# Developing Physics Simulation in Augmented Reality

Damon Shing-Min Liu , Chun-Hao Yung and Cheng-Hsuan Chung

Computer Science Department, National Chung Cheng University, Chiayi, Taiwan

*Abstract*— **Use of a physics engine to drive a virtual scene is becoming more common. Physics simulations aim to model the behaviors of objects in space so as to provide realistic motions in computer animation, thereby enriching the interaction and user experience in augmented reality. In this paper, we present a cross-platform environment in which handheld and desktop-computer users can collaborate with each other in a shared scene to accomplish physically realistic experiences during the course of interaction. In realizing the system, we explore and exploit several novel techniques, including bare-hand manipulation, a client/server computing framework, a tangible mobile phone interface, and a distributed scene graph structure. To demonstrate the effectiveness of the system, we developed JengAR, a simulation of the well-known tower building game.**

*Index Terms* — **Augmented reality; Mobile phone; Tangible interface; X3D; Hand interaction; Physics simulation;**

## I. INTRODUCTION

Augmented Reality (AR) seamlessly merges physical and virtual world through the accurate registration between image captured by a camera and the virtual objects. Over the past decade, there was an evolution in the types of AR interface being developed; moreover, interactive AR interfaces need to be as intuitive as possible to be accepted by users. Current design of such interfaces is considered deficient in two aspects. First, it does not fully exploit the use of bare-hand interaction to manipulate augmented objects. Bare-hand interfaces are more natural because hands are commonly used in real life [1]. It also provides a more natural and simple control type. Second, even though most AR designs allow users to interact with virtual objects, few actually simulate physical circumstances as in real world during the course of interaction between users and virtual objects.

Furthermore, many collaborative AR applications were previously developed using desktop computers. However, owing to the technique and knowledge of AR have grown to maturity, the successful experience from several desktop-based AR research projects inspire researchers to port AR applications onto mobile platform as well. Nowadays mobile phones have full color displays, built-in cameras, fast processors, wireless communication, even dedicated 3D graphics chips, and are almost accessible for everyone. Through wireless communication, people can use their phones to access information or play entertainments, no matter where they are. On the other hand, the size of the mobile phones is well suited to be used as a tangible object, so the developers can naturally use them to design intuitive interactions. Therefore, the handheld devices have become an ideal platform for AR.

Although a number of collaborative AR applications have been developed, few studies have integrated handheld and desktop AR environment into one where mobile and personal-computer users can collaborate with each other in a shared scene through PCs and mobile phone, respectively. In this paper, we aim to present a cross-platform, physics-based, collaborative system for AR. Particularly in desktop AR environment, we design an intuitive and convenient bare-hand interface that can be utilized by users to easily interact with virtual objects. Besides, we developed JengAR, an AR game with physics simulation, to demonstrate our collaborative system and natural hand interface.

The rest of the paper is organized as follows. Section II introduces development tools and technologies that are exploited in our system. Section III provides a hand interaction method which intuitively manipulates the virtual objects. Section IV describes architecture of our client/server-based collaborative system. Section V illustrates manipulation of our demonstrative JengAR game. Section VI presents the performance data of testing Jenga application and discusses the user feedback. Finally, we conclude the paper in Section VII.

## II. DEVELOPMENT TOOLS AND TECHNOLOGIES

This section describes the tools and technologies used to develop our physics-based and marker-based AR system.

### *2.1 Marker Tracking Library*

There are several well-known computer vision based tracking libraries, such as ARToolKit [2], ARTag [3], ARToolKitPlus [4], StudierStube Tracker [5], Simplified Spatial Target Tracker (SSTT) [6], and OSGART [7]. In our work, we exploit ARToolKitPlus which is a successor to ARToolKit pose tracking library for building the marker-based AR.

ARToolKitPlus is optimized and extended for use on mobile devices. It provides several new features as well. It re-implemented the most important functions in ARToolKit with fixed-point arithmetic to speed up computing on mobile

Manuscript Received on Novermber 1, 2011
E-mall：damon@cs.ccu.edu.tw.

platform. It supported the native pixel formats for phone cameras to reduce the need for image conversion. It used the simple Id-encoded markers similar to those in ARTag to improve the marker system in ARToolKit. It does automatic thresholding by looking at the marker pattern to provide more stable tracking adjustable to changing lighting condition. It provided simple vignetting compensation to prevent that some cameras in mobile phones may exhibit strong vignetting [8].

### 2.2 Scene Graph

A scene graph is usually in the form of a tree data structure that holds nodes to different objects in a scene. There is a starting node known as *root node*, which represents the base of the entire hierarchy. A node may have many children, all of which are affected by it, but only a single parent. When 3D graphics scene becomes more and more complex, scene graph is useful in handling a great number of objects in a virtual scene, making it easier to interact with and modify the scene, and allowing for rapid development of arbitrary 3D applications. There are several commercial and non-commercial high-level graphics toolkits available such as OpenSceneGraph [9], OpenGL Performer [10], OpenSG [11], and Open Inventor [12]. Unfortunately, these toolkits do not exist on mobile phone even though they have a number of advantages as opposed to low-level graphics APIs. Our solution is to implement our own scene graph according to X3D specification [13].

X3D is a royalty-free open standard file format and scene-graph architecture to represent 3D scenes and objects using XML. In our system we use the public Microsoft XML Core Services (MSXML) to develop the *X3D parser* which loads an X3D file into a set of X3D nodes. Owing to that the scene graph of virtual scene can be written into an X3D file, we can provide several useful features: 1) users can join the running game late at any moment; 2) users can store the virtual scene into their device or server database for restoring it whenever they want to; 3) users can view and edit the virtual scene on other X3D editors; 4) developers can easily create 3D graphics applications on mobile devices.

### 2.3 Physics Engine

The objective of using a physics engine is to enhance the degree of realism by simulating the behavior of objects in real world. Through simulating Newtonian physical models and phenomena, a more realistic interaction experience for users in AR scenario can be accomplished. There are several well-known physics engines, such as PhysX [14], Havok [15], Bullet [16], Open Dynamics Engine [17], and Newton Game Dynamics [18]. We chose Bullet Physics Dynamics library, an open source engine which performs collision detection, resolves collisions and other constraints, and provides the updated world transform for all virtual objects. A simple demonstration which simulates 125 falling collision boxes is shown in Figure 1.

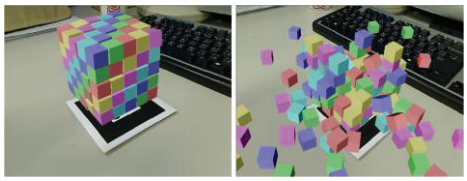

Fig. 1. Integrating Bullet Physics to AR and simulating 125 falling boxes system.

## III. HAND INTERACTION

In order to enable users to directly interact with the virtual objects using their bare-hand in real-time, we proposed a hand tracking method to track possible gesture in each video frame. It is further divided into *skin-color detection* and *fingertips finding*. First, we utilize the skin-color segmentation method that can efficiently distinguish hand region through a skin-color classifier which is operated in $YC_bC_r$ color space from the input frame. Subsequently, we exploit a fingertips finding algorithm to extract the possible fingertips for intuitively hand interaction. Due to that the algorithm can run in real-time, it enables users to naturally manipulate the virtual object in our desktop AR environment. Furthermore, to robustly track the fingertip, the shape of the finger is also taken into consideration to eliminate some false detection.

### 3.1 Skin Detection

To extract bare-hand images from the input video, the hand segmentation must be performed in the first place. We decided to use skin-color segmentation method due to its simplicity, rapidity and accuracy. Skin-color segmentation is a pixel-based skin detection technique [19], which can rapidly and effectively distinguish skin region as segmented hand contour. It can be performed in several color spaces such as RGB, HSV or $YC_bC_r$ [20]. We adopted $YC_bC_r$ color space to perform the skin-color segmentation. Color in $YC_bC_r$ is represented using luminance component (Y) and two chrominance components, blueness ($C_b$) and redness ($C_r$). The transformation simplicity and explicit separation of luminance and chrominance components make this color space suitable for skin color modeling. We also exploit an explicit skin classifier which was proposed by Garcia and Tziritas [21] for better skin-color segmentation. Figure 2 shows the results of skin-color segmentation.

Fig. 2. Results of skin detection.

### 3.2 Finger Detection

After accomplishing the skin segmentation, fingertips are then needed to be detected from the skin region for hand interaction. Our fingertip finding method is based on Hardenberg's algorithm [22]. Their algorithm explores two properties of fingertips: 1) the center of the fingertips is surrounded by a circle of skin region; 2) along a square outside the inner circle, fingertips are surrounded by a short chain of filled skin pixels and a long chain of non-skin pixels (see Figure 3).

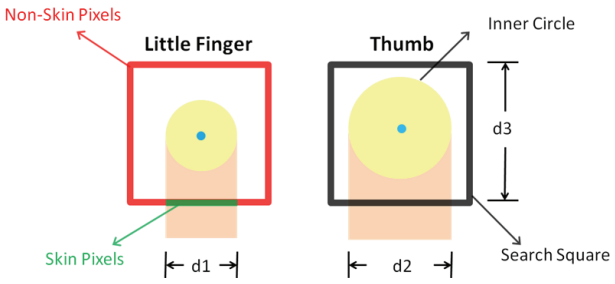


Fig. 3. A simple model of the fingertip.

Given the fingertip candidates with relative pixel location ($x$, $y$) in frame image, we then need to obtain the number and interaction point of detected fingertips for hand interaction. First, we group all candidates by computing the distance between each candidate pair. If the distance between two candidates is less than a threshold, these two candidates are allocated to the same group; otherwise, they belong to different groups. Then we sort the list, which stores all fingertip candidates in it, to accomplish the grouping procedure. After grouping all fingertip candidates, we choose the center of pixel location from candidates as an interaction point in every group. Thus, the number and the interaction point of fingertips can be calculated. Figure 4 shows results of fingertip detection.

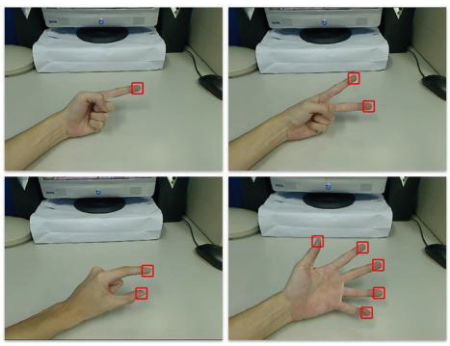

Fig. 4. Results of fingertip detection.

### 3.3 Finger Shape Filtering

Hardenberg's algorithm can reliably detect fingertip most of the times. However, it may produce false detections on the finger end that connects with one's palm due to its similar shape. To eliminate these false detections, we referred to the finger shape detection method proposed by Song et al. [23]. As a fingertip always has a long chain of filled skin pixels connected to the palm, and the width of the chain of filled skin pixels abruptly changes at the connection from the finger to the palm.

In our implementation, two steps are iterated to calculate the width of finger in each row: 1) we first choose a pixel location ($x$, $y$), which is the skin pixel on a chain along finger direction, as a *searching pivot*; 2) then we find out boundary points of the finger shape by shifting left and right from the searching pivot respectively. Thus, the width value in current row can be calculated for comparison process.

After the width finding procedure, the width value in current row can be utilized to compare with the stored width value in previous row. As mentioned in Song's algorithm, if there is no dramatically increase in length of finger, the fingertip is eliminated as a false detection. Through employing the finger shape filtering, we can effectively compensate some false fingertips detection.

## IV. PROPOSED COLLABORATIVE AR SYSTEM

We design a component-based architecture framework which can accelerate the task of developing multi-user collaborative AR application where users can interact with each other in an augmented shared space through mobile phones and PCs respectively. To this end, we exploit client/server architecture to connect different devices, where server handles all communication. When server receives the modified requests, it will broadcast the update messages to all attached devices once it approves the modifications. In order to avoid losing update messages to make some replicated scenes inconsistent, we use the TCP transport protocol, which is reliable and guarantees that each transmitted data unit is received. The overall system architecture is shown in Figure 5. There are several auxiliary modules, each of which provides different functionality.

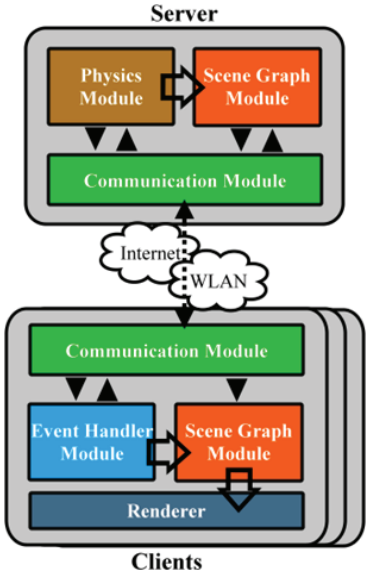


Fig. 5. The overall system architecture.

- The *Communication* module is a middleware encapsulating all networking functionality, which provides connectivity between server and client devices. Owing to that every client device attached to the server has its own copy of the scene

graph, during interaction, its copy must be kept synchronized. To this end, we also design a protocol for ensuring consistency between the shared scenes.

• The *Scene Graph* module provides convenience and flexibility to manage the virtual objects and lets *Renderer* use a simple process to draw the scene. The virtual scene can be saved in an X3D file for continuing to use. In addition, it supports a large subset of the X3D interchange profile.

• The *Event Handler* module manages all input events. Its input is a set of events and its output is a set of changes in the attributes of X3D nodes that define the scene. In addition to changing local scene, it also transmits update data to *Communication* module when the client is attached to the server.

• The *Physics* module is implemented using Bullet physics library, which provides physics simulation to simulate physical objects and phenomena.

• The *Renderer* manages all the rendering process and handles the augmented reality. Once the virtual coordinate system is constructed, the *Renderer* will use the transform matrix and a tree composed of X3D nodes to build the AR scene.

### *4.1 Server*

Client/server architecture separates computers and application software into two categories to better employ available computing resources and share data processing loads. In our system, server is a host computer which handles all connection requests from clients, maintains and manages the virtual content shared by users, and processes data storage and retrieval. Figure 6 illustrates the three threads and the connections between these modules and main window in server.

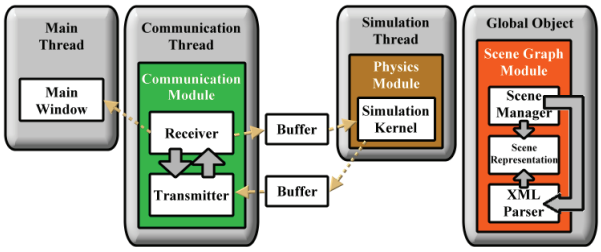


Fig. 6. Communication between modules in server.

#### *4.1.1) Main thread*

Main thread opens a dialog box that asks the user to specify what IP address and port number the server will be configured to listen on. It invokes functions on the Scene Manager to parse an X3D file for building the initial scene when user clicks the "Start Server" button. Its input is an X3D file supplied from the database and its output is the set of X3D nodes that defines the 3D scene and will be sent to the Scene Representation for initializing the scene graph. After initialization of the scene, main thread starts the communication thread and the physics thread for awaiting client connection.

#### *4.1.2) Communication thread*

Owing to that the amount and arrival time of request messages sent from clients are not predictable, for effectively using computing resources, we use a useful asynchronous event notification I/O model provided by Winsock to wait for incoming connections or data. Therefore, communication thread will be waiting until it receives the notification of network events. The reception component, i.e., Receiver, waits for the packages which are sent by clients. When server accepts a connection on a socket, Receiver will receive the user information including user name, device type (e.g., mobile phone or desktop), IP address and port number, and then forward this information to the main window. In addition, to match different processing speeds we introduce two queue buffers between communication thread and simulation thread. One buffer is used to store all content change requests, including insert object, select object, delete object, release object, and rotate object, from clients. The other is used to store the simulation results. On the other hand, the transmission component, i.e., Transmitter, is responsible for transmitting the shared scene (an X3D file) and broadcasting the update messages to the clients.

#### *4.1.3) Physics thread*

When physics thread is running, the Simulation Kernel builds a dynamics world according to the content in scene graph and then runs the simulation loop until the server is closed. The main task of Simulation Kernel is to perform collision detection, resolve collisions and constraints, and provide the updated world transform for all the objects. After each simulation, the Simulation Kernel determines whether the position and orientation of each object is changed or not and only transmits the update transform matrix of the moved objects to the buffer.

### *4.2 Client*

Figure 7 illustrates the two threads and the connections between modules and *Renderer* in client.

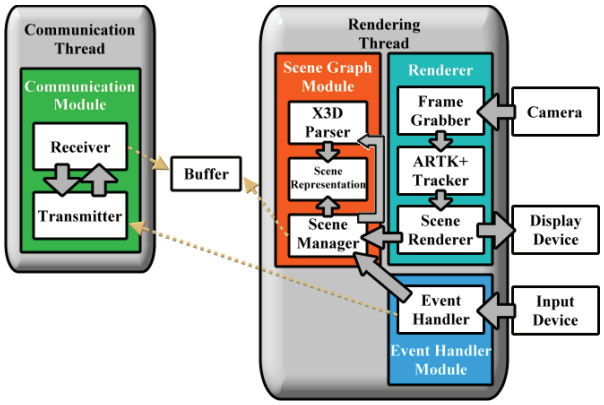


Fig. 7. Communication between modules in client.

#### *4.2.1) Rendering thread*

Frame Grabber is initiated for acquiring a video stream from a camera. Afterward, Frame Grabber grabs a sample image which is then transmitted to Scene Renderer for rendering video background of display device and to ARTK+ Tracker for detection and pose estimation of 2D fiducial markers on each

frame. As soon as the external device triggers the input event, the Event Handler will invoke the functions with parameters according to the event type, and pass the parameters to communication thread if the client is attached to server. For example, when a LeftKey (e.g., ←) event is triggered on the desktop platform, the InsertNewBox function will be invoked to add a box node to the scene graph. There is a queue buffer introduced between communication thread and rendering thread to match different processing speeds. This buffer is used to store update messages from server to change the shared scene. In addition, the scene graph is also updated according to the parameters stored in received update messages after the client has established a connection to server. The Scene Renderer creates the AR scene which is constructed using the tree of X3D nodes with the transformation matrix computed through the ARTK+ Tracker. Finally, this thread continuously runs through the infinite loop.

*4.2.2) Communication thread*

In the client, the reception component and the transmission component are different from those in server. As soon as this thread initially starts, the Transmitter sends user information to the server immediately, and then waits for the parameters passed from the Event Handler. On the other hand, the main task of the Receiver is to receive the X3D file for initializing the shared scene by all clients and the update messages for keeping the shared scene synchronized.

### *4.3 Client-Server Interaction*

In Section 4.1 and Section 4.2, we introduce the operation details of the server and the client. Here we describe the communication flow between the client and the server. After a client creates a workgroup, the server maintains and manages a shared scene for the workgroup and continues to wait for the client connection. At the beginning, the client program enters a render loop where the *Renderer* continuously manages all the rendering process and handles the augmented reality. When user tries to join the workgroup, the client connects to the server and the *Renderer* stops until the server accepts this connection. Then, the communication thread starts and the user information is sent to the server. The server converts a tree structure of X3D nodes that define the shared scene into an XML format file and sends it back to the client. Afterward, the *Receiver* notifies the *Renderer* that an X3D file is received. The *Receiver* then invokes the method on the *Scene Manager* to parse this X3D file for constructing a new scene graph and uses *it* to build the AR scene which is the same as the workgroup's.

## V. JENGAR

To demonstrate the use of our collaborative framework, we developed an AR game – JengAR (Figure 8), which refers to the well-known tower building game *Jenga* [24]. Furthermore, due to that the physics engine has been integrated into our system, the behavior of virtual objects in JengAR is just like those blocks which can be manipulated in the real world.


Fig. 8. AR game – JengAR in initial stage.

For handheld AR, Henrysson et al. [25] [26] performed a user study of virtual object manipulations on mobile phones. They found that tangible input techniques are best for *translation* tasks, while keypad input is best for *rotation* tasks. The usability experiments provide a good direction for us to design interactive interfaces. In our work there are several interaction options for virtual object manipulation.

*1)* Selection: We use a method which is fast and requires no complex calculations. Essentially the way how this method works is that each object in a virtual scene is assigned a unique alpha value. In other words, the object ID is stored in the alpha channel. This means a virtual scene can contain 255 objects at most (an alpha value for video background).

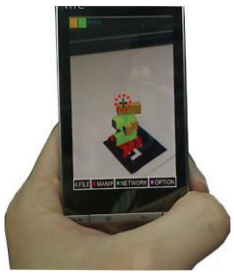

Fig. 9. The crosshair.

When user picks an object, the application with this method simply reads back the alpha value of the pixel under the crosshair's center (Figure 9) and then searches through the list of objects looking for a matched alpha ID. If one is found, the user has a selection and a white wire bounding box appears around the selected object (Figure 10).

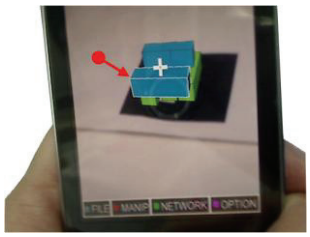

Fig. 10. The white wire bounding box.

*2)* PickColor: To allow users to select colors from real world to change color of the virtual object, we also call glReadPixels to read back the color value of the pixel under the

crosshair's center.

*3)* Translation: Owing to mobile phone and webcam can be easily manipulated in hand, it is therefore possible to use them as a tangible input device. In this translation case, the virtual object is fixed in space relative to the phone, thereby can be positioned and translated at the same time. In other words, during the course of interaction user can physically move the phone, and then a picked object is translated in the same direction.

*4)* Rotation: The selected objects can be rotated around x-aixs, y-aixs, or z-aixs by pressing a finger touch.

*5)* Insertion: We compute the point of intersection between a ray from the near clipping plane to the far clipping plane and the marker plane, and then add object into the scene according to the position of the point.

*6)* Deletion: For removing virtual objects from the AR scene, we also read back the alpha value of the pixel under the crosshair's center and then search through the list of objects looking for a matched alpha ID.

*7)* Finger based user graphical interface: Currently, most smartphones have no physical keypad and joypad, instead, users use a touch-screen as an input device to quickly access the device's functions and applications. Therefore, we design a finger based graphical user interface (GUI) for collaborative AR applications. In addition, we make each menu item as virtual button to enable / disable specific functions, perform certain tasks, or interact with the virtual objects (Figure 11). Owing to that we place the GUI at the bottom of the screen (Figure 12), users can use a simple touch of the thumb to select options from a hierarchical menu.


Fig. 11. Menu items.

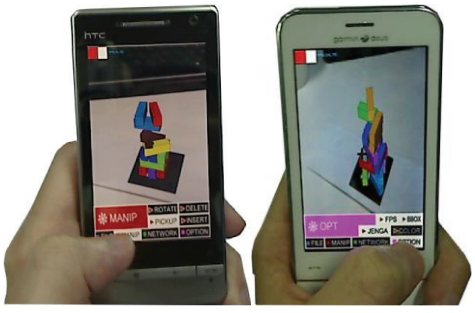

Fig. 12. Graphical user interface.

In a handheld AR environment, players can add blocks into the scene and stack them to build a tower. They can specify the position of blocks and move the selected object to another desired position by physically moving of the mobile phone. In addition to adding blocks for stacking, the players also can remove blocks from anywhere below the top layer and put them on the top of the tower (Figure 13).

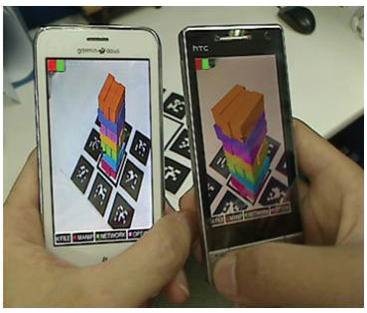

Fig. 13. JengAR in handheld device.

## VI. EVALUATION

### *6.1 Temporal Performance*

The Jenga application was tested on HTC Diamond2, ASUS M10, and HP Desktop PC, and the video capture resolution in these devices are 240x320 pixels, 176x144 pixels, and 640x480 pixels, respectively. The overall average performance is shown in Figure 14. In general, choosing lower resolutions often results in better performance but reduces tracking accuracy and viewing quality. In order to further improve the performance on mobile phones at higher resolutions, we measured the average processing time in each of several parts including 2D graphics rendering, single marker tracking, multi-marker tracking, 2D texture mapping for video background, and 3D graphics rendering, to realize the performance bottleneck in the system. As can be seen in Table 1, we found that most of the performance time is spent on marker tracking and 2D texture mapping. Since OpenGL|ES only accepts textures with size $2^m$ x $2^n$, we use *glTexSubImage2D* to update part of the texture on mobile phones. It is much slower than updating a whole new texture with *glTexImage2D*. Therefore, to draw video background without OpenGL|ES texture mapping may be a simple and nice way for increasing the processing speed on the mobile phone.

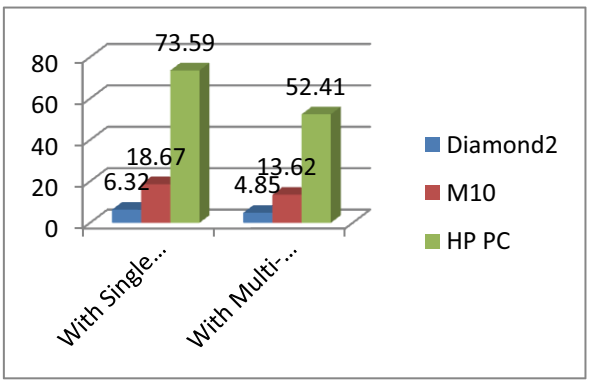


Fig. 14. The overall average performance.

TABLE 1. AVERAGE PROCESSING TIME OF EACH PART

| | HTC Di-amond2 | ASUS M10 | HP Desk-top PC |
|---|---|---|---|
| 2D graphics rendering | 17ms | 4ms | 1ms |
| Single marker tracking | 42ms | 19ms | 4ms |
| Multi-marker tracking (8 markers) | 114ms | 42ms | 15ms |
| 2D texture mapping | 84ms | 18ms | 10ms |
| 3D graphics rendering (10 blocks) | 18ms | 5ms | 2ms |

### 6.2 *User Study*

For understanding the experience of user collaboration and assessing usability of our proposed system for collaborative AR, we conducted a small user study on 15 participants, including 13 males and 2 females, with their age ranging from 22 to 31 years old. All of the participants were graduate students from our university. Before the user study, most of them were familiar with the concept of augmented reality but only three had the experience of playing augmented reality games.

The procedure consists of three parts: 1) General training: A researcher gave an introduction to the game interface and the concept of our system by following a pre-written script and showing a pre-recorded video. After that, the participants were asked to play the Jenga game on single user mode for learning the game controls on mobile phone and PC; 2) Playing: Each group of three participants played the Jenga game together. One used HP desktop PC and the others used HTC Diamond2 and ASUS M10 respectively. Besides, each of them has his/her own work space; 3) Questionnaire: After playing the game the participants fill out the questionnaire independently. Each of the questions was answered on a scale 1 to 7 where 1 = Not Very Easy/Enjoyable and 7 = Very Easy/Enjoyable. The questions asked listed in Table 2.

TABLE 2. QUESTIONNAIRE

1) I can easily move blocks to the desired position.
<Not Very Easy 1 2 3 4 5 6 7 Very Easy>
2) I can easily to be aware of what my partners were doing.
<Not Very Easy 1 2 3 4 5 6 7 Very Easy>
3) I can easily interact with my partners.
<Not Very Easy 1 2 3 4 5 6 7 Very Easy>
4) I felt AR interface brings more fun to the Jenga game.
<Not Very Enjoyable 1 2 3 4 5 6 7 Very Enjoyable>

Through the user study, the participants gave us positive feedback (see Figure 15). Most of participants felt it was easy to move blocks to the desired position and tangible device is more intuitive and natural than using other input devices for translating the virtual objects. As can be seen from the responses to Questions 2 and 3 each member of a group can utilize whether PC or mobile phone to easily interact with each other. In general almost all of participants agreed that AR interface brings more fun to our Jenga game. Moreover, through playing and introduction to the concept of our system they thought our system is useful for developing collaborative AR applications.

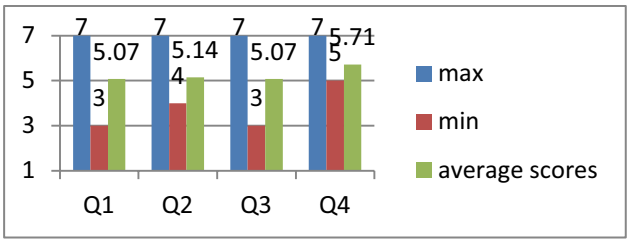


Fig. 15. Feedback scores.

## VII. CONCLUSION

In this paper, we present a cross-platform system which integrates desktop computers and mobile devices for developing collaborative augmented reality applications. Combining augmented reality, mobile computing, and computer supported collaborative work (CSCW) can bring realistic and mobile advantages over common virtual system.

Moreover, we have designed a natural hand interface that enables users to directly touch virtual objects using their bare-hand in a desktop AR environment. Compared to traditional interfaces such as mouse and keyboard, our hand interface can be controlled in an intuitive way for AR, since people are accustomed to interact using their hands with objects in real life.

We also developed a distributed scene graph structure according to X3D specification to maintain the AR scene shared by multiple users. In it, users are allowed to late join and save scene to a file for later collaboration sessions. The communication between each client is realized through client/server architecture, in which server handles all connections and each client needs not worry about the existence of other clients. Thus, the distributed features allow users to freely choose to utilize whether PCs or mobile devices to easily interact with each other.

Finally, we implement a simple tower building AR game – JengAR to demonstrate our collaborative system and natural hand interface. Furthermore, we integrate a physics engine into our system for enriching more realistic behavior on virtual objects, so that when users pull a virtual block from the tower, they can see that the tower structures are affected by direction and speed of pulling operations according to physics law.

## REFERENCES

[1] M. K. Lee and M. Billinghurst, (2009). Space-based gesture classification in an augmented reality environment. In *Proc. the International Workshop on Ubiquitous Virtual Reality (IWUVR 2009)*, pp. 36-39.
[2] ARToolKit – http://www.hitl.washington.edu/artoolkit/

[3] ARTag – http://www.artag.net/
[4] ARToolKitPlus – http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php
[5] StudierStubeTracker – http://studierstube.icg.tu-graz.ac.at/handheld_ar/stbtracker.php
[6] Simplified Spatial Target Tracker (SSTT) – http://technotecture.com/augmentedreality
[7] OSGART – http://www.osgart.org/wiki/Main_Page
[8] Wagner, D. and Schmalstieg, D. (2007). ARToolKitPlus for pose tracking on mobile devices. In *Proceedings of 12th Computer Vision Winter Workshop*.
[9] OpenSceneGraph – http://www.openscenegraph.org/
[10] OpenGL Performer – http://web.archive.org/web/20071224141002/www.sgi.com/products/software/performer/
[11] OpenSG – http://www.opensg.org/
[12] Strauss, P. and Carey, R. (1992). An object oriented 3D graphics toolkit. In *Proc. SIGGRAPH '92*, pp. 341-349.
[13] X3D – http://www.web3d.org/x3d/
[14] nvidia PhysX – http://www.nvidia.com.tw/object/ physx_new_tw.html
[15] Havok – http://www.havok.com/
[16] Bullet Physics – http://bulletphysics.org/wordpress/
[17] Open Dynamics Engine – http://www.ode.org/
[18] Newton Dynamics Engine – http://newtondynamics.com/forum/newton.php
[19] P. Peer, J. Kovac, and F.Solina, (2003). Human skin colour clustering for face detection. In *Proc. EUROCON 2003 – International Conference on Computer as a Tool*.
[20] V. Vezhnevets, V. Sazonov, and A. Andreeva, (2003). A survey on pixel-based skin color detection techniques. In *Proc. International Conference on Computer Graphics and Vision (GRAPHICON2003)*.
[21] C. Garcia and G. Tziritas, (1999). Face detection using quantized skin color regions merging and wavelet packet analysis. *IEEE Transactions on Multimedia*, 1(3), pp. 264-277.
[22] C. Hardenberg and F. Bérard, (2001). Bare-hand human-computer interface. In *Proc. the 2001 workshop on Perceptive user interfaces*, vol. 15, pp. 1-8.
[23] P. Song, S. Winkler, S. Gilani, and Z. Zhou, (2007). Vision-based projected tabletop interface for finger interactions. In *Proc. Human-Computer Interaction*, pp. 49-58.
[24] D.S.M. Liu, C.H. Yung, and C.H. Chung, (2011). A physics-based augmented reality Jenga stacking game. In *Proc. the 2011Workshop of Digital Media and Digital Content Management*.
[25] A. Henrysson, M. Billinghurst, and M. Ollila, (2005). Virtual object manipulation using a mobile phone. In *Proc. the 2005 International Conference on Augmented Tele-Existence,* pp. 164-171.
[26] A. Henrysson, J. Marshall and M. Billinghurst, (2007). Experiments in 3D interaction for mobile phone AR. In *Proc. the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, pp. 187-194.

**Damon Shing-Min Liu** is on the faculty of the Department of Computer Science and Information Engineering at National Chung Cheng University. His research interests include computer graphics and animation, virtual reality and simulation systems, high-performance I/O for data-intensive Grid/Cloud computing applications, interactive real-time scientific visualization and data analysis. He received his BSc in Electrical Engineering from National Taiwan University, his MSc in Computer Science from Brown University, and his PhD in Computer Science from University of California, Los Angeles (UCLA). He is a member of the ACM and the IEEE Computer Society.

**Chun-Hao Yung** has received his Bachelor's degree (2008) in Computer Science from National Chi Nan University and Master's degree (2010) in Computer Science from National Chung Cheng University. He is associated with Innovative Computing and Visualization Laboratory. His research interests lies in augmented reality, human-computer interaction, image processing, and physics simulation.

**Cheng-Hsuan Chung** has received his bachelor's degree (2008) in Computer Science from National Taiwan University of Science and Technology and Master's degree (2010) in Computer Science from National Chung Cheng University. He is associated with Innovative Computing and Visualization Laboratory at National Chung Cheng University. His research interests are augmented reality (AR) on mobile devices and non-photorealistic rendering (NPR).