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 describe our AR framework using the Microsoft Kinect as a means to create AR experiences with physically-based interaction and environment awareness. A tabletop racing game and blocks manipulating applications are created based on this framework. Depth information about the environment is obtained using the Kinect, and an integrated physics engine support a real-time interaction of real and virtual objects. The resulting augmented environment can afford occlusion, shadow and physically-based interaction.**

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

I. Introduction

Augmented Reality (AR) integrates users' task and communication space together by overlaying virtual information into the real environment. Therefore multiple users can share and collaborate on the virtual task while carrying out a natural face-to-face communication among them through gestures, speech, gaze etc. This makes AR, an ideal user interface to support collaborative works. A tabletop environment is a common workspace for such collaboration.

AR permits user interaction of real and virtual objects in the real space. Hands are not only used for gestures but also serve as a natural medium for interaction with task's artifacts. Virtual objects' manipulation in three-dimensional (3D) space in AR is commonly achieved through Tangible AR interface [1] and hence utilizing knowledge and skills in natural interaction of users in the physical world.

Despite its benefits, a user study on Tangible AR has shown that the use of physical input devices invite actions that users normally apply on physical objects [2]. However, such actions may not be supported by the system. We shall elaborate on this shortcoming with the following example. A paddle is made of a fiducial marker with a handle and it is a type of Tangible AR. It can be used for repositioning the virtual objects in AR. Once the object is positioned on the paddle, in a real world sense, user would expect the object to slide off the paddle due to gravity when it is tilted but this is not possible for a system without the 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 hinder user experience.

In this paper, we present an AR framework that provides physically-based interaction and environment awareness which are the basis for achieving better AR experience and promote AR as the user interface for collaboration on a tabletop environment. The Microsoft Kinect [3], a low cost consumer device that provides both RGB and depth-sensing cameras, has been integrated into our system. Kinect is used to examine a 3D task space above a tabletop, and so with the known transformation between the Kinect and the AR viewing camera, virtual content can be realistically composited in the environment. The user can then 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 described here. 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. Lastly, the conclusion and future works are described in section VII.

II. Related Works

This section includes two sub-sections covering physically-based interaction in AR in the former sub-section and environment awareness in AR in the latter.

A. Physically-based interaction in AR

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

Many AR researches had integrated physics in their systems. Song et al. [4] had implemented two applications, Finger Fishing and Jenga using Newton Game Dynamics and fingertip detection using a stereo camera to find the tip of an index finger and thus could perform 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] had 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] had 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 researches 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 that we found to be 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.

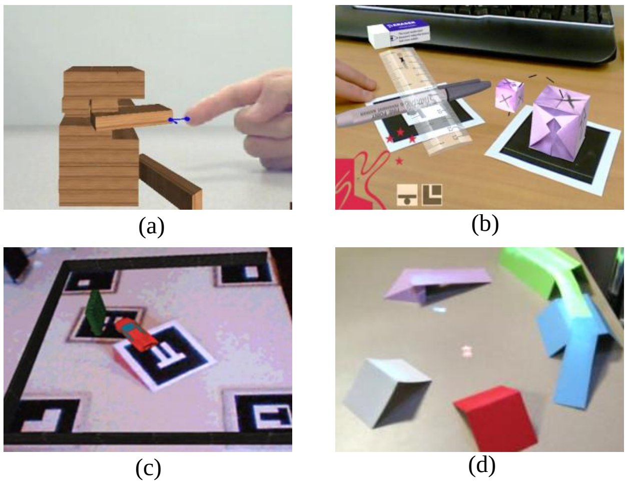


Figure 1: (a) Physically-based fingertip interaction in Jenga (b) Rube Goldberg machine in AR (c) A car accelerates up the fiducial marker slope in a car racing AR (d) Two virtual cars are projected on the tabletop racing up the real tangible ramps in Micromotocross.

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 initialization process to learn about the surrounding.

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

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

III. FRAMEWORK

There are five sub-sections covered in this section. The first sub-section is the overview of the framework. The remaining sub-sections are 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 that will be explained further in section VI.

Our AR framework makes use of five 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

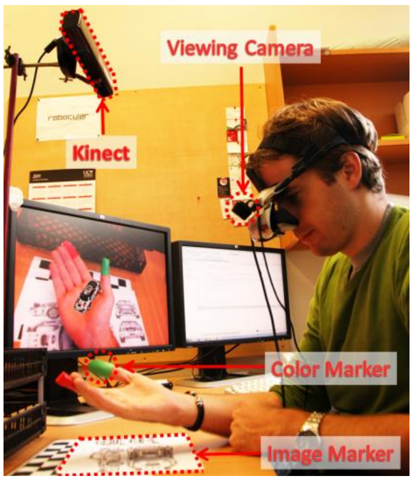


Figure 2: The interaction set up. The Kinect is suspended above the interaction volume, with the reference image marker below it.

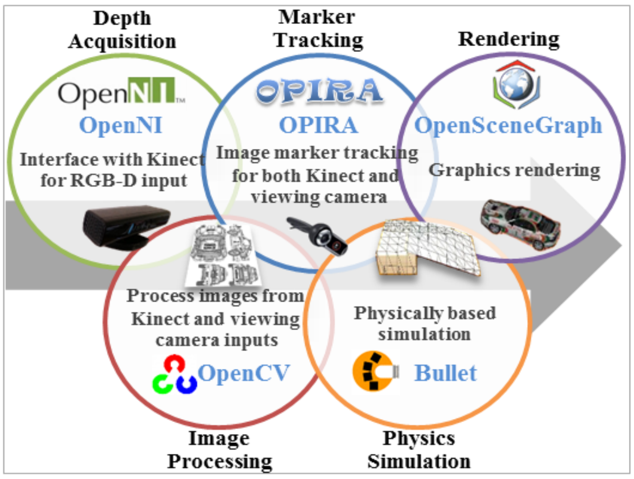

Depth Acquisition
Marker Tracking
Rendering
OpenNI
OPIRA
OpenSceneGraph
Interface with Kinect for RGB-D input
Image marker tracking for both Kinect and viewing camera
Graphics rendering
Process images from Kinect and viewing camera inputs
Physically based simulation
OpenCV
Bullet
Image Processing
Physics Simulation

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

1) OpenNI or Open Natural Interaction[10]*:* OpenNI offers much functionality such as user's skeleton tracking and audio application programming interface (API). Nevertheless, we only require this library to interface with the Kinect through the Primesense's driver[11]. Color and depth images can be accessed through the API with an advantage that they can be transformed and mapped together automatically. It also offer unit conversion from projective to the real world coordinate and vice versa.

2) OpenCV or Open Source Computer Vision[12]*:* OpenCV is used mainly for all image processing tasks such as logical operation, type conversion and smoothing algorithms.

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

4) Bullet Physics: Bullet can simulate both rigid body and soft body dynamics. It 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 one in our AR application.

5) OpenSceneGraph: is a 3D graphics API that utilizes nodes in 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 transformation node in the scene graph for the visual feedback.

The following sub-sections describe our framework in greater details and the accompanying illustration of the data flow process within the framework is illustrated in Fig. 4.

B. Depth Acquisition and Marker Tracking

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 a function that can maps the two images onto each other using the *GetAlternativeViewPointCap( )* and then *SetViewPoint( )* for mapping depth to color and vice versa. 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), of the input image into the corresponding 3D point in millimeter units, (x_0, y_0, z_0), of the real world coordinate.

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.

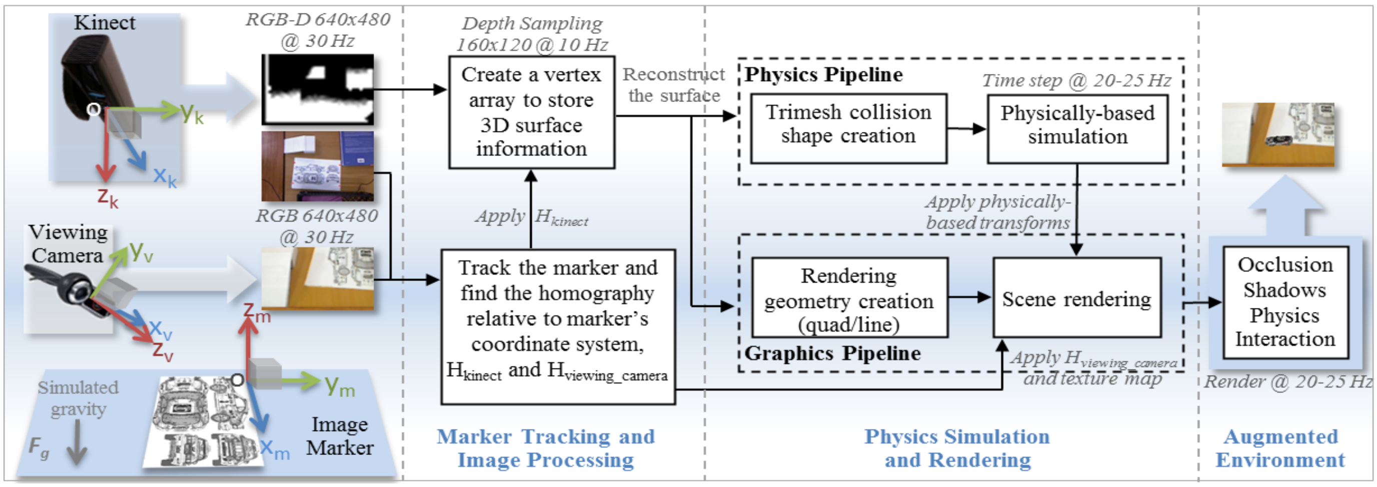
With the transformation between the Kinect and the AR coordinate systems known, 3D data from the Kinect can be transformed into the AR coordinate system. Assuming the plane that the marker lays on is the ground plane, object segmentation can be easily achieved by using a simple threshold of the distance 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. Images 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/



Kinect
y_k
x_k
z_k
RGB-D 640x480 @ 30 Hz
RGB 640x480 @ 30 Hz
Viewing Camera
y_v
x_v
z_v
z_m
y_m
x_m
Simulated gravity
F_g
Image Marker
Depth Sampling 160x120 @ 10 Hz
Create a vertex array to store 3D surface information
Apply H_{kinect}
Track the marker and find the homography relative to marker's coordinate system, H_{kinect} and $H_{viewing\_camera}$
Reconstruct the surface
Physics Pipeline
Time step @ 20-25 Hz
Trimesh collision shape creation
Physically-based simulation
Apply physically-based transforms
Rendering geometry creation (quad/line)
Scene rendering
Graphics Pipeline
Apply $H_{viewing\_camera}$ and texture map
Occlusion Shadows Physics Interaction
Render @ 20-25 Hz
Marker Tracking and Image Processing
Physics Simulation and Rendering
Augmented Environment

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, are stored in the vertex array with the unit 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 the 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, we need to assume that the marker lies flat on the tabletop and the simulated force of gravity is directed in a downward and perpendicular to the marker because there is no indicator of the actual direction of the real world gravity. One possible solution is to make use of the Kinect's built-in accelerometer to determine the Kinect orientation in the real world and so we can align the virtual gravity relative to the real one.

As the trimesh in the physics simulation is being updated in near real-time at 10 Hz, the user can 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. The 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 shader is 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).

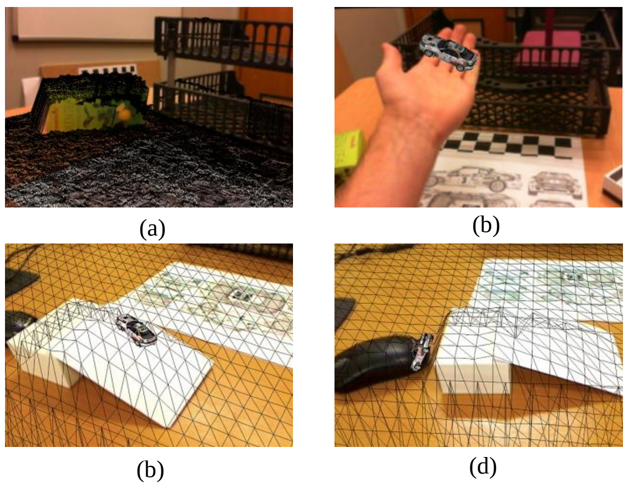


Figure 5: (a) Point clouds render on the table surface (b) The physically simulated car is resting on a user's hand (c) The virtual car accelerates up the paper ramp with wireframes overlaying the actual terrain and (d) falling off the real ramp.

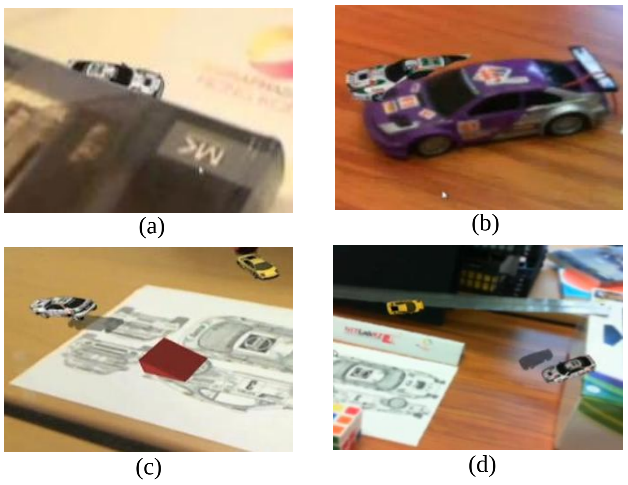


Figure 6: (a) The virtual car is occluded by a real book and (b) occluded by a real toy car (c) The car is flying off a red virtual ramp and cast a shadow on the ground and (d) falling off a real box and cast a shadow on the floor.

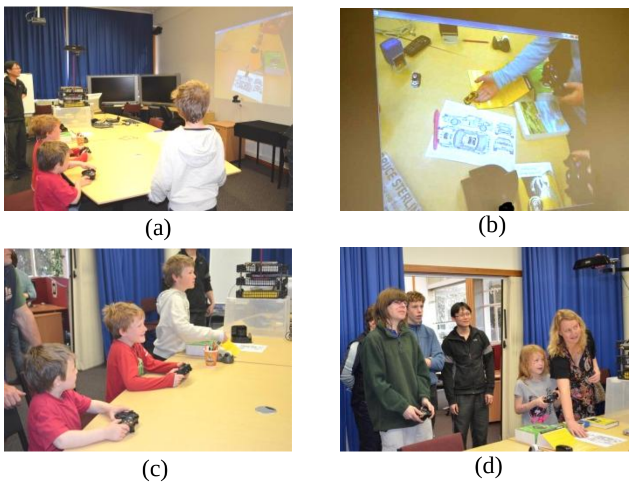


Figure 7: (a) Multiple participants can take part in ARMicroMachines (b) Projective display for a shared display (c) and Two participants control the car while the other rearranging the obstacle course.

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

AR Micromachines offers two virtual cars which can be controlled by Xbox 360 controllers for PC in the interaction space. A physics proxy 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

Single RGB camera is used as a viewing camera with a projector for a shared large screen display, as shown in Figure 7. Participants were given 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 there have been no formal evaluation or interviews with the participants at this stage, 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 by a physical proxy with spherical shape. The reason for using colored marker instead of the skin-color segmentation because we do not impose a constraint on the color of the tabletop which is in the same range as the skin-color. 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, hence the position of each fingertip.

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

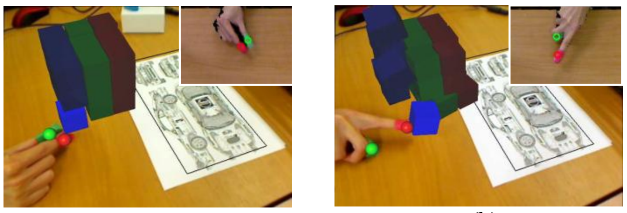

(a) (b)

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

VII. Conlusion and Future Works

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

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 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.