# Design and Implementation of a Built-in Camera based User Interface for Mobile Games

**Khoa Nguyen Tran**[*]
School of Computing, National University of Singapore

**Zhiyong Huang**[†]
Institute for Infocomm Research ($I^2R$), A*STAR, Singapore

## Abstract

In this paper, we present a novel user interface design for mobile games using its built-in camera. It is motivated by the problem of limited number of keys on the small keypad of mobile devices. We implement two different ways, with and without the use of markers, to study the effectiveness of the design in a 2-dimensional and 3-dimensional game respectively. The algorithms implemented in the design are lightweight enough so that they are executable in real-time and at the same time, able to produce realism for the games.

**CCS Categories:** I.3.4 [Graphics Utilities]: Virtual device interfaces; I.3.7 [Three-Dimensional Graphics and Realism]: Virtual reality; I.4.1 [Digitization and Image Capture]: Camera calibration; I.4.8 [Scene Analysis]: Motion

**Keywords:** User Interfaces, Motion Detection, Mobile Games, Augmented Reality, Built-in Camera, Mobile Devices

## 1 Introduction

Nowadays, the use of mobile phones has penetrated to mass population rapidly. As reported by *Business Times* on April 7, 2007, Singapore has a mobile phone subscriber rate of more than one per person. *SingTel, StarHub* and *M1* have a combined customer base of 4.61 million for a population of *4.5* million.

Playing games on mobile phones has also been growing rapidly with the subscription. Due to the rapid development of mobile embedded hardware, mobile phone has been used beyond its original purpose of sending SMS and making phone calls, more like a Personal Digital Assistance (PDA), a Pocket PC and a Personal Entertainment Device (PED) for storing and processing information, as well as playing electronic games. Most of the recently made mobile phones have everything that is needed for a low-end laptop, e.g., a processor that matches a Pentium II speed, Gigabytes of external memory, a brilliant LCD display, a built-in camera, and satisfactory quality speaker. The latest Nokia N93 is even equipped with a dedicated graphic card. Note that more than *50%* of them have built-in cameras. Compared to a laptop PC, the number of keys on the small keypad of a mobile phone is a limit for user interface of the games, hence solving this problem is the motivation to our work. Use of build-in camera for user interface is the major idea of our work.

In this paper, we present a user interface design for mobile games using its built-in camera. We implement two different ways, with and without the use of markers, to study the effectiveness of the design in a 2-dimensional and 3-dimensional game respectively. The algorithms implemented in the design are lightweight enough so that they are executable in real-time and at the same time, able to produce realism for the games. The contributions of our work are summarized as:

(1) the design and implementation of user interface using the built-in camera of mobile phone,

(2) implementation of a 2D game without the use of marker for the built-in camera based the user interface, and

(3) implementation of a 3D augmented reality game with the use of 2D markers. The experimental study results showed the effectiveness of the proposed design.

## 2 Related Work

User interface design is an important area in computer science [Shneiderman 2004]. It is the design of computers, appliances, machines, mobile communication devices, software applications, and websites with the focus on the user's experience and interaction. The goal is to make the user's interaction as friendly as possible.

Other related work includes [Rekimoto 1996, Harrison et al. 2005], where the tilt input was used for navigating menus, maps, and 3-D scenes and for scrolling through documents and lists respectively.

A prototype TinyMotion was proposed for detecting a mobile phone user's hand movement in real-time by analyzing image sequences captured by the built-in camera [Wang et al. 2006]. However, we focus more on the efficiency of the computing in deriving the hand motion and its application to mobile games. The use of markers in the built-in camera based user interface for 3D game is also different from their work.

The most relevant work to our 3D part is [Möhring et al. 2004]. A similar implementation was presented in which the markers consist of four non-coplanar points and three colored axes that intersect at the origin. In contrast to our design that uses binary color-code, the colored blob features were used to encode maker's ID. The colored markers are harder to produce and more sensitive to the variation of light intensity.

## 3 Our Work

In this section, we will start to describe the technical details of our work. First, we present the design of built-in camera based user

[*] khoanguyen@nus.edu.sg
[†] zyhuang@i2r.a-star.edu.sg


GRAPHITE 2007, Perth, Western Australia, December 1–4, 2007.
© 2007 ACM 978-1-59593-912-8/07/0012 $5.00

interface. Then, the application of the interface is showcased by a 2D game and 3D augmented game implementations.

### 3.1 A Design of Built-in Camera based User Interface

The original purpose of a built-in camera in a mobile phone is to take photos and videos. The wire and wireless network allow users to upload and transmit the photos and videos, sharing them with other people.

The built-in camera can be used as user interface, in particular, to detect and use the hand motion as demonstrated in [Wamg et al. 2006]. It is particularly useful for game player on a mobile phone, where there are a limited number of keys on a small keypad. In a typical game, the most frequently used keys are "up directional key", "down directional key", "left directional key" and "right directional key". The direction of hand movement, detected by the built-in camera, can be used for the purpose. It not only adds four most frequently used keys to the game set, but also makes a player more immersive in the games.

In computer vision, optical flow is a technique used to describe image motion. It is usually applied to a series of images that have a small time step between them, for example, video frames. Optical flow calculates a velocity for points within the images, and provides an estimation of where points could be in the next image sequence. However, computing optical flow is time consuming for real-time application [Barron et al. 2006]. Further more, for the purpose of user interface, we do not need the accurate and detailed movement information represented by optical flow. Thus, we need to explore a simple, less accurate and fast method.

A simple and fast method for motion estimation usually comprises the following typical procedures:

(1) Capture two consecutive images,

(2) Filter the images to reduce noise,

(3) Filter the images to reduce noise,

(4) Divide each image into rectangle macro blocks of the same size,

(5) Locate blocks with good features,

(6) Perform block matching on every block with good features to derive motion vector for individual block,

(7) Use robust method to filter out outliners, and

(8) Compute overall motion vector.

Initially, we implemented the method above on a Nokia 7610 mobile phone. The results showed that it is not feasible to perform all seven steps due to its limited processing power. Thus, we have to either simplify or skip some steps. Through many experiments and data analysis, we worked out a method that balances between accuracy and real-time performance. It is based on corner detection and consists of four steps:

(1) Perform Grayscale conversion,

(2) Generate three multi-resolution layers and apply Gaussian noise filter,

(3) Identify four blocks at the four corners with good features, and

(4) Perform the block matching on the four blocks starting from lowest resolution and refining in higher resolution images.

In each step, we further optimize the computations. For examples, integer and bit-shifting operations are used instead of floating-point ones, looping is avoided whenever possible as it is very slow. The algorithm used in each step is also simplified to speed up the process. For example, feature detection is reduced to simply counting of pixels above a threshold. A three-step hierarchical block matching is applied and the process stops earlier upon a well-experimented threshold.

### 3.2 Implementation of the User Interface in a 2D Game

In order to show the effectiveness of the built-in camera based user interface, we implement it in a 2D game. We design the game in such a way that it has most of the characteristics of a typical game such as human-computer interactions, real-time animations, and sound effects.

Duck Hunter is a first person shooter game in which the player's mission is to shoot down all the flying virtual ducks. The user can aim the gun at the ducks by physically moving the mobile phone in the horizontal or vertical direction. By using the built-in camera based user interface, the only key needed is "5" on the keypad when the player wants to fire the virtual duck.

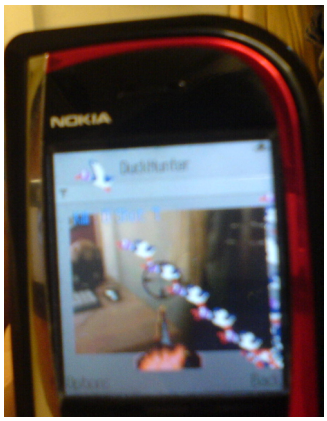



**Figure 1.** Snapshot of Duck Hunter game in action

### 3.3 Performance Analysis of the 2D Hand Motion Detection and Game

The four step method was tested on Nokia 7610.The frame rate is 8 frames per second. It has been tried on a variety of backgrounds and we find out that it is sensitive to contrast and illumination of the background. With normal lighting, it works in any conditions as long as there are some feature points in the captured background; under poorly contrasted and dark background, even though the exact distance of the hand movement is not correctly determined, the direction of motion is still correct.

The only case in which the four step process does not work is when the background is completely void of any feature points, e.g., completely plain background such as a piece of white paper, or completely dark environment such as a pitched-dark room.

The game runs at a rate of 6 frames per second which is faster than the frame rate of movement of the flying ducks. Hence, there is no jerky motion felt during playing. In this game, we tried to reduce the four step motion detection process further to only three steps:

(1) Grayscale conversion,

(2) Multi-resolution layers generation and Gaussian filter. Only layer 2 is generated, and

(3) Block matching algorithm using 4 corners on layer 0 only.

There are two reasons. First, the game deals with slow motion within a short range and it is not necessary to refine the movement at higher resolution layers and divide it by two later on. Second, the game does not require accurate movement as the player can shoot at anywhere on the duck body.

Therefore, it is not necessary to refine the motion vector. This further simplification can be applied to other similar games.

### 3.4 Implementation of the User Interface in a 3D Augmented Reality Game

For this game, we use markers to determine to camera position instead of detection of hand movement only. The markers we used consist of four black non-coplanar circles that form a 3D Cartesian coordinate system. One of them will be defined as the origin of the coordinate system. The other three are as basic points (X, Y, Z). There is a special code marker placed in one of the three planes, i.e., OXY, OXZ, OYZ. The code marker is used to identify which object should be rendered into the scene.

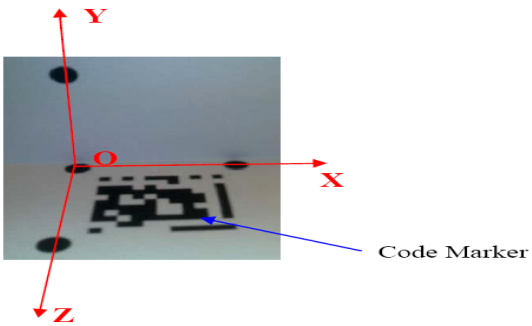



**Figure 2.** The Code Marker and the coordinate system used in our implementation

The visual code marker standard, invented by Rohs [Rohs 2004], is used as it is universal and simple to detect.

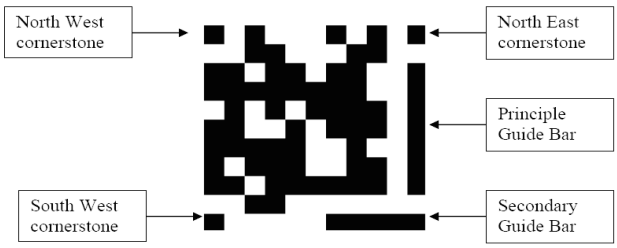



**Figure 3.** An illustration of a visual code marker and its fixed features

In Figure 3, it shows an example of the visual code marker. The standard marker has a dimension of *11x11*. There are *2* fixed guide bars and *3* fixed cornerstones. The dimensions of the principle and secondary guide bars are *1x7* and *5x1* respectively. The cornerstone is *1x1*. These fixed elements are crucial as we will use them as guiding anchors to locate the marker. There are six stages from preprocessing to projecting the 3D models to the scene,

(1) **Preprocessing:** The image is first converted to grayscale. After that, we apply adaptive threshold method proposed in [Wellner 1993] to obtain a binary image of the original image.

(2) **Connected region labeling:** A modified version of the fast connected component labeling algorithm [Stefano and Andrea 1999] is adopted. Only regions that have the size fall within a range (MIN_SIZE, MAX_SIZE) are kept. Too small or too big ones are discarded.


(a) Original image


(b) Grayscale image


(c) Regions-segmented image

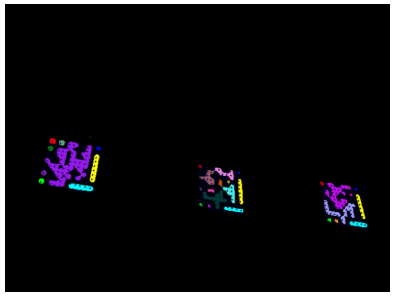
**Figure 4.** Original image is converted into grayscale and segmented into regions

(3) **Locating fixed elements:** At first, we search for all the possible candidates for principle guide bars, then for each candidate we search for its respective secondary guide bar and NE (North East) cornerstone. After that, using the secondary guide bar, we search for respective SW (South West) cornerstone. Finally, we use all the information extracted from the principle, secondary guide bar, NE and SW cornerstone, to locate the NW (North West) cornerstone. After we find a respective secondary guide bar, and 3 cornerstones for a principle guide candidate, we identify them as the fixed elements of a visual marker. Moments, the major and minor axis of each region, are computed to identify candidate regions.

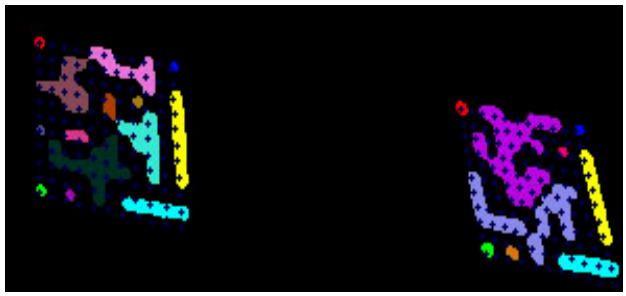
(4) **Reading code marker:** Once the marker is detected, a system of simultaneous equations (Equation 1) is solved in order to map the detected marker into the standard 11x11 marker. After that, the bit information of the detected code can then be read.

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ u_4 & v_4 & 1 & 0 & 0 & 0 & -u_4x_4 & -v_4x_4 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2y_2 & -v_2y_2 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3y_3 & -v_3y_3 \\ 0 & 0 & 0 & u_4 & v_4 & 1 & -u_4y_4 & -v_4y_4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}, \quad (1)$$

where $(x_i,y_i)$ and $(u_i,v_i)$, $i=1,2,3,4$ are the screen, and the local coordinates of the centers of 4 fixed features : *NW, NE, SW* cornerstone and principle guidebar of the detected and standard marker respectively; $a$ to $h$ are $8$ unknowns that form the projection matrix which maps the detected marker to standard one.



(a) Detected markers before bits information is extracted



(b) Detected markers after bits information is extracted, each bit is marked with a blue dot. There are 11x11 bits in each marker

**Figure 5.** Example of detected code markers and extracted bits information

(5) **Basic point detection:** The connected regions are searched to locate 4 circles which are the origin/basic points of the coordinate system. The points are sorted to determine the origin point.

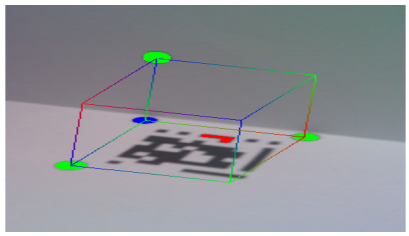


**Figure 6.** Detected origin (blue region) and basis points (green regions)

(6) **Projection of virtual 3D model into real scene:** Weak perspective projection is used to project the 3D virtual object into the real scene (Figure 7).

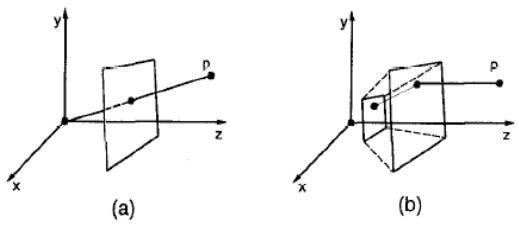



**Figure 7.** Projection models

(a) ***Perspective projection:*** Point $p = [x\ y\ z\ 1]^T$ is projected to $p = [f_x/d\ f_y/d\ 1]^T$, where $d$ is the point's distance from the center of projection and $f$ is the camera's focal length.

(b) ***Weak perspective projection:*** Point p is projected to $[f_x/d_{avg}\ f_y/d_{avg}\ 1]^T$ where $d_{avg}$ is the average distance of the object's points from the center of projection. Weak perspective projection amounts to first projecting the object orthographically and then scaling its image by $f/d_{avg}$; it is a good approximation to perspective projection when the camera's distance to the object is much larger than the size of the object itself.

Accurate projection of a virtual object requires knowing precisely the combined effect of the object-to-world, world-to-camera and camera-to-image transformations, i.e., the *model-view matrix* [Shreiner et al. 2005] as illustrated in Figure 8.

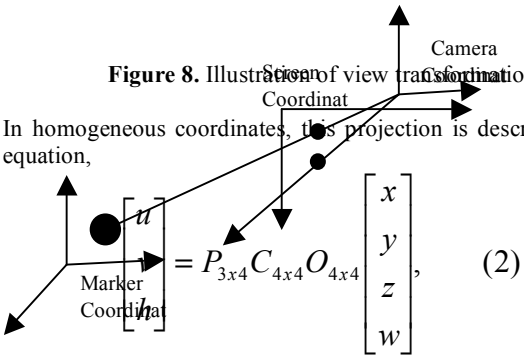



**Figure 8.** Illustration of view transformation

In homogeneous coordinates, this projection is described by the equation,

$$\begin{bmatrix} u \\ v \\ h \end{bmatrix} = P_{3x4} C_{4x4} O_{4x4} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}, \qquad (2)$$

where $[x\ y\ z\ w]^T$ is a point on the virtual object, $[u\ w\ h]^T$ is its projection, $0_{4X4}$ and $C_{4X4}$ are the matrices corresponding to the object-to-world and world-to-camera homogeneous transformations respectively. $P_{3X4}$ is the matrix modeling the object's projection onto the image plane.

The main idea of our approach is to represent both the object and the camera in a single, non-Euclidean coordinate frame defined by basis points that can be tracked across frames in real time. In particular, Equation 2 becomes

$$\begin{bmatrix} u \\ v \\ h \end{bmatrix} = \Pi_{3x4} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}, \qquad (3)$$

where $[x'\ y'\ z'\ 1]^T$ are the transformed coordinates of point $[z\ y\ z\ 1]^T$ and $\Pi_{3X4}$ models the combined effects of the change in the object's representation as well as the object-to-world, world-to-camera and projection transformations.

Non-calibrating augmented reality technique [Valino and Kutulokos 2001] uses results from the theory of affine-invariant object representations which become important because they can be constructed for any virtual object without requiring any information about the object-to-world, world-to-camera, or camera-to-image transformations. The image coordinates of the basis points contains all the information needed to project the virtual object; the 3D position and calibration parameters of the camera as well as the 3D location of the basis points can be unknown.

Hence, the transformation from marker's local coordinate systems to camera coordinates system is simplified to multiplying the *Model-View* matrix by matrix $T$:

$$T = \begin{bmatrix} u_{b1} & u_{b2} & u_{b3} & u_{p0} \\ v_{b1} & v_{b2} & v_{b3} & v_{b0} \\ & \xi^t & & \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad (4)$$

where

$$\xi = \begin{bmatrix} u_{b1} - u_{p0} \\ u_{b2} - u_{p0} \\ u_{b3} - u_{p0} \end{bmatrix} \times \begin{bmatrix} v_{b1} - v_{p0} \\ v_{b2} - v_{p0} \\ v_{b3} - v_{p0} \end{bmatrix}$$

and $[u_{p0}, v_{p0}]$ ,$[u_{bi}\ v_{bi}\ 1]^T$ $(i = 1, 2, 3)$ are the image locations of the affine origin/basis points in Equation (1).

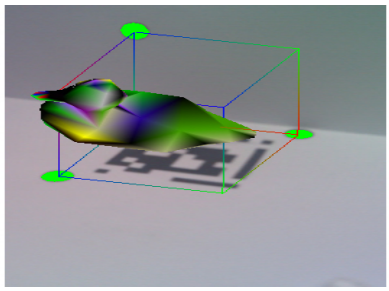


**Figure 9.** The 3D virtual duck is projected into the real scene

Notice that there is no camera calibration involved in this step. This is a tremendous improvement in speed from the typical pose estimation using camera calibration technique which is very computationally expensive and requires the users to initialize the camera before hand. This technique is much faster than camera-calibration technique. However, its accuracy is inferior.

Again, it is a matter of comprising between accuracy and real-time performance. We implemented the non-calibrating technique and found that it can produce satisfactory results on mobile devices.

One variation of the 3D Augmented Reality game is a Chinese Character Tutor. We embed a Chinese character inside the code marker as shown below,

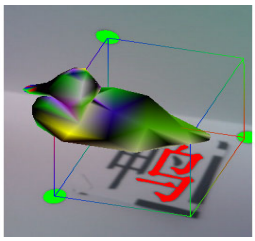
**Figure 10.** Example of pictorial code markers, the left Chinese character is Duck and the right one is Cube

The game can be used for language learning purpose. The player has a set of cards that contain the code markers with the character embedded inside, and a coordinate system that contains four circles as described previously.

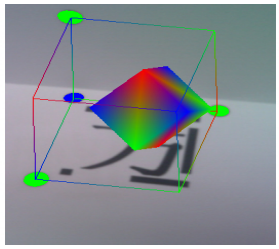
There are two empty spaces in the two perpendicular planes of the coordinate system for the player to place the card inside. The cards are designed to fit nicely inside the empty space and the player must make sure that the cards do not block any of the circles that form the coordinate system. An illustration of the cards and the coordinate system is shown in Figure 11.

**Figure 11.** The card samples and the coordinate system (four dots at Chinese character Cube). The selected card Cube is placed in the region highlighted in pink. Other cards are Chinese characters of the Rice, Person, and Duck

When a player points the phone camera to the marker, the phone will analyze and detect the code, then retrieve the respective model from its database. After that, it will superimpose the virtual object over the code marker. In Figure 12, it shows a 3D duck and a cube together with the real scene of the Chinese characters Duck and Cube in the code markers.



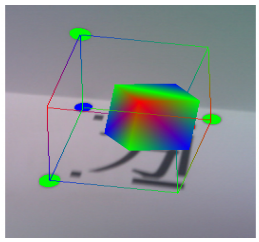
(a) Virtual duck is super-imposed over the character Duck



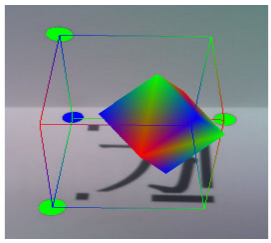
(b) Virtual cube is super-imposed over the character Cube

**Figure 12.** Illustration of the Chinese Character Tutor

The rule of the game is that, the teacher will give the picture of the duck and the player's job is to pick up a card with the character Duck among a stack of cards. Once the player selects the card, he/she will place it on the coordinate system and point the phone camera there. The player will see the 3D virtual object associated with that character popping out. The magic of augmented reality game will definitely amazes the curiosity of the player and drive him/her to learn more. The object can be animated to perform associating action e.g. a, we manage to do simple rotation animation as show in the Figure 13.



(a) The cube tilts to the right



(b) The cube rotates and tilts to the left

**Figure 13.** Animated virtual 3D cube in real space

### 3.5 Performance Analysis of the 3D Augmented Reality Game

Compared to the 2D game, the 3D Augmented Reality game is far more computationally expensive, in which higher resolution images are needed for marker detection, complicated region segmentation, recognition, and especially OPENGL ES rendering process. Hence, the frame rate plunged to *2* frames per second which is too slow for any real-time game like first-person-shooter. However, we can use it in games that do not require instant response such as the Chinese Character Tutor game.

One way to speed up the game is that we separate the code marker detection and the OPENGL ES rendering instead of performing both concurrently. First the user selects an option from the menu to start detecting the marker, once the marker to recognized, only pose-estimation and OPENGL ES rendering is performed until the user select marker detection option once again. In this way, the more computationally expensive operation is removed from the game, and the game runs many times faster. In our experiment, we have doubled the frame rate to *4* frames per second.

## 4 Conclusion and Future Work

In this paper, we present a design of built-in camera based user interface for mobile games. Our approach is carefully selecting and adapting the techniques of a desktop platform to a mobile platform. We design and implement two different algorithms with and without the use of markers and demonstrate the effectiveness of the design. The results are promising. Further more, we have implemented the design in API with all the important functions such as marker detection, pose estimation, motion detection for others to use this built-in camera based user interface for other applications.

For the future work, we are conducting usability study. We have shown a demo to public in the "Window into SoC" on May 19, 2007 School of Computing, NUS. Feedbacks from users have been summarized and used for the improvement of our design.

Finally, we would like to thank the comments from Graphite 2007 reviewers.

## References

K. L. BARRON, D. J. FLEET, and S. BEAUCHEMIN, Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1), 1994, pp. 43-77.

K. L. HARRISON, K. P. FISHKIN, A. GUJAR, C. MOCHON, and R. WANT, Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. *In Proc. of CHI'98*, April 1998, pp. 17–24.

M. MÖHRING, C. LESSIG, and O. BIMBER, Optical tracking and video see-through AR on consumer cell phones, *in: Proc. of Workshop on Virtual and Augmented Reality of the GI-Fachgruppe AR/VR,* 2004, pp. 193-204.

J. REKIMOTO, Tilting operations for small screen interfaces. *In Proceedings of UIST* 1996, pp. 167–168.

M. ROHS, Real-world interaction with camera phones. *In 2nd International Symposium on Ubiquitous Computing system (UCS 2004)*, Tokyo, Japan 2004., pp. 74-89.

B. SHNEIDERMAN, *Designing the user interface: strategies for effective human-computer interaction*, 4th ed., Addison-Wesley, 2004.

D. SHREINER, M. WOO, J. NEIDER, and T. DAVIS, *OpenGL Architecture Review Board, OpenGL programming guide: The official guide to learning OpenGL, Version 2 (5th Edition)*, Addison-Wesley Professional; 5 edition, August 1, 2005.

L. STEFANO and A. BULGARELLI, A simple and efficient connected component labeling algorithm, *Proceedings. International Conference on Digital Object Identifier*, 1999, pp. 322-327.

J. VALLINO and K. N. KUTULAKOS, Augmented reality using affine object representations. *Fundamental of Wearable Computers and Augmented Reality*, Lawrence Erlbaum, 2001, pp.157-182.

J. WANG, S. ZHAI, and J. CANNY, Camera phone based motion sensing: interaction techniques, applications and performance study, *In ACM UIST 2006*, Montreux, Switzerland, October 15-18, 2006, pp. 101-110.

P.D. WELLNER, Adaptive thresholding for the DigitalDesk, *Xerox Technical Report*, vol. EPC-1993-110, 1993.