Physically-based Interaction for Tabletop Augmented Reality Using a Depth-sensing Camera for Environment Mapping

Thammathip Piumsomboon
The HIT Lab NZ
University of Canterbury,
Christchurch, New Zealand
thammathip.piumsomboon@pg.canterbury.ac.nz

Adrian Clark
The HIT Lab NZ
University of Canterbury,
Christchurch, New Zealand
adrian.clark@hitlabnz.org

Mark Billinghurst
The HIT Lab NZ
University of Canterbury,
Christchurch, New Zealand
mark.billinghurst@hitlabnz.org

***Abstract*— In this paper, we present our AR framework which uses information obtained from a Microsoft Kinect camera to create AR experiences with physically-based interaction and environment awareness. A tabletop racing game and block manipulating application are created based on this framework. Our framework comprises of five steps in the process: (1) marker tracking (2) depth acquisition (3) image processing (4) physics simulation and (5) rendering. The resulting augmented environment supports occlusion, shadows, and physically-based interaction of real and virtual objects.**

Keywords: augmented reality; physically-based interaction; Kinect; physics engine; environment awareness

I. INTRODUCTION

Augmented Reality (AR) combines the task and communication spaces together by overlaying virtual information into the real environment. In this environment multiple users can share and collaborate on virtual tasks while carrying out natural face-to-face communication through gestures, speech and gaze. This overlap of task and communication spaces makes AR an ideal user interface to support collaborative works.

In AR, users are free to interact with real and virtual objects in the task space. Manipulation of virtual objects in three-dimensional (3D) space in AR is commonly achieved through Tangible AR interfaces [1], which utilize the users' knowledge and skills in interaction with the physical world. Additionally, users' hands are not only used for interacting with objects, but also to form gestures that the system can recognize and process as inputs as a mean of natural interaction.

Despite the benefits of Tangible AR, a user study has shown that the use of physical input devices encourage actions that users normally apply on physical objects which may not be supported by the system [2]. This shortcoming can be illustrated with the following example. A paddle, consisting of a fiducial marker with a handle, can be used for repositioning the virtual objects in AR. Once the object is positioned on the paddle, users would expect the object to slide off the paddle due to gravity when it is tilted as it would in the real world. However, this is not possible for a system without a physical simulation.

To improve the AR experience, virtual content should behave realistically in the physical environment it is placed in. Furthermore, fiducial marker and natural feature registration algorithms are able to calculate the pose of a given target, but have no awareness about the environment the target exists in. This lack of awareness can cause the virtual content to float above objects or appear inside them, or occlude objects it should appear behind, breaking the illusion that the virtual content exists in the real world and consequently hindering user experience.

In this paper, we present an AR framework that provides physically-based interaction and environment awareness with the aim to create better AR experiences. The Microsoft Kinect [3], a low cost consumer device that provides both RGB and depth-sensing cameras, has been integrated into our system. The Kinect is used to examine a 3D task space above a tabletop, and with the transformation between the Kinect and the AR viewing camera known, virtual content can be realistically composited in the environment. The user can interact with the virtual content in a natural and intuitive way through physical objects and natural gestures.

The main contributions of our work are:

- Development of an AR framework that supports environment awareness and physically-based interaction through integration of the Microsoft Kinect.
- Development of two AR applications to demonstrate the support for environment awareness and physically-based interaction.

The organization of the remaining sections is as follows. The related works is covered in Section II. Section III explains our AR framework. The performance of our framework is discussed in Section IV. The two applications are presented in Section V and VI. Finally, the conclusion and future works are described in section VII.

II. RELATED WORKS

In this section we first review research covering physically-based interaction in AR , followed by environment awareness in AR.

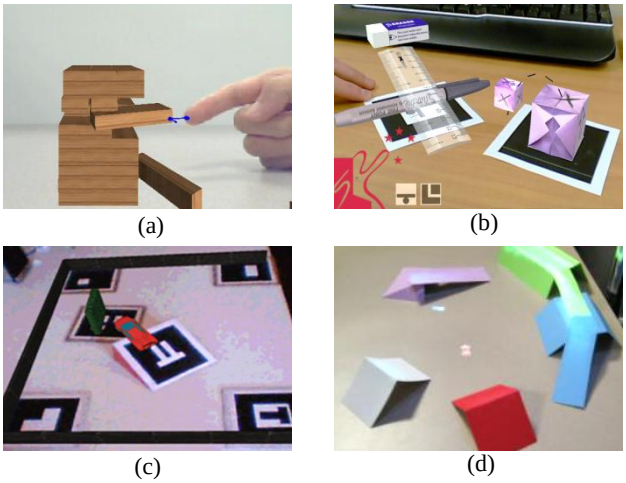
A. *Physically-based interaction in AR*

Physical simulation in AR can be achieved through the implementation of a new physics engine or an integration of an existing physics engine into the AR framework. A physics engine is a software component that can simulate physical systems such as rigid body and soft body dynamics. There are both open source physics engines such as the Bullet physics library[1], Newton Game Dynamics[2], Open Dynamics Engine[3] (ODE), and Tokamak physics engine[4], and proprietary such as Havok[5] and Nvidia's PhysX[6].

The use of physical simulation in AR has been thoroughly researched. Song et al. [4] had implemented two applications, Finger Fishing and Jenga using Newton Game Dynamics. Stereo based Fingertip detection was used to find the tip of an index finger, allowing simple interaction such as picking and dragging (Fig. 1 (a)). However, without the use of a reference marker, the camera was not movable and could only provide a fixed and pre-calibrated view.

Buchanan et al. [5] integrated ODE and OpenSceneGraph[7] [6] to simulate and render rigid body dynamics in their educational application aimed at teaching users about abstraction of forces as shown in Fig. 1 (b). MacNamee et al. [7] created a tabletop racing game and a forklift robot simulation using ODE and Havok, respectively, as well as OpenGL[8] for graphics rendering (Fig. 1 (c)). Both used ARToolKit[9] library for tracking fiducial markers. However, the physical interaction is limited to a pre-defined physics proxy that represented by a corresponding marker only.

The research closest to our work was by Wilson [8]. He used a depth-sensing camera to reconstruct the surface of the interactive area of the table and created a terrain for a car racing game, Micromotocross (Fig. 1 (d)). However, he only used a two-dimensional projective display on to the tabletop without any support for a 3D display as in AR.



(a) (b) (c) (d)

Figure 1: (a) Physically-based fingertip interaction in Jenga (b) Rube Goldberg machine in AR (c) A car accelerates up the fiducial marker slope in a car racing AR (d) Two virtual cars are projected on the tabletop racing up the real tangible ramps in Micromotocross.

B. *Environment awareness in AR*

Awareness of the environment within AR is a well-researched area. It is required to achieve correct occlusion [9], collision detection [10], and realistic illumination and shadowing effects [11]. While these features are not necessary for augmented reality, it has been shown that applications which include such cues can establish a stronger connection between real and virtual content [12].

Early attempts at environment awareness required manual modeling of all the real objects in the environment, and online localization of the camera to ensure virtual objects interact with real objects appropriately [10, 13]. This method is both time consuming and inflexible, as any changes in the environment will require recalibration.

Later approaches involved more automatic techniques, such as contour based object segmentation [14], depth information from stereo cameras [15] and time-of-flight cameras [16], or online SLAM [17]. By automatically acquiring the relevant information from the scene, there is no offline calibration requirement, and the system can correctly process the environment even when objects change or are added and removed. However, these techniques often fail in untextured scenes due to homogeneous objects, poor illumination or require and initialization process.

Recently, Izadi et al. [18] introduced KinectFusion, a system that supported a real time environment reconstruction and physically-based interaction using Kinect. However, the Kinect was used as a viewing camera and without a shared spatial reference, such as a fiducial marker, support for multiple views would not be possible.

Our approach uses Kinect to obtain the spatial information for reconstruction and employs another camera that is free to move around the scene for viewing. This method is scalable to support multiple viewing cameras and make individual viewpoints possible. We describe our framework in the next section.

III. FRAMEWORK

In this section, we begin with an overview of the framework. Following this is a discussion of the five primary components of this framework, which are marker tracking, depth acquisition, image processing, physics simulation and rendering.

A. *Overview*

To create an interaction volume, the Kinect is positioned above the desired interaction space facing downwards, as shown in Fig. 2. A reference marker is placed in the interaction space to calculate the transform between the Kinect coordinate system and the coordinate system used by the AR viewing camera. Users can also wear color markers on their fingers for pre-defined gesture interaction as described in Section VI.

Our AR framework makes use of five existing software libraries. The libraries and their functions, that we utilize, are as follows (also see Fig. 3):

[1] http://www.bulletphysics.org
[2] http://www.ode.org
[3] http://www.newtondynamics.com
[4] http://www.tokamakphysics.com
[5] http://www.havok.com/
[6] http://developer.nvidia.com/physx
[7] http://www.openscenegraph.org
[8] http://www.opengl.org
[9] http://www.hitl.washington.edu/artoolkit

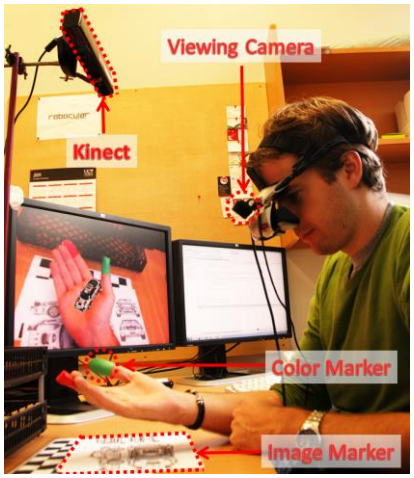


Figure 2: The interaction set up. The Kinect is suspended above the interaction volume, with the reference image marker below it.

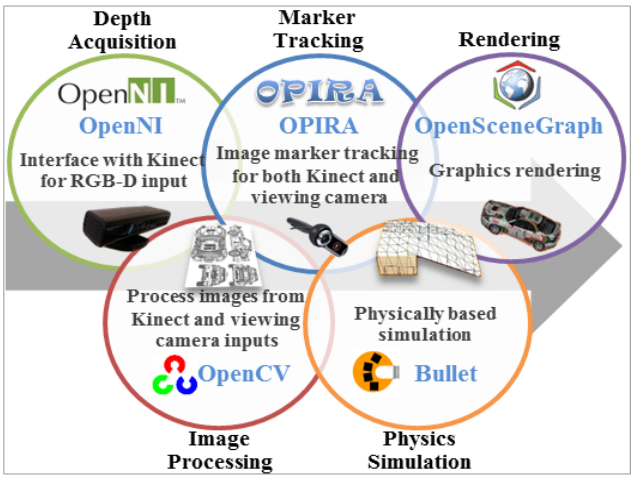


Figure 3: Primary components of our framework and its underlying software libraries.

1) OpenNI or Open Natural Interaction[10]*:* The OpenNI library is used as a means to interface with the Kinect through the Primesense's driver[11]. Color and depth images can be accessed through the API, and the two images can be automatically transformed and mapped together. OpenNIalsos provides unit conversion from projective to the real world coordinate and vice versa.

2) OpenCV or Open Source Computer Vision[12]*:* OpenCV provides a variety of functions for image processing. Of particular interest are the functions provided for logical operations on images, type conversion and smoothing algorithms.

3) OPIRA or Optical-flow Perspective Invariant Registration Augmentation: The OPIRA natural feature registration library [19] is used for all natural feature based registration due to the robustness to perspective distortion and other image transformations of OPIRA. Due to their fast computation time, SURF features [20] are used as the feature descriptor.

4) Bullet Physics: The Bullet physics library simulates both rigid body and soft body dynamics, and also provides collision detection. In this framework, we implement only the rigid body dynamics support to simulate a parallel physical world that co-exists with the real world.

5) OpenSceneGraph: is a 3D graphics API that utilizes a tree structure for managing the rendered scene. The dynamic transformation of the physical proxy in the simulated world created using Bullet can be easily applied to the scene graph for the visual feedback.

The following sections describe the components of the framework in greater detail. The data flow process within the framework is illustrated in Fig. 4.

B. Marker Tracking and Depth Acquisition

In the initialization phase, the transformation from the marker to the Kinect is calculated using natural feature based registration in OPIRA. With the Kinect's homography, H_{Kinect}, the four corners of the marker are identified and projected into the 2D Kinect's color and depth images. Since Kinect produces the color and depth image with an offset along its x-axis (see Fig. 4) due to its stereo-view, OpenNI provides functions that map the two images onto each other using the *GetAlternativeViewPointCap( )* and *SetViewPoint( ).* The 3D position for each corner is calculated using another OpenNI function, *ConvertProjectiveToRealWorld( ).* This function converts the 2D point in pixel units, (u_0, v_0), in the input image into the corresponding 3D point in millimeter units, (x_0, y_0, z_0), in the real world coordinate system.

The size of the marker is calculated in millimeters using the Euclidian distance between corners, and the AR coordinate system is established at this scale with the origin at the top left corner of the marker. Finally, the transformation between the corner positions in the Kinect coordinate system and the corner positions in the AR coordinate system is calculated and stored. The input image from the viewing camera is processed in the same way for finding the homography, $H_{viewing\_camera}$, which is subsequently applied to the camera in the OpenSceneGraph's scene.

With the transformation between the Kinect and the AR coordinate systems known, 3D data from the Kinect can be transformed into the AR coordinate system. Assuming the plane that the marker lays on is the ground plane, object segmentation can be easily achieved by using a simple threshold of the distance of each pixel from the ground plane. Fig. 5 (a) shows the point cloud data captured by the Kinect projected into the AR coordinate system in mesh form.

C. Image Processing

The depth image obtained by the Kinect is prone to missing values due to shadowing of the infrared data. To resolve this, missing values are identified and an inpainting algorithm is used to estimate their values. The OpenCV function, *cvInpaint(_)*, is used with Navier-Stokes inpainting method [21] for this purpose.

The depth image is then resized from the resolution of 640x480 to 160x120 using the nearest neighbor interpolation through OpenCV's function, *cvResize( ).* As mentioned earlier that the coordinate systems of the Kinect, (x_k, y_k, z_k), is aligned

[10] http://www.openni.org
[11] https://github.com/avin2/SensorKinect
[12] http://opencv.willowgarage.com/wiki/

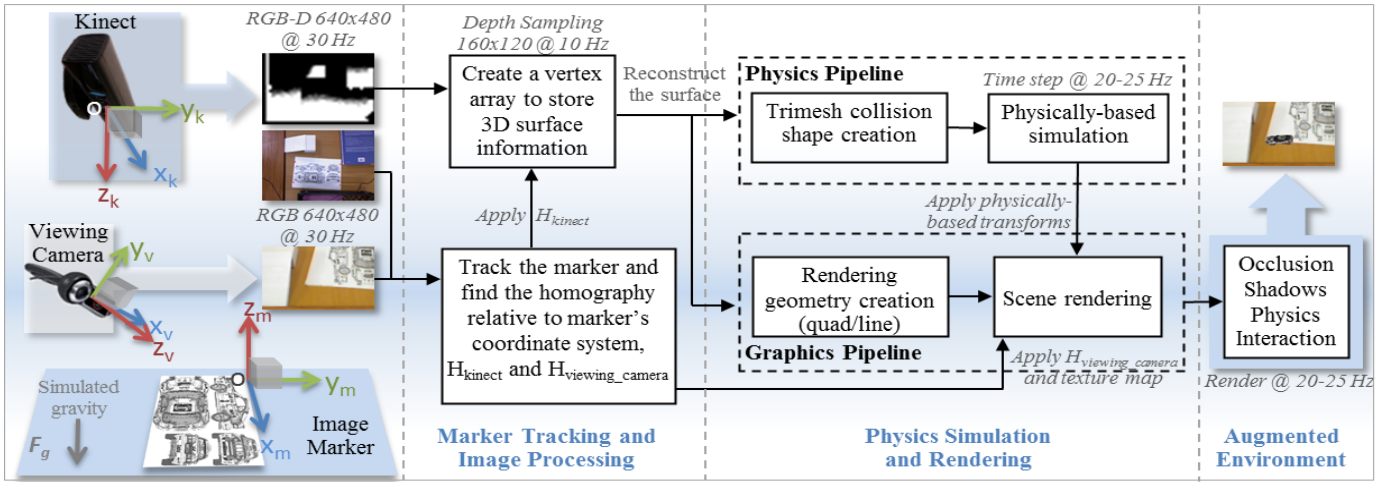


Figure 4: Overall illustration of our AR framework's architecture includes coordinate systems and information processing in each stage of the process.

to the image-based coordinate, (x_m, y_m, z_m), such that the upper left corner of the marker represents the origin in both the real and the virtual world. The depth information from the 160x120 depth image, where each pixel's value represents the height of the pixel, z_m, is stored in the vertex array with the scale in millimeters. The origin has a height equal to zero, any points above the table where the marker lies will have positive heights and those that are lower than the ground plane will be negative. This vertex array represents the spatial information of the surface of the interaction space above the tabletop.

D. *Physics Simulation*

The data in the vertex array is used to reconstruct a physical proxy of the table surface. The proxy is created as a triangle mesh (trimesh) with the Bullet's function *btBvhTriangleMeshShape()*. The trimesh is used by the physics engine for collision detection which represents the terrain on the tabletop. Optionally, the trimesh can be rendered using Delaunay triangulation in AR view to show the physics representation of the world, as shown in Figure 5 (c) and (d).

At the current stage of the simulation, it is assumed that the marker lies flat on the tabletop and the simulated force of gravity is directed downward perpendicular to the marker because there is no indicator of the actual direction of the real world gravity. One possible solution to this is to make use of the Kinect's built-in accelerometer to determine the Kinect orientation in the real world and align the virtual gravity relative to the real one.

The trimesh in the physics simulation is updated at 10 Hz, which allows the user to interact with the virtual content with realistic physics (Fig. 5 (b)). For example, the user's hand or a book is viewed as part of the dynamic scene, and can be used to pick up or push the car around. However, this interaction is limited, especially for more complex shaped objects, due to the single point of view of the Kinect. More realistic interaction would require multiple views or 3D tracking of objects to determine the orientation and complete shape of the object.

E. *Rendering*

The OpenSceneGraph framework is used for rendering in the application. The input video image is rendered as the background, with all the virtual objects rendered on top. At the top level of the scene graph, the viewing transformation is applied such that all virtual objects are transformed so as to appear attached to the real world. The terrain data is rendered as an array of quads, with an alpha value of zero. This allows realistic occlusion effects of the terrain and virtual objects, while not affecting the users' view of the real environment, as shown in Fig. 6 (a) and (b).

As planes with an alpha value of zero cannot have shadows rendered on them, a custom fragment shader was written which allows shadows from the virtual objects to be cast on the invisible terrain map. The shadows add an additional level of realism, as well as important depth cues which allow users to immediately understand the distance that the car is from the ground, which is particularly useful when the car is performing jumps or falling. The shadow casting can be seen in Fig. 6 (c) and (d).

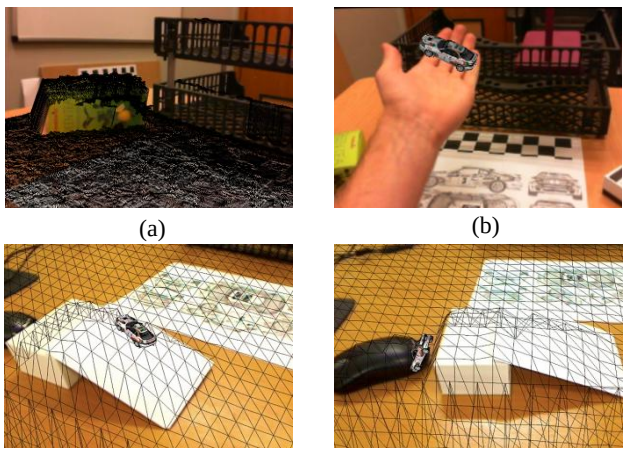
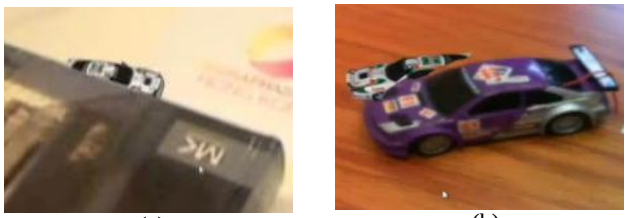
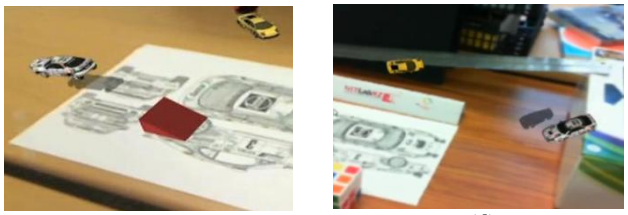


Figure 5: (a) Point clouds render on the table surface (b) The physically simulated car is resting on a user's hand (c) The virtual car accelerates up the paper ramp with wireframes overlaying the actual terrain and (d) falling off the real ramp.



(a) (b)



(c) (d)

Figure 6: (a) The virtual car is occluded by a real book and (b) occluded by a real toy car (c) The car is flying off a red virtual ramp and cast a shadow on the ground and (d) falling off a real box and cast a shadow on the floor.

IV. Performance

The goal of this research is to create a more realistic and engaging AR experience. To achieve this, the application must be capable of running in real time while ensuring the virtual content behaves appropriately in the environment which it is in, and that the user can interact with it in a natural and intuitive manner.

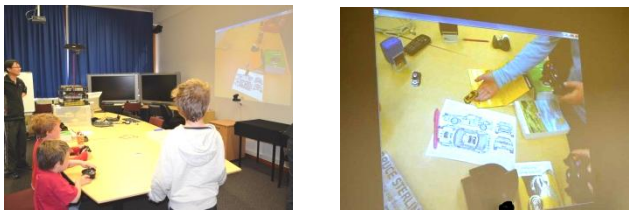
On an Intel 2.4Ghz quad core desktop computer, our applications are capable of running at over 25 frames per second. The Kinect provides an error of less than half a centimeter at the ground plane when placed between 70 - 120 centimeters from the ground plane. This error is small enough that the virtual car appears to realistically interact with real objects.

V. Applications I - ARMicroMachines

In the AR Micromachines application, users can control two virtual cars using Xbox 360 controllers. The physics simulation allows the car to react to real world objects in real-time using depth information provided by the Kinect, allowing users to create obstacle courses using real objects. The car is represented in the physics world using the Bullet's raycast vehicle object, *btRaycastVehicle*, which provides physical attributes such as tire friction, engine and breaking forces, and suspension characteristics for each car to increase realism.

A. User Experience

A single RGB camera is used as a viewing camera, and a projector provides a shared large screen display, as shown in Figure 7. Participants were given a general description of the system and shown how to use the controller. Over 50 participants have experienced the AR Micromachines application, ranging in age from elementary school children to adults in their 70s. Of these participants, their experience with AR ranged from minimal to expert users and developers. While we have not yet conducted a formal evaluation or interviews



(a) (b)

(c) (d)

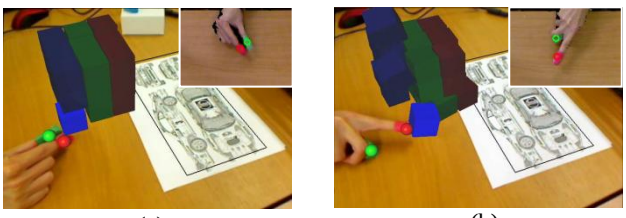
Figure 7: (a) Multiple participants can take part in ARMicroMachines (b) Projective display for a shared display (c) and Two participants control the car while the other rearranging the obstacle course.

with the participants, a number of interesting observations were made.

With only minimal explanation of the system, participants were able to engage in the game quickly. Some groups designed their own tracks and challenged their friends to races. Others examined the interaction and explored the limitation of the system by directly manipulating the car with objects and hands. Many participants said they found the game play unique and fun. All users were impressed with the realistic physics which were possible with the environment awareness afforded by the Kinect depth information. Even novice users of AR were able to intuitively understand how the interaction area could be changed by placing real objects in the view of the Kinect.

VI. Applications II - ARBlocks

In ARBlocks, a stack of rigid body cube shaped blocks is created for the user to interact with, as shown in Fig. 8. The users wear colored markers on their fingertips, which are represented in the physical simulation with spherical shape proxy. The reason for using colored marker instead of the skin-color segmentation is to remove any constraint on the color of the tabletop and select a rarer color for our markers. Fig. 8 shows that the skin and table's color are in the same range. We use the HSV color space segmentation of the Kinect's color image to isolate each colored marker and depth image to identify the 3D position of the center of mass of each marker, and estimate the position of each fingertip.



(a) (b)

Figure 8: (a) Pinching to pull the block (b) The block can be pushed with a single finger.

The pre-defined gestures that users can perform are pushing the block with the fingertip and pulling with a pinching action. The pinch is defined as moving a thumb and an index finger close to each other, and pulls the block toward the center of the two fingers and will eventually make it snap to the fingertips. The block can be rotated and picked up using these gestures.

VII. Conlusion and Future Works

In this work, we presented an AR framework which uses the Microsoft Kinect to create AR experiences with physically-based interaction and environment awareness. Depth information about the environment was obtained using the Kinect, and a physics engine was integrated to support real-time interaction of real and virtual objects. A tabletop racing game and block manipulating applications were created based on this framework.

In future, we plan to investigate additional methods of object segmentation to provide more realistic interactions between the real and virtual worlds through hand gestures. We hope to investigate using multiple Kinect cameras for a more complete representation of the environment for the physics simulation. We also want to implement multiple viewing camera support for each user. Eventually, we plan to evaluate these approaches to improving the AR experience to quantify how effective these techniques are.

References

[1] M. Billinghurst, H. Kato, and I. Poupyrev, "Tangible augmented reality," presented at the ACM SIGGRAPH ASIA 2008 courses, Singapore, 2008.

[2] E. Hornecker and A. Dünser, "Of Pages and Paddles: Children's Expectations and Mistaken Interactions with Physical-Digital Too," *Interacting with Computers*, pp. 95-107, 2009.

[3] T. Leyvand, C. Meekhof, W. Yi-Chen, S. Jian, and G. Baining, "Kinect Identity: Technology and Experience," *Computer*, vol. 44, pp. 94-6, 2011.

[4] P. Song, H. Yu, and S. Winkler, "Vision-based 3D finger interactions for mixed reality games with physics simulation," in *7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry, VRCAI 2008, December 8, 2008 - December 9, 2008*, Singapore, Singapore, 2008.

[5] P. Buchanan, H. Seichter, M. Billinghurst, and R. Grasset, "Augmented reality and rigid body simulation for edutainment: The interesting mechanism - An AR puzzle to teach Newton physics," in *2008 International Conference on Advances in Computer Entertainment Technology, ACE 2008, December 3, 2008 - December 5, 2008*, Yokohama, Japan, 2008, pp. 17-20.

[6] D. Burns and R. Osfield, "Tutorial: Open scene graph A: introduction tutorial: Open scene graph B: examples and applications," in *Virtual Reality, 2004. Proceedings. IEEE*, 2004, pp. 265-265.

[7] B. MacNamee, D. Beaney, and D. Qingqing, "Motion in Augmented Reality Games: An Engine for Creating Plausible Physical Interactions in Augmented Reality Games," *International Journal of Computer Games Technology*, p. 979235 (8 pp.), 2010.

[8] A. D. Wilson, "Depth-Sensing Video Cameras for 3D Tangible Tabletop Interaction," in *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on*, 2007, pp. 201-204.

[9] V. Lepetit and M. O. Berger, "Handling occlusion in augmented reality systems: a semi-automatic method," in *Proceedings IEEE and ACM International Symposium on Augmented Reality (ISAR 2000), 5-6 Oct. 2000*, Piscataway, NJ, USA, 2000, pp. 137-46.

[10] D. E. Breen, E. Rose, and R. T. Whitaker, "Interactive Occlusion and Collision of Real and Virtual Objects in Augmented Reality," *Technical Report ECRC-95-02, ECRC, Munich, Germany*, 1995.

[11] Y. Wang and D. Samaras, "Estimation of multiple directional light sources for synthesis of augmented reality images," *Graphical Models*, vol. 65, pp. 185-205, 2003.

[12] N. Sugano, H. Kato, and K. Tachibana, "The effects of shadow representation of virtual objects in augmented reality," in *Proceedings the Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 7-10 Oct. 2003*, Los Alamitos, CA, USA, 2003, pp. 76-83.

[13] B. MacIntyre, M. Gandy, S. Dow, and J. D. Bolter, "DART: A toolkit for rapid design exploration of augmented reality experiences," in *ACM SIGGRAPH 2005, July 31, 2005 - August 4, 2005*, Los Angeles, CA, United states, 2005, p. 932.

[14] M. O. Berger, "Resolving occlusion in augmented reality: a contour based approach without 3D reconstruction," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, 1997, pp. 91-96.

[15] J. Zhu, Z. Pan, C. Sun, and W. Chen, "Handling occlusions in video-based augmented reality using depth information," *Computer Animation and Virtual Worlds*, vol. 21, pp. 509-521, 2010.

[16] J. F. a. B. H. a. A. Schilling, "Using Time-of-Flight Range Data for Occlusion Handling in Augmented Reality," *Eurographics Symposium on Virtual Environments (EGVE)*, pp. 109-116, 2007.

[17] J. Ventura and T. Hollerer, "Online environment model estimation for augmented reality," in *8th IEEE 2009 International Symposium on Mixed and Augmented Reality, ISMAR 2009 - Science and Technology, October 19, 2009 - October 22, 2009*, Orlando, FL, United states, 2009, pp. 103-106.

[18] R. N. Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Steve Hodges, Pushmeet Kohli, Andrew Davison, and Andrew Fitzgibbon, "KinectFusion: Real-Time Dynamic 3D Surface Reconstruction and Interaction," in *SIGGRAPH Talks*, ed: ACM SIGGRAPH 2011.

[19] A. J. Clark, R. D. Green, and R. N. Grant, "Perspective correction for improved visual registration using natural features," in *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, 2008, pp. 1-6.

[20] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *9th European Conference on Computer Vision, ECCV 2006, May 7, 2006 - May 13, 2006*, Graz, Austria, 2006, pp. 404-417.

[21] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, "Navier-Stokes, fluid dynamics, and image and video inpainting," in *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, December 8, 2001 - December 14, 2001*, Kauai, HI, United states, 2001, pp. I355-I362.