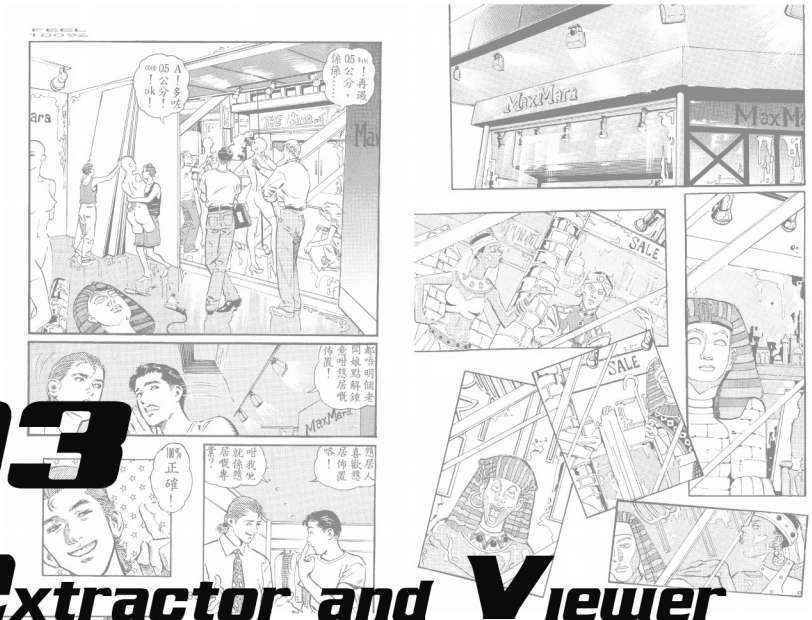
Supervised by
Prof. Michael R. Lyu



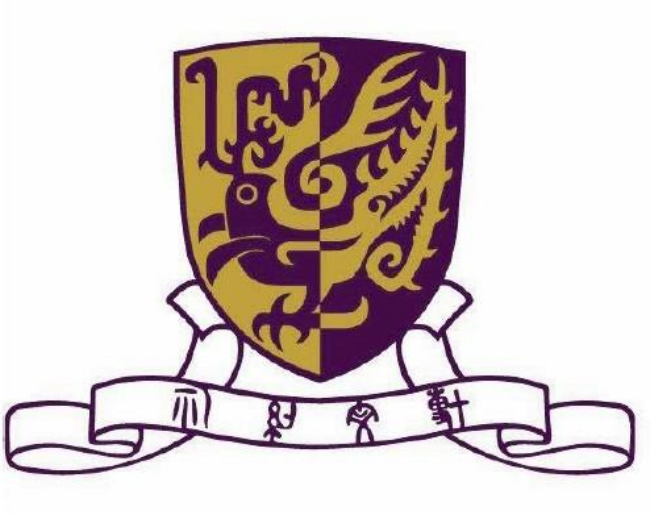
LYU1003

Comic Panel Extractor and Viewer

for Enhancing Accessibility on iOS4

CUHK CSE
CHEUNG Kam Shun
1008619982

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



2010-2011
Final Year Project (2nd Term)

LYU1003
Comic Panel Extractor and Viewer for iPhone

CHEUNG Kam Shun
1008619982
kscheun8@cse.cuhk.edu.hk

Supervised by
Prof. Michael R.Lyu

Abstract

This report covers our study and progress in image processing for comic panel extractor in this year. We will give a detailed report on what we have studied through out the two terms.

It begins with an introduction about our project. We will give some background information about comic reading on mobile device, the motivation, the objective of our final year project as well as development environment related information.

As a full report, it is then followed by a part for background researches done in last term. We will first introduce several approaches that are proposed for similar system. Then it is followed by the analysis of some common or essential algorithms for image processing that might need, e.g. Watershed Segmentation, etc.

For the product design part, we will give an overview of our system architecture. We will go through every assumption that we have made for those specific parts, as well as briefly introducing each component.

After this, we will be focus on our development progress as well as the experimental results, difficulties we faced and current limitations. The progress part will be devided into two sub parts – review for works done in last term and the progress for this semester.

As we use Dropbox to share our paper works, editing each other's work might be involved. Therefore, this report will contain both of ours work, together with a part for stating contribution of work.

Table of Contents

Chapter 1 Introduction

In this chapter, we will give some background information of this topic as well as the project motivation and objective. Some development environment related information will be given in this chapter also.

This chapter is constructed as following list:

- Background Information
- Motivation
- Objective
- Development Environment
- Runtime Environment

1.1 Background Information

Reading comic is a hobby to lots of Hong Kong teenagers, especially those who play computer games. However, the most common way in Hong Kong that people get their comics is not to buy them, but to view a softcopy of the comic.

Figure 1.1 - A sample page of a famous Japanese comic

There are usually two ways to view the softcopy of a comic. The first way is to visit an online comic website. There are plenty of online comic services provided by local websites or websites of the Mainland, e.g. Dm5, Manmee, kukudm, 8comic, etc. Another way is through online forums. There are also lots of online forums that provide free comic download, e.g. 2000FUN, KTXP, HKORZ, etc. In this project as well as the rest of this report, when we talk about comics, we mean downloaded comics, which are scanned pictures.

1.2 Motivation

Most people read comics on their PCs in the past few years. However, with the prevalence of mobile devices nowadays, more and more people read comics on these devices. Tablet PCs, especially iPad, successfully open a market for mobile comics.

The success of iPad comic raises a question – can we have smart phone comic? Smart phone comic then became a hot issue to the comic industry as well as many Apps developers. Due to the dimensions of the iPad, the page size is roughly 75%-80% of an actual comic book page, this is acceptable to most comic reader. However, when it comes to smart phone comic, screen size becomes a problem. There exist some comic Apps on the market. In general, there are two kinds of these Apps.

The first kind is usually produced by the comic industry. It has finely tuned the user interface, the comic panel size as well as panel sequence for comic reading on a specific device. The screen size problem has then been specifically handled. Hence, it provides a nice

experience to the user. However, the main problem of this kind of Apps is that, the comic reader usually bundled with a specific comic series and cannot support comics from the third party.

The other kind of comic readers usually target on scanned pictures. However, most of them act as a normal image viewer only. Due to the screen limitation of the smart phones, reading traditional page-sized comic images through these Apps would be troublesome because of the frequent manual scrolling and zooming.

In the following table, we will compare the comic reading platforms in detail.

Platform	Advantage	Disadvantage
Traditional PC	● Provide easiest way to get the comic resource	● Default image viewer is not powerful enough ● Lack of mobility
Tablet PC	● Higher mobility comparing to PC ● Similar page size to original comic	● Need extra support to get and manage the comic resource
Smart phone	● Highest mobility ● Highest market share	● Screen size too small ● Need extra support to get and manage the comic resource

Table 1.1 – Comparison among comic reading platforms

An existing comic reader either acts as no more than a picture viewer or bundles with limited comic resource. Considering the considerable market share and the high mobility of smart phone, we believe it has the potential to share the comic market with tablet PC.

In order to achieve this, the first question is how we can make use of the existing comic resource on the Net. Therefore, we will implement a comic panel extractor and viewer accepting scanned pictures of existing comic resource on the Net as input and generating suggested reading sequence automatically.

Figure 1.2 - Size Comparison. The screen size of a cell phone is only about ¼ of a traditional comic page

1.3 Objective

The objective of our project is to:

- Analyze the existing image processing algorithms for comic panel extraction,
- Implement comic panel extraction and generate suggested reading sequence and,
- Implement a simple comic viewer for reading the extracted panel in suggested reading sequence.

1.4 Development Environment

Content	Environment & Tool
Algorithm analysis	• Visual Studio 2008, Microsoft Windows • C with OpenCV
Viewer implementation	• Xcode, Mac OS, iOS4.1 • Objective C • C++
Server	• PHP with MySQL, Amazon Linux AMI

Table 1.2 – Development Environment

Most of the algorithm investigation are carried out on the PC environment first as this platform has less restriction and the debugging is much more convenient.

For the implementation of the Viewer App, most of the works are done on a Mac with IPhone simulator and the testing part is carried

out on the iOS.

Figure 1.3 - IPhone Simulator

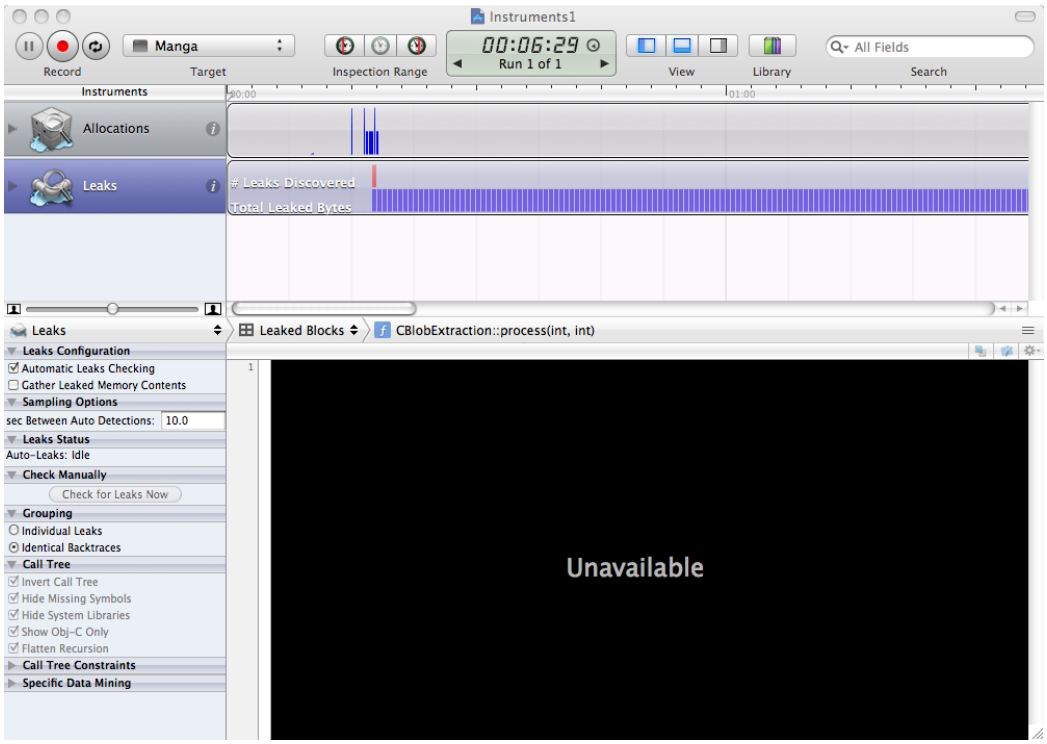


Figure 1.4 - Instruments-A Profiler for iOS

1.5 Runtime Environment

The iOS developed by Apple Inc. is chose as the target platform of the project. iOS is the operating system of iPhone, iPad and iPod touch. The development on iPad is a bit different from others so the development on iPad will not be focused. Moreover, the screen size of iPad is much larger, so a specific viewer on iPad is not so necessary.

The first version of the system released on 2007. Originally it is called the "iPhone uses mac OS X". It runs on ARM family processor and is build on top of the Unix-like kernel Darwin. There are four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer.

Figure 1.5 - Home Screen Capture of iOS

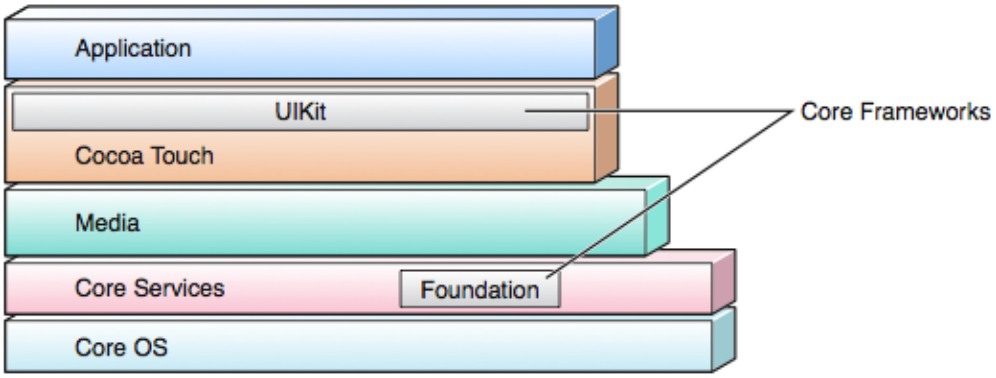


Figure 1.6 - Structure of iOS

The main reason to choose iOS is that iOS is mature in terms of both amount of users and development environment.

Multi-tasking is a very important new feature to iOS that is beneficial to every developer. Computation power also makes progress. With multi-tasking as well as the more advanced computation power, more and more complicated calculations can be done on iPhone. For instance, iPhone might be treated as an I/O device and it might need a PC to be the server for complicated calculations. This situation will apparently change after iOS4.0 came to the world.

Figure 1.7 - Task Bar for Multitasking

iOS is a popular mobile operating system nowadays. By a survey from Nielsen, iOS owns the second largest market share on the smart phone market, following the RIM developed by Blackberry. On top of it, there is an increasing trend on the number of user of iPhone. Besides, not only iPhone uses iOS system, the iPod touch is also running iOS. Therefore the total number of iOS distribution is much higher than the statistic.

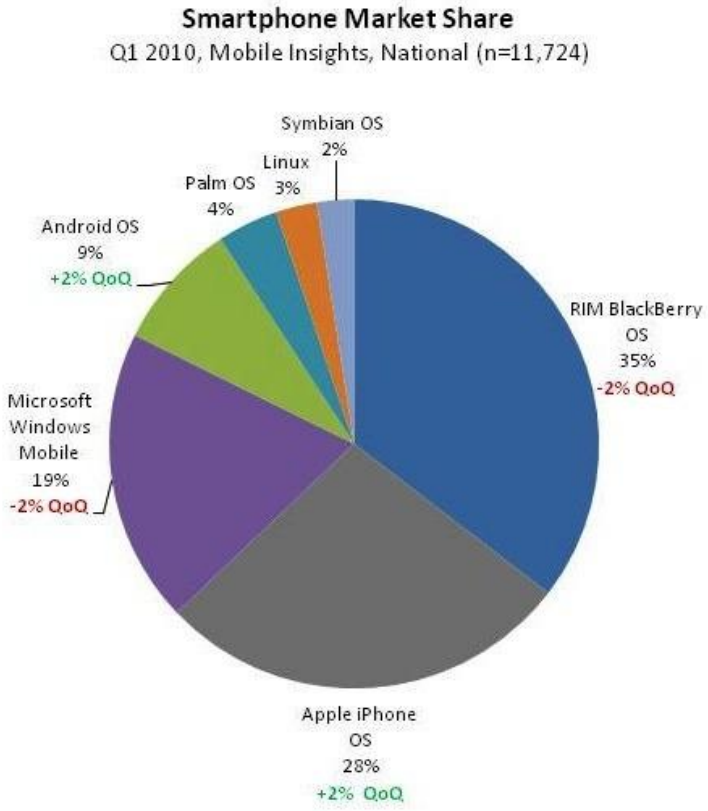


Figure 1.8 - Market Share Statistics of Smartphone from Nielsen[1]

Apart from it, the development environment of iOS is also very mature. The IDE, XCode has been developed for a long time since the

[1]http://big5.xinhuanet.com/gate/big5/news.xinhuanet.com/eworld/2010-06/07/c_12189124.htm

selling of Mac OS X. The debugger is the infamous GDB. The base framework, Cocca Touch, provided many useful functions and APIs. Moreover, a lot of 3rd party library framework has been developed. One important example is the OpenCV ported on the iOS. This facilitates the development process a lot.

Also, the documentation supported by Apple Inc. is sufficient. All the documents can be found on the Apple Development Connection website and it is free of charge.

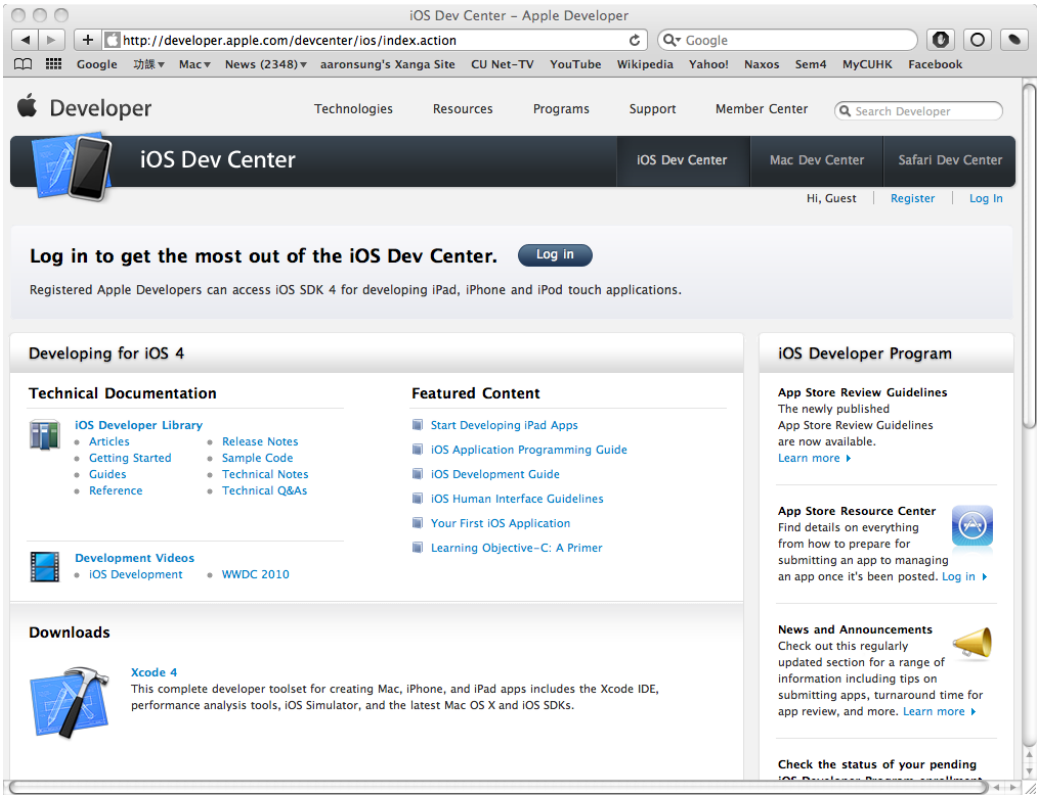


Figure 1.9 - Home Page of iOS Dev Center

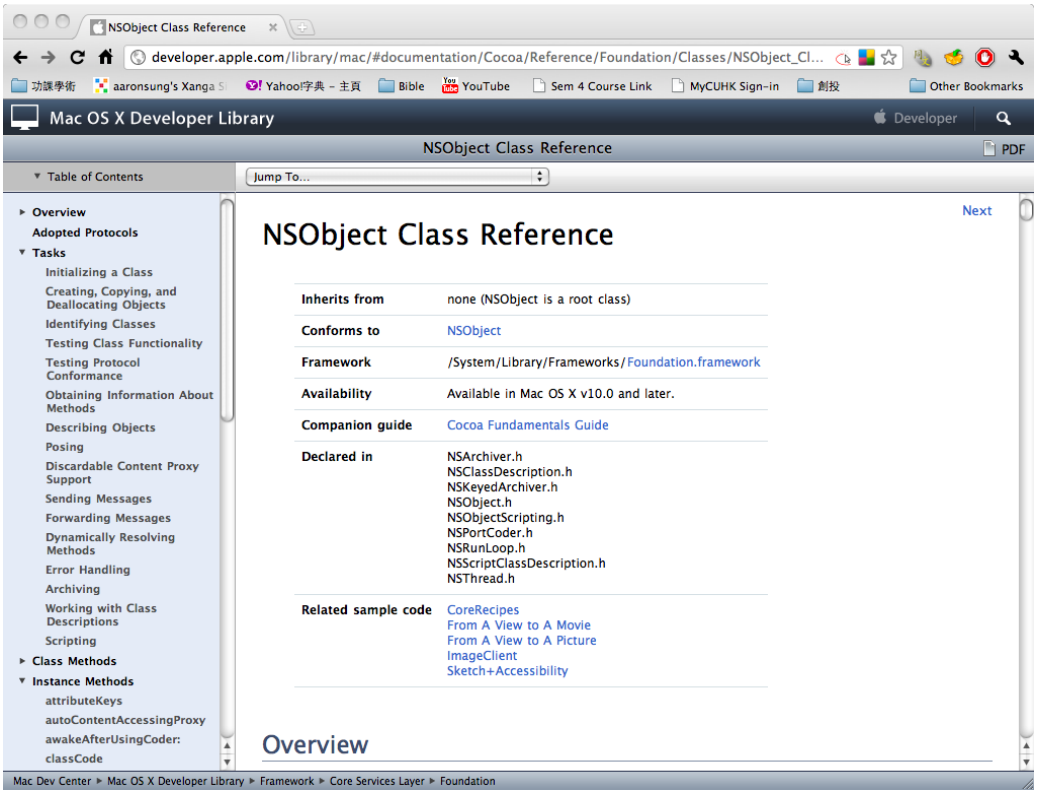


Figure 1.10 - A Reference Page from Developer Library

Last but not least, the hardware platform of iOS devices are unified and standardized. It is especially important for the resolution of the screen. The resolution of non-iPad iOS device is either 480x320 or 960x640. 960x640 is a double of 480x320, that means the layout can remain the same in all kind of non-iPad iOS devices. This can let the developer to focus more on the functionality instead of the compatibility.

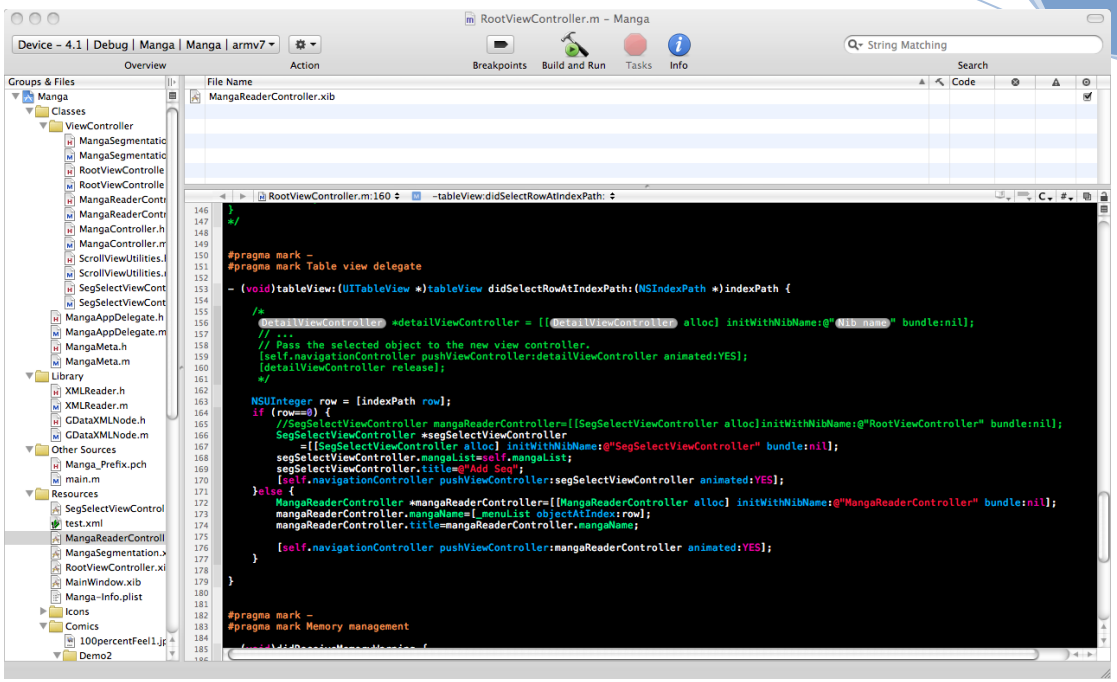


Figure 1.11 - Screen Capture of the XCode IDE

Chapter 2 Background Researches

In this chapter, we will describe our background researches as the following list:

- Different Existing Approaches
  - Multi-pass Approach
  - Recursion Approach
- Commonly Used Algorithms
  - Contour Finding
  - Hough Line Finding
  - Watershed Segmentation
  - Flood Fill
  - Connected Component Labeling

2.1 Different Existing Approaches

Due to the popularity of comic reading on mobile devices, the problem has been investigated by others before. There are two main approaches proposed by current researches. The first is multi-pass approach while the other one is recursion method.

In the following paragraphs, we will talk about these two approaches.

2.1.1 Multi-pass Approach

Multi-pass approach involves segmentation algorithm with domain specific pre-processing and post-processing[2]. The following list shows the main steps of one example of panel extraction:

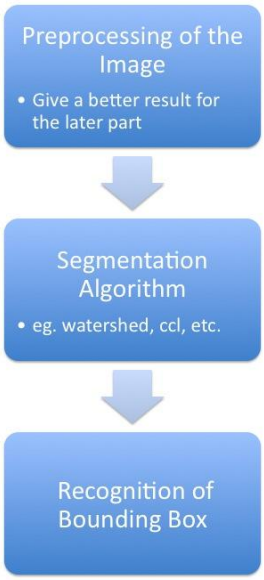


Figure 2.1 – Multi-pass Approach Flow Chart

i. Convert image to grayscale,
ii. Threshold on white for capturing the whole background,
iii. Fill the background to black,
iv. Compute the different with the previous image,
v. Perform Watershed Segmentation on the binary image,
vi. Compute the bounding boxes covering each detected region,
vii. Remove small region and merge highly overlapping regions.

The first 4 steps are standard image processing primitives that ensure the image is under best condition for segmentation. Step vii is usually needed to reduce error. Note that the above algorithm assumes the panels are disjoint and the background color is white.

Besides the use of watershed as the segmentation, a modified connected component labeling algorithm[3,4] is also proposed as a segmentation algorithm.

In general, the multi-pass approaches are faster in speed. It is due to the heuristic division line detection in recursion

[2] C. Ponsard, V. Fries - *Enhancing the Accessibility for All of Digital Comic Books*
[3] F. Chang, C-J. Chen, and C-J. Lu. —*A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique",* Computer Vision and Image Understanding, 93(2):pp. 206-220, 2004.
[4] K. Arai & H. Tolle - *Automatic E-Comic Content Adaptation*

approaches that will be discussed later. One of the support evidences is the experiment taken by K. Arai & H. Tolle:

Comic Page	Processing Time (in seconds)		
	[6]	[10]	Our Method
1 comic page offline	3	25	0.250
1 comic page online	-	-	0.513
30 comic page offline	90	750	10
30 comic page online	-	-	16

Figure 2.2 – Processing time of the recursion approach

Where the "Our method" means the modified CCL and "[10]" is the method mentioned in the paper of H.C. Chung *et el.*[4]

For mobile devices, the different in performance is significant.

2.1.2 Recursion Approach

In this approach, uniform color stripes are first identified and used as separators to segment the color comic.

In this approach, uniform color stripes are first identified and used as separators to segment the color comic. A line can be represented as follow:

$$\rho = x\cos\theta + y\sin\theta$$

or

$$ax+by+c=0$$

Then different value of {ρ, θ} or {a, b, c} are tired to get a probability of being a stripe. If the value is higher than the threshold, then it will be treated as a stripe.

After that, a page will be segmented into sub-regions in a recursive manner. Panels are recognized as the sub-regions that cannot be further segmented. The structure of the panels can be determined through the panel extraction process which turns out to give us a suggested reading sequence.

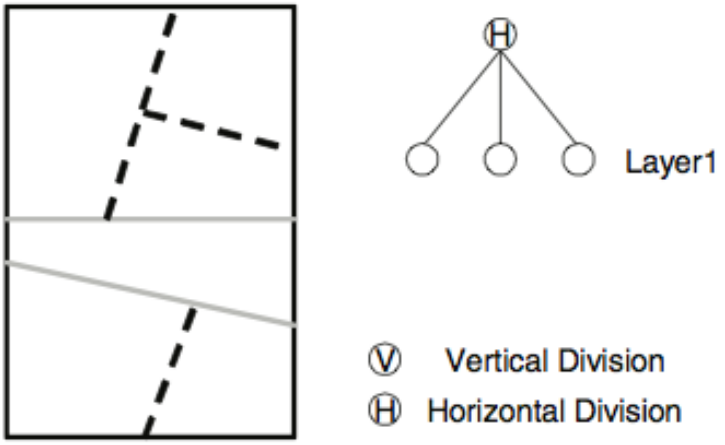


Figure 3: The structure of a comics page at first step (left) and the tree structure of it (right).

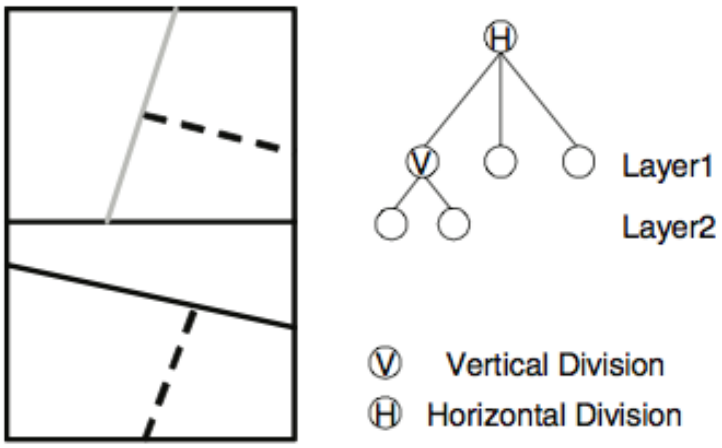


Figure 4: The structure of a comics page at second step (left) and the tree structure of it (right).

Figure 2.3 - Generation of the reading sequence with extracted panels simultaneously.[5]

The following flow chart illustrates this approach in a clearer way:

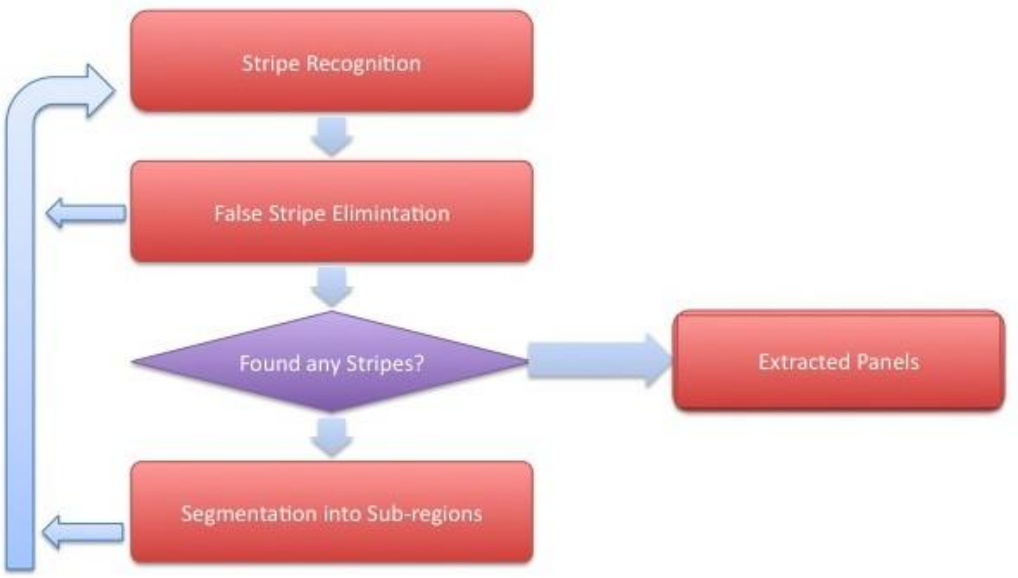


Figure 2.4 – Flow chart of the recursion approach

Two methods of stripe detections are proposed. The first one is by calculating the histogram[5] while the second one is calculate the density gradient[6].

From the research of Chung[5], the histogram of a stripe will have a sharp peak comparing to non-stripe.

[5] T. Tanaka et el., *Layout Analysis of Tree-Structured Scene Frames in Comic Images*, IJCAI'07 Proceedings of the 20th international joint conference on Artifical intelligence
[6] H.C. Chung, H. Leung & T.Komura, "*Automatic Panel Extraction of Color Comic Images*," Advances Multimedia Inform. Processing – Pcm 2007, vol. 4810/2007, pp. 775–784, 2007.

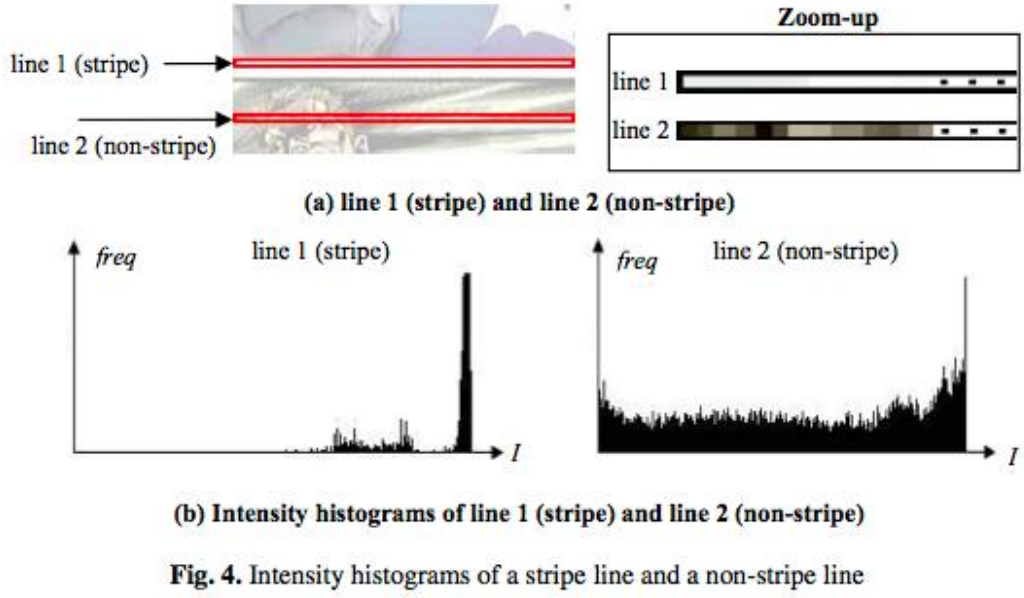


Figure 2.5 – A sharp peak occur on the right side for a stripe on white background comic (captured from the research paper of Chung [6])

A threshold of 90% of pixels around the peak in the intensity histogram is being suggested. Then it will be determined whether it is a false stripe. Two conditions of false stripe occurrence are being suggested.

- False stripes often appear inside small panel regions.
- A real stripe has a large contrast with the surrounding pixels. [6]

If the line stripe passes the two conditions above, then it will go through the segmentation part.

For the gradient density approaches suggested by T. Takana *et al.*, the density gradient gθ (x, y) of the θ direction at the position (x, y) on an image is given as follows:

$$g_\theta(x,y) = g_x(x,y)\cos\theta + g_y(x,y)\sin\theta$$

where $g_x(x, y)$ is the density gradient of the X-axis direction,

and g_y (x, y) is the density gradient of the Y-axis direction. The accumulation value A(ρ,θ) of the density gradient value of the θ direction is obtained along the straight line L(ρ, θ). The density gradient value A(ρ, θ) of the straight line L(ρ, θ) on an image is given as follows:

$$A(\rho,\theta) = \sum_{(x,y)\in S(\rho,\theta)} g_\theta(x,y)$$

In order to reduce the false detection of division lines, the value must undergo some process first. Normally, the central part of the image is expected that there is the high possibility a division line is located. Therefore, in order to raise the weight of the central part of the image, a weighted accumulation value of gradient AW (ρ, θ) is obtained by multiplying A(ρ, θ) by a Gaussian function G(ρ) on ρ :

$$AW(\rho,\theta) = G(\rho)A(\rho,\theta)$$

where G(ρ) = exp(−ρ2/σ2), and σ = [Image height]/4.

After that, the evaluation value H_W (ρ, θ) of segment without white pixels of some extent length is lowered by multiplying A_W (ρ, θ) by the ratio of the length of the longest white pixel column to that of the segment. In order to avoid duplicate detection, the straight line judged as division line is stored in a cache list. The straight lines which are similar to the straight lines stored in the cache list are not chosen as a division line in the further division line detection. Under the constraint, the candidate of division line is obtained as

$$(\rho_p, \theta_p) = \arg\max_{\rho,\theta} H_W(\rho, \theta).$$

Whether $L(\rho_p,\theta_p)$ is division line or not is decided by whether $H_W(\rho p, \theta p) \geq h_{th}$ or not. As h_{th} is raised, division lines detected are limited more. We decided h_{th} heuristically. In the implementation, θ is changed from -90° to 90° at 1° step, and ρ is changed within the image at 1 pixel step.[5]

Despite advantages of the recursion algorithm, it is worth to note that the recursion approach has two drawbacks.

- It may not deal with comics with just one panel on a page correctly.
- The division lines on the page which are not in uniform color will fail.

To conclude, the outline of method, advantages and disadvantage as list as follows:

	Description	Pros	Cons
Multi Pass Approaches	The whole image undergo a few stages of processing	Relatively less computation power required	Fail if there are overlapping panels
Recursion Approaches	The image goes through a process-divide recursion	Can generate the reading sequence at the same time	• Relative slower • Page with one panel only will fail

Table 2.1 – Pros & Cons of the two approaches

2.2 Commonly Used Functions

In this section, we will highlight some functions that might be useful for our project. We will describe them in a theoretical manner, including how they work and what parameters they require.

2.2.1 Contour Finding

OpenCV has a series of functions for contour finding. We have studied the basic one, cvFindContour().

This function retrieves contours from the binary image. The contours are a useful tool for shape analysis, object detection and recognition. Therefore we can get some useful information from the contours for comic panel extraction.

The function will return the number of retrieved contours and fill up the first contour pointer as well. This pointer points to the first contour founded or simply NULL if the image is totally in black and no contour is found. Other contours can be reached from first contour using the h next and v next links.[7]

There are some other functions that support cvFindContour(), e.g. cvCreateContourTree(), cvDrawContours(), etc. Contours found belong to different connected components, which can be used to generate bounding boxes to give a rough sight of the panel structure of the whole image.

[7] OpenCV Reference Manual, v2.1, March 18,2010

2.2.2 Hough Line Finding

Hough line finding is another important technique that might help us to gather useful information for panel extraction. We have examined cvHoughLines2() which finds lines in a binary image using a Hough transform.

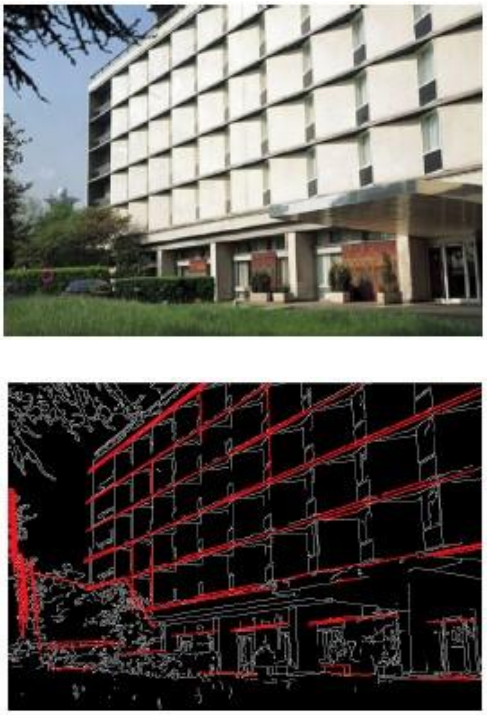


Figure 2.6 - Demonstration for Hough line detection

This function accepts 3 methods as transform variant:

- CV_HOUGH_STANDARD
- CV_HOUGH_PROBABILISTIC
- CV_HOUGH_MULTI_SCALE

CV_HOUGH_STANDARD is classical Hough transform. Under this method, every line is represented by two floating-point

numbers (p, q), where p is the distance between the origin and the line, and q is the angle between the normal of the line and x-axis.

CV_HOUGH_PROBABILISTIC is the probabilistic Hough transform. It will be more efficient if the target image contains a few long linear segments. This method returns segments, represented by starting point and end point, instead of the whole line.

CV_HOUGH_MULTI_SCALE is the multi-scale variant of the classical Hough transform. The lines are thus encoded in the same way as in CV_HOUGH_STANDARD.[7]

Noted that Hough lines do not equal to the contours of the target image, they are the supplementary information to the contours.

2.2.3 Watershed Segmentation

Watershed segmentation is also a very important part to our study.

"A drop of water falling on a topographic relief flows along a path to finally reach a local minimum. Intuitively, the watershed of a relief corresponds to the limits of the adjacent catchment basins of the drops of water."[8] This illustrates a brief idea of watershed in image processing.

In image processing, when it comes to watershed, we are usually talking about segmentation. Different watershed lines can be computed for segmentation purpose. Watershed segmentation requires contour information first. We can imagine that once we drop a drop of water onto the image, it spreads out until reaching the contour.

In OpenCV, cvWatershed() implements one of the variants of the Watershed algorithms, non-parametric marker-based segmentation.

Users are required to manually input some markers roughly indicating different regions of the target image. These markers are treated as seeds of the future image regions while the other pixels, which are not marked, should be defined by the algorithm and set to 0's. Each pixel in markers will then be set to one of values of the seed component, or -1 if it is at boundaries between the regions.[7]

[8] The watershed transformation for multiresolution image segmentation, S. Wegner, T. Harms, J. H. Builtjes, H. Oswald and E. Fleck

In other words, all pixels having a value greater than 0 are supposed to belong to a region, they are segmented according to the value assigned.

2.2.4 Flood Fill

The traditional flood fill algorithm takes three parameters:

- A start node,
- A target color and,
- A replacement color.

The algorithm looks for all nodes in the array which are connected to the start node by a path of the target color, and changes them to the replacement color. There are many ways to implement this algorithm, but they all make use of a queue or stack data structure, explicitly or implicitly. Here shows one implicitly stack-based recursive flood-fill implementation example for a two-dimensional array:

Flood-fill (node, target-color, replacement-color):

1. If the color of *node* is not equal to *target-color*, return.
2. Set the color of *node* to *replacement-color*.
3. Perform **Flood-fill** (one step to the west of *node*, *target-color*, *replacement-color*).

Perform **Flood-fill** (one step to the east of *node*, *target-color*, *replacement-color*).

Perform **Flood-fill** (one step to the north of *node*, *target-color*,

replacement-color).

Perform **Flood-fill** (one step to the south of *node*, *target-color*, *replacement-color*).

4. Return.

In OpenCV, Flood-fill is also implemented and it requires user to input a point in target image as seed. cvFloodFill() then fills a connected component with a given color.

2.2.5 Connected Component Labeling

Within an image, there are plenty of regions, some are connected while some are not, how can we label all the regions connected by unique labels? Connected component labeling is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. It should not be confused with segmentation.

In general, there are two methods for connected component labeling. One of them is recursion. It is pretty straightforward. You take a pixel, and check its neighbors for connectivity. However, it's inefficient. As the image size grows, the time taken by the algorithm increases rather quickly.

Another classical method is designed by Rosenfeld and Pfaltz in 1966 using the union-find data structure. We describe it classical as it uses a result from the classical algorithm for connectedness in graph theory. This algorithm consists of two passes. In the first pass, the algorithm goes through each pixel. It checks the pixel above and to the left. And using these pixel's labels (which have already been assigned), it assigns a label to the current pixel. And in the second pass, it cleans up any mess it might have created, like multiple labels for connected regions. This is the overview of how the algorithm works.

The details of the algorithm will be described as below:

The First Pass

In the first pass, every pixel is checked. One by one, starting at the top left corner, and moving linearly to the bottom right corner.

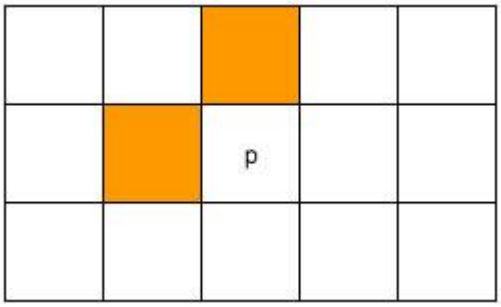


Figure 2.7 – Illustration of Connected Component Labeling algorithm, First Pass

If we're considering the pixel 'p', we can only check the orange pixels. Thus, at any given time, we only need to have two rows of the image in memory. This helped in memory efficiency in the past, but now we have at least 2GB RAMs. Hence, it is not an issue these days.

We will go through each step of the first pass one by one as the following paragraphs:

Step 1

Here we check if we're interested in a pixel or not. If the pixel is a background pixel (its value is zero, or whatever other criteria you want), we simply ignore it and move on to the next pixel. If not, you go to the next step.

Step 2, 3

For step 2 we fetch the label of the pixels just above

and to the left of 'p' and store them (into A and B here). Now, there are a few possible cases here:

The pixel above or/and the left aren't background pixels: In this case, things proceed as usual. We just go to the next step. A or/and B will have actual values (the labels).

Both pixels are background pixels: In this case, we cannot get labels. Therefore, we create a new label, and store it into A and B.

Step 4, 5

We figure out which one is smaller: A or B and then we set that label to pixel 'p'.

Step 6

Suppose we have a situation where pixel above has a label A and the pixel to the left has a label B. However, we know that these two labels are connected as the current pixel "connects them".

Thus, we need to store the information that the labels 'A' and 'B' are actually the same. And we do that using the union-find data structure. We set the label 'A' as the child of 'B'. Using this information, the algorithm will clean up the mess in the second mess.

The only thing we need to remember is that the smaller label get assigned to 'p', and the larger number becomes a child of the smaller number.

Step 7

Go to the next pixel.

The second pass

Again, the algorithm goes through each pixel, one by one. It checks the label of the current pixel. If the label is a 'root' in the union-find structure, it goes to the next pixel.

Otherwise, it follows the links to the parent until it reaches the root. Once it reaches the root, it assigns that label to the current pixels.[9]

To conclude, connected component labeling is used in computer vision to detect connected regions in binary digital images, although color images and data with higher-dimensionality can also be processed. [10,11] Blob extraction is generally performed on the resulting binary image from a thresholding step. Blobs may be counted, filtered, and tracked. Blob extraction is related to but distinct from blob detection.

[9] http://www.aishack.in/2010/03/connected-component-labelling/
[10] H. Samet and M. Tamminen (1988). "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees". IEEE Transactions on Pattern Analysis and Machine Intelligence (TIEEE Trans. Pattern Anal. Mach. Intell.) 10: 579. doi:10.1109/34.3918
[11] Michael B. Dillencourt and Hannan Samet and Markku Tamminen (1992). "A general approach to connected-component labeling for arbitrary image representations}". J. ACM.

Chapter 3 Product Design

In this chapter, we will focus on product design and anything related to it.

The structure of this chapter is shown as below:

- Convention
- Assumption
- System Architecture Design
- UI Design

3.1 Convention

We will talk about convention in this part clarifying some terms on a comic page.

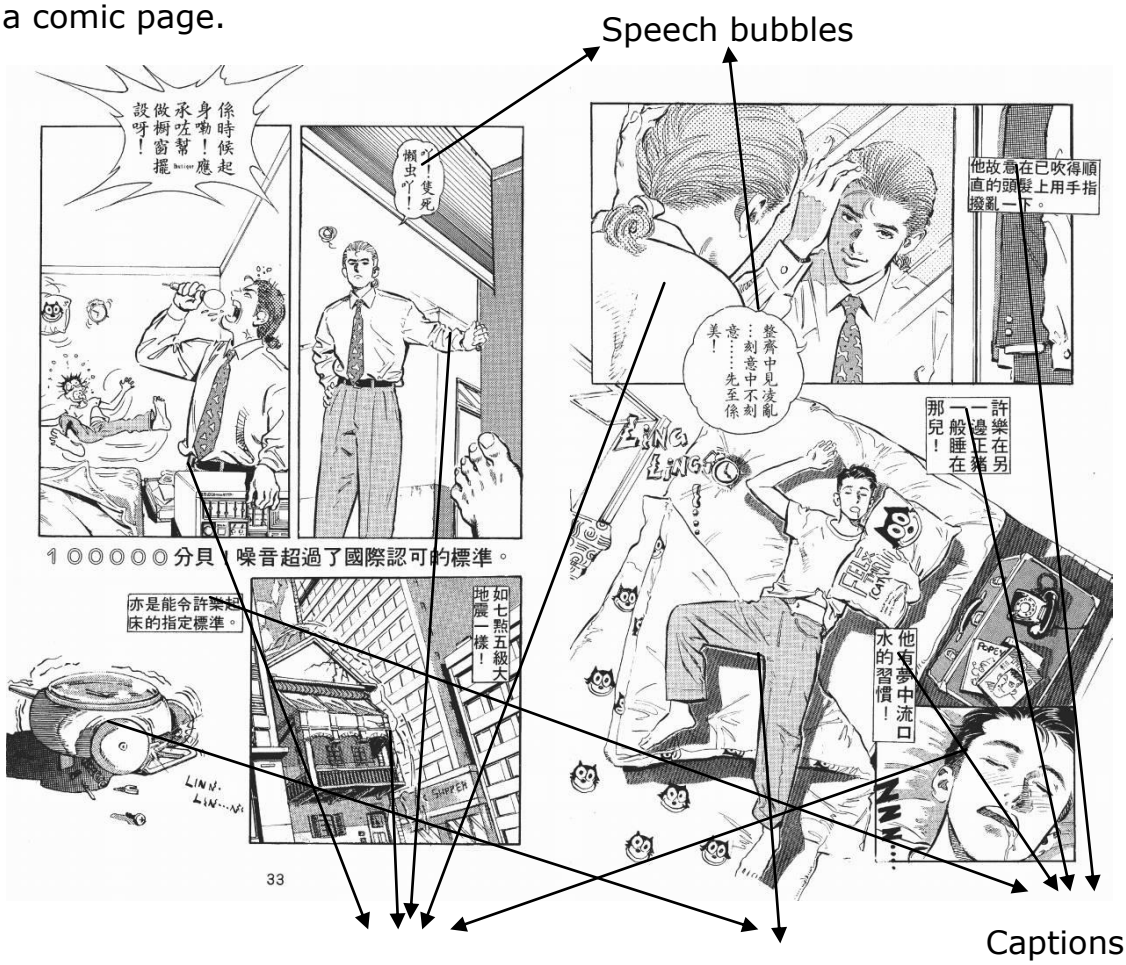


Figure 3.1 – Sample comic page

- *Panels* are individual frames that consist of a single drawing depicting a frozen moment.
- *Captions* are generally for narrative purpose. They are usually rectangles located near the edge of a panel.
- *Speech bubbles* indicate what a character says. In-panel speech bubbles are for characters that appear in the panel. Off-panel speech bubbles are for characters that cannot be seen in the panel.
- *Thought bubbles* are similar to speech bubbles but for thought expression only. Thought bubbles are not shown in the above picture.

3.2 Assumption

Due to the difficulties found on the research, some assumptions are made to tackle the problems. The project will start from some simpler cases, and investigate more complex cases later.

To simplify the complexity, we assume:

- No overlapping on panels of the comic panels is assumed.
- The background of the comic is assumed to be white.
- The comic must be in grayscale.

Comics with overlapping panels will be more difficult to perform traditional segmentation algorithm such as watershed. Modifications on algorithm must be done to deal with those comics.

The second and third assumption is also made to reduce the complexity of the problem.

Figure 3.2 - A Comic with non-white background

3.3 System Architecture Design

3.3.1 Viewer

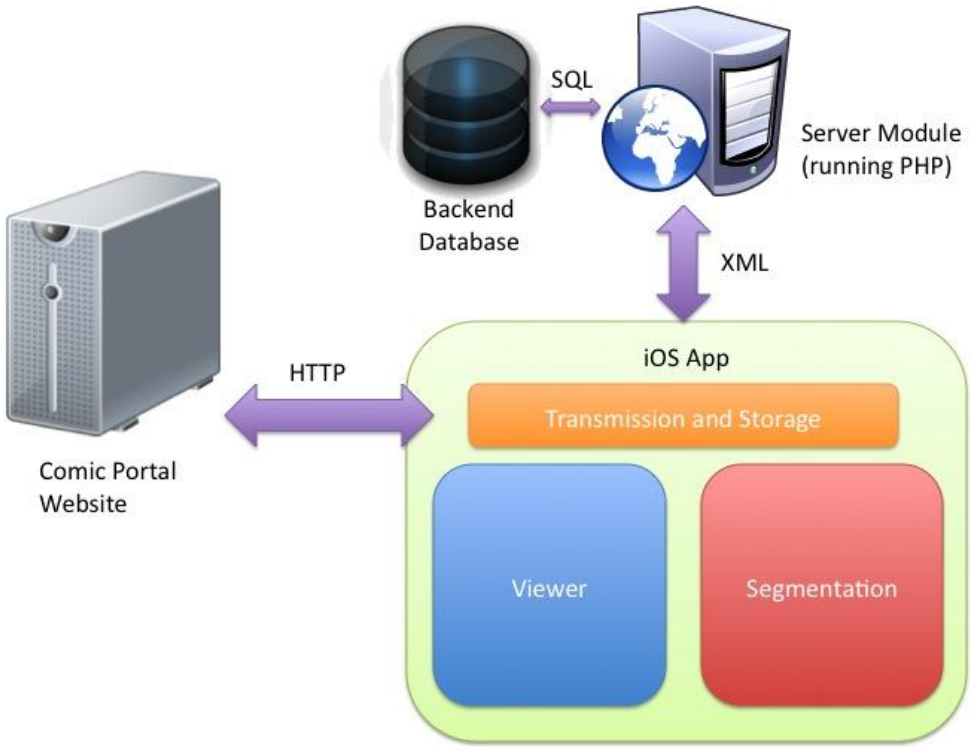


Figure 3.3 – System component

The architecture of the system consists of two major component, the server module and the Viewer app on iOS.

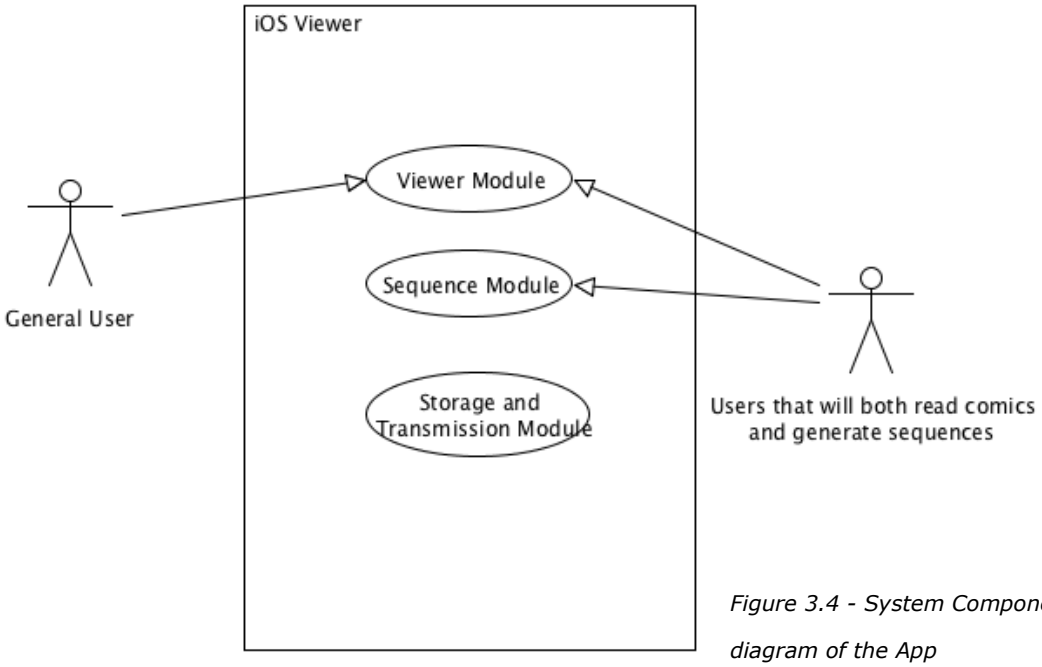


Figure 3.4 - System Component diagram of the App

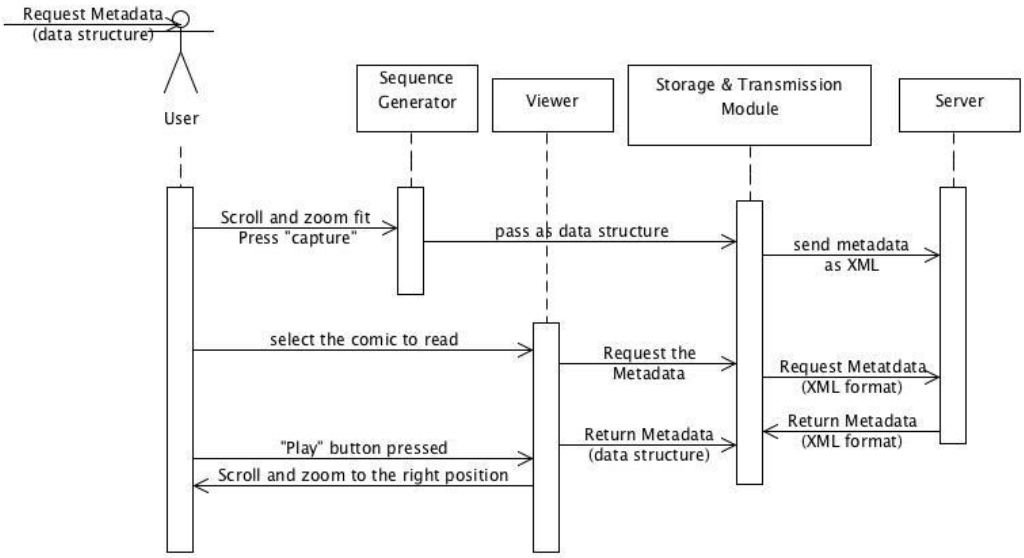


Figure 3.5 - Sequence diagram of the App

The Viewer App can be divided into three major parts, including Metadata Transmission and Storage Module, Viewing Module and Segmentation Creation Module.

3.3.1.1 Metadata Transmission and Storage Module

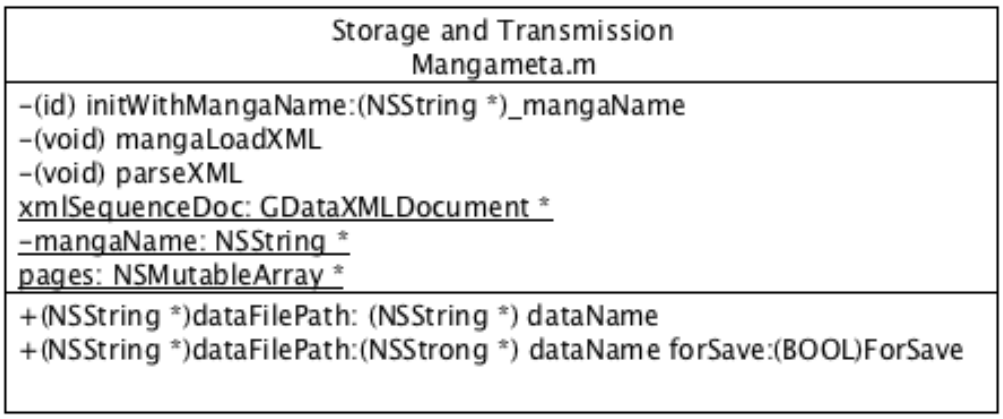


Figure 3.6 - First draft of the module

Metadata, including the panel and sequence information, is managed by this module. The backbone of this part is XML. There are serval advantages of XML over other approaches.

- XML fully supported Unicode
- XML can be transferred through HTTP
- XML is strictly defined but flexible so very suitable for medium of machine communication

The first one is very important for local comics. For the second one, both stateless communication method and persistent method have been considered. Since one single file is needed to be transfer for a single comic, so a stateless approach will be enough. HTTP is one of the most prevalent stateless transfer approach so a lot of work can be saved if building on it.

MetaData is stored in XML format. When the viewer modules tries to retrieve the data, it decode the XML and converted it into corresponding data structure. On the other hand,

if the segmentation module generates the panel and sequence information, it is converted by the module from data structure stored in ram to XML format.

Moreover, the module will also be responsible for obtaining the comi resources from the Internet.

The implementation of this part relied heavily on an external library by Google called GDataXML. It is written in Objective C and provides functions of XML read and write in DOM method. Moreover, it also supports XPath which may be useful in the future.

The transmission part relies heavily on the iOS Cocca Touch framework, the NSURL class and the instance method *writeToURL*.

3.3.1.2 Viewing Module

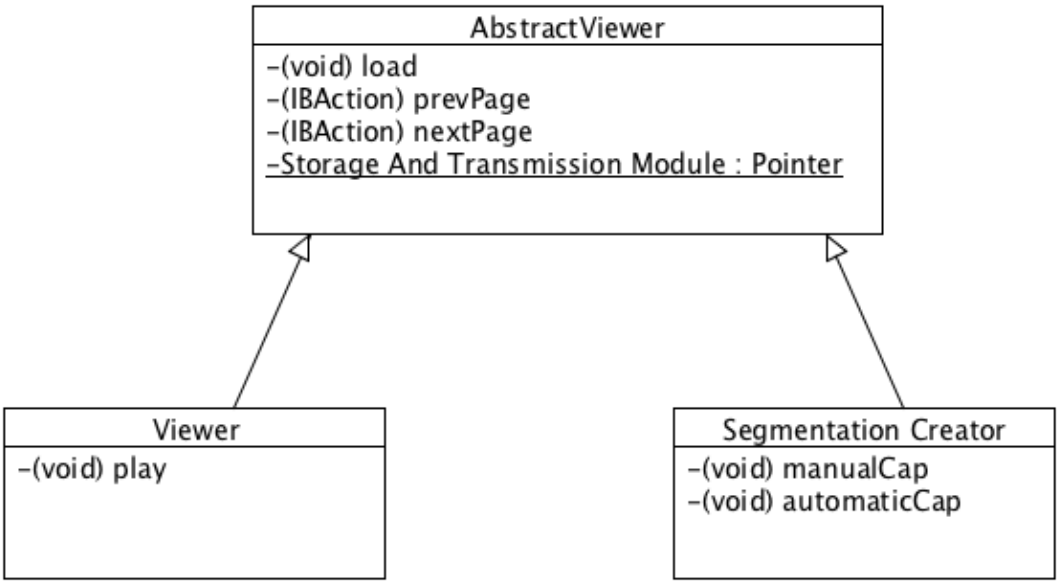


Figure 3.7 – Class diagram of the viewing module

Some of the navigation function is the same between Viewer module and Segmentation Creation Module, so a father class will be created first and the two module will inherited from this class.

The Viewing Module is responsible for the "playing" of the predefined sequence and zooming of the panel. It retrieves panel details and sequence data from the transmission and storage module. When the user pressed the play button on the tab bar, it will jump to the next panel.

Figure 3.8 - Capture of Viewer Module, the page navigation buttons are on the left-bottom while the play button is on right-bottom

Moreover, the viewer module also makes use of the multi-touch function on the iOS. The user can zoom in or out by pitching if they feel the default zoom rate is not suited for them.

3.3.1.3 Segmentation Creation Module

This part is responsible for generating metadata of the comics. Current it only supports the manual mode, semi-automatic or automatic mode may be supported in the future. The design of this module will use the abstract factory design pattern.

Under the manual mode, the user scroll to the right position and zoom fit. Then a capture button can be pressed to record the information of the current frame. The process is repeated until the data of the whole comics is recorded. The viewing sequence will also be generated at the same time according to the capture sequence.

Figure 3.9 - Capture of the Segmentation module, the "Capture" button is on the right-bottom

After that, the Segmentation module will pass the information to the Metadata Unit for local storage and sharing the information to the internet.

3.3.2 The Server

The main role of the server is to store and distributed shared metadata information. It includes two parts, the upload part and the request part. Users can generate metadata and upload it or download shared metadata from it.

The Server will use PHP to process request and use MySQL as the backend storage. This PHP+MySQL system is chose mainly because that

PHP and MySQL are popular and well developed tools. There are lots of documentations around the world so it is easier to get support. Also, there are mature libraries and frameworks on PHP, this shorten the development time and prevent reinventing the wheels. Moreover, lots of investigation on optimization has been done by other professionals. This saves a lot of time on tweaking.

Besides the backend MySQL, the Server Module can be further divided into two parts, upload.php and request.php

When the user opens a comic, the viewer app will automatically submit a request to request.php. request.php then check whether there are metadata. If yes, return it and return not found if no.

When the Viewer app generates a metadata file, it will upload the data through upload.php and the metadata will be saved into the backend database.

3.4 UI Design

This part is concerning user interface design.

3.4.1 Viewer

The UI of the Viewer App mainly consists of three parts, the home screen, Comic View Page and Manual Segment Page.

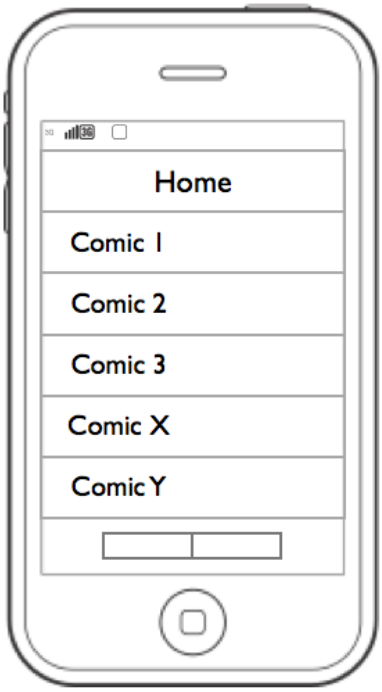


Figure 3.10 - The draft of Home Screen

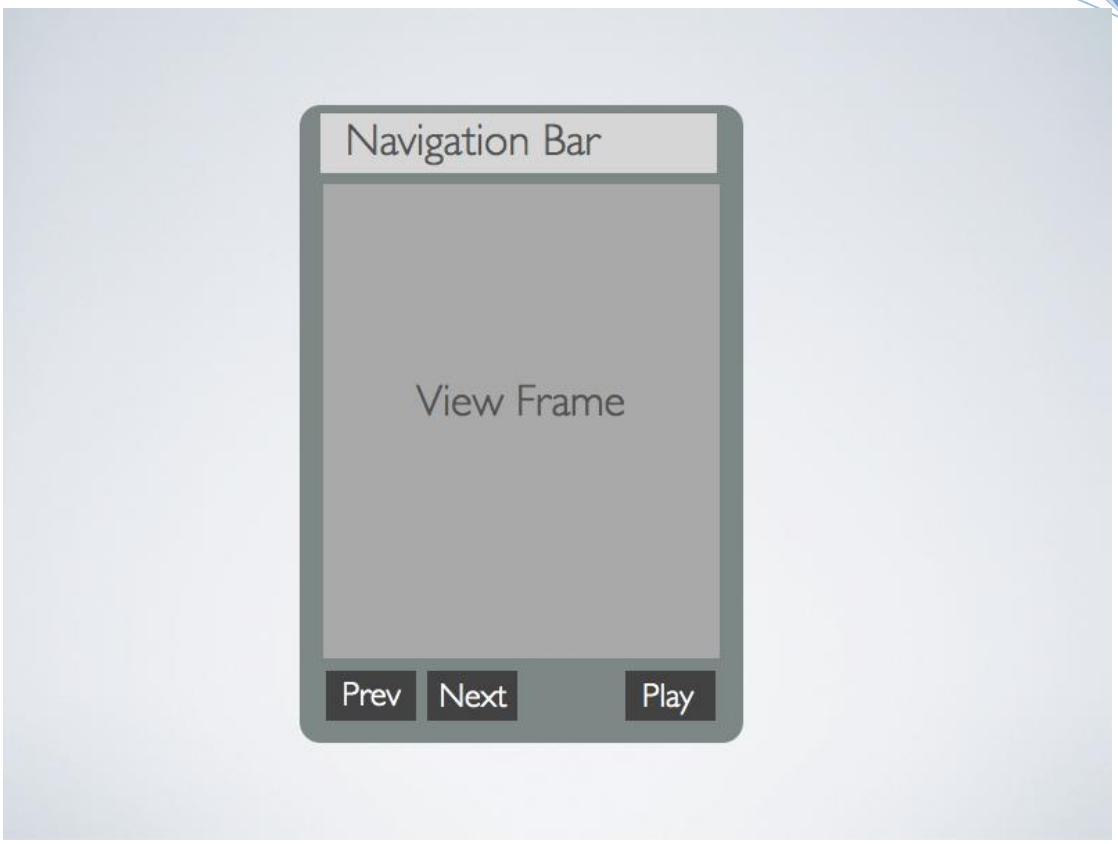


Figure 3.11 - Draft of the interface of the viewer

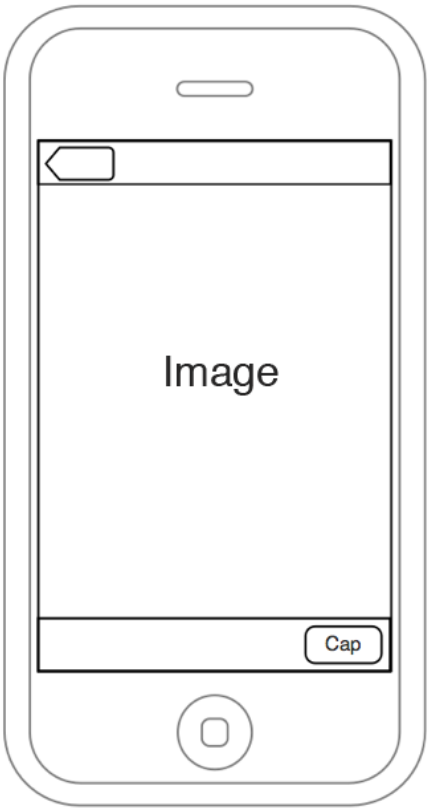


Figure 3.12 - Draft of the Manual Segmentation Page

3.4.2 Web Console

The web console is for demonstration purpose only, we would like to make it simple. It mainly contains two pages, namely mangaList and showXML, both are constructed by HTML tables.

The designs are shown as below:

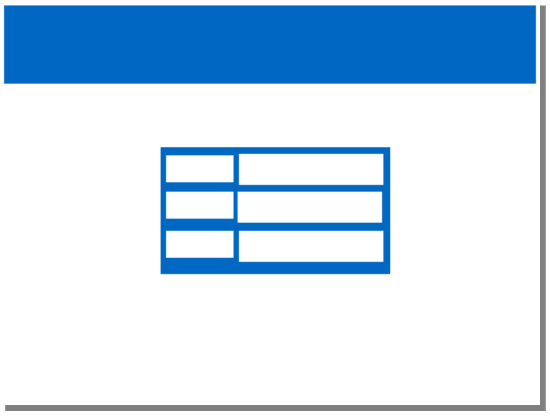


Figure 3.13 – Design of Web Console, mangaList Page

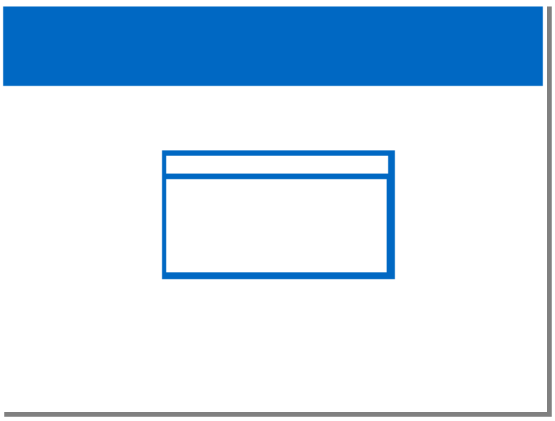


Figure 3.14 – Design of Web Console, showXML page

Chapter 4 Progress Review

We have tested some algorithms in PCs as well as implemented a comic viewer for iOS in Mac. Therefore, in this chapter, we will break down into two main focuses, talking about our project progress as the following list:

- Algorithm Study
  - Histogram
  - Contour Finding
  - Hough Lines
  - Watershed Segmentation
  - Flood-fill
- Performance Study
- Implementation
  - Panel Extractor
  - Viewer
  - Server
  - Porting library to iOS
  - Optimizations on iPhone
  - Web Console

4.1 Algorithm Study

The following algorithms are implemented in OpenCV, most of them are function calls. However, it requires a lot of pre-processing and post-processing on the target image, therefore we have written several separated programs to examine the algorithms respectively. It is helpful for us to:

- Get familiar with these functions provided by OpenCV,
- Examine the effect of these functions and,
- Determine whether a specific algorithm can provide useful information for panel extraction.

4.1.1 Histogram

For analysis purpose, we have calculated the Histogram of most target test cases. This helps us to cluster the test cases. For instance, we can cluster the test images once we know their color distribution.

In this program, we assume:

- Color range: 0-255, where 0 means totally black and 255 means totally white.
- Frequency of a certain color: 0-5000

4.1.2 Contour Finding

Contour provides useful information to panel extraction. No matter we use Watershed Segmentation or Flood-fill, contours should be pre-calculated.

This is the second program we wrote at the very beginning. To examine the image processing without Watershed Segmentation, we have written this program which can achieve the following functions:

- Image Binarization
- Contour Detection
- Bounding Box Calculation

We have tried and compared two methods for image binarization:

Method1: Directly use cvThreshold()

Method2:

- Get background color through 4 sample points and take average value
- Calculate distance between background color and the image
- Binarized the image by a threshold, same as the one cvThreshold() used

After image binarization, we perform contour detection and calculate the bounding box to the two binary results with same

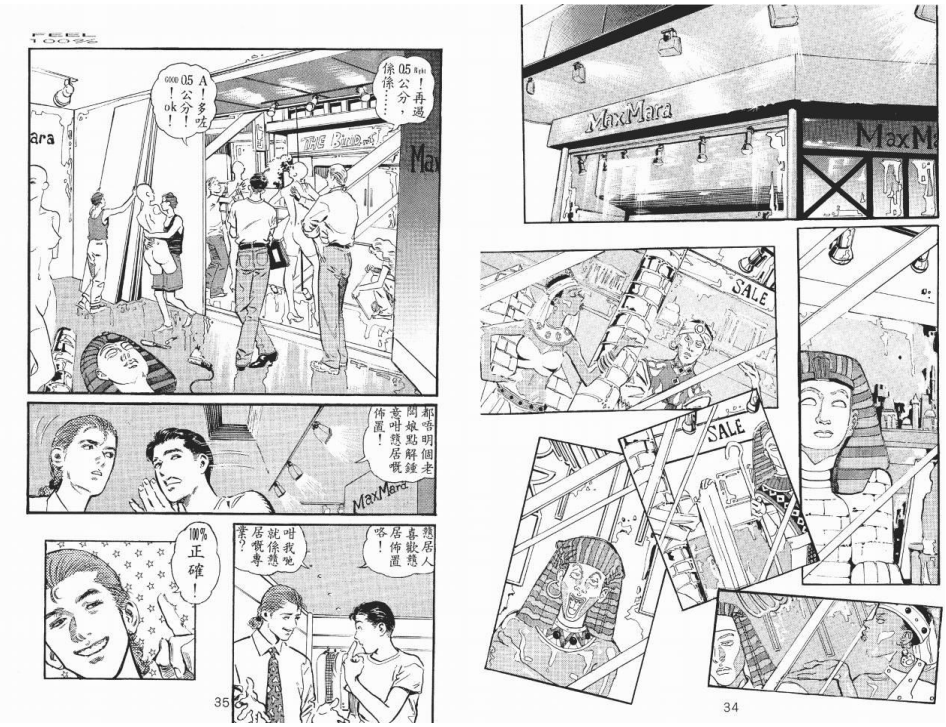
parameters. Before actual test cases, we have a pre-test case to evaluate the threshold we are going to use.

Case: Pre-test
Purpose: to evaluate the threshold for further testing

Figure 4.1 - Input image for Pre-test case

Case: #1
Purpose: to find the contours as well as bounding box for a simplest Four Column Comic
Remark: Same as Pre-test case input

Figure 4.2 - Input Image for contour finding, Test case 1



Case: #2

Purpose: to try finding the contours as well as bounding box for image with overlapped panels

Figure 4.3 - Input image for contour finding, Test case 2

Figure 4.4 - Input image for contour finding, Test case 3

This test image is chosen because:

- There exist some speech bubbles crossing two panels in general.
- Strong strokes of this image might affect the result.

Case: #4

Test case 4 makes use of the original image of case 3, with threshold on 255 for demonstration purpose only.

4.1.3 Hough Lines Finding

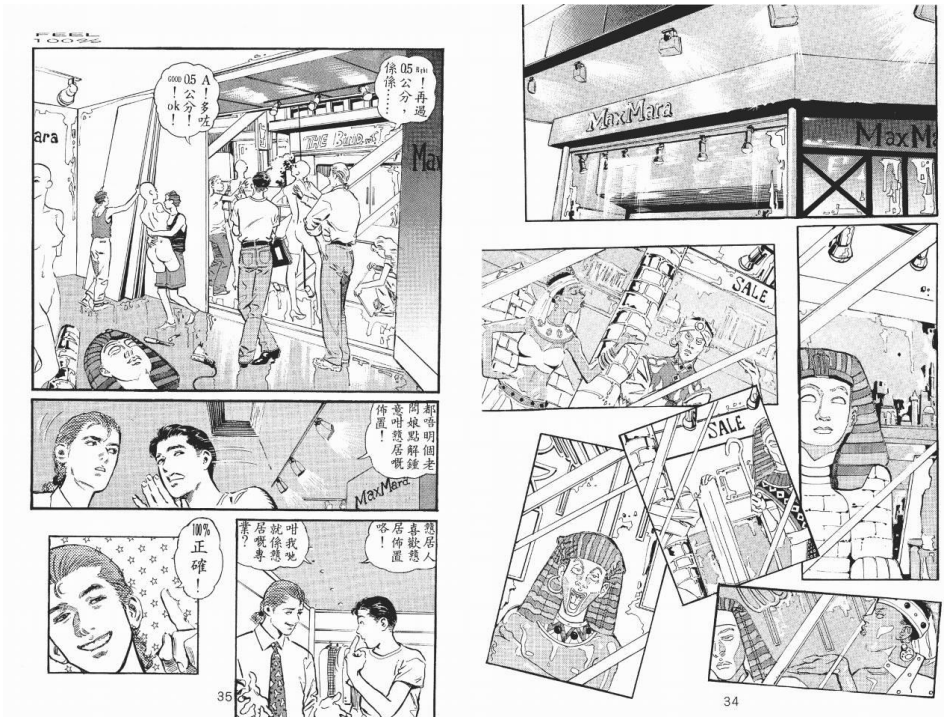
We have also examined on Hough lines finding. Hough lines are different with contours as contours contain arbitrary lines but what we can get from Hough line detection are straight lines only.

Case: #1
Purpose: to find the Hough lines in a simple Four-Column Comic page

Figure 4.5 - Input image for Hough Line finding, Test case 1

Case: #2
Purpose: to find the Hough lines in a traditional comic page with open panels

Figure 4.6 - Input image for Hough line finding, Test case 2



Case: #3
Purpose: to find the Hough lines in a traditional comic page with overlapped panels

Figure 4.7 - Input for Hough line finding, Test case 3

Case: #4

Purpose: to find the Hough lines in a comic page with strong strokes

Figure 4.8 - Input for Hough line finding, Test case 4

4.1.4 Watershed Segmentation

We have 4 test cases for Watershed Segmentation which requires manually input the markers for panels.

Case: #1
Purpose: to perform Watershed Segmentation in a simple colorful Four-Column Comic page

Figure 4.9 - Input image for Watershed Segmentation, Test case 1

Case: #2
Purpose: to perform Watershed Segmentation in a simple Four-Column Comic page in grey scale

Figure 4.10 - Input image for Watershed Segmentation, Test case 2

Case: #3
Purpose: to perform Watershed Segmentation in a comic page with overlapped panels

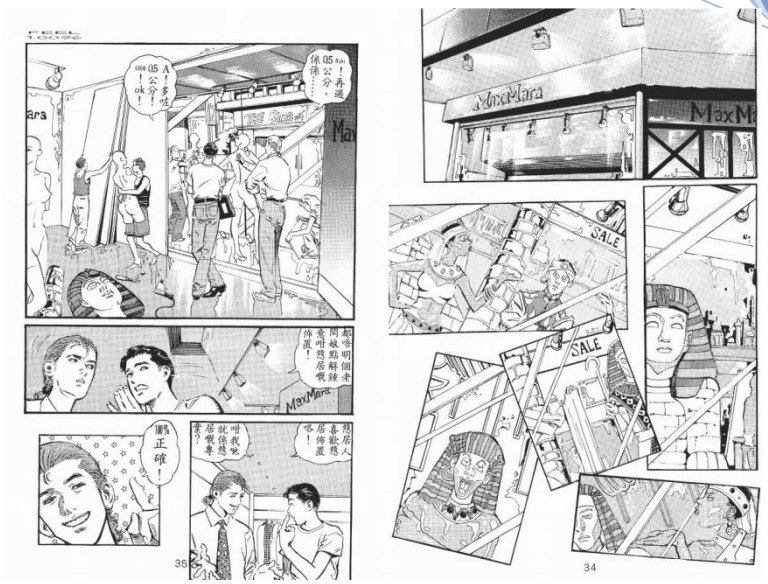


Figure 4.11 - Input image for Watershed Segmentation, Test case 3

Case: #4
Purpose: to perform Watershed Segmentation in a comic page with open panels

Figure 4.12 - Input image for Watershed Segmentation, Test case 4

Case: #5
Purpose: to perform Watershed Segmentation in a comic page with strong strokes

Figure 4.13 - Input image for Watershed Segmentation, Test case 5

4.1.5 Flood Fill

In this part, our concern is how to enhance the accuracy of Watershed Segmentation. Flood-fill is thus helpful to us as it can enhance the contrast between the background and the panels. Since this part does not involve too much, we will only demonstrate some outputs below. The background are manually selected and filled as red.

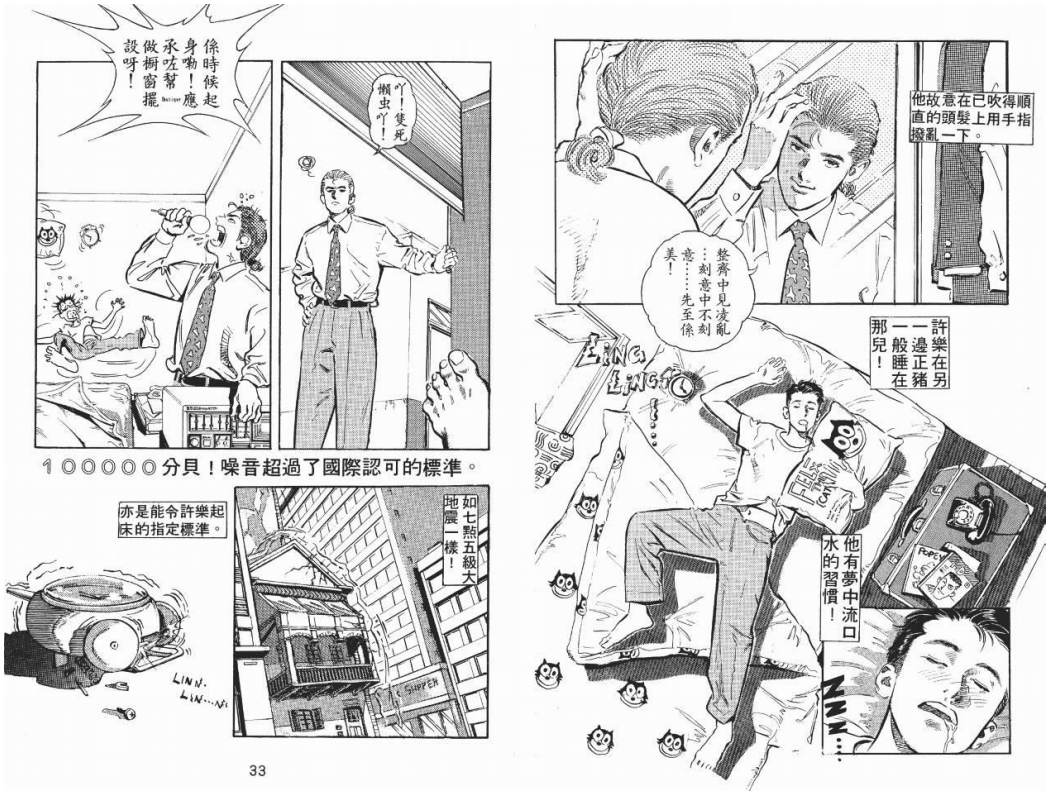


Figure 4.14 - Input image for Flood-fill, Test case 1

Case: #1
Purpose: to perform Flood-fill in a comic page with open panels
Remarks: Simple Four-Column Comic page is omitted as the result is trivial once Case #1 is done.

Case: #2
Purpose: to perform Flood-fill in a comic page with overlapped panels

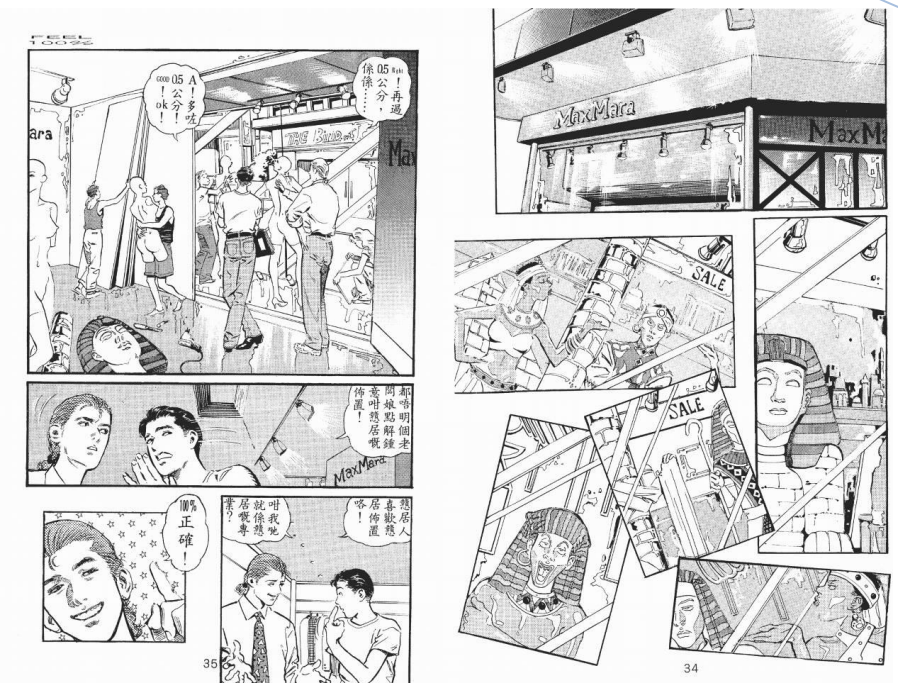


Figure 4.15 - Input image for Flood-fill, Test case 2

Case: #3
Purpose: to perform Flood-fill in a comic page with strong strokes

Figure 4.16 - Input image for Flood-fill, Test case 3

4.2 Performance Study

In order to study the performance of the algorithm, some experiments have be done. The CPU execution time has been measured. They have been done on both PC and iPhone. The speed and variation between image size and speed is being studied. The details will be discussed later.

4.3 Implementation

4.3.1 Panel Extractor

4.3.1.1 Manual Extractor on PC

We have studied some important algorithms and functions for panel extraction. In term 1, we have also implemented a manual panel extractor in PC which output an XML file describing the structure of the panels on the target image.

Input: Drawing bounding box manually by mouse
Output: Initial x and y coordinate as well as height and width of the bounding boxes, saved as XML.

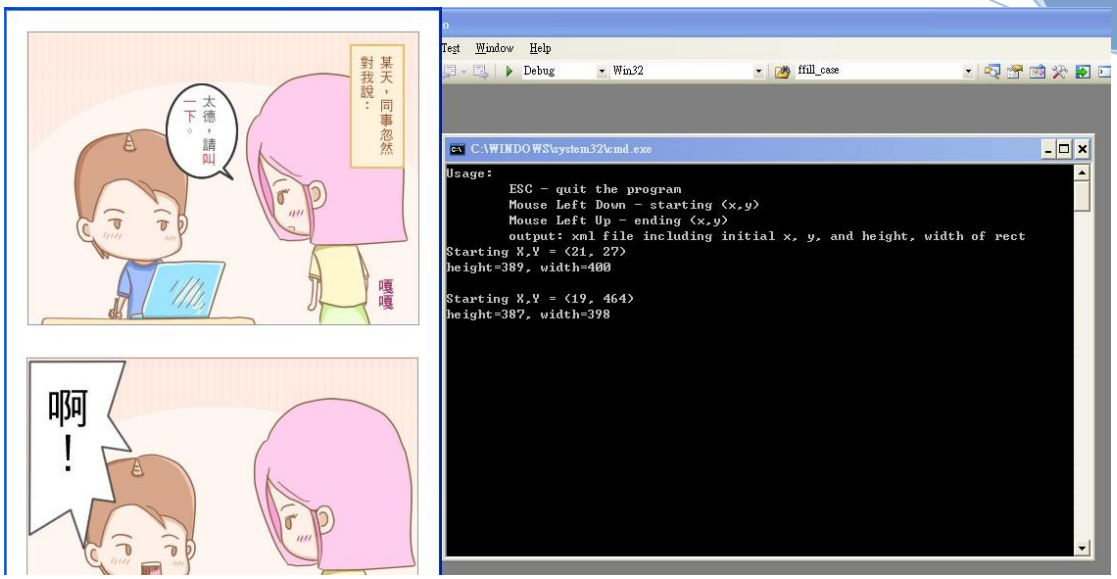


Figure 4.17 - Screen dump of the manual panel extractor

The output XML has a simple structure shown as follows,

where the depth of the branch represents the level of the node.

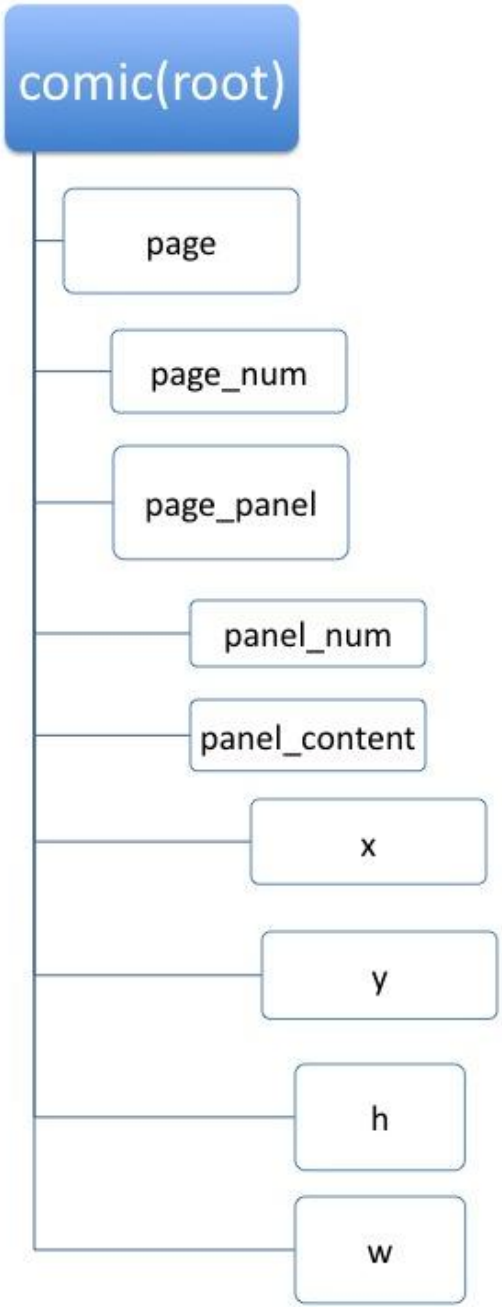


Figure 4.18 - Structure of the output XML

In this term, we have implemented another manual panel extractor as well as a semi-automatic one, in iPhone.

4.3.1.2 Manual Panel Extractor in iPhone

Figure 4.19 – Auto and Manual Taps of the App

By selecting the buttons in the below toolbar of the Home Screen, the App will enter the Manual Extractor Mode.

Figure 4.20 – Manual Extractor Mode

Users can zoom in and out the app. By tapping the "Capture" button, the current frame will be captured. By tapping the "Func" button, an Action Sheet for triggering save and clear functions will pop up.

Figure 4.21 – Save and Clear Button

User can save or clear the manual extracted panels by tapping the correspond buttons on the sheet.

4.3.1.3 Semi-Automatic Panel Extractor in iPhone

One of the selling features of the App is the semi-automatic panel extractor. The function can be called from the button in Action Sheet or set to auto process in the setting page.

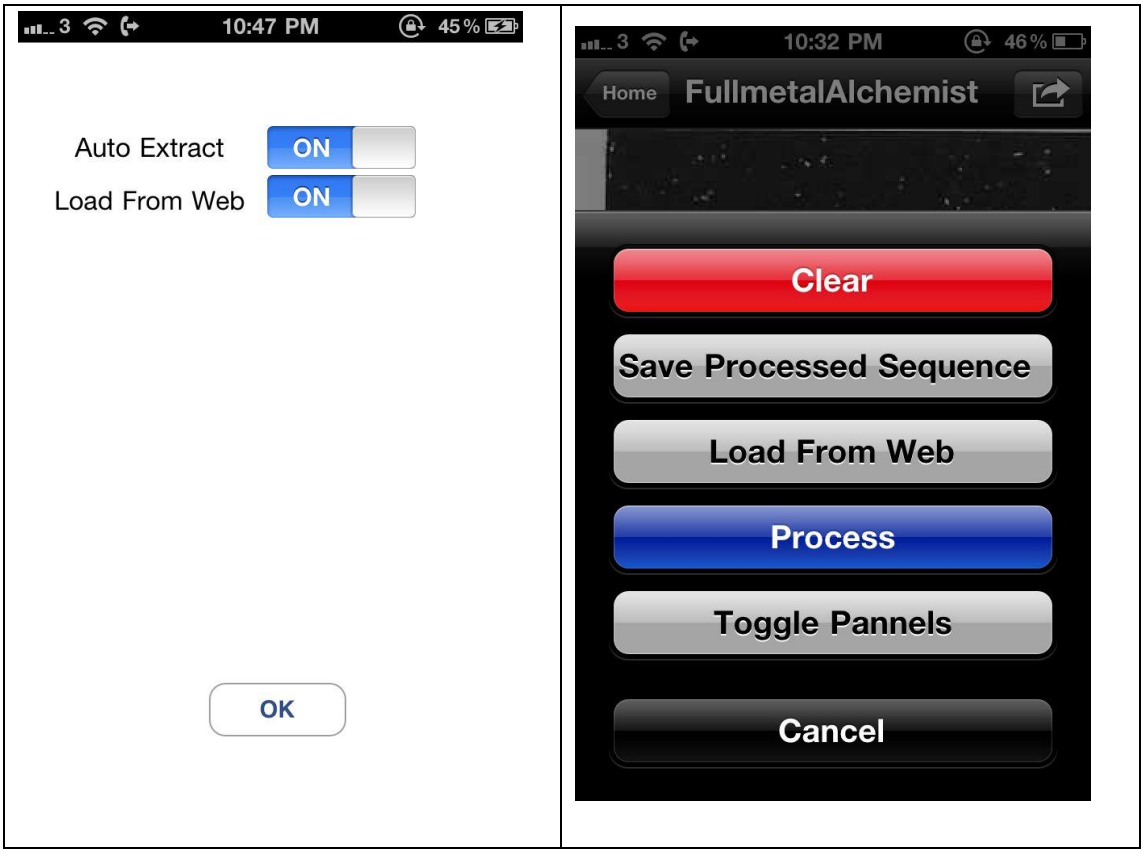


Figure 4.22 – Function Buttons of Semi-auto Panel Extractor

When the function is triggered, it will process and show the progress of the process.

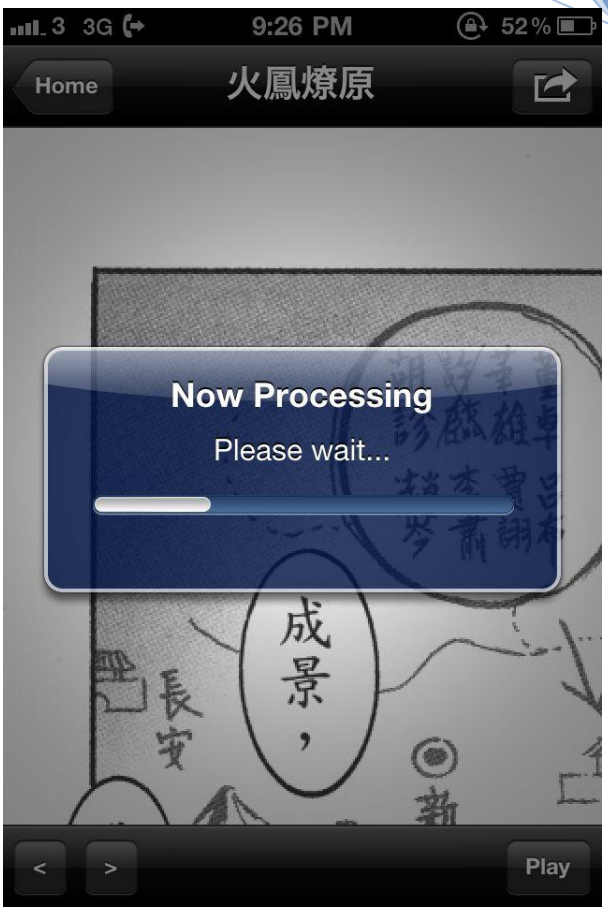


Figure 2.23 - Panel Extractor processing

4.3.1.3.1 Porting Library to iOS

In last term, we have tried out several algorithms that we believed is useful for comic panel extraction. In our first experimental program for extraction by contour finding, connected component is introduced. In fact, it involves several items that we mentioned.

It first starts with an Image Binarization process, then with Flood-filling we can get the contours, stored as a sequence of connected component, then we can find the bounding box.

Besides standard OpenCV, the panel extracting function on the viewer app also relies on an OpenCV extension library called cvBlobsLib. cvBlobsLib is an external OpenCV library that

integrates all the things we have done and it can perform binary images connected component labeling (similar to regionprops Matlab function). It also provides functions to manipulate, filter and extract results from the extracted blobs.

Though it is not easy, we have finally ported cvBlobsLib into iOS.

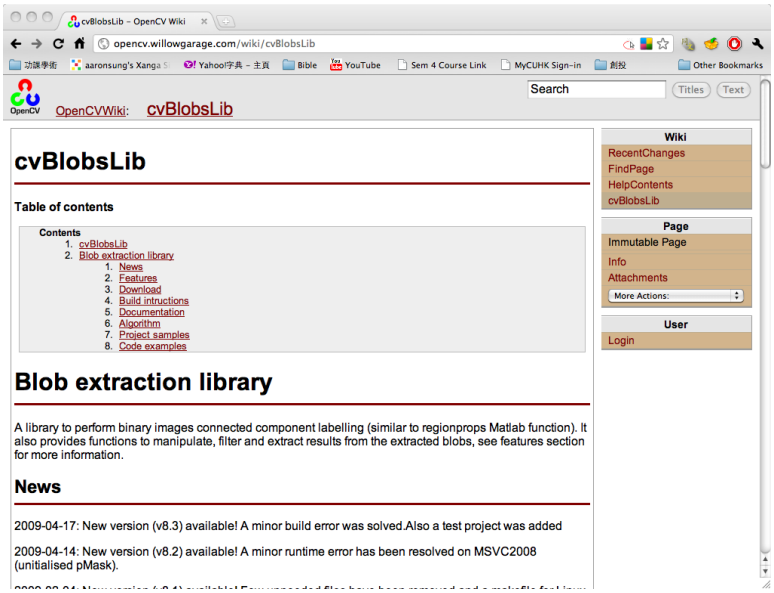


Figure 4.24 - Official Page of cvBlobsLib

This library provides two important features. The first one is the extraction of 8-connected components in binary or grayscale images. These connected components are referred as blobs. Second filter is provided to obtained blobs to get the interest objects in the image. Last be not least, the library is thread safe if different thread use different objects. The use of thread is important for the App in order to provide a faster response.

However, this library is only available on Windows and Linux. Moreover, the Linux version is far older behind the Windows versionAs the library is essential for our App, so the library must

be ported to iOS.

A step-by-step approach has been used to port the library, First it is being ported to Mac OS X. Then it is being ported to iOS.

The First Step is quite straight forward. Despite the library is not written for Unix system, the library can run successfully on Mac OS X after removing Windows debugging macro and some non necessary win32 API call.

The second step is porting to iOS. This step is much harder. cvBlobsLib is written by C++, while the 'official' language of iOS development is Objective C, which mean that Objective C must be used to call APIs and communicate with iOS and the devices.

The solution is that Apple provide a language called Objective C++. However, this is a mixture of C++ and Objective C rather that combination of them. Both C++ and Objective C can be used in Objective C++, however, the use of C++ and Objective C cannot be mixed. For example, you cannot inherit a C++ class from an Objective C class or vice versa. The main use of Objective-C++ is to build a wrapper of the C++ code.

As to increase the efficiency of performance, the main part of the semi-automatic extraction is also written in C++. So the self-written function is wrapped through Objective-C++ instead of the whole library.

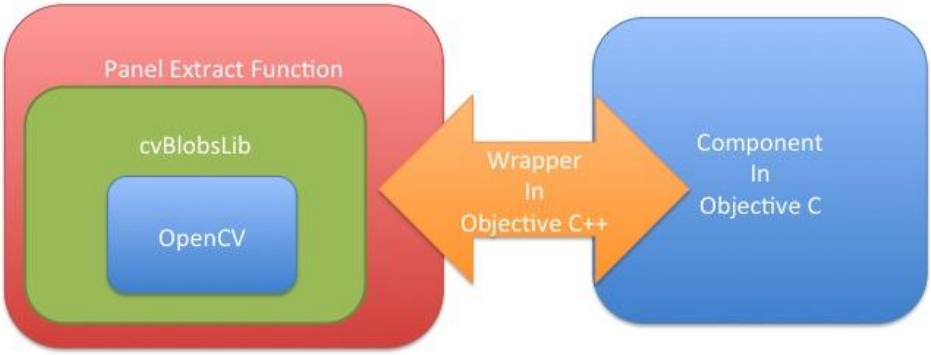


Figure 4.25 - Structure of the Porting

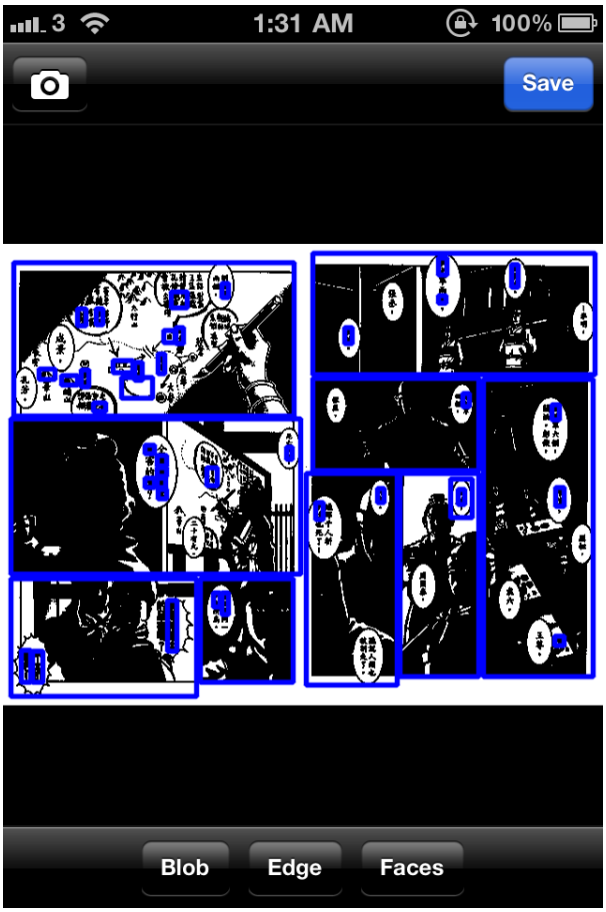


Figure 4.26 - The testing Program for the porting to iOS

4.3.1.3.2 Optimizations on iPhone

After the panel extract function is successfully run on iOS, optimization is being carried out to improve the performance on iOS devices.

Here is the typical configuration of iOS device. Obviously the configuration is much cheaper than the nowadays PCs.

Model	IPhone 3Gs	IPhone4	IPod 4th gen
CPU	833 MHz (underclocked to 600 MHz) ARM Cortex-A8[12]	1 GHz (underclocked to 800 MHz) ARM Cortex-A8 Apple A4[13]	1 GHz (underclocked to 800 MHz) ARM Cortex-A8 Apple A4
Memory	256 MB DRAM	512 MB DRAM[14]	256 MB DRAM

Table 4.1 – Typical configuration of iOS device

As the memory in iOS device is so limited, memory warnings are commonly see when running graphical library such as OpenCV and cvBlobsLib.

```
2011-04-18 03:41:30.672 Manga[821:6403] BlobExtract dealloc
2011-04-18 03:41:30.677 Manga[821:6403] before release
2011-04-18 03:41:30.681 Manga[821:6403] progress: 0.500000
2011-04-18 03:41:30.685 Manga[821:307] Progress 0.500000
2011-04-18 03:41:30.685 Manga[821:6403] panel Extract
2011-04-18 03:41:30.927 Manga[821:6403] initializing
2011-04-18 03:41:30.978 Manga[821:6403] processing...
2011-04-18 03:41:30.982 Manga[821:6403] Size is 952 1260
2011-04-18 03:41:34.515 Manga[821:307] Received memory warning. Level=1
2011-04-18 03:41:37.493 Manga[821:307] Received memory warning. Level=2
2011-04-18 03:41:39.360 Manga[821:307] Received memory warning. Level=1
2011-04-18 03:41:40.957 Manga[821:6403] number of Blobs 6
2011-04-18 03:41:40.963 Manga[821:6403] Blob Extracted
2011-04-18 03:41:40.980 Manga[821:6403] (
```

Figure 4.27 – Low Memory Warning come out sometimes

There are different levels of memory warning defined as

[12] Shimpi A. The iPhone 3GS Hardware Exposed & Analyzed. [Internet]. Available from: http://www.anandtech.com/gadgets/showdoc.aspx?i=3579&p=2.
[13] iPhone 4 Teardown – Page 2. [Internet]. [cited 2010 Jun 23]. Available from: http://www.ifixit.com/Teardown/iPhone-4-Teardown/3130/2.

[14] iPhone 4 Confirmed to Have 512MB of RAM (Twice the iPad and 3GS). [Internet]. 2011 Available from: http://www.macrumors.com/2010/06/17/iphone-4-confirmed-to-have-512mb-of-ram-twice-the-ipad-and-3gs/.

follow:

```
typedef enum {    OSMemoryNotificationLevelAny      = -1,
OSMemoryNotificationLevelNormal   =  0,
OSMemoryNotificationLevelWarning  =  1,
OSMemoryNotificationLevelUrgent   =  2,
OSMemoryNotificationLevelCritical =  3 }
OSMemoryNotificationLevel;
```

The corresponding meanings are here:

1. Warning (not-normal) — Relaunch, or delay auto relaunch of nonessential background apps e.g. Mail.
2. Urgent — Quit all background apps, e.g. Safari and iPod.
3. Critical and beyond — The kernel will take over, probably killing SpringBoard or even reboot.

As stated above, lack of memory can force the App to be quit, so allocated memory that no longer need to be use should be free as soon as possible and make sure to be completely free.

One important trick is the deallocation of NSAutorelease pool. An autorelease pool is an instance of NSAutoreleasePool that "contains" other objects that have received an autorelease message; when the autorelease pool is deallocated it sends a release message to each of those objects. An object can be put into an autorelease pool several times, and receives a release message for each time it was put into the pool. Thus, sending autorelease instead of release to an object extends the lifetime of that object at least until the pool itself is released (the object may survive longer if it is retained in the interim).

Autorelease pools are arranged in a stack, although they are commonly referred to as being "nested." When you create a new

autorelease pool, it is added to the top of the stack. When pools are deallocated, they are removed from the stack. When an object is sent an autorelease message, it is added to the current topmost pool for the current thread.

The Application Kit automatically creates a pool at the beginning of an event cycle (or event-loop iteration), such as a mouse down event, and drains it at the end.[15]

However, there are two situations that the programmer ought to manually handle the Auto Release Pool, one is in detach thread, and the other is during loop with heavy computation, which is the case going to be talked.

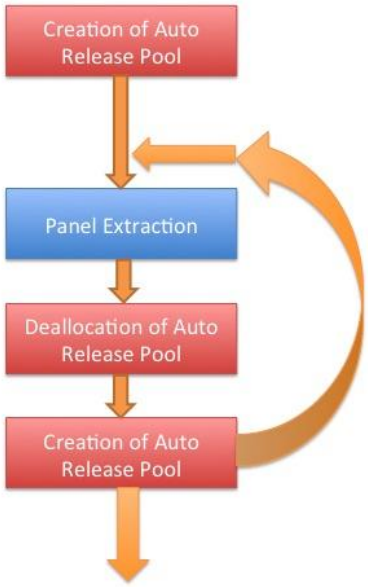


Figure 4.28 – Flow for allocation and deallocation of memory

In order to ensure enough memory is available, manually creation and deallocation must be done within the loop.

[15] Apple, Inc. Memory Management Programming Guide. [Internet]. [cited 2011 Apr 10]. Available from: http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/MemoryMgmt/Articles/mmAutoreleasePools.html#//apple_ref/doc/uid/20000047-1041876.

4.3.2 Viewer Implementation

The Viewer App is developed parallel with the algorithm study. The Spiral model is being adopted to provide a balance between fast prototyping and the meeting of the required functions.

As the OpenCV is act as the "heart" of the project, a suitable port on iOS must be ensured.

The porting from Yoshimasa Niwa[16] has been tested and chose. There are several advantages.

- It is a mature porting. Most of the functions of the OpenCV are working on this porting.
- There are precompiled version for both iOS Simulator and iOS device.
- It took use of the Accelerate Framework which is newly ported to iOS 4 by Apple Inc. This improves the performance.

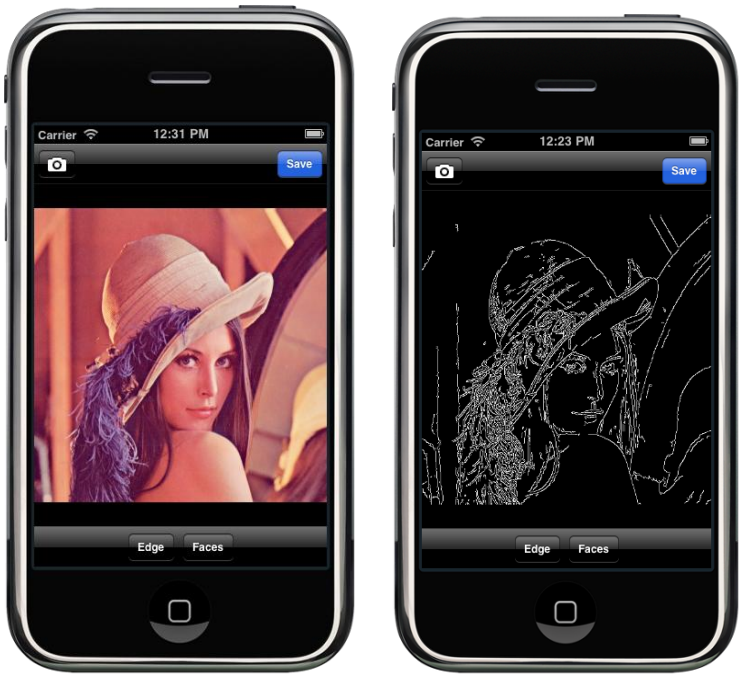


Figure 4.29 – OpenCV program on iPhone

[16] http://niw.at/articles/2009/03/14/using-opencv-on-iphone/en

Before starting the development of iOS, one important thing is to get a valid provision file for testing on iOS device.

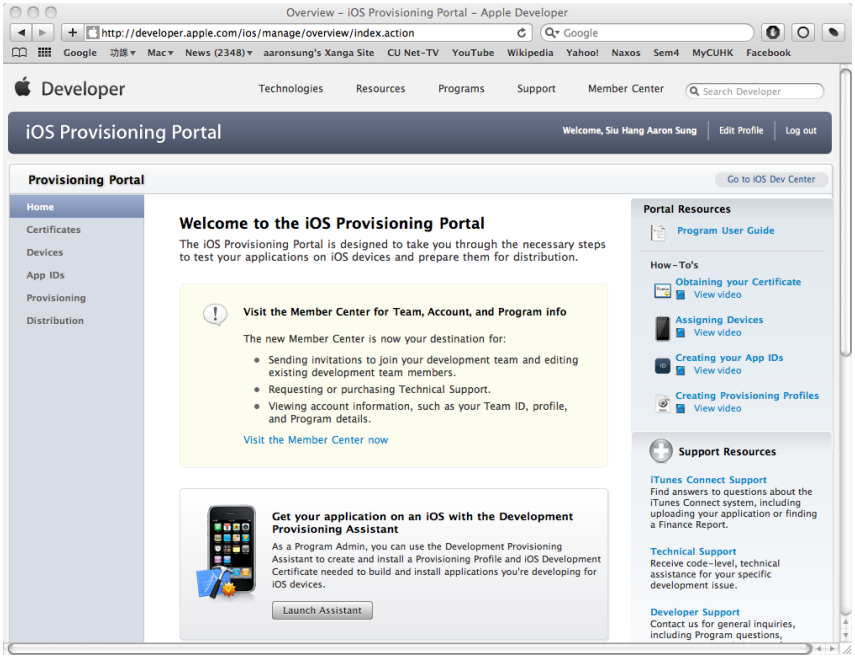


Figure 4.30 - The main page of iOS Provisioning Portal

The first thing is to generate a public-private key pair for verification of the developer identity.

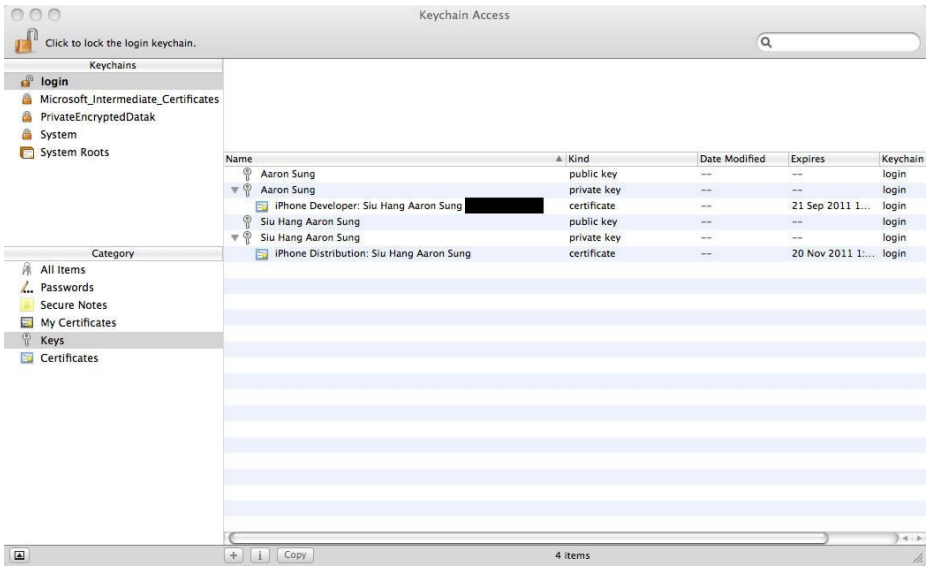


Figure 4.31 - The public-private key pair

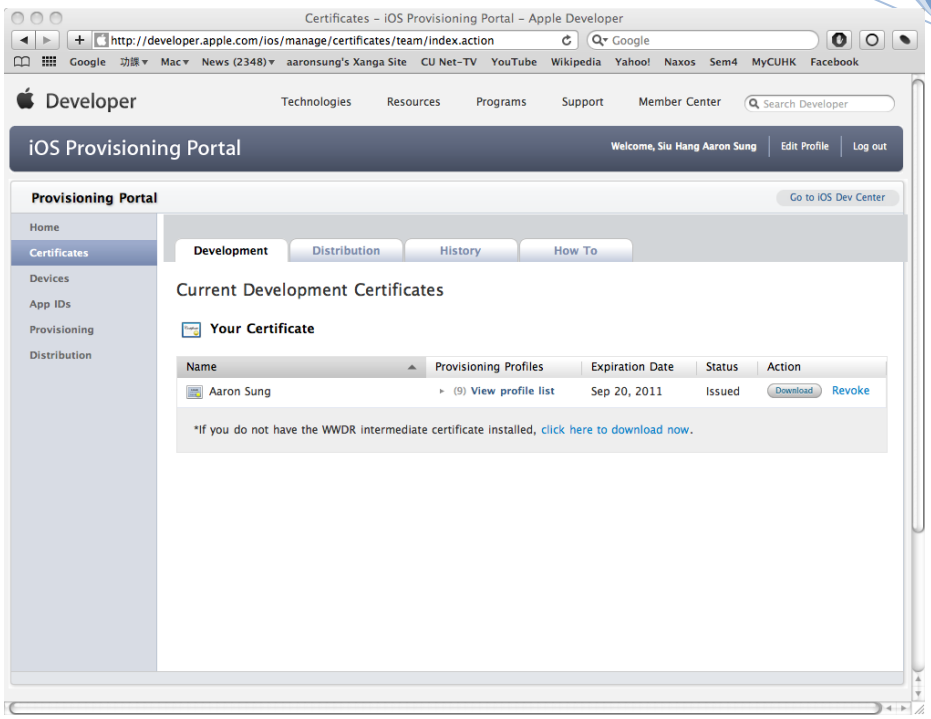


Figure 4.32 - Submition of the key pair to iOS Provisioning Portal

Then is obtained the UUID of the testing iOS device from the organizer of XCode and register it in the portal.

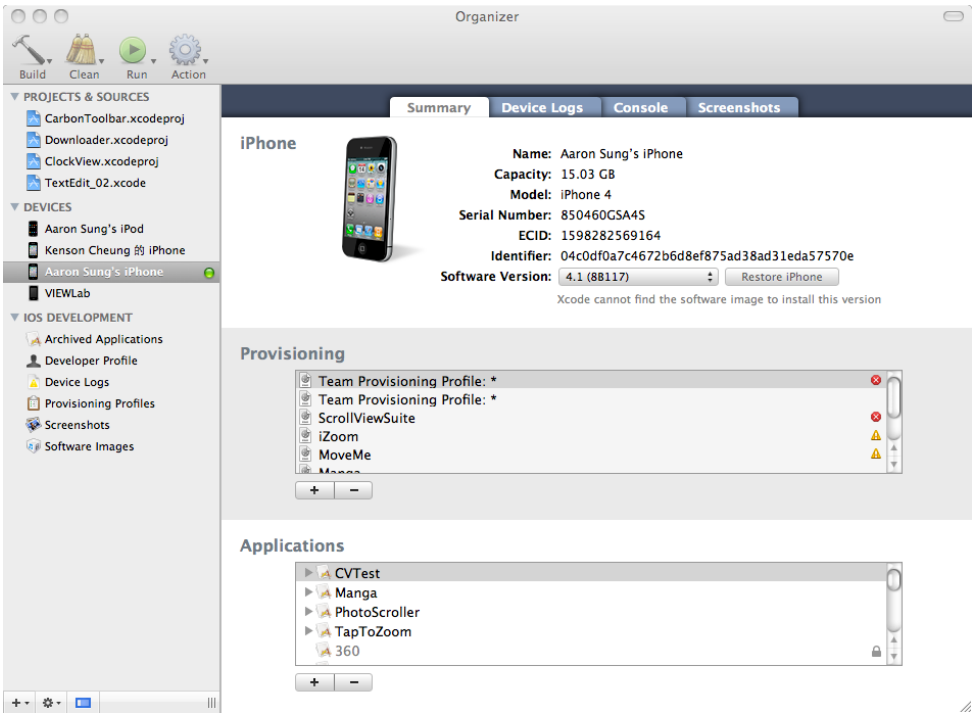


Figure 4.33 - The Organizer of XCode

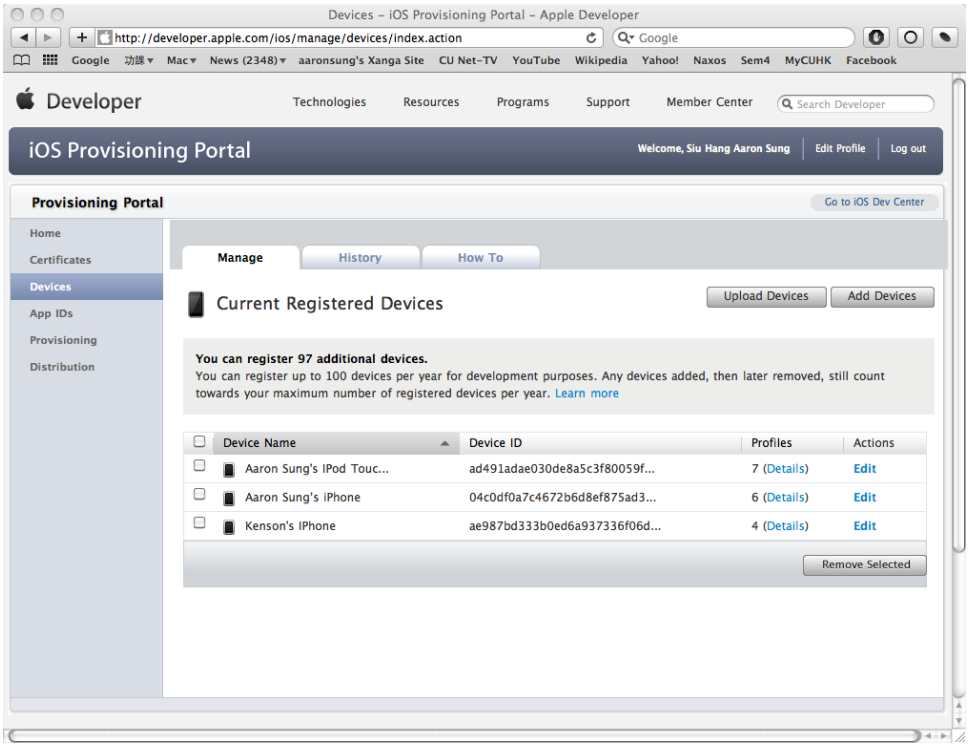


Figure 4.34 - The page of device registration

Then is to create an App ID by giving the bundle identifier and name of the App.

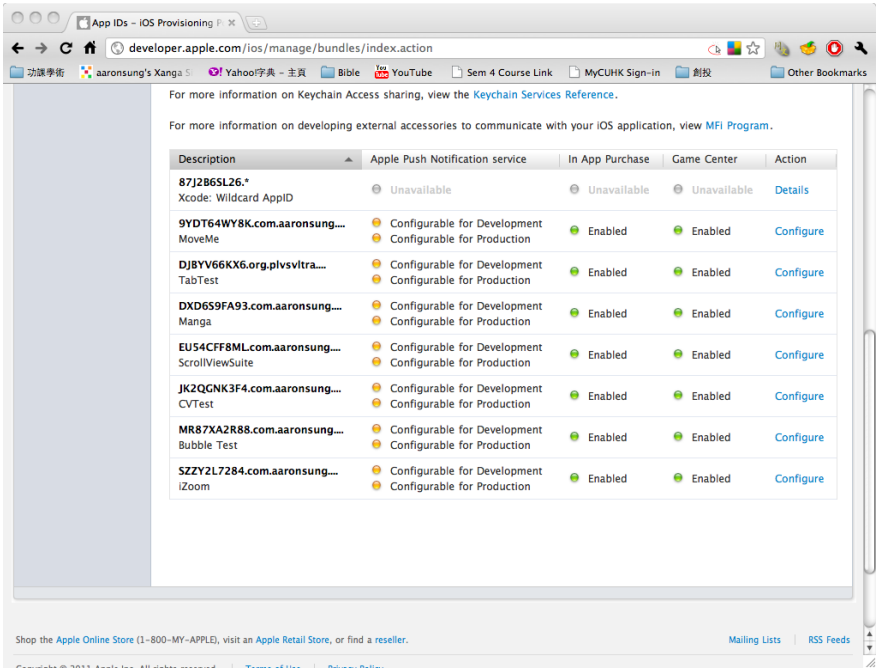


Figure 4.35 - The page for creating an App ID

Finally is to create a provision profile that match the app and the testing device.

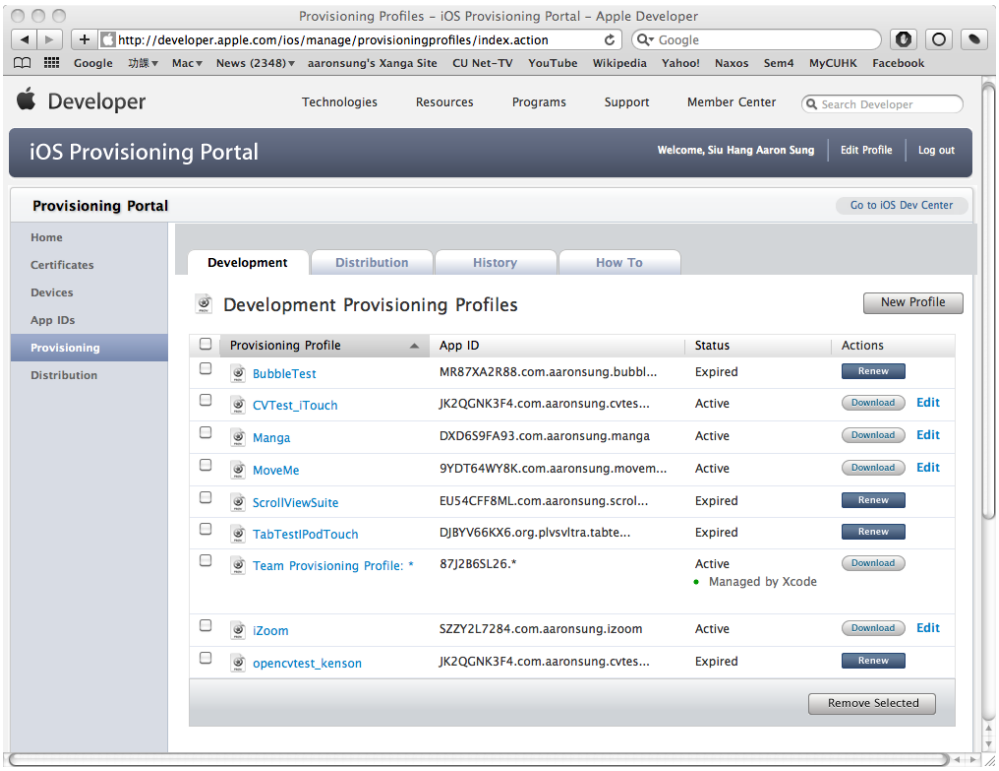


Figure 4.36 - Provision Profiles Page

Then download and install it through XCode Organizer. The installed profile can be see on iPhone in Setting>General>Profiles

Figure 4.37 – Provisioning Profiles

Finally add the code sign in the Project page in XCode.

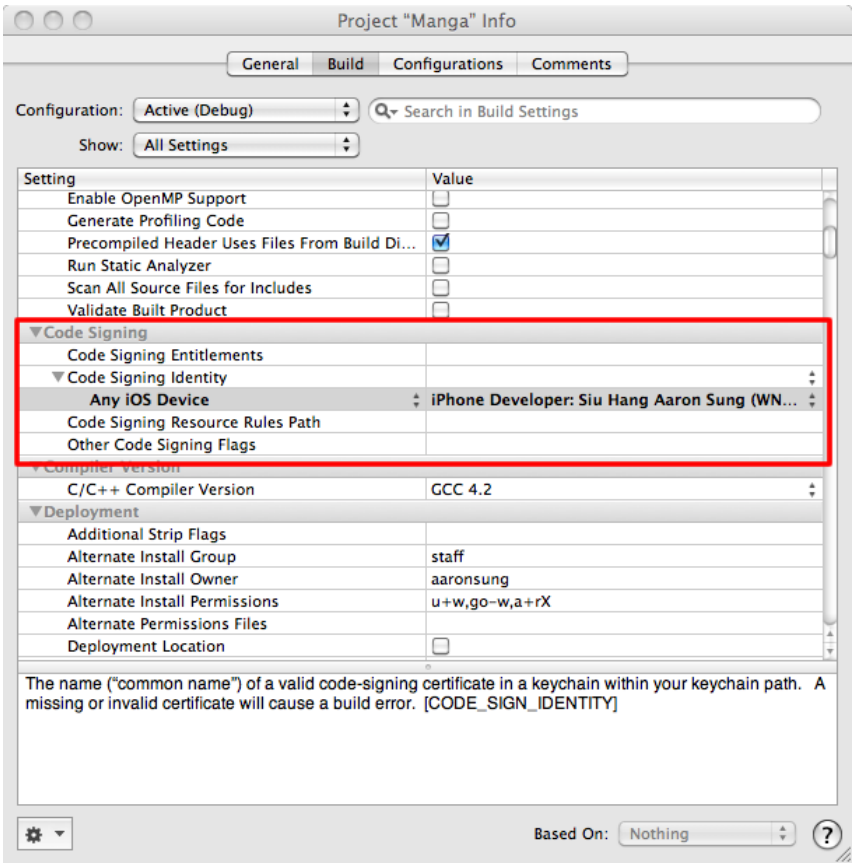


Figure 4.38 – Project Page in XCode

After setting of the provision profile, then the interface should be designed. Since the function of the viewer is to display the comic to the user, and the program structure of an iOS app is heavily related to how you organized the display of information, so the interface is being considered and draft first.

On top of this, a page is needed for the users to choose which comic to read. As a result, the navigation view mode with table is being chosen.

Table view contains a list of field that you can choose. The navigation view contains a button that can go back to the last page and can provide a “feel” of hierarchy.

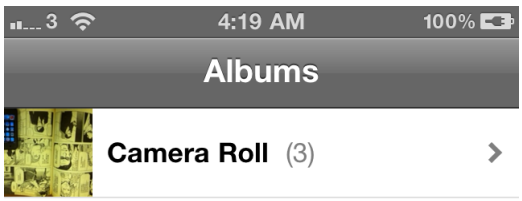


Figure 4.39 - Table view of the viewer

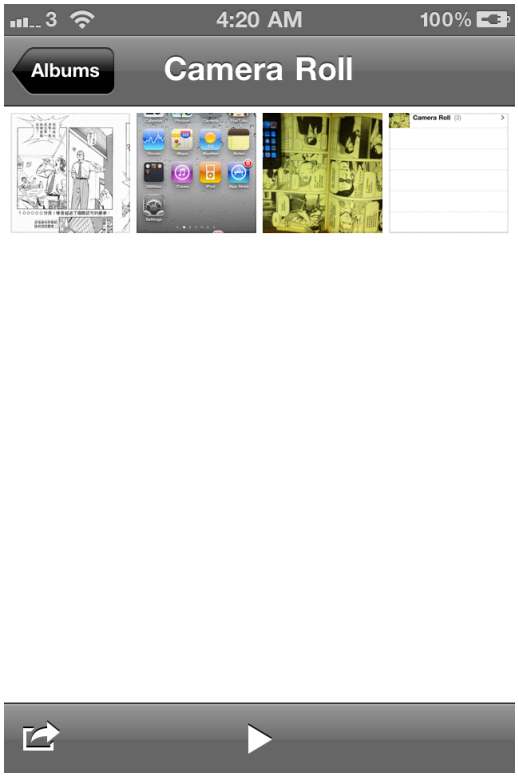


Figure 4.40 - Navigation View.

Note that there is a navigation bar on top

The first version of the viewer provides the following function:

- User can choose the comic they want and the comic will shown respectively
- The previous page and next page button can be passed to navigating through different pages.

The first version is simple but gives the backbone of the program.

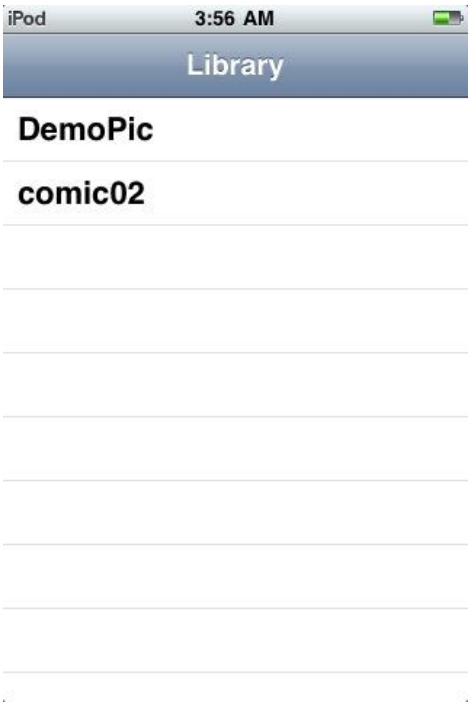


Figure 4.41 - The main view of the app, users can choose the comic they want

Figure 4.42 - The viewer

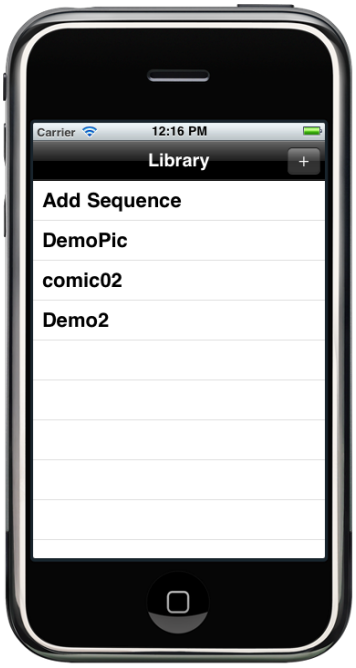


Figure 4.43 - Debugging on the iOS Simulator on Mac OS X

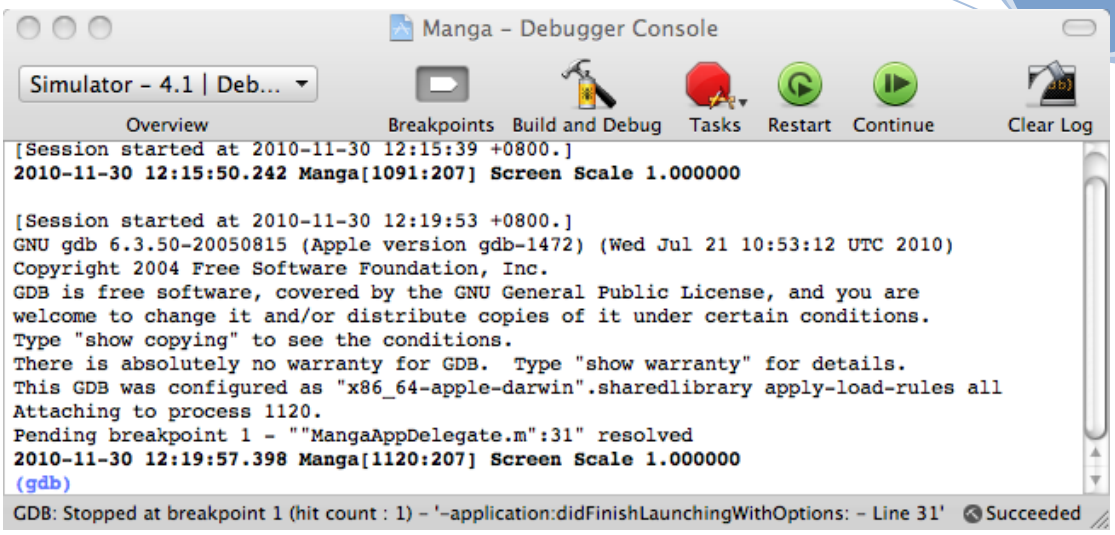


Figure 4.44 - Debugging by GDB

After the basic part is implemented, more advanced functions are added to it. One thing is the scrolling and zooming in the viewer. Besides it, the Segmentation module and Storage and Transmission module is also implemented.

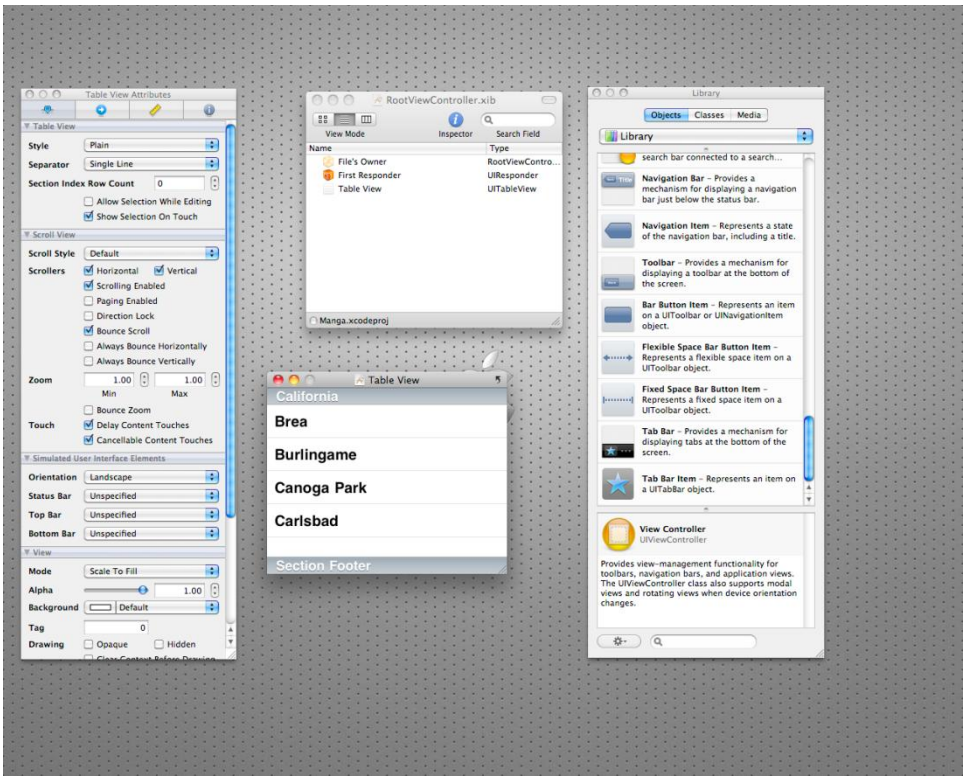


Figure 4.45 - Screen capture of the development of the interface.

In semester 2, the interface continues to improve, and it is

the final design.

Figure 4.46 - The Splash Screen

Figure 4.47 - Home Screen

By selecting the two buttons in the toolbar below, user can choose the segmentation is by auto or manual.

Figure 4.48 – Auto/Manual Tap

By tapping the button on top-right will come out the setting page.

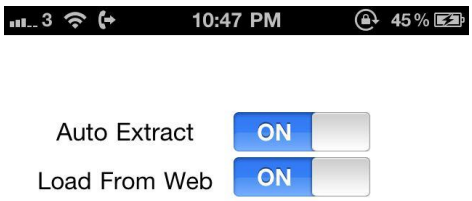


Figure 4.49 – Setting Page

Here is the final design for the Comic Viewing page.

Figure 4.50 – Comic Viewing Page

When a user clicks the Button at top-right hand corner, an Action Sheet will pop out.

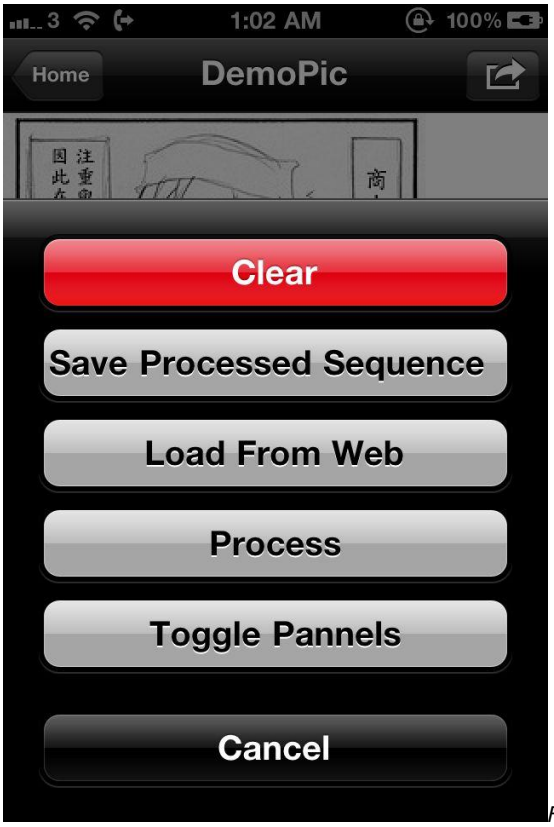


Figure 4.51 – Function Buttons

When a user taps save or load, the App will transfer data between the server and the App. A loading icon will be shown.

Figure 4.52 - Loading of XML Data from Server

4.3.3 Server

In our system, we used PHP to implement the server.

SERVER 1.0

Initially, we have created a simple uploader, testupload.php, for the user to upload XML sequence manually. Whenever a user views a manga image, the App will load a prepared XML from the server from manga.php, then display each panel sequentially and accordingly.

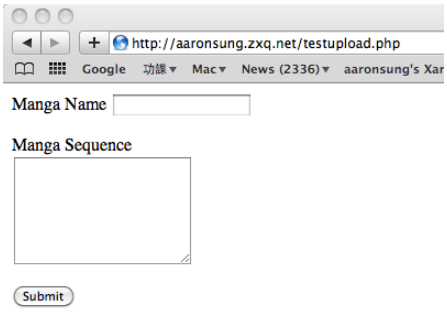


Figure 4.53 – UI of Manual XML uploader

SERVER 1.1

The manual uploader is then replaced by mangaupload.php, which is invoked by the App after the user manually generated an XML viewing sequence.

In term2, we have created a semi-automatic panel extractor process, which also makes use of the SERVER1.1, by uploading the generated XML to the server through mangaupload.php and request for the latest XML viewing sequence from manga.php.

We do not have any screen dump of the manga.php here as the content is pure XML and cannot by read unless viewing

the page source code.

In order to have a better demonstration on how the server works, we have also created a web console, which will be discussed in later parts.

4.3.4 Web Console

For demonstration purpose as well as better resource management, we have created a web console. This part is about the web console.

The initial design of the web console is several PHP files including a Manga List(mangalist.php) as well as a Manga Information Page(showxml.php) which displays the title of a manga and the related XML generated by the iPhone App, including manual and semi-automatic panel extraction results.

We have fine-tuned the structure of the web console so that:

I. It supports asynchronous page update.

II. The content is protected by a simple login system with cookie.

III. The system admin can remove any XML sequence.

IV. It looks better.

The flow chart below illustrates the initial logic of the initial design of the web console.

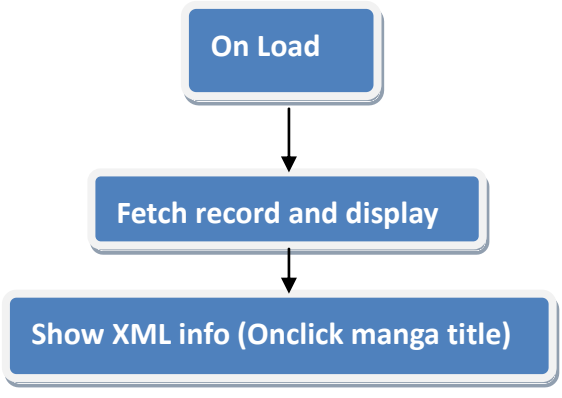


Figure 4.54 – Flow chart of web console

The following flow chart describes the simplified logic of the finalized web console.

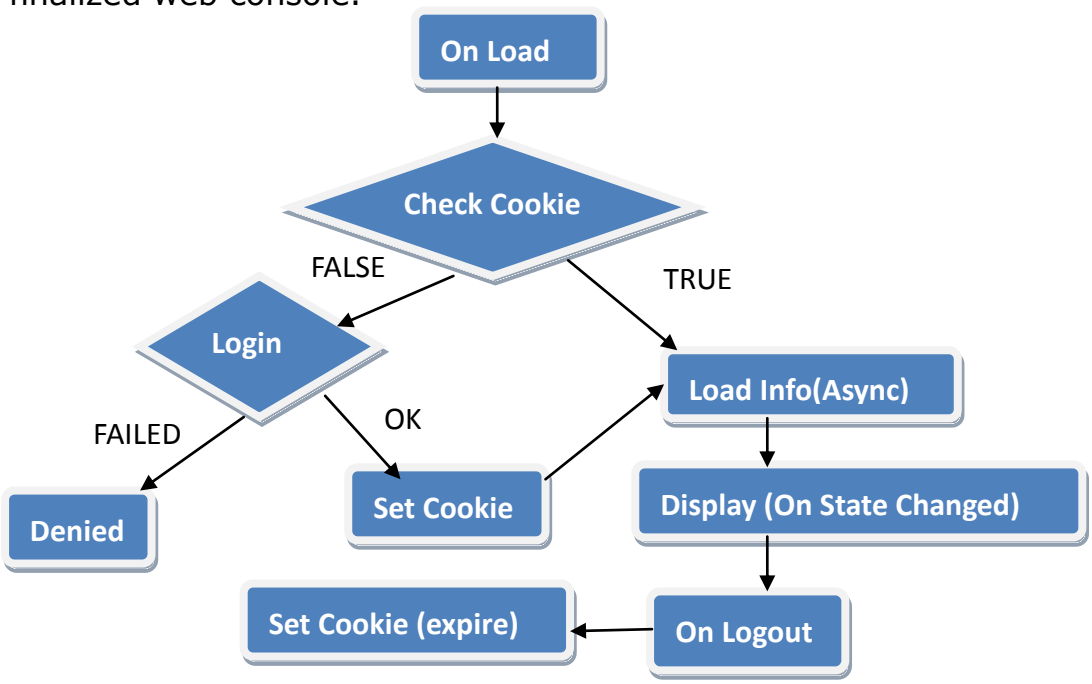


Figure 4.55 – Flow chart of web console, finalized

We will describe the web console in detail with some screen dump in the following paragraph.

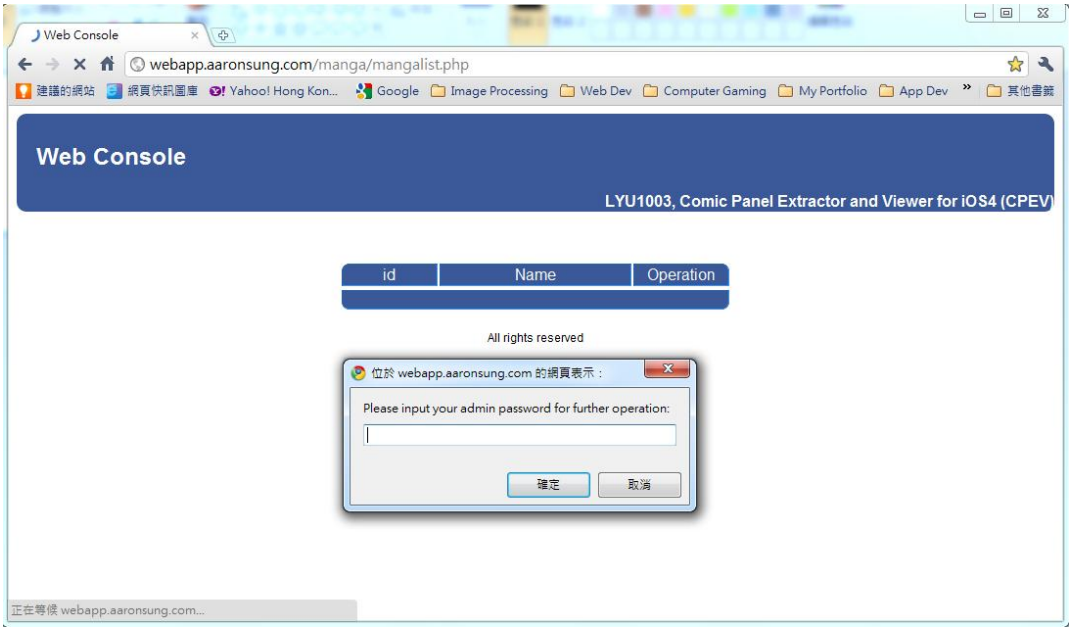


Figure 4.56 – simple login of Web Console

When the mangalist.php is loaded for the first time, the related cookie is null so that the system will ask for a password for

further operations.

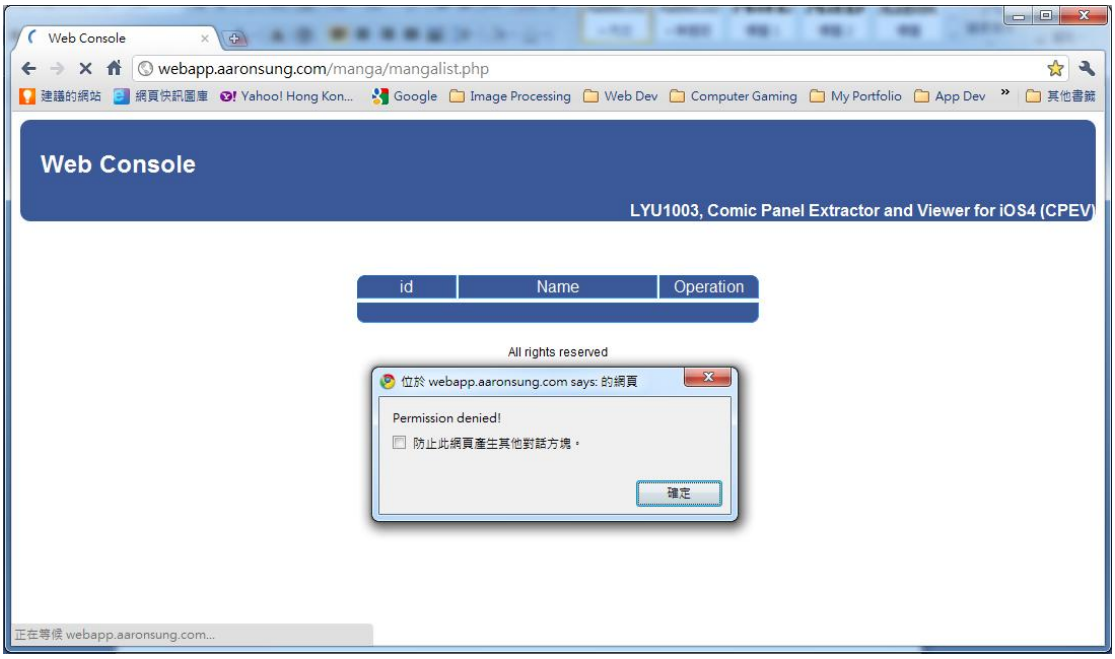


Figure 4.57 – login failed, Web Console

After the user input the password, the page will generate an asynchronous request and sent it by GET method to checkPW.php.

checkPW.php simply check if the password is expected or not, return 1 if so, else return 0. On the result returned, the Javascript on mangalist.php will first set cookie then load the information if the returned result is positive, else alert the user that permission denied.

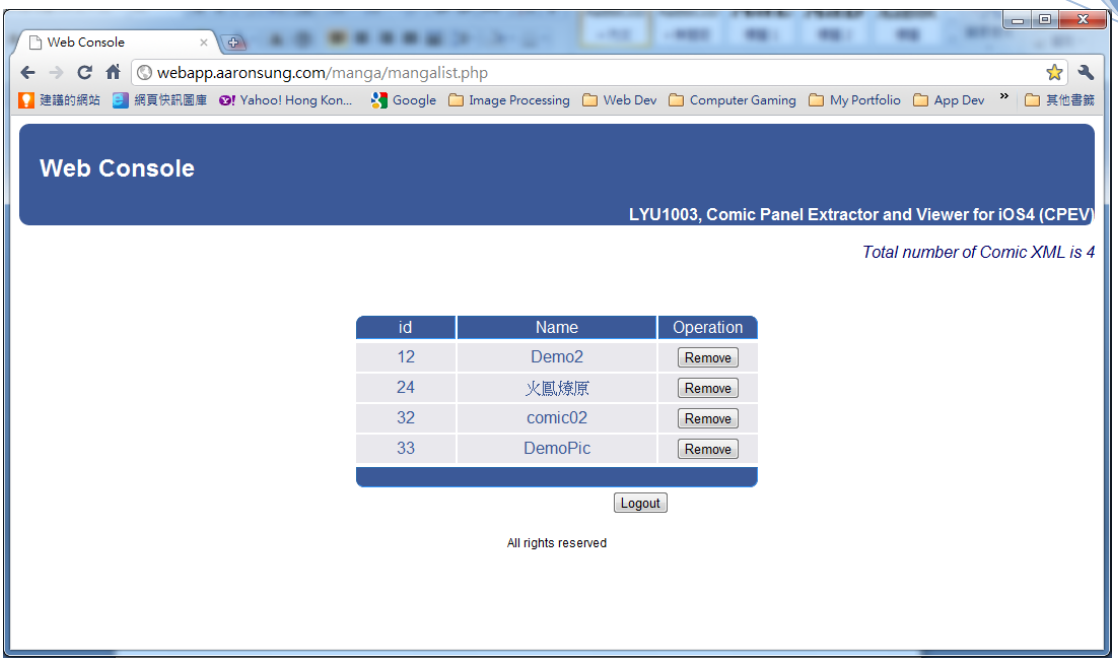


Figure 4.58 – mangalist.php of Web Console

Loading information of the XML sequences is also done by asynchronous request and response. The corresponding handler is loadInfo.php which, for the sake of simplicity, we prepared for the HTML format of the data fetched in loadInfo.php as a table, instead of returning a JSON structure. After the response, the update of mangalist.php is simply done by HTML-DOM innerHTML. In addition, we have another asynchronous page update for the total manga number.

For each record, we have an id, a name(title) which links to the showxml.php for XML sequences query, as well as a remove button. The corresponding handler of the remove button is the removeManga.php, which removes the record selected by asynchronous call, from the database and constructs another table for updating mangalist.php.

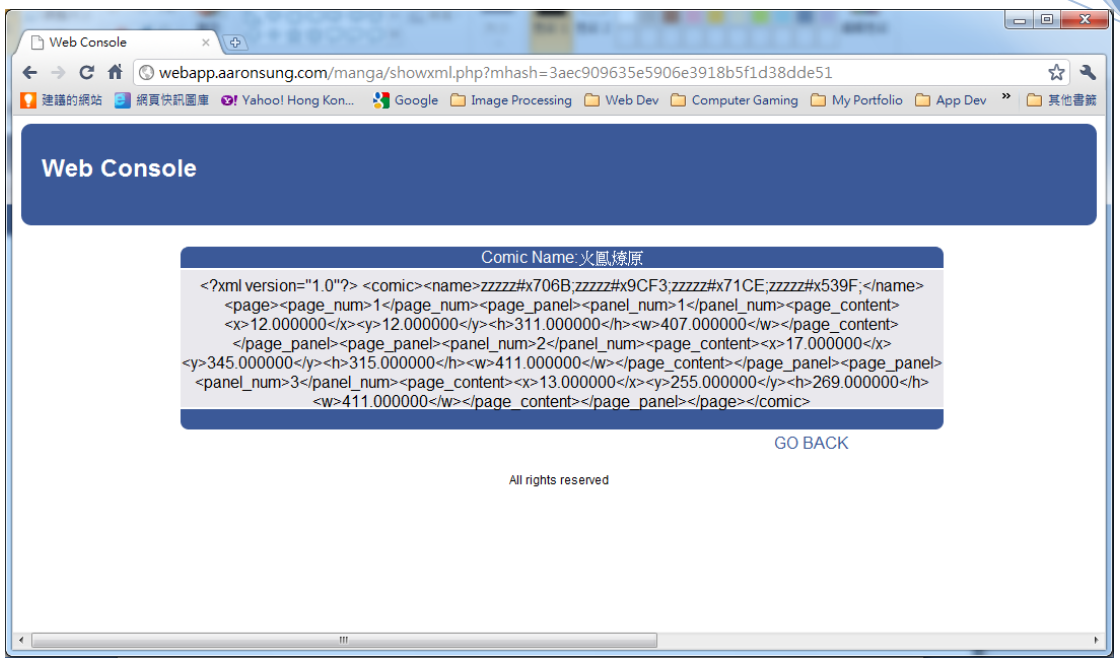


Figure 4.59 – showxml.php of Web Console

Above picture shows the showxml.php, which mainly displays the XML sequence of a specified manga.

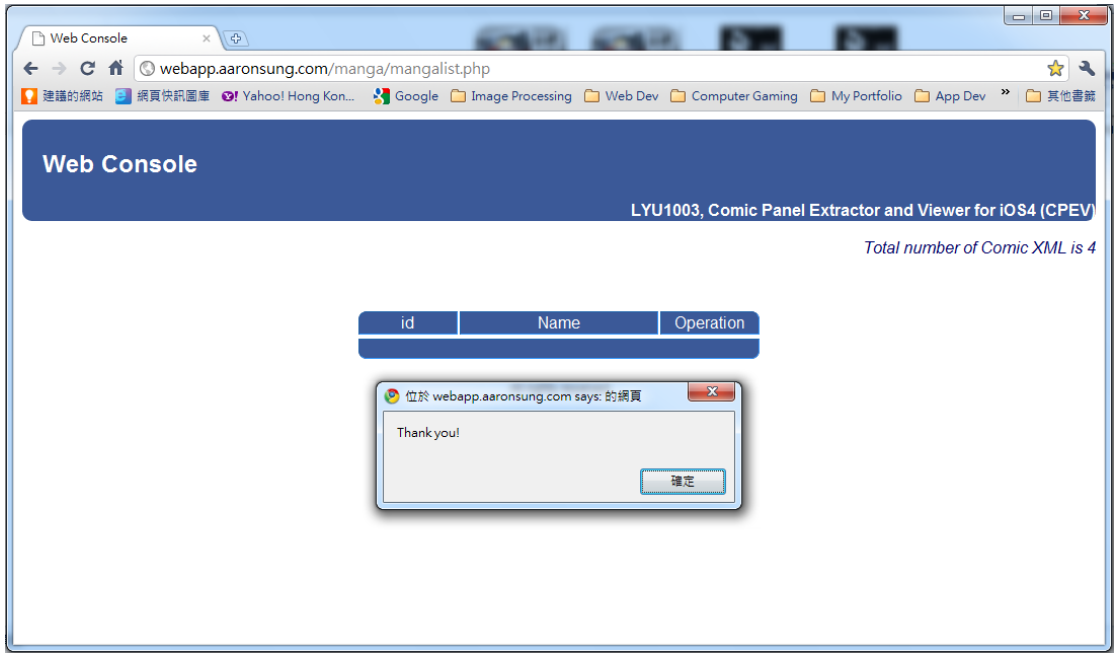


Figure 4.60 – logout, Web Console

On click logout button, cookie is first cleared by Javascript. The button and the manga list are then removed by jQuery.

Chapter 5 Experimental Result

In this chapter, we are going to talk about the experimental results of the algorithm study as the following list:

- Algorithm Study
  - Histogram
  - Contour Finding
  - Hough Lines
  - Watershed Segmentation
  - Flood-fill
- Performance Study
  - Running Time on CPU
  - Image Size against Time

5.1 Algorithm Study

5.1.1 Histogram

We will describe the Histograms of 5 test cases we chosen in this part.

Image: #1

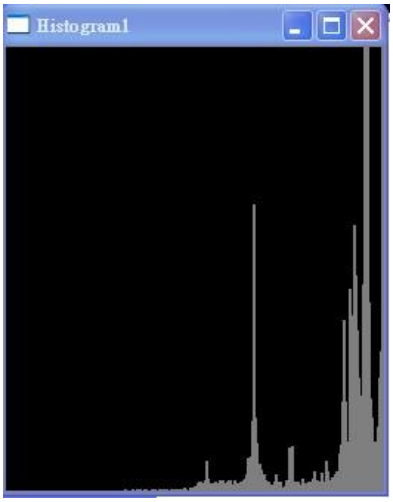


Figure 5.1 - Histogram for Image #1

Figure 5.2 - Input image #1

Image #1	Data
Max Value	762.277588
Index of Max. Value	242
Min Value	0.000000
Index of Min. Value	0

Table 5.1 - Peak value of Histogram for Image#1

Image: #2

Figure 5.3 - Histogram for Image #2

Figure 5.4 - Input image #2

This is another special case as we assume all images are in grey scale for development. We still include this image as it is used for testing Watershed Segmentation.

This image will be converted to grey scale before calculating the Histogram.

Image #2	Data
Max Value	1018.862000
Index of Max. Value	224
Min Value	0.000000
Index of Min. Value	0

Table 5.2 - Peak value of Histogram for Image#2

Figure 5.5 - Input image #3

Image: #3

Figure 5.6 - Histogram for Image #3

Image #3	Data
Max Value	656.016724
Index of Max. Value	0
Min Value	4.958656
Index of Min. Value	180

Table 5.3 - Peak value of Histogram for Image#3

This histogram is more special comparing with others as its index of max value is 0 which means pure black appears most frequently in this image. This is also related to the drawing style.

Image: #4

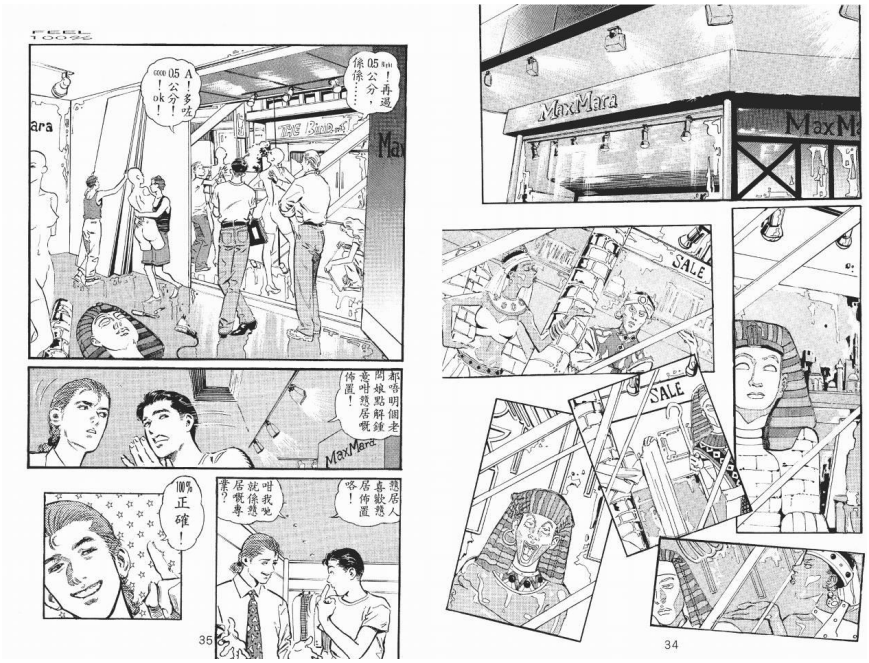


Figure 5.7 - Input image #4

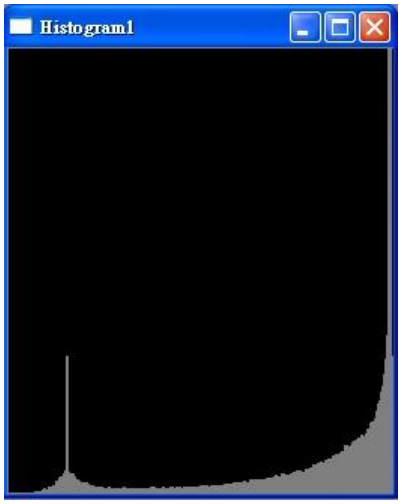


Figure 5.8 - Histogram for Image #4

Image #4	Data
Max Value	988.389404
Index of Max. Value	252
Min Value	0.000000
Index of Min. Value	1

Table 5.4 - Peak value of Histogram for Image#4

Image: #5

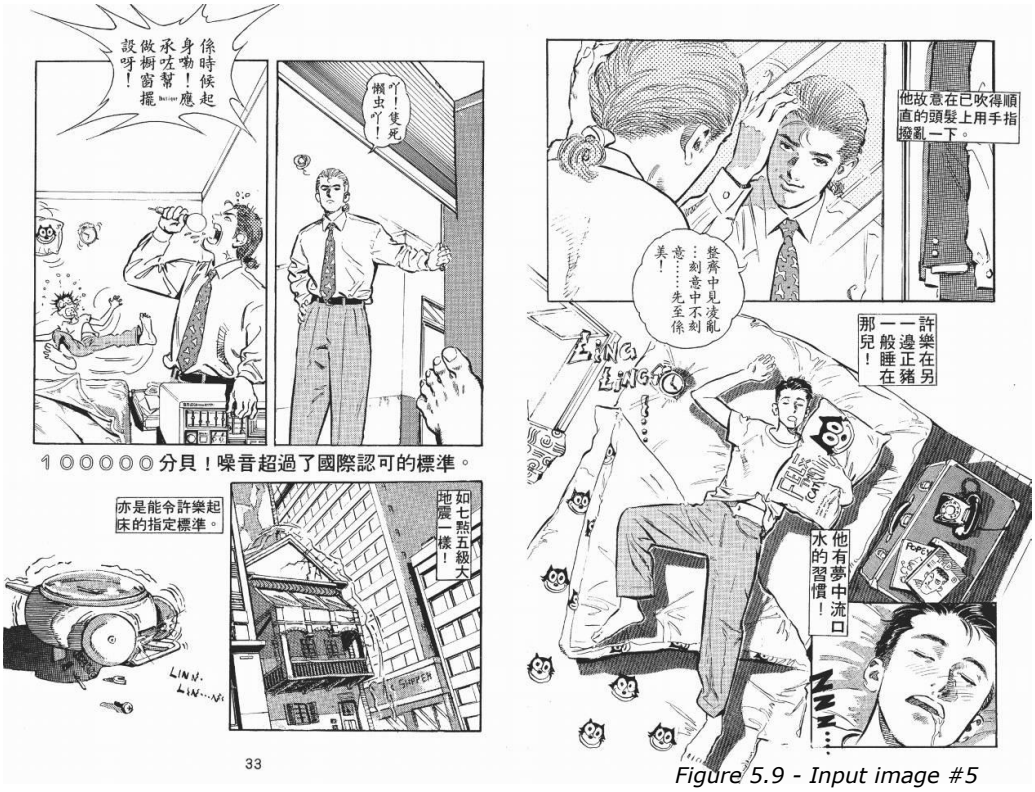


Figure 5.9 - Input image #5

Table 5.5 - Peak value of Histogram for Image#5

Image #5	Data
Max Value	1286.586304
Index of Max. Value	252
Min Value	0.000000
Index of Min. Value	0

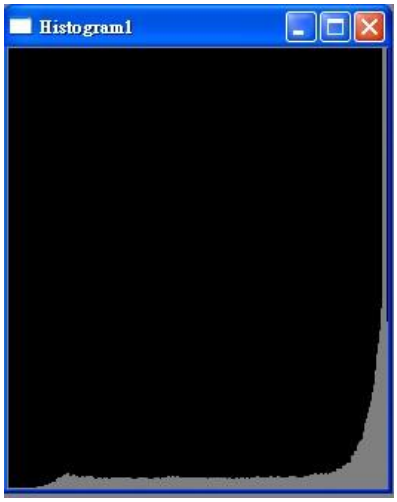


Figure 5.10 - Histogram for Image #5

Except image #3, max value relates to color index 220 to 255. This means, in general, white or a color near to white appears most frequently.

5.1.2 Contour Finding

Recalling that we have tried and compared two methods for image binarization:

Method1: Directly use cvThreshold()

Method2:

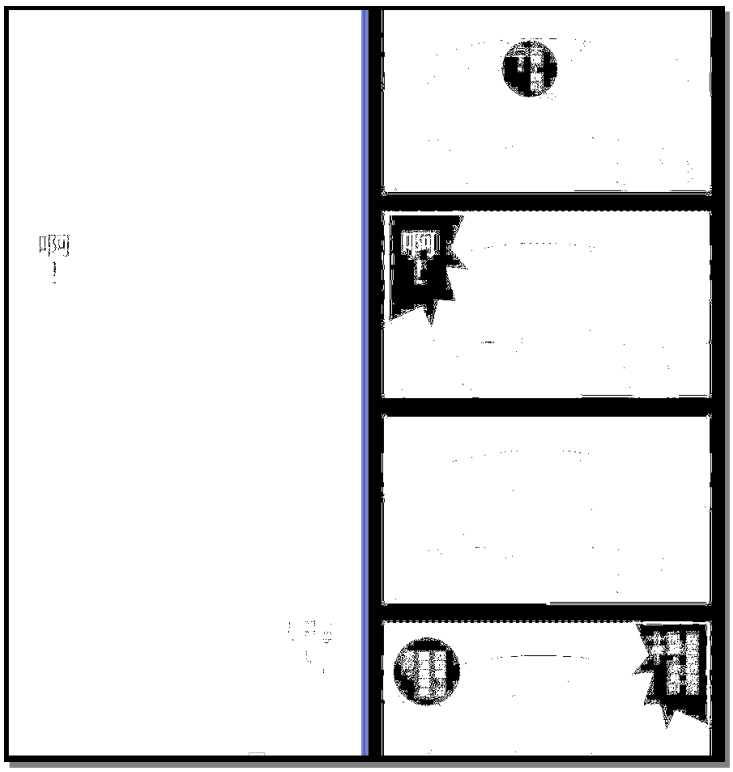
- Get background color through 4 sample points and take average value
- Calculate distance between background color and the image
- Binarized the image by a threshold, same as the one cvThreshold() used

We will describe the results of several test cases for the two methods respectively here.

Case: Pre-test
Purpose: to evaluate the threshold for further testing

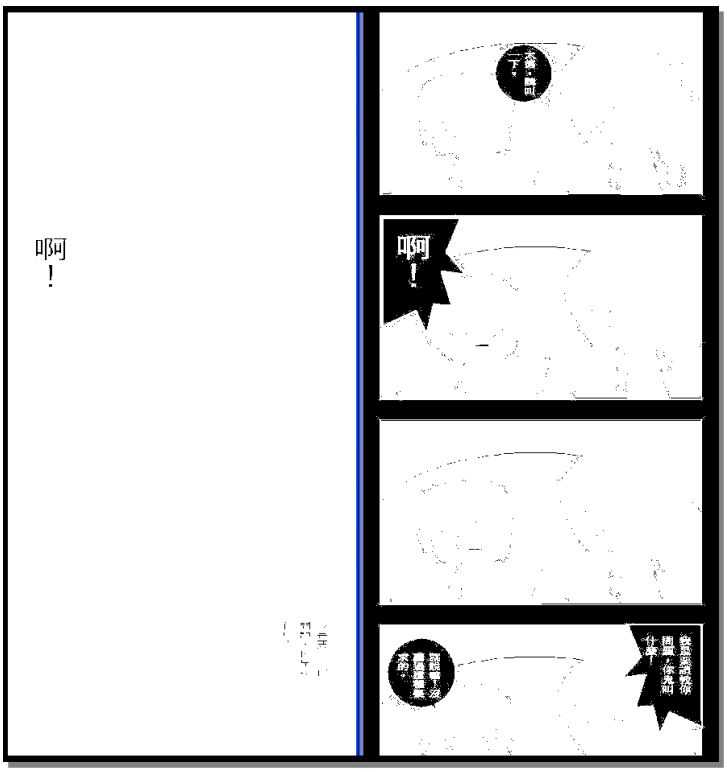
Figure 5.11 - Input image for Pre-test case

We have tried 5 sets of parameters for binarization. The following figures show the results of different parameters for the two methods. The left part is the result of method 1 while the right one is of method 2.



Case: Pre-test
Remarks: Contour result on threshold 0

Figure 5.12 - Binary results of Pre-test case on threshold 0



Case: Pre-test
Remarks: Contour result on threshold 10

Figure 5.13 - Binary results of Pre-test case on threshold 10

Case: Pre-test
Remarks: Contour result on threshold 100

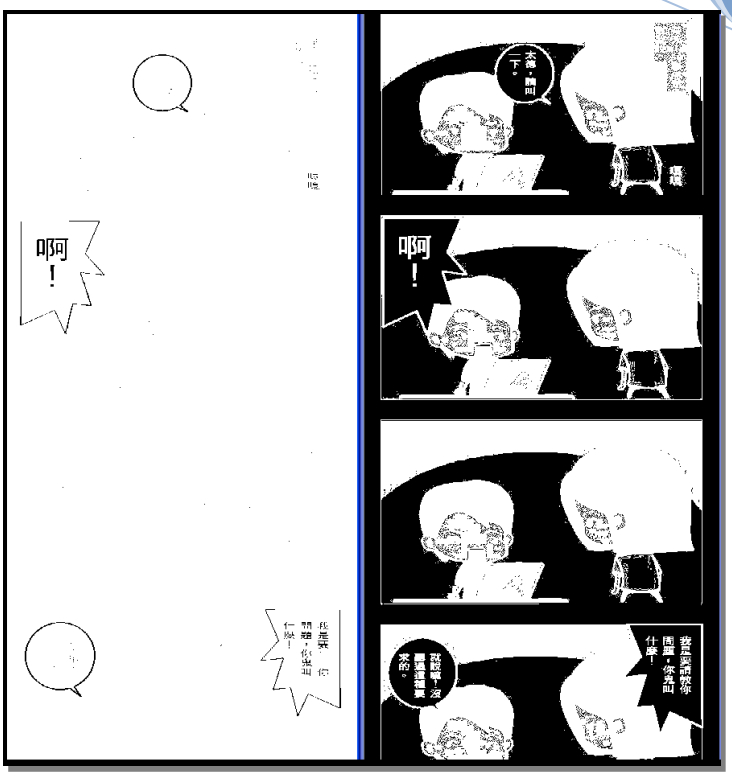
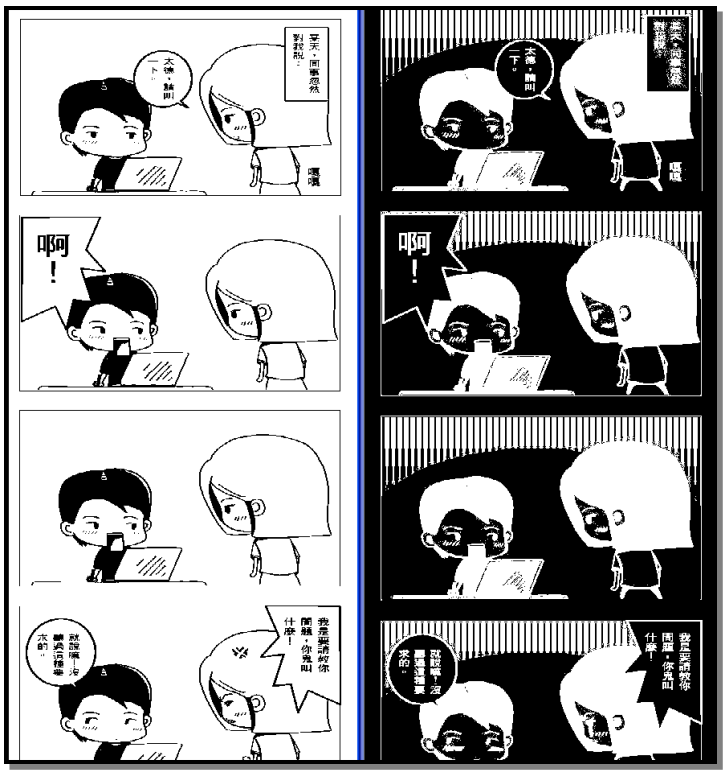


Figure 5.14 - Binary results of Pre-test case on threshold 100



Case: Pre-test
Remarks: Contour result on threshold 200

Figure 5.15 - Binary results of Pre-test case on threshold 200

Case: Pre-test
Remarks: Contour result on threshold 255

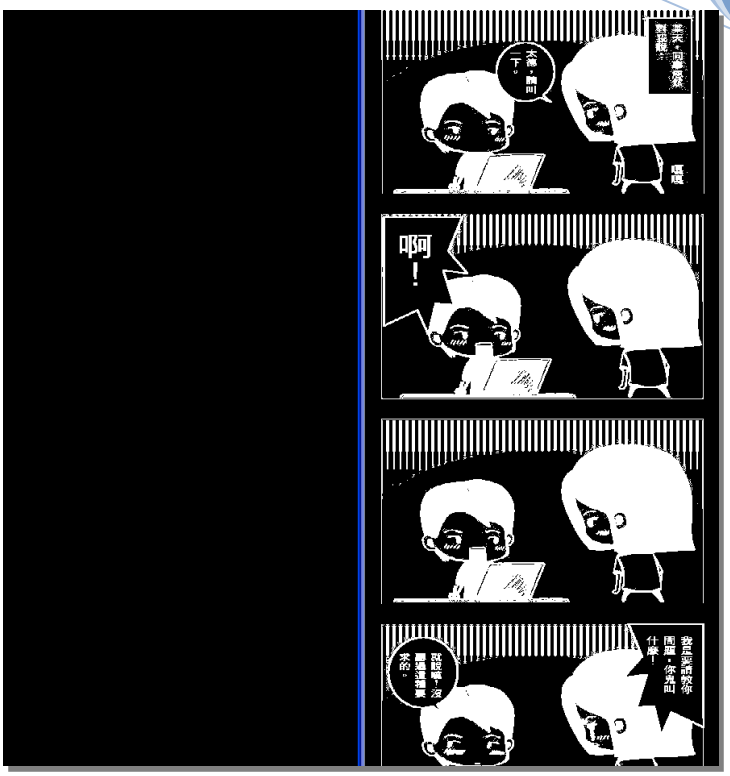


Figure 5.16 - Binary results of Pre-test case on threshold 255

By the pre-test case, we have chosen the threshold on 10 out of 255 for the following test cases. We focus on the right part of the result of the threshold on 10. With this result, we can get the outline of the four panels in a relatively good quality. Besides, we can also get the bounding box of the panels, which is some important information for zooming. However, as this is a very simple input image, we have to try some more complicated ones.

Following test cases is the full testing, including contour finding and bounding box locating.

Case: #1
Purpose: to find the contours as well as bounding box for a simplest Four Column Comic
Remark: Same as Pre-test case input

Figure 5.17 - Input Image for contour finding, Test case 1

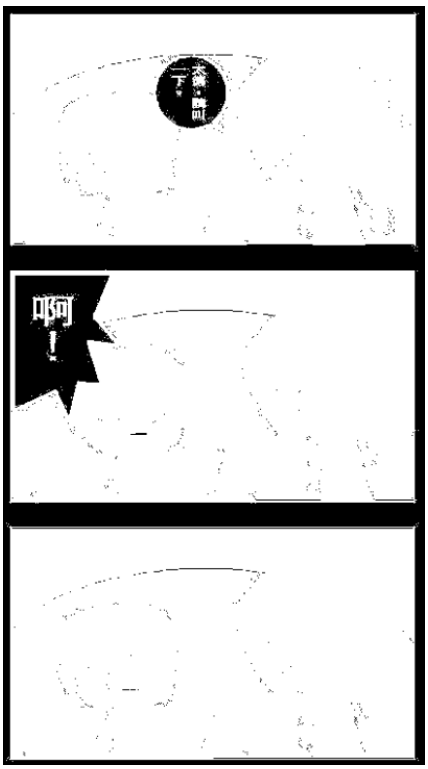


Figure 5.18 - Binary Image for contour finding, Test case 1

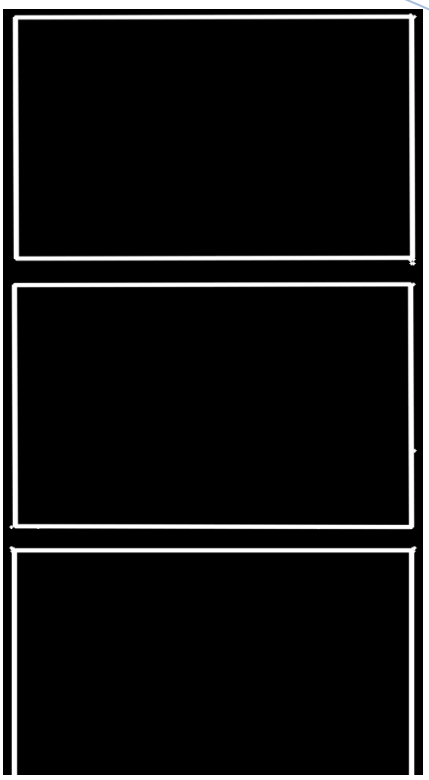


Figure 5.19 - Contour result (Partly), Test case 1

From the contour image shown above, we can have a very clear panel outline of the test case 1. The following screen dump shows the statistics of the bounding box.

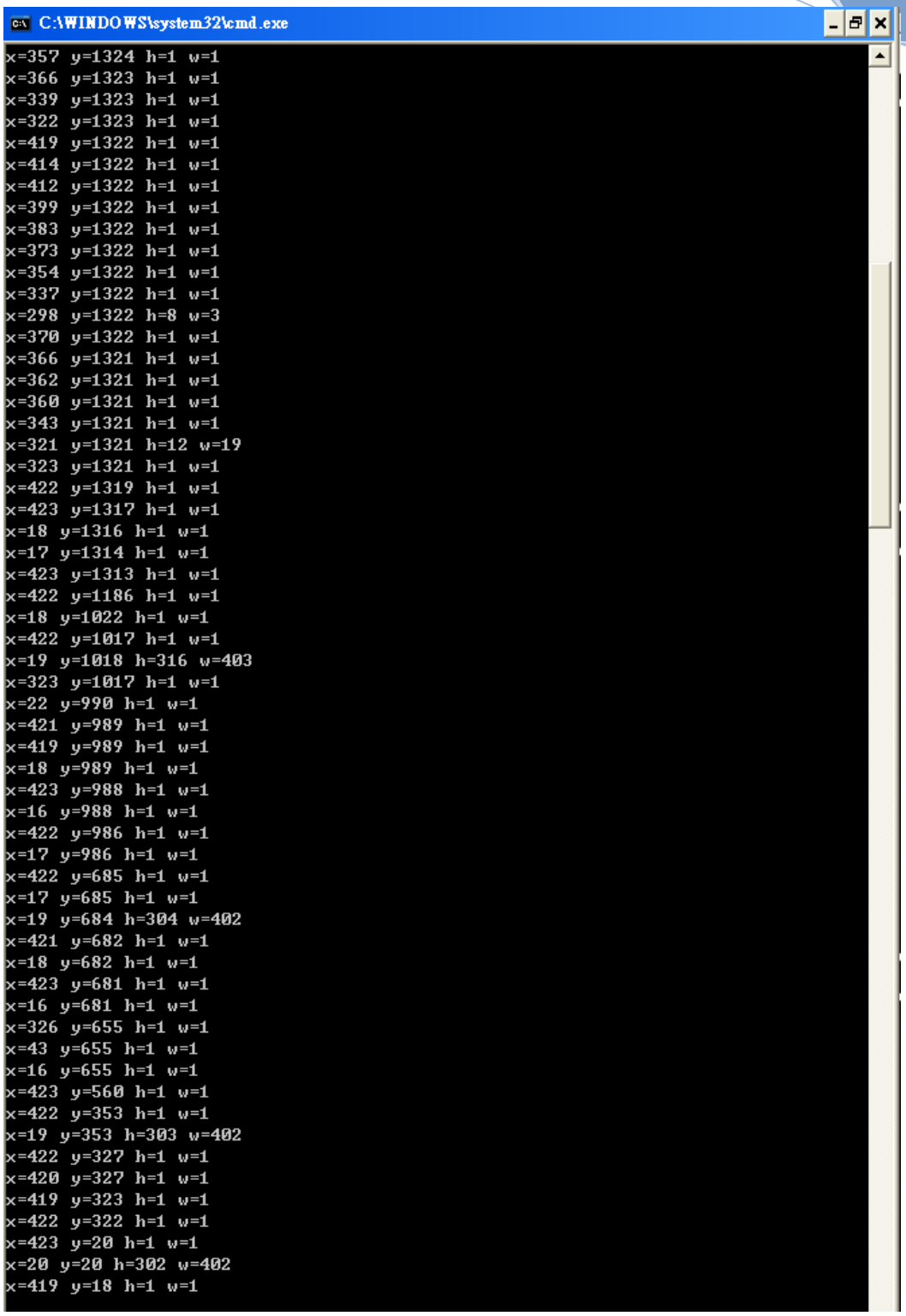


Figure 5.20 - Screen dump of statistics of bounding box, Test case 1

In this screen dump, x and y mean the starting x, y coordinates of the bounding rectangle while h and w are the height and width of the rectangle respectively. We can see that

most of the bounding boxes have width of 1 unit and height of 1 unit. These are all noise as they are single points. In addition, we have a rectangle with width of 19 units and height of 12 units. Its size is still too small to be a bounding box. Hence, it is noise too. All these needed to be eliminated by the algorithm when performing panel extraction. Noted that there are only 4 boxes in a reasonable size, this is what we want.

Case: #2
Purpose: to try finding the contours as well as bounding box for image with overlapped panels

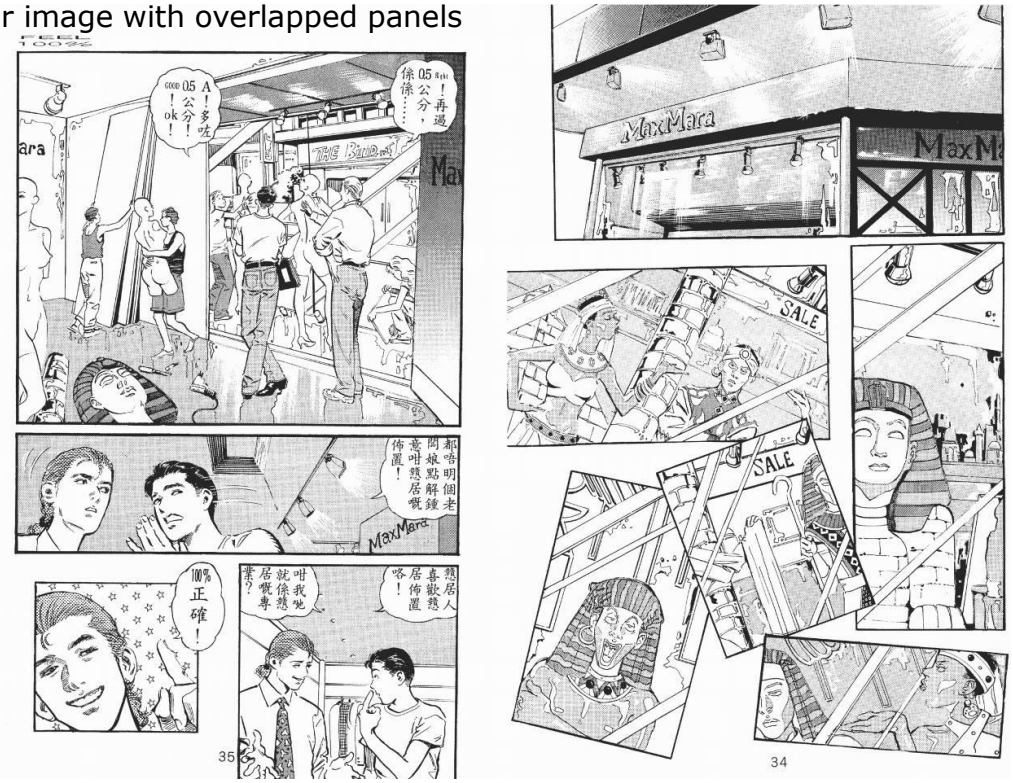


Figure 5.21 - Input image for contour finding, Test case 2

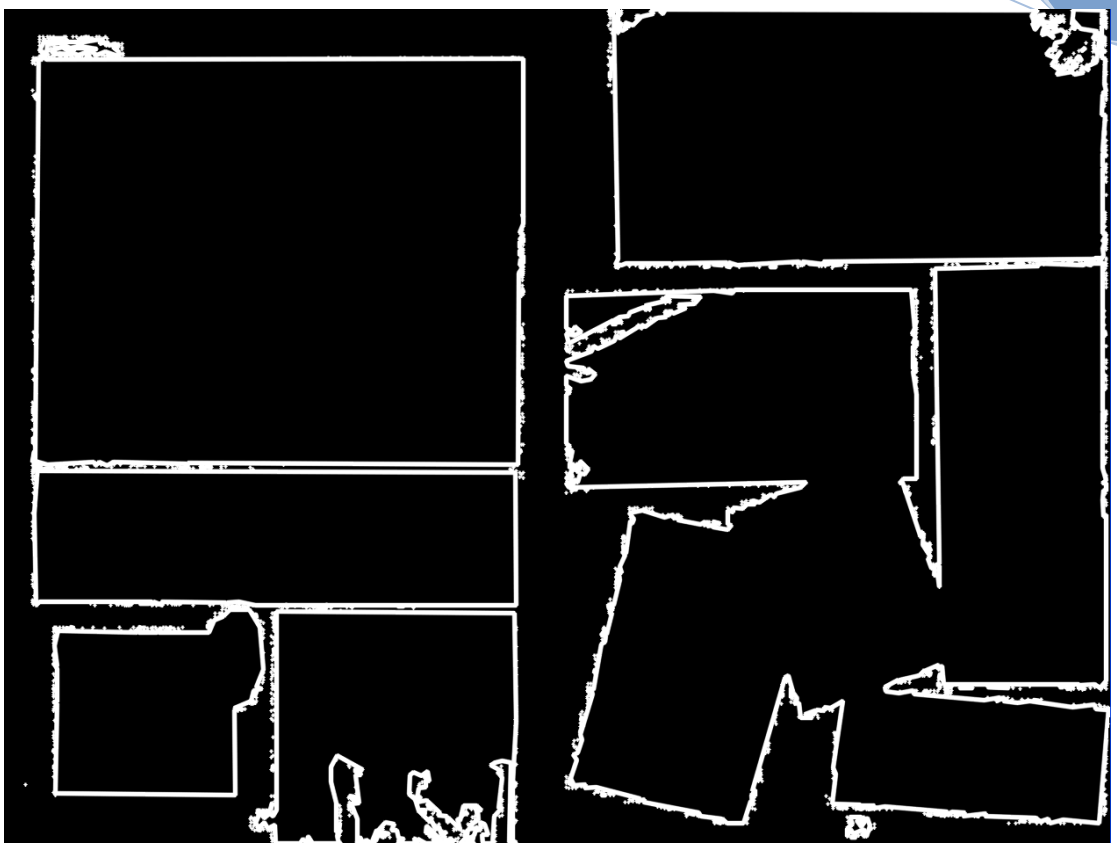


Figure 5.22 - Contour result, Test case 2

Test case 2 outputs a decent contour image as the one shown above. However, when it comes to bounding box, it cannot provide sufficient information as there are still some diagonals and some of the panels are not disjointed.

Case: #3
Purpose: to find the contours as well as bounding box for image with strong strokes and crossed-panel-blobs

Figure 5.23 - Input image for contour finding, Test case 3

This test case gave a total black result. There is only one bounding box which means only the universal bounding box is detected and no contour is found. This is because of the threshold. The threshold for test case 1 cannot fit this test case. Therefore, we have other test cases with the same image and threshold tuned. However, the results are not encouraging.

Case: #4

Test case 4 makes use of the original image of case 3, with threshold on 255 for demonstration purpose only. Other results are not shown here as either they have too complicated contours or the contours found not informative enough.

Figure 5.24 - Contour result, Test case 4

The above test cases show one thing – contour finding is a good tool to extract the panels. Yet, it is not powerful enough as we have to manually tune the threshold for comic pages in different style. Not only this, contour detection is not informative enough and we have to make use of other tools to gather more information. When we add up all results, we hope that some meaningful data can be retrieved to achieve the goal.

Summary:

- Comics with different style need different parameters to find out the panels.
- Contour is not enough for finding out the panel for all comic images.

5.1.3 Hough Lines Finding

As expected, the results cannot cover every edge of the panels.

Case: #1
Purpose: to find the Hough lines in a simple Four-Column Comic page

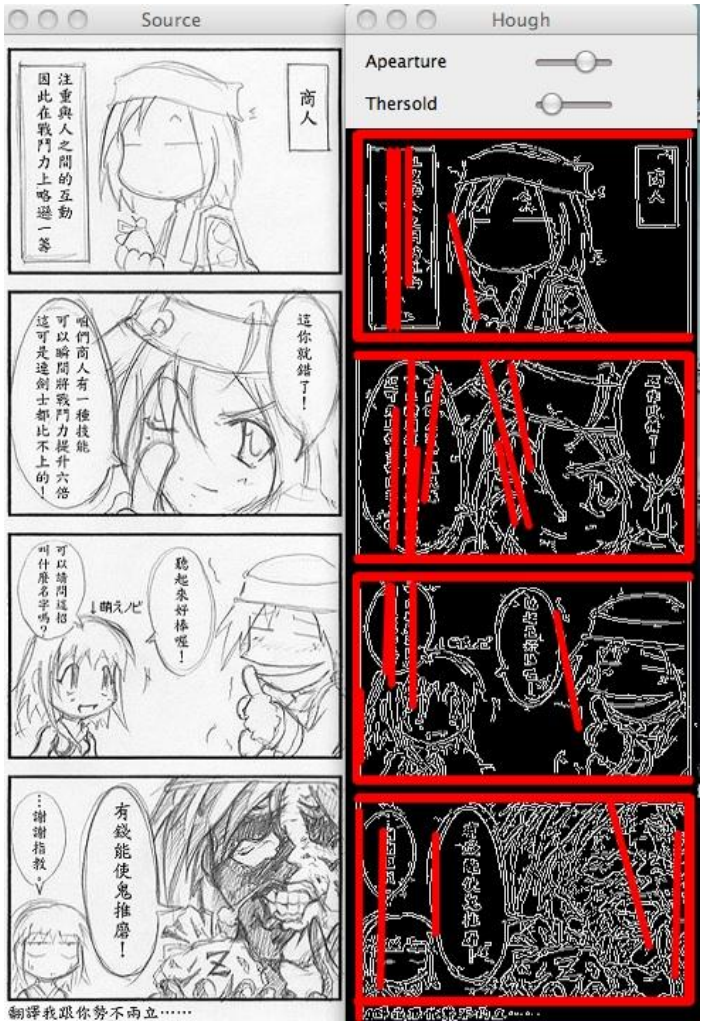


Figure 5.25 - Input and result for Hough Line finding, Test case 1

Case: #2
Purpose: to find the Hough lines in a traditional comic page with open panels

Figure 5.26 - Input image for Hough line finding, Test case 2

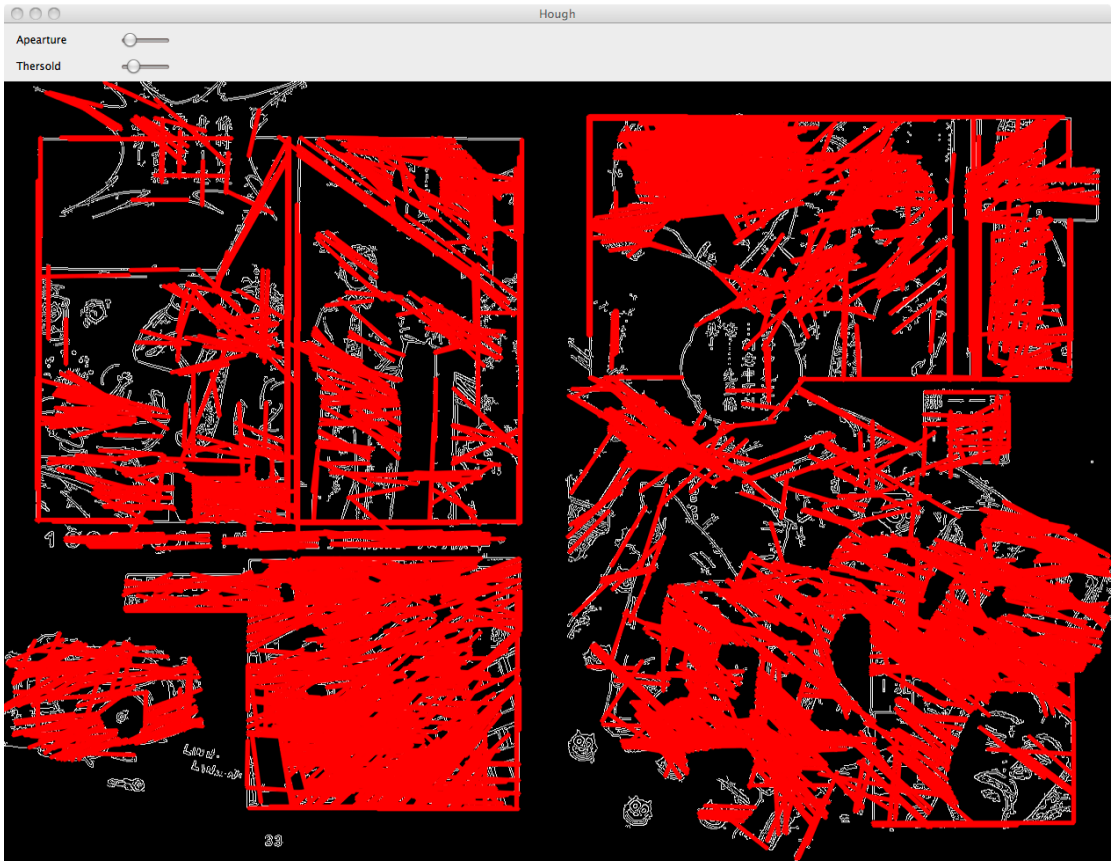


Figure 5.27 - Result of Hough line finding, Test case 2

Case: #3
Purpose: to find the Hough lines in a traditional comic page with overlapped panels

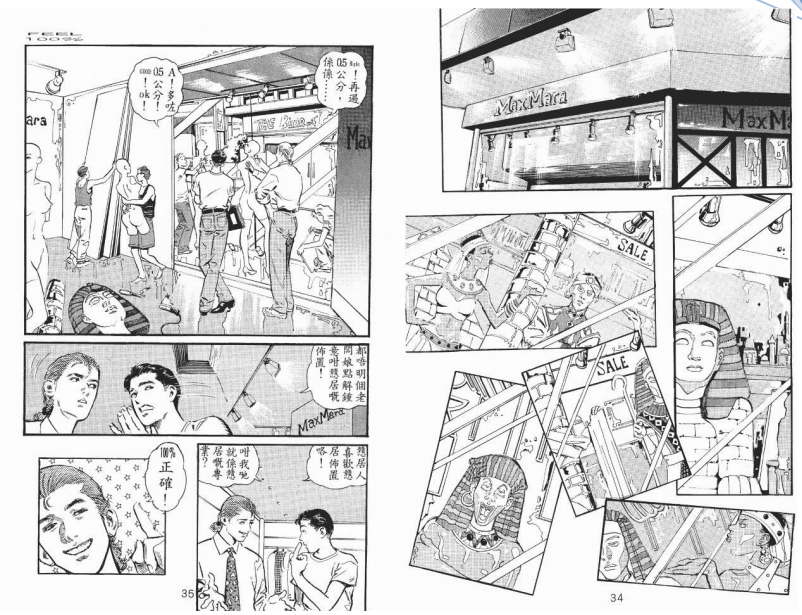


Figure 5.28 - Input for Hough line finding, Test case 3

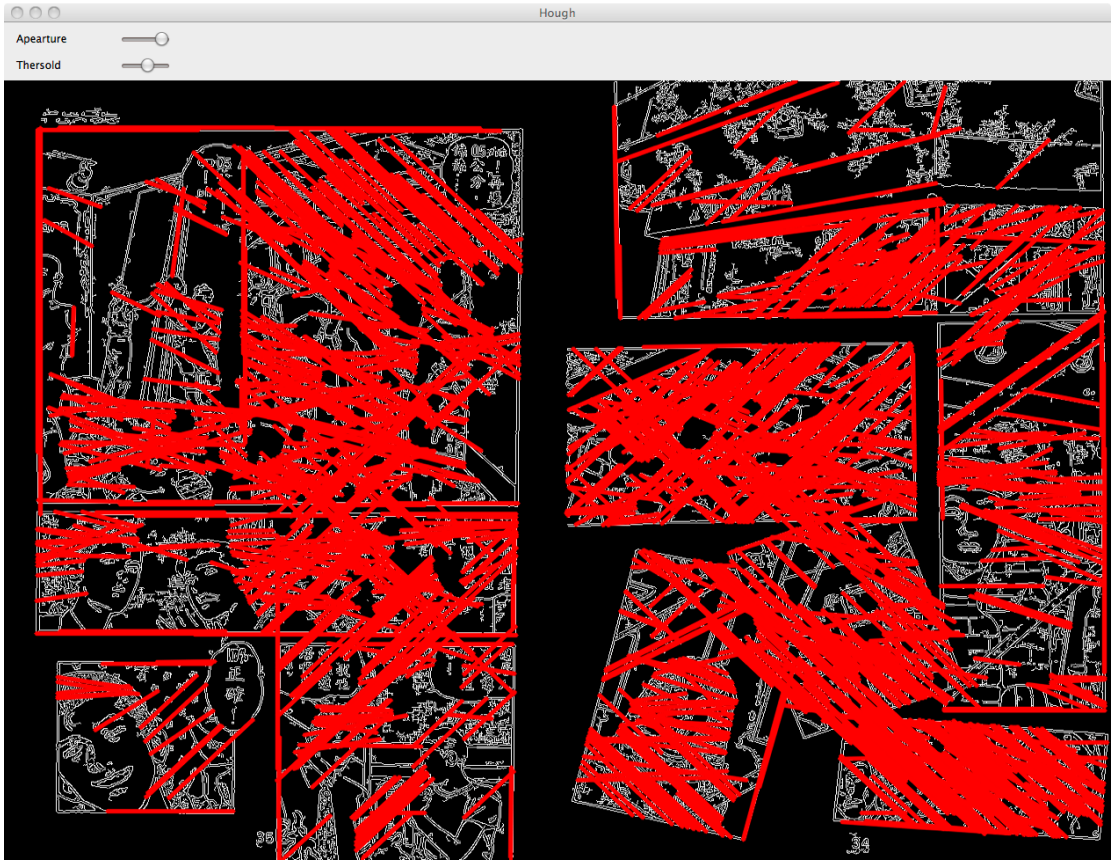


Figure 5.29 - Result for Hough line finding, Test case 3

Case: #4
Purpose: to find the Hough lines in a comic page with strong strokes

Figure 5.30 - Input for Hough line finding, Test case 4

Figure 5.31 - Result for Hough line finding, Test case 4

The above results indicate that the Hough lines may overlap

with the contours. Although this is not necessary, it is quite possible.

Another thing is that the longest Hough line in the result image has a strong probability to be a contour of the panel.

Summary:

- The longest Hough line in the result has a strong probability to be a contour of the panel.

5.1.4 Watershed Segmentation

We have 4 test cases for Watershed Segmentation which requires manually input the markers for panels.

Case: #1
Purpose: to perform Watershed Segmentation in a simple colorful Four-Column Comic page

Figure 5.32 - Input and result for Watershed Segmentation, Test case 1

In this program, contours are calculated before performing Watershed Segmentation. Noted that under most situations, it can segment the panels that we have marked, but it is not successful all the time. Test case 1 shows that although we three panels marked, it turns out to have filled only two of them.

Case: #2
Purpose: to perform Watershed Segmentation in a simple Four-Column Comic page in grey scale
Remarks: Watershed Segmentation can be performed on both colorful and grey scale image

Figure 5.33 - Input image for Watershed Segmentation, Test case 2

Figure 5.34 - Result image for Watershed Segmentation, Test case 2

Case: #3
Purpose: to perform Watershed Segmentation in a comic page with overlapped panels
Remarks:
a. Open panels not filled
b. Overlapped panels filled

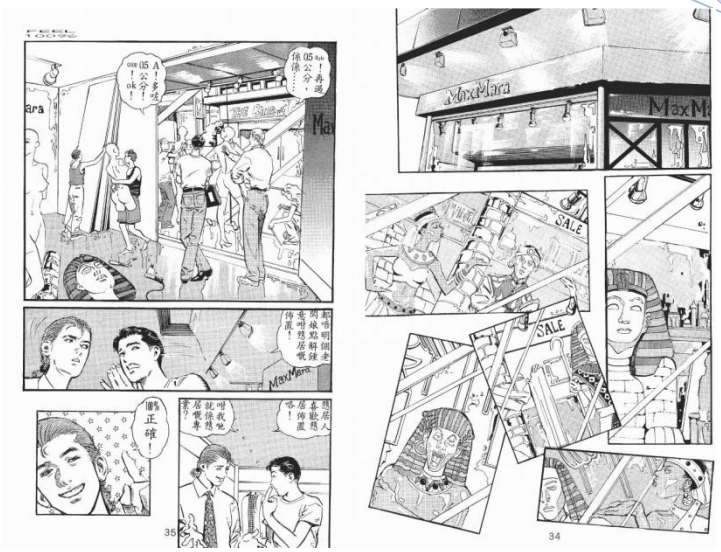


Figure 5.35 - Input image for Watershed Segmentation, Test case 3

Figure 5.36 - Result of Watershed Segmentation, Test case 3

Case: #4
Purpose: to perform Watershed Segmentation in a comic page with open panels
Remarks:
As expected, open panels not filled

Figure 5.37 - Input image for Watershed Segmentation, Test case 4

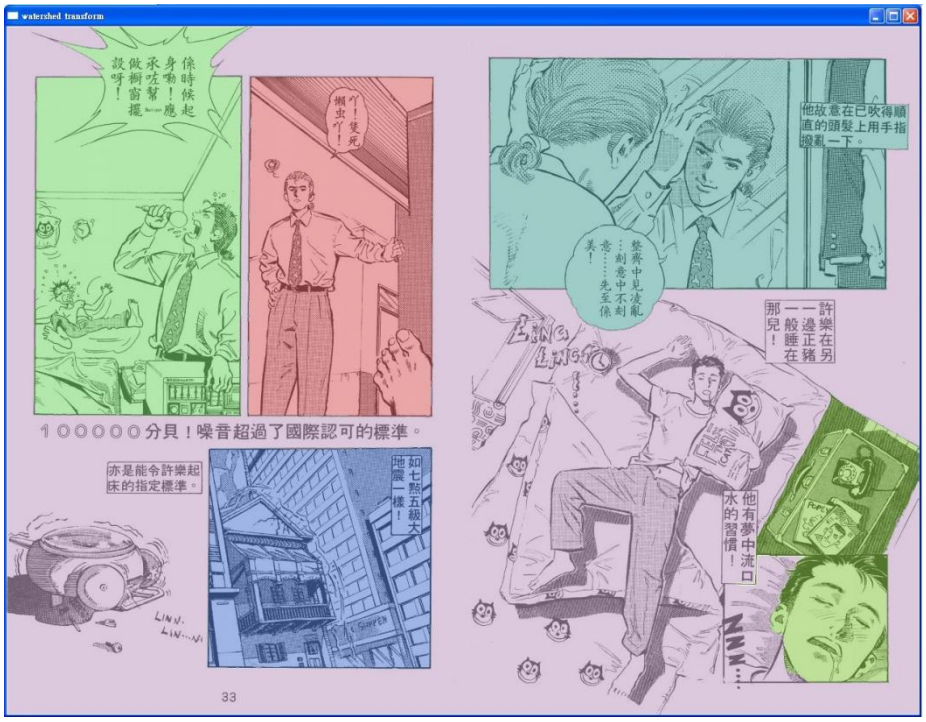


Figure 5.38 - Result of Watershed Segmentation, Test case 4

As we can see in the above two example, there are some panels or some regions with unclear contour are treated as the background and are filled together with the background. However, some overlapped panels are successfully filled with different colors.

Case: #5
Purpose: to perform Watershed Segmentation in a comic page with strong strokes
Remarks: Output not accurate

Figure 5.39 - Input image for Watershed Segmentation, Test case 5

Figure 5.40 - Result of Watershed Segmentation, Test case 5

The strong strokes of test case 4 affect the output of Watershed Segmentation. This is illustrated on the left figure.

Summary

- Strong strokes might affect Watershed Segmentation so that a panel cannot be fully filled.
- A region must be strictly closed so that Watershed Segmentation can be properly performed.

5.1.5 Flood Fill

Since this part does not involve too much, we will only demonstrate some outputs below. The background are manually selected and filled as red.

Figure 5.41 - Result of Flood-fill, Test case 1

Case: #1
Purpose: to perform Flood-fill in a comic page with open panels
Remarks: Simple Four-Column Comic page is omitted as the result is trivial once Case #1 is done.

Case: #2
Purpose: to perform Flood-fill in a comic page with overlapped panels

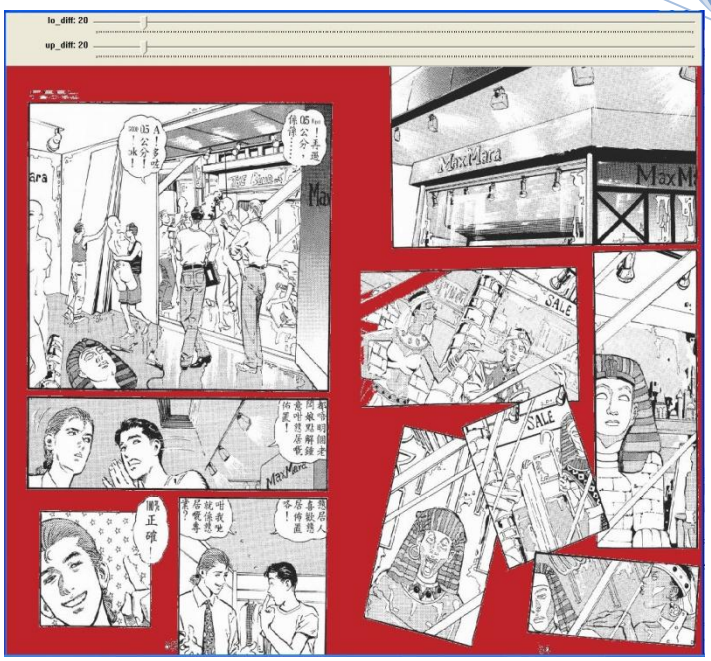


Figure 5.42 - Result of Flood-fill, Test case 2

Case 3:

The result of test case 3 shown on the left may be less helpful when comparing to case 1 and case 2.

Case: #3
Purpose: to perform Flood-fill in a comic page with strong strokes

Figure 5.43 - Result of Flood-fill, Test case 3

5.2 Performance Study

Some experiments have been carried out to investigate the performance of the algorithm and also the relationship between the size of input file and the time used. Since the main target is to measure the algorithmic performance, so CPU execution time is used for measurement. The main part of the program is bounded by two times() system clock in order to measure the CPU running time.

5.2.1 Running Time on CPU

The first part is testing the performance of the algorithm on PC. This part is done on a PC with 2.4Ghz Core 2 duo and 4 GB Ram. The PC is running on Mac OS X 10.6.

Name	火鳳燎原 1	火鳳燎原 4	足球小將 1	名偵探柯南 1
Page	102	108	91	94
Total Size(MB)	60.66	60.01	15.21	35.8
Time(s)	14.12	13.87	13.74	66.61

Table 5.6 – Performance on PC

5.2.2 Image Size against Time

This part focuses on the relationship of image size against time. This part of comparison is done on iPhone 4.

Size	1695x1300	1271x975	848x650	424x325	170x130
Time	1.324	0.727	0.416	0.167	0.035

Table 5.7 – Relation between Image Size and Time

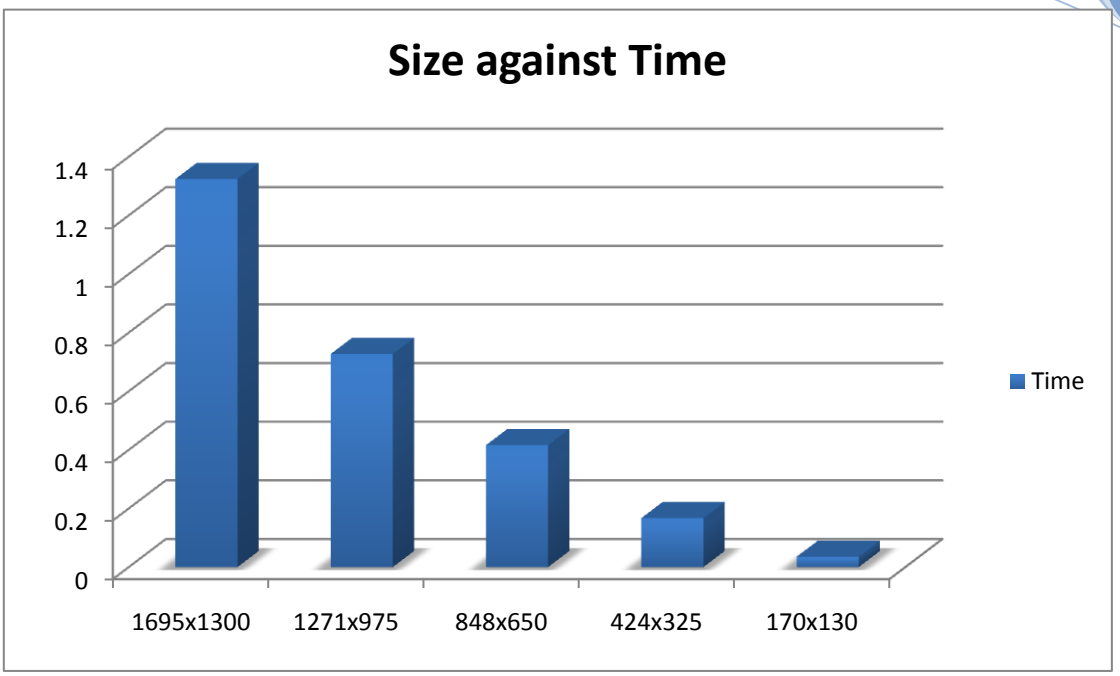


Figure 5.44 – Relation between Image Size and Time

Since the Algorithm is a linear time alogorithm (O(n)), when both the width and height is reduced by half, the total resolution is reduced to one-forth, and the processing time is reduced to one-forth also.

Chapter 6 Difficulties Encountered and Current Limitations

We are going to describe some difficulties we encountered while doing this project. There are 4 problems that we have encountered. Let us talk about them one by one.

6.1 Runtime Environment Problem

As we did not have an iPhone available at the beginning, runtime environment was a problem to our research. Hence, we used the iPhone simulator provided by XCode for debugging. However, when it comes to a concrete action, like zooming through multi-touching, it is impossible in simulator.

Therefore, we focus on image processing study on PC in the first term. All this algorithms are not ported into our iPhones due to limited time after our iPhones are available.

In addition, there are limitations of the device as well as the environment.

6.1.1 Limitations on Device

iOS devices are similar to other mobile devices, they only have limited computation power and memory. For those iOS4 devices, they have only 600Mhz to 800Mhz CPU and memory ranging from 256 Mb to 512 Mb. As a result, a lot of effort is spent on optimization, so to make sure that the App runs correctly under this environment.

6.1.2 Limitations on iOS

The memory management method on iOS framework always add extra burden on developers. Unlike some language such as Java, which provides automatic garbage collection, the memory management in iOS Framework is by a rather primitive reference counting technique. Besides, the common fork-and-exec for fighting memory leak on Unix-like platform cannot be used on iOS. This increases the difficulty for solving memory leak.

The memory leak problem is especially easy to come out when mixing the use of Objective C and C/C++. And sometimes you never know whether it is the problem of your own code or 3rd party library. As a result, a lot of time is needed to resolve the problem of memory leak.

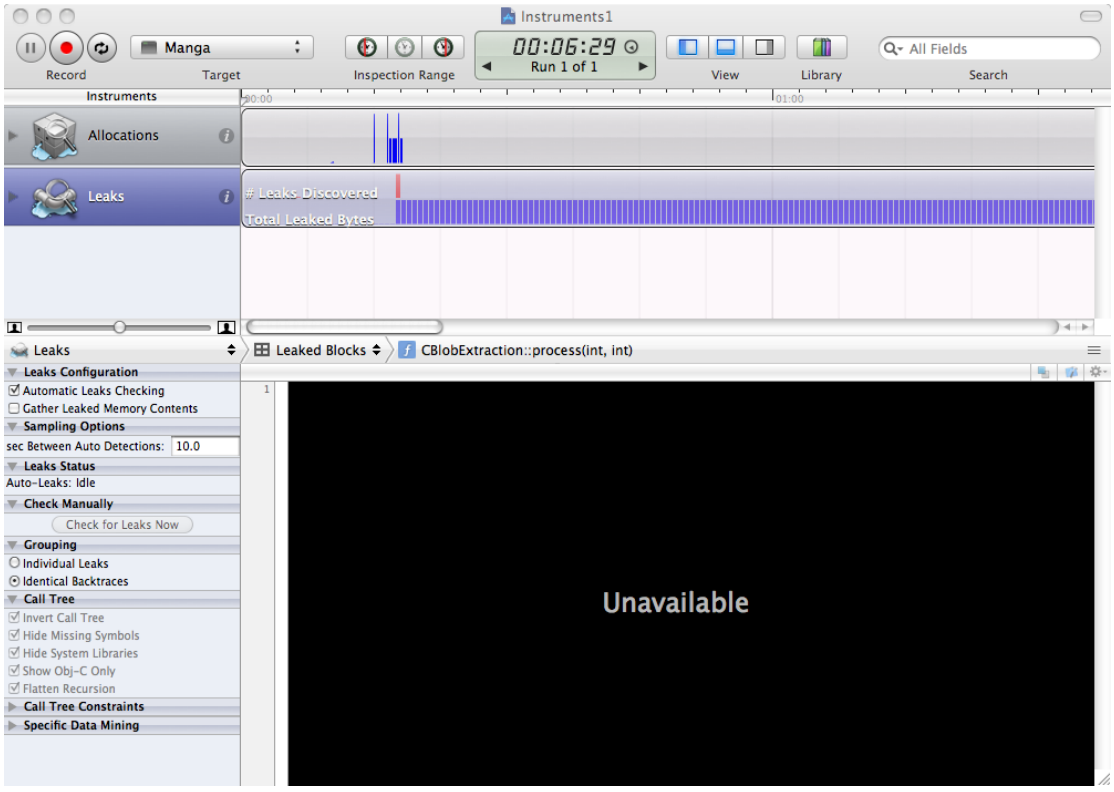


Figure 6.1 - A lot of time are spent on dealing memory leak

6.2 Automating the input

Our Watershed Segmentation needs manual input of some markers. It is a problem to automate this because, once we are able to automatically generate the markers, we have already got enough information for panel zooming. They are similar problems that we are facing.

In addition, the Watershed Segmentation does not always perform perfectly. To enhance the accuracy, we perform Flood-fill first. However, the Flood-fill also requires a manually-input seed point. Since the background pattern is not predictable in general, we cannot do this automatically without further assumptions.

6.3 Insufficient Libraries

iOS is mature enough for development purpose, however image processing libraries are currently not enough. For example, cvBlobsLib, which is for blob tracking, is not available in iOS. In case we need this library in the future, we have to port it into iOS ourselves.

6.4 Insufficient Documentation

Most of the 3rd party frameworks and libraries are lack of proper documentation. Many of them are lack of guidelines and tutorial so a lot of time is spent to understand how to manipulate them.

Two examples are OpenCV and GDataXML. Both of them have only one web page to describe how to use it. A lot of trial and error has been done. For example, there are condition build settings which is

different for simulator and iOS device but he documentation are not cleared enough so time has been spent to make it work.

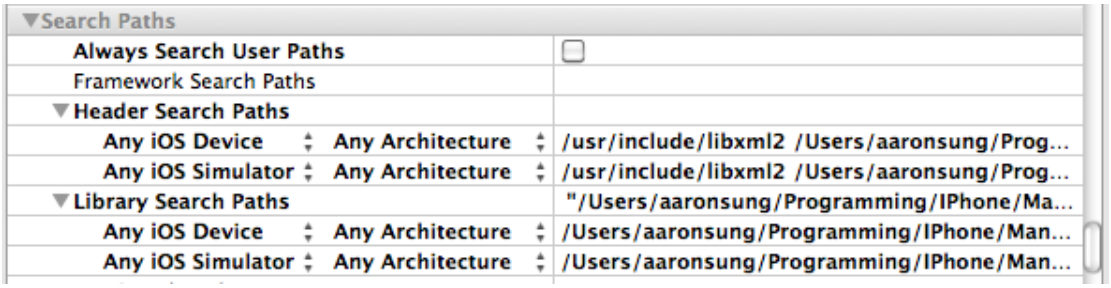


Figure 6.2 - Condition build settings for OpenCV on iOS

Another example is the GDataXML library. There is no documentation to explain what exactly the use of different functions and source code is needed to study before using it.

Chapter 7 Future works

In this chapter, we will talk about our future work.

7.1 Improve the Algorithm to Support Automatic Extraction

We have created a Manual Panel Extractor as well as a Semi-auto Panel Extractor on iOS4. However, the ultimate goal of us is to create an Automatic Panel Extractor on iOS4.

7.2 Investigate Comics that Contains Out of Scope Panels

Out of scope is a common phenomenon in comics. Many comics contain out of scope panels. However, due to the nature of most of the segmentation algorithms, such as Flood-fill or watershed, they fail to process out of scope panels. More research and investigation will be done in the future to see whether the problem can be solved.

Figure 7.1 - An example of page that out-of-scope panels occurs

7.3 Improve Sequencing Algorithm

The panel sequencing part is very important for generating the result and provides a good user experience. However, it is something that hard to develop well. Therefore, further investigation should be done to improve the algorithm.

7.4 Tile View on the App

Since the memory on iOS device is very limited, so some optimization can be used to lower the memory consumption while maintaining the viewing quality. One method is using tiling on the view, which is a method suggested by Apple.

The idea is a bit like mip mapping in 3D graphics. Different sets of image tiles of different zoom ratio are created. Only those image tiled that are being show on the screen is loaded. When user zoom in, those sets of tiles for a higher zoom ratio is loaded and vice versa. This can ensure the image is not blurred while reducing memory consumption by removing unnecessary part and details.

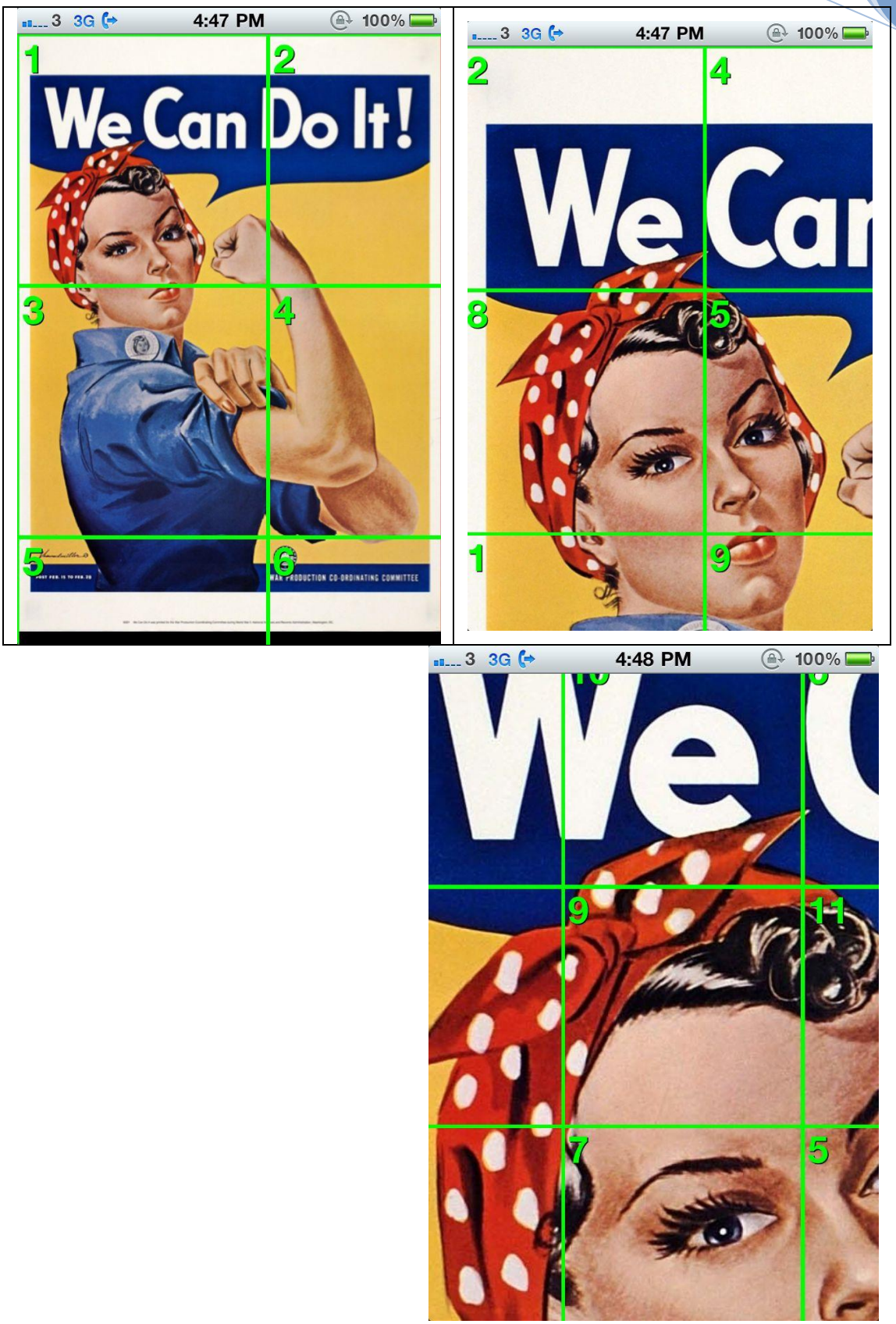


Figure 7.2 - Different set of tiles are used in different zoom ratio

7.5 Scale Down the Image for Extraction

This is kind of improvement of efficiency of the extraction process. The input image can be scaled down before applying panel extraction process. As we have study the relation between image-size against running time, we found out that there is room for scaling down the image with the result accuracy unchanged.

7.6 Mobile Webpage

The above suggestions are to improve the existing system while, the following 2 parts are suggestions for new approaches.

There are 3 types of iOS App:

I Webpage

II Webpage bundled with Safari

III Standalone App

Our App is of type 3. We suggest that a mobile webpage can be created and the panel extractor can be implemented in server-side.

Once a user upload a set of manga(comic), the panel extraction process will then be involved. Viewing sequence in XML format will then be stored in server together with the comic. With such a powerful browser, Safari, for iPhone, we might be able to use HTML-DOM manipulation to achieve the zooming effect for iPhone. For instance, we can use an image as background image of a div, modify the coordinates of the background can provide kind of zooming effect.

However, storing copyright work in our server is not wise.

7.7 Standalone App with Connection to Facebook

Facebook is the largest social community over the world. However, there is nothing to do with Facebook's popularity. What we want is the storage of Facebook. This idea is to establish the connection between the viewer and Facebook. The user can upload his/her comic to his/her personal album on Facebook, with privacy setting, in order to make the comic as private enjoyment. As a developer, we can get the permission from the user to retrieve their album and get the target comic.

This method is good as we can make use of Facebook's resource, the user can also share their comic with their friends.
However, the viewing sequence (XML) cannot be stored in Facebook. The linking between the comic content and the viewing sequence might be a problem, as the above two things are separated. We can have a unique ID for locating the XML in our server, but the ID is nothing to do with Facebook albums.

Chapter 8 Contribution of Work

Item	Aaron	Kenson
Background Research		
Existing Approaches	O	
Commonly Used Functions		O
Product Design		
System Architecture Design	O	
User Interface (Viewer)	O	O
User Interface (Web Console)		O
Algorithm Study & Experiment		
Histogram		O
Contour Finding		O
Hough Line Finding	O	
Watershed Segmentation		O
Flood-fill		O
Performance Study		
Size-Speed Relationship Study	O	
Implementation		
Manual Panel Extractor	O	
Semi-Automatic Panel Extractor	O	
Viewer	O	
Server	O	
Web Console	O	O
App Optimization	O	
Others		
Porting Library to iOS	O	
Graphics Design		O
Report Editing		
Introduction		O
Background Researches	O	O
Product Design	O	O
Progress Review	O	O
Experimental Result	O	O
Difficulties & Limitations	O	
Future Work & Suggestion	O	O

Table 8.1 – Contribution of work

Chapter 9 Conclusion

It is nice to cooperate with Aaron. My role is to explore possible algorithms for Panel Extraction in last semester and, in this term, Aaron focused on iOS App development and I worked as a support. For instance, I finalized the web console for this system, helped designing the user interface of the App, etc.

At the very beginning, we believed that we can finish the Semi-Auto Panel Extraction or even Automatic Panel Extraction with some basic algorithms, e.g. Contour Finding, Flood-fill, Connected Component Labeling, etc. We are suggested to get more information, e.g. Hough Lines, in order to generate an integrated result for the panel extraction.

We did a lot of experiments on each algorithm and thus spent plenty of time on it. Studying those algorithms is useful as we can learn the basic concept about image processing, that helps us to do further development.

Finally, we found an external OpenCV library, cvBlobsLib, which integrates all the algorithms that we want. Hence, we can finally finish the Semi-auto Panel Extractor. With a beta version for PC, which can detect the panels and store the viewing sequence of one page of comic, Aaron created an iPhone version which can deal with multi-paged-comics.

Though it is a bit simple, we proved that it is possible. It is also expected that such kind comic viewer for smart phone would be integrated with an automatic panel extractor, in the foreseeable future.

Chapter 10 Acknowledgement

We would like to take this chance to express our thanks to Prof. Michael Lyu, our supervisor of the final year project. The invaluable guidance and suggestion given from Professor benefit us a lot.

We would also express our thanks to Mr. Edward Yau for the incredible advices and support given by him.

Chapter 11 Reference

[1]http://big5.xinhuanet.com/gate/big5/news.xinhuanet.com/eworl

[2]C. Ponsard, V. Fries - *Enhancing the Accessibility for All of Digital Comic Books*

[3]F. Chang, C-J. Chen, and C-J. Lu. —*A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique",* Computer Vision and Image Understanding, 93(2):pp. 206-220, 2004.

[4]K. Arai & H. Tolle - *Automatic E-Comic Content Adaptation*

[5]T. Tanaka et el., *Layout Analysis of Tree-Structured Scene Frames in Comic Images*, IJCAI'07 Proceedings of the 20th international joint conference on Artifical intelligence

[6]H.C. Chung, H. Leung & T.Komura, "*Automatic Panel Extraction of Color Comic Images*," Advances Multimedia Inform. Processing – Pcm 2007, vol. 4810/2007, pp. 775–784, 2007.

[7]OpenCV Reference Manual, v2.1, March 18,2010

[8]The watershed transformation for multiresolution image segmentation, S. Wegner, T. Harms, J. H. Builtjes, H. Oswald and E. Fleck

[9]http://www.aishack.in/2010/03/connected-component-labelling/

[10]H. Samet and M. Tamminen (1988). "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees". IEEE Transactions on Pattern Analysis and Machine Intelligence (TIEEE Trans. Pattern Anal. Mach. Intell.) 10: 579. doi:10.1109/34.3918

[11]Michael B. Dillencourt and Hannan Samet and Markku Tamminen

(1992). "A general approach to connected-component labeling for arbitrary image representations}". J. ACM.

[12]http://niw.at/articles/2009/03/14/using-opencv-on-iphone/en

[13] Shimpi A. The iPhone 3GS Hardware Exposed & Analyzed. [Internet]. Available from: http://www.anandtech.com/gadgets/showdoc.aspx?i=3579&p=2.

[14] iPhone 4 Teardown – Page 2. [Internet]. [cited 2010 Jun 23]. Available from: http://www.ifixit.com/Teardown/iPhone-4-Teardown/3130/2.

[15] iPhone 4 Confirmed to Have 512MB of RAM (Twice the iPad and 3GS). [Internet]. 2011 Available from: http://www.macrumors.com/2010/06/17/iphone-4-confirmed-to-have-512mb-of-ram-twice-the-ipad-and-3gs/.

[16]Apple, Inc. Memory Management Programming Guide. [Internet]. [cited 2011 Apr 10]. Available from: http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/MemoryMgmt/Articles/mmAutoreleasePools.html#//apple_ref/doc/uid/20000047-1041876.

[17]http://yester-place.blogspot.com/2008/06/opencv.html

[18]http://www.opencv.org.cn/index.php/Template:Doc

[19]http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

[20]http://fsa.ia.ac.cn/opencv-doc-cn/opencv-doc-cn-0.9.7/ref/opencvref_cv.cn.htm

[21]http://opencv.willowgarage.com/documentation/cpp/index.html

[22]http://binarymillenium.com/2009/03/opencv-example-and-why

-does-google-do.html

[23]http://www.pages.drexel.edu/~nk752/tutorials.html

[24]http://blog.csdn.net/wqvbjhc/archive/2010/04/17/5497017.aspx

[25]《OpenCV 教程 基礎篇》北京航空航天大學出版社，劉瑞禎、于仕琪編著

[26]《Learn Objective-C on the Mac／Objective-C 基礎教程》人民郵電出版社，Mark Dalrymple、Scott Knaster 著，高朝勤、楊越、劉霞等譯

[27]《iPhone Apps 編寫速成班》Xtips 工具叢書系列月刊，Vol.56

[28]http://www.sciencenet.cn/blog/Econtent.aspx?id=374856

[29]http://dasl.mem.drexel.edu/~noahKuntz/openCVTut4.html

[30]http://www.fact-index.com/f/fl/flood_fill_example_in_c.html