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

2004/2005 Final Year Project
Final Report

LYU0402

Augmented Reality Table for Interactive Card Games

Supervisor
Professor Michael R. Lyu

By
Chow Chiu Hung
Lam Hei Tat

Prepared by Lam Hei Tat
14th April, 2004

Abstract

Augmented reality (AR) is becoming popular in digital entertainment. AR improves the existing game styles and produces new generation of games.

Our Final Year Project, “Augmented Reality Table for Interactive Card Games”, aims at providing a generic platform for playing trading card games. Augmented Reality Table (ART) is a platform using augmented reality technology to provide a virtual table that visualizes the battlefield of the game to players.

Throughout this report, we will have detail discussion on our project. This report will first describe the motivation and objective of our project, and introduce the concept of Augmented Reality and trading card games. Then we explain the architecture of ART system. Furthermore, there is a detailed explanation of the implementation of ART.

Table of Content

1. Introduction

1.1. Motivation

Real world games and computer games have their own distinct strengths. Augmented Reality allows us to combine both strengths, improves existing game styles and produces new ones.

Traditional Trading Cards Games are played on a table. Players draw cards alternately, and put cards on the table to summon monsters or cast spells. However, all the actions are based on player's own imagination. Players cannot really summon a monster or cast a spell. This greatly reduces the joy of the game.

While the popularity of trading card games is growing exponentially, the complexity of trading card games are increased and the games become more strategic. However, more complicated calculations are required in games. Even an experienced player may be confused by the game rule calculation.

Therefore, we need to develop a system that players can play trading card games traditionally. On top of it, this system should also provide sound and audio enhancement in the game play. Moreover, complicated calculation can also be done by the system.

1.2. Objective

Our project aims at providing a generic platform that players are allowed to play different traditional trading card game. It includes card recognition (which card a player has put), player command detection (such as cards flipping, cards orientation changing, card selection, etc), player command validation and complicated calculation by the game rules.

Our project is designed to achieve the following targets:

- Card Detection. It detects the real cards among the virtual scene generated by the computer.
- Card recognition. It identifies a card uniquely by searching the card within the large generic card database efficiently.
- Player Command Detection. It detects the player actions, such as cards flipping, attack command, and updates the game status according to player command.
- Game Rule Calculation. Complicated calculations are computed automatically according to the game status. These include calculation of hit point, attack point of the player, etc.
- Visual effect enhanced. It produces 3D animation to visualize the game play according to the current game status.
- Sound effect enhanced. It produces sound for game events.

2. Augmented Reality

2.1. Definition

Augmented Reality (AR) is a growing area in Mixed Reality research. Mixed Reality combines the content from the real world with virtual imaginary. Augmented Reality is a subset of this where virtual content is overlaid into real objects of the world. Extending the concept of AR, it includes virtual graphics and audio. In 1994, Paul Milgram characterized Mixed Reality interfaces on his Reality – Virtuality Continuum (Figure 1).

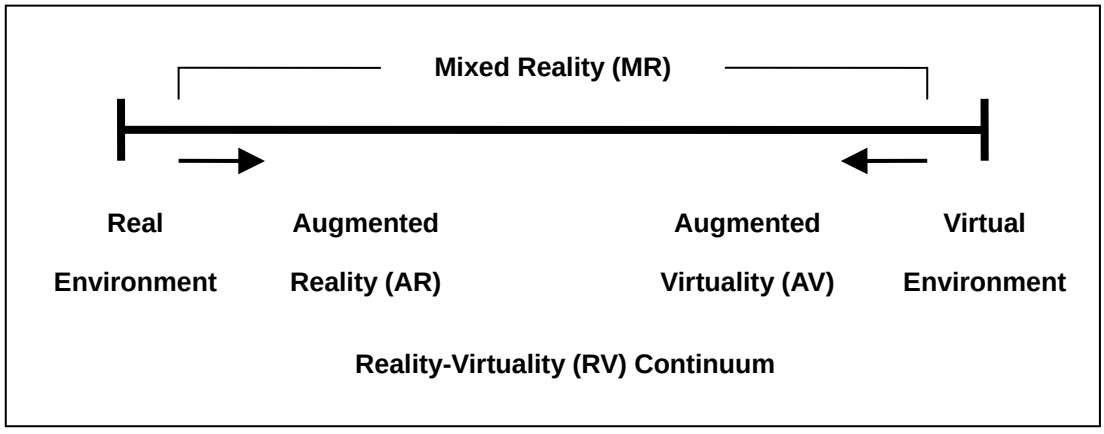


Figure 1: Milgram Reality-Virtuality (RV) Continuum

An Augmented Reality system supplements the real world with virtual objects. It means that virtual (computer-generated) content is added to the real world. An AR system has the following three main characteristics:

- Combines real and virtual objects in a real environment
- Runs interactively, and in real time
- Registers virtual objects onto the real world.

3. Trading Card Games

3.1. Overview

Trading Card Games is a kind of card games. The main difference between normal card games and trading card games is the strategic rules. Trading Card Games has a fundamental set of rules that describes the players' objectives, the categories of cards used in the game, and the basic rules for the cards interaction. Each card has additional text explaining the effect of the card in the game. This set of rules also generally represents some specific element derived from the game's genre, setting, or source material. The cards are illustrated and named for these source elements, and the card's game function may relate to the subject. For example, in “YU-GI-OH” trading card games, cards represent monsters, spells and traps.

Almost all trading card games are designed around a single basic resource. This may be the magic power, or the hit point of players. The pace of each game is controlled by this resource. Relative card strength is also often balanced by the consumption of this resource. The stronger the cards, the more the resources are consumed. These resources may be generated by some specific cards themselves, or by other means.

Players select which cards will compose their deck from the available pool of card. This pool is collected by the player. Player can trades cards among other players to strengthen his pool of cards.

Normally, trading cards game includes five kinds of action.

- Restore - make all in play cards ready for the next turn
- Draw cards - necessary in order to replenish the player's hand of cards
- Play cards - use the cards in hand to affect the game
- Attack/Challenge - the primary method of disrupting the opponent
- Discard cards - most games have a maximum hand size

3.2. An Example – “YU-GI-OH”

Among different kinds of trading card games, we choose “YU-GI-OH” as our first implementation of the system. It is because “YU-GI-OH” is a popular trading card game in Hong Kong. Besides, “YU-GI-OH” has a set of well defined game rules, which is easy to follow.

3.2.1. Cards

In “YU-GI-OH”, cards are divided into three main types of cards. They are monster cards, magic cards and trap cards. Different types of cards can be classified by their background color. The information stored on different types of cards is different (Figure 2 and 3).

Figure 2: The information stored on the monster cards.

Figure 3: The information stored on the magic cards and trap

Excepting the information stored on the cards, the action that the players act on the cards also needs to be detected. In “YU-GI-OH”, several actions need to be detect to maintain the game status.

The first action is card flipping. When the cards are put on the table upside-down, it means the player “set” the monster, or “set” the trap. Until players flip the card, the monster will be summoned or the trap will been activated.

The next action needed to be detected is the change of card orientation. If a player puts a monster card on the table vertically, it means the monster is in attack state. The monster is in defend state if it is placed on the table horizontally.

The third action is card selection. Players need to select a monster to start the attack, and select a opponent monster to be the target. In normal game, players will say the name of the two monsters to represent the card selection.

Generally, most of the trading card games are played with these actions. Therefore, we choose “YU-GI-OH” as it contains most of the main features of trading card games.

3.2.2. Game Rules

The objective of the Yu-Gi-Oh trading card game is to win a Match against your opponent. A single Match consists of 3 Duels. Each card battle against an opponent in which a win, loss, or draw is determined is referred to as a Duel.

A WIN

The player who:

- Is the first to win 2 Duels in a Match OR
- Has 1 win and 2 draws, is declared the WINNER.

A DRAW

If the Duel results are:

- 1 win, 1 loss and 1 draw OR
- 3 draws, the match is considered a DRAW.

The outcome of a Duel is decided according to the following Official Rules:

- Each player begins a Duel with 8000 Life Points.
- Life Points decrease as a result of damage calculation after battle.
- You win a Duel if you reduce your opponent's Life Points to 0. If your opponent reduces your Life Points to 0, YOU lose!
- If you and your opponent both reach 0 Life Points at the same time, the Duel is declared a DRAW.
- If either player's Deck runs out of cards during a Duel, the first player unable to draw a card is declared the LOSER. Bearing this in mind, a good duelist should make every card count.
- If at any time during the Duel you hold the following cards in your hand, you instantly win the Duel:

Phases of the Game

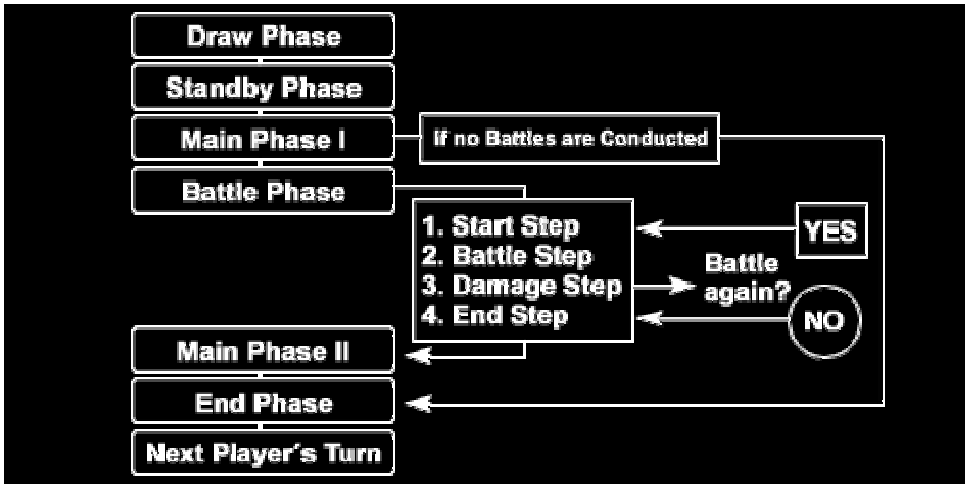


Figure 5: The game flow of the game phases of YU-GI_OH.

A. Draw Phase

During this phase, you are required to draw 1 card from the top of your Deck. A player who is out of cards and unable to draw during this phase is declared the loser.

B. Standby Phase

If there are any cards in play on the field that specifically state that certain actions must be taken during this phase, these must be dealt with prior to entering the Main Phase. Refer to the cards for specific details regarding the actions to be taken. If there are no such cards in play, proceed to Main Phase 1.

C. Main Phase 1

During this phase, you may Set or play Monster, Magic, and/or Trap Cards. Keep in mind that you may not exceed the 5-card limit for the Monster Card Zone or the Magic & Trap Card Zone.

D. Battle Phase

Once attack preparations have been made in Main Phase 1, you enter the Battle Phase. If you do not wish to conduct a Battle Phase, your turn proceeds to the End Phase.

E. Main Phase 2

When the Battle Phase is over, the turn proceeds to Main Phase 2. As in Main Phase 1, you may Set or play Monster, Magic, and/or Trap Cards. Remember that you are allowed to change the Attack or Defense Position of each monster or

perform a Normal Summon or a Set only ONCE PER TURN. Also keep in mind that you may not exceed the 5-card limit for the Monster Card Zone or the Magic & Trap Card Zone.

F. End Phase

Announce the end of your turn. If your hand contains more than 6 cards, discard to the Graveyard until only 6 cards remain in your hand. The opposing player then begins his/her turn with the Draw Phase.

G. End of the Due

Repeat Phases 1 through 6 in alternating turns until a winner is decided.

4. Architecture of ART

4.1. Introduction

To archive our objective, we need to build a platform for players to play trading card games. This includes hardware setup and software architecture. Our Project, ART, is one of the solutions to this problem.

Augmented Reality Table (ART) is a platform using augmented reality technology to provide a virtual table for players. The three major hardware components are a computer, a plasma monitor and a camera. The computer is used to process the input, maintain game rules and generate visual display. The camera is mounted overhead, while the plasma monitor is placed horizontally as a table. The overhead camera captures all the information shown on the screen of the plasma, the cards played on the plasma monitor, and the input command from players as well.

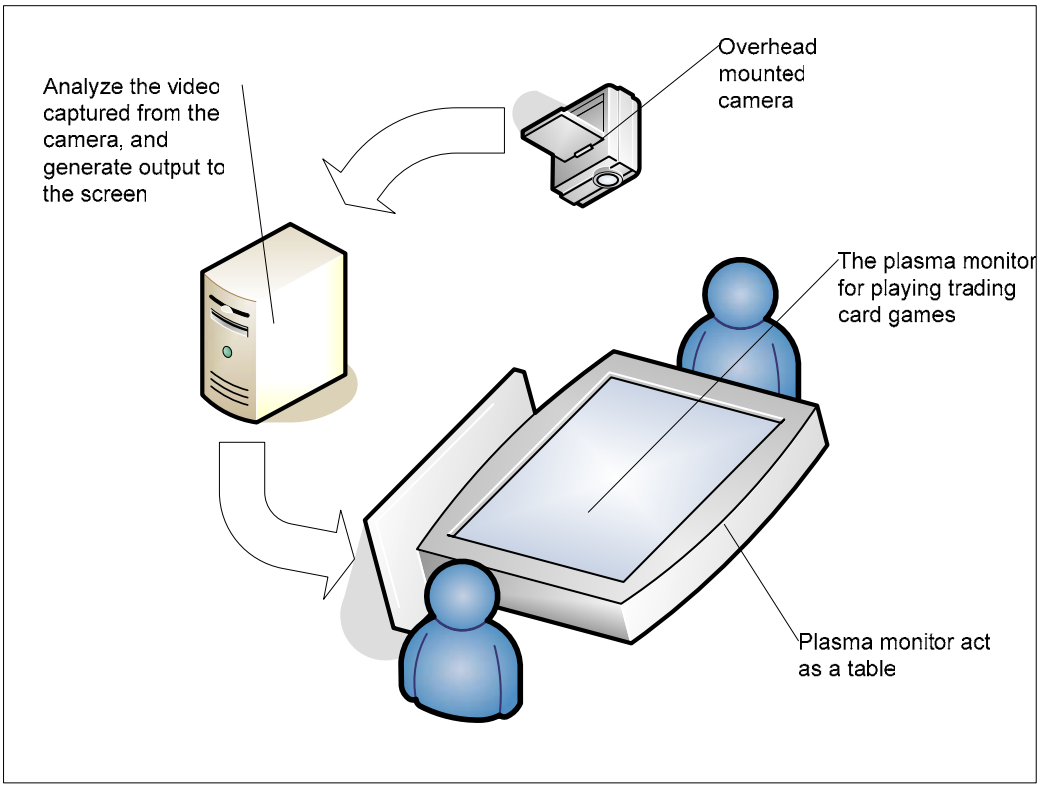


Figure 4: The overview setup of ART

4.2. Hardware Setup

Our system is setup by overhead mounted camera, plasma monitor, and a spot light.

4.2.1. Overhead mounted camera

The overhead camera is setup for capturing events in the arena. It is the only input device in the setup. Therefore, all input information would be capture by this camera. This camera can be a USB web camera, or a digital camera. Digital camera provides higher resolution, while USB web camera is more popular. The video captured will be sent to the computer analyzed. The game status and game events will be determined from the video captured.

In our implementation, we have chosen digital camera for higher resolution.

4.2.2. Plasma monitor

The plasma monitor is placed horizontally to act as a table. This setup provides a platform for players to play trading cards games traditionally. The virtual game map is shown on the screen. Besides being a table, the screen of the plasma monitor displays the virtual environment to players. This virtual environment includes the visual enhancement and sound enhancement for the game play. 3D animation would be displayed on the screen and sound would be produced by the speaker of the plasma monitor, if there are events of the game detected.

Besides, according to the game status determined from the camera, calculation of game rules can be automatically done by the computer. The hit points, attack points and defend points are calculated and shown on the screen, so that players need not to handle the complex calculations by themselves.

4.2.3. Spot Light

The lighting setting is critical for the ART system because it uses only a camera as input. We cannot use the original room lighting in the laboratory as their reflection from the flat plasma table results in bright region in the perceived image. Therefore, we have tried to use a spot light for this purpose, but the lighting is uneven such that the region far from the light is much darker than that near to the light. Finally, we set the spot light to a higher and further position, and cover the light with translucent paper to make the lighting more even.

4.3. Software architecture

In our ART system, the main purpose of the system is to read the video from the camera and output the virtual scene to the display. We can divide the system into four main modules (Figure 5). They are Perception module, Generic Card Game Database module, ART Card Game Core, and Game Enhancement module.

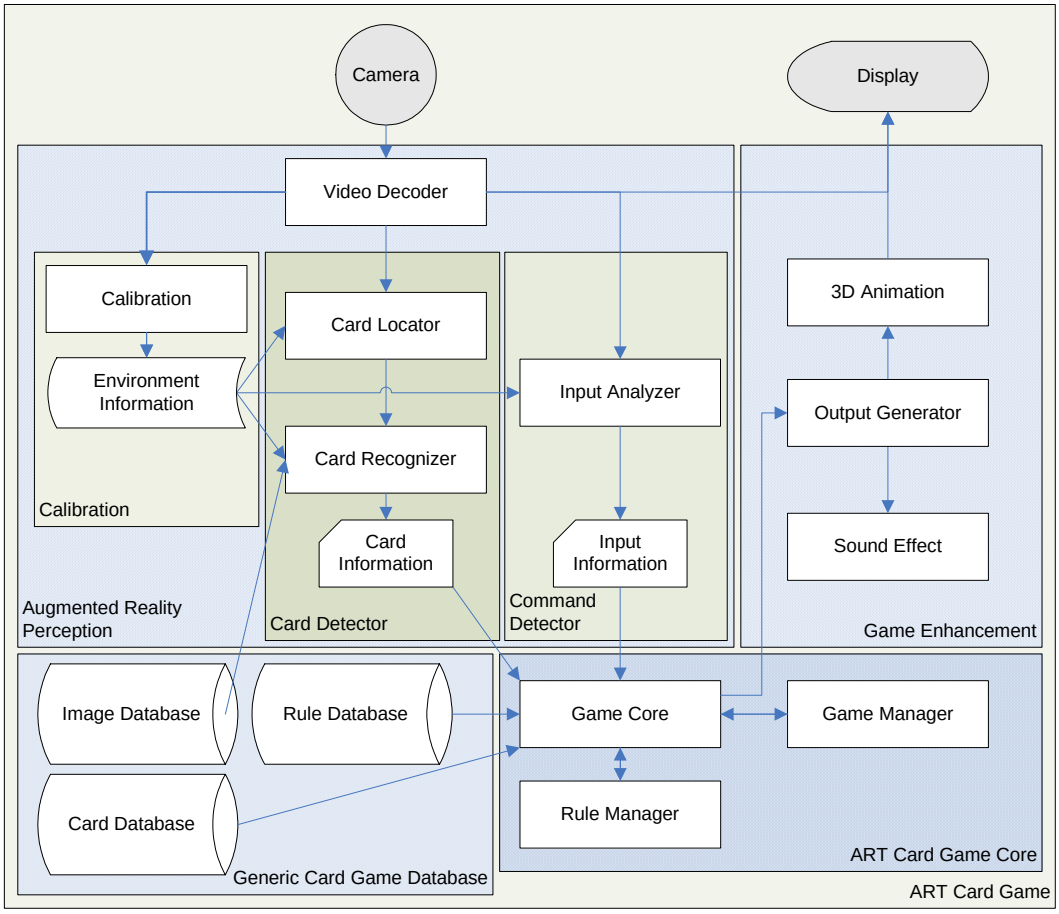


Figure 5: The system architecture of ART Card Game

Video captured from the camera will first be decoded so that the video can be process. Then this decoded information acts as an inputs to the Perception Module and Game Enhancement Module. The information extracted from Perception Module is then passed to the ART Card Game Core to update the game status. After that, the updated information is sent to the Game Enhancement Module. Finally the module produces the visual display for players.

4.3.1. Perception Module

This module read the raw video from the camera. The card information and input information will be extracted. Then the are sent to the ART Card Game Core. This module is divided into three part, they are Calibration, Card Detector and Command Detector (Figure 6).

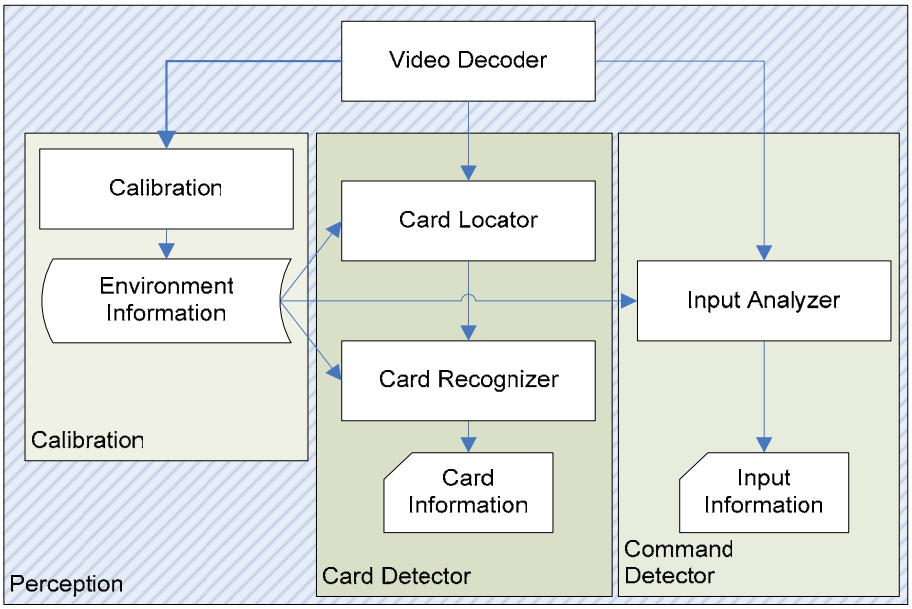


Figure 6: the architecture of Perception Module

The input format of the raw video is encoded with the raw format provided by the manufacturer. Therefore, we need to decode it into accessible format. In our case, we decode it into RGB format.

After decoding the raw video, each part is supplied with this decoded information.

Calibration calculates several constants of the environment, and update Environment Information. The environment information provides the essential parameters for Card Detector and Command Detector.

Card Detector serves two main function, card locator and card recognizer.

Card Locator detects any cards in the screen. It simply detects cards by edge detection algorithm. Then it sends the position of the card detected to the Card Recognizer.

Card Recognizer first transforms the distorted card image into undistorted version. Then it read the images from image database. It compares this image to the transformed image. The comparing method is done by pattern recognition algorithm. If specific card is found, it writes the card information to the ART Card Game Core.

4.3.2. ART Card Game Core

The ART Card Game Core reads the card information and input information from the Perception module. This information provides the current game status. The ART Card Game Core then analyzes this information, and checks the game rule database. Finally it outputs the game information to game enhancement module for output.

4.3.3. Generic Card Game Database

To develop a platform that can play different kind of trading card games, the generic card game database acts an important role. Basically there are three databases inside, the Card Image database, the Card database and the Game Rule database (Figure 7).

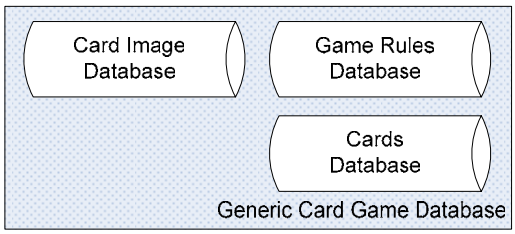


Figure 7: overview of Generic Card Games Database

The Card Image database provides the basic card pattern and card features for card recognition in Perception module. These basic card patterns are the same as the real cards,

and the card features is used for constructing search tree of card recognition.

Card database contains all card information that appears on the card. The special effects of the card are also defined in this database.

Game Rule database contains the basic game rules and the relationship betweens the cards. The game rules are defined as the relationship of cards, so we can define how the cards interact with each other easily.

4.3.4. Game Enhancement Module

The Game Enhancement module read the game information from ART Card Game Core, and then generates corresponding display and sound effect (Figure 8).

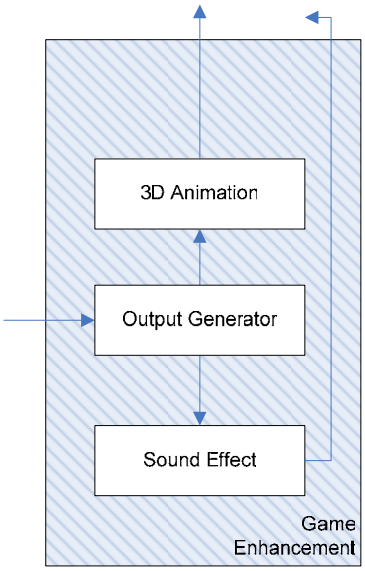


Figure 8: overview of Game Enhancement Module

5. Implementation

5.1. Overview

The hardware setup for the ART system consists of a computer, a plasma monitor and an overhead camera. However, a much simple setup is used for experimental purpose during current stage of development. We use a LCD monitor instead of the plasma monitor. A tripod, with the overhead camera fixed, is placed over the LCD monitor.

The software implementation of the ART system uses Microsoft Visual C++ 6.0. It is built on top of Microsoft DirectX 9.0 SDK, where DirectShow is used for video stream input, and Direct3D is used for graphics display. DirectX SDK would be introduce shortly.

5.2. DirectX SDK

Microsoft DirectX is a set of low-level application programming interfaces (APIs) for creating games with high-performance multimedia applications. It includes support for three-dimensional (3-D) graphics, sound effects and music.

Microsoft Direct X has two special features. The first feature is that DirectX can directly access the hardware, because many games need direct access for high performance. The second feather is that DirectX provides device independent through the hardware abstraction layer (HAL). Programmers can access the hardware through the DirectX interface. Under this interface, all hardware look like the same.

In our implementation, we have use Microsoft DirectX 9.0. Microsoft DirectX 9.0 is made up of eight components. We develop our project by using three of them. They are DirectX Graphics, Microsoft DirectMusic, and Microsoft DirectShow.

5.2.1. DirectX Graphics

DirectX Graphics combines the Microsoft DirectDraw and Microsoft Direct3D components into a single application programming interface (API). It provides the API for all graphics programming. The Direct3D API connects with HAL device and the Device Driver Interface (DDI). HAL devices provide hardware acceleration based on the supported feature of the graphics card.

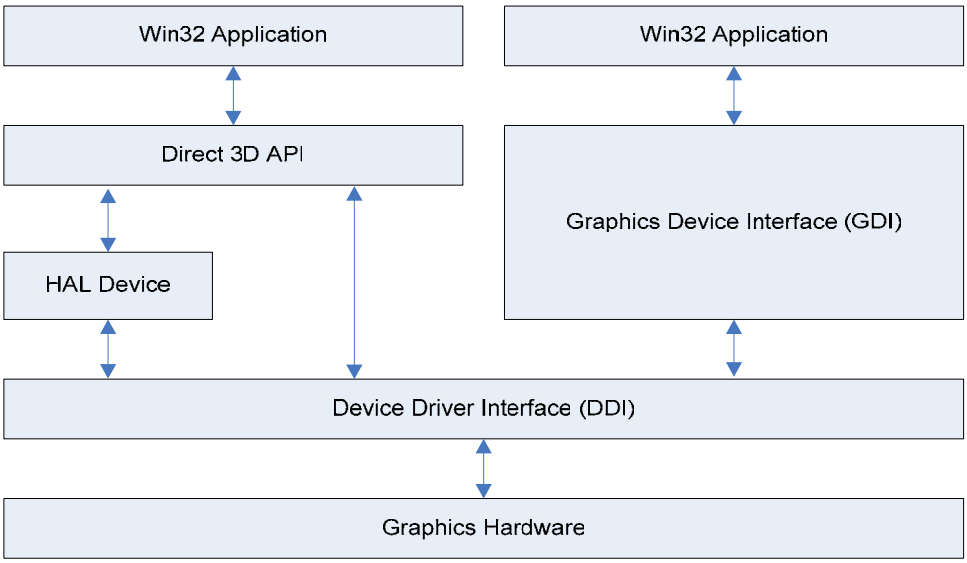


Figure 9: the system integration of Direct3D

DirectX Graphics provides many COM API interface and functions to generate 3D graphics, with win32 application written by C or C++. We can easily build a 3D virtual scene with high performance. Now we will have a brief description on the interface that we have used to build our system.

IDirect3D9 Interface

The IDirect3D9 interface contains methods to create Direct3D objects and set up the environment. This interface also includes methods for enumerating and retrieving capabilities. It is the interface to create the Direct3D device.

IDirect3Ddevice9 Interface

This interface is the basic render object in Direct3D. It performs the basic primitive-based rendering. It controls the render status and the transformation status for rendering. Besides, resources are created in this interface, such as index buffer and vertex buffer. Besides, this interface can work with system-level variables.

Idirect3DvertexBuffer9 Interface

This interface is to create the buffer to store vertices. In 3D environment, 3D model are made up of vertices. Each vertex has additional properties, such as texture coordinates and normal vector. Before we draw any 3D model, we need to create a buffer to store the vertex information.

Idirect3DindexBuffer9 Interface

This interface is to create the buffer to store indices representing the index of a vertex in vertex buffer. Indices are the sequence of several vertices. We need these indices to draw a 3D model.

Idirect3Dtexture9 Interface

The Idirect3Dtexture9 interface is create textures for the system. Textures are the surface of a 3D model. As there are more texture than our display memory, this interface provides functions to manipulate the texture resources.

5.2.2. Microsoft DirectMusic

The basic function of DirectMusic is playing sounds. It supports loading and playing sounds from files or resources in MIDI, WAV, or DirectMusic Producer run-time format. These sounds can be played simultaneously.Direct Music provides a simple API for basic task.

5.2.3. Microsoft DirectShow

Microsoft DirectShow is an architecture for streaming media on the Microsoft Windows platform. It provides high-quality capture and playback of multimedia streams. Since it supports a wide variety of formats, different camera can be access with the same interface. DirectShow is integrated with other DirectX technologies. It automatically detects and uses video and audio acceleration hardware when available, but also supports systems without acceleration hardware. Besides, DirectShow simplifies media playback, format conversion, and capture tasks. We can perform media processing simply access the DirectShow interface.

DirectShow uses a modular architecture, where each stage of processing is done by a COM object called a filter. DirectShow provides wide range of filters for applications to use. In Figure 8, it shows the overview of DirectShow system, it contains three kinds of filters. Source Filters captures the video from files or camera device, even in internet. Then it transforms the captured format to Direct Show media format, ImediaSample. In our implementation, we builds a filter that reads an ImediaSample object, and provides the information of our application.

DirectShow provides COM API, including different filters. We will have a brief description to COM interface that we have used to build our application.

IGraphBuilder Interface

The IgraphBuilder interface provides methods for us to build a filter graph in our application. IgraphBuilder Interface provides basic operations of building graph, such as adding a filter to the graph, or connecting two pins.

ICaptureGraphBuilder2 Interface

The IcaptureGraphBuilder2 interface builds capture graphs and other custom filter graphs. Since capture graph are often more complicated than file playback graphs, IcaptureGraphBuilder2 makes it easier to build a capture graph in our application.

At the beginning, we create a new instance of Filter Graph Manager and Capture Graph Builder, which implemented the IGraphBuilder and ICaptureGraphBuilder2 interface respectively. Then we initialize the Capture Graph Builder by setting its pointer to point to the Filter Graph Manager's IGraphBuilder interface (figure 10).

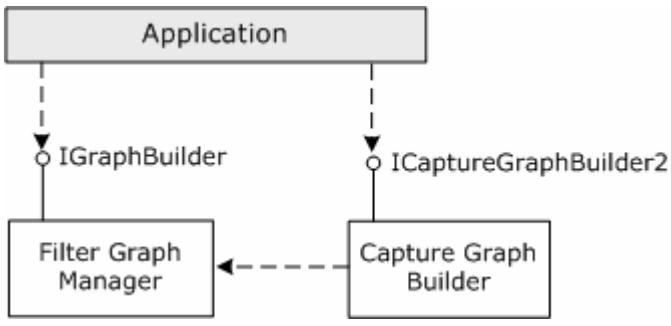


Figure 10: The architecture of ICaptureGraphBuilder Interface

IMediaControl Interface

This interface is to control the flow of data through the filter graph. We use the method in this interface to run, pause, and stop the graph. It is implemented in the Filter Graph Manager.

IMediaEvent Interface

The IMediaEvent interface retrieves the event notifications and overrides the default events handler of Filter Graph Manager. This interface handles events such as end of stream, or rendering error. We implement the mechanism of error handling in Filter Graph Manager and it will be override to this interface. This interface is implemented in the Filter Graph Manager.

IMediaSample Interface

The IMediaSample interface sets and retrieves properties on media samples. This media sample contains a block of media data, and these data can be shared among filters by shared memory. The DirectShow filters use this interface to set properties on samples, and deliver the samples to a downstream filter. The downstream filter uses this interface to retrieve the properties and read the media data. In the filter we implemented, we use this interface to modify the data, and pass it to the downstream filter.

IIPDVDec Interface

This interface is implemented in the DV Video Decoder filter, provided by DirectX SDK. This filter can decode the format of digital camera into an accessible format. In our implementation, we changed the DVSD format (Sony mini DV camera format, which is provided by Sony Corporation) to 24 bit RGB format.

5.3. Perception Module

The system receives user's input purely from the video captured by the camera. This requires reading from the video stream, and analysis on each frame of the video.

Since the Game Enhancement Module would produce real time visual effect for the game play, efficiency of the Perception Module plays an important role. If this module is not efficient enough, the processing in the Game Enhancement Module would be influent and delay, which would in turn affect the display fluency visible to the game players.

On the other hand, the accuracy of the Perception Module is also important. The players' inputs must be correctly recognized. It would be a big trouble if a player put a powerful card which lead the player to win, but the system incorrectly identified it as another weak card, and cause the player to lose.

5.3.1. Some background knowledge

We would introduced some background knowledge and idea used in the perception module here.

5.3.1.1. Computer Vision

Vision allows humans to perceive and understand the world surrounding us. Computer vision aims to duplicate the effect of human vision by electronically perceiving and understanding an image. However, when computers try to analyze objects in 3D space, available visual sensors, for example, a DV camera, usually give two dimensional images, and this projection to a lower number of dimensions incurs an enormous loss of information.

In order to simplify the task of computer vision understanding, two levels are usually distinguished; low level image processing and high level image understanding.

1. Low level methods usually use very little knowledge about the content of images. First an image is captured by a sensor (e.g. camera) and digitalize. Some technique such as suppressing noise, enhances some features, detecting edge is employed afterward.

Then object are segmented and classified.

2. High level processing is based on knowledge, goals, and plans of how to achieve those goals. Artificial intelligence (AI) methods are used in many cases. High level computer vision tries to imitate human cognition and the ability to make decisions according to the information contained in the image.

In our implementation, our task is to identify different cards placed in different positions. These positions are predefined as different “Card Zone”. Thus the image segmentation problem is then reduced to determine whether a Card Zone contains a card, and the exact location of the card.

5.3.1.2. Digital image processing

Image processing operations can be roughly divided into three major categories, Image Compression, Image Enhancement and Restoration, and Measurement Extraction. Image compression involves reducing the amount of memory needed to store a digital image. Image Enhancement and Restoration corrects image defects such as bad lighting. Measurement Extraction technique can be used obtain useful information from the image.

Our system mainly concern with Image Enhancement and Restoration and Measurement Extraction

The two-dimensional convolution operation is fundamental to the analysis of images. It ascribes a new value to a given pixel based on the evaluation of a weighted average of pixel values in a k x k neighborhood of the central pixel. The weights are supplied in a square matrix, usually referred to as the filter mask or the convolution kernel. By selection of different masks, divers operations may be performed. Fig.11 illustrates a convolution on an image.

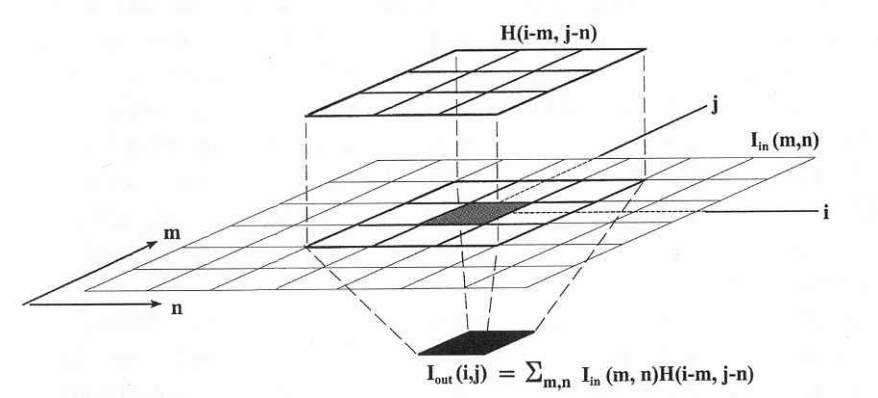


Figure 11:

5.3.2. Calibration

The process of submitting samples of known value to an instrument, in order to establish the relationship of value to instrumental output, is called calibration. It ensures that different equipment measurements correspond to same standards.

5.3.2.1. Color calibration

It is very importance that calibration allows the system to perform more accurate image matching to identify an input card perceiving from the overhead camera. Figure 12 shows the different between a card capture by the camera (a) and a reference card in our database (b). The color of the captured image varies with environmental lighting condition, and depends on the type of input device. Therefore, every time these conditions change, calibration for the new environment is necessary to maintain the accuracy and reliability of the system.

Figure 12: The captured card image (left) and the scanned card image (right).

5.3.2.2. Environment calibration

In addition, we can benefit from the characteristics of the system setup. Several assumptions are made base on these characteristics:

- The position of the overhead camera and table are fixed;
- The camera is approximately placing right above the table;
- Cards can only be placed in predefined card zones.

By these assumptions, we can set the position of card zones and search window at the calibration stage. Card area threshold for each card zone are also calculate in this stage.

Some of the implementations in the following session take advantage from these environment information set in the calibration stage.

5.3.2.3. Calibration Procedure

When the ART System starts, the calibration procedure is activated as follows:

First, the table screen shows a calibration mat (Figure. 13).

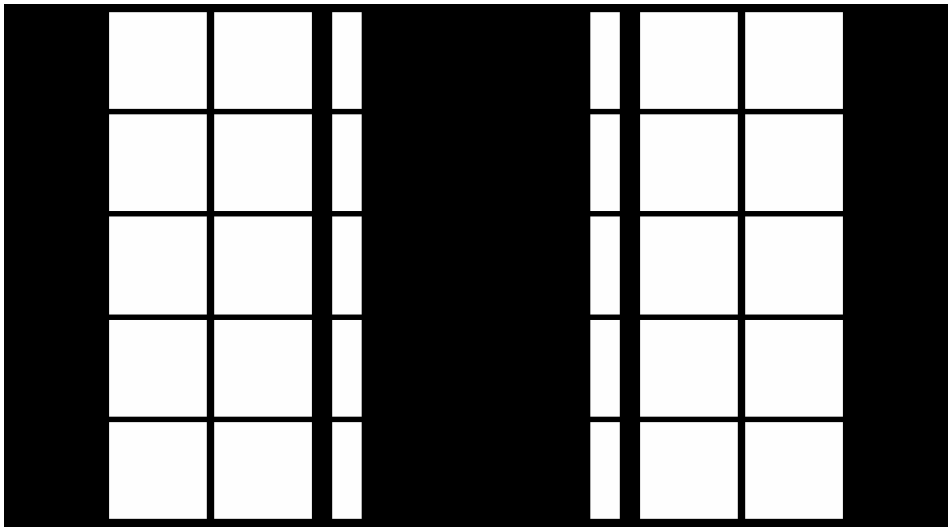


Figure 13: The calibration mat

Then the overhead camera captures the image and check for any error in the environmental setup. In this stage, we apply a threshold to the perceived image and obtain a binary image by the follow formula:

$$I(x, y) \begin{cases} \geq \Phi_1 & \textit{set to white} \\ < \Phi_1 & \textit{set to black} \end{cases}$$

where Φ_1 is the threshold for binary image.

Figure 14 and figure 15 show the perceived image and the corresponding binary image. We can see that there is some distortion in the perceived image and must be calibrated for more accurate input detection.

Figure 14: The perceived image

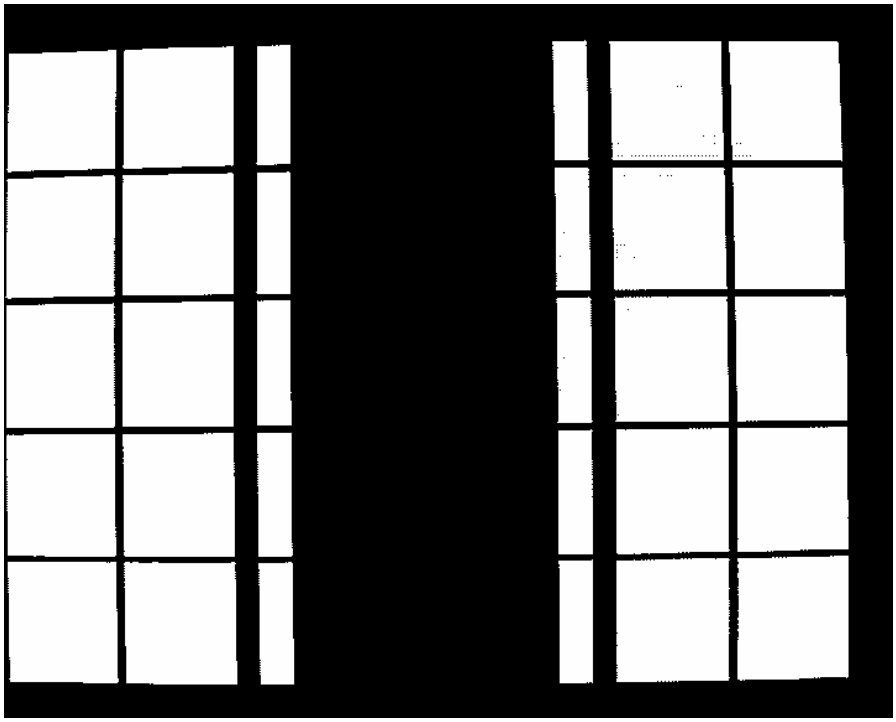


Figure 15: The binary image from the perceived image

After the binary image is obtained, Canny algorithm is applied to extract the edges, followed by Contour detection, and the rectangles corresponding to the card zones and buttons are obtained. The technique used to extract the search window will be discussed in detailed later in Card Detector Section.

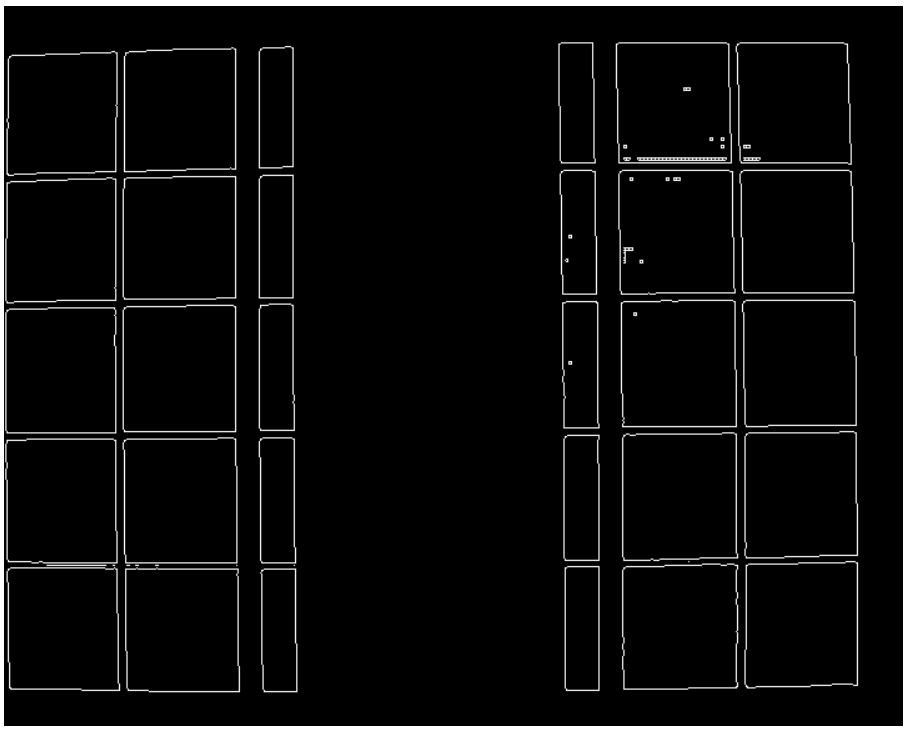


Figure 16: show the result after applying Canny

Finally, the extracted rectangles are sorted by their coordinates and assigned to appropriate card zones or buttons accordingly. If there are errors, the calibration step will start again. These errors may due to the error in extracting the rectangles, or the environment setup is not set properly.

The system gives ten times trial for the calibration to take place, and announce the players if the calibration failed.

5.3.3. Search window locator

The system receives player card input by searching for cards inside a given frame captured from the camera. To search every perceiving frame and every pixel of the frame is time exhausted. Some heuristics is needed to avoid searching in any unrelated regions and search only regions corresponding to player input.

We use search window to minimize the search required. A search window defines a small region for the search to operate. By locating minimum number and size of search windows which are suspected to contain player inputs, we can reduce both frequency and time to search.

5.3.3.1. Predefined search windows

In our system, the position of the camera and the table is fixed. In addition, the regions which player can put the card are predefined as Card Zone. Thus, the position of each Card Zone is fixed with respect to the camera perspective. So the predefined search windows for card detection are defined as different Card Zone regions.

5.3.3.2. Locate search window

Any player input action introduces changes to the perceiving frame. As the camera and the table are fixed, these changes cannot be caused by movement of the camera or the table. So these changes must be the result of some player input. They can be detected by comparing the current frame with the previous frame as follows:

$$I_{diff}(x,y) = \left| I_{current}(x,y) - I_{previous}(x,y) \right|$$

where $I_{diff}(x,y)$, $I_{current}(x,y)$ and $I_{previous}(x,y)$ are the pixel value of pixel (x,y) on the resulting different image, current frame, and previous frame respectively.

(Figure 17) shows the result image of the comparison. It is the absolute different between current frame and previous frame.

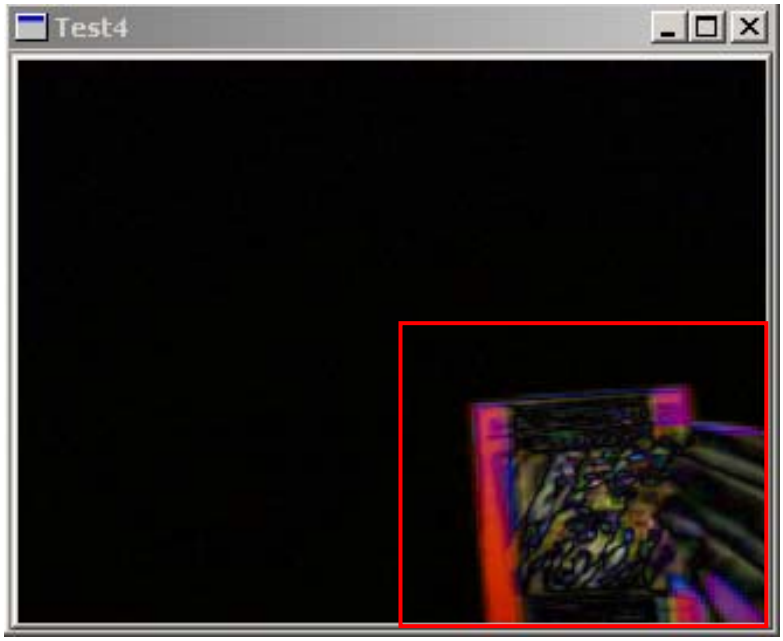


Figure 17: Resulting image of comparing current frame and previous

The next step is to locate the search window. Search windows with only little changes are rejected because these changes are probably caused by noise of the input. Two thresholds are set for this purpose.

A pixel is said to be changed if

$I_{diff}(x, y) > \Phi_1$ where Φ_1 is the threshold of changed pixel.

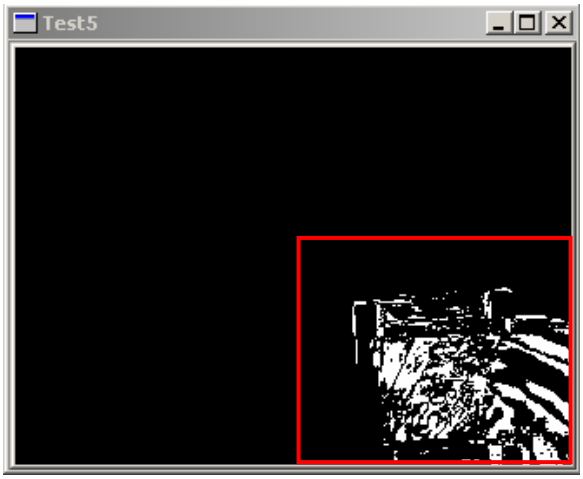


Figure 18: Binery image indicating change of pixels (white region).

(Figure 18) shows the binary image after applying the threshold of changed pixel to previous different image. Changed pixel take the value of 1 and unchanged pixel take the value of 2.

A search window is said to be changed if

$\sum I_{diff}(x, y) > \Phi_2$ where Φ_2 is the threshold of changed search window.

This approach selects only search window where player input are most likely taking place, and the search window is small enough to contain any necessary information for detecting an input. It minimizes the number of search window to process.

5.3.3.3. Activate search window

Further improvement for locating search window is to reduce the frequency of searching. The idea is to activate a search window when it just stops changing.

For an interactive card game, a card input means a player place a card on the table. The input is finished at the time when the player withdraws his hand after putting the card. Searching before the action finished is completely meaningless. To determine the input, search only at the point when the input action just finished is enough.

The system maintain the state of change of each predefined search window, and activate it only when its state transits from “change” to “unchanged”, because this implies the action is just finished. The activated search window is then search once to find any player input.

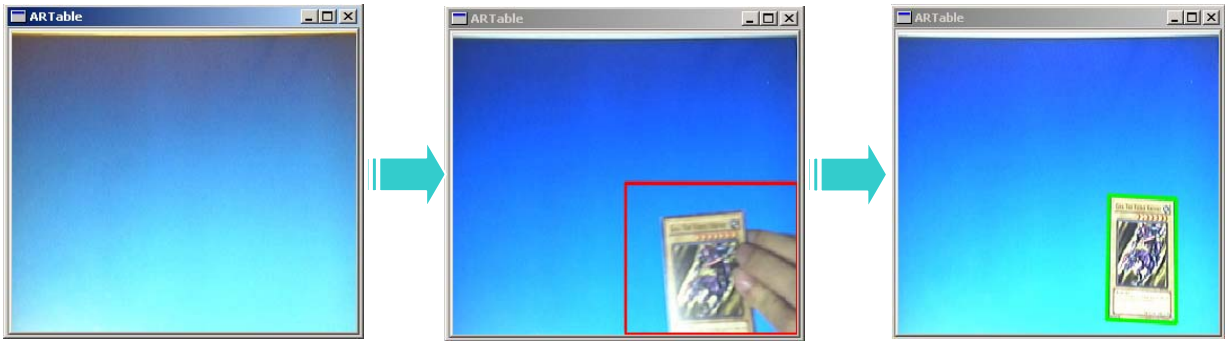


Figure 19: The steps of detecting an input by search window.

(Figure 19) shows the step of detecting an input by search window. (a) all search window are unchanged. (b) The search window at the lower right corner (marked as the red box) detects changes. (c) When the content of the search window just stop to change, the search window is activate and a card is detected (marked as the green box) by performing a search on this search window.

5.3.4. Card locator

When a search window suspected to contain player input is detected, the next step is check whether it contain a card input. If there is, the position of the four corner point of the card is located for further process.

These take several steps to find the boundary and the position of the card.

5.3.4.1. Detect edges

The first step is to find all edges from the image region defined by the search window. These edges are then used to found the card boundary. The Canny edge detection algorithm is used in this step. We will briefly explain how the algorithm works.

Canny Algorithm

A good edge detection algorithm should have low error rate, well localized edge points, and single edge response. The Canny edge detection algorithm is known to many as the optimal edge detector according to these criteria. The algorithm works in a 4-stage process

Gaussian convolution

The first step is to filter out any noise in the original image and smooth the image before trying to locate and detect edges. A Gaussian filter mask (Figure 20) has the form of a bell-shaped curve, with a high point in the center and symmetrically tapering sections to either side. Application of the Gaussian filter produces, for each pixel in the image, a weighted average such that central pixels contribute more significantly to the result than pixels at the mask edges.

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figure 20: An example of a 5 x 5 Gaussian filter mask.

Edge strength and direction

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The most basic definition of an edge is that it marks a change in image intensity separating distinct regions. An edge will therefore manifest itself in the derivative of the image intensity I(x,y), which can be regarded as the gradient at that point.

The Sobel operators serve to find the approximate absolute gradient magnitude at each point. The Sobel operator uses a pair of 3 x 3 convolution masks (Figure 21), one estimates the gradient in vertical direction and the other estimates the gradient in horizontal direction.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 21: A pair of 3 x 3 Sobel operator.

The edge strength of the gradient is then approximated as follows:

$$|G| = |G_x| + |G_y|$$

And the edge direction is approximated as follows:

$$\theta = \begin{cases} 0^\circ & when \quad G_x = 0 \; and \; G_y = 0 \\ 90^\circ & when \quad G_x = 0 \; and \; G_y \neq 0 \\ \tan^{-1}(\frac{G_y}{G_x}) & otherwise \end{cases}$$

Non-maximum suppression

After the edge directions are know, non-maximum suppression is applied to give a thin line in the output image represent an edge. Non-maximum suppression trace along the edge in the edge direction and suppress any pixel value that is not considered to be an edge to zero.

Hysteresis

Finally, hysteresis is applied to eliminate streaking. Streaking is the breaking up of an edge contour caused by the operator output fluctuating above and below the threshold. Hysteresis is controlled by two thresholds T1 and T2. Tracking can only begin at a point with edge strength higher than T1, then continues in both directions until the edge strength falls below T2.

Figure 22: Resulting image after applying Canny algorithm.

5.3.4.2. Find contours

After finding all edges, apply dilation to the image to thicken the edge lines and join those which is closed to each other. Then find all possible contours in the image.

5.3.4.3. Locate card

Finally, the card is located by searching all contours to find the one representing the boundary rectangle of the card. As the overhead camera is right above the table and the view from the camera is orthogonal to the table surface, we can assume that the card approximately remains its rectangular shape with respect to the perspective view of the camera.

Under this assumption, the contour for the card should satisfies the following criteria:

- composes of exactly 4 points;
- contains 4 nearly right angle corners;
- with area within certain thresholds.

Number of corner points

The first condition is easily verified by counting the number of point in the contour.

Angle between joint edge

The second condition requires to check all the four angles, and check if the smallest angle is nearly 90°, since this would implies all angles are also 90° for a rectangle with angle sum equals 360°. We compute the cosines of all the angles and find the one with maximum magnitude, which corresponds to the smallest angle. If this absolute value of cosine is smaller than a threshold, then the angle is nearly 90° and the condition holds.

The cosine of the angle is computed using vector method as follows:

$$\cos\alpha = \frac{\Delta x_1 \times \Delta x_2 + \Delta y_1 \times \Delta y_2}{\sqrt{(\Delta x_1)^2 + (\Delta y_1)^2}\sqrt{(\Delta x_2)^2 + (\Delta y_2)^2}}$$

where Δx_i and Δy_i are different between the x-coordinate and y-coordinate of the two end point of the i^{th} edge.

Area

The third condition is to check the area bounded by the contour. Since the distance between the overhead camera and the table is fixed, the area of a card is fixed for the camera point of view. This condition ensures that the contour marks the boundary of the card, not the inner rectangle of the card. If the area is larger than the lower bound threshold and smaller than the upper bound threshold, the condition is satisfied.

The area is computed as follows:

$$Area = \frac{1}{2}\begin{Vmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{Vmatrix}$$

where x_i and y_i are the xy-coordinates of the ith point.

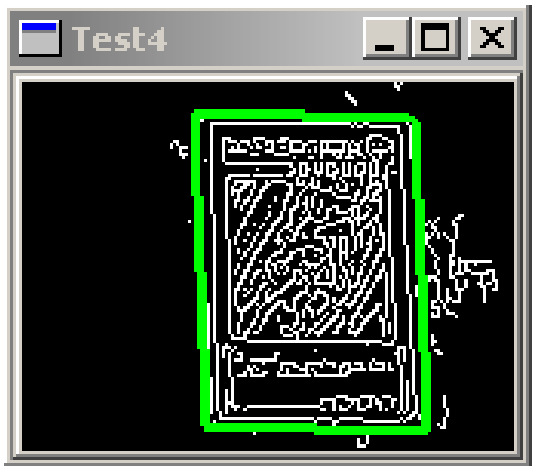


Figure 23: Result of card location marked by the green box.

5.3.5. Card recognition

Finally, the last step is to recognize and identify the undistorted card image extracted from the previous module.

5.3.5.1. Orientation

The orientation of a card in a typical card game gives different meaning to the input. It can either be lateral or vertical. The detection is very simple. We only need to check the length the bottom edge and the leftmost edge. If the bottom edge is longer, the card is in lateral position. Otherwise, the card is in vertical position

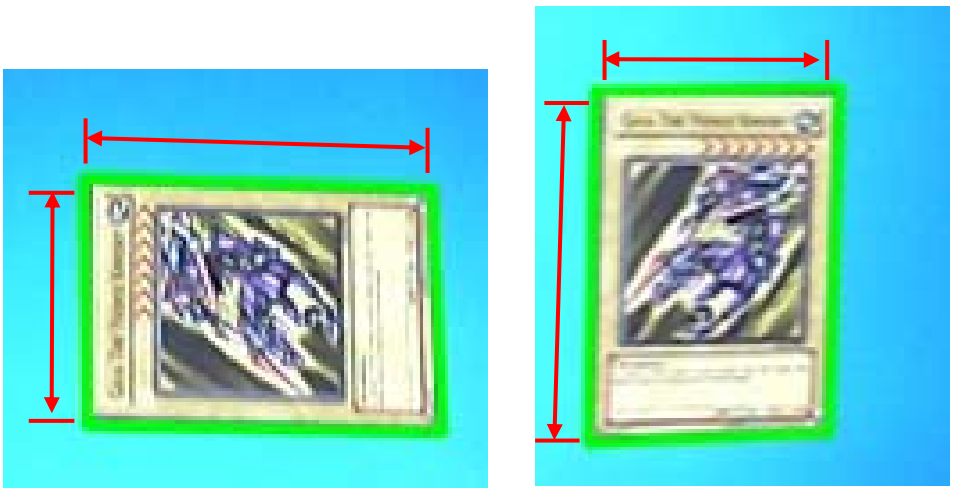


Figure 24: Different orientation of the card in lateral (left) and in vertical (right).

However, more detailed orientation is needed in further card recognition. For cards of “YU-GI-OH”, the color of upper region is darker than lower region (Figure 25). By comparing the average brightness of these two regions, we can determine which side is the lower part of the card. Then, the orientation extend to four directions, north, east, south and west.

Figure 25: Regions to check for card orientation.

5.3.5.2. Undistorted Card Image

To ease further process such as extracting features to determine the card type and the card ID, the system first compute an undistorted version of the card from the distorted one captured from the camera. It is done by a geometric transformation on the distorted card image argumented by the four-point card position on the 2-D image space.

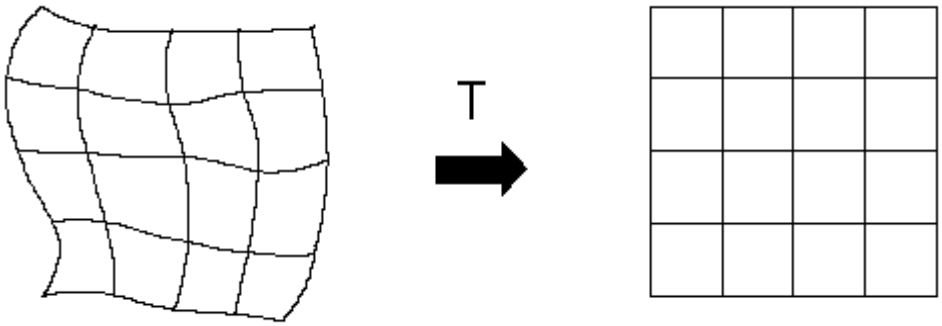


Figure 26: Geometric transformation.

Geometric transforms

Geometric transforms permit the elimination of geometric distortion that occurs when an image is captured. It consists of two basic steps,

Pixel coordinate transformations

The pixel coordinate transformation maps the pixel (x,y) to a new pixel (x',y'). The new pixel coordinate may not be an integer, that is, the pixel lies between some pixels, which have different brightness value.

Brightness interpolation

The brightness of the new pixel is an interpolation of the brightness of several neighboring points. There are some typical interpolation methods such as nearest neighborhood interpolation, linear interpolation, bicubic interpolation, etc.

Pixel coordinate transformation

There are many ways to perform a pixel coordinate transformation. For example, the affine transformation such as rotation, translation, shear and scaling.

For efficiency purpose, we choose a much simpler way to implement the pixel coordinate transformation. It is an approximated version of the transformation. We take the advantage that the overhead camera is placed right above the card. So we assume the distortion is mainly due to rotation, and less is due to the perspective view of the camera.

The transformation maps a point (x,y) on the original distorted image to a point (x',y') on the output undistorted image as shown in (Fig.27).

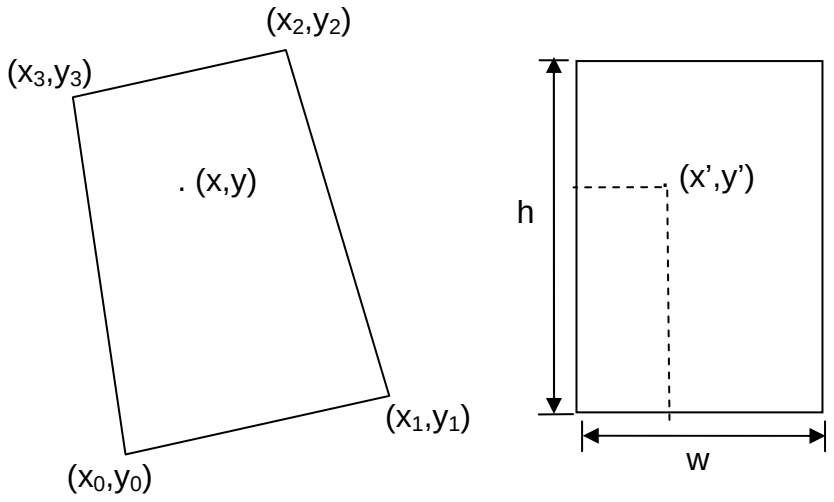


Figure 27: The transformation from a point (x,y) to (x', y').

By performing the inverse of the transformation, we can find the original point in the input image that corresponds to the point in the output image

The mapping function is as follows:

$$\begin{cases} x = (1-\frac{x'}{w})(1-\frac{y'}{h})x_0 + (\frac{x'}{w})(1-\frac{y'}{h})x_1 + (\frac{x'}{w})(\frac{y'}{h})x_2 + (1-\frac{x'}{w})(\frac{y'}{h})x_3 \\ y = (1-\frac{x'}{w})(1-\frac{y'}{h})y_0 + (\frac{x'}{w})(1-\frac{y'}{h})y_1 + (\frac{x'}{w})(\frac{y'}{h})y_2 + (1-\frac{x'}{w})(\frac{y'}{h})y_3 \end{cases}$$

The function compute the new coordinates (x',y') by the weighted sum of the four corner coordinates of the distorted card image, where the weights are the ratios of the distance between point (x,y) and the four boundaries to the height and width of the undistorted image respectively.

Note that the new coordinates (x,y) are continuous real values rather than discrete integer values.

Figure 28: The distorted card image (left) and the resulting undistorted card image (right).

Brightness interpolation

The new point (x,y) found by the previous transformation does not fit with the discrete raster of the original image. Thus, the brightness of the pixel (x',y') on the undistorted image is computed by interpolating some neighboring points of the point (x,y) on the original image. We consider three methods.

Nearest Neighbor Interpolation

The brightness value of the point (x,y) is assigned with that of the nearest neighbor pixel.

The brightness value is computed as follows:

$$f_{nearest}(x, y) = I(round(x), round(y))$$

where $f_{nearest}(x, y)$ is the new brightness value of the point (x,y), and $I(h, k)$ is the brightness value of the pixel (a,b) on the original image.

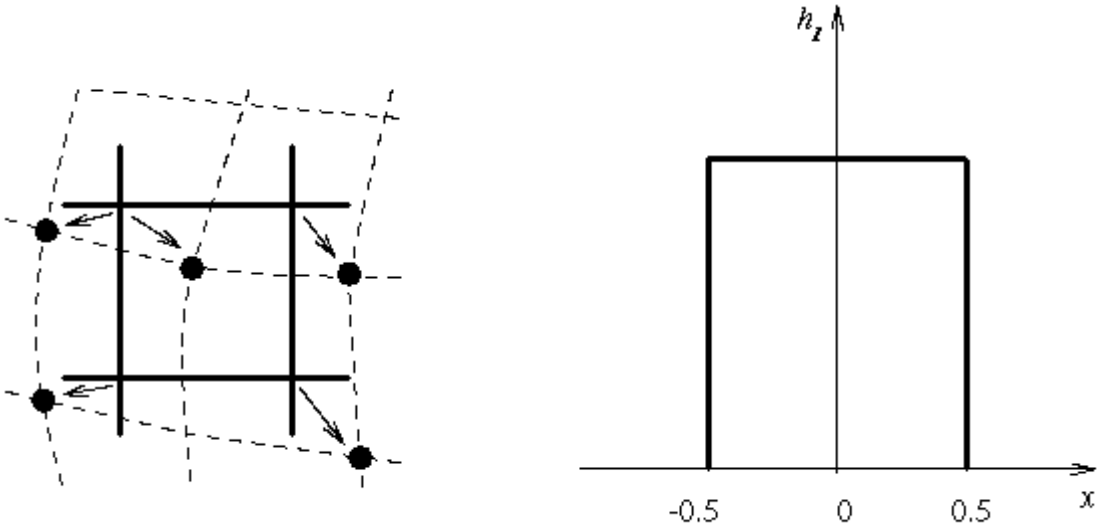


Figure 29: Nearest neighbor interpolation and its brightness function.

(Figure 29) illustrates how to assign the brightness value and the brightness function for this method (variation between brightness and distance of a point).

Bilinear Interpolation

The brightness value of the point (x,y) is assigned with the weighted average of the four nearest neighboring pixels.

The brightness value is computed as follows:

$$\begin{aligned} f_{bilinear}(x,y) = {} & (1-a)(1-b)I(h,k) \\ & + (a)(1-b)I(h+1,k) \\ & + (1-a)(b)I(h,k+1) \\ & + (a)(b)I(h+1,k+1) \end{aligned}$$

$$\begin{aligned} h &= floor(x), \qquad a = x - floor(x) \\ k &= floor(y), \qquad b = y - floor(y) \end{aligned}$$

where $f_{bilinear}(x,y)$ is the new brightness value of the point (x,y), and $I(h,k)$ is the brightness value of the pixel (a,b) on the original image.

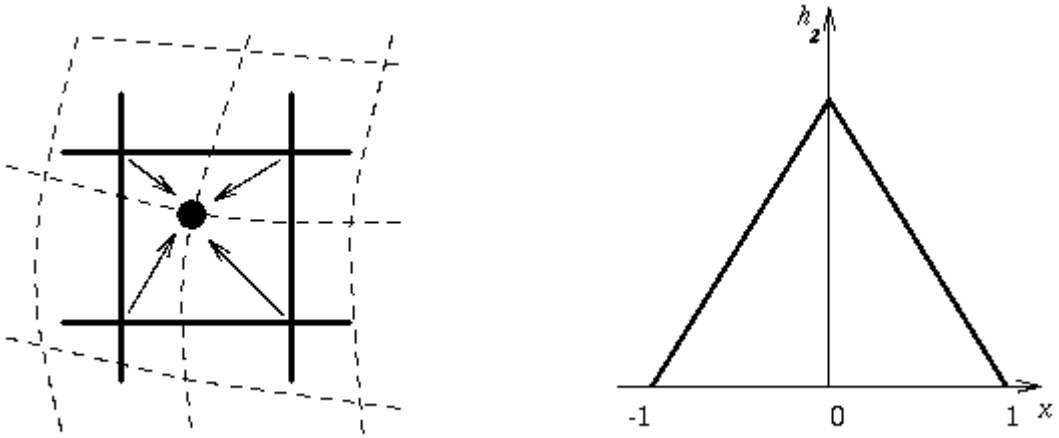


Figure 30: Linear interpolation and its brightness function.

(Figure 30) shows how to assign the brightness value and the brightness function for this method (variation between brightness and distance of a point).

Bicubic interpolation

The brightness value of the point (x,y) is assigned with the weighted average of the sixteen nearest neighboring pixels using an improved model of brightness function that approximate brightness locally by a bicubic polynomial surface.

The brightness value is computed as follows:

$$h_{bicubic} = \begin{cases} 1-2|x|^2+|x|^3 & for\ 0<|x|<1 \\ 4-8|x|+5|x|^2-|x|^3 & for\ 1<|x|<2 \\ 0 & otherwise \end{cases}$$

where $h_{bicubic}$ is the brightness function, and $|x|$ is the distance of the point of interest to the point (x,y).

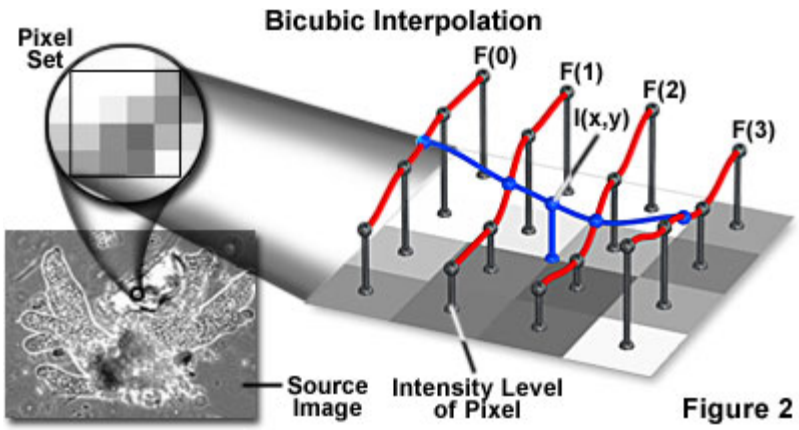


Figure 31: Illustration of bicubic interpolation.

5.3.5.3. Identify card ID

The final step is to identify the card by its image. The Card Image Database provide surface to query the unique card ID by the card image. By passing the undistorted version of the card image captured from the camera to the Database, the unique card ID of the card the player input is retrieved. The detection of player card input is then finished.

5.3.6. Input Button

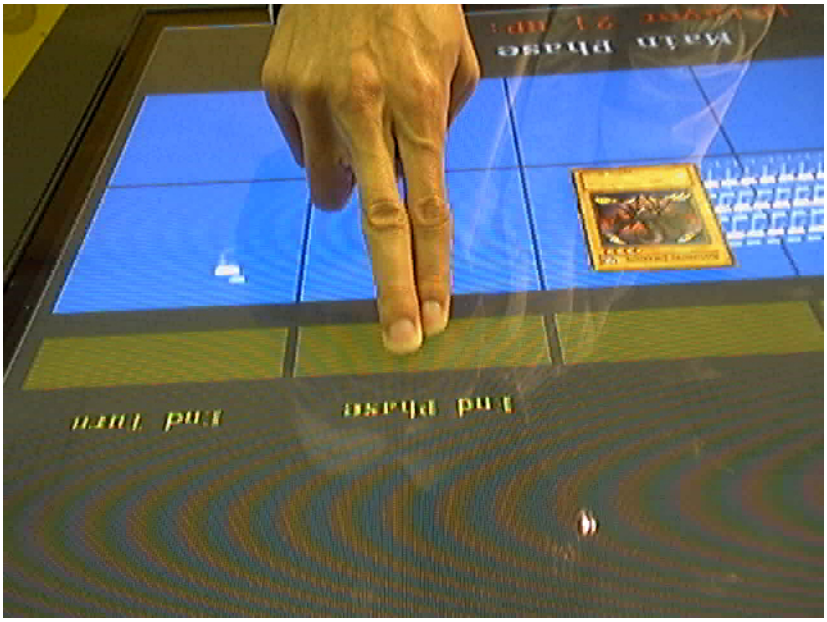
Other than the card inputs, players can also use buttons on the screen as input. This allows player to make more advance actions, like pressing button to trigger attack, and set targets involved in the attack action.

The plasma TV table is not a touch screen devices, so once again, the overhead camera is the only device to detect the input button action. The system receives player button input by processing a given frame captured from the camera.

To locate the button pressed, we use similar technique for card detection.

5.3.6.1. Predefined Search Windows

At the beginning, the location of each button is calibrated in the calibration step, from which the search window of each button is pre-calculated. Since the positions of the camera and the table are fixed, the calibrated location of the buttons and their corresponding search windows will not during the game play.



5.3.6.2. Locate Search Windows

During game play, the system keeps track on the predefined search windows of the button for changes. As the camera and the table are fixed, these changes must be the result of some player input. They can be detected by comparing the current frame with the previous frame as follows:

$$I_{diff}(x, y) = \left| I_{current}(x, y) - I_{previous}(x, y) \right|$$

where $I_{diff}(x, y)$, $I_{current}(x, y)$ and $I_{previous}(x, y)$ are the pixel value of pixel (x,y) on the resulting different image, current frame, and previous frame respectively.

Search windows with only little changes are rejected because these changes are probably caused by noise of the input. Two thresholds are set for this purpose.

A pixel is said to be changed if

$I_{diff}(x, y) > \Phi_1$ where Φ_1 is the threshold of changed pixel.

A search window is said to be changed if

$\sum I_{diff}(x, y) > \Phi_2$ where Φ_2 is the threshold of changed search window.

5.3.6.3. Locate Button Pressed

The changed search window found in the previous step does not necessarily mean that the player is pressing the corresponding button. May be the change is due to the withdrawal of the player's finger from the button.

To ensure the change is made when the player press the button only once, we apply a little heuristic here.

We assume that the player press the button very quickly and withdraw his fingers immediately after pressing the button. 10 camera frames should be a suitable upper bound time limit for this action to take place.

Therefore, the button press event is fired only when the system detects changes in its corresponding search window at the first time, and never fired any event within 10 frame for this button.

5.4. Generic Card Game Database Module

The Generic Card Game Database Module contains all game-specific information about the card game, and provides method to retrieve these information. Concerning the performance of Game Enhancement Module which is visible to players as before, the efficiency and accuracy of this module is again very important.

5.4.1. Card Image Database

The Card Image Database is responsible for identifying a card uniquely. By querying the database by a card image, the unique card ID is retrieved. The card ID is then used to obtain card details from the Card Database at later time.

For our system, the query process can be subdivided into three main stages. The first stage is to determine the card type. The second stage is to retrieve similar cards among all the cards in the image database. This stage requires the Image Retrieval technique. The third stage is to select one card out of those cards found in the previous stage, and confirm the card match the querying image.

5.4.1.1. Identify card type

The first stage classifies the query card image into different card types. To identify a card uniquely, searching for a match image in the image database is necessary. Such classification reduces the search space and search time, and increases the accuracy as well.

For “YU-GI-OH”, different card types have different background colors. The system computes the average color of the background (marked as the red box in Figure 32) and compares it to the value record from calibration procedure. The input image is said to match a card type if the different between its average background color and the calibrated background color is smaller than some threshold. Then the best match card type is chosen among all matched card types.

Figure 32: A monster card (left), a spell card (center) and a trap card (right).

5.4.1.2. Image retrieval

The second stage can be regarded as an image retrieval stage, from which relevant images of the same card type are retrieved using an image database query.

An image retrieval system is quite different from a traditional database system. For a traditional database system known to most people, a large amount of alphanumeric data is stored in a local repository and accessed by content through appropriate query language. While for an image retrieval system, the “data” refer to low/intermediate-level features (content-dependent metadata), such as color, texture, shape, and so on, and their combinations. Sometimes data may refer to content semantics (content-descriptive

metadata) which concerned with relationships of image entities with real-world entities or temporal events, emotions and meaning associated with visual signs and scenes. Thus other kinds of metric are used for searching in an image retrieval system, rather than textual search.

Our image database belongs to color-based retrieval system for 2D still images. The reason why we do not use other image feature is due to the difficulty to extract high level features (e.g. edges, objects) from a very low resolution query image (about 40x40). Since many features are lost when a high resolution image is downgraded to a low resolution image, we cannot compare the similarity between the query image and the image in the database. Therefore, using color content suits our system requirement the most.

We consider three methods based on the global image chromatic content here.

Color Histogram

After knowing the card type, the search continues using the color histogram of the card. The histogram intersection is defined as follows:

$$D_H(I_Q, I_D) = \frac{\sum_{j=1}^{n} \min(H(I_Q, j), H(I_D, j))}{\sum_{j=1}^{n} H(H_D, j)}$$

where $H(I_Q, j)$ is the j[th] bin of the query image histogram,
and $H(I_D, j)$ is the j[th] bin of the database image histogram.

Figure 33 shows the Histogram intersection between two images.

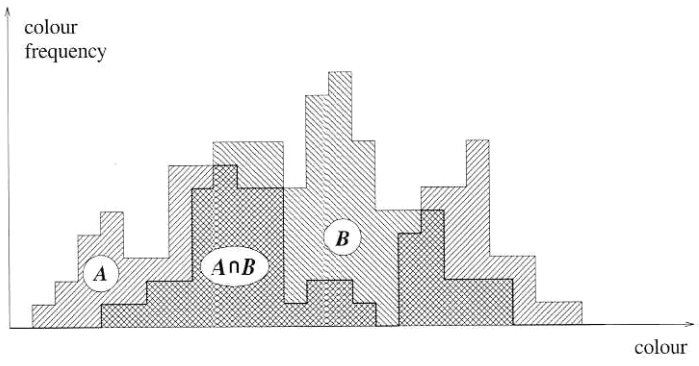


Figure 33: Histogram intersection between two images.

The histogram intersection can be used for color matching. If the histogram intersection is bigger than a threshold, we said that they are similar in the sense of color content of the image.

Incremental intersection

A variant of histogram intersection, called incremental intersection, improves retrieval effectiveness. This method uses only one bin of the histograms of the query and database images at a time to compute a partial histogram intersection. The search starts with the largest bin of the query image, and continues with the next largest bin, and so on.

Average color distance

Another method to improve efficiency is to make use of average color distance. The average color distance is calculated as follows:

$$D_{H_{avg}}(I_{Q_{avg}}, I_{D_{avg}}) = (I_{Q_{avg}} - I_{D_{avg}})^t (I_{Q_{avg}} - I_{D_{avg}})$$

where $I_{Q_{avg}}$ and $I_{D_{avg}}$ are 3 x 1 average color vectors of color histograms $H(I_Q, j)$ and $H(I_D, j)$.

This distance operator are used to perform a sort of prefiltering before applying complete histogram matching.

5.4.1.3. Block Matching Algorithm

The final step is to select a card from several candidate cards selected from the previous stage that exactly matches the query image. As stated before, many features are lost due to the extremely low resolution of the card image. Therefore, performing a pixel by pixel matching would be the most accurate way. Block matching algorithm is an example.

Before performing the image matching, we will split the image into four channels. They are hue channel, red channel, green channel and the blue channel (Figure 34).

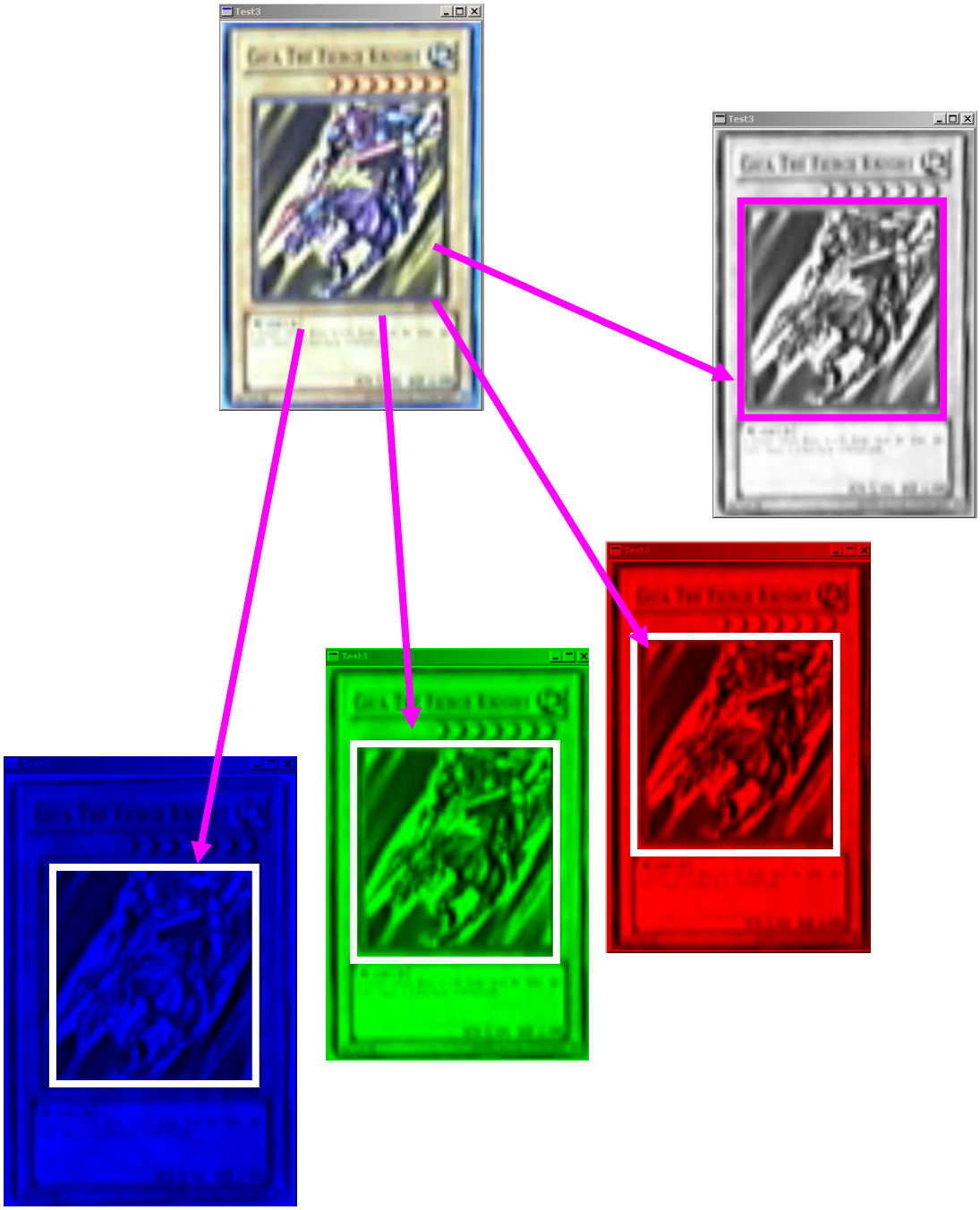


Figure 34: A candidate image is split to four channels. From right to left is Hue channel, Red channel, Green channel and Blue channel.

After we split the image into four channels, we compare the candidate image to image database. Only the inner part of image will be compared since the inner part is distinct among each card.

In each inner image of each channel, the overall pixel difference of the query image and database image are calculated as follows:

$$D_P(I_Q, I_D) = \sum_{i=1}^{w} \sum_{j=1}^{h} (I_Q(i, j) - I_D(i, j))^2$$

where $I_Q(i, j)$ and $I_D(i, j)$ are the pixels of query image I_Q and database image I_D respectively.

If the value of $D_P(I_Q, I_D)$ for a candidate card is smaller than some threshold, we accept this channel of candidate card. If all four channels are accepted, then we accept this candidate card.

In case of several candidate cards are accepted, the one with the smallest $D_P(I_Q, I_D)$ is selected for best matched.

5.4.1.4. Improved Block Matching Algorithm

To begin with, let look at how this error is caused.

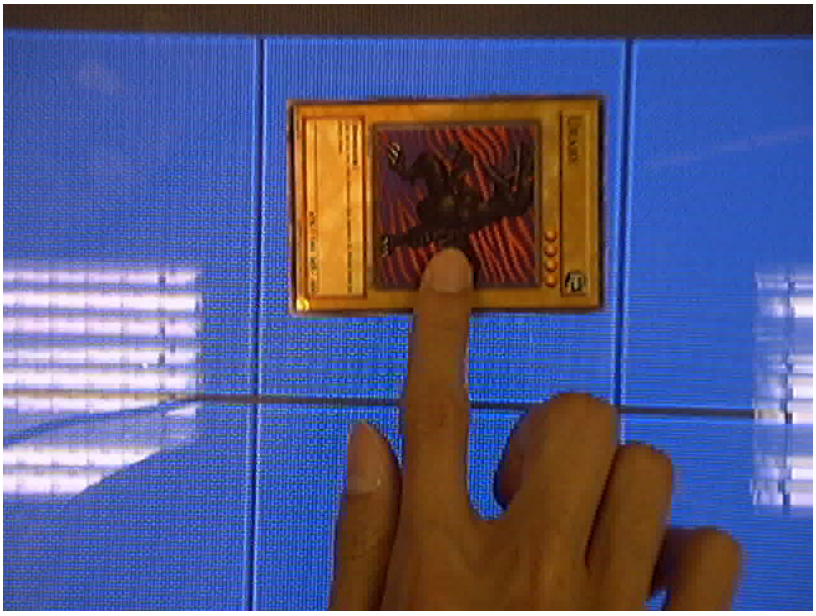


Figure 35: a screen captured for real game playing

The above figure is captured during the game play, since the user usually put his finger on the card, the recognition process is always faster than the player withdrawal of his finger. Therefore, the recognition process detects the card and starts the blocking matching algorithm.

Since the whole block has some error pixels, the pixel difference will slightly different from the accurate value. However, this reason doesn't account for why the difference is greater than the threshold, as a result, the system will recognize it as another card.

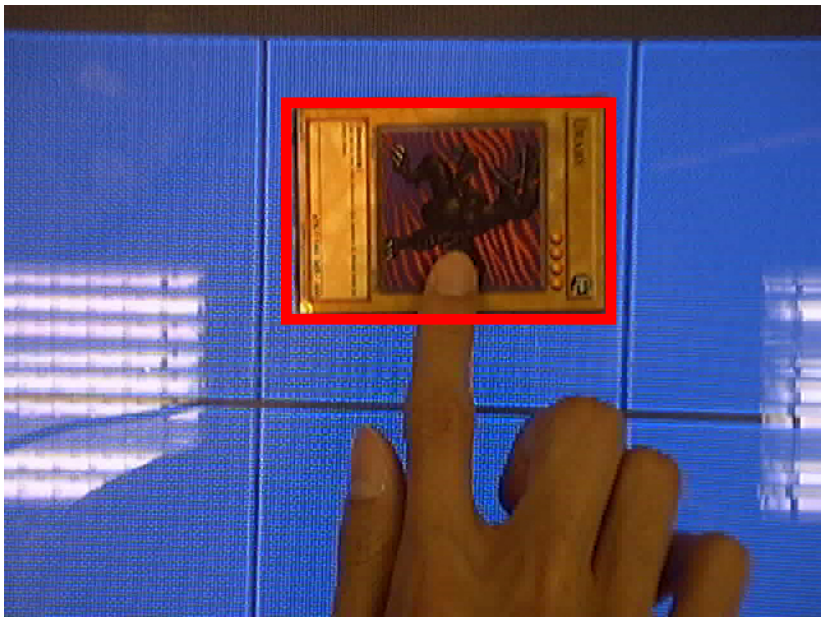


Figure 36: a screen captured for real game playing

= ?

Figure 37: a card extracted for the captured image

To solve this problem, we propose the improved Block Matching Algorithm. This algorithm is the same as the block matching algorithm; however, we will do something before calculating the pixel difference.

First, we divide the card into 9 squares. Then we apply the blocking matching algorithm to each square. It means that we will get 4 pixel differences (the R, G, B and H channel) from each square. The formula to calculate the pixel difference of each square as follows:

$$D_P(I_Q, I_D, n) = \sum_{i=(w/3)(n/3)}^{(w/3)((n/3)+1)} \sum_{j=(h/3)(n\%3)}^{(h/3)((h\%3)+1)} (I_Q(i,j) - I_D(i,j))^2$$

Where n ranged from 0 to 8.

Figure 38: the improved block matching algorithm

We accept a card if the total 36 pixel differences are below the threshold. This can prevent the error in real time game playing. It does not have great improvement of accuracy to Block Matching Algorithm, however, it does slightly improve the efficiency.

5.4.2. Card Database

The Card Database is responsible for storing all the card details. The card details include the unique card ID, card type, and other game-specific information such as attack, defend, effect, and so on, depends on what card game is playing.

5.4.2.1. Card Information

The card information is stored in an ASCII file. The card database first reads the file and stores all the card information in memory. It provides methods for other modules to query for when necessary.

The card information are game-specific, but most generally they contain a unique card ID, name, some descriptions, and card type. In many card games, the card information is also type-specific, to be more precise, there are some extended information depends on the card type. For example, monster cards usually contain monster attack and defend points, and also their races and attributes, while spell cards usually contain the spell types and effects of the card.

For our implement, the fields of a card are shown in the following table.

Field	**Explaination**	**Remarks**
Card ID	Unique ID of the card	
Card Type	Type of the card	e.g. “Normal Monster”, “Spell”, “Trap”
Card Name	Name of the card	
Description	Description of the card, explain some information and effect of the card	They are only plain-text explain without any effects to the game
Detail Type	More specific type of the card, with respect to card type	e.g. “Dragon” in “Normal Monster”, “Quick-Play” in “Spell”
Attribute	Attribute of the card	Only available for Monster-type cards
Level	Level of the card	Only available for Monster-type cards
Attack	Attack point of the card	Only available for Monster-type cards
Defend	Defend point of the card	Only available for Monster-type cards
Rule Index	Index of the rule belonging to the card	Only available for cards which have special effect

5.4.2.2. CardInfo Editor

A card games usually contains hundreds or thousands of different cards. An editor is provided for the input of the huge number of card information. The following is a screenshot of the editor.

Figure 39: Database Editor

5.4.3. Rule Database

The Rule Database is responsible for storing all the game rules and card rules. The rules include all the game logic and card effects of the game.

5.4.3.1. Rule Information

The rule information is stored in two separate ASCII files, one for game rules and one for card rules. The rule database first reads the files and stores all the game rules and card rule in memory. It provides methods for the Game Core to query for when necessary.

The rule information in the database consist of two parts, a rule follows the premise-conclusion form in first-order logic, and a list of actions.

The rule in premise-conclusion form contains a set of premises, which we called predicates, and a conclusion. The Rule Manager of the Game Core Module inference these rules to maintain the game flow. We will discuss it in detail in the Game Core Module.
The action list contains a set of actions to be performed when the rule is fired. The actions include loading or

removing a rule, updating game status, etc. We will also discuss it in detail in the Game Core Module later.

5.4.3.2. Rule Editor

A card games usually contains hundreds or thousands of different cards. An editor is provided for the input of the huge number of card rules. The following is a screenshot of the editor.

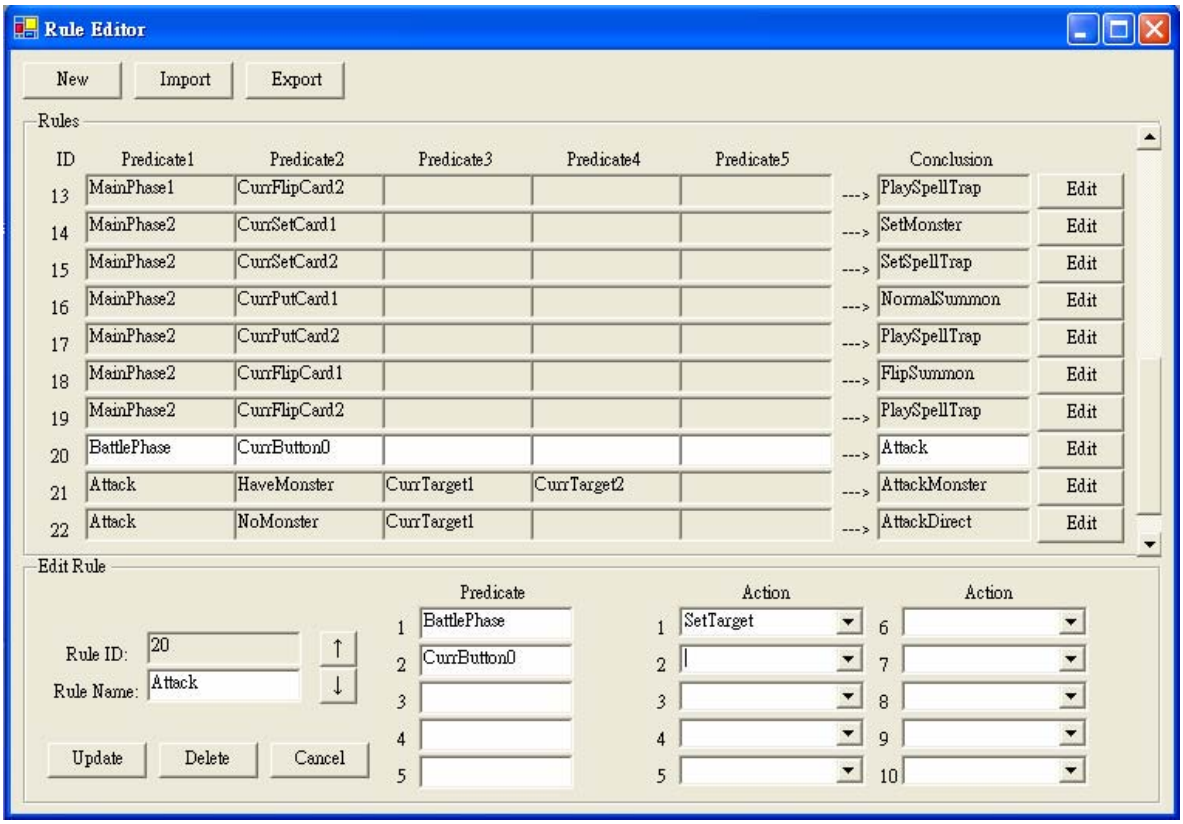


Figure 40: Rule Editor

5.5. Game Core Module

The Game Core Module is the most important part in the ART System. It co-operates the Perception Module for input, Game Enhancement Module for output, and the Database Module for game data. It also maintains game states and game flow as well.

The Module consists of two parts: the Game Manager which manipulates the game states, the Rule Manager which stores and inferences game rules, and the Game Core, which control the interaction between all managers.

5.5.1. Rule-based Game Engine

The Game Core is designed as a rule-based game engine. A rule-based game engine receives events and raise appropriate events accordingly by the pre-defined game rules. The whole system is driven by the rules and events.

A rule-based game engine is considered to be the most suitable for the implementation of a generic card game engine. The following two sections explain some of the reasons.

5.5.1.1. Difficulties

There are some characteristics for card games which considered to be difficult for implementation of a generic card game engine:

Addition of new cards

In general, card games contain hundreds or thousands of different cards. New cards are invented and added continuously when expansion set is released.

Therefore, the system must consider future extension of new cards. It cannot simply assign each card effect a procedure.

Unpredictable effects

Usually, card games are very flexible. There are no formal constraints on the effect of the cards. In other words, some cards may have some effects that are unpredictable in the design phase. These effects may change the game states traumatically, and even influence the normal game flow!

Implementing the card effects by brute-force method becomes impossible.

Generic concern

Although general card games share some common properties, their game rules are quite different and some of them are hard to develop from the old one.

A generic game engine is required to ease the implementation of other card games.

5.5.1.2. Advantages of Rule-based

With the above difficulties, it is hard to design a game engine that is generic and flexible enough to satisfy these requirements.

However, a rule-based game engine has the following advantages:

Extensible

A rule-based system is extensible. New cards can be extended by adding new rules and new card info to the database alone.

Flexible

Contrary to brute-force programming, the game logic is maintained in term of rules, which is more flexible. So the unpredictable card effects can still work if their rules are set properly.

Generic

Since the game logic and game flow are solely determined by game rules, new card game can be implemented by modeling the game into a new set of game rules.

5.5.2. Game Manager

While the game logic and game flow are maintained by the Rule Manager, the game states are stored in the Game Manager.

The Game Manager is responsible for storing and manipulating game states. It also provides method for Rule Manager to execute appropriate actions when a rule is fired. These game states include each player's life point, monsters, spells, traps, both at their mat and their graveyard. The actions will be discussed in the Rule Manger section.

5.5.3. Rule Manager

The Rule Manager is responsible for storing all the game rules and inferring the rules to trigger appropriate events.

It contains a pool that load and store the rule dynamically during game play. These rules will fire if all its condition is satisfy, and then the corresponding actions are carried out.

5.5.3.1. Rule Structure

A rule consists of a rule body which is in premise-conclusion form, and an action list as shown in figure.41.

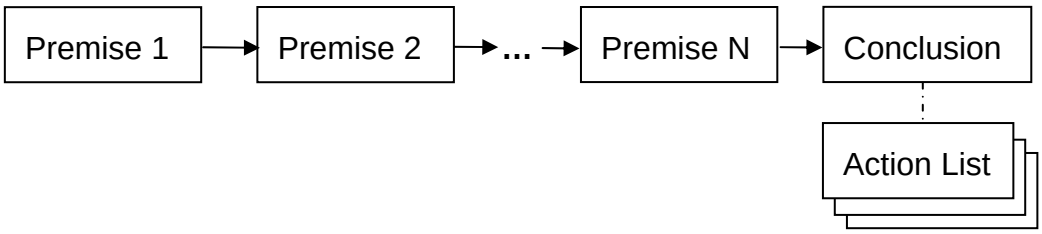


Figure 41: a rule structure

5.5.3.2. Rule Inference Mechanism

In premise-conclusion form following first-order logic, each rule is actually an if-then relation. The general form is as follows:

$$\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n \rightarrow \psi$$

where ϕ_i is the i[th] premise and ψ is the conclusion

The formula means that if premise 1 to n are all true, then the conclusion is also true and the rule is said to be fired. For Example, the rule shown in Fig.42 represents if both of the premises “BattlePhase” and “Button1” are true, then the conclusion “Attack” is true, and this rules is fired.

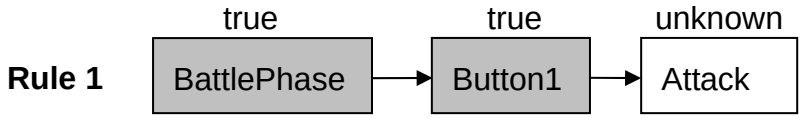


Figure 42: a sample rule

The inference mechanism used here is forward-checking. When all its premises are true, the rule is fired and its conclusion is used as a premise for other rules. For example, if there is a rule as shown in Fig.43, after “Rule 1” fired, its conclusion “Attack” is use to infer other rule and so “Rule 2” is also fired.

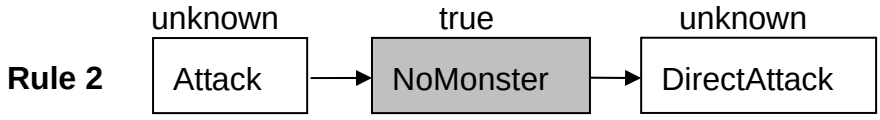


Figure 43: another sample rule

The inference procedure continues until there is no new rule being fired.

5.5.3.3. Action List

Once a rule is fired, its action is carried out accordingly. There are two types of action: actions that update current game states, and actions that manipulate rules in the pool.

Update Game States

For the actions that update game states, these actions are executed in the Game Manager. Some of them only update the score, and some of them required update to the screen, like summoning of a monster, which is passed to the Game Enhancement Module.

Update Rules

For the actions that manipulate rules in the pool, there are six fundamental actions:

Action	Effect
Set Premise	Set a premise to TRUE
Unset Premise	Set a premise to FALSE
Reset Premise	Reset a premise to UNKNOWN
Reset Rule	Reset a rule to UNKNOWN state
Load Rule	Load a particular to the pool
Remove Rule	Remove a particular from the pool

The first four actions, “Set Premise”, “Unset Premise”, “Reset Premise” and “Reset Rule” are four essential actions to keep the game logic.

The “Load Rule” action is executed when a new card with new effect is use during the game play. The rule is load since now the game rule must include this new card rule, so that the card effect can be take place to influence the game flow.

The “Remove Rule” action is executed when a card is remove from the game play. When the card is removed, its effect no longer takes place and the rule representing the card effect must be removed.

5.5.4. Game Core

The Game Core is responsible for controlling the interaction between Rule Manager, Game Manager, and the Input and Output Manager.
The Game Core is event driven. When the Perception Module detects inputs, it generates and sends the event to Game Core.

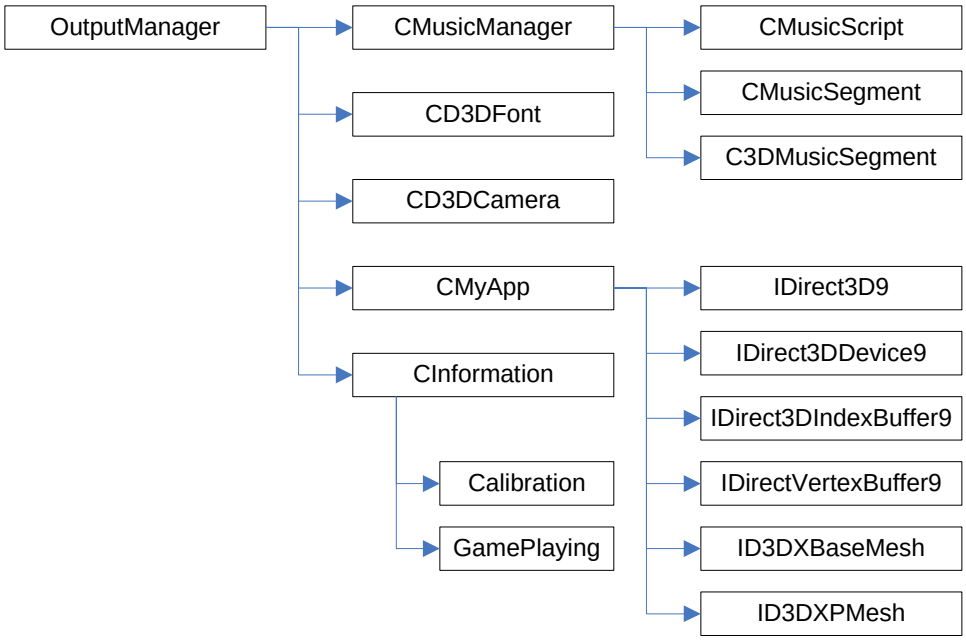
The Game Core then asks Game Manager to update the game states and asks Rule Manager to set appropriate premise. The Rule Manager then performs forward-checking to check whether there is rule fired. If some rules are fired, their corresponding actions are carried out and forward-checking is continuous. Some actions may update the game states and required update to the display.

These update output command is then passed from the Game Core to the Game Enhancement Module to update the display to players.

5.6. Game Enhancement Module

The Game Enhancement module read the game information from ART Card Game Core, and then generates corresponding display and sound effect.

This module mainly depends on Microsoft DirectX API. Since it provides functions to access the display buffer and sound buffer. It greatly reduces the coding time with hardware such as display card and sound card.



The above shows the architecture of the output module. CMusicManager is the main part to produce sound. CD3DFont will produce 3D font image, which is used by CMyApp. CD3DCamera controls the camera in the 3D virtual sense. CMyApp is the main part of display, it controls all display device, as well as the display buffer. Cinformation is the generic part of this module. All render information is stored under Cinformation. If we want to add other phases, or even change the game, we only need to add class under CInformation, and use the API provided by CInformation to create the information state.

This module can be divided into two parts, the display part, which output the virtual scene, and the sound production part.

5.6.1. Display

There are two main phases for display part. The first one is the calibration process. It will show the calibration map for camera to calibrate the required position. Another phase is the game playing phase. It displays the 3D game map and buttons, as well as the animation.

5.6.1.1. Calibration

During the calibration process, the game enhancement modules will display a 2D image. This image consists of two colors, black and white. The 20 white squares represent the card zones, and the 10 white rectangles represents the buttons.

This can be shown as the following figure.

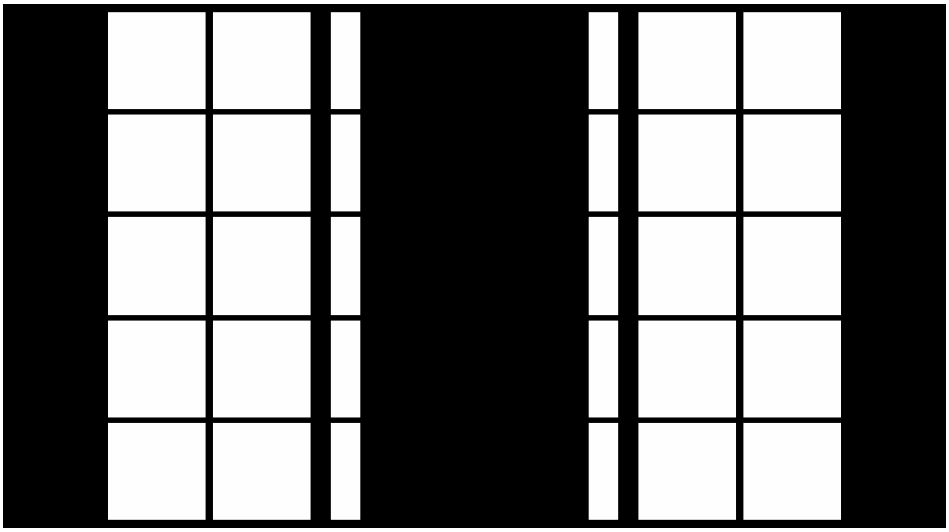


Figure 44: a calibration mat

5.6.1.2. Game Playing

During the game playing stage, several information need to be output to the user.

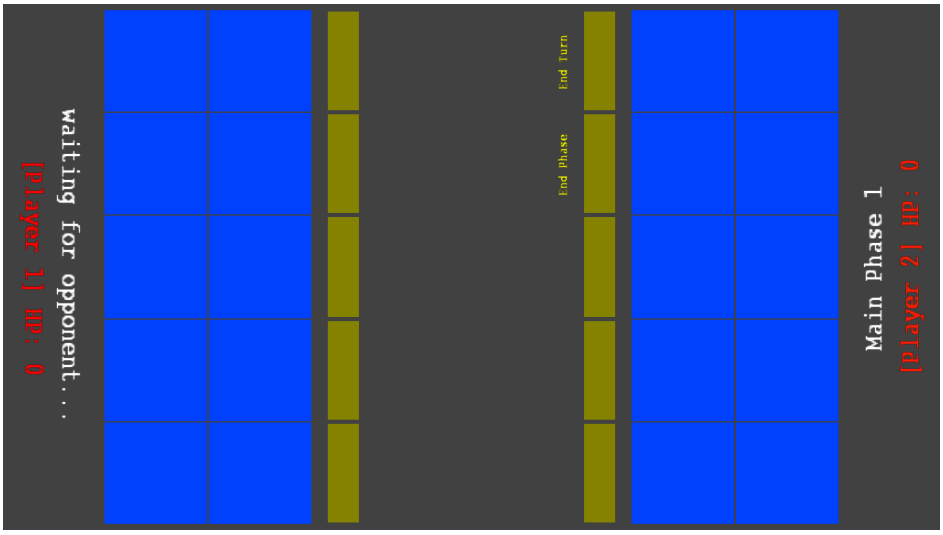


Figure 45: a game play mat

Life Point Label

This part shows the remains life point of the player. When this modules receive a LOSE action from the game core, the life point label will be automatically reduced to the value sent by game core.

Figure 46: Life Point Label

Instruction Label

This part shows the instruction to the user.

Figure 47: Instruction Label

When the game core sends the action that related to the phases, for example, START, STANDBYPHASE, MAINPHASE, BATTLEPHASE, MAINPHASE1, ENDPHASE, the corresponding player's instruction label will change to the appropriate string.

When the game core sends the action related to choosing target, for example, SETTARGET and ENDTARGET, The instruction label also will be changed. It will instruct player to choose target for SETTARGET action, while showing the current phase for ENDTARGET action.

When the game core sends the action related to battle, for example, WIN, LOSE, ATTACKMONSTER, REMOVEMONSTER, the instruction label shows the change of life point, and it also will tell the players to remove the monster if it had been killed.

Card Zones

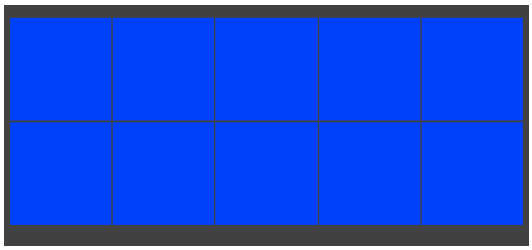


Figure 48: Card Zones

The card zones represent the area that the user should be put the card in. Each user have 10 different card zones. The top five of a player is monster card zones, while the other are spell card zones. When the game core send the action about monster, for example, ATTACKMONSTER, REMOVEMONSTER, SUMMONMONSTER, the corresponding card zone will be highlighted by a yellow square. It represents the card zone have just receive some event.

Animation

It responds to the action about monster and battle, for example, ATTACKMONSTER, REMOVEMONSTER. When these action have been received, correspond animation will be shown.

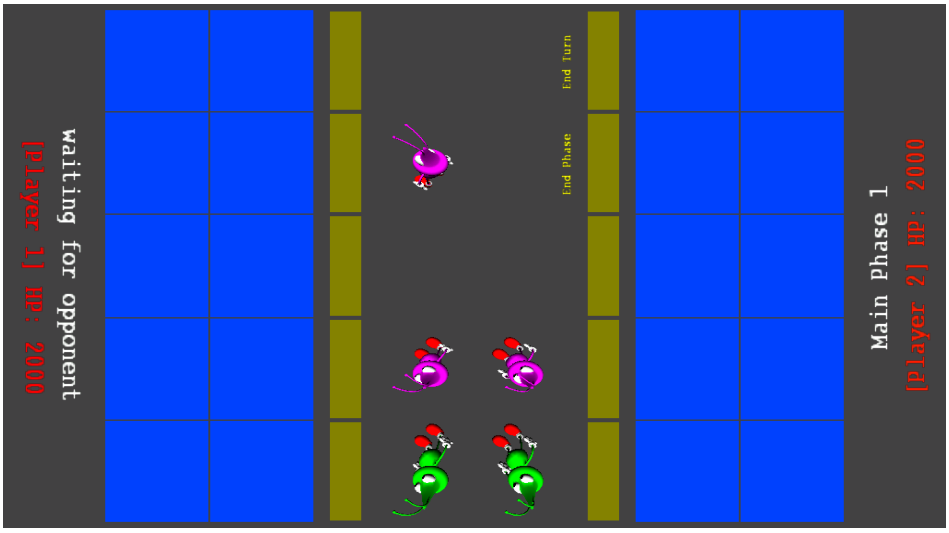


Figure 49: animations in the center of the game mat

In our implementation, we have only make a module of monster, and use different color to represent the different type. The API we provided can enhance the further development.

Button and Button Label

Figure 50: Button and Button Label

Buttons are drawn as rectangles. These rectangles are yellow in color. When the display module receives actions about button, then the display module will highlight the button which has been clicked. Since the Perception module will keep checking whether the button have changes or not, therefore, we change the color of button label instead of the area of the buttons. When this module receives an action related to choosing target, then the button label will also be changed.

5.6.2. Sound Production

This part only functions on the game playing phase. In our project, we have implemented the API to produce sound, it enhances the further development. In our implementation, we have only chosen some events that would be produce sound.

To produce sound, we need to load the wave file into buffer in the initialize stage. After that, we just need to call play to play the sound segment from the Sound Manager.

6. Difficulties

During the design and implementation of the Augmented Reality Table, we have encountered several difficulties and problems. Some are solved while some are not.

6.1. Solved problem

Problem 1

The edge detection algorithm is very slow. We need to apply some edge detection algorithm to locate a card, but the display delays quite a lot. This inefficiency will affect the performance of game enhancement module visible to user.

Solution

The use of search window introduced in the implementation session greatly reduces the frequency and space of search. This allows us to use more accurate but less efficient algorithm.

Problem 2

The Card Locator locates card by searching rectangles in the search windows. Sometimes it locates rectangles inside the card.

Solution

Use of card area in the search ensures the resulting rectangle must be with certain size, so any smaller rectangle inside the card is ignored.

Problem 3

The resolution of the image of a card perceiving from the camera is very low, typically about 40 x 40 pixels for a 640 x 480 camera. It is difficult to recognize a card image in such low resolution.

Solution

Use high quality digital camera to achieve better resolution. Color-based algorithm is also used which depends less on resolution of the image.

Problem 4

The system now used only one camera for input of both players. In this case, the camera needed to locate high and the resolution of the card appear in the video input is then dropped.

Solution

We have considered using two separated cameras over each player input area, but this will introduce efficiency overhead and complexity to the system. Now we found that a single high resolution camera is

enough to have both high efficiency and accuracy.

Problem 5

The Block Matching Algorithm can recognize cards quite accurately in experiment. However, in practical investigation, we found that some errors are introduced because user may cover part of the card accidentally in the recognition step.

Solution

We use an Improved Block Matching Algorithm, which subdivide the card into nine blocks and apply the original Block Matching Algorithm to each sub-block.

Problem 6

The lighting setting is critical for the ART System because it use only a camera as input. We cannot use the original room lighting in the laboratory as their reflection from the flat plasma table results in bright region in the perceived image.

Solution

We have tried to use a spot light for this purpose, but the lighting is uneven such that the region far from the light is much darker than that near to the light. Finally, we set the spot light to a higher and further position, and cover the light with translucent paper to make the lighting more even.

Problem 7

Usually there is hundreds or thousands of cards in a card game. Even for our prototype system, we use about a hundred cards. Input and edit the card information and rules to the database are too laborious.

Solution

We have write a card info editor to input and edit card information, and a rule editor to input and edit game rules and card rules..

Problem 8

General card games are extensible and flexible. We found it difficult to design a card game engine that can still retain these characteristics. Making the system generic to number of card games is also another problem.

Solution

We have developed a generic rule-base card game engine. With the use of rule-driven game engine, the system is flexible, extensible, and generic to most of the card game.

6.2. Unsolved problem

Problem 1

The lighting condition and background color can affect the accuracy of the system seriously. We have complete the game mat calibration, but yet color calibration is need to compensate varies lighting condition in different environment.

7. Project Progress

Month	Task Completed
June 2004	Get familiar with the programming environment of Microsoft Visual C++
	Study the background information of augmented reality.
	Study and Test different software API for video processing
	Study and Test different rendering library
July 2004	Study COM model for Window programming
	Study the documentation and sample programs of DirectX SDK
	Study the media processing
August 2004	Design the software system architecture of ART
	Study the image processing
September 2004	Study different edge detection algorithm for Card Locator
	Study and test different pattern recognition algorithm for Card Recognizer
	Study the calibration of camera
	Integrate Card Locator and Card Recognizer into Card Detector
October 2004	Integrate Calibration part and Card Detector part into perception module
	Implement a simplified database and game core

November 2004	Implement a simplified Game Enhancement module
	Integrate four modules into a simplified ART Card Game application
	Prepare first term final year project presentation and demonstration
	Write the first term final year project report
December 2004	Study and test color calibration
	Improve the efficiency of Card Recognizer
January 2004	Investigate a reliable algorithm to detect player input
	Integrate color calibration into perception module
	Construct 3D model for animation
February 2004	Implement the Input Detector
	Implement the ART Card Game Core
March 2004	Complete the database
	Implement the game enhancement module
April 2004	Integrate four modules and finalize the source code of our system
	Prepare second term final year project presentation and demonstration
	Write the second term final year project report

Summer | First Term | Second Term

8. Experimental Result

We have conducted several experiments to analyst the quality of our system. The first experiment is used to determine the accuracy of our algorithm. The second experiment is used to analyst the Block Matching Algorithm, and the last one is used to analyst how the color of background affects the captured image.

8.1. Image Recognition (1)

We have used 85 cards to test the accuracy of our improved block matching algorithm. To perform the test, we recognize each card ten times to check the result is correct or not.

Besides, we have also conducted a control experiment. We use 85 cards to test the accuracy of the block matching algorithm. We use the same method but this time we only test each card for 5 times.

Here, we get the result as follows:

The result of Improved Block Matching Algorithm

Number of hits	Number of cards	Percentage
Always (No miss)	73	86%
Sometimes (miss 1-9 times)	9	11%
Never (miss 10 times)	3	4%

The result of Block Matching Algorithm

Number of hits	Number of cards	Percentage
Always (No miss)	66	78%
Sometimes (miss 1-5 times)	15	18%
Never (miss 5 times)	3	4%

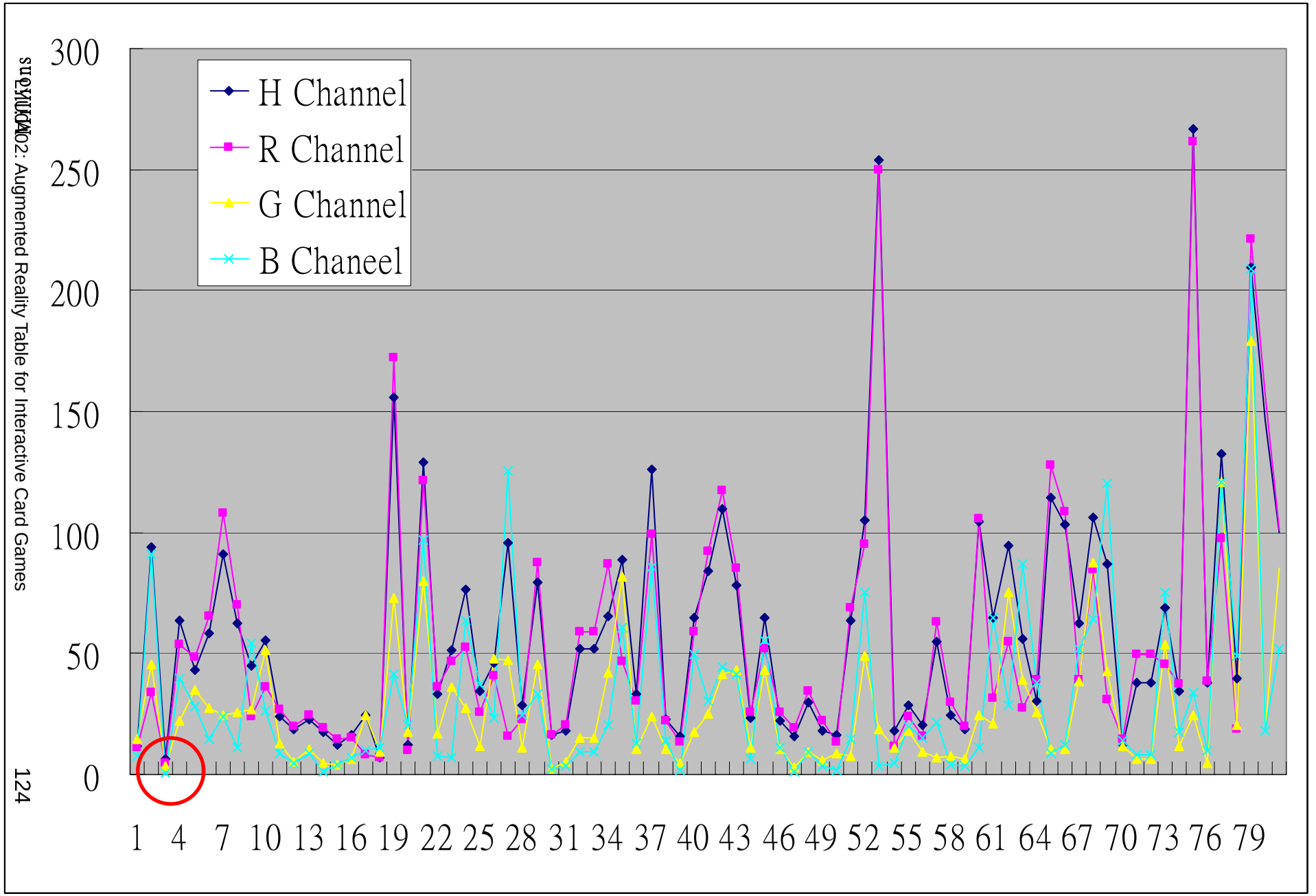
From the result, we can see that the improved block matching algorithm has slightly increased the accuracy. However, the increase in the real game is much greater. It is because when we are conducting the experiment, everything will do in normal way. In real game, users always use unpredictable methods to put the cards. As a result, unexpected error is caused.

8.2. Image Recognition (2)

We have also conducted another experiment that is related to the accuracy of the algorithm. To analyze the strength and weakness of the algorithm, we have record the pixel difference of each channels between a target cards and others cards that stored in database.

Since there are total 36 channels (9 blocks x 4 channels) for each card. Therefore, we have simply sum up the difference to reduce 36 channels to 4 channels. The following graph will show the pixel difference of a card against other cards.

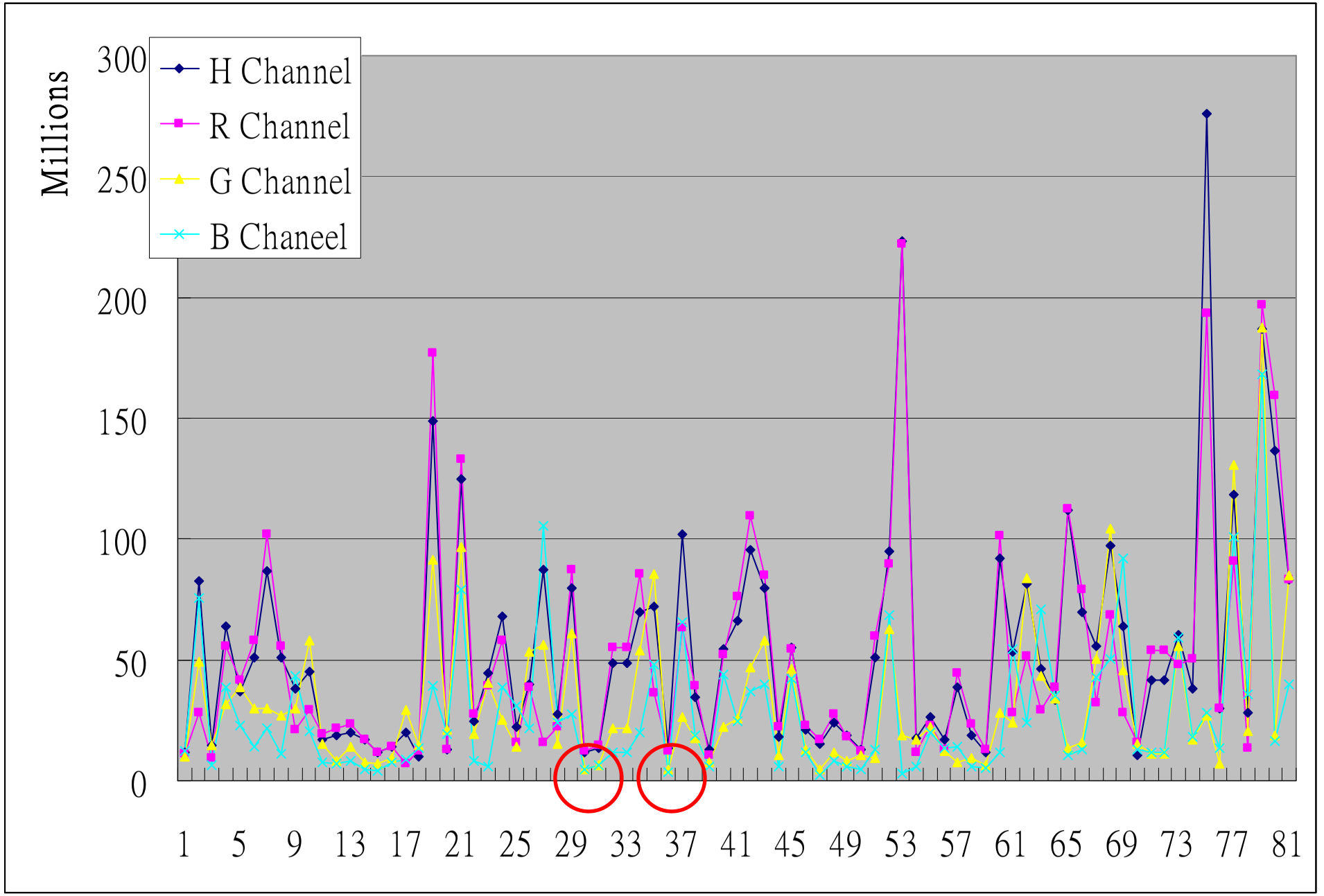
Among the 85 cards, we have chosen 2 cases for our discussion.


H Channel
R Channel
G Channel
B Chaneel
300
250
200
150
100
50
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79

8.2.1. Card ID:2

In this figure, we can see that the target card should be ID:2. Since the pixel difference of four channels are all the lowest.

It means that this card is very different to other cards.


Millions
300
250
200
150
100
50
0
H Channel
R Channel
G Channel
B Chaneel
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81

8.2.2. Card ID: 35

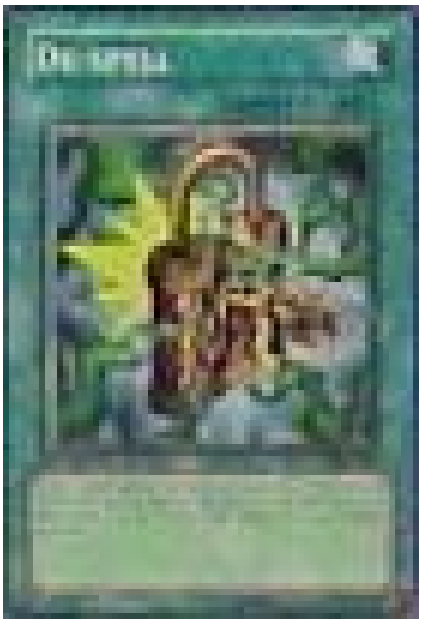
In this figure, it is difficult to say which card is the target card. Since there are two cards with nearly the same result, they are card ID: 29 and card ID: 35. As a result, sometimes it will return 29 and sometimes will return 35.

We found that there are other reasons affect the accuracy.

The first reason is the Image is captured in low resolution. Since we will use approximation to extract the card from the captured image, thus the small error in cards position detection will lead to inaccuracy in recognition process. Especially for those cards with similar color to others.

The second reason is the error of the Reference images.

Since reference images are recorded from camera, and then cut it to a image by human eye. Therefore, a tiny error in preparing reference can lead to a very large difference to image recognition process. Besides, there are also some errors in recording the image.



8.3. Block Matching Algorithm

We have tested different cards against our database. By using the card recognition algorithm described before, we have recognized each card successfully.

In our database, we have chosen 15 cards. Some of them are distinct, and some of them with similar color. The following is the card image of our database.

In the following, we have chosen three cards for the discussion. In our discussion, we use overall pixel different threshold = 1500.

The first card we will discuss is Card 14. The difference of Card 14 among the card images in our database can be shown as follow:

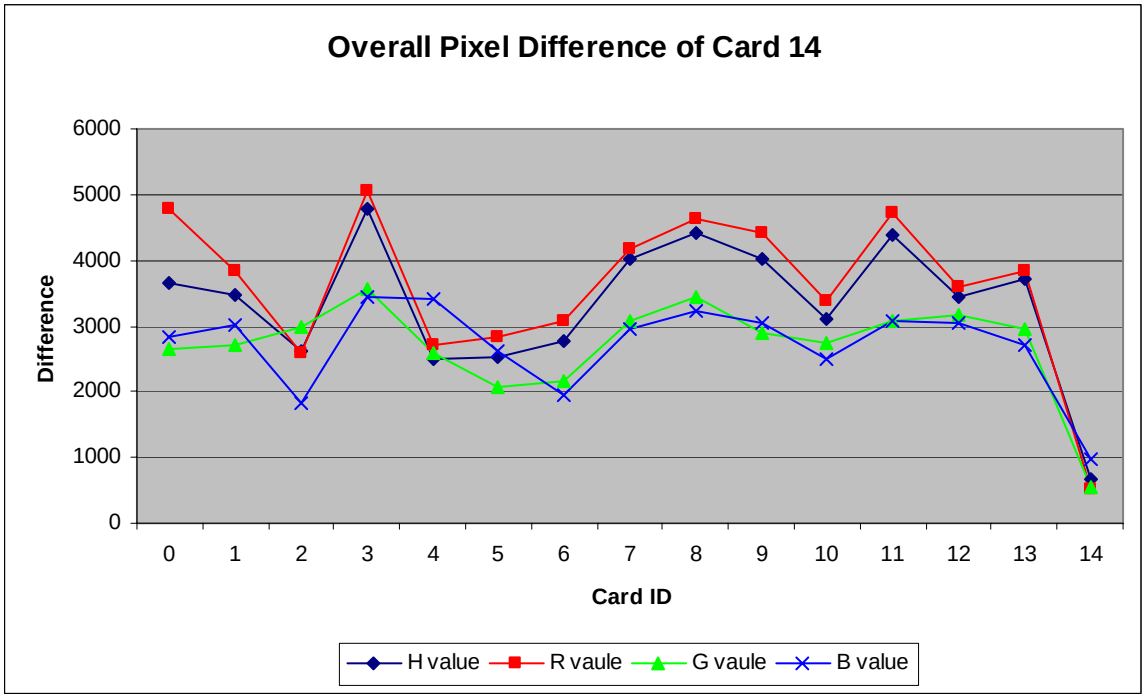


Figure 30: the overall pixel difference of Card14

The first card is Card 14. From the graph above, we can see that the card is distinct to other cards since it got the smallest difference and all below the threshold (1500). None of the other cards in our database is accepted, since their difference for all channels is larger than the threshold.

Figure 31: the image of card 14

The second card we will discuss is Card 13. The difference of Card 13 among the card images in our database can be shown as follow:

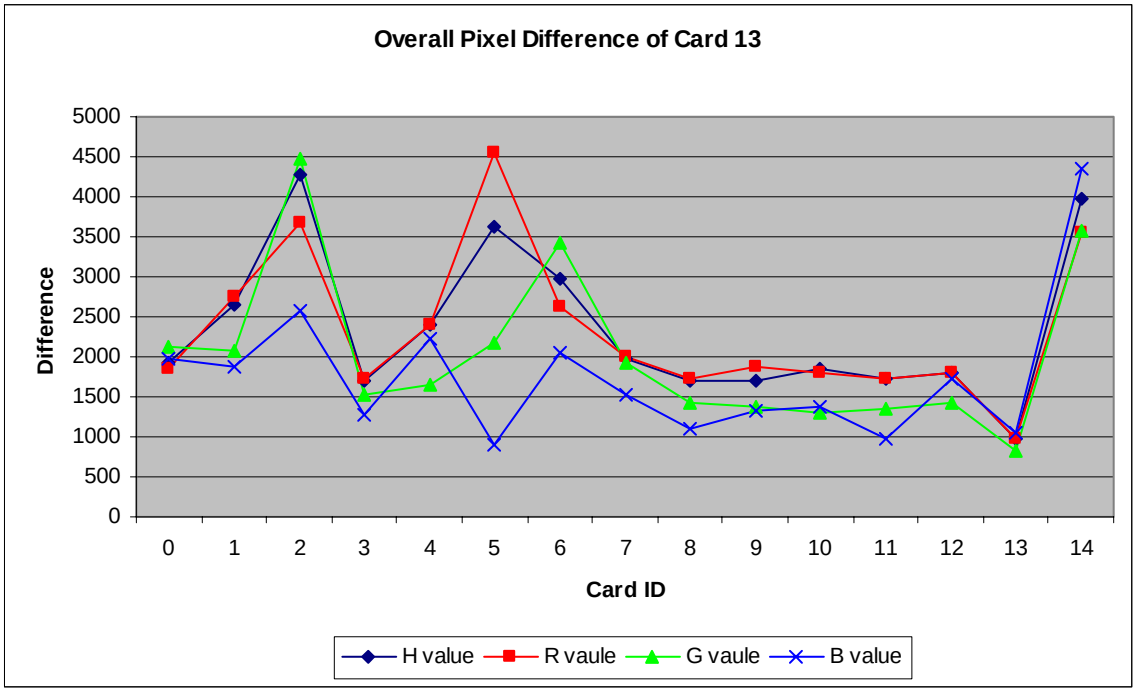


Figure 32: the overall pixel difference of Card 13

This card is not as perfect as Card 14. The blue and green channels are accepted with some invalid image in our database. However, none of them have been accepted for all channels, and only one card in our database can match the all four channels with Card 14.

Figure 33: the image of card 13

The third card is Card 9. The difference of Card 9 among the card images in our database can be shown as follow:

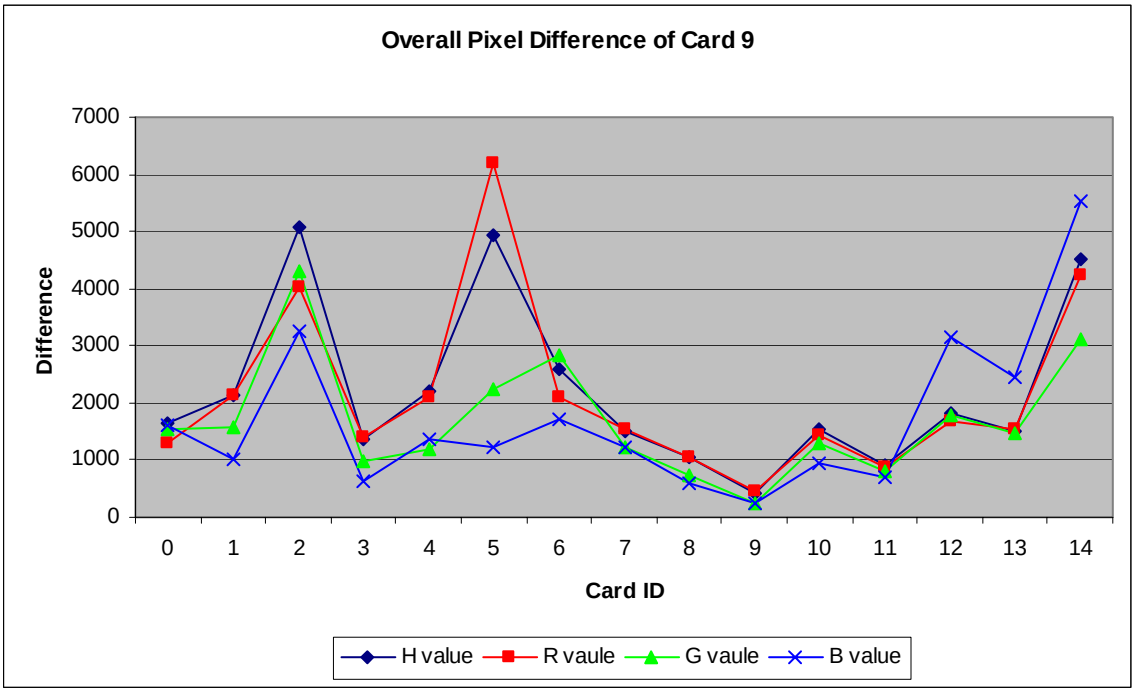


Figure 34: overall pixel difference of Card 9

Figure 35: the image of card 9

Figure 36: the image of card 3

Figure 37: the image of card 8

From the graph above, we can see that the difference among the card images is smaller than the two cases above. In our setting, more than one card is accepted in which all channels are below the threshold, 1500. We can see that Card 3, 7, 8, 9, 10, 11 are accepted. Therefore, we continue on further level of searching. We will sum up the four difference value. Since Card 9 images have the lowest sum of difference, therefore, we regard this image as Card 9.

8.4. Color change

Since the sensitivity of camera is various with the environment, therefore, we do some testing on how the image color changed. We use a LCD monitor to produce different color of background, and what is the changes of the color.

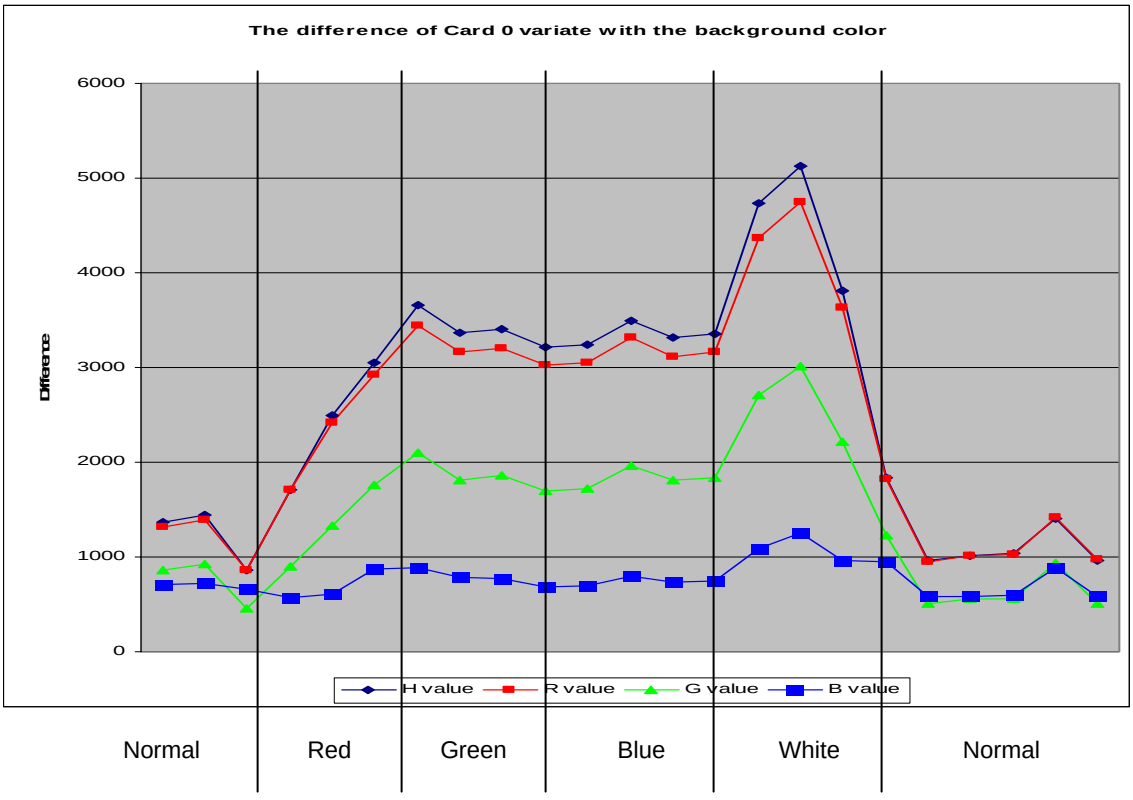


Figure 38: the image of card 14

In this experiment, we change the background color of the LCD monitor by time, to see how the colors change. We use a pale blue color as a normal background (RGB value: R(0), G(64), B(128)). The camera is sensitive with the intensity of light. It will automatically adjust the brightness of the screen captured. It the intensity of light is growing, the camera will reduce the brightness gradually. However, it leads to the problems when we changing the background color of our system.

Figure 39: the image of card 0

In this normal background, since the intensity of light is low, the color responds normally. In a Red Background (RGB value: R(255), G(0), B(0)), the brightness of the screen captured is increase, thus the brightness of the card is decrease. Since the card is going dimmer, the error between the image and the card image in database has increased.

We see that the difference is only slightly changes for red, green and blue background. It means the background color does not affect so much to the cards. However, since the background color is produced by the LCD monitor, the light intensity is high, thus it greatly reduce the brightness of card and enhance cause error.

When we use white color (RGB value: R(255), G(255), B(255)) as background, the LCD monitor produces the maximum intensity of

light. Thus all object captured in the screen becomes dark, and the result becomes further unreliable.

If we change the background color back to normal color, the intensity of light reduces to normal. However, the camera needs time to adopt the changes, thus the difference drops gradually.

9. Contribution of Work

This session concerns about the contribution of my work in this project, and the knowledge I learnt and experience I gained throughout this semester.

Preparation

This Final Year Project, titled “Augmented Reality Table for Interactive Card Game”, requires much knowledge on the field of image processing, image analysis, augmented reality, computer vision and computer graphics.

In summer, I spent much time on studying some background knowledge on augmented reality, image processing and analysis through reading some books and papers. This helps me to understand the project, including the problem we need to solve and what technique can be applied.

Besides, I also started to use Microsoft Visual C++ 6.0 to get familiar with the working environment. I’ve studied Component Object Model (COM) programming in Visual C++, and the DirectX SDK, focus mainly on DirectShow, and try to write some programs to capture and process the video input.

First Term

After I had enough background knowledge to the project, I started to discuss the project with Kevin. We started the implementation by simple hardware setup and simple testing program – using an LCD monitor and webcam and write some simple filters to extract features on the image.

After that, we came up with the architectural design of the system and divided it into four modules. We decide to focus on the Perception Module in the first semester, which is responsible for perceiving input video stream and detect the player input. The work can be divided into calibration, locating cards and identifying cards.

Throughout this semester, we thought of many ideas to improve the both the efficiency and accurate of the system, for example the use of search window and color-based image matching. Some ideas are come from reference papers and some are thought by us. This is a trial and error process and some experiments were done to verify the performance of these methods and algorithms. At last, the system is able to locate cards correctly and identify fifteen cards. Although it performs quite well, there are still rooms for improvement.

Second Term

In the second semester, we changed our focus to the other three modules, namely the Database Module, the Game Core Module and the Game Enhancement Module. At the same time, we also improved and completed the Perception Module.

The most challenging problem that I faced is to design a Game Core Module that is extensible, flexible and generic enough for most of the card games. Instead of brute-force implementation of the game logic, we came up with the rule-based game engine. At that time, we needed to choose between a simple brute-force approach, and a more difficult rule-based approach. Finally we made our decision to use the latter one because it suits our requirement more, and most importantly, we can learn more through the implementation of a rule-based game engine that we were not very familiar with.

Conclusion

After the work of this semester, I have gained a lot of experience using DirectX SDK with Visual C++, and I have learnt many techniques and concepts in image processing, analysis and rule-based system as well.

10. Conclusions

The ART system can be divided into four main modules: the Perception Module, the Generic Card Game Database Module, the Game Enhancement Module and the ART Card Game Core Module.

We mainly focus on the Perception Module, the Generic Card Game Database Module and the ART Card Game Core Module in this year. We have spent much time on familiarizing with the Microsoft Visual C++ and Microsoft DirectX SDK, and study some image process and analysis techniques in the first semester and implementation for these three modules in the second semester. However, we spent less effort on the Game Enhancement Module since the previous three modules are more fundamental for the system but enhancement to the game can be added in the future base on the prototype system.

As a conclusion, we have developed a prototype of the Augmented Reality Table for Card Game, which includes the basic game logic and a finite set of cards in the game. We also provide it with database editors to extend the game in future.

11. Future Work

Color Calibration

Since the color of the image changes with the environment, it will affect the performance of the system significantly. Color calibration is required if the environment is considered to have change.

Game Enhancement

More fancy 3D animations and sound effects can enhance the joy of during game play a lot. So the Game Enhancement Module is the main concern in future improvement

12. Acknowledgement

We would like to express our thanks to Professor Michael R. Lyu, our final year project supervisor, who has given us unlimited support and valuable suggestions throughout the project.

We would also like to thank Mr. Edward Yau, for giving us lots of technical information and ideas in our project.

13. References

[1] Abhay Sharma, "*Understanding color management*", Thomson/Delmar Learning, 2004.

[2] Agrawala, M., Beers, A. C., Bernd Fröhlich, Pat Hanrahan, "*The Two-User Responsive Workbench: Support for Collaboration Trough Individual Views of a Shared Space", SIGGRAPH 97*, August 1997, pp. 327-332.

[3] Alberto Del Bimbo, "*Visual Information Retrieval*", Academic Press, 1999.

[4] Akima, H., "*Bivariate Interpolation and Smooth Serface Fitting Based on Local Procedures*", Communication of the ACM, Vol. 17, No. 1, January 1974.

[5] Alexey V. Nefyodov, "Efficient *Image Matching Algorithms Based on Procedures of Searching for 2D Templates*", SCIA03.

[6] B. S. Manjunath and W. Y. Ma, "*Texture Features for Browsing and Retrieval of Image Data*", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1996.

[7] Bryson, Steve. "*Measurement and Calibration of Static Distortion of Position Data from 3D Trackers*". *Proceedings of SPIE Vol. 1669: Stereoscopic Displays and Applications III* (San Jose, CA, 12-13

February 1992), 244-255.

[8] Caudell, Thomas P. "*Introduction to Augmented Reality*". *SPIE Proceedings volume 2351: Telemanipulator and Telepresence Technologies* (Boston, MA, 31 October - 4 November 1994), 272-281.

[9] Chen, Shenchang Eric, and Lance Williams. "*View Interpolation for Image Synthesis.*" *Proceedings of SIGGRAPH '93* (Anaheim, CA, 1-6 August 1993). In *Computer Graphics*, Annual Conference Series, 1993, 279-288.

[10] Chris Harris and Mike Stephens, "*A Combined Corner and Edge Detector*", the Forth Alvey Vision Conference, 1988.

[11] Crain, I.K., "Computer *Interpolation and Contouring on Two-Dimension Data: A Review*", Geoexploration, Vol. 8, 1970.

[12] Fei Shen and Han Wang, "*A local edge detector used for finding corners*", the Third International Conference on Information, Communications and Signal Processing (ICICS), 2001.

[13] Feiner, S., MacIntyre, B., and Seligmann, D. "*Knowledgebased augmented reality*". Communications of the ACM, 36(7), July 1993, 52-62.

[14] Funt, B.V. and G.D. Finlayson, "*Colour Constant Colour Indexing*", IEEE Trans on Pattern Analysis and Machine Intelligence, 1995.

[15] Grasset, R., Gascuel, J., "*MARE : Multiuser Augmented Reality*

Environment on table setup," SIGGRAPH 2002 p. 213.

[16] Hafner, J., H.S. Sawhney, W. Equit, M. Flickner and W. Niblack, "*Efficient Colour Histogram Indexing for Quadratic Form Distance Functions*", IEEE Trans. On Pattern Analysis and Machine Intelligence, 1995.

[17] Hideki Koiko, Yoichi Sato and Yoshinori Kobayashi, "*Integrating Paper and Digital Information on Enhanced Desk: A Method for Realtime Finger Tracking on an Augmented Desk System*", ACM Transactions on Computer-Human Interaction, December 2001.

[18] Hand, C., "*A Survey of 3D Interaction Techniques*", *Computer Graphics Forum*, 16(5), Dec. 1997, pp. 269-281.

[19] Hu, M.K., "*Visual Pattern Recognition by Moment Invariants*", IRE Transactions on Information Theory, 1962.

[20] J. Canny, "*A Computational Approach to Edge Detection*", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, Nov 1986.

[21] Jain, A.K., "*Fundamentals of Digital Image Processing*", Prentice-Hall, 1989.

[22] K. Rangarajan, M. Shah and D.V. Brackle, "*Optimal Corner Detector*", Computing Vision, Graphics, and Image Processing, 1989.

[23] Krüger, W., Bohn, C.-A., Fröhlich, B., Schüth, H., Strauss, W., Wesche, G., "*The responsive workbench: A Virtual Work Environment*", *IEEE Computer*, July 1995, pp. 42-48.

[24] Liang, K.C. and C.C.J. Kuo, "*Progressive image indexing and retrieval based on embedded wavelet coding*", ICIP'97.

[25] Madritsch, F. and M. Gervautz. "*CCD-Camera Based Optical Beacon Tracking for Virtual and Augmented Reality*". *Proceedings of Eurographics '96* (Futuroscope - Poitiers, France, 26-30 August 1996).

[26] Michael Seul, Lawrence O'Gorman and Michael J. Sammon, "*Practical Algorithms for Image Analysis: Description, Examples, and Code*", Cambridge University Press, 2000.

[27] Mike James, "*Pattern Recognition*", BSP Professional Press, 1987.

[28] Milan Sonka, Vaclav Hlavac, and Roger Boyle, *"Image Processing, Analysis, and Machine Vision",* PWS, 1998,

[29] Milgram, P., Kishino, F, 1994. *"Augmented Reality: A Class of Displays on the Reality-virtuality Continuum." SPIE, Telemanipulator and Telepresence Technologies, 2351* (34). 42-48.

[30] Niblack, W., R. Barber, W. Equitz, M. Flickner, E. Glassman, D. Petkovic and P. Yanker, "*The QBIC project: Querying images by content using colour, texture and shape.*", SPIE 1908, Storage and Retrieval for Image and Video Databases, February, 1993.

[31] Poupyrev, I., Billinghurst, M. Kato, H., May, R.(2000) *Integrating Real and Virtual Worlds in Shared Space.* In proceedings of the 5th International Symposium on

[32] Artificial Life and Robotics (AROB 5th'00), Oita, Japan, 26-28 January, 2000, Vol. 1, pp. 22-25.

[33] R. Gonzalez and R. Woods, "*Digital Image Processing*", Addison-Wesley Publishing Company, 1992.

[34] Ronald T. Azuma, *"A Survey of Augmented Reality"*, Teleoperators and Virtual Environments 6, 4 (august 1997), 355-385.

[35] S.M. Smith and M. Brady, "*SUSAN – a New Approach to Low Level Image Processing*", 1997.

[36] Shahzad Malik, Chris McDonald and Gerhard Roth, "*Hand Tracking for Interactive Pattern-based Augmented Reality*", the International Symposium on Mixed and Augmented Reality, 2002.

[37] Smith, G., Mariani, J., "*Using Subjective Views to Enhance 3D Applications*", *Proceedings of VRST'97*, 1997, pp. 139-146.

[38] Stapleton C. B., Hughes C. E., Moshell M. "*Mixed Reality and the interactive imagination",* Swedish American Simulation Conference, 2002.

[39] Swain, M.J. and D.H. Ballard, "*Colour Indexing*", International Journal

of Computer Vision, 1991.

[40] Szalavari, Z., Eckstein, E., Gervautz, M., "*Collaborative Gaming in Augmented Reality*," *Proceedings of VRST'98*, pp.195-204.

[41] Szalavári, Zs., Eckstein, E., Gervautz, M., "Studierstube" – An Environment for Collaboration in Augmented Reality, *Virtual Reality – Research Development and Applications*, 3(1), 1998, pp. 37-48.

[42] The, C.H. and R.T. Chin, "*On Digital Approximation of Moment Invariants*", Computer Vision, Graphics and Image Processing", 1986.