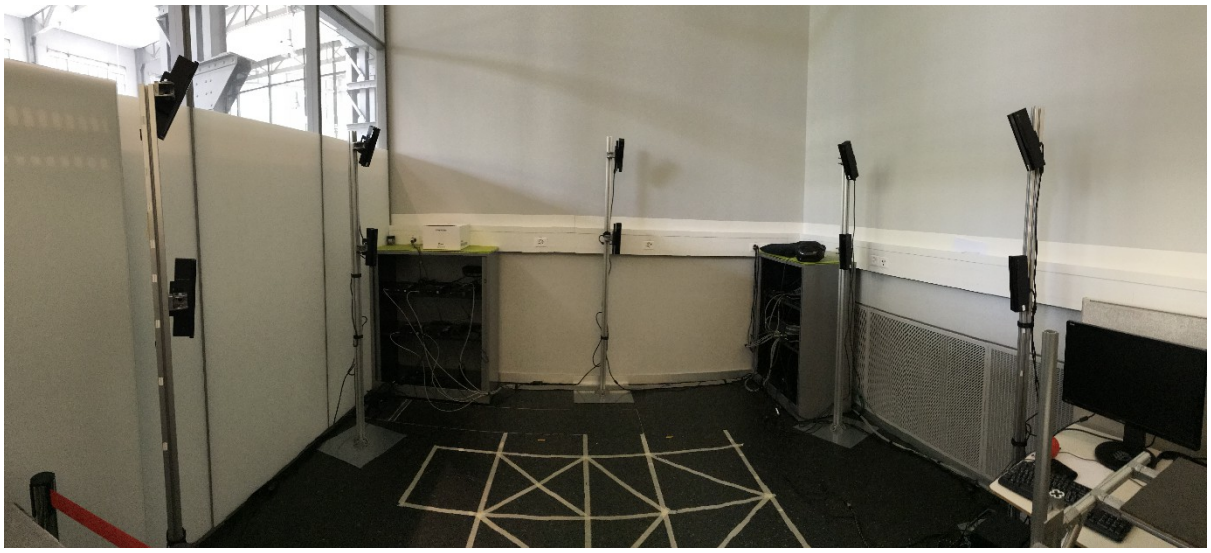

Fig. 5. Configuration of the 10 Kinect v2 in the room (facing inward configuration)

For testing purpose, we built a simple Unity 3D scene representing an empty room, where we displayed our point cloud. We report some scenes examples annotated with its number of vertices in Table 1. The capture quality is unavoidably limited by the quality of the Kinect v2 data, but the scalability of the system is demonstrated with its ability to maintain 90FPS with a high number of vertices.

		
1 person ≈ 170000 vertices	2 people ≈ 220000 vertices	6 people ≈ 470000 vertices

Table 1. Real-time reconstruction inside Unity 3D. Number of vertices corresponds to the total number of points displayed from all Kinect v2 point cloud.

The system was successfully combined in Unity3D with virtual reality headsets. However, only VR headset based on Simultaneous Location and Mapping tracking technology (e.g. Samsung Odissey) are compatible. Others systems based on IR tracking (HTC Vive or Oculus Rift) are too much susceptible to perturbations from the Kinect v2 illumination.

5. Conclusion and perspectives

We have presented a low-cost solution to create 360° volumetric view of one or more users in real-time using an array of Microsoft Kinect v2 devices. Our current system is composed of a server computer running the server application linked to several client computers, each running the client application and managing a single Kinect v2 device. It allows reconstructing in real-time the point cloud of a scene and can in particular shows the body of users standing in the center of the capture area. It provides a very convincing representation of the user's body, is not limited by changes in clothing (also during immersion), and can capture complex poses and even interactions between several persons or with physical objects.

As a simple update for our system, it would be straightforward to upgrade the 1Gb/s Ethernet network to 10Gb/s; by increasing maximum bandwidth per port, we would also increase the maximum number of Kinect v2 that can be handled by the system, and/or support higher-resolution sensors. This would be required for the future planned upgrade to replace Kinect v2 sensors with the new incoming and promising Kinect azure sensors. As Kinect Azure depth sensor resolution run with a 640x576 resolution at 30FPS (compared to Kinect v2 depth sensor resolution of 512x424 at 30FPS), data to be transmitted from a client computer to the master computer would be around 1.7x bigger. Thanks to the flexibility of our architecture, we should be able to handle directly our 10 client computers updated with Kinect Azure devices in a 10Gb/s network. In effect, this sensor upgrade will greatly increase the precision of the visual reconstructed scene by increasing the number of captured points in a same area.

One limiting feature of the Microsoft Kinect's RGB-D sensors for scene reconstruction in real time is their maximum frame rate of 30FPS. The use of RGB-D sensors running above 60FPS would highly enhance the usability of such systems, and could democratize its use with novel technologies as Augmented Reality and Virtual Reality. Such sensors are sadly not democratized on consumer market and hardly affordable.

On the software part, it would first be possible to add a filtering pass on the master computer before rendering, so as to remove duplicated points reconstructed by different sensors due to visibility overlaps. Second, it would be possible to implement an advanced solution for skeleton tracking. Kinects V2 sensors can provide information on joints and movement of people in the field of view, that our system already allows to display. Advanced tracking solutions exists (markerless body tracking, marker-based object tracking) using multiple Kinect v2 devices, such as the one developed by Rietzler et al. [16] who designed an open source software providing low-cost tracking system. In their experiment on skeleton-based tracking, using a statistical nonlinear boosting model that allows predicting the magnitude of tracking errors with accuracy, they were able to reduce drastically the error compared to a single Kinect v2. Merging skeleton tracking with out system could be used to improve calibration (possibly online) and to enable advanced interaction with virtual objects.

Our system can be used for, but is not limited to, applications in experimental human research, where playing in real time and/or replaying recorded scenes from different perspectives can be used for memory or self-representation experiments. It could also be used in entertainment or visual effects, as recently highlighted by Intel™ who started last year the creation of the world largest 360° movie set (10.000-square-foot-geodesic dome outfitted with 96 high-resolution 5K cameras), allowing to record scenes inside the dome from all directions at once, using volumetric capture. One can expect that, in a close future, the real-time processing and rendering available with our approach will meet the high visual quality of this new hardware.

References

[1] E. Alexiou et Al, "On the performance of metrics to predict quality in point cloud representations", in *Applications of Digital Image Processing XL*, International Society for Optics and Photonics, Vol. 10396, 2017, pp. 103961H, https://doi.org/10.1117/12.2275142

[2] J. Mihal'Ov et Al, "Kinect: From Entertainment to Scientific Research on Virtual Movement", In proceedings of 20th International Student Conference on Electrical Engineering, POSTER 2016, Prague.

[3] C. D. Mutto et Al, “Time-of-flight cameras and microsoft kinect (TM)”, in *Springer Publishing Company*, Incorporated, 2012, http://dx.doi.org/10.1007/978-1-4614-3807-6

[4] D. Butler et Al, "Shake'n'sense: reducing interference for overlapping structured light depth cameras", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2012, pp. 1933-1936, https://doi.org/10.1145/2207676.2208335

[5] H. Sarbolandi et Al, "Kinect range sensing: Structured-light versus Time-of-Flight Kinect", in *Computer vision and image understanding*, 139, 2015, pp. 1-20, https://doi.org/10.1016/j.cviu.2015.05.006

[6] M. Kowalski et. Al, “Livescan3d: A fast and inexpensive 3d data acquisition system for multiple Kinect v2 sensors”, in *2015 International Conference on 3D Vision*, IEEE, 2015, pp. 318-325, https://doi.org/10.1109/3DV.2015.43

[7] A. Bhandari et Al, “Resolving multipath interference in Kinect: An inverse problem approach”, in *IEEE Sensors Journal*, 2015, vol.16, no 10, pp. 614-617, https://doi.org/10.1109/ICSENS.2014.6985073

[8] M. Kajitaniet et Al, “Point Cloud Streaming for 3D Avatar Communication” in *Stereo Vision*, 2008, IntechOpen, https://doi.org/10.5772/5902

[9] S. Izadiet 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* (UIST '11), 2011, ACM, New York, NY, USA, 559-568, https://doi.org/10.1145/2047196.2047270

[10] J. Tong et Al, “Scanning 3d full human bodies using kinects”, in *IEEE transactions on visualization and computer graphics*, 2012, vol. 18, no 4, pp. 643-650, https://doi.org/10.1109/TVCG.2012.56

[11] S. Li et Al, “3D Reconstruction by kinect sensor: a brief review”, in *Computer-Aided Drafting, Desigh and Manufacturing*, 2014, vol. 1, no 1, pp. 1-11,

[12] M. Zollhöferet Al, “State of the Art on 3D Reconstruction with RGB-D Cameras”, in *Computer graphics forum*, 2018, Vol. 37, No. 2, pp. 625-652, https://doi.org/10.1111/cgf.13386

[13] S. Orts-Escolano et Al, “Holoportation: Virtual 3d teleportation in real-time”, in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 2016, ACM, pp. 741-754, https://doi.org/10.1145/2984511.2984517

[14] Libfreebenect2, Release 0.2, https://doi.org/10.5281/zenodo.50641

[15] J. Besl et Al. "A 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, http://doi.org/10.1117/12.57955

[16] M. Rietzler et Al, "Fusionkit: A generic toolkit for skeleton marker and rigid-body tracking", in *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS, 2016, pp. 73-84, https://doi.org/10.1145/2933242.2933263