Time Coherent Animation of Dynamic 3D Objects from Kinect Camera using 3D Features

Naveed Ahmed
Department of Computer Science
University of Sharjah
Sharjah, United Arab Emirates
nahmed@sharjah.ac.ae

Salam Khalifa
Department of Computer Science
University of Sharjah
Sharjah, United Arab Emirates
salam.khalifa@gmail.com

Abstract— **We present a system for creating a time coherent animation of dynamic 3D objects from Kinect sensor using 3D features. We record a dynamic object using the Kinect sensor resulting in a sequence of Red, Green, Blue and Depth (RGB-D) frames of that object. We extract 3D features from the depth data and using these features, we estimate the motion of unrelated 3D geometry between two consecutive frames. Extending this motion compensation over the complete sequence, we manage to create a time coherent 3D animation of a dynamic 3D object, which can be used in a number of applications that require smooth temporal data for post capture analysis, e.g., action or object recognition, motion capture, 2D or 3D gesture recognition, motion editing, or non-interactive 3D animation.**

Keywords-Kinect; Dynamic Point Clouds; Point Sampling; 3D Animation

I. INTRODUCTION

Kinect sensor [13] has opened a completely new way of how we can visualize real world objects in the digital domain. It not only allows capturing the general appearance of a real-world object by means of a Red, Green, and Blue (RGB) camera, but also its 3D geometric shape using the depth camera. The latter feature allows it to be used in a number of applications that rely on capturing the true appearance of any object or gesture-based natural user interface. When used for capturing the dynamic 3D geometry, Kinect captures a depth image for each frame of the recorded video. The depth image provides the distance from the object to the camera. This depth image can be resampled into a 3D point cloud, in which every point has a specific location in the 3D space. If one has to visualize the dynamic 3D geometry of a real world object then this dynamic 3D point cloud representation is sufficient. Unfortunately, this representation is not suitable for a number of post processing tasks, e.g., motion analysis, motion compression, action recognition, etc. The reason being that each frame of the dynamic 3D point cloud is completely independent of the other, and there is no time coherence in the data. If this data has to be used in any of the aforementioned applications, then this time coherence has to be established, by either some form of mapping between each frame of the animation or estimating the motion of the dynamic object over the whole sequence.

In this paper, we present a new method for creating time coherent 3D animation from a sequence of depth images obtained from a Kinect camera. Our method is not confined to the depth image representation, rather we resample the depth images into dynamic 3D point clouds, and therefore our approach will work for any 3D animation in the form of 3D point clouds. We show that by means of 3D feature extraction from 3D point clouds, we can estimate motion of a dynamic object between two consecutive frames. Tracking the object over the whole sequence, results in a compact motion compensation representation of a time coherent 3D animation. As the result of our work, we do not need to store a dynamic object at each frame; rather the motion of the dynamic object is encoded at each subsequent frame. Thus, our method smoothly tracks one dynamic 3D object over the whole sequence that goes through the same animation that was captured in the non-coherent representation. The resulting time coherent representation can be employed in a number of applications for a post processing and post recording analysis.

A number of methods has been proposed to create 3D animations. Carranza et al. [7] presented a Free-viewpoint system for 3D animation that recorded a dynamic object using eight RGB cameras. The video data from eight cameras was used to capture the motion and shape of the actor. They used a template model to capture the approximate shape of the real-world actor. They did not use the multi-view data to reconstruct the dynamic 3D geometry. One of the drawbacks of this work was that it did not capture the surface reflectance of dynamic objects. Theobalt et al. [16] extended the work of Carranza et al. [7] and added dynamic surface reflectance estimation. Still, the method used a course template mesh, which was not the true geometry reconstructed from the video data. Different approaches were employed by Vlasic et al. [17] and Aguiar et al. [8] to capture realistic shape appearance and motion of dynamic objects. For the shape, both methods relied on high quality laser scans of the person. Both methods used high definition RGB cameras to capture the appearance of the actor. They differ in their approach on capturing the dynamic shape. Vlasic et al. [17] used a skeleton-based method to deform the template geometry. Their method works well for most of the objects, but had some limitations for the objects where the skeleton representation does not apply. On the other hand, Aguiar et al. [8] used a data driven approach to capture shape deformations. Their method is well suited to any kind of 3D geometry representation, as long as a high quality template model is available. Ahmed et al. [1] presented a work on reconstructing time coherent 3D animation without using the template geometry. They first created dynamic 3D visual hulls of the real world object, which were then tracked using a feature based dense matching algorithm. Their feature based matching does not incorporate any geometric features. They obtain Scale Invariant Feature Transform (SIFT) features from RGB images and match them over the animation while mapping them on the visual hulls. Their method is not suitable for 3D point cloud representation because it explicitly requires a smooth 3D surface to be available for calculating the geodesic distance between two points. Whereas, our method does not rely on any surface information because getting a smooth surface representation from noise Kinect data is a very challenging task.

In the past four or five years, an increasing number of methods for 3D geometry and animation reconstruction have started using depth cameras. It was initially made possible with the availability of the relatively low cost Time-of-Flight [11] depth cameras that can provide low-resolution dynamic geometry at high frame rate. A number of applications were proposed [3] [11] using the Time-of-Flight cameras. Microsoft completely changed the landscape of a general-purpose depth camera by bringing the extremely low cost Kinect for Xbox 360 as a general consumer electronics equipment. Kinect was a revolutionary device, because it could capture both color and depth data at 30 frames per second. The resolution of both cameras is really low (640x480) but because of its lower cost and deployment with Xbox 360, it was widely adopted. Apart of the gaming community, the research community also employed Kinect in a number of applications. A number of new methods were proposed in the areas of gesture recognition, motion capture, surface deformation, and motion editing.

Researchers have been employing depth cameras for reconstructing both dynamic and static real-world objects. One or more depth cameras were used by Kim et al. [11] and Castaneda et al. [6] for reconstructing a three-dimensional representation of static objects. Depth cameras are also used to reconstruct 3D shape, pose, and motion in the works presented by Berger et al. [4], Girshich et al. [9], Weiss et al. [18], and Baak et al. [3]. Multiple depth sensors are employed for capturing the dynamic scenes. Ahmed et al. [19] used six Kinect sensors to record a dynamic 3D object and create a 3D animation. Kim et al. [10] and Berger et al. [4] also used multiple depth sensors for object acquisition. Both of these methods do not establish any time coherence in the time varying data. On the other hand, Ahmed et al. [19] do reconstruct the time-coherent 3D animation but their work relies on RGB data for the feature points, whereas we show that one can reconstruct time coherent animation only using the geometric features.

Our work derives from the motivation of not using RGB data in the time coherent animation reconstruction. Even though RGB data has been successfully used in this line of research, it requires an additional mapping from depth data to RGB. In case of Kinect, this mapping is only one directional, i.e., from depth to RGB and that too is many to one. It means that multiple depth values can be mapped to a single RGB pixel. Thus, a feature point in RGB has an ambiguous representation in the three-space geometry. We therefore propose a framework that can work on the acquisition by one or more depth cameras and only utilizes the depth data for time coherent 3D animation reconstruction. We record a sequence using the Kinect camera and resample it in the form of a dynamic 3D point cloud representation. These point clouds are not time coherent and are completely independent of each other. In the following step, we extract a number of 3D features from each point cloud and match two consecutive frames using these features, starting from the first frame of the sequence. Using the mapping between the first two frames, we estimate the motion of the 3D point cloud between the two frames. This tracking is done over the whole sequence and we end up with a representation where for each frame we only need to store the motion with respect to the previous frame. Thus, our main contribution is a motion compensation representation by means of tracking using 3D features that creates a time coherent animation of a dynamic object.

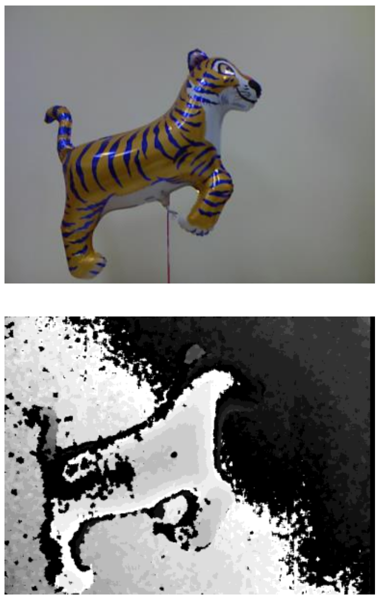


Figure. 1 One RGB and depth frame captured from Kinect.

II. VIDEO ACQUISITION

We acquire the dynamic 3D object using one Microsoft Kinect camera. Our method is not limited to a single camera setup, and can easily be extended to data from multiple cameras, as long as it is registered in a global coordinate space. An example of a multi-view setup can be seen in the work of Ahmed et al. [19].

The Kinect can capture two simultaneous video data streams, one RGB stream, and one depth stream. We use Microsoft Kinect Software Development Kit (SDK) to capture RGB-D data. For our method, we do not use the RGB stream but we capture to verify the acquisition process and make sure that the results are consistent with the depth stream. Kinect SDK can record both streams at different resolutions. At 30 frames per second, it can only record at 640x480 or a lower resolution. It can also capture at 1024x768 but the frame rate drops to 15 frames per second. Since we are interested in recording a dynamic object, the frame rate gets higher preference than the resolution. Therefore, we record both streams at 640x480 at 30 frames per second. The recording is stored in a high-speed memory buffer to avoid any input/output (IO) read/write overhead during the process. Once the recording is finished, each frame of the captured data is written to the disk.

The acquisition setup provides us with a frame-by-frame sequence of both RGB and depth data. One RGB and depth frame of the captured sequence can be seen in Fig. 1.

III. CALIBRATION AND BACKGROUND SUBTRACTION

Our acquisition system provides us with both RGB and depth streams. Each stream is comprised of a sequence of frames. For example, each frame of the depth stream is an intensity image of the resolution 640x480, where each pixel is associated to a depth value. There is no notion of how these depth values will be mapped to the three space for the visualization. Similarly, there is no relationship between the depth and RGB stream. For some methods that need both RGB and depth streams, a mapping has to be established between them.

For our work, we need two types of calibrations. First, we need to estimate the intrinsic parameters of the depth camera. Then, we need to find the mapping of the depth values provided by Kinect in a form of a two-dimensional depth image in the three-dimensional world coordinate space. Optionally, we also obtain the mapping between the RGB and depth stream to verify the correct acquisition of the data.

We use Matlab Camera Calibration toolkit [23] for the intrinsic calibration. We record a checkerboard from both color and infrared sensors to facilitate this calibration. We use the tool Kinect RGB Demo by Nicola Burrus [5] to convert depth data to real world three space distances, and find the mapping between RGB and depth streams. The depth camera calibration allows us to resample a depth image into a 3D point cloud and the RGB and depth stream mapping allows us to visualize the resampled point cloud with the color information to validate our acquisition setup. An example of the resampled 3D point cloud and the mapping of RGB to depth can be seen in Fig. 2.

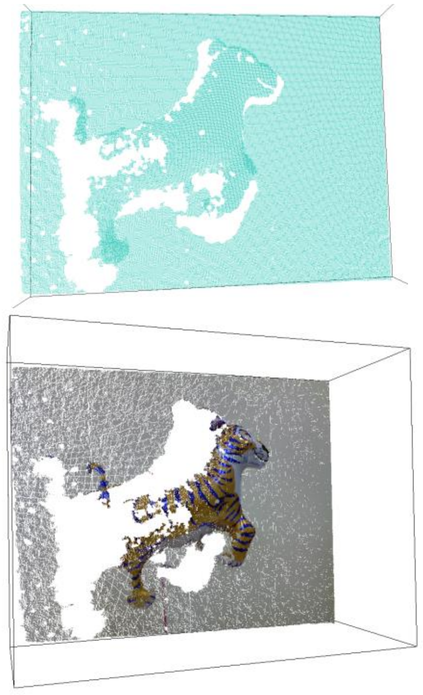


Figure. 2 A resampled 3D point cloud can be seen in the top image, while the same point cloud with RGB colors using the RGB and depth mapping can be seen in the bottom image.

We store 3D point clouds in the Point Cloud Data (PCD) file format using the Point Cloud Library [14]. The Point Cloud Library allows for efficient storage and manipulation of the point cloud data. It also has a number of algorithms implemented that can be used to analyze the point cloud data. We make an extensive use of this library in our work that will be explained in the next section.

After obtaining the resampled point cloud, we perform background subtraction to separate the dynamic object from the static background. In the first step, we record the same scene without the dynamic object. For the background subtraction, we record 30 frames of the background. Afterward, we average the 30 depth frames to average out the noise in the data. The mean background depth image is then subtracted from each depth frame of the recorded video sequence. This results in a separation of the dynamic model from the background and significantly reduces the storage cost for the point cloud. The depth data from Kinect is marred by very high temporal noise. This is a limitation of the technology and because of the high frame rate, it can be really pronounced when visualized. Therefore, using the

Point Cloud library we also de-noise the data by means of simple Gaussian filtering. A 3D point cloud after background subtraction and filtering can be seen in Fig. 3.

IV. Time Coherent Animation

So far, we have obtained a segmented 3D point cloud for each frame of the video sequence. These point clouds are completely independent of each other and there is no coherence from one frame to the other. This is the preliminary requirement of data representation for our method to create time coherent 3D animation. Our method is not limited to data obtained from Kinect. As long as dynamic 3D point cloud data is available, from either depth or RGB cameras, our method will work equally well. The only reason we are using Kinect is that we can obtain dynamic 3D point cloud representation from just one camera, whereas in a traditional RGB camera acquisition system, at least two cameras are required to reconstruct the depth information.

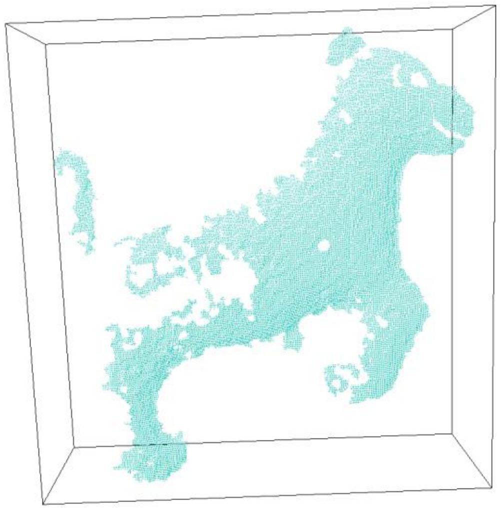


Figure. 3 Result from the background subtraction. The bounding box is significantly reduced compared to the original point cloud in Fig. 2.

To reconstruct time coherent animation we start by estimating a mapping between two consecutive frames of the dynamic scene sequence. We start by extracting 3D features from the first two frames t_0 and t_1. These features are then matched to find a sparse mapping between the two frames. This sparse matching is used to estimate the motion between the two frames. If the object undergoes a simple motion, e.g., translation, then only one match between the two frames is sufficient to track the point cloud from one frame to the next. Three or more matches can estimate a rigid body transform. On the other hand, if the motion is non-linear, which is true in our recordings then we need to find the motion of every point in the point cloud. We estimate the motion of all the points in the point cloud by using the sparse matching as the starting point. In the subsequent steps, we track t_0 over the whole sequence, resulting in a time coherent animation. Thus, our time coherent animation reconstruction algorithm takes the following form:

1) *Find 3D feature points at each frame*
2) *Match two consecutive frames starting from t_0 & t_1*
3) *Estimate motion of each point on t_0*
4) *Using the estimated motion at t0, track it to t_1*
5) *Loop from step 2 and track t_0 over the sequence*

In the first step, we find a number of 3D feature points for each frame of the 3D point cloud. We use the Point Cloud Library to estimate the following 3D features:

1) Estimate 3D SIFT over the depth image. The depth image is treated as an intensity image, and every feature point has a unique three-space location [20].
2) For every point on the point cloud, we estimate its underlying curvature and normal.
3) Using the normal information from step 2, estimate Clustered Viewpoint Feature Histogram (CVFH) descriptor [21].

These 3D features are then used to find a sparse correspondence between t_0 and t_1. 3D SIFT features are matched over the two depth images. It provides us with a one to one mapping for a sparse number of 3D positions. While matching 3D SIFT descriptors, we make use of curvature and normal to ensure that the matching is not an outlier. On the other hand, CVFH provides us with the matching clusters. Sparse matching approach is incorporated in earlier works, e.g., [1] and [22]. Our method is significantly different from those works, because it is incorporating 3D features. In order to find the one to one matching from the sparse correspondences, we make use of the approach from Salam et al. [22]. The one to one matching from 3D SIFT allows us to estimate the motion vector for the sparse matching points:

$$M_S = FP_1 - FP_0 \tag{1}$$

where $\boldsymbol{M_s}$ is the set of motion vectors for all 3D SIFT feature points. $\boldsymbol{FP_1}$ and $\boldsymbol{FP_0}$ are the feature points at frame 1 and 0 respectively. Similarly, for each cluster from CVFH we estimate its motion vector:

$$M_C = CP_1 - CP_0 \tag{2}$$

where $\boldsymbol{M_C}$ is the set of motion vectors for all clusters. $\boldsymbol{CP_1}$ and $\boldsymbol{CP_0}$ are the centroids of the clusters at frame 1 and 0 respectively. In the next step, we need to estimate the motion of all the points at t_0. For every point at t_0, we find the four nearest points in $\boldsymbol{FP_0}$ and the nearest cluster with respect to its centroid $\mathbf{CP_0}$. Each of these nearest points and cluster has an associated motion vector, i.e. $\mathbf{M_{S0}}$, $\mathbf{M_{S1}}$, $\mathbf{M_{S2}}$, $\mathbf{M_{S3}}$, and $\mathbf{M_{C0}}$. The motion vector for any point at t_0 is then defined as:

$$M_V = (M_{S0} + M_{S1} + M_{S2} + M_{S3} + M_{C0})/5 \tag{3}$$

where $\boldsymbol{M_v}$ is the average motion vector for the 3D point. Once this motion vector for each point is established, it is used to track t_0 to t_1. Thus, for the time step t_1 we do not need to store the complete point cloud, rather we can represent t_0 at t_1 in a motion compensation representation.

Using the estimated motion t_0 and t_1, we trivially track t_0 over the whole sequence. For example, in the next step the mapping between t_1 and t_2 is established but this mapping is used to find the motion vector of each point of tracked t_0. The same procedure then follows for all subsequent frames.

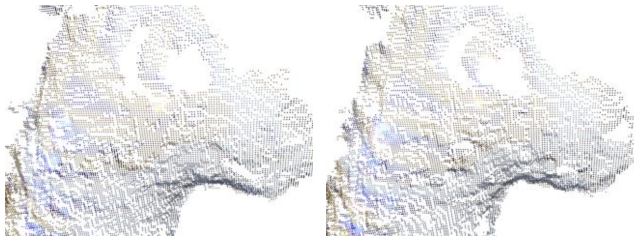
V. RESULTS

We use two types of data sets to validate our method. Both data sets are acquired through a single Kinect and each is 100 frames long. In the first sequence, we only have one object in the scene whereas in the second sequence there are two dynamic objects. Our method for creating time coherent animation managed to track both sequences completely. The result of the animation is a single point cloud tracked over the whole sequence.

Our method is very efficient in its implementation. On average we can track 10 frames each second, thus tracking a 100 frames animation takes less than 2 minutes on a Core i5 2.4 Ghz processor. Some results of our tracking method can be seen in Fig. 4.

Our method is subject to some limitations. One of the major limitations is the quality of the data. Depending on the speed of the motion, the number of 3D features can decrease, which will result in low quality of time coherent animation. Even using RGB images for detecting feature points will not solve this problem because fast motion introduces motion blur, which reduces the quality of the RGB data. This limitation can be rectified by using high frame-rate cameras. Other limitation is the choice of 3D features. We are limited to the types of 3D features because of our data representation. Most of the 3D features require a surface representation. In principal, one can generate the surface from a 3D point cloud. For the data from Kinect, it is a difficult problem because the depth data from Kinect has a very high temporal noise, which makes surface estimation a research problem in itself. In future, we would like to simulate a smoother point cloud and test surface reconstruction on it and evaluate the results from different types of 3D features.

Despite the limitations, we show that it is possible to create time coherent animation from dynamic 3D point clouds from Kinect using only the 3D features from the depth data.


(a)

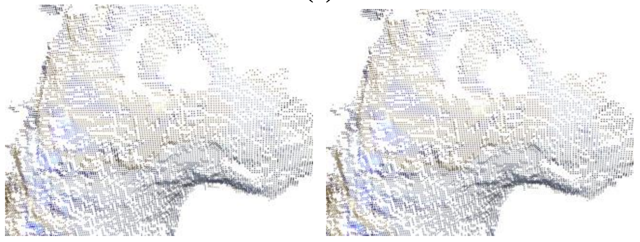

(b)

Figure. 4 Two non-coherent consecutive frames of 3D point cloud are shown in (a). Whereas (b) shows the same two frames generated using time coherent animation method. The frames are frame #0 and frame #80. It can be seen that in the non-coherent point clouds (a), the points change between the frames, esp. the effect is visible in the shape of the eye and around. The point cloud at frame #0 is tracked to frame #80 and does not show any changes in its shape by frame #80 (b).

VI. CONCLUSION

We presented a method to create time coherent animation from dynamic 3D point clouds using only 3D features from the depth data. We show that noisy data from a Kinect camera can be resampled to create a dynamic 3D point cloud representation of a dynamic object. After the internal calibration and background subtraction, we manage to isolate the dynamic object for creating a time coherent animation. Our time coherent animation reconstruction method is an iterative process, which uses 3D features from the point cloud to match two consecutive frames. The initial matching is propagated from first frame to the last resulting in a time coherent animation where a single 3D point cloud is tracked over the complete sequence. Our method is not restricted to the data obtained from Kinect. It can work for any animation as long as it is represented in the form of dynamic 3D point clouds. In future, we plan to extend our work to incorporate dynamic surface reconstruction and new 3D feature representations. The resulting time coherent animation from our method can be used in a number of applications, e.g., action or object recognition, gesture recognition, motion capture, analysis and compression.

ACKNOWLEDGMENT

This work was supported by the seed grant (110226) for new faculty members from the Department of Graduate Studies, University of Sharjah, UAE.

REFERENCES

[1] N. Ahmed, C. Theobalt, C. Rossl, S. Thrun, and H. P. Seidel, "Dense Correspondence Finding for Parametrization-free Animation Reconstruction from Video," Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 08), IEEE Press, June 2008, pp. 1-8, doi:10.1109/CVPR.2008.4587758.

[2] N. Ahmed, "A System for 360 degree Acquisition and 3D Animation Reconstruction using Multiple RGB-D Cameras," Proc. Computer Animation and Social Agents (CASA 2012), May 2012, pp. 1-4.

[3] A. Baak, M. Muller, G. Bharaj, H. P. Seidel, and C. Theobalt, "A Data-driven Approach for Real-time Full Bodypose Reconstruction from a Depth Camera," Proc. IEEE International Conference on Computer Vision (ICCV 2011), November 2011, pp. 1092-1099, doi:10.1109/ICCV.2011.6126356.

[4] K. Berger, K. Ruhl, Y. Schroeder, C. Bruemmer, A. Scholz, and M. A. Magnor, "Markerless Motion Capture Using Multiple Color-depth Sensors," Proc. 16th International Workshop on Vision Modeling and Visualization (VMV 2011), October 2011, pp. 317–324.

[5] N. Burrus, "Kinect RGB Demo," http://www.computervisiononline.com/software/kinect-rgbdemo, last access 07/03/2015.

[6] V. Castaneda, D. Mateus, and N. Navab, "Stereo Time-of-Flight," Proc. IEEE International Conference on Computer Vision (ICCV 2011), November 2011, pp. 1684-1691, doi:10.1109/ICCV.2011.6126431.

[7] J. Carranza, C. Theobalt, M. Magnor, and H. P. Seidel, "Free-viewpoint Video of Human Actors," Proc. ACM Siggraph, August 2003, pp. 569-577, doi:10.1145/882262.882309.

[8] E. D. Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H. P. Seidel, and S. Thrun, "Performance Capture from Sparse Multi-view Video," Proc. ACM Siggraph, August 2008, pp. 15-23, doi:10.1145/1399504.1360697.

[9] R. Girshick, J. Shotton, P. Kohli, and A. C. Fitzgibbon, "Efficient Regression of General-activity Human Poses from Depth Images," Proc. IEEE International Conference on Computer Vision (ICCV 2011), November 2011, pp. 415-422, doi:10.1109/ICCV.2011.6126270.

[10] Y. M. Kim, D. Chan, C. Theobalt, and S. Thrun, "Design and Calibration of a Multi-view ToF Sensor Fusion System," Proc. IEEE Computer Vision and Pattern Recognition Workshop on Time-of-Flight, June 2008, pp. 55-62.

[11] Y. M. Kim, C. Theobalt, J. Diebel, J. Kosecka, B. Micusik, and S. Thrun, "Multi-view Image and ToF sensor Fusion for Dense 3D Reconstruction," Proc. IEEE Workshop on 3-D Digital Imaging and Modeling (3DIM09), September 2009, pp. 1542-1549, doi:10.1109/ICCVW.2009.5457430.

[12] D. G. Lowe, "Object Recognition from Local Scale Invariant Features," Proc. IEEE International Conference on Computer Vision (ICCV), 1999, pp. 1150–1157, doi:10.1109/ICCV.1999.790410.

[13] Microsoft, "Kinect for microsoft windows and xbox 360," http://www.kinectforwindows.org/, last access 07/03/2015.

[14] B. R. Radu, and C. Steve, "3D is here: Point Cloud Library (PCL)," IEEE International Conference on Robotics and Automation (ICRA), May 2011, pp. 1-4, doi:10.1109/ICRA.2011.5980567.

[15] A Tevs, A. Berner, M. Wand, I. Ihrke, and H. P. Seidel, "Intrinsic Shape Matching by Planned Landmark Sampling," Proc. 32nd International Conference of the European Association for Computer Graphics (Eurographics), April 2011, pp. 543-552.

[16] C. Theobalt, N. Ahmed, G. Ziegler, and H. P. Seidel, "High-Quality Reconstruction of Virtual Actors from Multi-view Video Streams," IEEE Signal Processing Magazine, vol 24, 2007, pp. 45-57.

[17] D. Vlasic, I. Baran, W. Matusik, and J. Popovic, "Articulated Mesh Animation from Multi-view Silhouettes," Proc. ACM Siggraph, August 2008, pp. 24-31, doi: 10.1145/1399504.1360696.

[18] A. Weiss, D. Hirshberg, and M. J. Black, "Home 3D Body Scans from Noisy Image and Range Data," Proc. IEEE Internation Conference on Computer Vision (ICCV), November 2011, pp. 1951-1958, doi:10.1109/ICCV.2011.6126465.

[19] N. Ahmed, and I. Junejo, "A System for 3D Video Acquisition and Spatio-Temporally Coherent 3D Animation Reconstruction using Multiple RGB-D Cameras," International Journal of Signal Processing, Image Processing and Pattern Recognition, vol. 6, 2013, pp. 113-129.

[20] D. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, vol 60, 2004, pp. 91-110.

[21] A. Aldoma, N. Blodow, D. Gossow, S. Gedikli, R.B. Rusu, M. Vincze and G. Bradski, "CAD-Model Recognition and 6 DOF Pose Estimation," Proc. IEEE International Conference on Computer Vision Worskhops, 2011, pp. 585-592, doi:10.1109/ICCVW.2011.6130296.

[22] S. Khalifa, and N. Ahmed, "Temporally Coherent 3D Animation Reconstruction from RGB-D Video Data", Proc. International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing (ICCGCVIP), 2014, pp. 1383-1389.

[23] J. Y Bouguet, "Camera Calibration Toolbox for Matlab," http://www.vision.caltech.edu/bouguetj/calib_doc/, last access 07/03/2015.