



Department of Computer Science and Engineering  
The Chinese University of Hong Kong



# iDigitable

Final year project 2012

Final Year Project 2011-2012 (2nd Term)

LYU1103

---

**Project Title:** i.Digi.T.able - Digital Interactive Game Interface Table Apps for iPad  
**Prepared by:** Ng Ka Hung (1009615714) khng9@cse.cuhk.edu.hk  
**Supervisor:** Professor Michael R. Lyu

< This is a blank page >

# Abstract

---

This report covers details of our idea, design and implementation of our final year project *iDigi.T.able* in year 2011 – 2012.

Our project goal is to implement an **Augmented Reality (AR)** game on iPad which allows multi-users to share a gaming interface via network.

We first introduce the technologies we based on and some basic thoughts about why we are doing this project in the Objective section.

We will then discuss about techniques we have investigate that could help and methodologies we have used in the project. For the AR engine part, we will justify the functionalities of different SDKs and analyze them in detail. The network connection method will also be discussed.

Then we will cover the design of our system. In this part, we will present the architecture and implementation plan for our project. Each module will be discussed in detail.

In the upcoming experiment part, we will focus on how we try out new techniques to assist our work in the project. We also tested the tools and see potential room of improve in that. Difficulties and challenges we are currently facing will also be mentioned.

Finally, in the conclusion part, summary of work things learnt will be recorded.

# Contents

---

|   |    |
|---|----|
| <b>Abstract</b> .....                     | 3  |
| <b>Contents</b> .....                     | 4  |
| <b>Chapter 1. Introduction</b> .....      | 6  |
| 1.1 Motivation.....                       | 6  |
| 1.2 Background.....                       | 9  |
| 1.3 Objective .....                       | 12 |
| 1.4 Development Environment .....         | 12 |
| 1.5 Runtime Environment.....              | 14 |
| <b>Chapter 2. Augmented Reality</b> ..... | 17 |
| 2.1 History.....                          | 17 |
| 2.2 Types of AR.....                      | 18 |
| 2.3 Applications .....                    | 19 |
| 2.4 AR in game industry .....             | 20 |
| 2.5 Marker detection and recognition..... | 21 |
| <b>Chapter 3 Qualcomm AR SDK</b> .....    | 25 |
| 3.1 Introduction.....                     | 25 |
| 3.2 System architecture.....              | 26 |
| 3.3 Trackables .....                      | 28 |
| 3.4 Target management system.....         | 31 |
| 3.5 Development on iOS .....              | 36 |
| 3.6 Compare with String AR.....           | 37 |
| <b>Chapter 4. Networking</b> .....        | 40 |
| 4.1 Network Socket.....                   | 40 |
| 4.2 HTTP Request .....                    | 41 |
| 4.3 Game center .....                     | 42 |
| 4.4 Peer to Peer.....                     | 42 |
| 4.5 JSON .....                            | 43 |

|  |     |
|--|-----|
| <b>Chapter 5. Design and Implementation</b> .....  | 45  |
| 5.1 Idea .....                                     | 45  |
| 5.2 Settings.....                                  | 46  |
| 5.3 Modules .....                                  | 47  |
| 5.4 Game design.....                               | 58  |
| <b>Chapter 6. Pong</b> .....                       | 60  |
| 6.1 Background .....                               | 60  |
| 6.2 Overview.....                                  | 61  |
| 6.3 Definitions.....                               | 62  |
| 6.4 General Flow .....                             | 64  |
| 6.5 System Architect.....                          | 67  |
| 6.5.4 User Control .....                           | 113 |
| <b>Chapter 7. Experiment</b> .....                 | 114 |
| 7.1 Camera match-moving .....                      | 114 |
| 7.2 Networking.....                                | 120 |
| <b>Chapter 8. Contribution of work</b> .....       | 123 |
| <b>Chapter 9. Conclusion</b> .....                 | 126 |
| <b>Chapter 10. Progress and difficulties</b> ..... | 127 |
| 10.1 Difficulties and challenges.....              | 128 |
| <b>Chapter 11. Evaluation</b> .....                | 130 |
| <b>Chapter 12. Acknowledgement</b> .....           | 132 |
| <b>References</b> .....                            | 133 |

# Chapter 1. Introduction

## 1.1 Motivation

The previous project *i.Digi.T.able* (supervised by Prof. R Michael Lyu) implements a **Digital Interactive Game Interface Table** using plasma display monitors and cameras to create a shared gaming platform over internet. The previous version of i.Digi.T.able enables users to play chess games and mini board games (e.g. Go chess, Chinese chess, Uno) in different place using real chess/game setup. The video image of the chess board is recorded and sent over the Internet. Hence creates a platform for users in different room to play in the same game interacting with the real object.

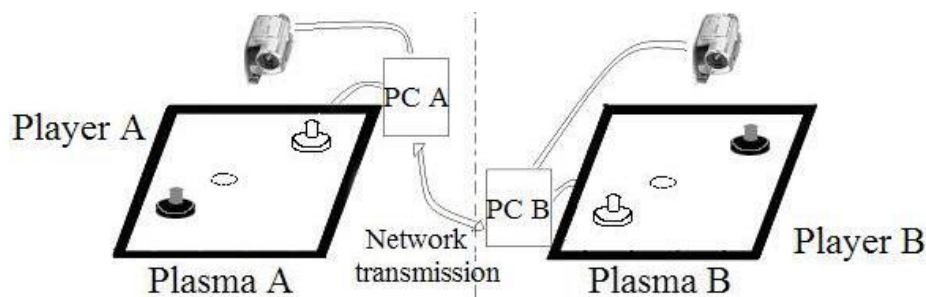


Fig.1.1.1 Original project implemented by Plasmas and Overhead cameras

When we review the above project, we are inspired by how multi-user can interact “within a same platform while they are in different place” . We also found that the concept of “combining reality and computer graphics” to enhance user experience in the Augment Reality projects very interesting. Hence, we came up the raw idea of the project.

### Augmented Reality (AR)

AR has been introduced to computer industry since 1950s. However, due to limitation of hardware and low efficiency of related algorithm, its developmental progress was so slow. Until recent years, AR becomes popular especially on *i.Digi.T.able* - Digital Interactive Game Interface Table Apps for iPad

mobile devices and has a wide variety of application. Adding AR element to games can easily enhance the realism and impressiveness.



Fig.1.1.2 Example - Hoops AR

There are some outstanding AR games available on the App Store. For example the Hoops AR, it is a game played with a virtual basketball court superimposed on the location of the predefined marker i.e. the ticket.



Fig.1.1.3 Example – Wikitude World Browser

Another example is **Wikitude World Browser**. It makes use of GPS, accelerometer and digital compass. By detecting surroundings, users can access information on the augmented layer instantly.



Fig.1.1.4 Rock 'Em sock 'Em Robots is an AR game example

We observe that multi-player AR games are so rare on the market. One of the real cases is the **Rock 'Em Sock 'Em Robots** made corporately by Mattel and Qualcomm. However, this game has made not much difference from a single-player game since it requires user to use the same piece of marker.

## Our motivation

We wish to depend on the relative position of device to the marker. In this way, we can construct a virtual space between two devices. Each device can view object respect to its relative position in the virtual space. It is also possible to view another device's virtual position and interact with it.

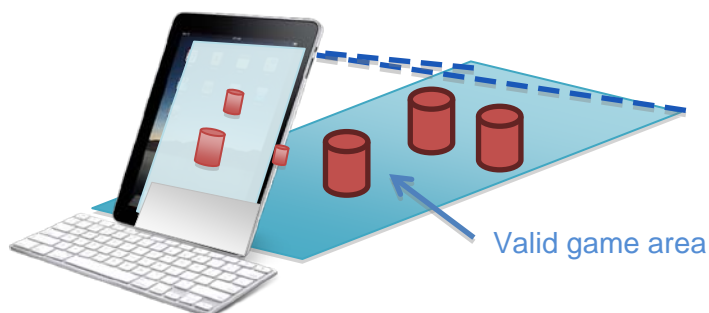


Fig.1.1.5 Our first idea of the i.Digi.T.able

i.Digi.T.able - Digital Interactive Game Interface Table Apps for iPad





## 1.2 Background

Despite the rise of AR applications, there are only a few mature software libraries which provide well developed tool kit. Two of them are the **Qualcom** and **String** AR SDKs. The Qualcom AR SDK is named **Vuforia** in February 2012.

The Qualcomm AR SDK (**Vuforia**) utilizes computer vision technology to tightly align graphics with underlying objects and features support for image targets, frame markers, virtual buttons and simple 3D objects. It is open-source and available for both Android and iOS platforms. The String AR SDK provides less in the aspect of functionality and its extendibility is very limited for free license.

Below is a brief comparison between two:

|                        |  |  |
|------------------------|--|--|
|                        |  |  |
| License                | Free   | Free for limited version   |
| Platform               | iOS, Android   | iOS, Android (in progress)   |
| Multiple markers       | Yes  | No   |
| 3-rd Party Integration | Yes, Unity3D   | Yes, Unity3D   |

Detailed comparison on functionalities will be further discussed in later chapters. AR applications have be deployed on different platforms, each platform has its own characteristic.

Here we have a brief comparison of AR application on different platforms:

| Platform  | Advantage   | Disadvantage  |
|-----------|---|---|
| PC        | <ol style="list-style-type: none"> <li>1. Large display screen</li> <li>2. Higher computational power*</li> </ol>   | <ol style="list-style-type: none"> <li>1. Lack of mobility</li> <li>2. Need external camera</li> </ol>            |
| Mobile    | <ol style="list-style-type: none"> <li>1. High market share</li> <li>2. High mobility</li> <li>3. Touch gesture control</li> <li>4. Built-in camera</li> </ol>                                  | <ol style="list-style-type: none"> <li>1. Smaller display screen</li> <li>2. Lower computational power</li> </ol> |
| Tablet PC | <ol style="list-style-type: none"> <li>1. High market share</li> <li>2. High mobility</li> <li>3. Touch gesture control</li> <li>4. Large display screen</li> <li>5. Built-in camera</li> </ol> | <ol style="list-style-type: none"> <li>1. Lower hardware extendibility</li> </ol>                                 |

\* nowadays, the computational power of Tablet PC and mobile has already achieved a PC-standard level, different between PC and mobile device in this aspect is not very significant.

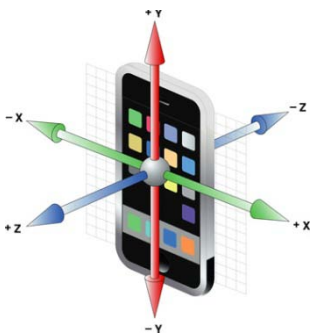
Instead of the old i.Digi.T.able system design, we are going to design another system to achieve the goal: creating a “space sharing” platform. We have considered the special characteristic of our client device - iPad, hence we chose to adapt a more fashionable and interesting design.

Such set up should make good use of the functionalities iPad naturally provides. Possible impressive features we can consider to include:

1. The front cam can be used for video conferencing; players can communicate with each other directly.

However video conferencing function will use up lots of network traffic.

2. The front cam can be used to track the user's position towards to screen, which potentially can be used to implement VPT (Visual Perception Technology) if possible.
3. Players can interact with the game using the touch monitor other than only moving the game objects around.
4. The game can be set up on any plane platform.
5. Game virtual objects can be displayed in a 3-D manner instead of top-view only.
6. The GPS and accelerometer built in the iPad can help enhancing the system.



#### 1.2.1 accelerometer helps detect device orientation and rolling

7. Wireless network system enables iPad to go anywhere, however we may need a server to host and process the data.

## 1.3 Objective

In our Final Year Project, our group is trying to use iPad as the development platform to implement an AR game.

The game should enable users using 2 different iPad clients having the experience that they are sharing the same virtual space.

Main objectives of our project:

- Track the real-object mark and determine the camera's position
- Display simple objects on virtual space depends on real space scenes
- Exchange position information between 2 iPad clients
- Implement a simple AR game on iOS platform (iPad)

## 1.4 Development Environment

| Development      | Platform / tool   | Programming Language(s) involved |
|------------------|-------------------|----------------------------------|
| Client program   | Mac OS/ Xcode 4.0 | Objective-C                      |
| Server (network) | Linux / vim,pico  | PHP / Shell script / C / SQL     |

Most of the development processes are carried on Mac OS. It is an obligation for iOS applications to be developed on MacOS.



### Xcode

Xcode is an integrated IDE for developing MacOS and iOS application. It includes a modified version of free software GNU Compiler Collection, supporting several common programming

languages, such as C, C++, JAVA, Objective-C, Python, Ruby, etc. .Its package

[i.Digi.T.able](#) - Digital Interactive Game Interface Table Apps for iPad

comes with Cocoa and Cocoa Touch frameworks. The frameworks provide User Interface, Gesture detection control and more necessary libraries for iOS devices.

The Xcode version we are currently using in our development is 4.0.2. This version is released on March 2011.

We have chosen Xcode as our main development environment as many other iOS application developers do. It is well-supported for syntax and library API code auto completion. Moreover the debug and compiling tools have no better replacement. It is smooth and handy to use Xcode to finish our client side program.



### PHP

For the server, we have chosen PHP (PHP: Hypertext Preprocessor) to develop the server program. It is mainly because of its powerfulness as a server side programming language.

PHP is a general-purpose scripting language mainly use for web development. It is free software under PHP License. PHP can be set up on almost every operating system.

PHP includes free and open source library on its build. It also embeds with MySQL, SQLite as database server.

PHP programs can be deployed as web application or stand alone programs. In our implementation, we use http requests as main technique to communicate between machines. PHP is easy to write and provides various of good libraries to support our work.

## SQLite



SQLite is a C library that implements an embeddable SQL database engine. PHP programs can use SQL database service without running a separate RDBMS process.

SQLite is not a client library used to connect to a big database server. The SQLite library reads and writes directly to and from the database files on disk, say *info.db*. The transaction of SQLite is atomic and consistent, which provides a durable feature for safe database usage. Another promising advantage of SQLite is zero-configuration. Setup is no needed before use.

SQLite can be used as application database for mobile apps. It can also be used as a database for gadgets such as some hardware device (e.g. mp3, PDA etc.). Some medium-size websites also use SQLite because of its easiness of setup.

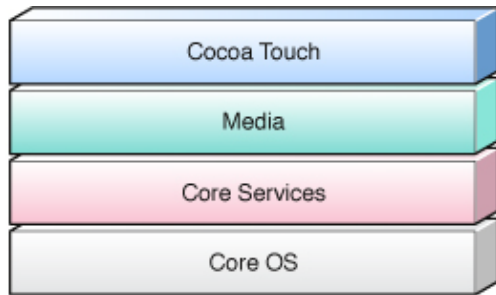
Using SQLite is handy, we can insert SQL queries in PHP statements to invoke database functions. Returning results can be used for future computation as output.

## 1.5 Runtime Environment

For the development, we have chosen iPad as the platform. iPad is a trendy tablet computing device nowadays. iOS is a mobile operation system developed by Apple Inc. Originally it is designed for iPhone but now is extended onto other Apple's product such as iPad and iPod touch. The first iOS was released in 2007.

In year 2011, there are 500,000 Apps for the iOS platform. iOS has become a popular platform on mobile. The iOS version for our runtime environment is iOS4.3 running on iPad2.

iOS4.3 runs on ARM family processor. For iPad2, it is using the Apple A5 processor and builds on top of a unix-like kernel. The abstract layering architecture of iOS gives developer choices when developing their applications in different layer.



**Fig.1.5.1 iOS technology layer**

The main reason iOS is adopted as our runtime platform is due to the User Interface experience and portable feature of iPad. With a 9.7 inch multi-touch sensible monitor, virtual world can be displayed realistically. We also take advantage of the front/rear cameras built-in on iPad. The process of tracking real object is done by analyzing the camera buffer.

There are also much API available since iOS is getting more popular, this would help developers in coding on their project more efficiently.

## iOS simulator



**Fig.1.5.2 iPad Screen on iOS simulator**

Apple provides a debug tool for iOS application developers that let developers need not compile the program to a designated hardware machine every time.

iOS simulator could simulate gestures on real device (such as drag and drop, pinching). Orientation, networking and memory management can also be simulated.

However for our project, camera tracking is an essential element that cannot be simulated by iOS simulator. When we test and debug our work, we shall load the program onto the machine directly.



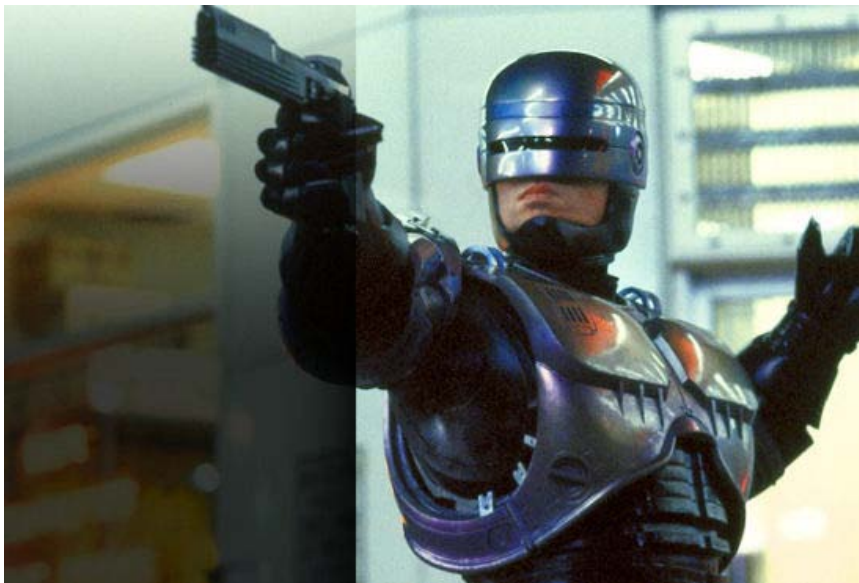
## Chapter 2. Augmented Reality

---

### 2.1 History

In 1966 Professor Ivan Sutherland of Electrical Engineering at Harvard University invented the first model of one of the most important devices used in AR today - the head-mounted display. Although the model was experimental and unusable, it was the first step in the development of AR.

Until 1999, AR remained unfamiliar to the consumer. It was because of the requirement of bulky, expensive equipment and complicated software. However, AR gained recognition in some sci-fi films. One instance was RoboCop (1987). The big helmet on Murphy's head gives a stream of data and information to enhance his vision and understanding of what's around.



**Fig. 2.1.1 RoboCop Movie Screen Shot**

In recent years, smart phones become so common and most are equipped with camera, GPS, digital compass, etc. Therefore, the role of AR in application becomes more significant. From geo-navigation to video gaming, AR enhances the user experience and realism.

## 2.2 Types of AR

### 2.2.1 Marker-less

Marker-less AR typically uses the GPS or digital compass feature of mobile devices to locate and interact with surroundings. Sometimes, camera is also used. AR information will be displayed on the video.



Fig. 2.2.1.1 Marker-less AR example

### 2.2.2 Marker-based

Marker-based AR typically uses the camera feature of mobile devices to analyze markers captured in video. QR codes are probably the most seen application of this type. Besides, the pose information of the marker may be useful sometimes. Users can move the device to view the virtual model in different angle.



Fig. 2.2.2.1 Marker-based AR example

## 2.3 Applications

There are some augmented reality apps for iPhone:

### 1. Geo-navigation e.g. SkyGlass

SkyGlass is an AR compass that display geo-information augmented to the real scene on the screen. It also makes use of GPS tracker,



Fig. 2.3.1 SkyGlass

### 2. Informative e.g. Wikitude, Layer

Wikitude is an application that when user view a location with the camera, additional and interactive information would be shown on the screen.

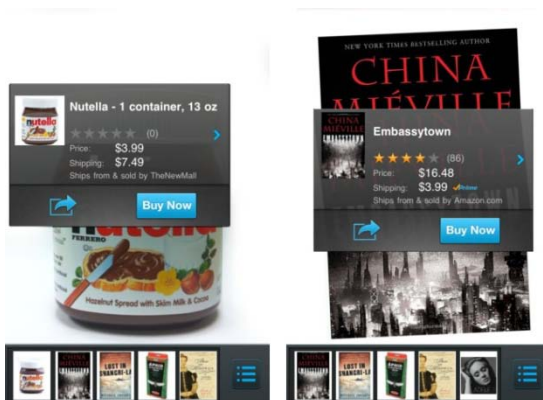


Fig. 2.3.2 Labeling information in Augmented Reality

### 3. Translation e.g. Word Lens

Word lens is an application used to translate words in the real world. The result will be displayed directly on the same place and same location on the screen.

What the user see is the language is translated into their preferred one.



Fig2.3.3 Word lens

#### 4. Sampler e.g. The Sampler by Converse

This kind of applications usually is developed by product resellers. Customers can view the virtual object on device. The device can be rotated and transposed by moving the real marker in front of the camera. These applications bring new experience to costumers and it is a quite attractive advertisement method.

## 2.4 AR in game industry

AR has its appearance on gaming consoles and mobile devices.

### 1. Consoles e.g EyePet

EyePet is a petting game on PS3. User can interact with the pet using marker and gesture. This is a quite new idea on existing game console platform.

### 2. AR Defender

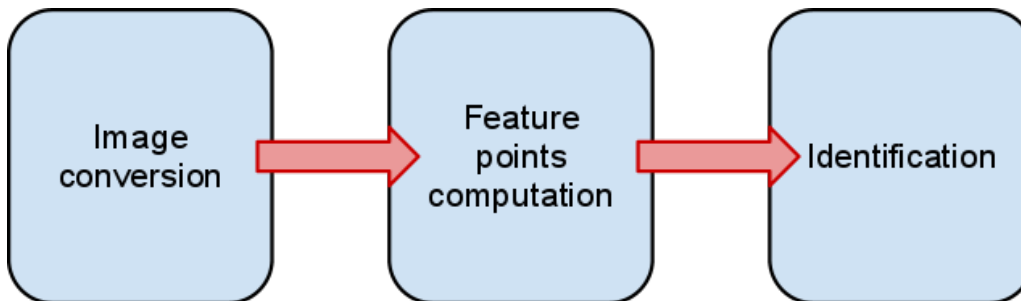
AR defender is a mobile AR game that involves tower and weapons. Tower defender is a quite popular game on different platforms. User experience is enhanced, user can move around with the device-in-hand as live viewer.



Fig2.4.1 AR defender game interface

## 2.5 Marker detection and recognition

Marker detection and recognition process involves three stages:



### 2.5.1 Image conversion

The first step is to convert the captured frame from colored into binary image. This process is called thresholding and it is the most basic form of image segmentation.

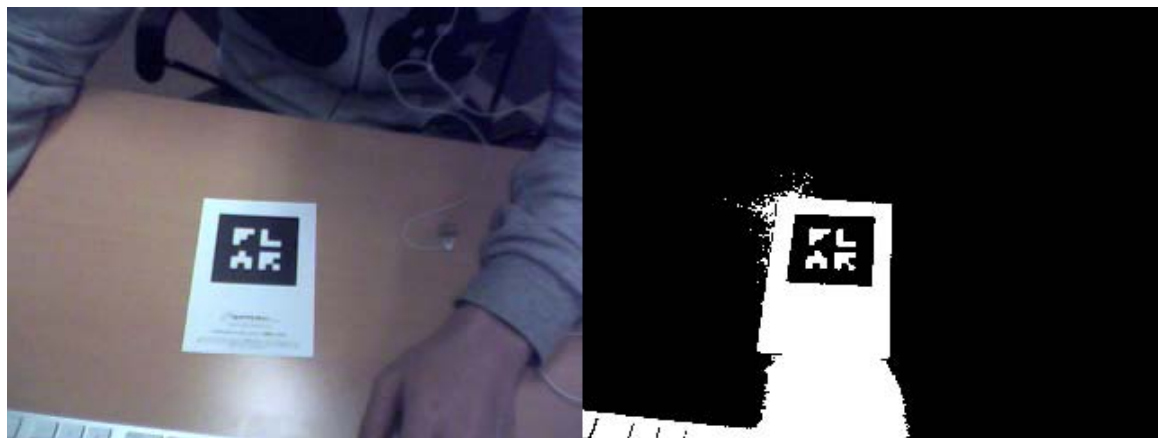


Fig2.5.1.1 Image after threshold processing

The most basic thresholding of image is to choose a fixed threshold value and then compare every pixel to that value. However, fixed thresholding is very much affected by the illumination changes in image or video stream.

In order to solve the problem caused by variations in illumination, adaptive thresholding is invented. The major difference between the two approach is that a different threshold value is computed for each pixel instead.

Adaptive thresholding is more robust in this way.

Since the process is repeated over and over while camera capturing video stream, the implementation should be kept as simple and fast as possible. The pseudo code below demonstrates adaptive thresholding for input image *in*, output binary image *out*, image width *w* and image height *h*.

```
function adaptive_threshold(in, out, w, h)
  for i = 0 to w do
    sum = 0
    for j = 0 to h do
      sum = sum + in[i, j]
      if i = 0 then
        intImg[i, j] = sum
      else
        intImg[i, j] = intImg[i - 1, j] + sum
      end if
    end for
  end for
  for i = 0 to w do
    for j = 0 to h do
      x1 = i - s/2
      x2 = i + s/2
      y1 = j - s/2
      y2 = j + s/2
      count = (x2 - x1) x (y2 - y1)
      sum = intImg[x2,y2] - intImg[x2,y1 - 1] -
            intImg[x1 - 1,y2] + intImg[x1 - 1,y1 -
            1]
      if (in[i, j] x count) ≤ (sum x (100 -t)/100)
        then
```

```

        out[i, j] = 0
    else
        out[i, j] = 255
    end if
end for
end for
end function

```

### 2.5.2 Feature points computation

The next step is to compute the feature points on binary image. The corners need to be detected accurately in order to have reliable camera pose estimation.

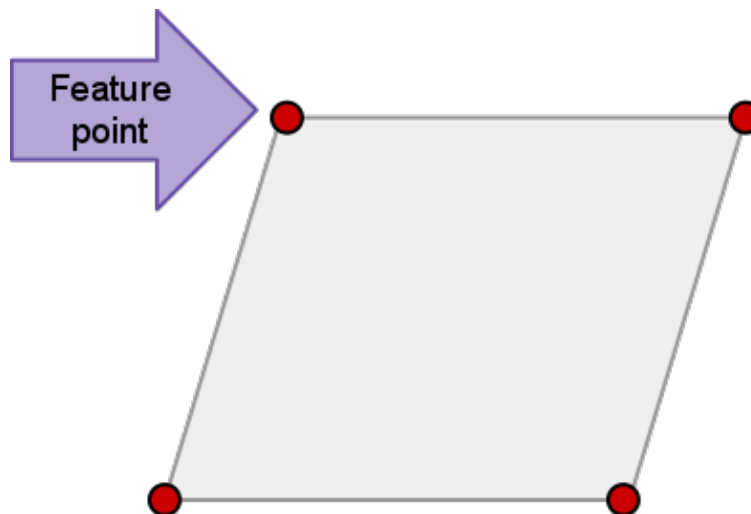


Fig. 2.5.2.1 Feature point example

### 2.5.3 Identification

The final step is to restore the effect of rotation, translation and perspective transformation by solving a simple linear system.

Suppose we have found the positions of four corners by feature points computation and the 3D coordinates in object space of the marker's corners are given by  $(x_i, y_i, 0)$  and the measured coordinates of the corners in the image are given by  $(X_i, Y_i)$ . The  $z$  coordinate is set to 0 because the marker is planar. All we need to compute are 9 parameters

for rotation matrix R and 3 parameters for translation vector T. For simplification, we can divide the numerator and denominator by tz and rearrange to obtain:

$$X_i = \frac{a_1 x_i + a_2 y_i + a_3}{a_7 x_i + a_8 y_i + 1}$$

$$Y_i = \frac{a_4 x_i + a_5 y_i + a_6}{a_7 x_i + a_8 y_i + 1}$$

Hence we can calculate for only 8 elements. Given the 3D coordinates and 2D image coordinates, we solve the linear system below:

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -X_1 x_1 & -X_1 y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -X_2 x_2 & -X_2 y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -X_3 x_3 & -X_3 y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -X_4 x_4 & -X_4 y_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -Y_1 x_1 & -Y_1 y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -Y_2 x_2 & -Y_2 y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -Y_3 x_3 & -Y_3 y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -Y_4 x_4 & -Y_4 y_4 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix}$$

The result vector of the linear system implies a normalized marker. By using the result vector, the system can provide users the pose information for drawing virtual 3D objects.



## Chapter 3 Qualcomm AR SDK

### 3.1 Introduction

Qualcomm AR SDK **Vuforia** (QCAR SDK in short) fetches live streaming from the device camera. It then analyzes the video by marker detection and provides the 3D spatial information of the detected marker via API. Programmers can use such information to draw appropriate virtual 3D objects on the camera video. As a result, the virtual objects are blended into real footage in real-time.



Fig. 3.1.1 A virtual 3D car is superimposed on top of live camera preview

Below is the overview of application development with the Qualcomm AR Platform. The platform consists of two components:

#### 1. Target Management System (hosted on QDevNet <http://ar.qualcomm.at>)

Allows developers to upload input image for the markers to be tracked and then download the corresponding target resources.

## 2. QCAR SDK Vuforia

Provides developers to link their application to the static library i.e. libQCAR.a on iOS or libQCAR.so on Android.

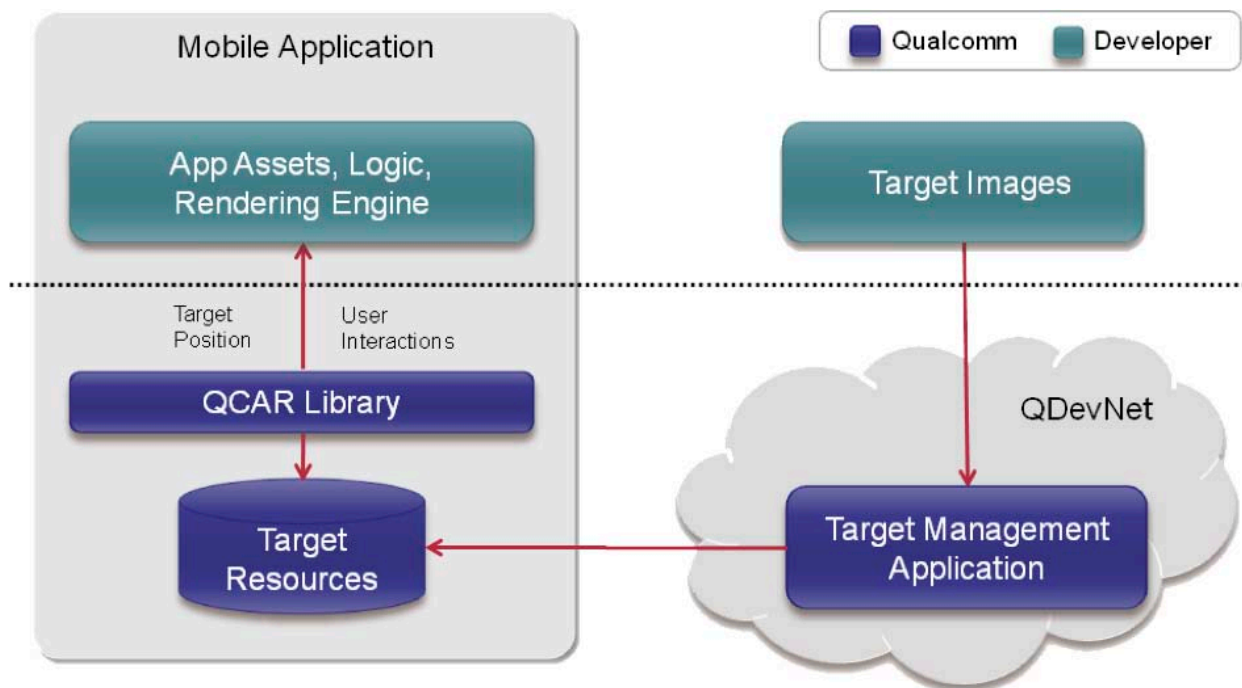


Fig. 3.1.2 QCAR SDK library

## 3.2 System architecture

The following are the core components of a QCAR-based application:

### Camera

The camera class is a singleton. It gives captured camera frames to the tracker for marker detection and recognition. Developers can decide when to start and stop the camera capture.

### Image Converter

The image converter class is a singleton. It converts captured camera frames from the camera format YUV12 to the RGB565 format for OpenGL ES rendering and the luminance format for marker tracking.

### **Tracker**

The tracker class is a singleton. It uses the computer vision algorithms to detect and track real world objects in captured camera frames. The target objects are evaluated and the results are stored in a state object that is accessible from application code.

### **Renderer**

The renderer class is a singleton. It renders captured camera frame to the video background. Its performance is optimized for specific devices.

### **Application Code**

The application code must involve initialization of all the above components. While the state object is updated for each processed frame, the application code should also update the virtual object location.

### **Target Resources**

Target resources are generated by the Target Management System. The output files from the system is a binary file storing the marker features and an XML configuration file. They are bundled in the application.

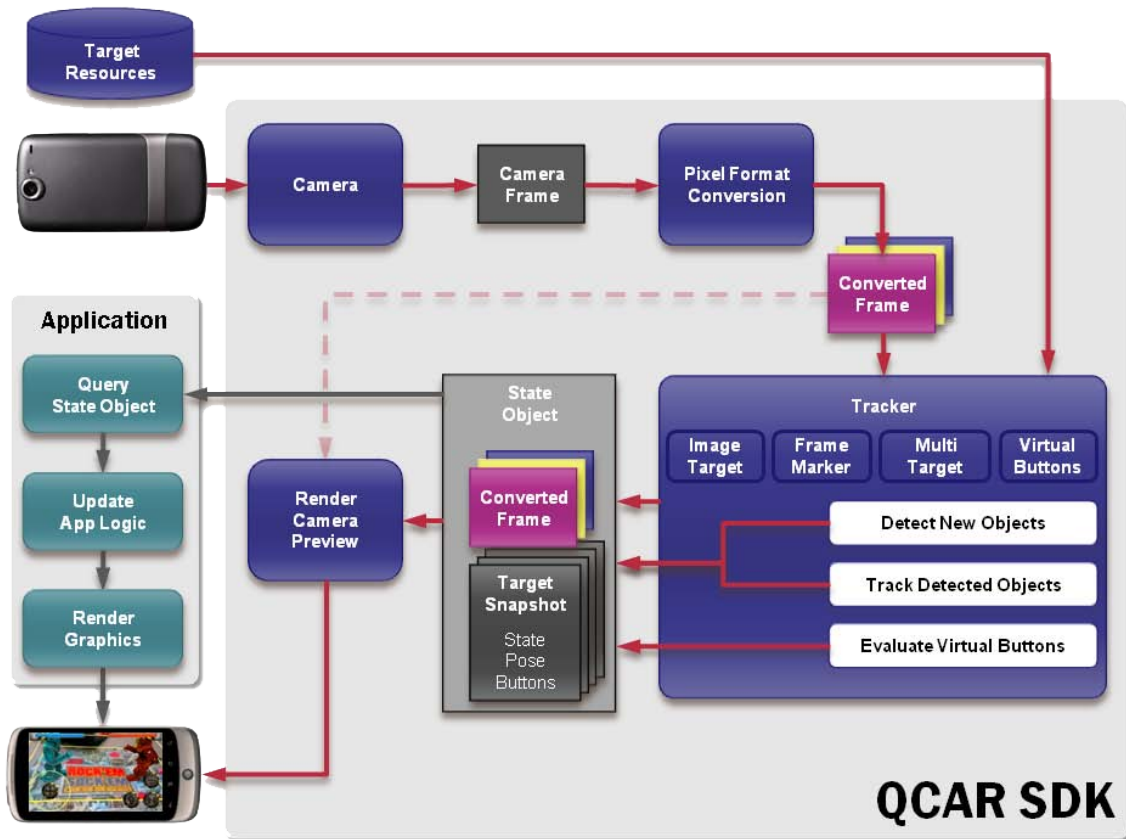


Fig. 3.2.1 QCAR SDK System Architecture Overview

### 3.3 Trackables



Fig 3.3.1 QCAR Trackable Marker Samples

i.Digi.T.able - Digital Interactive Game Interface Table Apps for iPad

## Definition

A trackable is any real world object that the QCAR SDK can track in six degrees-of-freedom. A trackable has a name, an ID, status and pose information which are stored in the state object. Image Targets is one kind of trackables. In the following paragraph, we will only discuss Image Targets.

## Parameters

### 1. Trackable type

**UNKNOWN\_TYPE**: an unknown trackable

**IMAGE\_TARGET**: an Image Target trackable

**MULTI\_TARGET**: a MultiTarget trackable

**MARKER**: a Marker trackable

### 2. Trackable name

A string which uniquely identifies the trackable from the database of targets. It has 64 characters in maximum and only contains a-z, A-Z, 0-9, [-\_.]

### 3. Trackable status

**UNKNOWN**: the state of the trackable is unknown. Usually returned before tracker initialization.

**UNDEFINED**: the state of the trackable is not defined.

**NOT\_FOUND**: the trackable is not found in the database of targets

**DETECTED**: the trackable is detected in this frame

**TRACKED**: the trackable is tracked in this frame

### 4. Trackable pose

A 3x4 matrix in row-major order which represents the pose information of detected or tracked trackable.

## Coordinate Systems

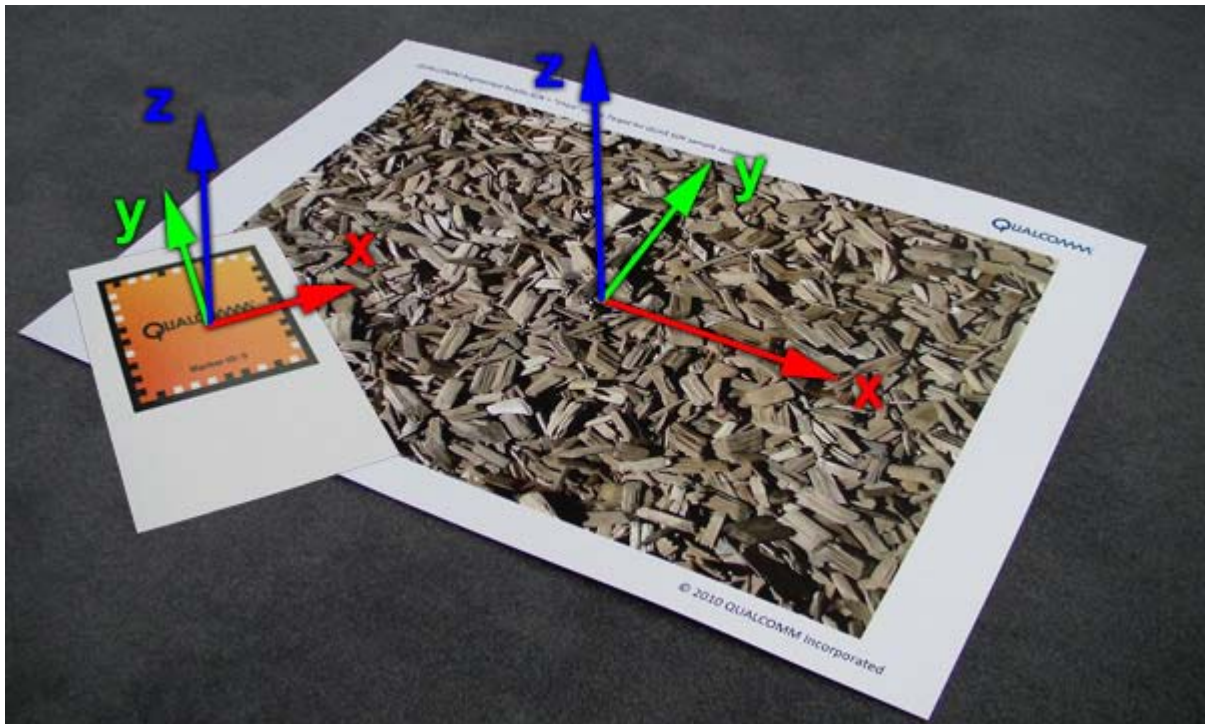


Fig. 3.3.2 In the QCAR SDK, right-handed coordinate system is used.

### Relevant API calls

|                                  |  |
|----------------------------------|--|
| virtual <b>TYPE</b>              | <b>getType</b> () const =0<br>Returns the type of 3D object (e.g. MARKER)                              |
| virtual bool                     | <b>isOfType</b> (TYPE type) const =0<br>Returns true if the object is of or derived of the given type. |
| virtual <b>STATUS</b>            | <b>getStatus</b> () const =0<br>Returns the tracking status.   |
| virtual int                      | <b>getId</b> () const =0<br>Returns a unique id for all 3D trackable objects.                          |
| virtual const char *             | <b>getName</b> () const =0<br>Returns the Trackable's name.  |
| virtual const <b>Matrix34F</b> & | <b>getPose</b> () const =0<br>Returns the current pose matrix in row-major order.                      |

|                          |   |
|--------------------------|---|
| <b>Frame</b>             | <b>getFrame ()</b> const<br>Returns the <b>Frame</b> object that is stored in the <b>State</b> .                            |
| <b>int</b>               | <b>getNumTrackables ()</b> const<br>Returns the number of <b>Trackable</b> objects currently known to the SDK.              |
| <b>const Trackable *</b> | <b>getTrackable (int idx)</b> const<br>Provides access to a specific <b>Trackable</b> .                                     |
| <b>int</b>               | <b>getNumActiveTrackables ()</b> const<br>Returns the number of <b>Trackable</b> objects currently being tracked.           |
| <b>const Trackable *</b> | <b>getActiveTrackable (int idx)</b> const<br>Provides access to a specific <b>Trackable</b> object currently being tracked. |

### 3.4 Target management system

The Qualcomm Target Management System allows developers to upload input image and generate feature dataset as target resources. Compiled with the target resources, the application can match images in frame against the feature dataset. To access the system, developer account is required.

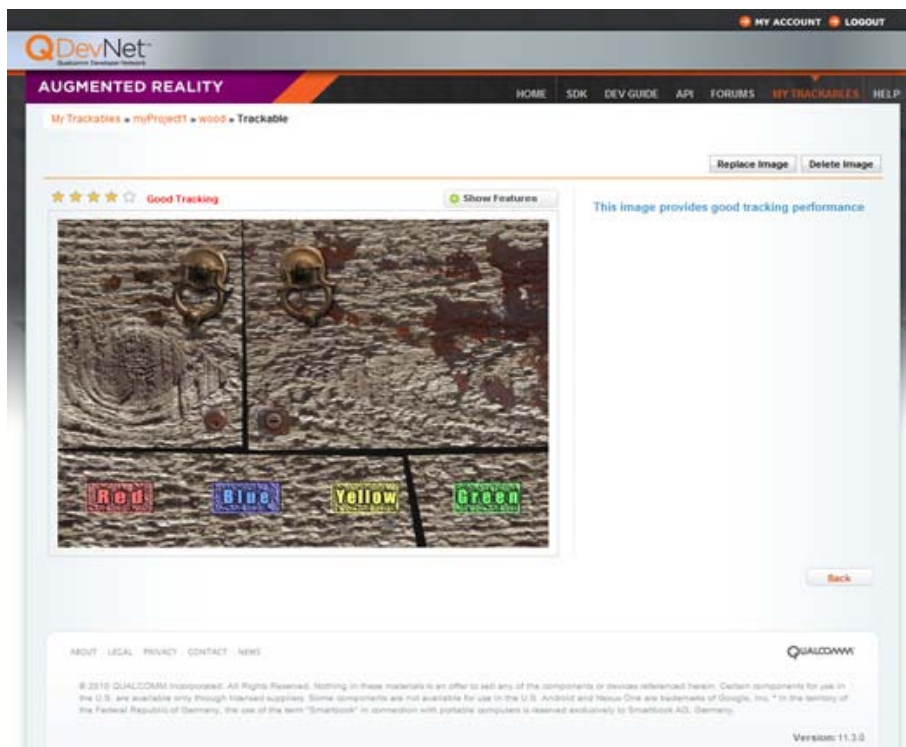


Fig. 3.4.1 Marker management interface

## Workspace and Projects

Once the user is logged in, the server will show the workspace. The page has all of the user's projects. Given an input image, a target is computed by processing its natural features. The feature sets used in the runtime application can have more than one target. Projects having a set of targets can be combined to create target resources for download. Only one target resource file is accepted in the runtime application and it can have multiple targets to be detected and tracked by the QCAR SDK. Typically, a new project is created for a new application. A number of images will be uploaded to compute the target data sets. Once the images to be included in target resource are decided, the user can pick those targets and download the dataset of merged natural feature.

### Target Resources

An XML configuration file is included in the target resource. It allows to configure certain trackable features and a binary file which has the trackable database. The user can download the target resource file which is named by the project.

## Sample Targets

The SDK contains input images for the ImageTargets sample applications. The official suggests developers to start by testing the target creation process with these images before uploading their own. In the paragraph below, chips.jpg and stones.jpg will be used for a more detailed explanation of this process. These images are located in:

### ANDROID:

```
<DEVELOPMENT_ROOT>\qcar-sdk-xx-yy-zz\samples\ImageTargets\media
```

### iOS:

```
<DEVELOPMENT_ROOT>\qcar-sdk-ios-xx-yy-zz\samples\ImageTargets\media
```



### How to create an Image Target

First, choose “Create a trackable” and choose “Single Image” as Trackable Type. Input a name for the result target. The name assigned will be used in the application to identify the target during marker detection and tracking. For using chips.jpg and stones.jpg, enter the trackable names “chips” and “stones” respectively.



Fig.3.4.2 The dialog for creating trackable

The width value refers to the printed size of the trackable in millimeters. For instance, the above input will generate a trackable with width 247 millimeters.

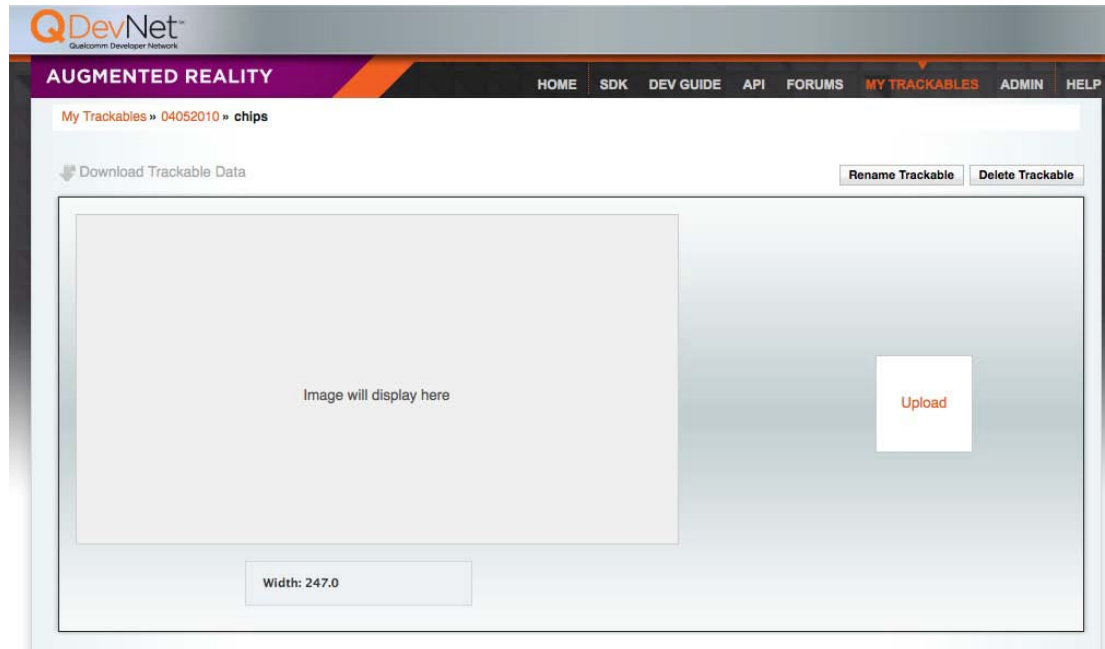


Fig.3.4.3 Image Target trackable page

This step is to upload image for marker. Select the “Upload” on the right and select appropriate image. The upload will start the target creation process. After completing the process, a thumbnail of the uploaded image will be shown with rating on a five-star scale. The rating implies the quality of the target for detection and tracking.

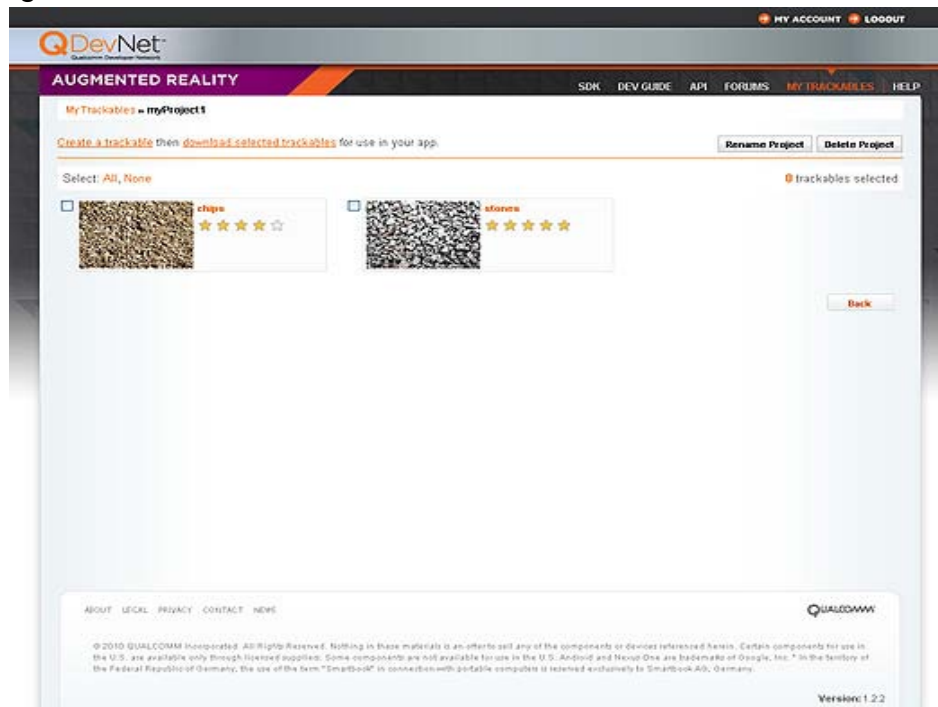


Fig. 3.4.4 A Projects page showing two uploaded targets

When the above steps are done, the developer can download the trackable by choosing “Download Trackable Data”. Alternatively, he can go back to the projects page and click “Download the Target Resources”.

### Chose of input images

There are some notes on choosing image to be high-quality marker

#### 1. Rich in detail

( for example, sport scenes, street scenes or a mixture of items)

#### 2. Good in contrast

( for example, both bright and dark regions are present on the image )

#### 3. No repeatitive patterns

( for example, a checkerboard should not be used )

The image needs to be printed in high resolution ( more than 200 to 300 dpi).



Fig. 3.4.5 High resolution printed image for creating target resource pipeline

To create target, the original image should be down-scaled to a resolution similar to the live preview camera resolution on the mobile device. The aspect ratio of

the target image must kept the same on printed. Only 8- or 24-bit PNG and JPG formats and less than 2MB size images are acceptable.

### 3.5 Development on iOS

The Qualcomm AR SDK has been tested successfully on Mac OS X 10.6 Snow Leopard and 10.7 Lion. The downloaded installer has the following result directory:

|  |  |
|--|--|
| <code>&lt;DEVELOPMENT_ROOT&gt;/</code> |  |
| <code>qcar-ios-xx-yy-zz/</code>        |  |
| <code>build/</code>                    | QUALCOMM Augmented Reality SDK   |
| <code>include/</code>                  | Commented header files   |
| <code>lib/</code>                      | Static link libraries  |
| <code>licenses/</code>                 | License Agreements   |
| <code>samples/</code>                  | Sample applications with full source code  |
| <code>Dominoes/</code>                 | Dominoes game featuring dynamic virtual buttons, sound and touch screen interactions |
| <code>ImageTargets/</code>             | Sample app that tracks two Image Targets   |
| <code>FrameMarkers/</code>             | Sample app that tracks multiple Markers  |
| <code>MultiTargets/</code>             | Sample app that tracks a Multi Target  |
| <code>VirtualButtons/</code>           | Sample app that shows Virtual Button interactions                                    |
| <code>assets/</code>                   | Additional assets for the QCAR SDK   |
| <code>readme.txt</code>                | Starting read-me document  |

The SDK is separated from developer applications to ensure easier updates to the SDK while not affecting the application files.

## 3.6 Compare with String AR

We want to compare Qualcomm AR SDK with String AR in terms of development, performance and licensing.

### Portability

Both Qualcomm AR SDK and String AR are available on iOS. They can be easily integrated to iOS projects with a few lines of code and proper configuration for linking library files. However, Qualcomm AR SDK does not provide project templates and new developers need to start from the sample code.

Regarding the Android platform, only Qualcomm AR SDK has provided complete support to developers. In fact, the SDK begins at Android earlier than iOS. On the other hand, the Android version of String AR is still in progress of development and unavailable to developers. Therefore, the Qualcomm AR SDK seems to be a better option when considering porting our application to Android platform in the future.

### Flexibility

The Qualcomm AR SDK has an object-oriented structure. Most of its core classes are singletons and some creates their own threads. The classes are also interdependent. For example, the tracker class depends on the renderer class. The strong coupling between components limits the freedom of developers to do logic and integrate with other system such as rendering engines. We think that the Qualcomm AR SDK is not flexible enough.

Since we do not have any experience with the String AR SDK, we have no comment on the API's flexibility.

### Documentation and reference

The Qualcomm AR SDK provides online documentation at <https://ar.qualcomm.at/qdevnet/api>. Its search function is very easy to use. All

functions are properly described in text. Besides, sample projects are offered in the download files of the SDK. Each sample project provides very good beginning of using different features of the system. We appreciate the effort made by Qualcomm officials for such complete, informative documentation and reference.

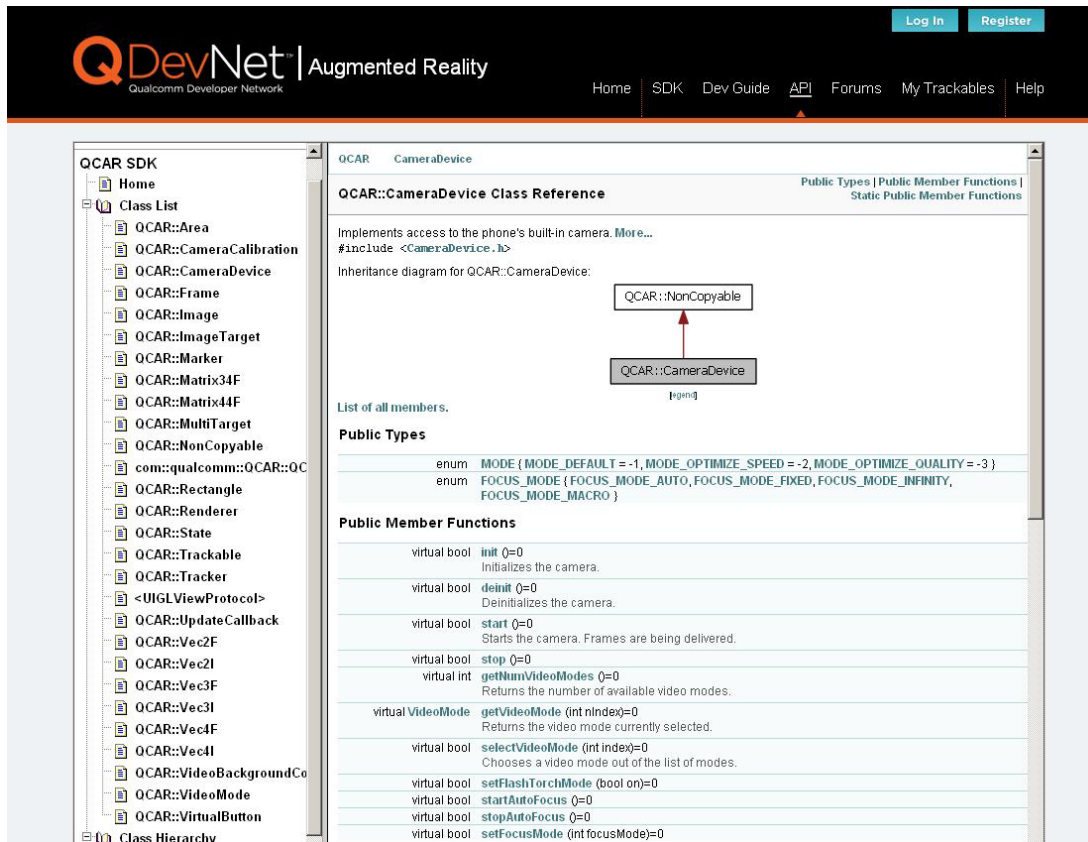


Fig. 3.6.1 QCAR SDK class reference

The String AR SDK has no online documentation but some reference PDF files in the download files. Sample projects are also provided. However, compared with the Qualcomm AR SDK, its documentation needs to improve.

## Community

The community of Qualcomm AR is open to public and growing.

The following is the statistics about its online forum  
(<https://ar.qualcomm.at/qdevnet/forums>):

|                       |
|-----------------------|
| Threads: 1,171        |
| Posts: 5,267          |
| Members: 15,017       |
| Active Members: 1,704 |

Their moderators are quite responsible and quick in responding any questions related to the SDK.

The String AR seems to not have a public online area for user feedback or questions since we cannot find any linkage on their homepage.

### **Performance**

For tracking a single marker (which is enough for our project), both Qualcomm AR and String AR show promising results in performance. The programs response instantly and smoothly to marker detection regardless how complex the rendering objects are. The frame rate is always kept at 60 fps.

### **Licensing**

The Qualcomm AR SDK is free for development and distribution. Therefore, it is very suitable for doing research.

The String AR SDK, however, is only free for limited use. The limited version is for demo and only one marker is trackable. Also, release on App Store is not allowed. The minimum spending of String AR developer plans is USD \$99 which we think too expensive for our project. As there is a free option provided by Qualcomm, that is the main reason why we have chosen **Vuforia** instead of String.

## Chapter 4. Networking

---

### 4.1 Connection

As the objective of our project is to create a same virtual space for both users on different iPad, clients have to exchange data through network connection. During our early phase, we investigated a few methods for implementing the network connection part. This section covers our findings and justification.

#### 4.1.1 Network Socket

Network Socket is based on Internet-protocol to let computers communicate and exchange data via a connected network, for example the Internet. In such implementation, we need a known IP address and a designated port number. Clients can connect to the server by setting up a socket connection.

There are different types of sockets commonly used:

**Datagram sockets**, which does not require a connection before sending data. This kind of socket uses User datagram Protocol, in real life, DNS and many online game uses UDP to transfer data.

**Stream sockets**, the most famous protocol stream sockets implement on is TCP. Data should be sent after connection is established. Video service providers have well-population of using such technique to stream videos over network.

**Raw sockets**, it is used by the router to pass though raw packets. ICMP (Internet Control Message Protocol)

Implementing data exchange by socket is a very efficient way in terms of data transfer. However it would be more difficult to set up.



## 4.1.2 HTTP Request

HTTP Request is based on the Representational state transfer (REST) architecture, it is a commonly used software architecture for distributed hypermedia systems. A typical RESTful architecture consists of clients and a server. A request is sent from the client side, and the server receives and processes the request. Response is then replied from the server to the client.

The REST Architecture has six main constraints:

### **Client-server**

This model separates client and server apart. They can be replaced independently since the server concerns with data storage and management and client side concerns interface only.

### **Stateless**

For each request to the server, there is no state different. The context or state of client will not be stored in the server.

### **Cacheable**

Clients can cache response therefore reducing unnecessary bandwidth.

### **Layered system**

Components in the system are grouped in a hierarchical arrangement. Lower layers provide functions and service to higher layers. The layers are there to enforce security policies.

### **Code on demand(optional)**

Servers are able to transfer executable code to client on demand, such as Javascript code or Java applets.

### **Uniform interface**

There is a uniform interface for messages to be sent over the network. For example the server does not sends the database information in raw format, but rather in XML or JSON.

By using HTTP Request, we transfer data through HTTP (Hypertext Transfer Protocol). Each time, client request the server for updates, client also sends server the most updated information (such as user input, location moved). Server receives client information and log them down, then response with the updated information from another client.

### 4.1.3 Game center



Game Center is an online multi-player social gaming network released by Apple.

The main functionalities of the Game Center are:

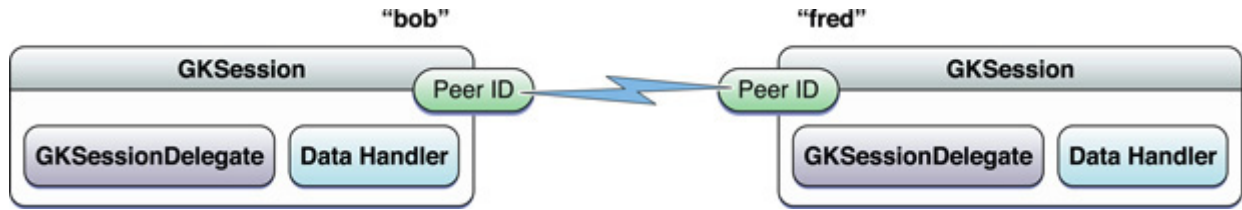
- Users can invite friends to play a game
- Match online with another user via game center
- Authentication
- Achievement tracking
- Leaderboards

Game Center supports iOS version 4.1 or above.

There is no restriction on data format as stated in the Apple developer's note.

### 4.1.4 Peer to Peer

iOS gamekit provides a few protocol for iOS device to connect with each other locally. For example, the ad-hoc connection between two peer device via local wireless network. Sessions are created and disclosed, then the devices are connected to the network. Data can be sent through the data channel.



However, peer picker controller only creates Bluetooth or Local wireless connections. However, the Bluetooth is only a short-distance wireless connection protocol, with the maximum range around 100m. We prefer a connection method supports a larger range.

## 4.2 Data exchange

In this part, we are going to introduce what data is going to be transmitted in order to be effective.

### 4.2.1 JSON

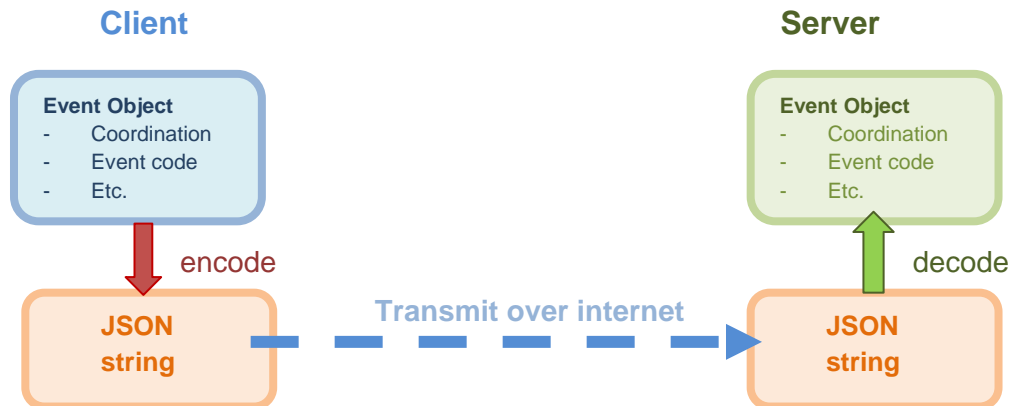


JSON , abbreviation of JavaScript Object Notation is a lightweight data-interchange format. The most remarkable advantage is its high writability and readability. A JSON string is derived from a JavaScript language for representing simple data structures and associative arrays. It is language independent and there are quite a lot of parsers to process them in different programming languages such as C, ASP, Python, PHP, Java, Perl, Objective C etc.

To represent data efficiently, we encode our data into a JSON string and send the encoded string over the Internet. The server receives and reconstructs it into object with the JSON parser. That is one of the requirements of REST Architecture – Uniform Interface.

Another format commonly used in the past (and even now) is XML. It constructs a hierarchy data structure with tags. However, we consider that JSON is more compact in format and more user-friendly.

How JSON is used in our system



For object **Event**, it includes entities "clientToken", "eventId", "eventType", "coordination". "coordination" has three sub-fields "x", "y" and "z". The event object is created by the client iPad after real-time calculation. The following fragment shows a sample JSON representation of data to be transmitted.

```
{
  "clientToken": "btfpm7d3qj7pagirfarvur64b5lk56",
  "eventId": "2",
  "eventType": 3,
  "coordination": {
    "x": 20,
    "y": 30,
    "z": 0
  }
}
```

Values with "" quotes are expected to be String type and otherwise as generic types such as numbers or Boolean values.

## Chapter 5. Design and Implementation

### 5.1 Idea

Our idea is to construct a virtual space for two connected players to interact with each other. The game is played like dodge ball. To shoot a ball, the player has to tap the screen. The player needs to move left and right in order to dodge balls thrown from the opposite side.

First, both users need a trackable marker to start the game. User should use the front camera to track the position of the marker. The program will analyze data and generate client's relative position. So that user can view the virtual world in the iPad display screen directly in a first-person point of view.

After the relative position of a client is generated, client would communicate with the server. Position information is updated to the remote server during the communication. At the same time, server would notify the client if there is any update from another client.



**Fig. 5.1.1** Implementation of position tracking

## 5.2 Settings

The system contains client side and server side. In terms of implementation, we sub-divide the components of this project into three main parts:

### 1. Marker tracking module

To analysis camera input, to identify and track the camera movement base on a marker. Hence calculate the position and movement data for other modules.

### 2. Network connection module

This module provide interface for other modules to communicate with e server. Data exchange via the network is through the network connection module.

### 3. Virtual world construction module

The virtual world construction module visualizes data and position information generated by other modules. It is responsible for the computation of the screen according to point of view.

### 4. Game engine

The game engine includes the AR SDK part and game rules computation module. It reads inputs from other modules and processes it. Result will be output to Virtual world construction module to display.

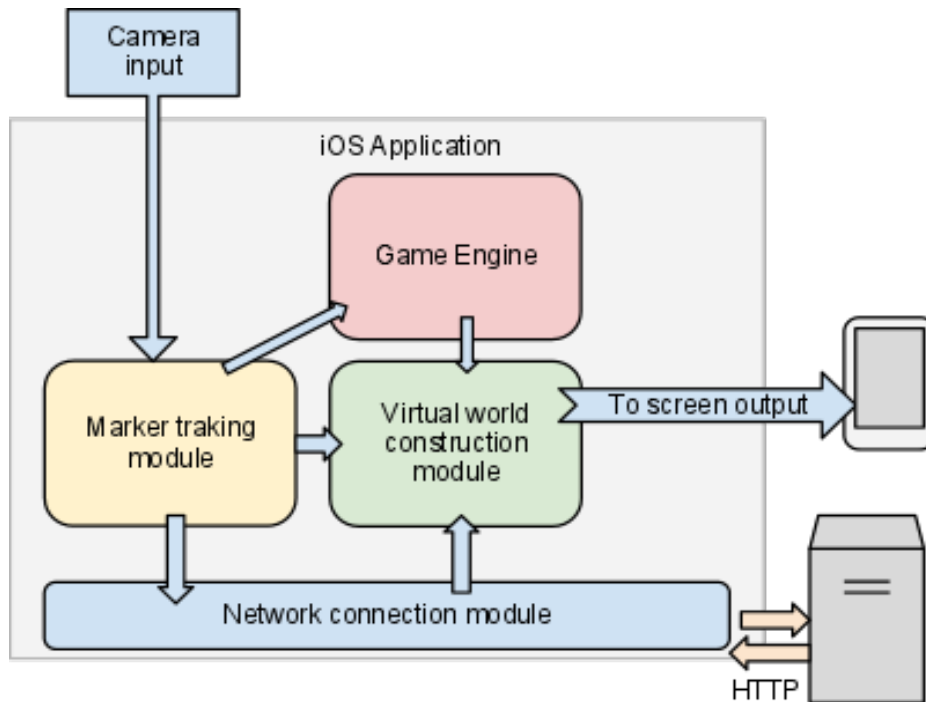


Fig. 5.2.1 Overview of the system design

## 5.3 Modules

### 5.3.1 Marker Tracking Module

The Marker Tracking Module is the agent that directly communicates with the Qualcomm AR SDK. It is a finite state machine with the following status:

**APPSTATUS\_UNINITED:** the application and the QCAR component is uninitialized

**APPSTATUS\_INIT\_APP:** the application is initialized

**APPSTATUS\_INIT\_QCAR:** the QCAR component is initialized

**APPSTATUS\_INIT\_APP\_AR:** the video renderer is initialized

**APPSTATUS\_INIT\_TRACKER:** the tracker and target resources are initialized

**APPSTATUS\_INITED:** the QCAR component is started

**APPSTATUS\_CAMERA\_STOPPED:** the camera is stopped capturing

**APPSTATUS\_CAMERA\_RUNNING:** the camera is capturing

**APPSTATUS\_ERROR:** error occurs

Every time when the state is changed, the function `updateApplicationStatus:(status)newStatus` is called. The parameter `newStatus` is checked and do suitable actions.

When all initialization works are done, the QCAR SDK will call the method `renderFrameQCAR` on a single background thread. The function is implemented by the user like below:

```
- (void)renderFrameQCAR
{
    [self setFramebuffer];

    // Clear colour and depth buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render video background and retrieve tracking state
    QCAR::State state = QCAR::Renderer::getInstance().begin();

    if (QCAR::GL_11 & ARData.QCARFlags) {
        glEnable(GL_TEXTURE_2D);
        glDisable(GL_LIGHTING);
        glEnableClientState(GL_VERTEX_ARRAY);
        glEnableClientState(GL_NORMAL_ARRAY);
        glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    }

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    ImageTargetsAppDelegate* delegate = (ImageTargetsAppDelegate
*)[[UIApplication sharedApplication] delegate];

    for (int i = 0; i < state.getNumActiveTrackables(); ++i) {
        // Get the trackable
        const QCAR::Trackable* trackable =
state.getActiveTrackable(i);
        QCAR::Matrix44F modelViewMatrix =
QCAR::Tool::convertPose2GLMatrix(trackable->getPose());

        if ( RECSTATUS_RECORDING == delegate.recStatus ) {
            QCAR::Matrix44F invModelViewMatrix;
            ShaderUtils::invertMatrix(&modelViewMatrix.data[0],
&invModelViewMatrix.data[0]);
            // Get the camera's position from the inverse matrix
```



```

        NSArray* camPos = [[NSArray alloc]
initWithObjects:[NSNumber
numberWithFloat:invModelViewMatrix.data[12]], [NSNumber
numberWithFloat:invModelViewMatrix.data[13]], [NSNumber
numberWithFloat:invModelViewMatrix.data[14]], nil];
        [delegate.objStates addObject:(id)camPos];
        break;
    } else if ( RECSTATUS_PLAYING == delegate.recStatus ||
RECSTATUS_PAUSED == delegate.recStatus ) {
        NSArray* tmpPos = [delegate.objStates
objectAtIndex:delegate.currentFrame];
        float x = [[tmpPos objectAtIndex:0] floatValue];
        float y = [[tmpPos objectAtIndex:1] floatValue];
        float z = [[tmpPos objectAtIndex:2] floatValue];

        // Choose the texture based on the target name
        int textureIndex = (!strcmp(trackable->getName(),
"stones")) ? 0 : 1;
        const Texture* const thisTexture = [ARData.textures
objectAtIndex:textureIndex];

        // Render using the appropriate version of OpenGL
        if (QCAR::GL_11 & ARData.QCARFlags) {
            // Load the projection matrix
            glMatrixMode(GL_PROJECTION);
            glLoadMatrixf(projectionMatrix.data);

            // Load the model-view matrix
            glMatrixMode(GL_MODELVIEW);
            glLoadMatrixf(modelViewMatrix.data);
            glTranslatef(x, y, z - kObjectScale);
            glScalef(kObjectScale/2, kObjectScale/2,
kObjectScale/2);

            // Draw object
            glBindTexture(GL_TEXTURE_2D, [thisTexture
textureID]);
            glTexCoordPointer(2, GL_FLOAT, 0, (const
GLvoid*)&teapotTexCoords[0]);
            glVertexPointer(3, GL_FLOAT, 0, (const GLvoid*)
&teapotVertices[0]);
            glNormalPointer(GL_FLOAT, 0, (const
GLvoid*)&teapotNormals[0]);
            glDrawElements(GL_TRIANGLES,
NUM_TEAPOT_OBJECT_INDEX, GL_UNSIGNED_SHORT, (const
GLvoid*)&teapotIndices[0]);
        }
        if ( RECSTATUS_PLAYING == delegate.recStatus ) {
            if ( delegate.currentFrame <
delegate.objStates.count - 1 ) {

```

```
        delegate.currentFrame++;
    }
}

glDisable(GL_DEPTH_TEST);
glDisable(GL_CULL_FACE);

if (QCAR::GL_11 & ARData.QCARFlags) {
    glDisable(GL_TEXTURE_2D);
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
}

QCAR::Renderer::getInstance().end();
[self presentFramebuffer];
}
```

The most important part of the previous code is:

```
for (int i = 0; i < state.getNumActiveTrackables(); ++i) {
    const QCAR::Trackable* trackable =
state.getActiveTrackable(i);
    QCAR::Matrix44F modelViewMatrix =
QCAR::Tool::convertPose2GLMatrix(trackable->getPose());
}
```

The function call `state.getNumActiveTrackables()` returns the total number of markers detected and tracked in the current camera frame. In our case, it always returns at most 1 because we only want to track one marker. Then a reference to the trackable object is obtained from the state object. By using the helper functions provided in the SDK, we can get the pose information and draw virtual 3D objects based on them. The workflow is illustrated as below:

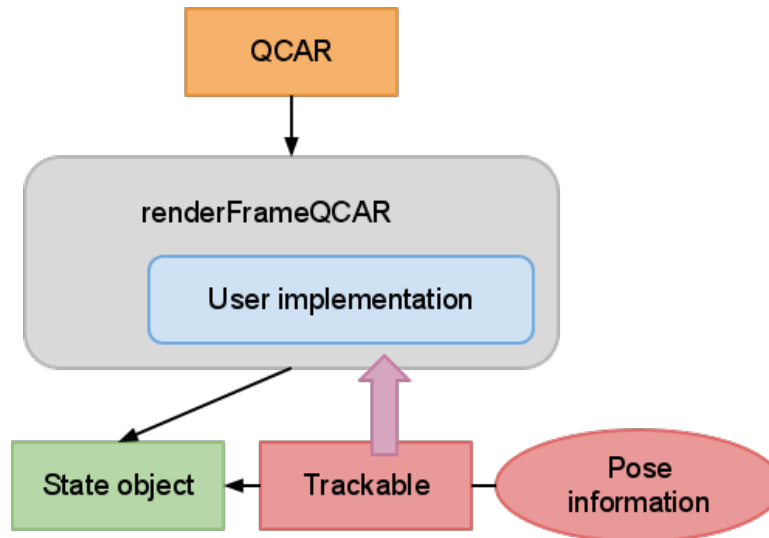


Fig. 5.3.1 Marker tracking based on QCAR SDK

### 5.3.2 Network Connection Module

We have discussed some prototypes about how should we exchange data via the Internet.

Here are the criteria we came up when we design the methodology for the network connection module.

#### 1. Data size

The data to be sent over the Internet should be kept in a smaller packet size. Only critical and important information should be exchanged. The format of how the data is represented is also a critical factor.

#### 2. Network load

As our system depends quite a lot on real time data exchange, the design of the network streaming protocol should minimize the network load in order to simulate a real time environment on different clients smoothly.

#### 3. Accessibility

The server should be accessible when needed. However, after we have investigate the servers that can be set up in the CSE department. Some

ports are not opened to outside world. For stability, we found hosting the server program on web server a fairly good choice.

### Connection Protocol

For the network connection methodology, we decided to set up a web server. The server acts as the middle-man and remembering information needed by the clients. The server is also responsible for recognizing clients and redirect correct update information for each side.

### Connection register phase

A client sends a request to register its identity the server.

Since the Server-client model is stateless, we have to make sure the server recognize different kinds (and possibly incorrect connections) correctly; a 32-digit random token string is generated when server replies the connection request initiated by client.

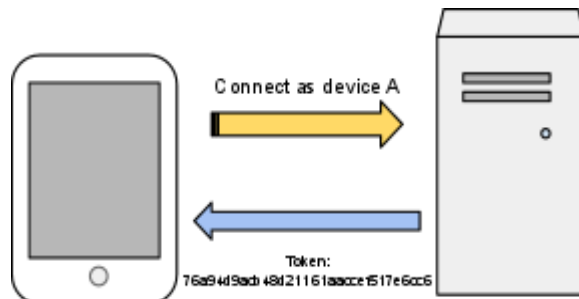


Fig.5.3.2.1 connection established

Server then saves down the token string as a connection reference.

### Data update phase

Client connects to server presenting the token previous assigned. Server checks if the token refers to a registered machine. If server could identify the request from client, it updates information in server and sends most updated data to client if it has been modified.

Since object detail (shapes, colours, etc.) has been stored locally on client machines, data of exact object image is not sent in order to minimize

network traffic. In our design, information is sent in a notification message form.

There are two types of information should be transmit to the server. Then the server will process them and forward to the designated receiving client.

Position information generally represents relative position of client, it should be updated frequently to achieve a real-time effect.

An Event update is generated on demand. For example, user touches the monitor, ball is hit, user enter/ quit game. Each event is sent with a predefined event type code. Server and the opposite client will decide what to do according to event received.

Summary of information to be transmitted

| <b>Types of information</b> | <b>Position</b>            | <b>Event</b>                                |
|-----------------------------|----------------------------|---|
| <b>Update frequency</b>     | On time Interval           | On change                                   |
| <b>Information included</b> | Client id,<br>coordination | Event id , type,<br>Coordination(if needed) |

### **Advantages**

The previous version we came up is the server distinguish clients depends on their IP address. However we found that it may not work all the time.

Some problems are:

1. Client IP may change due to dynamic IP allocation
2. Machine behind gateway may share a common IP address, so that server could not identify between the devices

Moreover, current method ensures connection session persists even after network re-connection.

This implementation is inspired by Session management on web browsers. Web browser holds a special session key for a particular website. When a web requires an identity authentication, client presents the key recorded in server database to finish the checking procedure. With this method of implementation, server can manage different client connection easily:

- Server can handle connections without knowing IP address
- Server can timeout expired key on demand

In the upgraded server side program, we have used SQLite to manage the data. A persistent file handler is opened to reference the database. It can minimize the I/O overhead in order to improve the performance.

### 5.3.3 Virtual world construction module - Graphics and UI



The graphics in the game is mainly created using OpenGL ES library. The OpenGL ES (OpenGL for Embedded Systems) edition is a subset of the OPENGL library, it is specially designed for embedded systems, such as phones, tablets etc.

OPENGL is a famous 2D/3D graphics library that can create computer graphics with easy logic commands. It provides a low-level applications programming interface (API) between software applications and hardware or software graphics engines.

OpenGL ES is a lighter version that removes *glBegin/glEnd* for primitive objects, *GL\_QUADS*, *GL\_POLYGONS* and other non-essential elements.

The main advantage of OpenGL ES is its low power consumption compared to the standard OPENGL.

Version we are currently using in our project is OpenGL ES 1.1.

The virtual world construction module is responsible for generating the virtual scenes on augmented to the reality. It takes input arguments from the Marker Tracking module, Network connection module and Game engine. Corresponding view is then generated according to the input.

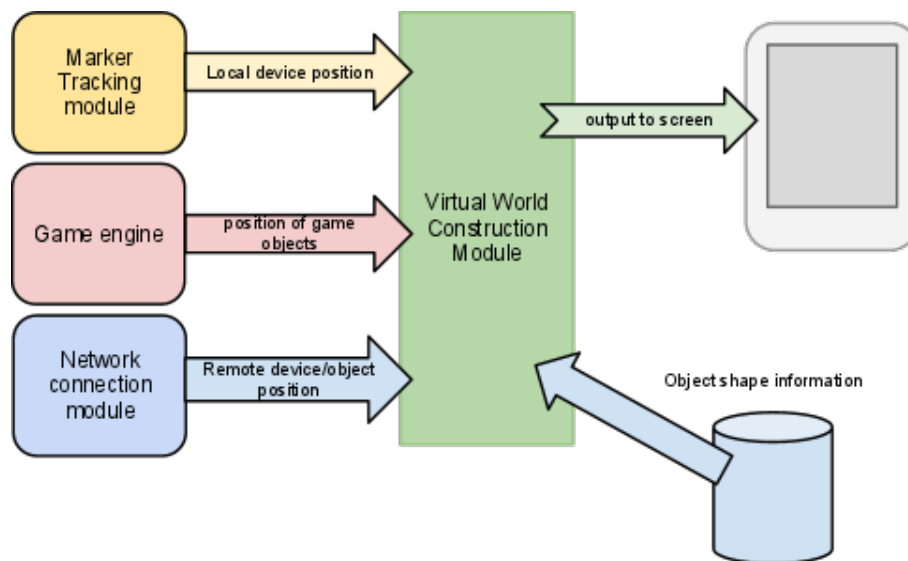


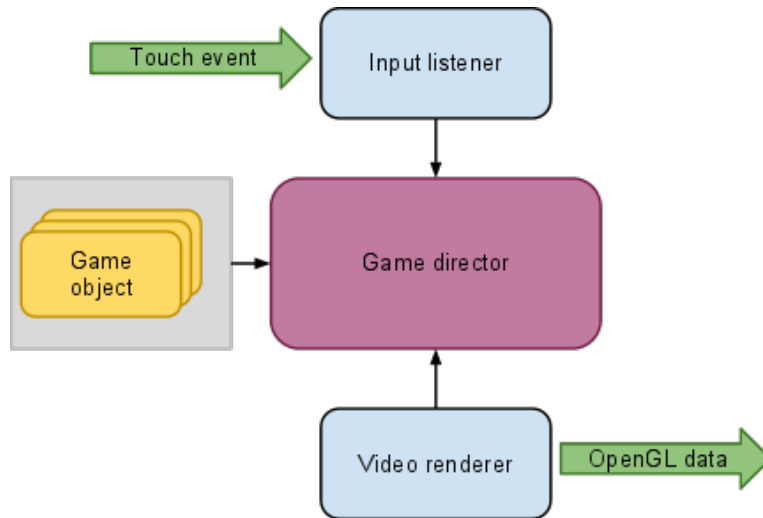
Fig 5.3.3.1 Input and output of the Virtual World Construction Module

### 5.3.4 Game Engine

The game engine computes game logic part. Rule and game method is defined. The game engine process inputs according to the rules.

The game engine's design is quite dependent on the game itself. For our sample game dodge ball, game engine takes consideration of user tap input and device position regarding to the marker.

This is the overview of the game engine:



**Fig 5.3.4.1 Game engine system overview**

The input listener is an abstract layer for catching user input to the device. It will do callback functions defined by the user when respective events have been fired. The event object has information about the event such as the position where the user tap on the screen and the gesture of user motion. In the callback functions, the user can reference the game director for game objects.

The design of the input listener process matches with that of iOS. A responder chain is created for event delivery.



The below is an example of responder chain:

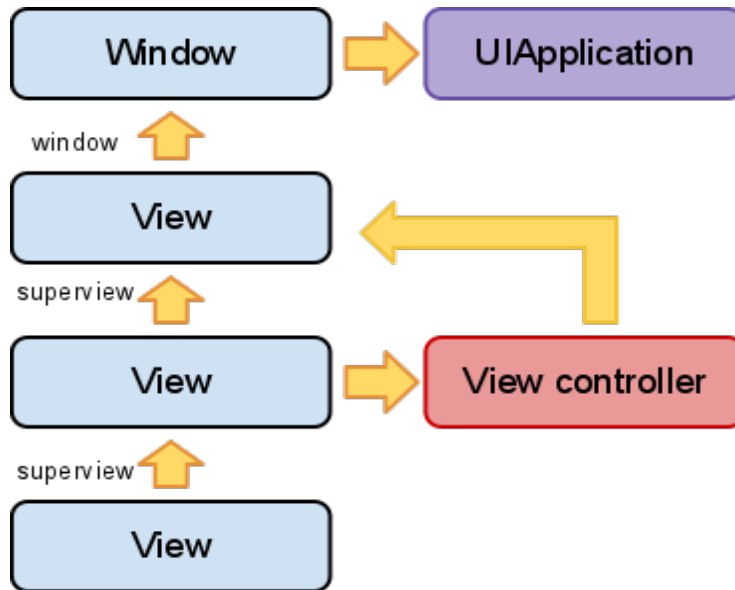
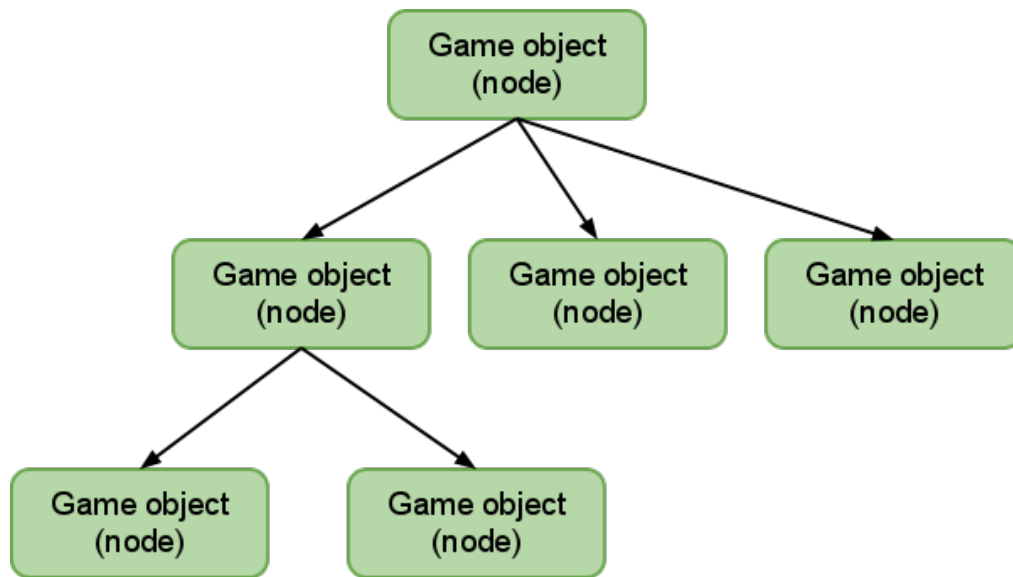


Fig 5.3.4.2 Responder chain in iOS

An event object is first delivery to the base view and then its inherited views.

The game director is a centralized class responsible for manage the game world. It is the agent for object creation and deletion. The game director provides a callback function to the user to write their own game logic. Before this callback, the director prepares everything such as initializing the video buffer. After this callback, the director finalize everything such as redrawing the updated video buffer. In the callback, the user can access the game object reference and get or update their information such as the x, y, z position in the virtual 3D world.

Every game object is treated as a node. Each node can have child nodes. The relation tree is like below:



**Fig 5.3.4.3 Sample relationship of game objects**

Operations upon a node will have same effect on its child nodes. Just like a robot can be a node and its body parts are the child nodes of itself.

When the robot moves, its body parts also move.

## 5.4 Game design

Here we try to illustrate the game that we implemented to demonstration AR effect. The game is a 2-player version Dodge ball game.

### Set Up

An A4 Marker should be set up on the wall. Each user is advised to stand at a place a few meters away the marker. The camera should be able to view the marker clearly.

Clients then should select which player they want to be. For example Player A. User then should use hold up the iPad pointing to the marker in order to set up the initial location.

If the connection is successful, each side should see another player in the virtual

world on screen. User can now try to move the iPad to around, see if the perspective of view changes according to the view.

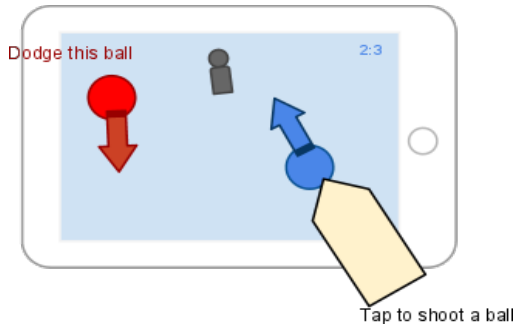


Fig. 5.4.1 Game interface illustration

### Start the game

After connection on both sides are established, user can then start the game. The game rule is simple, user can move around in the virtual world by moving iPad they hold. The aim for this game is to hit the opponent with ball as many time as you can; at the same time keep clear from balls thrown by remote user.

### Control

As mentioned, moving the device is equivalent to moving in the game. To throw a ball, just tap the screen, then a ball will be thrown to the other side.

### Winning criteria

Player A wins a game if:

*Number of times that Player B is hit by ball > Number of times that Player A is hit by ball*

The game ties if :

*Number of times that Player B is hit by ball = Number of times that Player A is hit by ball*

or

*No one is hit.*

## Chapter 6. Pong

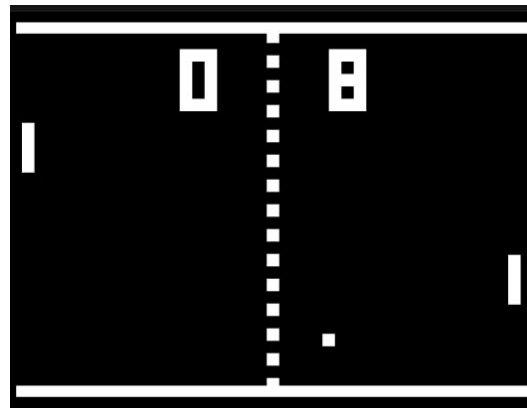
### 6.1 Background



Pong is a two-dimensional sports game played by two players against each other. It is similar to table-tennis although the game has a pretty much simpler rule.

Since released by Atari Inc. in 1972, it was one of the most popular video games once. It was also a commercial success and it and led to the development of consumer video games. Many companies started to copy its gameplay and produced new types of the game. Until now, the game has existence on many different platforms such as arcade machines, home consoles and mobiles.

The game is constructed by two paddles, one ball and two walls. In a new round, one player serves the ball and the opponent needs to return the ball by hitting it with his paddle. Otherwise, one point will be given to the player but serve the ball in the next round. The first player who scores 11 points will win the game.



In our semester two, we would like to incorporate augmented reality into Pong. Pong is good choice because its rule is simple and it requires player control which can be achieved by camera tracking.

## 6.2 Overview

This section describes how to add augmented reality element to Pong. First, we need to consider the requirement of playing Pong. To play Pong, there must be a two-dimensional square area. As the game arena, the players will focus on this area. Thus, we can add that arena in our augmented reality view by laying it on the marker.



Fig 6.2.1 iPad screen when marker detected

Second, we need to consider how the paddles are controlled. In Pong, the movement of paddles is very limited as they can only move up and down along the ends of arena. Therefore, as a control of the paddle, the player needs only to move the iPad up and down.

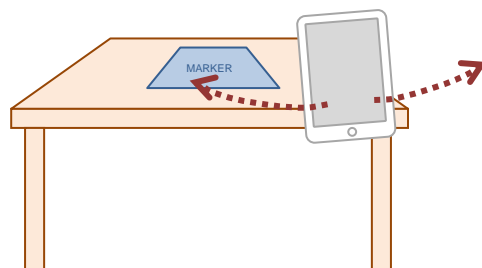


Fig 6.2.2 Control paddle with iPad movement

In addition, the view-angle should be correct that the player can see another player. This is quite different from the previous game Dodge ball which only requires the player to view the marker through iPad in horizontal. Rather, Pong requires the player to view the marker in oblique angle such that he can see how the other player control the paddle to move.

Last, the network part is required to be responsive as Pong is a real-time action game. Every changes of iPad movement by the player need to be tracked and synchronized so that the game can run smoothly.

## 6.3 Definitions

The definitions of game entities are listed below:

### **Paddle**

A long rectangular racket which is controlled by the player to hit the ball. They cannot rotate and can only translate along one direction. Also, they are bounded by the walls.

### **Ball**

A square or circle shaped object which bounces in the area between two walls. It can be hit by a paddle. It always bounces off with 45 degree angle. It moves faster and faster in one round.

### **Wall**

There are two walls in the arena. They are rigid and unmovable. When the ball hits them, the ball will bounce off.

### **Arena**

The main stage for running this game. If the ball goes outside the , the game will start over with a new round.

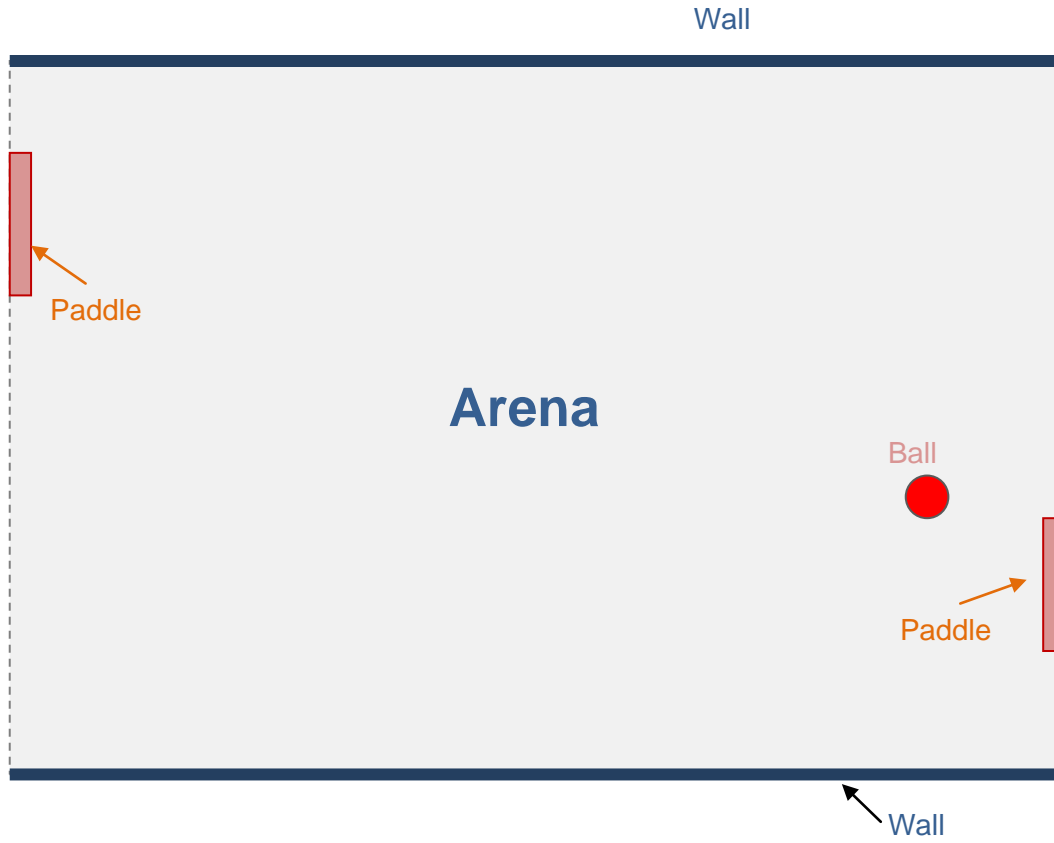


Fig 6.3.1 Arena layout

## 6.4 General Flow

### Splash screen

The splash screen is the first view of our application. It shows the title of our project. When the application first loads, this screen will appear.



### Menu

The menu is shown after splash screen. It shows four buttons which the user can click to perform different actions.

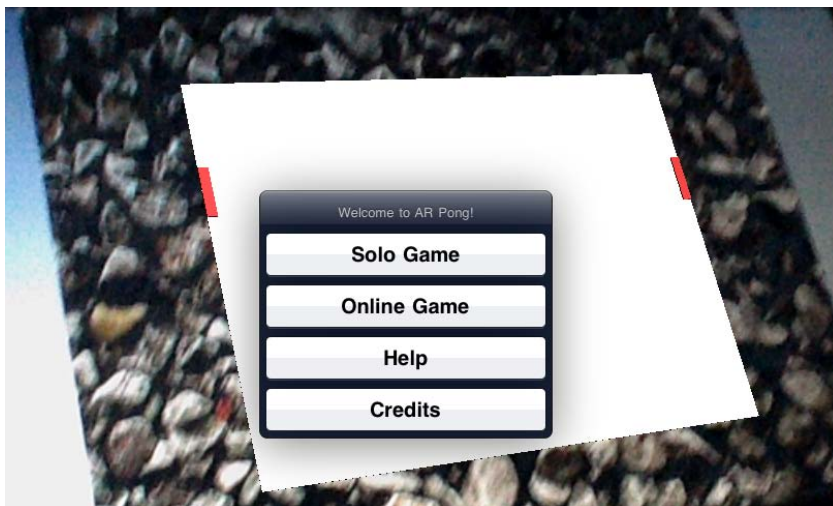


Fig 6.4.1 Game start menu



**Play Solo:** clicking this button navigates the user to the single player mode.

**Play Online:** clicking this button navigates the user to the multi-player mode.

**Help:** clicking this button shows the user a help page.

**Credits:** clicking this button shows the user developers information and acknowledgement.

To feature that our game is based on augmented reality, the background is the camera view and showing a demo which has two bot players playing against each other.

### Difficulty Selection Screen

The difficulty selection menu allows users to select the level of difficulty for single player game. There are basically three levels: easy, normal and hard.

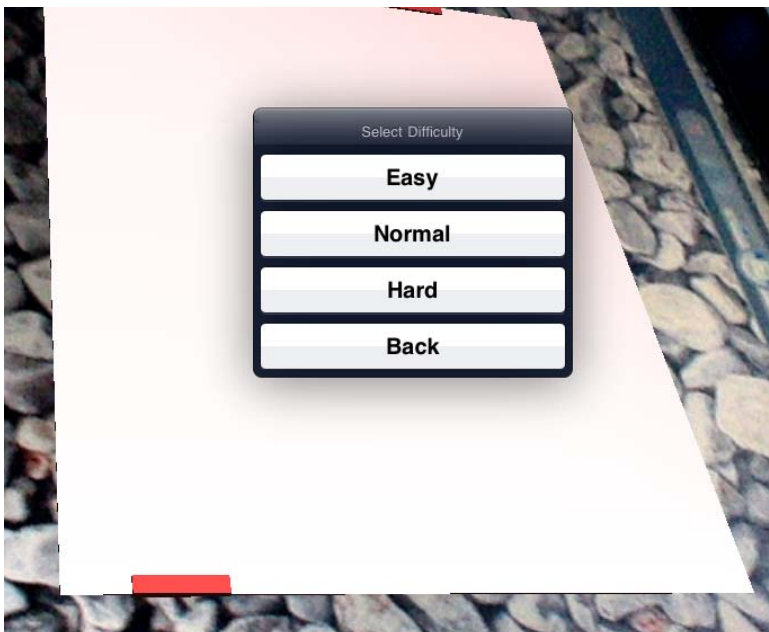


Fig 6.4.2 Difficulty selection

For the easy mode, the response time of bot player is long and the prediction error is large.

For the normal mode, the criterion of bot player is adjusted to simulate human level.

For the hard mode, bot player has very short response time and small prediction error.

After selecting a difficulty level, the player will enter the in-game screen and start a new game immediately.

### Multiplayer Pre-game Screen

This screen let the user to prepare for a multiplayer game.

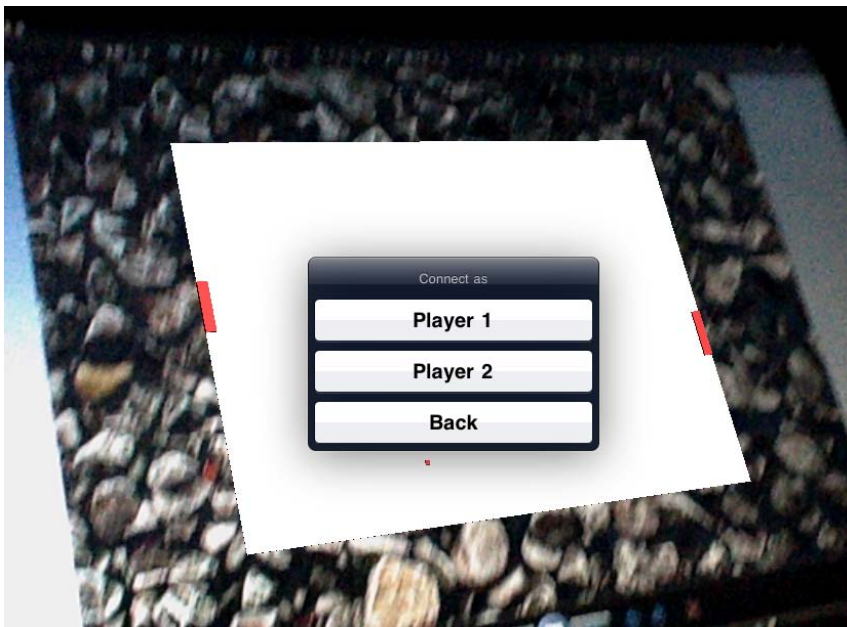


Fig 6.4.3 Network game start manual

## 6.5 System Architect

### 6.5.1 Introduction

This part discusses about the architecture of the whole game. Although this is a simple game, the system behind has a robust design and allow further extension.

Encapsulation is highly emphasized in our system design since this is a group project and we only want to show meaningful contractual interface functions. Once encapsulation is achieved, our Pong game can be changed in play style easily.

For example, it may allow 4 players in total and extra blocks inside the arena.

Practically, we provide basic functions of a object management system such as creation, retrieve and update functions.

Taking the functions related to paddle as example, we have the following to interact with a logical entity of paddle:

```
int addPaddle(int* id);
int getPaddle(int id, Paddle* _paddle);
int setPaddleGameId(int id, int gameId);
int setPaddleArea(int id, Area area);
int setPaddlePosition(int id, float x, float y);
int setPaddleDirection(int id, float dy);
int movePaddleUp(int id);
int movePaddleDown(int id);
int stopPaddleUp(int id);
int stopPaddleDown(int id);
int updatePaddle(int id, float dt);
```

`addPaddle(int* id)` registers a new paddle in the system. The user does not need to know how it is created or how memory is allocated but remember the object id returned by the function.

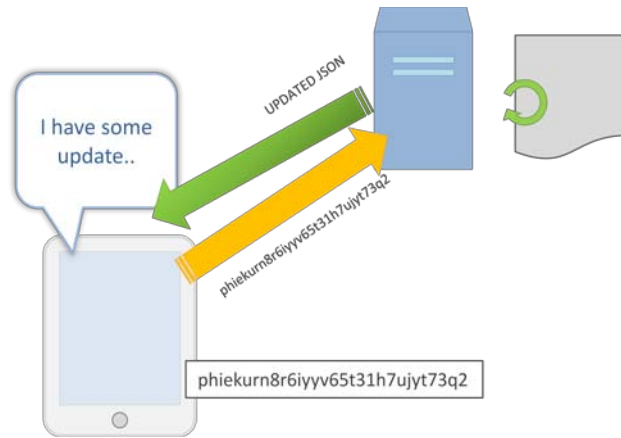
`getPaddle(int id, Paddle* _paddle)` retrieves a copy of the paddle with provide object id. The mutable copy `_paddle` contains all information of the selected paddle such as positions and dimensions.

The setter functions i.e. `setPaddleGameId`, `setPaddleArea`, `setPaddlePosition`, `setPaddleDirection` let users to set some attributes of a paddle. It ensures that users do not have a direct way to interact with the instance copy of a paddle as it is a dangerous practice. Values supplied to function calls are checked before they are actually written to the object memory. Also, some attributes such as paddle width, height, movement speed are not allowed to change as they are considered constant. Therefore, there is not setter functions for them.

Other functions such as `movePaddleUp`, `movePaddleDown`, `stopPaddleUp`, `stopPaddleDown`, `updatePaddle` are some utility functions which make the program logic clear.

Apart from encapsulation, we need also consider the RESTfulness of our application since it will communicates through the HTTP protocol.

Our application will send current states to the PHP links and perform actions based on the returning states.



**Fig. 6.5.1 Update states to the server and gets update**

Since Pong is a real-time action game, every event needs to be taken into account. Especially for the paddles' positions which are controlled by different players, the properties are sent to the server in real-time and then processed and sent back to other players.

Delay must be minimized as a small latency can lead to different consequences. For example, if they movement of paddle not tracked up-to-time, the ball may pass it and lead to a score while the other player see the ball bounces off the paddle.

Synchronization is very important in our application. To achieve that, we divide the functions as small as possible. Each function sends signal to the server as soon as their job is done. In this way, the server can receive up-to-date information from individual devices.

### 6.5.2 Objects and States

The system uses a number of objects to construct the game world. Each object has unique functionalities and is independent of other objects. The objects are referenced by their unique object id.

Basically, the system objects can be divided into three categories: core, logic and presentation.

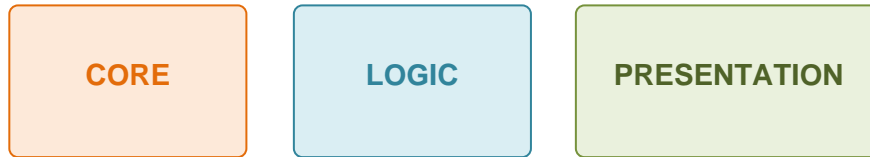


Fig 6.5.2.1 System objects

### **Core Part**

There is only one type of object for the core part. It is the Object:

```
typedef struct _Object {  
    int id;  
} Object;
```

Object has only one attribute – id. This is what uniquely identifies all system objects as they inherit from this class.

Usually, when user needs to interact with a system object, the object's id must be provided for calling functions.

### **Logic Part**

The logic part is the major part of the system. All game logic related objects are involved in this part. Below is the list of the objects:

**Game:** represents a Pong game.

**Player:** represents a game player

**Bot:** represents a computer controlled entity

**Paddle:** represents the rectangular block for hitting the ball

**Ball:** represents a block that bounces between players

**Court:** represents the game arena

**Motion:** represents the ball's motion states

**Contact:** represents the contact point made by the ball and paddles

The Game object defines a Pong game. It acts as the container for storing ids of component object and game states for one game session.

```
typedef struct _Game : Object {
    int playerCount;
    int playerId[2];
    int botCount;
    int botId[2];
    int paddleCount;
    int paddleId[2];
    int ballCount;
    int ballId[1];
    int courtCount;
    int courtId[1];
    GameState state;
    GameMode mode;
    GameDifficulty difficulty;
    int winningScore;
} Game;
```

The `playerId` array stores ids of the two players. This can be extended to 3 or 4 to support more players. There are always 2 player ids regardless of the game mode being whether single-player or multiplayer or demo.

The `botId` array stores ids of bot players. If it is a demo game, two bot ids are used. If it is a single-player game, one bot id is used.

The `paddleId` array stores ids of paddles. There are always 2 paddle id for a normal Pong game.

The `ballId` array stores id of the ball. Normally, there is only one ball in game but it certainly can extend to more balls.

The `courtId` array stores id of the court. There is only one court per game.



The state attribute defines the current state of a game object. It can be one of the following:

```
typedef enum _GameState {  
    Ready,  
    Running,  
    Paused,  
    Finished  
} GameState;
```

For Ready state, the game is ready to play. However, all drawable objects are not presented in screen and the game will not be updated in main loop.

For Running state, the game running and presented in screen.

For Paused state, the game is and will not be updated in main loop. However, all drawable objects are still presented in screen.

For Finished state, the game has just finished and the final result is ready to be shown. In this state, the game is not updated but objects are shown in addition of the scoreboard.

The mode attribute defines the game mode of a game object.

```
typedef enum _GameMode {  
    Solo,  
    Online,  
    Demo  
} GameMode;
```

For Solo mode, one human-controlled paddle against one bot-controlled paddle.

For Online mode, two human-controlled paddles against each other via network connection.

For Demo mode, two bot-controlled paddles against each other.

The difficulty attribute defines the level of difficulty for single-player mode.

```
typedef enum _GameDifficulty {  
    Easy,  
    Normal,  
    Hard  
} GameDifficulty;
```

For Easy, the bot player's AI is weak. It will have long response time and a large prediction error.

For Normal, the bot player's AI is average as human. It will have medium response time and prediction error.

For Hard, the bot player's AI is quite strong. It will have very short response time and very small prediction error.

The `winningScore` defines the number of points that a player needs to get in order to win the match.

The Player object stores a player's profile. It is the logical representation of a player being controlled by either human or bot.

```
typedef struct _Player : Object {  
    int gameId;  
    int score;  
} Player;
```

The `gameId` refers to which game this player is involved. It is set once the game is created.

The `score` records the player's current score.

Bot object stores attributes for AI-driven player.

```
typedef struct _Bot : Object {
    int gameId;
    int aiLevel;
    float reaction;
    int error;
    int paddleId;
    int ballId;
    int predictedAction; // do nothing: 0, should move: 1
    float predictedSince;
    float predictedBallX, predictedBallY;
    float predictedBallDx, predictedBallDy;
} Bot;
```

`gameId` refers to the game object from which this bot is created.

`aiLevel` indicates how intelligent the bot is. The lower it is, the more intelligent the bot is. It determines the values for `reaction` and `error`.

`reaction` is the reaction or response time of the bot taken when the ball is returning to its side. The bot makes prediction for every period of reaction time passes.

`error` is the positional error of the bot's prediction for the returning ball's final position.

`paddleId` refers to the paddle that is controlled by this bot. Each bot can only control exactly one paddle.

`ballId` refers to the ball that the bot calculate prediction for.

`predictedAction` indicates whether the bot should do nothing or should move according to its prediction.

`predictedSince` is the time past since last prediction is made.

`predictedBallX` and `predictedBallY` are the predicted position of the target ball object.

Paddle object stores the position, movement and dimension information for a paddle.

```
typedef struct _Paddle : Object {  
    int gameId;  
    int modelId;  
    Area area;  
    float width, height;  
    float minY, maxY;  
    float speed;  
    float x, y;  
    float left, right, top, bottom;  
    float up, down;  
} Paddle;
```

`gameId` refers to the game from which this paddle is created from.

`modelId` refers to the model which draws this paddle.

`area` tells on which region this paddle is. Normally, it is either North or South. For 4-player game, it can even be West or East.

`width` and `height` define the dimensions of the paddle.

`minY` and `maxY` bounds the vertical movement of the paddle.

`speed` is the translational speed of the paddle.

`x` and `y` define the position of the paddle.

`left`, `right`, `top`, `bottom` define the four corners of the paddle.

`up` and `down` determines whether the paddle is moving up or down.

Ball object stores the position, movement and dimension information for a ball.

```
typedef struct _Ball : Object {  
    int gameId;  
    int modelId;  
    Area area;  
    float radius;  
    float minX, minY, maxX, maxY;  
    float speed;  
    float accel;  
    float x, y;  
    float left, right, top, bottom;  
    float dx, dy;  
} Ball;
```

`gameId` refers to the game from which this ball is created from.

`modelId` refers to the model which draws this ball.

`area` tells on which region this ball is fired. Normally, it is either North or South. For 4-player game, it can even be West or East.

`radius` determines how big the ball is.

`minx, minY, maxX, maxY` bounds the ball movement.

`speed` determines how fast the ball moves.

`accel` is the acceleration of the ball.



$x$  and  $y$  define the position of the ball.

`left`, `right`, `top`, `bottom` define the four corners of the ball.

$dx$ ,  $dy$  are the change in position of the ball.

Court object defines a court in a Pong game.

```
typedef struct _Court : Object {  
    int gameId;  
    int modelId;  
    float width, height;  
    float wallWidth;  
} Court;
```

gameId refers to the game from which this ball is created from.

modelId refers to the model which draws this ball.

width and height define the dimension of the court.

wallWidth determines the width of the walls.



Fig 6.5.2.2 court object defines how a court looks like

Motion object is a temporary object for determining the motion of a ball.

```
typedef struct _Motion {  
    float nx, ny;  
    float x, y;  
    float dx, dy;  
} Motion;
```

`nx` and `ny` define the next position of the object.

`x` and `y` define the current position of the object.

`dx` and `dy` are the change in position of the object.

Contact object is a temporary object to record a contact point between paddle and ball.

```
typedef struct _Contact {  
    float x, y;  
    int d; // right: 1, left: 2, bottom: 3, top: 4  
} Contact;
```

$x$  and  $y$  define the contact point's position.

$d$  represents the direction of the contact point.

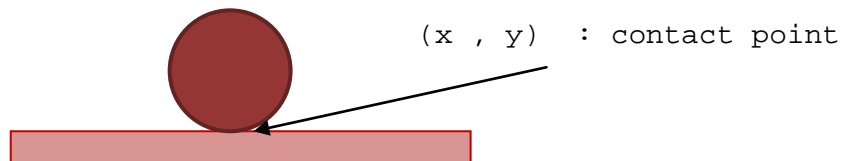


Fig 6.5.2.3 Ball and paddle contact

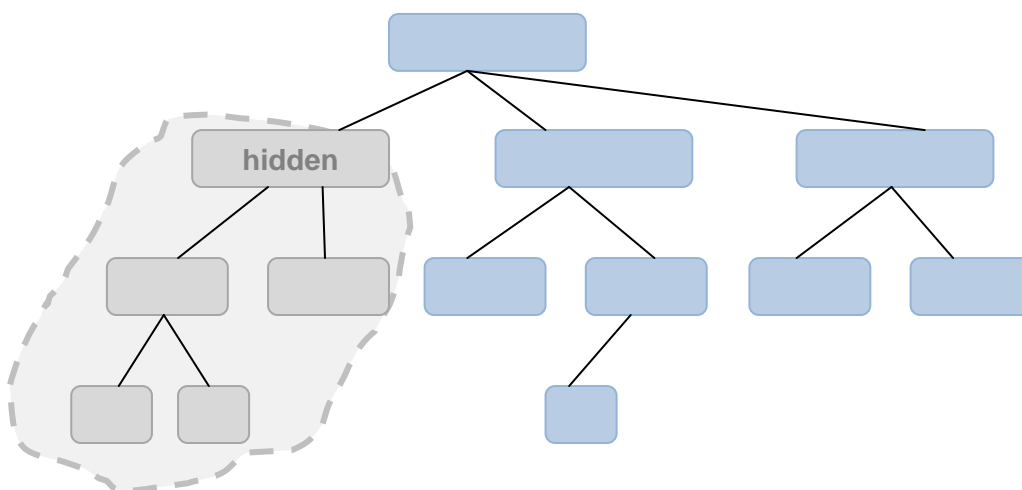
## Presentation Part

The presentation part is highly connected with the OpenGL library as it is about how to draw things on the screen.

Most of the game objects such as the paddles, the ball and the arena need to be shown on the screen. Hence, they are associated with models for presentation. The model class is defined as below:

```
typedef struct _Model : Object {  
    int parentId;  
    bool hidden;  
    QCAR::Vec3F position;  
    QCAR::Vec3F scale;  
    QCAR::Matrix44F transform;  
} Model;
```

parented is the id of another model object which is the parent of this model. The system employs a hierarchical structure for models. Positional or rotational changes to a model will affect its child models accordingly. Also, hiding a model would hide its child models as well.



**Fig 6.5.2.4 hierarchical structure of model object**

Hidden determines whether this model will be rendered or not.

Position defines the three-dimensional position of the model.

Scale defines the relative size of the model.

Transform is the transformation matrix state of the model.

### 6.5.3 Implementation

The application is developed with the iOS framework and the Vuforia AR SDK. Using the project template provided by Vuforia AR SDK, we could start development very quickly. The application is mainly written in C and Objective-C.

For source control, we have Pong.h and Pong.mm. Pong.h contains the interface code for users. Pong.mm contains the implementation code.

There are three categories for the implementation codes: application codes, game logic codes, object management codes and utility codes.

The application codes are responsible for connecting the game to the view controllers defined by Vuforia AR SDK. Also, they make calls to the OpenGL library to draw related models.

The game logic codes are responsible for how the game performs. For instance, it updates the ball's position by its speed and receive user input to control the paddle.

The object management codes are internal codes for managing the allocation and modification of system objects. They also determine how to interact with the server.

The utility codes are mostly mathematical and physical calculation for collision detection. They are used to detect whether the ball would hit another object and how it bounces off.

### Application Codes

`setTexture` set the array of textures to be used.

```
void setTextures(NSArray* t) { textures = t; }
```

`startDemo`, `stopDemo`, `startSolo`, `stopSolo` are used to control the game on the main menu.

```
void  
startDemo()  
{  
    setGameState(demoGameId, Running);  
}  
  
void  
stopDemo()  
{  
    setGameState(demoGameId, Ready);  
}  
  
void  
startSolo(GameDifficulty difficulty)  
{  
    setGameDifficulty(soloGameId, difficulty);  
}
```

```
        setGameState(soloGameId, Running);
    }

    void
    stopSolo()
    {
        setGameState(soloGameId, Ready);
    }

    initAll initialize all variables and reset object system to
    zero.

    void
    initAll()
    {
        // Initialize the globals
        // Note: the GL context is not set up at this point

        gameCount = 0;
        nextGameId = 1;
        modelCount = 0;
        nextModelId = 1;
        playerCount = 0;
        nextPlayerId = 1;
        botCount = 0;
        nextBotId = 1;
        paddleCount = 0;
        nextPaddleId = 1;
        ballCount = 0;
        nextBallId = 1;
        courtCount = 0;
        nextCourtId = 1;

        lastSystemTime = getCurrentTimeMS();
    }
}
```



```
initSoundEffect();

addGame(&soloGameId);
setGameState(soloGameId, Ready);
setGameMode(soloGameId, Solo);
setGameDifficulty(soloGameId, Normal);

addGame(&demoGameId);
setGameState(demoGameId, Ready);
setGameMode(demoGameId, Demo);

startDemo();
}
```

**renderAll** prepare a OpenGL frame to draw and get trackable markers transformation from the AR library. Then it updates the augmentation and render the models.

```
void
renderAll()
{
    // Get the time delta since the last frame
    unsigned long currentTime = getCurrentTimeMS();
    float dt = (currentTime - lastSystemTime)
/ 1000.0f;

    // Clear the color and depth buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Render the video background
    QCAR::State state =
QCAR::Renderer::getInstance().begin();
    QCAR::Renderer::getInstance().drawVideoBackground();

    // Did we find any trackables this frame?
    if (state.getNumActiveTrackables() > 0) {

        // Get the first trackable
        const QCAR::Trackable* trackable =
state.getActiveTrackable(0);

        // Get the model view matrix
        modelViewMatrix =
QCAR::Tool::convertPose2GLMatrix(trackable->getPose());

        // Update the augmentation
        updateAugmentation(trackable, dt);
        // Render the augmentation
```

```
        renderAugmentation(trackable);  
    }  
  
    QCAR::Renderer::getInstance().end();  
  
    // Store the current time  
    lastSystemTime = currentSystemTime;  
}
```

**initSoundEffect and playSoundEffect are responsible for play sound effects.**

```
void  
initSoundEffect()  
{  
    NSString* path = [[NSBundle mainBundle]  
pathForResource:@"Pong_tap" ofType:@"wav"];  
    NSURL* filePath = [NSURL URLWithString:path  
isDirectory:NO];  
    AudioServicesCreateSystemSoundID((CFURLRef) filePath,  
&soundID);  
}  
  
void  
playSoundEffect()  
{  
    AudioServicesPlaySystemSound(soundID);  
}
```

**updateAugmentation** updates all running games with the change in time dt.

```
void
updateAugmentation(const QCAR::Trackable*
trackable, float dt)
{
    Game* game;
    for (int i = 0; i < gameCount; i++) {
        game = &gameArray[i];
        if (game->state == Running) {
            updateGame(game->id, dt);
        }
    }
}
```

**renderAugmentation** prepares the OpenGL environment and render all visible models. The models are rendered in box form by calling the function **renderCube**.

```
void
renderAugmentation(const QCAR::Trackable* trackable)
{
    //const Texture* const modelTexture = [textures
objectAtIndex:0];

#ifdef USE_OPENGL1
    // Set GL11 flags
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```

```
    glTexCoordPointer(2, GL_FLOAT, 0, (const GLvoid*)
&cubeTexCoords[0]);
    glVertexPointer(3, GL_FLOAT, 0, (const GLvoid*)
&cubeVertices[0]);
    glNormalPointer(GL_FLOAT, 0, (const GLvoid*)
&cubeNormals[0]);

    //glEnable(GL_TEXTURE_2D);
    //glDisable(GL_LIGHTING);

    // Create light components
    GLfloat ambientLight[] = { 0.0f, 0.0f, 0.0f, 1.0f };
    GLfloat diffuseLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat specularLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat position[] = { 1.0f, 1.0f, 10.0f, 1.0f };

    // Assign created components to GL_LIGHT0
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
    glLightfv(GL_LIGHT0, GL_POSITION, position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);

    // Load projection matrix
    glMatrixMode(GL_PROJECTION);
    glLoadMatrixf([QCARutils
getInstance].projectionMatrix.data);

    // Load model view matrix
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(modelViewMatrix.data);
```

```
        glScalef(0.40f, 0.40f, 0.40f);
        glTranslatef(-kWidth / 2.0f, -kHeight / 2.0f, 0.0f);
    #else
        // Do nothing for OpenGL 2.0
    #endif

    glEnable(GL_DEPTH_TEST);

    // Render the Model
    //glBindTexture(GL_TEXTURE_2D, modelTexture.textureID);>
    for (int i = 0; i < modelCount; i++) {
        Model* Model = &modelArray[i];

        if ( Model->hidden == 0 ) {
            renderCube(&Model->transform.data[0]);
        }
    }

    ShaderUtils::checkGLError("Model renderFrame");

    glDisable(GL_DEPTH_TEST);

#ifdef USE_OPENGL1
    //glDisable(GL_TEXTURE_2D);
    glDisableClientState(GL_VERTEX_ARRAY);
    glDisableClientState(GL_NORMAL_ARRAY);
    glDisableClientState(GL_TEXTURE_COORD_ARRAY);
#else
    // Do nothing for OpenGL 2.0
#endif
}

void
```

```
renderCube(float* transform)
{
    // Render a cube with the given transform
    // Assumes prior GL setup

#ifdef USE_OPENGL1
    glPushMatrix();
    glColor4f(0.5, 0.7, 0.2, 1.0);
    glMultMatrixf(transform);
    glScalef(0.5f, 0.5f, 0.5f);
    glDrawElements(GL_TRIANGLES, NUM_CUBE_INDEX,
GL_UNSIGNED_SHORT, (const GLvoid*) &cubeIndices[0]);
    glPopMatrix();
#else
    // Do nothing for OpenGL 2.0
#endif
}
```

## Game Logic Codes

updateGame is the main part of game logic.

It first checks the user input by calculating how is his line of sight deviated from the x-axis.

Next, it updates all game objects such as the bots, paddles and the balls based on the change of time dt.

Finally, it validates game objects' positional information against the dimension of the arena.

```
int
updateGame(int id, float dt)
{
    Game* game = getGame(id);

    if ( game == NULL ) {
        return 0;
    }

    if ( (game->mode == Solo || game->mode == Online) &&
game->difficulty == Easy ) {
        // Determine user input
        QCAR::Matrix44F inverse =
SampleMath::Matrix44FInverse(modelViewMatrix);
        static float lastY = 0;
        static const float threshold = 0.5f;
        float y = inverse.data[7];
        if ( y - lastY > threshold ) {
            setPaddleDirection(game->paddleId[0], 1);
            //setPaddleDirection(1, 1);
        }
    }
}
```



```
        } else if ( lastY - y > threshold ) {
            setPaddleDirection(game->paddleId[0], -1);
            //setPaddleDirection(1, -1);
        } else {
            setPaddleDirection(game->paddleId[0], 0);
            //setPaddleDirection(1, 0);
        }
        lastY = y;
    }
    if ( (game->mode == Solo || game->mode == Online) &&
        (game->difficulty == Normal || game->difficulty == Hard) ) {
        // Determine user input
        QCAR::Matrix44F inverse =
        SampleMath::Matrix44FInverse(modelViewMatrix);
        float pan_angle = atan2(inverse.data[7], -
        inverse.data[3]) * (180/M_PI);
        float displacement = kHeight/2.0f + (kHeight/2.0f) *
        (pan_angle/15.0f);
        if ( displacement > kHeight - kPaddleHeight -
        kWallWidth ) {
            displacement = kHeight - kPaddleHeight -
            kWallWidth;
        } else if ( displacement < kWallWidth ) {
            displacement = kWallWidth;
        }

        Paddle paddle;
        getPaddle(game->paddleId[0], &paddle);

        setPaddlePosition(game->paddleId[0], paddle.x,
        displacement);
    }

    if ( game->mode == Solo ) {
```

```
        updateBot(game->botId[0], dt);

    } else if ( game->mode == Demo ) {
        updateBot(game->botId[0], dt);
        updateBot(game->botId[1], dt);
    }

    updatePaddle(game->paddleId[0], dt);
    updatePaddle(game->paddleId[1], dt);

    updateBall(game->ballId[0], dt);

    Ball ball;
    getBall(game->ballId[0], &ball);

    if ( ball.left > kWidth ) {
        addPlayerScore(game->playerId[0], 1);
        setBallArea(game->ballId[0], North);
    } else if ( ball.right < 0 ) {
        addPlayerScore(game->playerId[1], 1);
        setBallArea(game->ballId[0], South);
    }

    return 1;
}
```

**updateModelTransform** updates model's matrix transform state by multiplying its position matrix with scale matrix.

```
int
updateModelTransform(int id)
{
```

```
Model* model = getModel(id);

if ( model == NULL ) {
    return 0;
}

// Reset the Pong transform to the identity matrix
model->transform = SampleMath::Matrix44FIdentity();
float* transformPtr = &model->transform.data[0];

// The following transformations happen in reverse order
// We want to scale the Pong, tip the Pong (on its
leading edge), pivot and then position the Pong
    ShaderUtils::translatePoseMatrix(model->position.data[0],
model->position.data[1], 0.0f, transformPtr);
    ShaderUtils::scalePoseMatrix(model->scale.data[0],
model->scale.data[1], model->scale.data[2], transformPtr);

    return 1;
}
```

**updateBot** updates the bot's action as the time advanced with dt. It calls the **updateBotPrediction** to determine whether it needs to move up or down.

**updateBotPrediction** updates the bot's prediction. It will calculate the ball's new position by its speed and acceleration.

```
int
updateBot(int id, float dt)
{
    Bot* bot = getBot(id);

    if ( bot == NULL ) {
```

```
        return 0;
    }

    Ball ball;
    getBall(bot->ballId, &ball);
    Paddle paddle;
    getPaddle(bot->paddleId, &paddle);

    if (((ball.x < paddle.left) && (ball.dx < 0)) ||
        ((ball.x > paddle.right) && (ball.dx > 0))) {
        stopPaddleUp(paddle.id);
        stopPaddleDown(paddle.id);
        return 1;
    }

    updateBotPrediction(id, dt);

    if ( bot->predictedAction == 1 ) {
        if (bot->predictedBallY < (paddle.top +
paddle.height/2 - 5)) {
            movePaddleUp(paddle.id);
            stopPaddleDown(paddle.id);
        } else if (bot->predictedBallY > (paddle.bottom -
paddle.height/2 + 5)) {
            stopPaddleUp(paddle.id);
            movePaddleDown(paddle.id);
        } else {
            stopPaddleUp(paddle.id);
            stopPaddleDown(paddle.id);
        }
    }

    return 2;
}
```

```
int
updateBotPrediction(int id, float dt)
{
    Bot* bot = getBot(id);

    if ( bot == NULL ) {
        return 0;
    }

    Ball ball;
    getBall(bot->ballId, &ball);
    Paddle paddle;
    getPaddle(bot->paddleId, &paddle);

    // only re-predict if the ball changed direction, or its
    // been some amount of time since last prediction
    if ((bot->predictedAction == 1) &&
        ((bot->predictedBallDx * ball.dx) > 0) &&
        ((bot->predictedBallDy * ball.dy) > 0) &&
        (bot->predictedSince < bot->reaction)) {
        bot->predictedSince += dt;
        return 1;
    }

    Paddle* tmp_paddle = new Paddle;
    tmp_paddle->left = paddle.left;
    tmp_paddle->right = paddle.right;
    tmp_paddle->top = -10000;
    tmp_paddle->bottom = 10000;

    Contact* pt = findBallPaddleIntercept(&ball, tmp_paddle,
    ball.dx * 10, ball.dy * 10);
```

```
delete tmp_paddle;

if (pt) {
    float t = paddle.minY + ball.radius;
    float b = paddle.maxY + paddle.height - ball.radius;

    while ((pt->y < t) || (pt->y > b)) {
        if (pt->y < t) {
            pt->y = t + (t - pt->y);
        }
        else if (pt->y > b) {
            pt->y = t + (b - t) - (pt->y - b);
        }
    }

    bot->predictedAction = 1;
    bot->predictedSince = 0;
    bot->predictedBallX = pt->x;
    bot->predictedBallY = pt->y;
    bot->predictedBallDx = ball.dx;
    bot->predictedBallDy = ball.dy;

    float closeness = (ball.dx < 0 ? ball.x -
paddle.right : paddle.left - ball.x) / kWidth;
    int error = bot->error * closeness;
    bot->predictedBallY = bot->predictedBallY +
(float)(-error + rand()%(2*error));
    } else {
        bot->predictedAction = 0;
    }

    return 2;
}
```

`updatePaddle` updates the paddle position with time advanced with `dt`. Also, it will update the position of associated model.

```
int
updatePaddle(int id, float dt)
{
    Paddle* paddle = getPaddle(id);

    if ( paddle == NULL ) {
        return 0;
    }

    float amount = paddle->down - paddle->up;
    if (amount != 0) {
        float y = paddle->y + (amount * dt * paddle->speed);

        if (y < paddle->minY) {
            y = paddle->minY;
        } else if (y > paddle->maxY) {
            y = paddle->maxY;
        }

        setPaddlePosition(id, paddle->x, y);
    }

    setModelPosition(paddle->modelId, QCAR::Vec3F(paddle->x
+ paddle->width / 2.0f, paddle->y + paddle->height
/ 2.0f, 1.0f));
    updateModelTransform(paddle->modelId);

    return 1;
}
```

**updateBall** updates the ball's new position by its current position, speed, acceleration and change in time dt.

**It checks for collision detection with paddles. If any collision is detected, it will calculate the new direction such that the ball would bounce off.**

```
int
updateBall(int id, float dt)
{
    Ball* ball = getBall(id);

    if ( ball == NULL ) {
        return 0;
    }

    Motion* pos = accelerate(ball->x, ball->y, ball->dx,
ball->dy, ball->accel, dt);

    if ((pos->dy > 0) && (pos->y > ball->maxY)) {
        pos->y = ball->maxY;
        pos->dy = -pos->dy;
    }
    else if ((pos->dy < 0) && (pos->y < ball->minY)) {
        pos->y = ball->minY;
        pos->dy = -pos->dy;
    }

    Game game;
    getGame(ball->gameId, &game);

    int paddleId = (pos->dx < 0) ? game.paddleId[0] :
game.paddleId[1];
    Paddle paddle;
    getPaddle(paddleId, &paddle);

    Contact* pt      = findBallPaddleIntercept(ball, &paddle,
```



```

pos->nx, pos->ny);
    if (pt) {
        switch(pt->d) {
            case 1:
            case 2:
                pos->x = pt->x;
                pos->dx = -pos->dx;
                break;
            case 3:
            case 4:
                pos->y = pt->y;
                pos->dy = -pos->dy;
                break;
        }

        // add/remove spin based on paddle direction
        if (paddle.up) {
            pos->dy = pos->dy * (pos->dy < 0 ? 0.5 : 1.5);
        } else if (paddle.down) {
            pos->dy = pos->dy * (pos->dy > 0 ? 0.5 : 1.5);
        }

        delete pt;
    }

    setBallPosition(id, pos->x, pos->y);
    setBallDirection(id, pos->dx, pos->dy);
    delete pos;
    setModelPosition(ball->modelId, QCAR::Vec3F(ball->x,
ball->y, 1.0f));
    updateModelTransform(ball->modelId);

    return 1;
}

```

## Object Management Codes

The object management codes are about how a new game object is created or modified or deleted.

The following are the arrays for storing the game object data. Also, there are get functions for internal use.

```
Game gameArray[MAX_GAMES];
int gameCount;
int nextGameId;
Game* getGame(int id);
void clearGames();
```

```
Model modelArray[MAX_MODELS];
int modelCount;
int nextModelId;
Model* getModel(int id);
void clearModels();
```

```
Player playerArray[MAX_PLAYERS];
int playerCount;
int nextPlayerId;
Player* getPlayer(int id);
void clearPlayers();
```

```
Bot botArray[MAX_BOTS];
int botCount;
int nextBotId;
Bot* getBot(int id);
void clearBots();
```

```
Paddle paddleArray[MAX_PADDLES];
int paddleCount;
```

```
int nextPaddleId;
Paddle* getPaddle(int id);
void clearPaddles();

Ball ballArray[MAX_BALLS];
int ballCount;
int nextBallId;
Ball* getBall(int id);
void clearBalls();

Court courtArray[MAX_COURTS];
int courtCount;
int nextCourtId;
Court* getCourt(int id);
void clearCourts();

int addGame(int* id);
int getGame(int id, Game* game);
int resetGame(int id);
int setGameState(int id, GameState state);
int setGameMode(int id, GameMode mode);
int setGameDifficulty(int id, GameDifficulty
difficulty);

int addModel(int* id);
int getModel(int id, Model* model);
int setModelParentId(int id, int parentId);
int setModelHidden(int id, bool hidden);
int setModelPosition(int id, QCAR::Vec3F
position);
int setModelScale(int id, QCAR::Vec3F scale);
```

```
int addPlayer(int* id);
int getPlayer(int id, Player* _player);
int setPlayerGameId(int id, int gameId);
int setPlayerScore(int id, int score);
int addPlayerScore(int id, int score);

int addBot(int* id);
int getBot(int id, Bot* _bot);
int setBotGameId(int id, int gameId);
int setBotAiLevel(int id, int aiLevel);
int setBotPaddleId(int id, int paddleId);
int setBotBallId(int id, int ballId);

int addPaddle(int* id);
int getPaddle(int id, Paddle* _paddle);
int setPaddleGameId(int id, int gameId);
int setPaddleArea(int id, Area area);
int setPaddlePosition(int id, float x, float y);
int setPaddleDirection(int id, float dy);
int movePaddleUp(int id);
int movePaddleDown(int id);
int stopPaddleUp(int id);
int stopPaddleDown(int id);

int addBall(int* id);
int getBall(int id, Ball* _ball);
int setBallGameId(int id, int gameId);
```

```
int setBallArea(int id, Area area);  
int setBallPosition(int id, float x, float y);  
int setBallDirection(int id, float dx, float dy);  
  
int addCourt(int* id);  
int getCourt(int id, Court* _court);  
int setCourtGameId(int id, int gameId);
```

## Utility Codes

The utility codes are some time functions and collision detection functions.

They are used in other part of codes so that the program looks clearer.

```
unsigned long  
getCurrentTimeMS() {  
    struct timeval tv;  
    gettimeofday(&tv, NULL);  
    unsigned long s = tv.tv_sec * 1000;  
    unsigned long us = tv.tv_usec / 1000;  
    return s + us;  
}
```

**accelerate** calculates and returns the positional states of the ball during the time interval **dt**.

```
Motion*  
accelerate(float x, float y, float dx, float dy, float accel  
, float dt)  
{  
    float x2 = x + (dt * dx) + (accel * dt * dt * 0.5);  
    float y2 = y + (dt * dy) + (accel * dt * dt * 0.5);  
    float dx2 = dx + (accel * dt) * (dx > 0 ? 1 : -1);  
    float dy2 = dy + (accel * dt) * (dy > 0 ? 1 : -1);  
  
    Motion* motion = new Motion;  
    motion->nx = x2 - x;  
    motion->ny = y2 - y;  
    motion->x = x2;  
    motion->y = y2;  
    motion->dx = dx2;  
    motion->dy = dy2;  
    return motion;  
}
```

## Intercept accurately detects whether the ball would intercept with a paddle during the time interval dt.

The ball moves from  $(x_1, y_1)$  to  $(x_2, y_2)$  during the time interval (dt) and the paddle edge stretches from  $(x_3, y_3)$  to  $(x_4, y_4)$ . We need to see if these line segments intersect.

We want to see if the ball intercepts the side of the racket.

If the ball is moving left, check the right edge of player 1's racket.

If the ball is moving right, check the left edge of player 2's racket.

```
Contact*
intercept(float x1, float y1, float x2, float y2, float x3, float
y3, float x4, float y4, int d)
{
    float denom = ((y4-y3) * (x2-x1)) - ((x4-x3) * (y2-y1));
    if (denom != 0) {
        float ua = (((x4-x3) * (y1-y3)) - ((y4-y3) * (x1-x3))) /
denom;
        if ((ua >= 0) && (ua <= 1)) {
            float ub = (((x2-x1) * (y1-y3)) - ((y2-y1) * (x1-x3)))
/ denom;
            if ((ub >= 0) && (ub <= 1)) {
                float x = x1 + (ua * (x2-x1));
                float y = y1 + (ua * (y2-y1));
                Contact* contact = new Contact;
                contact->x = x;
                contact->y = y;
                contact->d = d;
                return contact;
            }
        }
    }
    return NULL;
}
```

```
}

Contact*
findBallPaddleIntercept(Ball* ball, Paddle*
paddle, float nx, float ny)
{
    Contact* contact;
    if (nx < 0) {
        contact = intercept(ball->x, ball->y, ball->x + nx, ball-
>y + ny,
                            paddle->right + ball->radius,
                            paddle->top - ball->radius,
                            paddle->right + ball->radius,
                            paddle->bottom + ball->radius,
                            1); // 1: right
    }
    else if (nx > 0) {
        contact = intercept(ball->x, ball->y, ball->x + nx, ball-
>y + ny,
                            paddle->left - ball->radius,
                            paddle->top - ball->radius,
                            paddle->left - ball->radius,
                            paddle->bottom + ball->radius,
                            2); // 2: left
    }
    if (!contact) {
        if (ny < 0) {
            contact = intercept(ball->x, ball->y, ball->x + nx,
ball->y + ny,
                                paddle->left - ball->radius,
                                paddle->bottom + ball->radius,
                                paddle->right + ball->radius,
                                paddle->bottom + ball->radius,
                                3); // 3: bottom
        }
        else if (ny > 0) {
            contact = intercept(ball->x, ball->y, ball->x + nx, ball->y + ny,
                                paddle->left - ball->radius,
```



```

        paddle->top    - ball->radius,
        paddle->right  + ball->radius,
        paddle->top    - ball->radius,
        4); // 4: top
    }
}
return contact;
}

```

## 6.5.4 User Control

There are two types of user control: relative positioning and absolute positioning. Both use the information of the transformation data of tracked marker but in different ways.

Relative positioning detects the direction of device movement. User needs to keep moving device in the same direction in order to move the paddle. Previous positional information is not recorded.

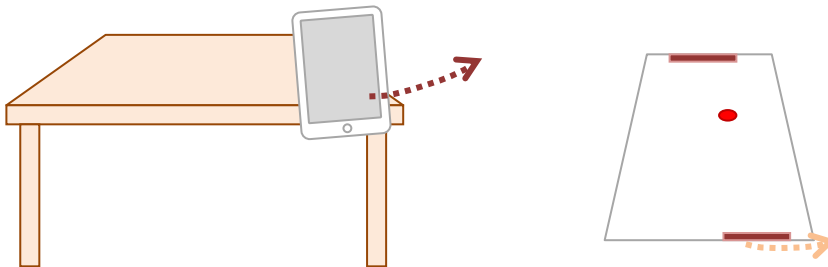


Fig 6.5.4.1 iPad moves to control the paddle

Absolute positioning detects the deviated angle of the device from the x-axis. Since the detected difference is absolute, the paddle's position represents the device's position

### Comparison

After several testing, we find that absolute positioning gives stable user control. Therefore, the absolute positioning method is better.

# Chapter 7. Experiment

---

## 7.1 Camera match-moving

### Definition

Camera match-moving is the process of analyzing a video clip or film shot to determine the location of 3D camera. It is the method that we use to find out the relative position of player with respect to their markers. Therefore, a stable camera match-moving is very important for our project.

### Objective

Since the algorithm and computation are provided by the QCAR SDK as internal code, we cannot change or even view it. We can only investigate the effect of external criteria. In this experiment, we are going to investigate the effect of marker's properties e.g. size, number of features on the stability of camera match-moving.

### Setup

The experiment setup is shown as above. An iPad is placed near the edge of table such that its front camera can capture the paper surface underneath. The paper is printed with a marker for each control. We have created an application to record the position of iPad itself by camera match-moving. When the application starts to record, the paper is pushed forward steadily. Having moved 10cm, the recording ends. The application can replay the movement of iPad by representing it as a 3D model. By observing the replay and position data, we can determine the stability of camera match-moving.

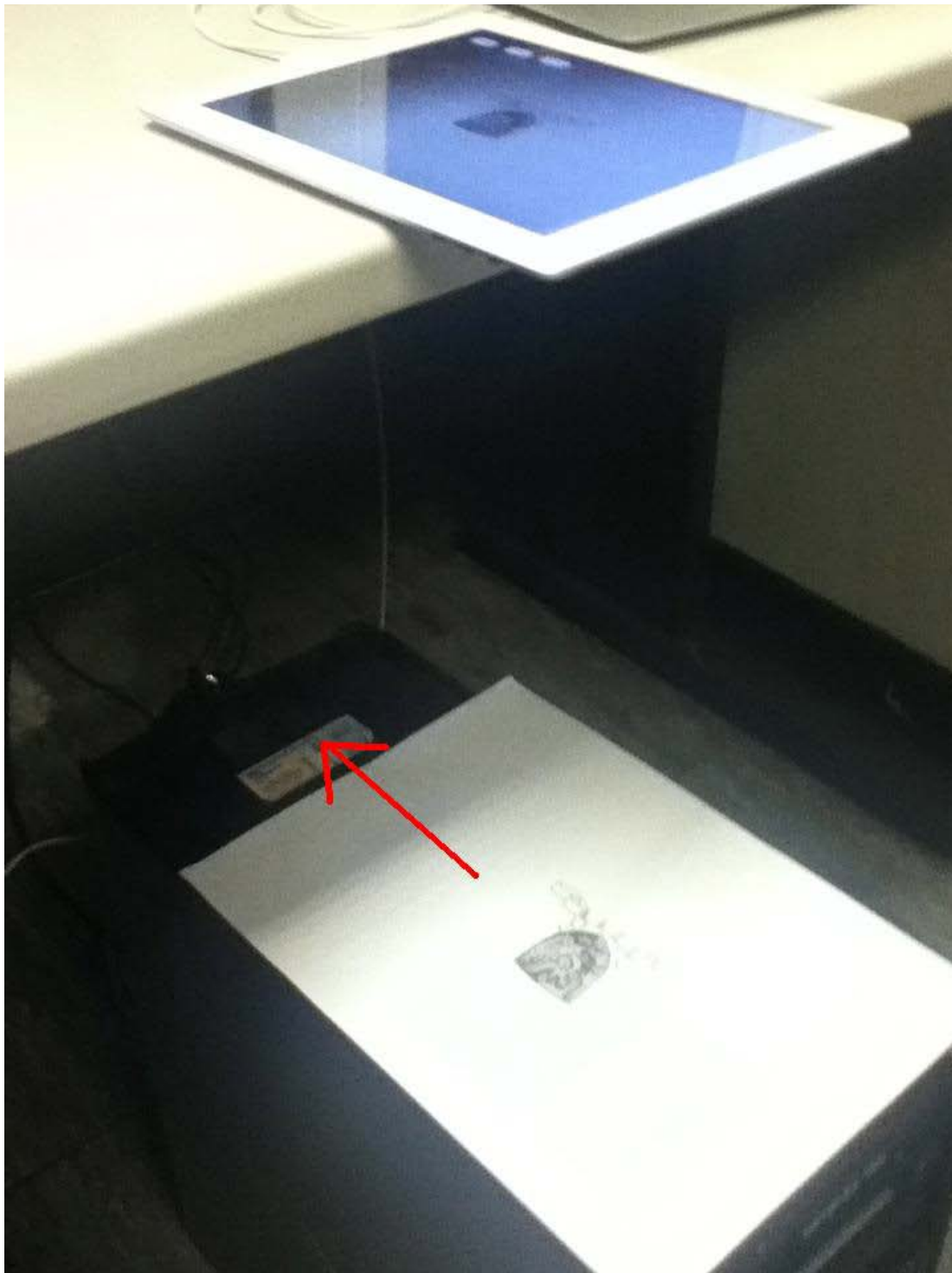


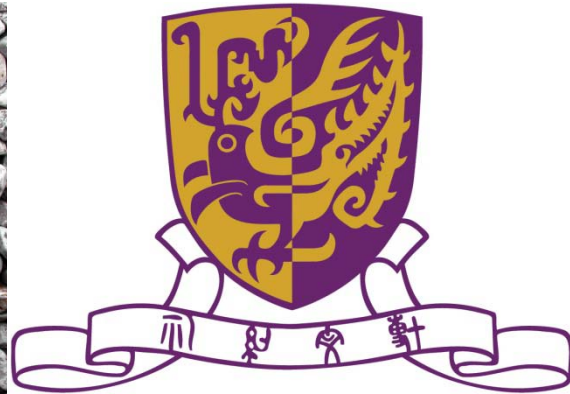
Fig. 7.1.1 Setup of the experiment

### Controls

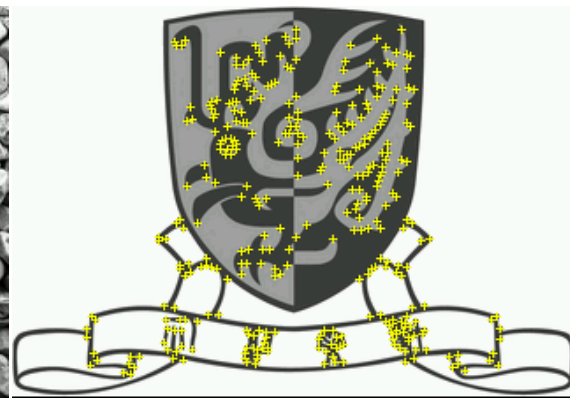
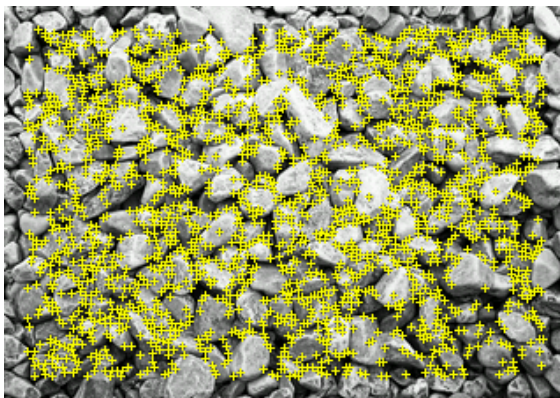
We will have 4 set of setups. Each one has different size and number of features of marker. For different number of features, we prepared two markers as below:

*i.Digi.T.able* - Digital Interactive Game Interface Table Apps for iPad

### Original images



### With features shown



The CUHK logo has less number of features than the stones image. Below is the description from Qualcomm AR Online Management System:

#### **CUHK Logo**

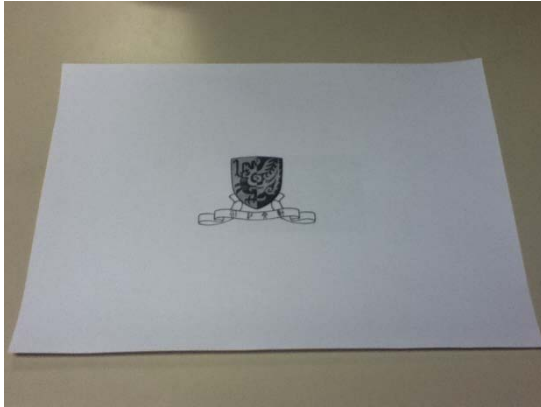
This image will track in most conditions but may not be robust to occlusions. If your application demands the best tracking performance then please consider improving the image based by Increasing total number of features in the image by adding more visual detail to the whole scene.

#### **Stones**

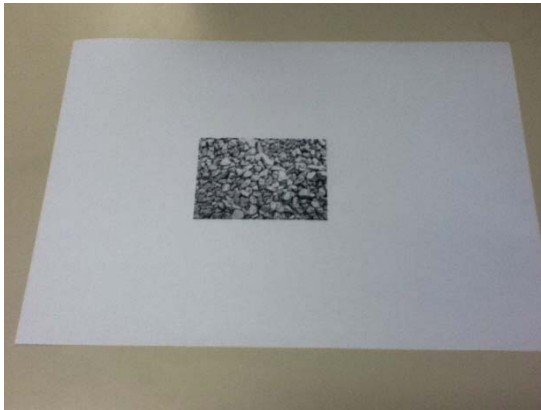
This image provides excellent tracking performance

The 4 different markers (all A4-sized) are the following:

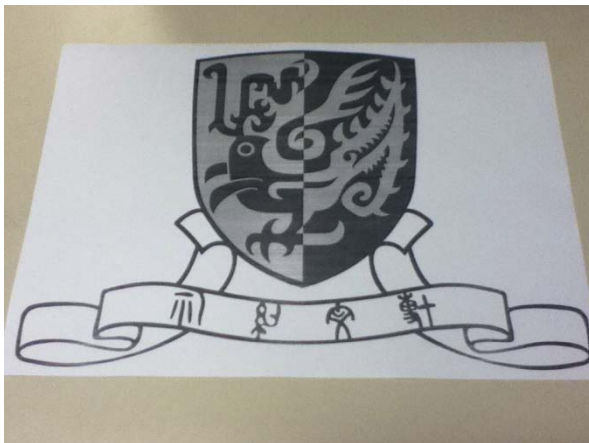
**Control A (less feature and small size)**



**Control B (more feature and small size)**



**Control C (less feature and large size)**



### Control D (more feature and large size)



### Results

For the detailed result, Please see appendix.

### Observations

For all controls, the z value keeps constant. It is because the marker is always on the same level of horizon. Hence, we can ignore the z value from our evaluation.

For control A and control B, the x value changes regularly. However, the change differences are not constant for successive frames. Also, the y value should be unchanged since the movement of the paper is aligned to the camera capture area. However, the y value oscillates regularly.

For control C and control D, the x value changes regularly. In addition, the change differences are constant and small for successive frames. Also, the y value remains constant throughout the experiment.

### Evaluation

The best stability of camera match-moving is given by control C and control D. In the controls, the x value changes smoothly and the y value remains unchanged. In their replays, the movement of 3D model matches the camera movement.

Both control A and control B give unstable camera match-moving results. They are too sensitive to a small change in relative movement between the camera and the marker. In their replays, the 3D model seems to teleport from one place to another and the movement is obviously not smooth. This is an implication of bad camera match-moving.

### **Conclusion**

We found that the relative size of the marker is crucial to the stability of camera match-moving. A large marker gives more stable result than a small large marker. The marker size also outweighs that the number of features of the marker.

We think that a full-sized marker on A4 paper has the best result in camera match-moving.

Apart from the stability of camera match-moving, we are also interested in other factors.

### **Movement of the iPad**

The movement of users' iPads is very limited because it must have the marker detected in the field of camera view. Hence, we have to allow as more freedom as possible to the device by change the property of the marker.

We found that a larger marker give the device more space to move around. Like the previous result, we think that the best size is full A4 size.

### **Camera shake**

It is unavoidable that the camera shakes when user holds the device with hand. When the tracker detects minor shaking, triggers the Virtual world construction module to recalculate the objects' location in the virtual world frequently. Also information will be updated to the server. The displayed world would be shaky, which gives a worse experience to user.

We can apply algorithm to detect and compromise with minor camera shaking action. The algorithm should distinguish between “shaking” and “moving” track of the camera, so that the object display would stay more static in user’s point of view.

## 7.2 Networking

After we have set up the server, the protocol proposed in design section is tested.

### **Objective**

Since we will have to send data over the network very frequently, the stability and reliability of the network medium and protocol is important.

Therefore it is necessary to test out the implementation.

### **Setup**

Server program set up on web server of the department.

2 iPad clients and 2 identical markers.

Installed experimental program which can register device on server and send marker location information to server

Instead of a 3D coordination, we send a 2D coordination to the remote client and test on a 2D plane only.



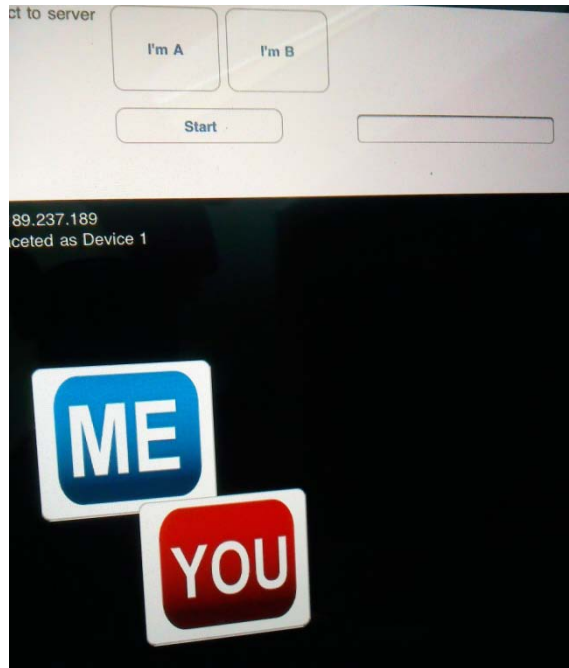


Fig. 6.2.1 Program interface

On this network testing client program, user can see **ME** and **YOU** label on the screen. To connect, press “I’m A” / “I’m B” on top of the menu bar.

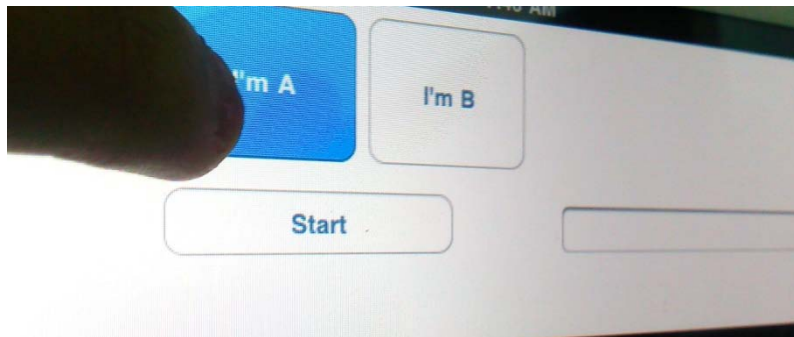
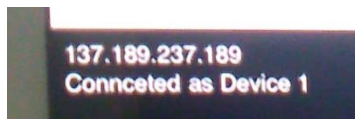


Fig. 7.2.2 Connecting to the server

If it is connected successfully, a message will be printed out.



*“(IP) Connected as Device (1/2)”*

Fig. 7.2.3 Message if connection succeed

### Procedure

For the testing procedure, we try to drag and drop the “ME” label, it will move according to the finger gesture.

The relative coordination is recorded and send to the server immediately.

During this process, the client will obtain updated coordination from another client and use this coordination to update the “YOU” label.

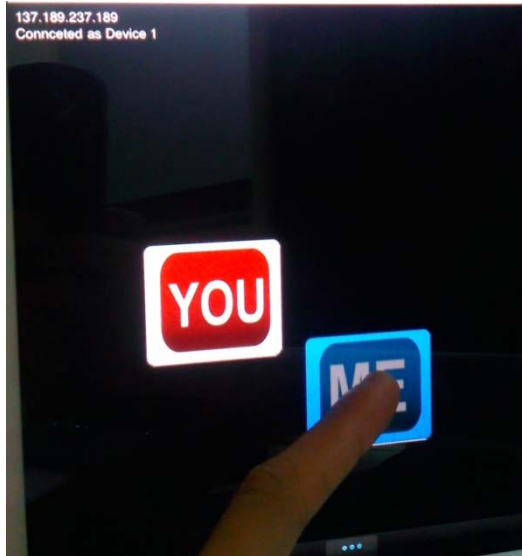


Fig. 7.2.4 Drag and drop the “ME” label

### Results

The observation result : the response is instant but the tracking of “YOU” label is a bit lag.

### Evaluation

The frequency of getting updated information should be raised. But that may need to a huge amount of requests from the client. One of the possible solution is to investigate a motion tweening algorithm on client side, so as to reduce the number of updates needed and hence reduce the use of network bandwidth.

## Chapter 8. Contribution of work

---

This part summarizes my contribution towards the project and experience gained from the journey of finishing this project.

### Summer 2011

After our group has chosen *i.Digi.T.able* as the final year project topic last summer, my partner and I had to study the background information and previous implementation of this topic.

General ideas were generated after several private discussions among our team; we have come to a different approach on how to implement the idea “digital interactive game interface table” on iPad. Then I started to draft a proposal on using Augmented Reality technology in our project. Later, research has been done on AR-related topic.

As I had an intern job during summer 2011, I only could take limited time to learn and get familiar with iOS programming with Objective-C. This helps to develop an application on iOS faster in later stages.

### 1<sup>st</sup> Semester

As the school term kicked started, we started our regular meeting on FYP with Prof. Lyu and the VIEW Lab team. First we presented our idea on the modified topic – using AR to implement instead of using an external camera. Then we have to design a game to illustrate our implementation.

The main aim of our topic is to share a platform over internet. We started searching for suitable tools and libraries to assist our work. Meanwhile, how data can be exchanged is also another important issue. Therefore, I started to

investigate how to connect two iPad using the Internet. Experiments had been performed to test the usability and effectiveness of the proposed protocol.

Limited by time, a simple server is set up to accept requests from two clients and forward them to respective clients in order to enable communication between two iPads.

After the first prototype has been built, we assemble client part and network part together to launch our sample program in 1<sup>st</sup> term.

## **2<sup>nd</sup> Semester**

In the 2<sup>nd</sup> semester, we evaluated our product in 1<sup>st</sup> term. We want to improve the experience of “feeling in the same space” therefore we proposed a new game.

We also include a newer version of AR library to improve the performance of tracker detection. Our main time has been spent on programming the new game and fine-tuning it. As we are not expert in OPENGL, we spent quite a lot of time figuring out about 3D graphics.

As the Pong game also includes a network-based mode, the server is also enhanced in the 2<sup>nd</sup> term. In this version, data is sent in JSON format. SQLite is also used for handling data.

After finishing both parts, we merged the components together and apply beautification on the interface. I helped designing graphics and interface in the app.

## **Conclusion**

Having finishing this final year project, I gained experiences in iOS programming and learnt a new language Objective-C. Secondly, I have

understanding on how data can be exchanged through internet in a real time manner.

Moreover, in a one-year-long project, collaboration and communication is important. I think I have room of improvement in this part. The project gives us a little bit taste of a small-scale Software development – from designing to implementing and launching. During the development, I also learnt that user experience is an important issue in designing software.

It is very valuable to me as I might have chance to face even more complex situations in the future. The journey working on this project has equipped me with much precious experience and fruitful knowledge in this field.

## Chapter 9. Conclusion

---

To conclude, we have achieved the followings in our final year project:

- Track the real-object mark and determine the camera's position
- Display simple objects on virtual space depends on real space scenes
- Exchange position information between 2 iPad clients
- Implement a simple AR game on iOS platform (iPad)

In the 1<sup>st</sup> semester, we mainly focused on tracking AR marker and analysis positional data for the device with the QCAR SDK. We also investigated how to exchange data via the network.

To demonstrate the progress, we designed a simple game, which let two users play Dodge ball over the network.

In 2<sup>nd</sup> term, we tried to enhance the AR marker's preciseness and the networking protocol. We also developed another game to demonstrate the Augmented Reality concept.

We can see Augmented Reality an interesting and exiting field to develop. It is a new experience implementing this technology on iPad. By using AR technology in gaming industry, level of interactive between players and computer graphics can be enhanced.

## Chapter 10. Progress and difficulties

Here listed our work and progress during the first semester:

| Period                | Progress  |
|-----------------------|---|
| <b>Summer 2011</b>    | Research on topic<br>Refine on topic<br>Learn basic iOS development skill<br>Hands on Objective-C language                                |
| <b>September 2011</b> | Search for suitable SDK<br>Test SDK and modify them<br>Test on iOS device<br>Draft game design<br>Server set up                           |
| <b>October 2011</b>   | Hands on OpenGL on iOS<br>Server and network part testing<br>Build prototype  |
| <b>November 2011</b>  | Integrate client and server part together<br>Modify program prototype<br>Refining final design<br>Prepare for 1 <sup>st</sup> term report |
| <b>December 2011</b>  | Evaluate first semester product<br>Draft design   |
| <b>January 2012</b>   | Update SDK and continue coding  |
| <b>February 2012</b>  | Network part modification<br>New game - Pong development  |
| <b>March 2012</b>     | Interface design and modification<br>Game development   |
| <b>April 2012</b>     | Refining final product<br>Prepare for 2 <sup>nd</sup> term report   |

## 10.1 Difficulties and challenges

### Learning Objective-C language and iOS programming

Objective-C is the main language for developing applications on iOS. It is a reflective, object-oriented programming language. It also adds some smalltalk style syntax which makes it confusing. It also requires much effort to learn iOS programming before we would actually start our working on project. There are much API documentation needed to be gone through for real understanding the mechanism behind.

Therefore we spent quite much time learning and digging into the details of iOS programming.

### Searching and testing SDKs

At the beginning, we have searched a few AR libraries. Since we are inexperienced with the AR field, we have no idea which one is most suitable for us. Finally, we have chosen the Qualcomm AR SDK because it has most features that we want to have in our project. However, the SDK is quite complicated at first sight and we do not understand how it works. We tried to compiled the sample files and solved a number of errors such as linking error.

### Simple trials and testings on iPad

The two iPads used in this project are the first Apple product we had ever touched. We were impressed by the user-friendliness of Apple devices. But before we can run our first app, we must install developer profile files to development. The whole process is not a simple task and we had spent some time on it.



### **Server set up**

We was planning to set up a server on CSE Department server, but during the process we found that it requests CSE VPN for outside world to connecting to servers with the department. However there is no support for SSL or CISCO VPN on iPad, so that we cannot connect our iPad to the department machine deriectly.

### **Game design and implementation**

The hardest part of this project is to implement a real game with the idea of remote augmented reality multiple users' interaction. In the second term, we have designed a new Pong game to demonstrate how AR can be applied on game design. We could not think about a highly creative idea but sme general ones which can demonstrate the effect.

# Chapter 11. Evaluation

---

We have some review and reflection on the product and progress of our Final Year Project throughout the year. Here we summarize them below:

## 11.1 Stabilized marker tracking

As mentioned in experiment part, the marker tracking algorithm is yet to be stabilized. Effort has been put into improving the accuracy of the marker tracking. The newer version of AR SDK has also paid an important role in stabilizing the marker detection process.

## 11.2 Network connection

The network connect is now using HTTP requests technique, which requires a connection each time. Server's ability on handling data has been improved.

## 11.3 User Interface

Apple Inc. emphasises much on UI design and user experience much. As we are now developing an application on iPad, it is nice to enhance user experience by improving the UI.

## 11.4 Assist with Ipad accessories

There are some assistant devices on iPad which we may also consider using in order to take advantage in the project, such as gyroscope, assisted GPS, ambient light sensor ,etc. We think these are great opportunity to extend our project to a better level if we try to make good use of these devices.

### **11.5 More on Game design**

Due to our limited techniques on 3D graphics, there is much room for improvement. For example a more complex 3D game like snooker or Jenga which may demonstrate the technique better.

### **11.6 Investigate possibility for more clients**

The system is now designed to support two clients only. However it would seem more interesting to support more users at the same time. The server architecture is really for multiple users, but some code should be upgraded in order to achieve this.

## Chapter 12. Acknowledgement

---

Heartfelt thanks must be expressed to our final year project supervisor:

**Professor Michael Lyu** for his resources provided and valuable advices on our project. Professor Lyu has given us guidance and reminders though out the project process.

Besides, we would also like to thank the research staff team - **Mr.Edward Yau** and **Mr. Un Tze Lung** in VIEW Lab for their very helpful assistance on setting up facilities for our project. We also thank for their brilliant ideas which really gave us some insights and inspirations.

# References

---

List of references:

- [1] Daniel Chun-Ming Leung, Pak-Shing Au, Irwin King, and Edward Hon-Hei Yau. 2007. Remote augmented reality for multiple players over network. In *Proceedings of the international conference on Advances in computer entertainment technology (ACE '07)*. ACM, New York, NY, USA, 220-223.  
DOI=10.1145/1255047.1255094 <http://doi.acm.org/10.1145/1255047.1255094>
- [2] R. Azuma, "A Survey of Augmented Reality," *Presence: Teleoperators and Virtual Environments*. vol. 6, no. 4, Aug. 1997, pp. 355-385.
- [3] J.P. Rolland, L.D. Davis and Y. Baillot, "A Survey of Tracking Technologies for Virtual Environments," *Fundamentals of Wearable Computers and Augmented Reality*, W. Barfield and T. Caudell, eds., Lawrence Erlbaum, Mahwah, N.J.,2001, pp. 67-112.
- [4] Taehee Lee, Hollerer, T., "Multithreaded Hybrid Feature Tracking for Markerless Augmented Reality", *Visualization and Computer Graphics, IEEE Transactions on*, On page(s): 355 - 368, Volume: 15 Issue: 3, May-June 2009
- [5] FIALA, M.. ARTag, a fiducial marker system using digital techniques. In *Proc. of Computer Vision and Pattern Recognition*, vol. 2, 590–596 , 2005
- [6] PINTARIC, T. 2003. An adaptive thresholding algorithm for the augmented reality toolkit. In *IEEE Int. Augmented Reality Toolkit Workshop*. ,2003

- [7] Woohun Lee, Jun Park “Augmented foam: touchable and graspable augmented reality for product design simulation” *Bulletin of Japanese Society for the Design Science* (2006)
- [8] “Apple developer tool” <http://developer.apple.com/technologies/tools/>
- [9] “Apple press info” 2010  
<http://www.apple.com/pr/library/2010/09/01Apple-Introduces-New-iPod-touch.html>
- [10] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- [11] “OpenGL ES” <http://www.khronos.org/opengles/>
- [12] “SQLite” <http://sqlite.org/>