Environment Capturing with Microsoft Kinect

Kevin Mackay[1], Hubert P. H. Shum[2], and Taku Komura[1]

[1]The University of Edinburgh, UK
[2]The Northumbria University, UK

Constructing virtual scenes that incorporate human and object interaction has traditionally been a time consuming process in computer animation, whereby the motion of an actor is first recorded and any objects used in the scene are then intricately added by an animator. The Microsoft Kinect utilizes a synchronized RGBD stream to provide markerless skeletal tracking of humans, enabling efficient motion capture; however, the problem of capturing environment objects remains unsolved. In this paper, we propose a new framework to segment and track three major types of environment objects using Kinect, namely background planes, stationary objects and dynamic objects. We demonstrate that the motion of an actor and their surrounding environment can be obtained at the same time, saving considerable effort for the animators. Our proposed system is best to be applied to applications involving extensive human-object interactions, such as console games and animation designs.

Index Terms — **Computer vision and image processing, human computer interaction, graphical modeling and simulations**

I. Introduction

Traditional motion tracking devices such as the optical motion capture system require complex user setup and heavy data post-processing, making it unsuitable for real-time applications. Recently, Microsoft introduced an integrated motion sensing device called Kinect that enables markerless motion tracking, which provides new opportunities for human computer interaction research. These projects range from using skeletal gesture as a replacement to keyboard and mouse input [13] to computer graphics application such as recognizing and creating facial animations [16].

A major problem with the Kinect, as well as most of the Kinect related research, is that tracking is only performed for the human user. When creating an animation, the environment objects are equally important to define the context of the scene. Without the environment being captured, animators have to manually place environment objects into the scene, including the background and any objects the character is interacting with, which entail a large amount of post capture work.

In this paper, we propose to utilize the depth and color cameras in Kinect to capture environment objects. We observed that environment object can usually be classified into three groups. (1) Background planes in the capture area such as walls, doors, and floor. (2) Stationary objects that remain unmoved throughout a capture session, such as tables and chairs. (3) Dynamic objects that move and are interacted with by the human user, such as a football and a cup. It is difficult, if not impossible, to design a single segmentation algorithm that works well for the three types of objects, due to their different features and behaviors. Therefore, we propose a framework to segment and track each type of object using specifically designed algorithms, and combined the results to create the final scene.

Combined with the Kinect's built-in motion tracking capability [18], our system can capture and animate the whole scene including the character and environment in real-time. As a result, our proposed system can save considerable time for the animator, removing the need for intricate object placement into the captured scene. Furthermore, we can improve the realism of the final animation, as the structure of the scene and the movements of the objects are based on real data. This is beneficial to both the film and video game industries, as it will provide a new system for near-immediate prototyping of environments and the actor's interaction with that environment.

Our system is most beneficial for capturing animations involving heavy human-object interaction. The captured scene can be employed as a prototype of desired animation, or guidance for the animator during post-processing of the animation.

II. Related Works

In this section, we review related work from the computer animation and vision fields, focusing on developments in object interaction and markerless tracking.

A. Computer Animations

Synthesizing human movement for a virtual character that realistically interacts with an environment is an ongoing problem. Koga et al. [7] demonstrated that motion planning techniques could be applied to computer animation, animating a virtual character's arms to reach for and grasp an object without collision. Kallmann et al. [6] took this work further by using probabilistic roadmaps for the automatic synthesis of multiple limbs at a time. More recently, Shapiro et al. [12] showed that they can edit captured full-body motions such that the virtual character can adapt to complex dynamic environments. They create motion plans to avoid undesired collision, while allowing interaction with some objects. The drawback of this approach is that it is heavily reliant on inverse kinematics, which can only be applied to linked-limbs but not the entire body without degrading the quality of the movements. These papers focused on motion planning through editing an already existing capture or synthesizing reaching and grasping. We believed that by capturing both human

Manuscript received May 31, 2012. Corresponding author: Hubert P. H. Shum (e-mail: hubert.shum@northumbria.ac.uk).
Digital Object Identifier inserted by IEEE

motion and environment objects at the same time, human movement should not require editing. Any objects required for the scene should be captured by the depth camera and inserted into to the final scene with a polygonal representation.

For passive object interaction, a different approach has been presented by Abe and Popović [1], who have created a control algorithm that considers the external constraints, the character and object models to produce a physical simulation of the desired animation. Example animations include the virtual character holding a chain and picking up a box. The major limitation of this approach is the limited motions the character could perform. For example, the character cannot jump, and this would result in a loss of control on certain degrees of freedom. Using physical simulation for animation has been taken further by Jain and Liu [5]. They implemented a system, based on dynamic equations, that allows the animator to edit the trajectory of a bouncing ball and calculate the physics for any corresponding contact with the virtual character. The character is animated by marked motion capture and can also be edited to change the point of contact with the passive object. A drawback of this approach is that there is no feedback of the impact of the collision between character and object. If a ball hit a character, it would simply bounce off the character, which remains unperturbed. Rather than replicating the physics and manually building the object interaction, our algorithm applies visual tracking to capture the motion of objects. Furthermore, since the actor and the objects interact in the real world, the generated animation by definition is realistic.

B. *Computer Visions*

Here, we review some of the recent research regarding markerless motion capture, particularly those which are designed for computer animation.

Pekelny and Gotsman [9] have shown that it is possible to reconstruct a complete 3D mesh of an articulated object, such as a humanoid monster or toy truck using only a single depth camera. They can then estimate a rigid-body skeleton for that object, without a need for an initial template, subsequently allowing new motions to be generated for the reconstructed mesh and generated skeleton.

Increasing the number of cameras will increase the data available to work with and subsequently improve the results. Gall et al. [7] have shown that by using a multi-view video sequence, they can perform accurate skeleton and surface estimation through a series of interleaved captures. Therefore, they are able to capture both the rigid transformations of the skeleton and the non-rigid transformations of human hair and clothing.

Taylor and Cowley [14] present a plane segmentation and analysis technique. They use an RGB image to suggest an interpretation for a corresponding depth image, and therefore improve the resulting plane segmentation. This paper demonstrates the benefits of synchronized RGBD data for scene analysis, especially when dealing with unknown and complex environmental layouts.

C. *Contributions*

The major contribution of this paper is to propose a new system to capture environmental objects, along with the interaction between a user and the environment, using Microsoft Kinect. We apply tailored algorithms to segment the three major types of objects in a scene, namely background plane, stationary objects and dynamic objects. With accurate and efficient object segmentation algorithms, it is possible to create animations of human-environment interactions with minimal effort.

III. System Overview

The overview of our system is shown in Fig. 1. Based on the depth image, we create a point cloud in the 3D space (Sec. IV-A), from which we segment the background plane (Sec. IV-B) and the stationary objects (Sec. IV-C and Sec. IV-D) separately. Using the color image, we segment the dynamic objects (Sec. IV-E and Sec. IV-F). All the segmented results are combined to form a final animation.

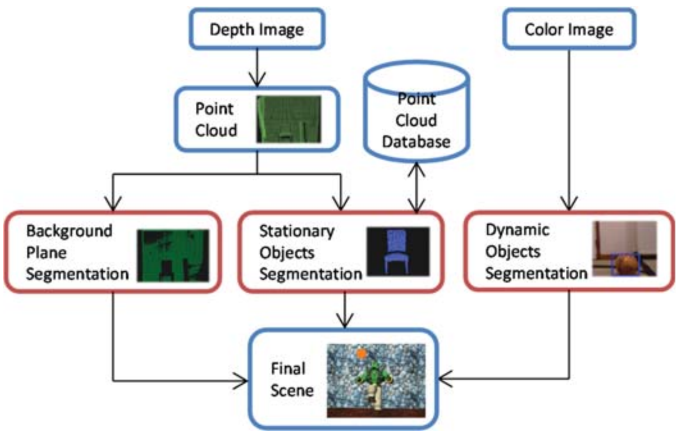
Depth Image
Color Image
Point Cloud
Point Cloud Database
Background Plane Segmentation
Stationary Objects Segmentation
Dynamic Objects Segmentation
Final Scene

Fig. 1: Overview of the proposed system. Based on the features of the three different types of objects, we designed different segmentation algorithm to obtain the most accurate and computational efficient results.

IV. Methodology

In this section, we explain how we construct the point cloud from the depth image and how we apply different algorithms to segment three classes of objects, including planer background, stationary objects and dynamic objects.

A. *Point Cloud Creation with Depth Image*

Here, we describe the process to create point cloud from Kinect inputs. Each pixel in the Kinect depth image corresponds to a point in the 3D point cloud. In other words, for each frame, we can create a point cloud of the scene by mapping the pixels in the depth image on to a 3D virtual world.

In Fig. 2, the real scene and the corresponding point cloud is shown. Notice that the floor is not particularly well captured with relatively large areas missing. This is because of the depth measurements algorithm in Kinect that is derived from triangulation via structured light rather than time of flight. Thus, capture results are poor for horizontal surfaces, especially when they are far from the camera.

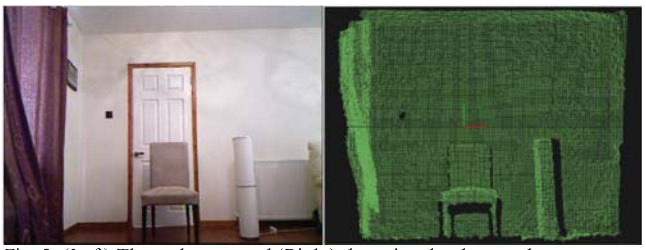
Fig. 2: (Left) The real scene and (Right) the point cloud created.

The cloud is then down sampled using a voxel grid, which approximates the centroid of all points contained within a 1cm^3 voxel. This provides a much cleaner resultant cloud and a more consistent distance between the individual points.

B. *Background Planes Segmentation with RANSAC*

Here, we explain the process to identify and segment background planes in the point cloud with Random Sample Consensus (RANSAC). We define background planes as large, flat planes lying at the far end of the capture area, such as the walls in a room. Non-planer objects such as the curtain on the wall are considered as stationary objects and will be considered in the next section.

We apply RANSAC [3] as it can robustly estimate the parameters for any planar regions within the point cloud. The algorithm iteratively estimates the inlier points that fit a planar model, or otherwise classifies the points as outliers. Upon successfully fitting a plane to the cloud, the plane's coefficients including the normal and the distance from the camera are computed.

Among the detected planes from RANSAC, we obtain the background as the planes with area larger than a predefined threshold. Fig. 3 (left) shows a successfully segmented background plane. Notice that the segmented point cloud may contain holes due to the occlusion of objects. The position and orientation of the plane is stored for future rendering.

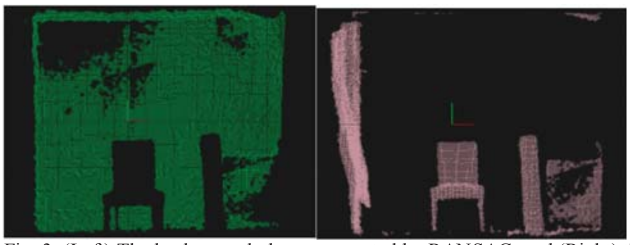
Fig. 3: (Left) The background plane segmented by RANSAC, and (Right) the remaining point cloud

It is possible to detect the floor plane using RANSAC. However, it requires carefully tuned parameters for the number of iterations and distance threshold, due to the poor capture quality.

C. *Stationary Objects Segmentation with Euclidean Cluster Extraction*

Here, we explain the process to segment stationary objects in the scene using Euclidean Cluster Extraction [10]. We define stationary objects as the objects with constant position and orientation throughout one capture session.

By removing the point cloud of the background plane, we can obtain the foreground, which consists of multiple stationary objects (Fig. 3 right). The Euclidean Cluster Extraction algorithm is applied to segment the point cloud into individual objects. The algorithm iteratively searches for connecting points per cluster through nearest neighbors. A spherical radius is used to decide if a point belongs to a cluster, which is set to 3cm in our experiments. The position of the object is then represented by its 3D centroid.

At this point, an animator can optionally associate objects in the point cloud with a polygon mesh for rendering in the final scene. This also allows the animator to combine multiple objects into a single logical object for better representation of the 3D centroid. This may be necessary when single objects are segmented into separate parts due to gaps in the point cloud. For example Fig 4 shows the chairs seat and back requiring manual combination after being separated during object segmentation.

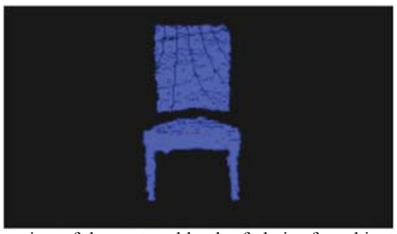
Fig. 4: Concatenation of the seat and back of chair after object segmentation

While it is possible to use the segmented result directly in the final scene, we prefer to store the point cloud in a database, such that if we see the same object in future capture sessions, we can automatically identify the orientation of the object. This process is explained in the next section.

D. *Stationary Object Transformation by Registration*

It is common for different capture sessions to have the same stationary objects in different positions. It would be time consuming to manually identify the objects in every capture session. Instead, we make use of the point cloud database to identify the rigid transformation of different stationary objects in the scene.

We apply the registration algorithm [2] [10] to calculate the rigid transformations of stationary objects. The point cloud stored in the database such as the chair in Fig. 4 is called a template cloud. The registration algorithm aligns the template cloud with the cloud in a new capture session, such that we can identify the chair and obtain the rigid transformation automatically. It involves the following three steps:

Estimating Surface Normal: We estimate the surface normal of a point cloud by the normal of a plane tangent to the surface of nearby points. The plane is obtained by fitting the nearest neighbors with least-square method.

Point Features Histograms: Point Feature Histograms (PFH) make use of the estimated surface normal to produce further discriminative geometric information of a given point cloud. It attempts to model surface variations among a neighborhood of points in a cloud using the directions of the given surface normal. The outcome is a histogram of relationships between all pairs of points within a neighborhood.

Initial Alignment Algorithm: The initial alignment algorithm solves the problem of finding a match between two point clouds by producing a number of possible correspondences to match the input cloud against a template cloud. The distance between the PFHs for the clouds is first computed. The cloud with the lowest distance is then selected and Iterative Closest Point is used to evaluate the rigid transform between clouds.

Fig. 5 upper left shows the template cloud of the chair in the database and the point cloud in the current scene. Fig. 5 upper right is an estimated transformation that places the point cloud of the chair on top the template cloud. The computed rigid transformation has been applied to the chair object to create the scene in Fig. 5 lower. In Fig. 6, a more extreme example is shown, requiring the transformation of a large vertical box point cloud to a horizontal box template cloud.

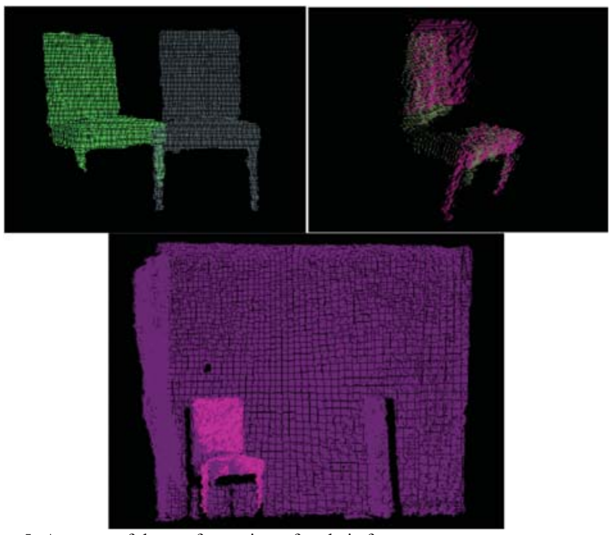
Fig. 5: A successful transformation of a chair for a new scene.

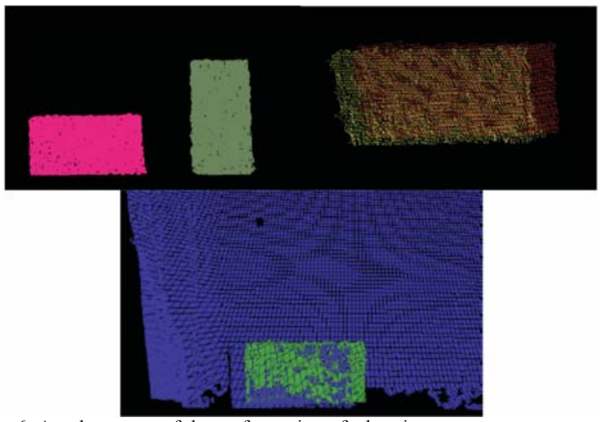
Fig. 6: Another successful transformation of a box in a new scene.

With the point cloud database and the registration algorithm, it becomes very efficient to setup a new scene in which stationary objects are placed in different positions.

E. Dynamic Object Segmentation with Background Subtraction

Although registration can evaluate the transformation of objects, it is too computational costly to be performed in run-time. Thus, we only apply registration to initialize stationary objects in a scene, and apply simpler background subtraction to track dynamic objects in run-time. Here, we define dynamic objects as environment objects that move during a capture session.

Since the camera is assumed to be steady and the dynamic object is moving, background subtraction provides the most robust and computationally efficient method to segment the objects. Given two successive frames from the color image, we calculate the absolute color difference for each pixel (Fig. 7 upper left) and threshold the pixel difference to generate a binary image (Fig. 7 upper right). After applying an erosion filter (Fig. 7 lower left), we use axis aligned bounding box to segment the dynamic object (Fig. 7 lower right).

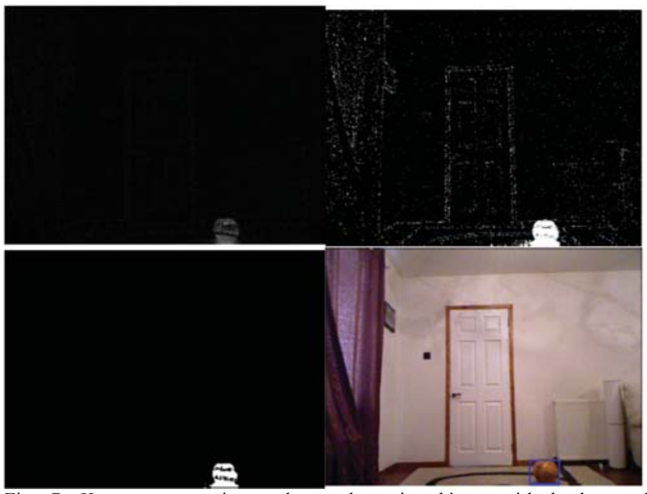
Fig. 7: Key processes in to detect dynamic objects with background subtraction. (Upper left) Difference image generated by background subtraction (Upper right) Binary image generated by thresholding (Lower left) Applying erosion filter (Lower right) Object segmented with axis aligned bounding box

Segmentation by background subtraction is accurate and fast, but can only be applied to moving objects. Thus it can only be used in dynamic objects but not stationary objects.

F. Dynamic Object Tracking with Mean Shift Tracking

Here, we explain the process to track a dynamic object across frames.

The mean shift algorithm is used to track dynamic objects in the scene [4]. The algorithm iteratively looks for the local maxima of a probability map on a frame-to-frame basis, starting from the position provided by the detected object's bounding box. A new centroid can then be found in the surrounding area of the object in the previous frame. It provides the position of the object and subsequently a new bounding box in the current frame. Mean shift terminates after a predefined number of iterations or when the local maxima found reach a threshold.

HSV color space is used as it allows the representation of color through a single channel, namely the hue (Fig. 8 upper left). The probability map is produced by back projection for each frame. It calculates possible regions of an image that match a given histogram (Fig. 8 upper right). We exclude any pixel with saturation smaller than a predefined threshold as colors in such a range become non-representative (Fig. 8 lower left). Finally, we can track the centroid of the object in the current frame (Fig. 8 lower right). We then obtain the corresponding location in the depth image, and evaluate the 3D position of the object.

Due to the household environment, parts of the door frame are clearly of a similar color to the ball as shown in the example of Fig. 8. Still, the algorithm is robust enough to deal with this situation.

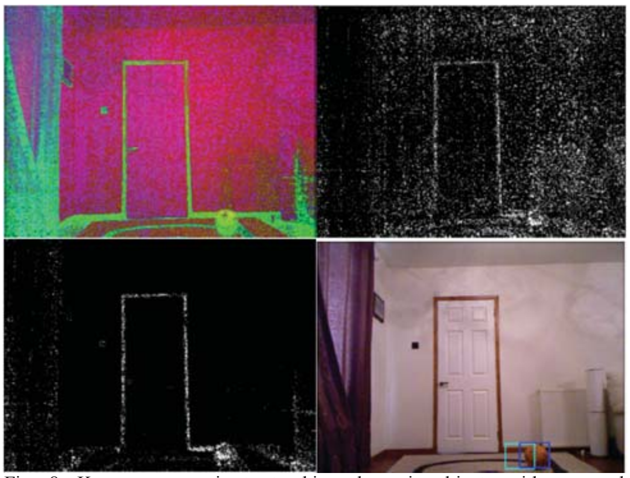
Fig. 8: Key processes in to tracking dynamic objects with mean shift tracking. (Upper left) HSV representation of the color image (Upper right) Probability map (Lower left) Thresholding saturation value (Lower right) Object tracked in the current frame

V. Experimental Results

In this section, we conduct different experiments to test the robustness and accuracy of our system.

During implementation, we use the OpenNI [8] to obtain depth and color image from Kinect. We then use the Point Cloud Library [11] to segment background plane and stationary objects, and use OpenCV [17] to segment dynamic objects. Finally, we use the Ogre3D to render the resultant scene.

A. Background Plan Segmentation

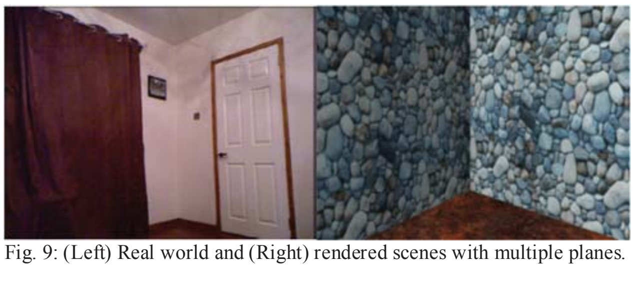
Fig. 9: (Left) Real world and (Right) rendered scenes with multiple planes.

We obtain background plane coefficients produced by RANSAC, and render the plane with textured polygons. In Fig. 9, Kinect has been positioned to capture multiple planes. After processing, two planes representing the walls have been detected by the plane segmentation algorithm. As shown in the figure, the rendered scene is an accurate representation of the real world scene, with the walls at the correct angle.

B. Stationary Objects Segmentation

In this experiment, we use a point cloud database that consists of a chair and a lamp. We place the stationary objects in the scene and apply registration to calculate their orientation and location. Finally, we render the corresponding polygon mesh as shown in Fig. 10.

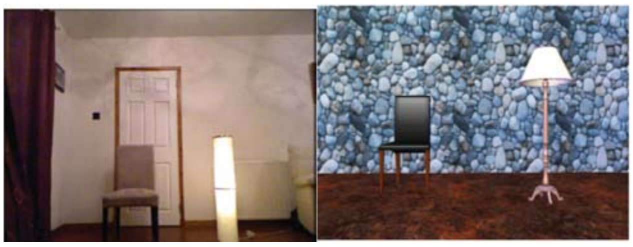
Fig. 10: (Left) Real world and (Right) rendered scenes of chair, light and rear wall.

Another experiment is conducted as shown in Fig. 11. The chair has been moved to the left and rotated, and an actor has also been introduced to scene. Registration is used again to initialize the orientation and position of the stationary objects. We use Kinect's internal human tracking functions to track the human user [18]. Several human-object interaction have been captured, such as tying a shoelace and resting an arm on the chair.

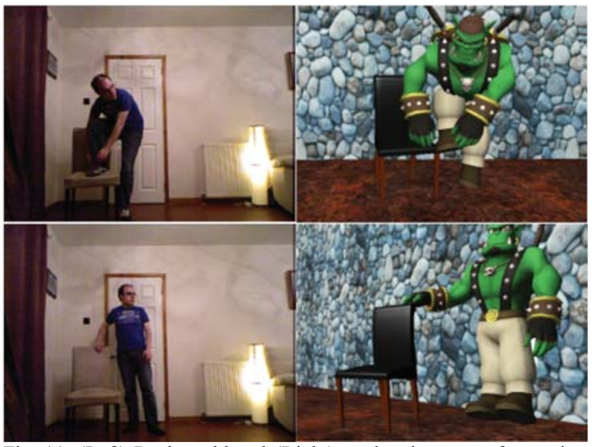
Fig. 11: (Left) Real world and (Right) rendered scenes of actor interacting with chair.

C. Dynamic Objects Segmentation

Here, an experiment is conducted to evaluate the performance on tracking dynamic objects with the present of a user. The dynamic object in this case is a ball as shown in Fig. 12. The results of the real-time dynamic object and human tracking are reasonably accurate. Furthermore, upon dropping the ball from a height, the ball is successfully tracked in its descent to the ground including the subsequent bounces. This creates realistic animation of a falling ball without any implementation of physics.

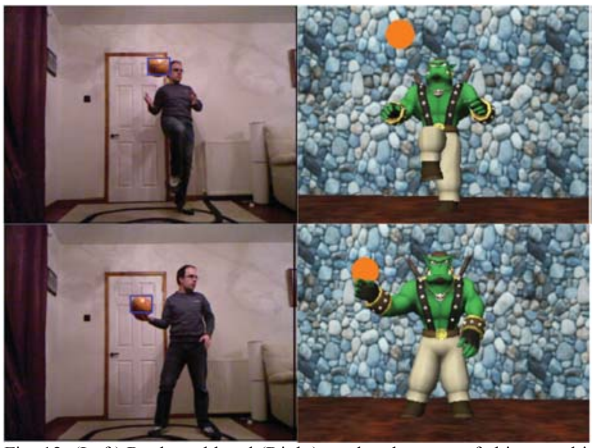
Fig. 12: (Left) Real world and (Right) rendered scenes of object tracking.

D. Performance

With skeletal and object tracking both active, the renderer maintains an interactive frame rate of 15-25 FPS on a modest laptop with the following hardware specification: Intel Core i3-370M @ 2.4GHz for the CPU, 4GB RAM and an NVidia GeForce G310M graphic card. Overall, the creation of a scene with stationary objects takes no more than 10 minutes of the animator's time from initial capture to render.

VI. Conclusion and Discussions

In this paper, we presented a new algorithm to capture human-environment interaction with the markerless Kinect system. We proposed tailored algorithms for segmenting and tracking three classes of objects, including the background planes, the stationary objects and the dynamic objects. We apply Kinect's built-in human tracker to capture the human motion. Experimental results showed a satisfactory quality of captures in real-time.

Synthesizing the object motion in Jain and Liu's work [5] requires input from the animator, who must decide how object's physics should be applied to the scene. It takes around 30-40 minutes for the animator to produce a scene, as oppose to 10 minutes in our system.

While the system produced is capable of producing scenes quickly, there is usually a jittering effect of dynamic objects, which degrade the quality of the animation. One way of improving the jittering is to apply a smoothing filter to the position determined from the tracking algorithm.

The quality of registration is limited due to the use of 2.5D point clouds rather than 3D (i.e. the point cloud suffers from occlusion of objects). One way of improvement is to employ multiple Kinects in set positions to capture different angles of the stationary object in order to generate a full 3D point cloud.

As a future direction, we are interested in investigating the contextual relationship between the human user and environmental objects. By understanding the contextual meaning of the human action, it is possible to apply the system to serious applications such as security monitors and sport training.

Acknowledgement

We thank the anonymous reviewers for their constructive comments. We also thank Mr. Chris Blythe for his help in revising the paper.

References

[1] Abe, Y. and Popović, J. (2006). "Interactive Animation of Dynamic Manipulation," *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation, SCA '06*, pages 195–204, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[2] Dixon, M. (2011). "Aligning Object Templates to a Point Cloud," http://pointclouds.org/documentation/tutorials/template_alignment.php#template-alignment.

[3] Fischler, M. A. and Bolles, R. C. (1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Commun. ACM*, 24:381–395.

[4] Fukunaga, K. and Hostetler, L. (1975). "The Estimation of the Gradient of a Density Function, with Applications in Pattern Recognition," *IEEE Transactions on Information Theory*, 21(1):32–40.

[5] Jain, S. and Liu, C. K. (2009). "Interactive Synthesis of Human-Object Interaction," *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 47–53, New York, NY, USA. ACM.

[6] Kallmann, M., Aubel, A., Abaci, T., and Thalmann, D. (2003). "Planning Collision-Free Reaching Motions for Interactive Object Manipulation and Grasping," *Computer graphics Forum (Proceedings of Eurographics'03)*, 22(3):313–322.

[7] Koga, Y., Kondo, K., Kuffner, J., and claude Latombe, J. (1994). "Planning Motions with Intentions."

[8] OpenNI (2010). "OpenNI / SampleAppSinbad," https://github.com/OpenNI/SampleAppSinbad.

[9] Pekelny, Y. and Gotsman, C. (2008). "Articulated Object Reconstruction and Markerless Motion Capture from Depth Video," *Computer Graphics Forum*, 27(2):399–408.

[10] Rusu, R. B. (2009). "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," PhD thesis, Computer Science Department, Technische Universitat Munchen, Germany.

[11] Rusu, R. B. and Cousins, S. (2011). "3D is here: Point Cloud Library (PCL)," *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

[12] Shapiro, A., Kallmann, M., and Faloutsos, P. (2007). "Interactive Motion Correction and Object Manipulation," *ACM SIGGRAPH Symposium on Interactive 3D graphics and Games (I3D'07)*, Seattle.

[13] Suma, E. A., Lange, B., Rizzo, S., Krum, D., and Bolas, M. (2010). "Flexible Action and Articulated Skeleton – FAAST," http://projects.ict.usc.edu/mxr/faast/.

[14] Taylor, C. and Cowley, A. (2011). "Segmentation and Analysis of RGB-D data," *RGB-D: Advanced Reasoning with Depth Cameras 2011*, Seattle, WA, USA.

[16] Weise, T., Bouaziz, S., Li, H., and Pauly, M. (2011). "Realtime Performance-Based Facial Animation," *ACM Transactions on Graphics (Proceedings SIGGRAPH2011)*, 30(4).

[17] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2008

[18] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. (2011) "Real-Time Human Pose Recognition in Parts from Single Depth Images," *IEEE Conference on Computer Vision and Pattern Recognition*, June 2011.