

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

2006-2007 Final Year Project

**First Term Report**

**LYU0602**

**Automatic PhotoHunt Generation**

Supervisor:

**Professor Michael R. Lyu**

Prepared by:

**Shum Hei Lung To Wan Chi**

## Abstract

---

Computing could replace man's job in certain ways. While the computation speed becomes higher and higher, even a mobile device has enough power to do some complicated jobs that could never be done in the satisfactory time in the past.

In this project, which is name Automatic Generation "PhotoHunt" and is the project LYU0602 from Prof. Michael R. Lyu, we are trying to automate the process of generating mistakable-images from the original one. Since the process is automatically done by itself, the possibility of the game is therefore unlimited.

What includes in this report is the description of the project, our motivation and objectives, the progress of development and what we have learnt in the semester 1. We also define the definition of the game. Several algorithms would be shown on the report such as segmentation and edge detection. Some tests and the game engine on web are included too. At last, limitation and difficulty that we encountered could be found here.

## Table of Contents

Abstract .....	2
Chapter 1 Introduction .....	7
1.1 Motivation .....	7
1.2 Project Objectives .....	8
Chapter 2 Analysis of the PhotoHunt Game .....	10
2.1 What is PhotoHunt? .....	10
2.2 Basic approach to generate image .....	11
2.3 What makes good images for PhotoHunt? .....	13
2.4 Technical Required .....	13
Chapter 3 Digital Image Processing .....	14
3.1 How does image processing apply to our project? .....	14
3.1.1 Smoothing .....	14
3.1.1.1 Median Filtering .....	15
3.1.1.2 Gaussian Filtering .....	15
3.1.2 Segmentation .....	17
3.1.2.1 Pixel-based Segmentation .....	18
3.1.2.2 Edge Detection .....	22
3.1.2.3 Region-based Approaches .....	25
Chapter 4 Pyramid Segmentation .....	26
4.1 Image Pyramid .....	26
4.2 Generation of the Gaussian pyramid .....	27
4.3 Segmentation by pyramid-linking .....	32
4.4 Gaussian Pyramid Interpolation .....	33

4.5	Generation of Laplacian Pyramid.....	34
4.6	Gaussian and Laplacian Pyramid in OpenCV.....	36
4.7	Pyramid Segmentation in OpenCV.....	37
4.7.1	The First Parameter - Threshold 1.....	37
4.7.2	The Second Parameter- Threshold 2.....	38
4.7.3	Estimation of the Parameter.....	38
4.7.4	Segmentation Result.....	39
<b>Chapter 5 Image Generation Engine.....</b>		<b>40</b>
5.1	Processing Flow.....	41
5.2	Segmentation module.....	43
5.3	Elimination Module.....	44
5.4	Color Change Module.....	44
5.5	Object Duplication Modules.....	45
5.6	Smoothing Image.....	45
<b>Chapter 6 Elimination Algorithms.....</b>		<b>46</b>
6.1	Direct Copy Algorithm.....	46
6.1.1	Testing Result.....	46
6.2	Horizontal Gradient Algorithm.....	48
6.2.1	Test Result.....	50
6.3	Nearest Boundary Algorithm.....	52
6.3.1	Testing Result.....	54
6.4	Enhanced Nearest Boundary Algorithm.....	56
6.4.1	Testing Result.....	56
6.5	Conclusion.....	59
<b>Chapter 7 Semi-Automatic Generation Program.....</b>		<b>60</b>
7.1	Introduction.....	60
7.2	Interface.....	60

7.3 Implementation.....	62
7.3.1 Flow Chart.....	62
7.3.2 Operations.....	64
Chapter 8 Game Engine for Web Application .....	67
8.1 Architecture.....	68
8.1.1 Data Flow .....	68
A. Processing User Requests.....	69
B. Game Playing .....	70
8.2 User Interface .....	70
8.2.1 Functionality.....	71
Chapter 9 Limitation.....	74
9.1 Segmented area found noise and distortion.....	74
9.2 No Global Information between the segments.....	74
9.3 No artificial intelligence control to the Applied effect.....	74
9.4 Not all images can be segmented.....	75
Chapter 10 Difficulties.....	76
Chapter 11 Project Progress.....	77
Chapter 12 Future Works .....	78
12.1 Continues to improve the segmentation algorithm .....	78
12.2 Building up a global view of the images .....	78
12.3 Introduce more effects .....	79
12.4 Adding features to the Web-Based game engine.....	79
Acknowledgement .....	80

Appendix 1 Testing Data & Result .....	81
Appendix 2 OpenCV .....	85
Reference .....	89

## Chapter 1 Introduction

Before going through the detail implementation, this chapter will first outline the motivation and the objectives our project.

### 1.1 Motivation

As people who live in modern city such as Hong Kong has a compact living space, they often entertain themselves by computers. Today's computers could fulfill lots of entertainment tasks, and the most important one is the computer games.



Many people fall in love with computer games, and that is the reason why there is always a great demand in computer games market. The most recent games contained complicated elements including 3Ds graphics, storytelling, dramatic performance and even artificial intelligence. While these games seem to be welcomed by many advanced game players, it does not represent the whole picture of computer games market.

There is always an alternative, since the tastes are varying from people to people. Despite the well-known computer games that are advertised everywhere, in reality, the simple games, still holds their position. Moreover, the simplicities of these games provided them high flexibility to implement in hardware with relative low processing power, which showed its unlimited potential in mobile entertainment and online capability. We believed that, together with the featuring of personal element and implementation of artificial intelligence, simple games can give more fun to the players.

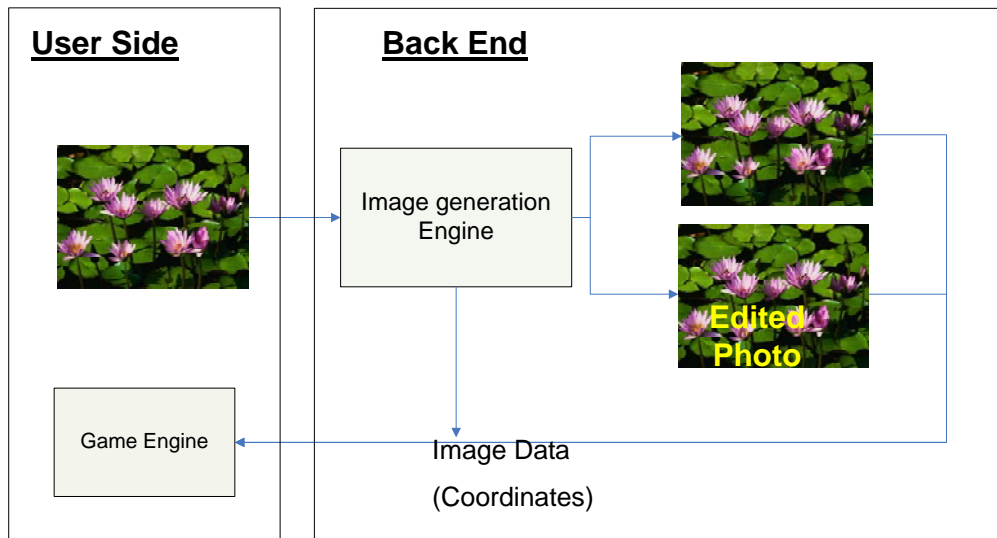
## 1.2 Project Objectives

PhotoHunt is a classic but evergreen spot-the-difference game welcomed by all range of ages. However, as the differences in the gamed photo have to be created manually, the scalability of PhotoHunt is sometime limited by manpower. In addition, one cannot play the game if the photo is edited by himself, as he simply knows the answers.

Our project is to eliminate this limitation by creating the automatic PhotoHunt generating engine. When the photo is sent to the engine, the photo is edited with several differences as the original image. Then, the two photographs are sent to the Game Engine for preparing the game application.



The following is the flow chart showing the whole process.



This Image generation Engine was implemented with various types of image processing technique with the help of OpenCV. In the following chapter, we will first give a brief introduction of Image processing, followed by our work in this semester.

## Chapter 2 Analysis of the PhotoHunt Game

---

Prior to any implementation of our project, we conducted a research to study the background about PhotoHunt. We collected information from websites [1], [2], [3]. In this chapter, we will show the game rules of the PhotoHunt, the common technique to implement the game and some related definition.

### 2.1 What is PhotoHunt?

PhotoHunt is a popular find-the-difference game available in the game centre all over the world. Players have to look for several differences between two images in specify time. Such images will looks similar but actually differences could be observed by carefully exam the images.



Fig 2.1 Sample PhotoHunt games in website [2] [3]

## 2.2 Basic approach to generate image

As our engine is going to mimic the human behavior on changing the image, we have to first investigate how nowadays PhotoHunt images are generated by man-power. The followings show our findings on the technique that are commonly used to generate image for PhotoHunt game:

1. Elimination

Elimination is the removal of object to the original picture. This is the most common technique used to create a game photo. This is done by masking object from texture of the surrounding.

2. Color modification

Color change will be sometimes applied to (component of) objects with flatten color. However, there are some constraints on applying this effect, i.e. the new color painted should be a nature color of the object. For instance, a tree should not be blue in color.

3. Cloning

Cloning is the duplication of object. Part of the image is extracted and then paste back to the background. This requires more sophisticated decision support, since not all objects can be duplicated. Also, it is difficult to tell where the copied image should be pasted.

4. Transformation

The common transformation that applied nowadays includes rotating, magnifying or diminishing objects in the image.



The different changes applied to the image retrieved from website [1]



Among the five main effects that can be applied to the object, we found that most of the changes (~70%) are produced by elimination. So we considered implemented the elimination module as our primary goal in this semester.

## 2.3 What makes good images for PhotoHunt?

After analyzing the existing PhotoHunt game, we summarized different factors and concluded the following definition of well generated Image:

### **NOT OBVIOUS YET DISCOVERABLE**

For “Not obvious”, we mean the modified parts should

- have similar color tone as the surrounding
- have similar brightness as the surrounding
- not be a large area that make up the main component of the image

For “Discoverable”, we mean the difference of images

- should be large enough for human vision
- must be comparable between the image

## 2.4 Technical Required

All the possible changes that mentioned above are now edited manually with the aid of image processing software. In order to implement the automatic generated PhotoHunt, a lot of Image processing techniques are needed. We will give more details on the technique as well as the elements of the image processing we have learnt in the next chapter. Then, we will further explain how we apply all those technique to implement our project.

## Chapter 3 Digital Image Processing

---

Digital Image processing is extremely important to our algorithm in generating the game picture. Since both of us have very limited knowledge to this field before, we spent time to learn and did numbers of testing on them in order to get familiar with it. In this chapter, we will give the image processing theory we understand followed by some testing result.

### 3.1 How does image processing apply to our project?

According to the wikipedia[4], image processing is any form of information processing for which both the input and output are images, such as photographs or frames of video. In our project, we rely heavily on the manipulation of digital image, from the retrieval of the pixels to the changes applied and also the display of the modified images. Each step requires certain technical implementation and some computer graphic theory behind.

#### 3.1.1 Smoothing

Smoothing is achieved by adding filter to the original image to make a blur effect. It enables us to mask noise of the generated image to make it appeared more realistic.

### 3.1.1.1 Median Filtering

33	32	31	33	32
32	32	32	34	33
33	34	33	35	31
34	36	35	34	31
35	33	35	35	29

This filter replaces the pixel in the area by the median pixel.

Thus, in the above case, the pixels in the area are

32, 32, 34, 34, 33, 35, 36, 35, 34

Firstly the sequence is sorted to become

32, 32, 33, 34, 34, 34, 35, 35, 36

Then the median is 34, and then 34 is used to as the intensity to replace the area.

### 3.1.1.2 Gaussian Filtering

In the Gaussian filtering, the degree of smoothing is determined by the standard deviation in the Gaussian distribution. The filter determined by the Gaussian gives weight to each pixel.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{s^2+y^2}{2\pi\sigma^2}}$$

Then, the Gaussian filter  $G(x,y)$  is applied to smooth the object:

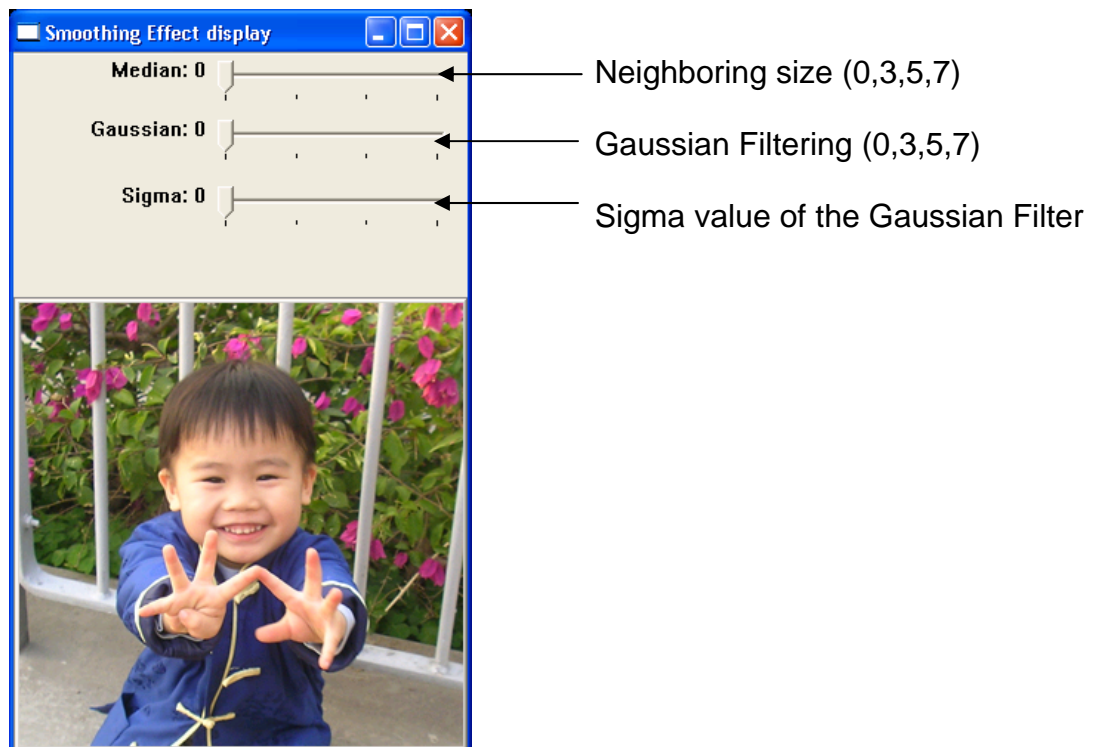
$$S(x, y) = \text{Image}(x, y) \cdot G(x, y)$$

$$\frac{1}{115}$$

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

Fig. 3.1 A sample of Gaussian filter with  $\sigma = 1.4$

We constructed a testing program to show the effect of the median filter and the Gaussian filter. The testing program is used as a reference to decide the correct smoothing approach that should be assigned in our PhotoHunt generating engine. Below is the interface and the result are shown on A1 in Appendix 1.





### 3.1.2 Segmentation

Image segmentation is the partition of a multimedia content  $R$  to a set of non-overlapping segments.

$$R = \bigcup_{i=1}^S R_i$$

such that  $R_i \cap R_j = \emptyset$  for all  $i, j \in S$  &  $i \neq j$

By separating image into group of homogenous region according to the image characteristics, we enable us to distinguish objects from the background. We have examined three commonly used approaches in segmenting an image. They are

1. Pixel-based segmentation
2. Edge-detection approach
3. Region-based approach

In the following part, we will analysis the possibility of applying these segmentation methods to our project.

### 3.1.2.1 Pixel-based Segmentation

#### (1) Histogram Thresholding

Thresholding is a straight-forward and simple method to segment an image. It divides the image to object part and background part.

Let "1" is object, "0" is background

$$c'(x, y) = \begin{cases} 0 & c(x, y) \leq T \\ 1 & \text{otherwise} \end{cases}$$

$c(x, y)$  = the local characteristic of the image at position (x,y).

$T$  is the selected Threshold

To visualize and consolidate our understanding on this thresholding segmentation, we wrote a testing program (ThresSeg.cpp). In this program, we implemented the threshold selection by the isodata algorithm. This algorithm computes and assigns a dynamic threshold to segment the image. The estimation of threshold is obtained by using an iterative approach, it is updated each iteration until  $T_k = T_{k-1}$ . The update formula is given by:

$$T_0 = 2^{D-1}$$

$$T_i = (m_{o,i-1} + m_{b,i-1}) / 2$$

where

$D$  = Color depth

$$m_{o,i} = \overline{c_k(x, y)} \quad \text{for all } c_k'(x, y) = 1$$

(i.e. the mean of the pixel label as object in the i-th iteration)

$$m_{b,i} = \overline{c_k(x, y)} \text{ for all } c_k'(x, y) = 0$$

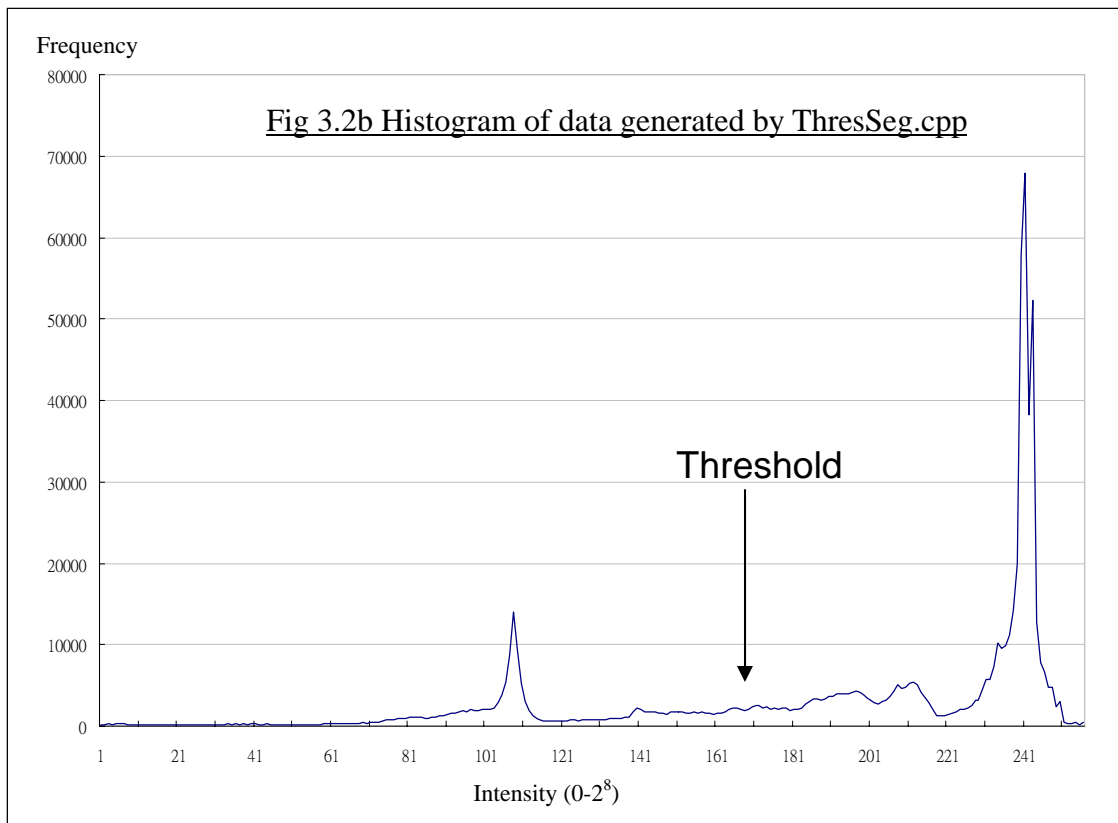
(i.e. the mean of the pixel label as background in the i-th iteration)



Fig 3.1a Gray-scale testing image for the Thresholding Segmentation (Dimension: 450\*541; Color depth= 8)

Our program first counted the number of pixel in each color level, and then it calculated the Threshold from the result. In order to visualize the process, we printed out the data\* and plotted fig 3.2b.

\*The data is shown on Table A2 in Appendix 1.



The second part of ThresSeg.cpp is to calculate the threshold, the intermediate results of the program are shown in Fig. 3.3.

```

ThresholdSeg.exe
Threshold 0 = 128
Threshold 1 = 156
Threshold 2 = 165
Threshold 3 = 167
Threshold 4 = 168
Threshold 5 = 169
Final Threshold = 169

```

Fig 3.3 Intermediate Result

Finally, the resulted Threshold (=169) is used to generate the segmented photo. All pixels above 169 are set to dark (object) while others are set to white (background). Fig3.4 shows the result photo



Fig 3.4 Image segmented by Threshold Segmentation

We have done 2 more testing to observe the result for how the algorithm works on realistic image; they are shown in A3 in Appendix 1.

#### **Analysis :**

The thresholding segmentation is simple and easy to implement. However, the algorithm success only in the simplest case where object and background leads to an outstanding gray level separation.

In our project, users are free to upload any picture with any magnitude in gray level separation. So, instead of using thresholding for our segmentation, we should seek for others segmentation methods that can be applied to more general problems.

### 3.1.2.2 Edge Detection

#### (1) Canny Edge Detection

Canny edge detection algorithm is proposed by John. Canny in [5], it is known to many due to its low error rate, well localized edge points and single edge response. It works as a 4-stages process:

##### 1. Image Smoothing :

The image data is smoothed by Gaussian filter

$$S(x, y) = G(x, y) * c(x, y) \quad S(x, y) = \text{smoothed image}$$

##### 2. Differentiation

The smoothed image is differentiated with respect to the x and y direction.

$$H(x, y) \approx (S(x, y+1) - S(x, y) + S(x+1, y+1) - S(x+1, y)) / 2$$

$$V(x, y) \approx (S(x+1, y) - S(x, y) + S(x+1, y+1) - S(x, y+1)) / 2$$

To get the magnitude matrix and direction M,  $\theta$

$$|M| = |H| + |V|$$

$$\theta(x, y) = \begin{cases} 0 & H(x, y) = 0 \text{ and } V(x, y) = 0 \\ \frac{\pi}{2} & H(x, y) = 0 \text{ and } V(x, y) \neq 0 \\ \tan^{-1}(V(x, y)/H(x, y)) & \text{otherwise} \end{cases}$$

##### 3. Non-maximum Suppression

After the rate of intensity change at each point in the image is known, edge is placed at the points of maximum. After the non-maximum suppression, the value at the local maxima point is preserved while all other are changed to 0.

#### 4. Hysteresis

In other algorithm using a single threshold, sometimes the edge line may appear broken when the edge value fluctuated above and below this value. In Canny's algorithm, this problem of streaking is eliminated by continuing the tracking until the Threshold falls behind the lower second threshold.



Fig3.5a shows image produced by Canny Edge Detection



Fig3.5b shows image produced by Canny Edge Detection

Conclusion:

Canny edge detection works well in more general problems. Although it does not provide the area, the position or the intensity of a particular segment of our primary interest, it can still be helpful in the process of detecting a plain area or background.



### **3.1.2.3 Region-based Approaches**

A region-based method partition images into connected regions by grouping neighboring pixels. Adjacent pixels are grouped to regions if they have similar intensity levels. Regions are then merged together based on their homogeneity or sharpness of region boundaries. Among the various region-based approaches including thresholding, clustering, region growing, we chose the hierarchical and efficient pyramid-based approaches to implement our segmentation process. The following chapter will be dedicated for this approach and will have detail explanation on it.

## Chapter 4 Pyramid Segmentation

---

Pyramid Segmentation is a region-based method to separate image into groups. It involves the process of building up the “Image pyramid” in a bottom-up approach and then applies the top-down solution refinement. This data representation in “pyramid” have been proven to be useful in many cases, and shown simultaneous and rapid results in different image analysis tasks [7].

In this chapter, we will show an overview of the Pyramid Segmentation base on the Burt’s algorithm[8] which is applied to the image in our PhotoHunt generating engine. This pyramid segmentation algorithm includes the following steps:

1. Computation of the Gaussian pyramid
2. Segmentation by pyramid-linking and Averaging of linked pixels.

The steps 2 and 3 are repeated iteratively until we reach a stable segmentation result.

### 4.1 Image Pyramid

Image pyramid is constructed by multiple copies of the same image of different scale. The upper levels of the pyramid are computed iteratively from the base of the pyramid. The resolutions of the images decrease when they are computed upward along the pyramid.

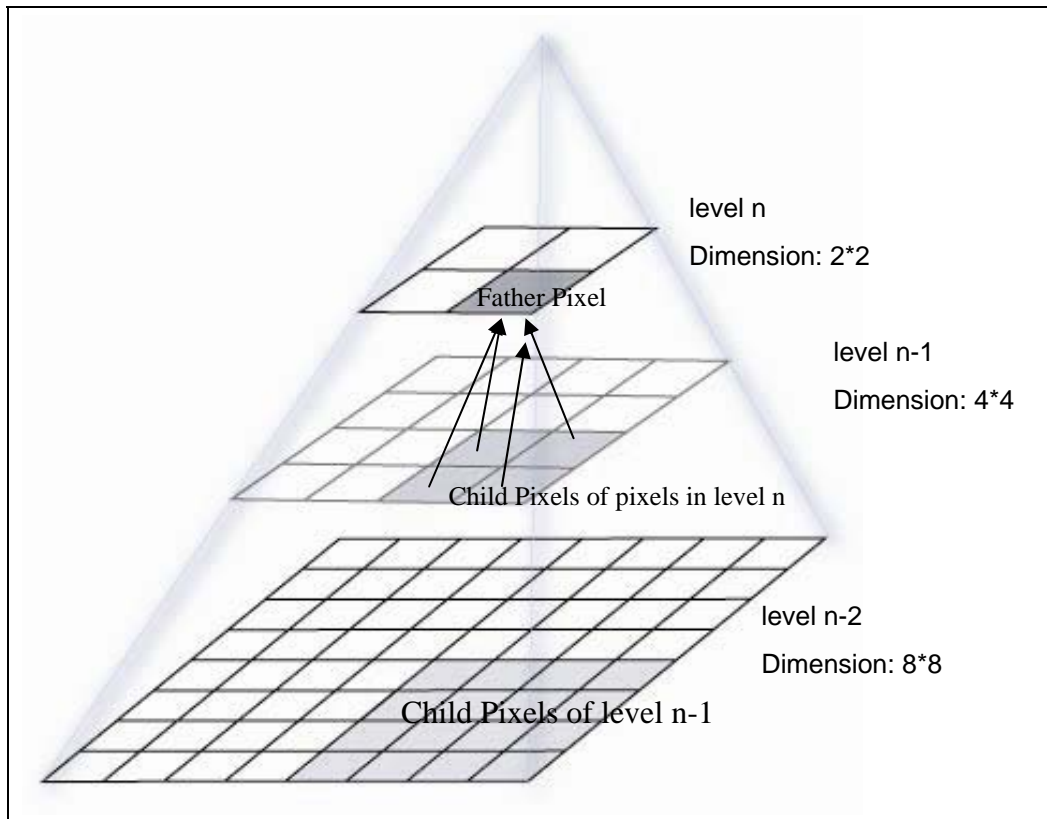


Fig 3.1 This diagram shows the graphical representation of the Image Pyramid.

## 4.2 Generation of the Gaussian pyramid

The first step of the Burt's Algorithm is to generate the Gaussian Pyramid. A Gaussian pyramid is one that built by applying the Gaussian filter. For simplicity, the image will only be reduced in 1-dimension in this illustration.

Prior to the illustration, the followings give the mathematic representation that will be used in the later section:

$l$  : level of pyramid

$g_l(i)$  : the value of the  $i$ -th element on level  $l$  of the Gaussian Pyramid

Assume  $w$  be the weight of the Gaussian filter,

$$\hat{w} = [\hat{w}(-2) \quad \hat{w}(-1) \quad \hat{w}(0) \quad \hat{w}(1) \quad \hat{w}(2)]$$

According to the clear explanation on the paper [8], the generating kernel should have the below properties:

a) Separable (when it is in 2 dimension0)

$$\hat{w}(x,y) = \hat{w}(x) \hat{w}(y)$$

b) Normalized

$$\sum_m \hat{w}(m) = 1$$

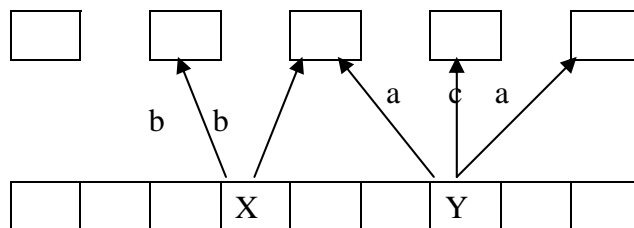
c) Symmetric

$$\hat{w}(-m) = \hat{w}(m) \text{ for } m=0 \text{ to } 2$$

d) Equal contribution

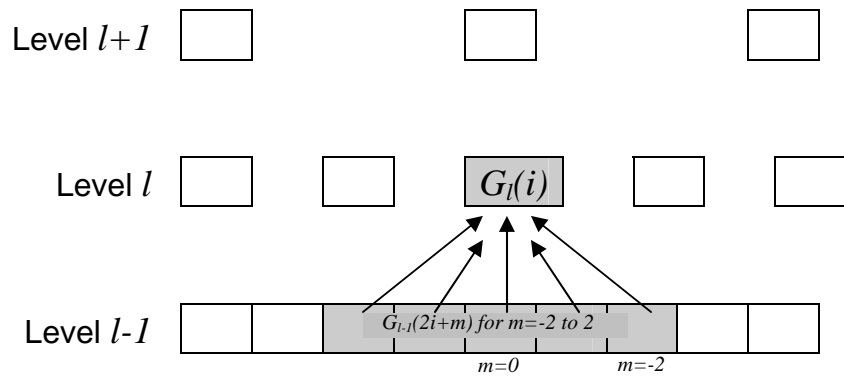
Each pixel should have equal contribution to the father pixel in the upper level.

For example,



In the above case, since pixel X and pixel Y should contribute equally to the father pixel, it can be deduced that

$$2b = 2a + c$$



The 5 colored-pixel contributed to the colored pixel at its higher level  $G_l(i)$  with weight  $w$ , which give the following equation:

$$\begin{aligned}
 g_l(i) &= w(-2)g_{l-1}(2i+m) + w(-1)g_{l-1}(2i+m) + w(0)g_{l-1}(2i+m) \\
 &+ w(1)g_{l-1}(2i+m) + w(2)g_{l-1}(2i+m) \\
 &= \sum_m w(m)G_{l-1}(2i+m)
 \end{aligned}$$

As the Gaussian filter is symmetric and separable, we can apply the similar computation again to get the 2-dimensions reduced image.

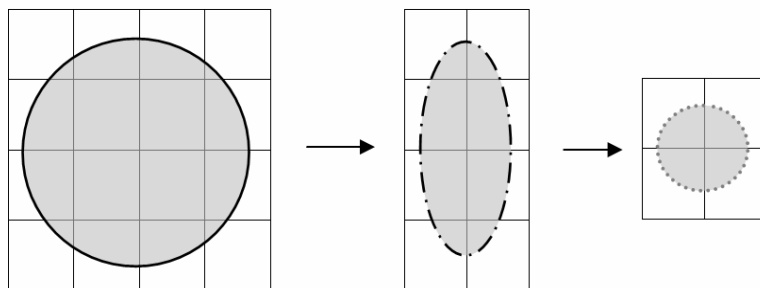


Fig 3.2 Applying reduction twice to get the father level

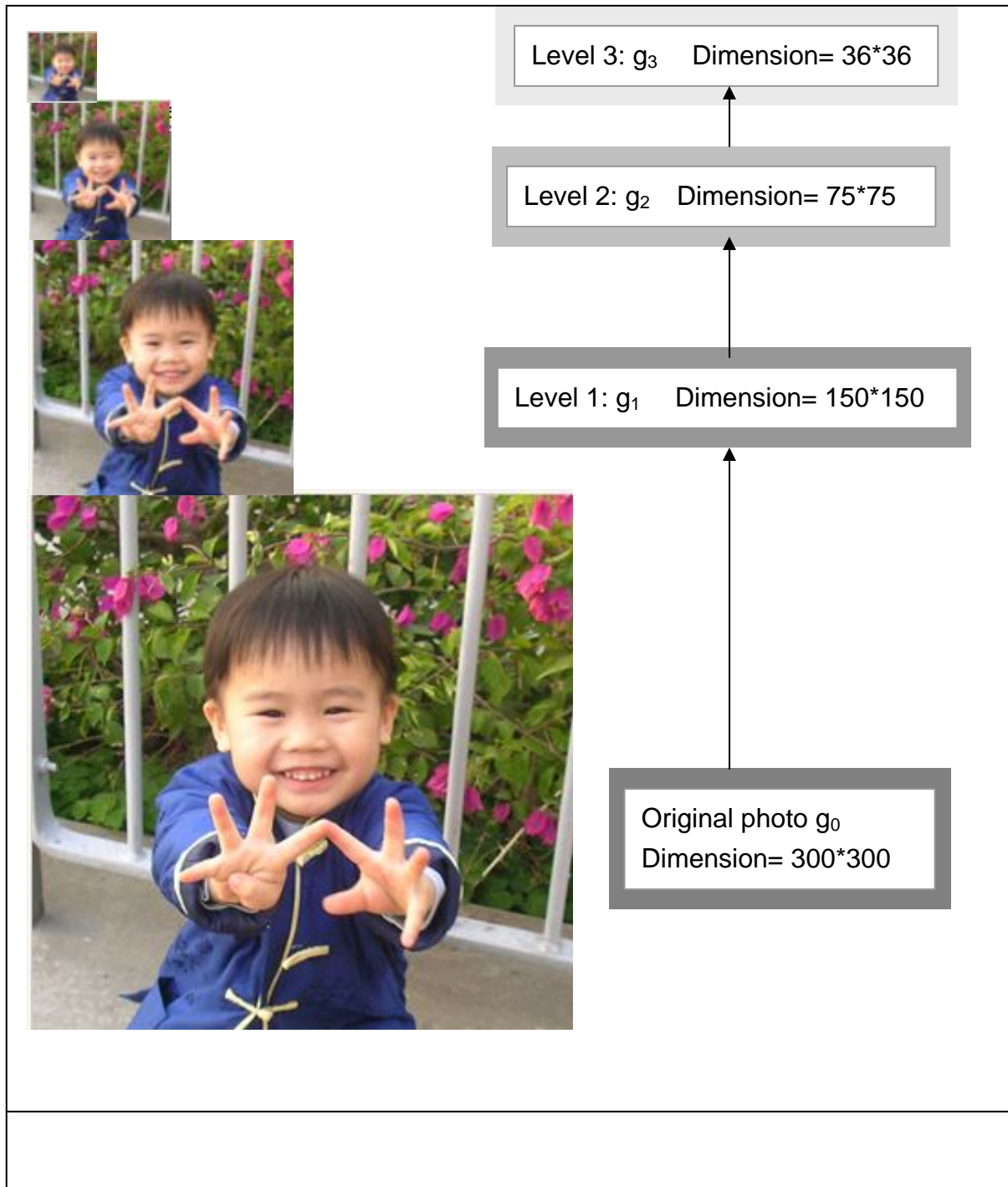
By applying this down sampling twice, the weight will be come 5\*5. the following equation proposed by Burt in the paper [9] is obtained:

$$g_l(i, j) = \sum_m \sum_n w(m, n) g_{l-1}(2i + m, 2j + n)$$

We wrote a testing program to test the process of the generating Gaussian pyramid. In the program, the image is down sampled three times with 5x5 Gaussian kernels shown below. Fig 3.3 shows the result of the program. (Note that the Gaussian pyramid is shown here for reference only, and would not be displayed in normal segmentation process)

$\frac{1}{331}$	1	4	7	4	1
	4	20	33	20	4
	7	33	55	33	7
	4	20	33	20	4
	1	4	7	4	1

Fig 3.3a 5x5 Gaussian Matrix



**Fig 3.3** Generation of Gaussian Pyramid

### 4.3 Segmentation by pyramid-linking

After the Gaussian Pyramid is built, the following step is applied in order to divide the segment:

Here are some notations that will be used in the algorithm:

- $area_i(x, y, l)$ : the area of pixel (x,y) at level l in the i-th iteration
- $c_i(x, y, l)$ : the local characteristic of pixel (x,y) at level l in the i-th iteration
- $s_i(x, y, l)$ : the segment property (the average local property) of the segment that the pixel (x,y) at level l belong to
- $f_i(x, y, l)$ : the father node of pixel (x,y) at level l in the i-th iteration
- L: Height of Pyramid

Then the following process is applied:

1. Assignment of all  $f_i(x, y, l)$  for  $i=0$  to L,

$f_i(x, y, l)$  stores the father node of pixel (x,y) in level l. While there are four father nodes for each pixel,  $f_i(x, y, l)$  should store the one with the lowest difference in local characteristics between the node and its father.

2. Initialize the local characteristic in the base level at the current iteration

$$c_i(x, y, 0) = c_0(x, y, 0)$$

$$area_i(x, y, 0) = 1$$

3. Compute the area and local characteristic of the upper level, for  $i=0$  to L

$$area_i(x, y, l) = \sum a_i(x, y, l-1)$$

$$c_i(x, y, l) = \frac{\sum (area_i(x', y', l-1) \cdot c(x', y', l-1))}{area_i(x, y, l)} \quad \text{for } (x', y') \text{ is the child pixels of } (x, y)$$

4. Compute the segment property

$$s_i(x, y, L) = c_i(x, y, L)$$

$$s_i(x, y, l) = c_i(x'', y'', l+1)$$

5. Repeat 2-4 until the segment property do not change



## 4.4 Gaussian Pyramid Interpolation

As a by-product, we used the generated Gaussian pyramid in step1, to generate Laplacian Pyramid.

In step 1 the Gaussian Pyramid is generated by down sampling of the image. In order to get the Laplacian Pyramid, we will then need to “expand” the Gaussian pyramid. According to the algorithm in paper [8],

Let  $g_{l,n}$  be the result of expanding  $g_l$   $n$  times,

i.e.  $g_{l,0} = g_l$

$$g_{l,n} = \text{EXPEND}(g_{l,n-1})$$

where EXPEND means the followings :

$$g_{l,n}(i, j) = 4 \sum_m \sum_n w(m, n) g_{l-1}\left(\frac{i-m}{2}, \frac{j-n}{2}\right)$$

Only for terms which with  $(i-m)$  and  $(j,n)$  divisible by 2

As  $g_{l,0} = g_l$  and  $g_l$  is the top of the Gaussian pyramid, by ”EXPEND”ing  $g_l$   $l$  times, we can get  $g_{l,l}$  which have the same size as the original photo, yet  $g_{l,l}$  is blurred.

## 4.5 Generation of Laplacian Pyramid

Laplacian is sequence of image that records the difference between adjacent levels in the Gaussian Pyramid. Due to this property, most of the element in the Laplacian Pyramid is zero. The Laplacian pyramid  $L$  is given by

$$L_l = g_l - \text{EXPAND}(g_{l+1})$$

where  $0 \leq l < \text{Height of Gaussian Pyramid}$

For testing purpose, we constructed a program in C++ to produce the generation of the Laplacian pyramid, the results are shown in the next page.

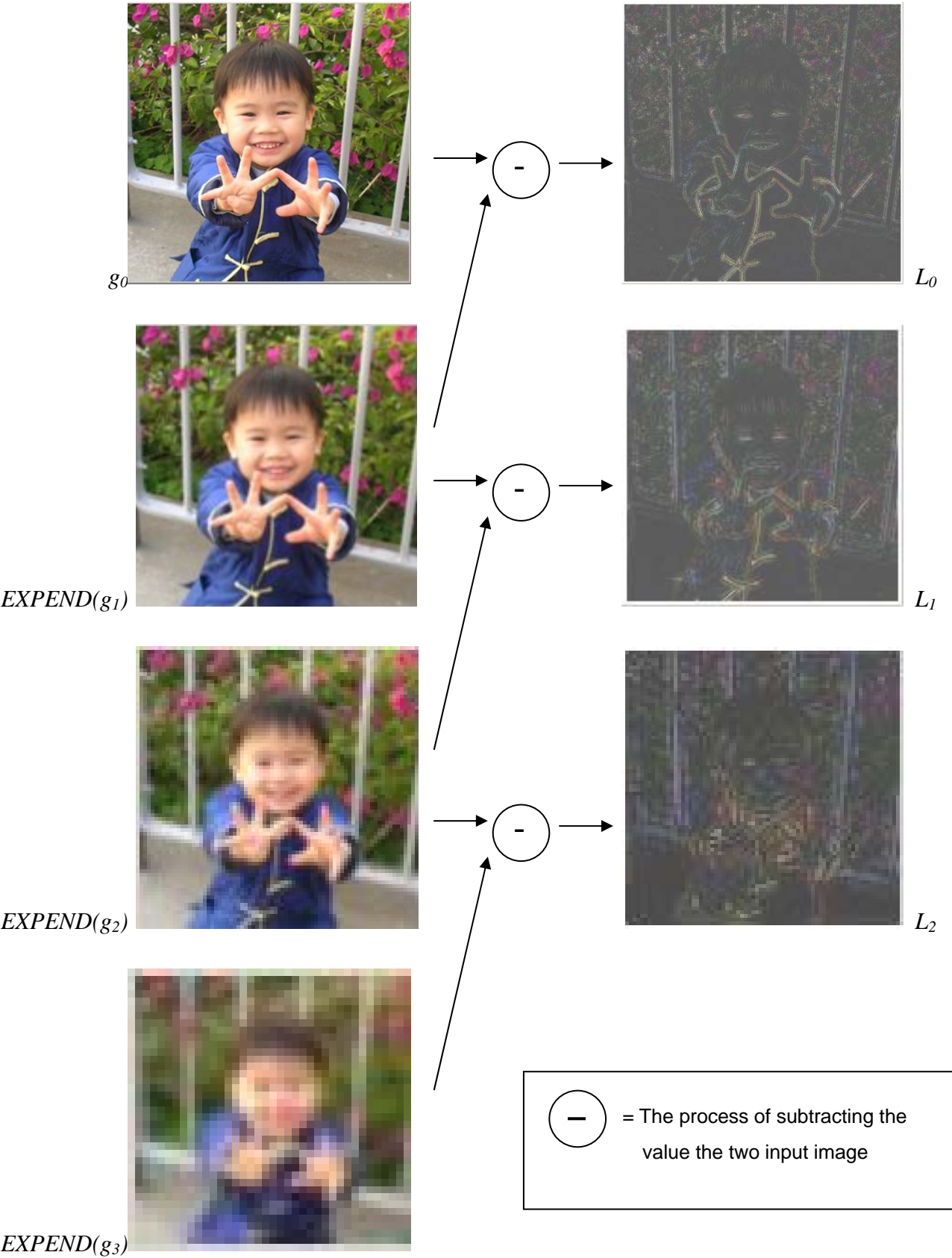


Fig 3.4 shows the Generation of Laplacian Pyramid with the aid of OpenCV

## 4.6 Gaussian and Laplacian Pyramid in OpenCV

The pyramid segmentation implementation in the OpenCV shown in website [9] is contributed by Mr. Alexander Pleskov. It supports the generation and reconstruction of Gaussian and Laplacian image pyramid. Also, there is a function for pyramidal color segmentation.

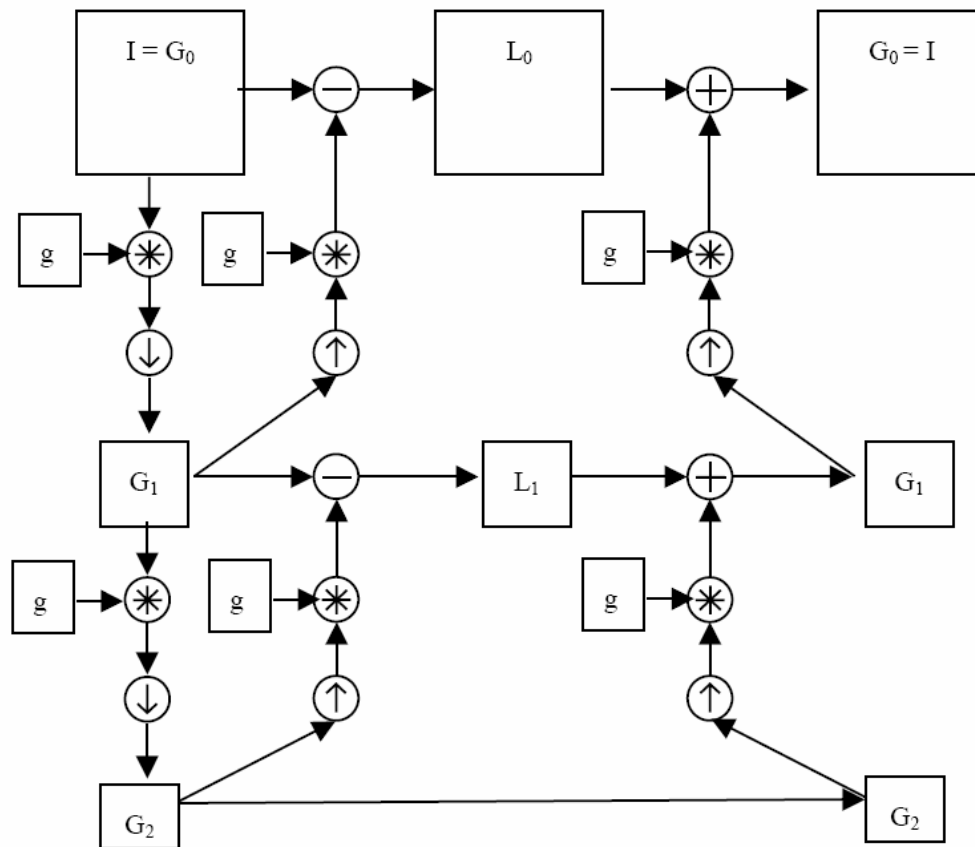


Fig 3.2 shows a Three-level Gaussian and Laplacian Pyramid

## 4.7 Pyramid Segmentation in OpenCV

Besides the source image and destination location, there are two more parameters, namely Threshold1 and Threshold2, which have to be provided prior to the usage of pyramid Segmentation in the OpenCV .

### 4.7.1 The First Parameter - Threshold 1

This threshold gives the lower bound to the difference of color between two segmented regions. To further explain the meaning of 'link', we study the Reference Manual of OpenCV [10]

The parameter Threshold1,  $T_1$ , determined whether link should be set between the father pixel in level  $n$  and another pixel in level  $n-1$ . The link will be established if the RGB value of the father pixel  $(x', y')$  and that of the child pixel  $(x, y)$  have a difference higher than the threshold, i.e.

$$d(c(x, y, n), c(x', y', n-1)) < T_1$$

where  $c(x, y, n) \in \mathfrak{R}^3$  = the local property of pixel  $(i,j)$  at level  $n$

and the Euclidean metric,

$$d(A, B) = 0.3 \cdot (r_A - r_B) + 0.59 \cdot (g_A - g_B) + 0.11 \cdot (b_A - b_B)$$

$r_X$ = Value of red channel at vector  $X$

$g_X$ = Value of green channel at vector  $X$

$b_X$ = Value of blue channel at vector  $X$

After updating the links, if the number of segment is less than the level  $L$  that specified, the local property of pixels on level  $L$  will be clustered to obtain the desired number of segments. (The  $T_2$  is the threshold for the segment clustering which will be given more detail in the next sub-section.) The average local property value of the grouped nodes or segment will be computed with their weight of their areas, and then assigns the each of the members.

#### **4.7.2 The Second Parameter- Threshold 2**



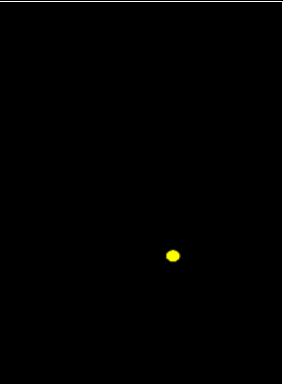


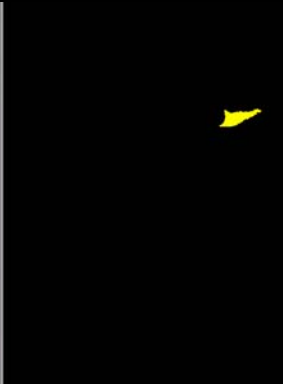

The second parameter restricted the variance of the value of pixel within the same region in the adjacent level. Any segment  $A$  and  $B$  belong to the same cluster if  $d(A,B) < T_2$ .

#### **4.7.3 Estimation of the Parameter**

The paper [11] suggested that the  $T_1$  is less significant than  $T_2$  to the segmentation especially the number of segments. We test different values for the two threshold and the observation shows that, generally, ranging from 120 to 170 for  $T_1$  and 50 to 70 for  $T_2$  give a better segmentation.

### 4.7.4 Segmentation Result

The following shows some selected segment of a picture by applying pyramid segmentation.

		Original Image	
			
Segment 1: the nose of the bear		Segment 2: Hair of the girl	
			
Segment 3: the pigtail of the girl		Segment 3: collar of the girl	

## Chapter 5 Image Generation Engine

---

Our project is supported by two main engines to complete the PhotoHunt game generation task, they are namely Image Generation Engine and Game Engine.

The users' input images are first processed by the image generation engine before sending to the Game Engine. This image generation engine is the core component of our project which serves two purposes:

1. Generate the game Image for PhotoHunt
2. Output the data of the generated image, the co-ordinates of the modified points

In this chapter, we will give detail description to the image generation engine.

Then, the implementation of game engine will be further explained in Chapter 8.

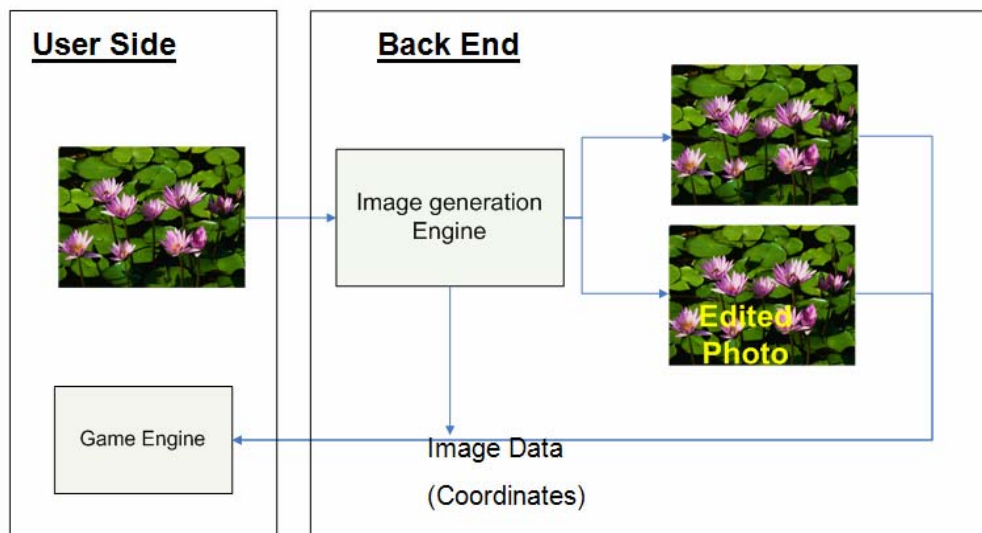


Fig 5.1 Flow Diagram of Automatic PhotoHunt generation



## 5.1 Processing Flow

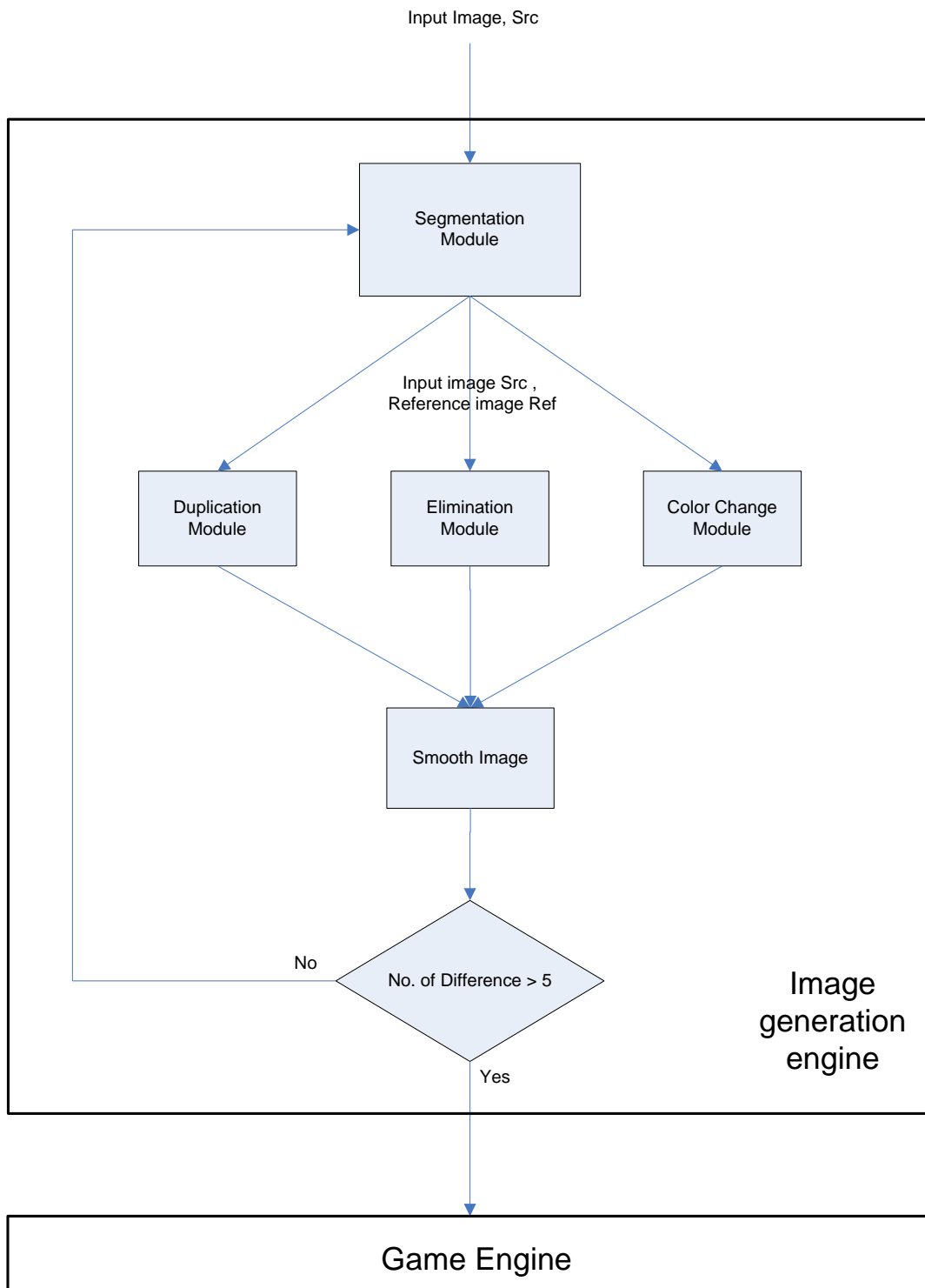


Fig. 5.1 Flow Diagram for the Image Generation Engine

## **Processing Flow**

Our engine is implemented to mimic the editing process of human-being. When a person gets an image, he would first have an overview to decide where/which object the effect should be applied to. This analysis and decision making process, in our engine, is carried out by the segmentation module. This module will break down the image into segments and then select an appropriate one according to some pre-set rules. Then the information of the selected segment including the coordinates is passed to one of the three effect creating modules. These modules are actually stimulating the human to edit the image which includes implementation of the elimination, color changing and duplication of the selected component. Each of the modules has its embedded algorithms to apply its dedicated effect. Then, the system will search again until the number of difference reach the desired value (default as 5). After all the changes have been assigned to the image, the edited image will then send out together with the answer to the game engine.

In the following sub-section we will go over the modules implementation in more detail.

## 5.2 Segmentation module

The main purpose of the segmentation module is to detect and extract segment from the input image. Basically, the module included three phases:

### 1. Pyramid Segmentation

In this phase, the input image is segmented by the pyramid segmentation algorithm described in Chapter 4. Then, one of the segments is randomly picked out.

### 2. Constraint Checking

In order to achieve the “NOT OBVIOUS YET DISCOVERABLE” principle we defined in Chapter 2, the selected segment are bounded to follow the below constraints:

- For “Not obvious”:  
Area of Segment < Area of Input Image/500
- For “Discoverable”:  
Area of Segment > Area of Input Image/10000

### 3. Reference image building

Based on the segments selected, this module is used to create a bit array which has a same dimension of the input, yet of different sizes. (Since the input image are in RGB channel while the bit array is only 0/1)

The value of the array  $R$  is defined by:

$$R(x, y) = \begin{cases} 0 & (x, y) \notin S \\ 1 & (x, y) \in S \end{cases}$$

where  $R$  is the reference image and  $S$  is the segment

The bit array is built so as to enhance the computation time in the later module by using bit operation. Also, the coordinates of the segment is obtained when we built the array  $R$ .

### **5.3 Elimination Module**

We observed that nearly 70% of changes made in PhotoHunt game are by Elimination. Thus, we put our main focus on the elimination algorithm; we have designed 4 algorithms which will be introduced in the next chapter.

Moreover, we discovered that this elimination algorithm can somehow create object to the image. This is done by applying a little change to our flow by employing a new concept. Instead of regarding the two images as original and modified one, the two images are viewed as Picture1 and Picture2. Now, elimination is applied to both of the images, an elimination in Picture 1 is seems to be a new object in Picture 2.

### **5.4 Color Change Module**

The main function of this module is to perform the color change effect. While we are still looking for a general rule to apply the color change, the engine now only supports change to the segments of a few specified colors.

Segment dominate color	Operation apply
Red	Swap red and blue channel
Green	Swap red and green channel
Blue	Swap green and blue channel

Dominate color is defined by the  $DChannel > 180$  and  $otherChannel < 50$

Dominate color segment: segment of more than 75% dominate color pixel

## 5.5 Object Duplication Modules

This module applies the object duplication effect to the image. At the current status, this module is only implemented at the semi-Automatic testing platform in which users have to input the transformation parameters. With a preliminary idea on how to complete this module, we are looking forward to implementing this module in the coming semester.

## 5.6 Smoothing Image

This module is not initially designed in the starting phase of our project. It is added until the generation of the first few images when we observed that the edge of the modified segment are affected by noises and caused distortion. In order to overcome these problems, we added a Gaussian Filter (Neighbor size=3, stand deviation=1) to smoothen the modified segment so as to make the generated image look more realistic.

## Chapter 6 Elimination Algorithms

---

In this Chapter, we will introduce several algorithms that we have designed and applied to eliminate the segment which is selected by the segmentation module. To achieve our aim that the segment should look “disappeared in the image”, we have developed four approaches, each have their own characteristics and optimal environment to operate.

### 6.1 Direct Copy Algorithm

This algorithm is the simplest one. Every line of the segmented region copies the color from the upper line and such process is ran from top to bottom and from left to right.

$$C(x, y) = C(x, y-1) \quad \text{for} \quad C(x, y) \in S_i$$

Where  $S_i$  is segmented region of the image for  $I = 0, 1, 2, \dots$

#### 6.1.1 Testing Result

We try this algorithm on both realistic images such as a photo taken by digital camera and unrealistic images, for example, a cartoon picture. The result shows the algorithm works well only for image segments which are surrounded by simple and purely-colored regions (Fig 6.3 and Fig. 6.4).. For the realistic picture, the results are not that acceptable since the color of the top level of the segmented region could be quite complicated (Fig. 6.1 and Fig. 6.2). In conclusion, although this algorithm is simple, it is not a generic solution to our elimination process.



Figure 6.1



Figure 6.2



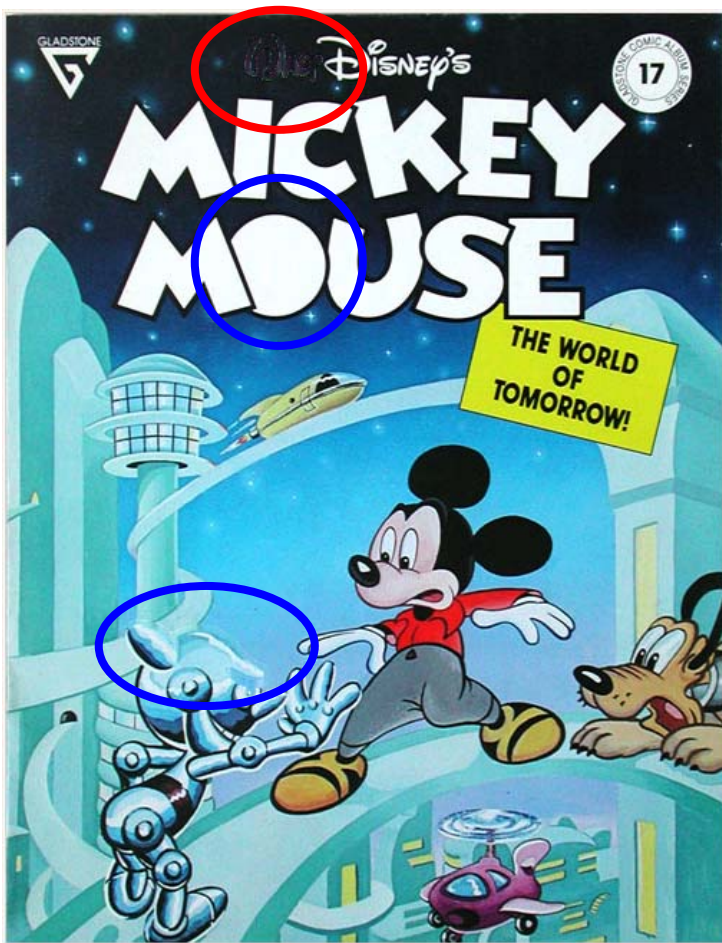


Fig. 6.3

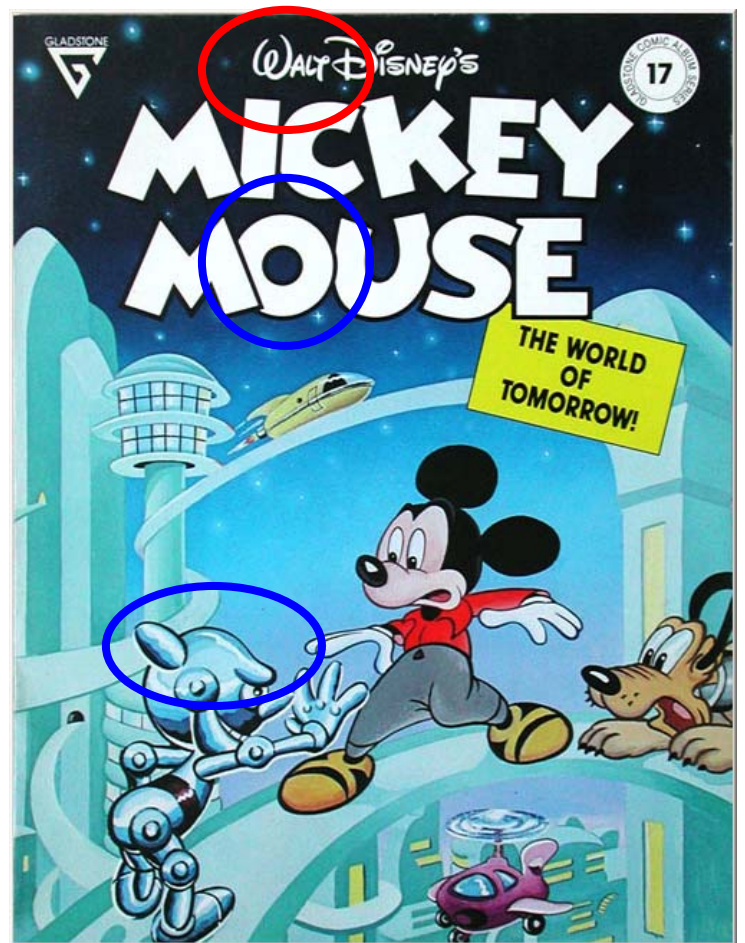


Fig. 6.4

## 6.2 Horizontal Gradient Algorithm

What we see in the previous algorithm is that simply copying the color could result in a sharp change in the edges of the segmented region. In order to reduce this oddness, we come up with an idea to smooth the change from one side to another side of the segmented area.



For every pixel of the eliminated region, it has two horizontal boundaries. The Fig. 6.5 shows the pixel N which has two horizontal boundaries P and Q.

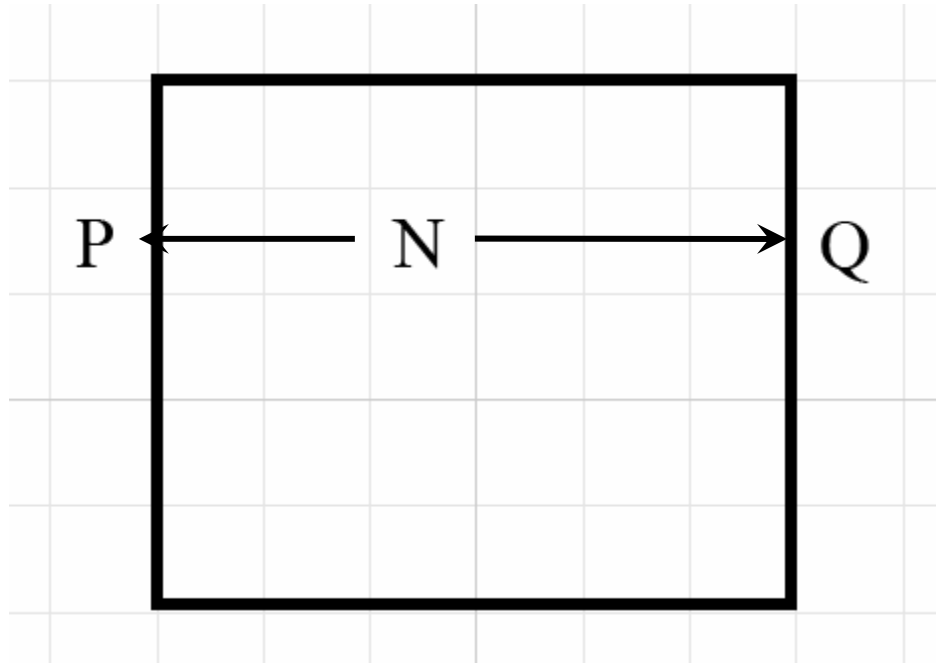


Fig. 6.5

We also know the distances between the pixel and two boundaries in terms of number of pixels. In the above figure, the distance is 2 pixels from N to P and 3 from N to Q. By weighting the colors of boundaries together with the distance, we derived the color of N by the following:

$$C(N) = C(P) \cdot \left(\frac{d_{NQ}}{d_{PQ}}\right) + C(Q) \cdot \left(\frac{d_{NP}}{d_{PQ}}\right)$$

where  $C(M)$  is the color vector of pixel  $M$  and  $d_{XY}$  is the distance between  $X$  and  $Y$

Fig. 6.6 shows that an example of the filled color in the horizontal line where N located to if the color of P and Q are black and white respectively.



Fig. 6.6

### 6.2.1 Test Result

This algorithm works even worse than the Direct Copy algorithm in most of the cases as shown on Fig. 6.7 and Fig. 6.8.

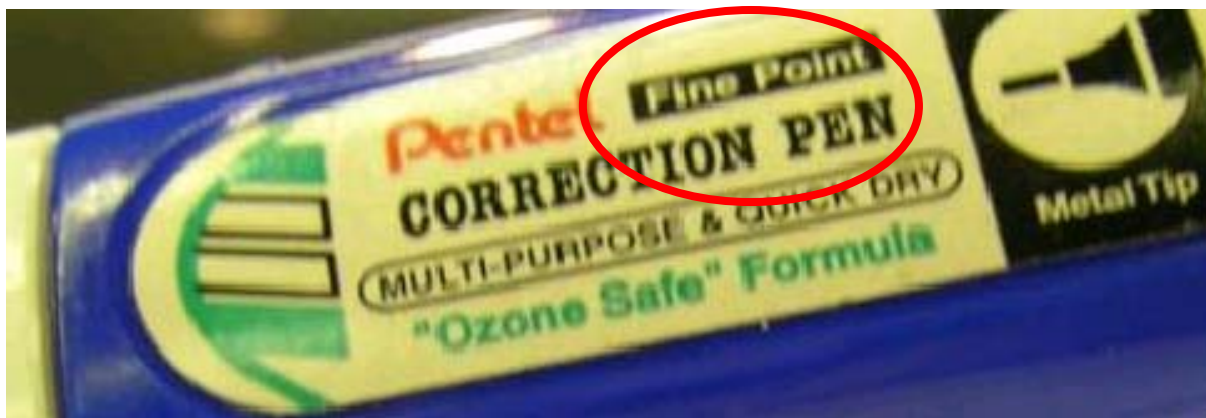


Fig. 6.7

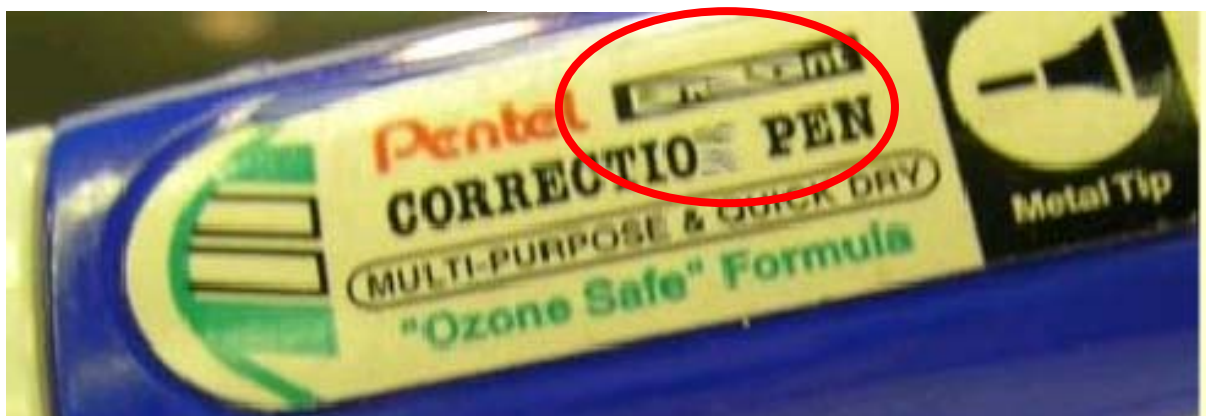
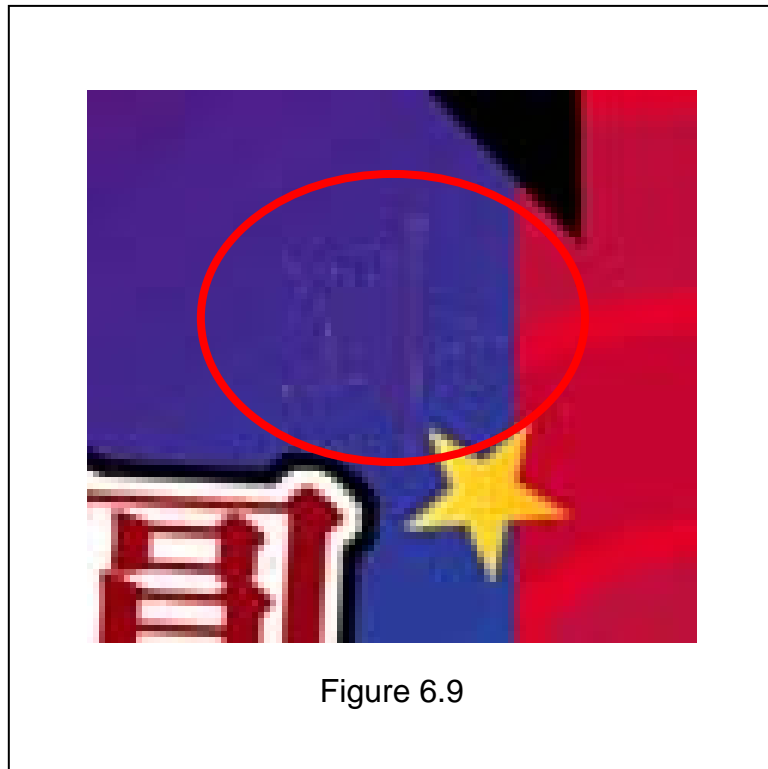


Fig. 6.8

This failure of the algorithm is mainly due to the exaggeration of noises that produced in the segmentation procedure.

The Fig 6.9 shows the noise at the boundaries of image. (The picture is produced by direct copy algorithm to show the effect)



We try to reduce this error effect at the boundary by offsetting the segment. However, the results are still not acceptable. Moreover, the segment may not be located at a monotonic-color background. If the segment is on a background with large difference color like the case shown in Fig. 6.10, color injection may occur. Realizing this feet of clay, we concluded that this algorithm could not be applicable unless some more constraints are provided.



Fig. 6.10

Besides, this algorithm ignores the surrounding texture while the calculation would generate some odd colors. As the results, we decide to abundant the Horizontal Gradient Algorithm.

### 6.3 Nearest Boundary Algorithm

In order to determine correct color to be filled and solve the overlapping problem which shown on Fig. 6.10 we designed another method as titled.

In the nearest boundary algorithm, four pixels are considered. These pixels are located at the intersections of boundary and the horizontal line or vertical line the draw from the current pixel. The following figure illustrates how the pixels are found (Fig. 6.11)

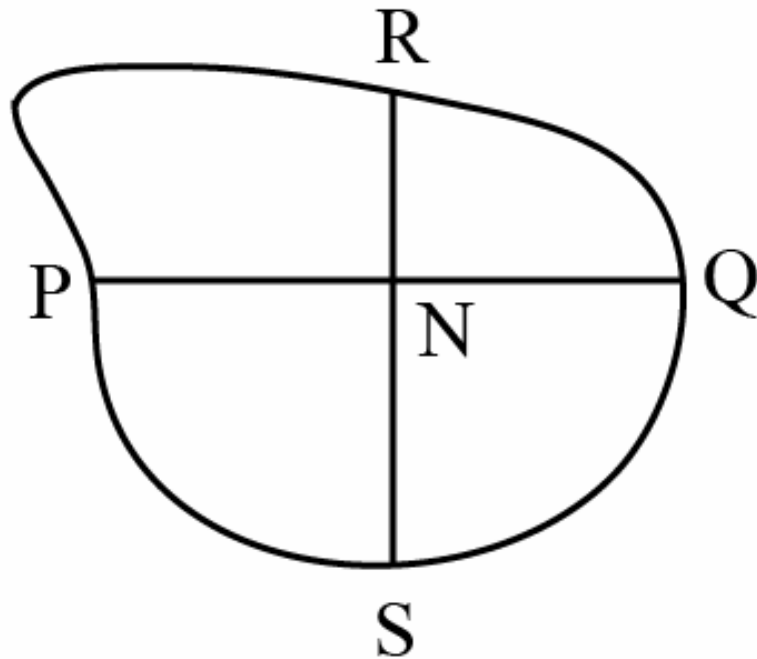


Fig. 6.11

Firstly, a horizontal line and a vertical line will draw across N. P, Q, R, S are the points of intersection between these two lines and the boundary. The color of N is then determined by the distance from N to those four pixels. The pixel with shortest distance to N, which is R in this case, would be used to replace the color of N.

$$C(N) = \begin{cases} C(P) & d_{PN} = d_{\min} \\ C(Q) & d_{QN} = d_{\min} \\ C(R) & d_{RN} = d_{\min} \\ C(S) & d_{SN} = d_{\min} \end{cases}$$

Where  $d_{\min} = \min(d_{PN}, d_{QN}, d_{RN}, d_{SN})$

### 6.3.1 Testing Result

The result is similar to the one from Direct Copy algorithm. As we can see, in Fig. 6.12 and Fig. 6.13, some results are closed to the man-made image, but some are unacceptable.

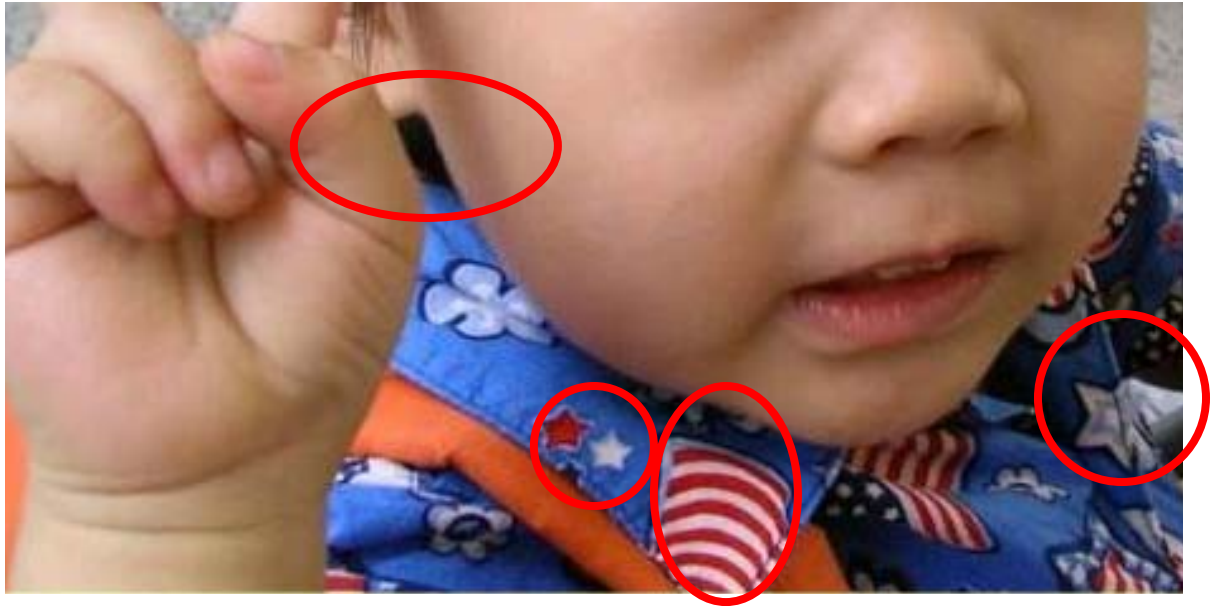


Fig. 6.12



Fig. 6.13



Fig. 6.14



Fig. 6.15

It is obvious that this algorithm has its strength in elimination. For instance, in Fig. 6.12 and Fig. 6.13, the eliminated region could even simulate the texture of the surrounding areas. However, it still needs improvement to increase its generality.

## 6.4 Enhanced Nearest Boundary Algorithm

This is the next version of the previous algorithm. Similarly to the Nearest Boundary Algorithm, it first finds the boundary pixels. Then the similarity of P and Q would compare to the similarity of R and S.

Once the comparison is done, the most similar group would be used to estimate the color. The rest are the same as Nearest Boundary Algorithm.

The following notation shows the process of the algorithm :

$$C(N) = \begin{cases} C(P) & \text{if } d_{PN} = d_{\min} \text{ and } S(P,Q) > S(R,S) \\ C(Q) & \text{if } d_{QN} = d_{\min} \text{ and } S(P,Q) > S(R,S) \\ C(R) & \text{if } d_{RN} = d_{\min} \text{ and } S(R,S) > S(P,Q) \\ C(S) & \text{if } d_{SN} = d_{\min} \text{ and } S(R,S) > S(P,Q) \end{cases}$$

Where  $d_{\min} = \min(d_{PN}, d_{QN}, d_{RN}, d_{SN})$

$S(A,B)$  is Euclidean distance between A and B

### 6.4.1 Testing Result

The rate of successfulness is increased. It works quite well in many cases, including the realistic photos or unrealistic pictures. However, some elimination still needs improvement.





Fig. 6.16



Fig. 6.17



## 6.5 Conclusion

Among all the four algorithms, the Enhanced Nearest Boundary algorithm generally gives more adequate result over the others. However, it is not applicable under all circumstance or at least in most of cases.

The latest algorithm has several limitations. Only considering four surrounding pixels might not be sufficient to determine the color to be filled. Also, the boundary pixel may not be a good choice in some cases since the segmentation algorithm might leave boundary on objects.

To conclude, further improvement is needed in order to achieve the automatic PhotoHunt generation engine that are comparable with human behavior.

## Chapter 7 Semi-Automatic Generation Program

### 7.1 Introduction

Before the generation could be completely automatic, we have implemented a semi-automatic program, which is used to test the result of variety algorithms. This program could show the segmented area, preview the changes by applying different effects, save the changes and save generated images.

### 7.2 Interface

The program is written in MFC and has an executable that runs on windows platform.

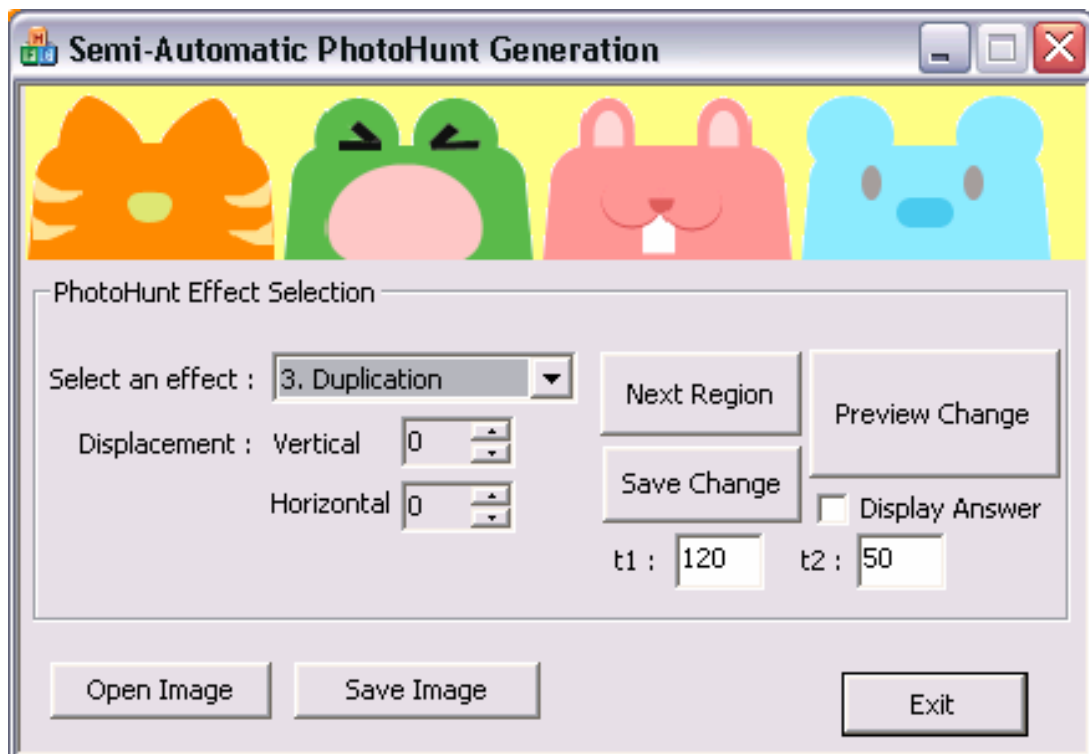


Fig. 7.1

There are several windows components such as buttons and text boxes that performs different functions. The table bellows shows the brief description of each component.

Description of Label/Button/Menu/Check Box	
Select an effect	Labels effect is selected.
Vertical	Vertical offset for Duplication effect only
Horizontal	Horizontal offset for Duplication effect only
[Pull Down Menu]	The menu is followed by the label. There are three effects current : 1. Elimination 2. Color Change 3. Duplication
Next Region	Shows the next segmented region on the original image
Preview Change	Preview the changes after applying one effect on current segmented area
Save Change	Save the changes on the image
Display Answer	Display the segmented region on a reference image
t1	Threshold1 for Pyramid Segmentation
t2	Threshod2 for Pyramid Segmentation
Open Image	Open an image file, currently supported BMP and JPEG format
Save Image	Save the generated image to file in JPEG format
Exit	Terminate the Program

## **7.3 Implementation**

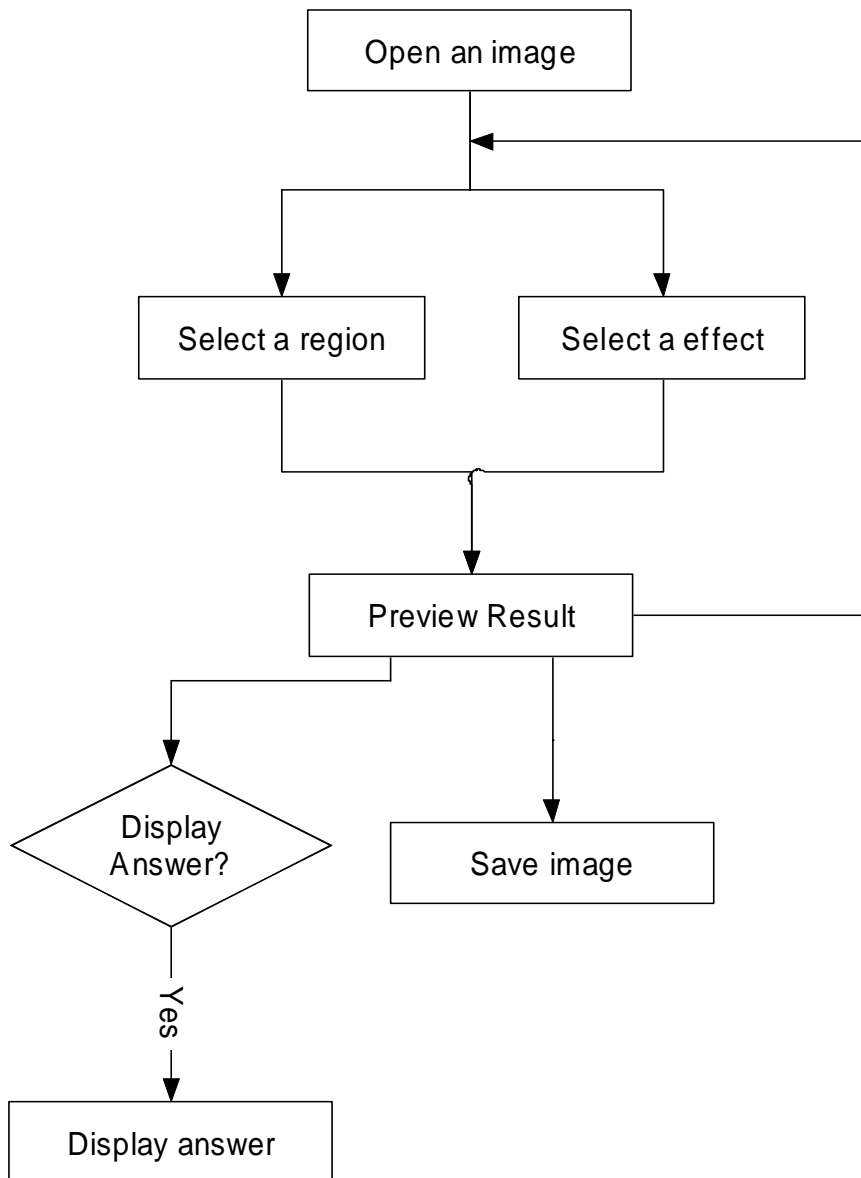
The Semi-Automatic Generation Engine (SAGE) is mainly supported by the Image Generation Engine (IGE). The program allows user to adjust several parameters such as thresholds and offset to the IGE, and send commands to the IGE while the IGE will give the results back to SAGE to display.

### **7.3.1 Flow Chart**

Every operation starts by opening an image and the image will be passed to segmentation module automatically.

The Semi-Automatic program allows the manual selection of segmented regions and effects, and then the result could be previewed. If the result is desirable, user could save the change. This process could be repeated several times to make numbers of differences on the image.

Finally, the program allows users to save the produced image to the local drive.





### 7.3.2 Operations

To generate different images, several steps are taken. The following screen shots and the description illustrate how the change is made and saved.

Step 1 : After opening an image, a segment (marked in red) was found by pressing "Next Region".



Fig. 7.2



Step 2: Clicking the “Preview Change” button with checked in “Display Answer” will perform the effects that selected in the effect box which is Elimination in the following figure. A preview windows will show the image with changes and a reference image (answer) will be shown too.

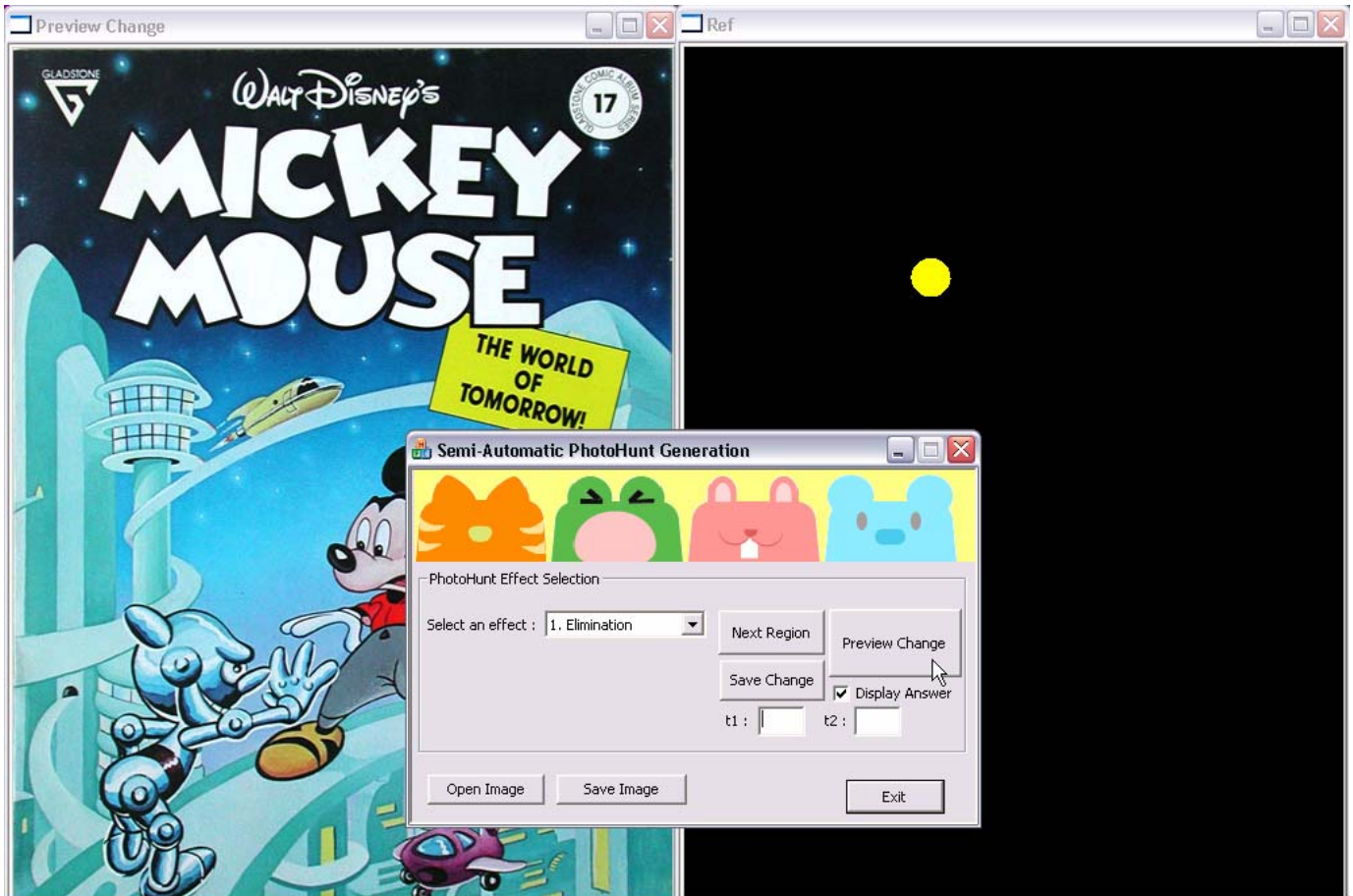
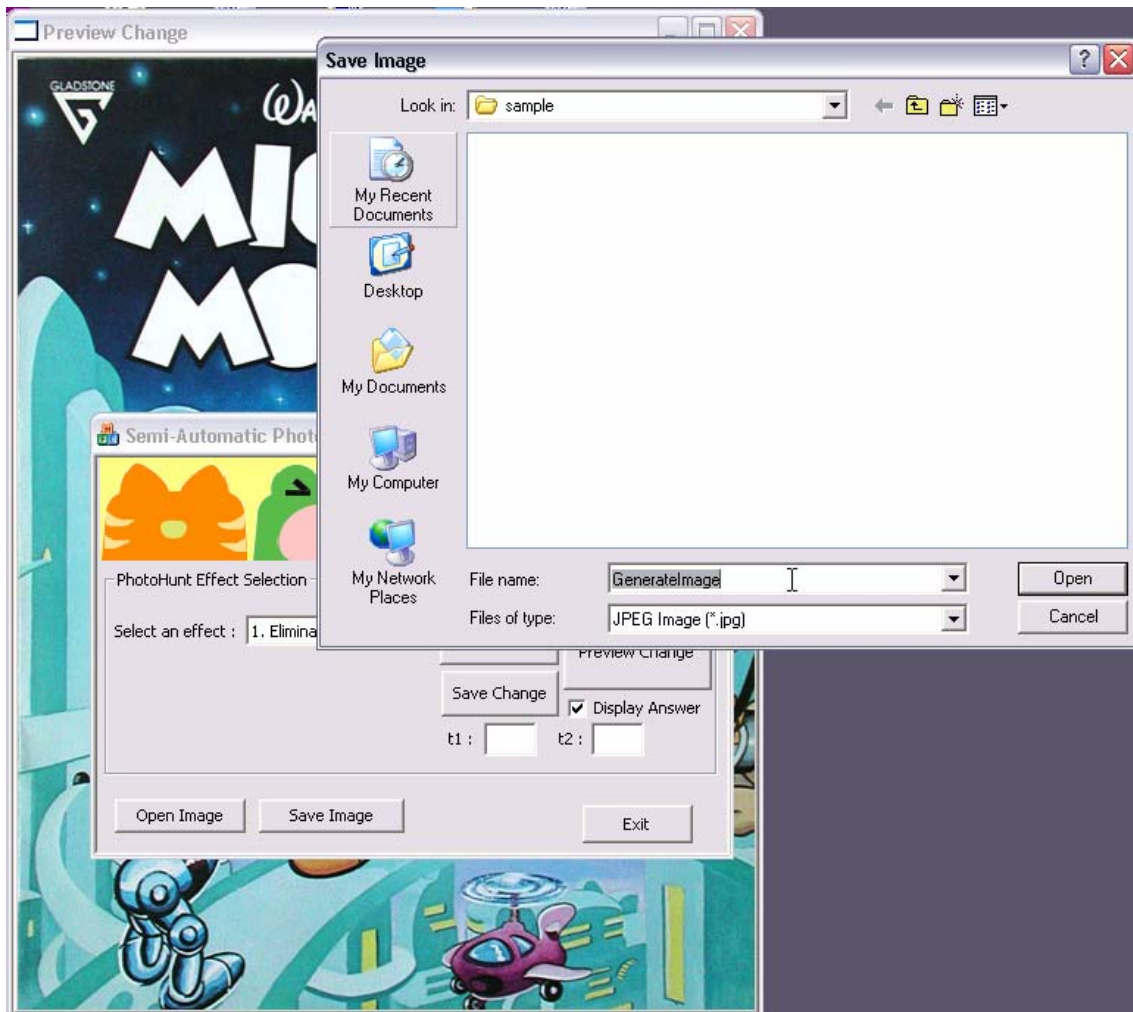


Fig. 7.3

Step 2 : The “Save Change” will save the changes the image generation engine just made. Finally the “Save Image” button is invoked and a save file dialog will be shown. The generated image could be saved in JPEG format.

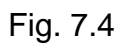


## Chapter 8 Game Engine for Web Application

---

After images had been generated by the Automatic Generation Engine, it will be passed to the front-end component, Game Engine, which will be used for playing the game. The engine made use of two common software and programming language; they are Flash 8 and PHP which are widely used for web applications.

This Game Engine comprised of two parts. One is the game interface which is implemented by Flash. It is the only part that could be seen by users. Through the interface, users could upload an arbitrary image. A generated image, together with the original one, will then return to the interface. The game is then ready to play.

Apart from visible component, the backend is supported by a PHP script behind. It handles all the operations of files  such as saving the image into proper location. It also acts as a bridge between the flash application and the Image Generation Engine. Besides, it helps to forward request to the Image generation engine and then pass the returned result back to the flash application.

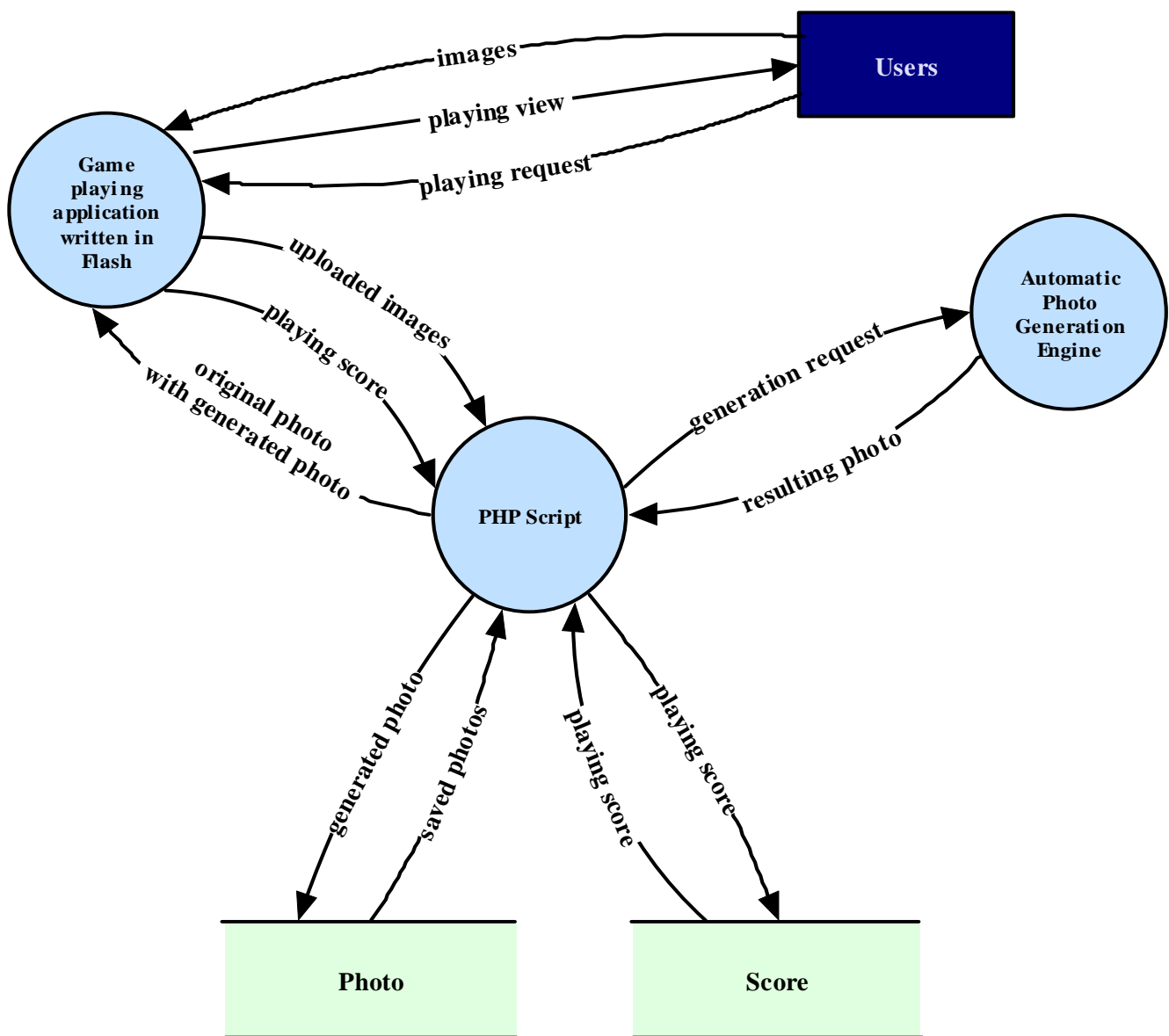
The game is played in the same way as a traditional PhotoHunt, but with more features, such as multiplayer mode, which will be implemented later on. In the following sections, a brief description of the engine will be given.

## 8.1 Architecture

### 8.1.1 Data Flow

The following diagram shows the data flow of the engine :

## Game Engine for Web Application



Starting from the user, there are two options. A user could choose either uploading an image or make a playing request to the PHP script through the flash application.

### **A. Processing User Requests**

When the PHP script receive an uploaded image, it saves the image as an original image and makes a record to database. The script then makes a request to generate images by calling an external component, which is the Image Generation Engine, associated with the original image. The generation engine carries out analysis of the image and performs usual manipulation in order to produce one or more “modified” image(s). Each of those images has several differences. The result will return to the PHP script afterward and the script will save the images as well. The flash application will retrieve two of these images and displays them at the same level horizontally for playing.

In another way, if the PHP script received a “playing request” from user, it looks into the database; randomly choose one from the records and send the images back to the flash application. Once again, the flash application displays the images as above.

## B. Game Playing

After user finished the game, a ranking would be given to the particular images according to the accuracy and the duration. This result will be automatically sent to the PHP script by the flash application. The PHP script will store this ranking to database for further usage.

## 8.2 User Interface

The Following image shows the first page of our engine for end user.

The entire front-end interface is implemented by flash, which requires the Flash player with version 8 and runs mainly on the web browser.



## 8.2.1 Functionality

The design of the interface is simple and straightforward. In the main page, only two buttons could be clicked to activate the game.

These two buttons are “Browse” and “Upload” as shown in Figure 9.1, with a white box nearby. The “Upload” button is disabled by default.



Figure 9.1

When the “Browse” button is pressed, a Select-File-To-Upload window will appear. User can then select image from the box. Only JPEG format is supported in current stage.

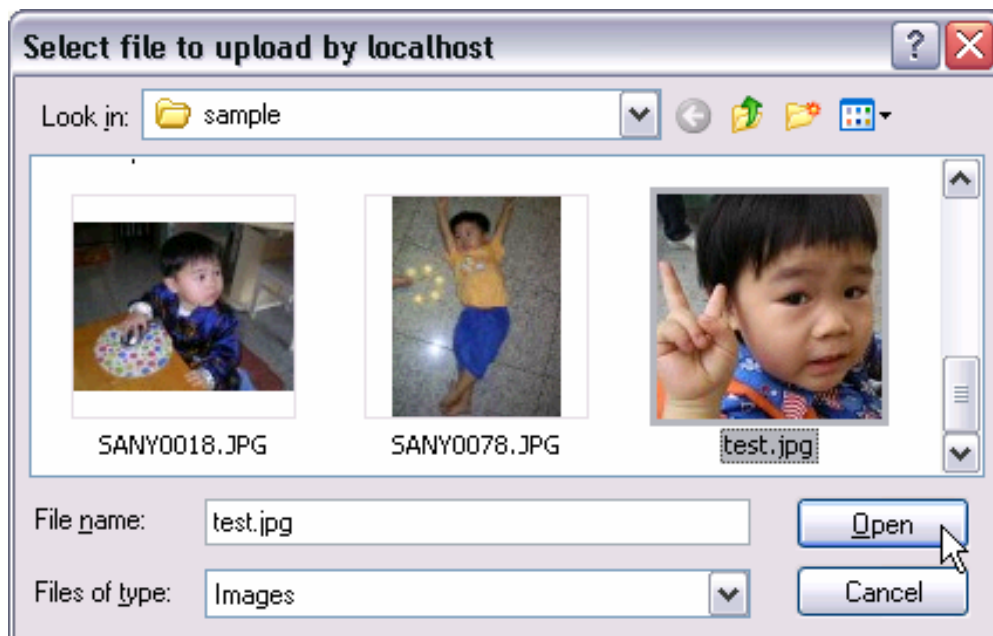


Figure 9.2

After selecting the images, clicking the “Open” button will hide the select-file box, while the actual path of the file will then be saved to the flash and the name will be shown in the white box. (Figure 3)



Figure 9.3

The “Upload” button is enabled after the file selection stage. We could click it to start transferring the local image to the server.

The White border which are surrounding the white box and two buttons are actually the upload progressing bar. It shows the percentage of completion of the uploading process. (Figure 9.4)

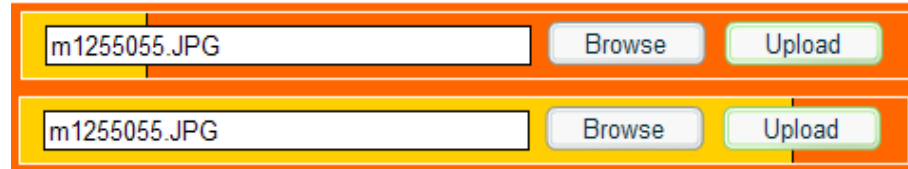


Figure 9.4

A modified image will be generated by the Automatic Generation Engine at background after the image is completely uploaded to the server. The flash is then asking the server to retrieve the original image and the modified one to display them in the main pane of the interface.



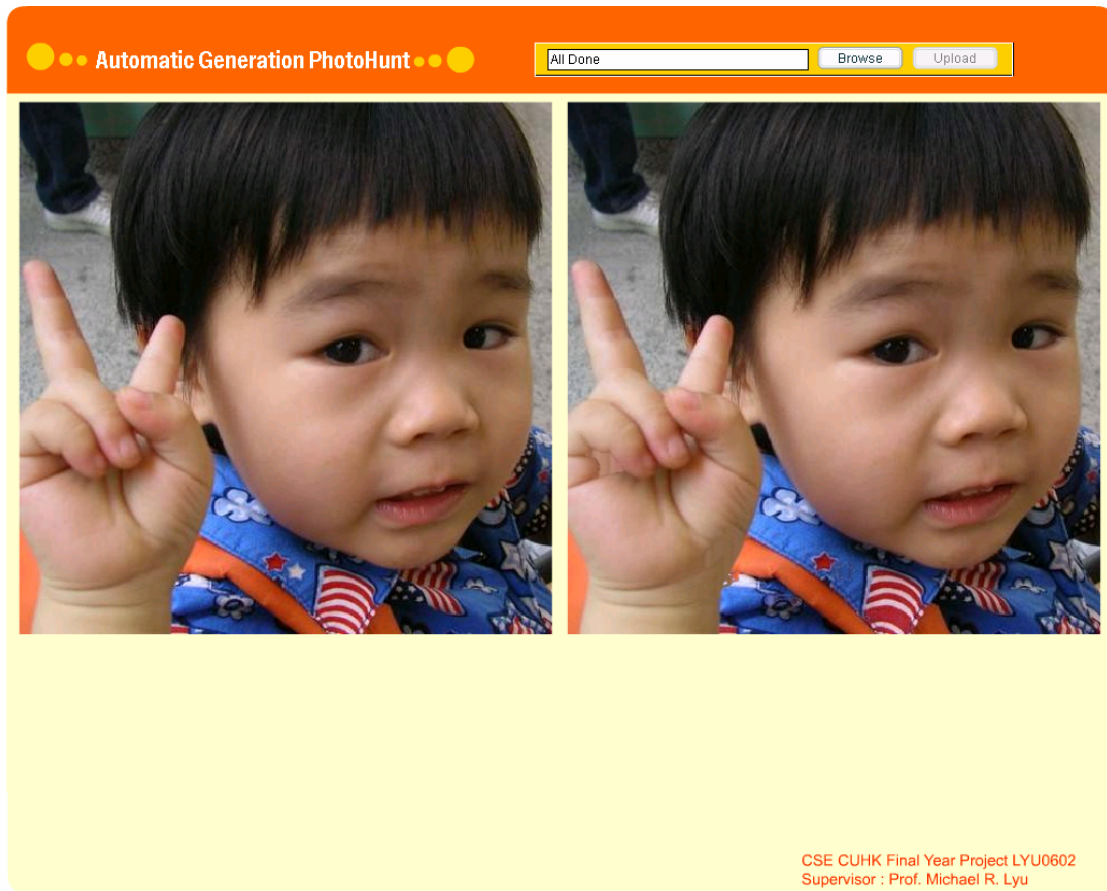


Figure 9.5

## **Chapter 9 Limitation**

---

### **9.1 Segmented area found noise and distortion**

The segmentation sometimes leaves the boundaries of modified area in the generated image. We tried to offset the selected area by a few pixels; however it might even cause a more significant problem when the neighbor segments are too close. Although this distortion problem is alleviated by applying the filter, we believed that there are still rooms for improvement.

### **9.2 No Global Information between the segments**

The segmentation method we are employing return only set of segments to the effect changing module, it does not provide any information between the segments. This information can be useful for us to understand the surrounding of the objects in order to apply proper changes

### **9.3 No artificial intelligence control to the Applied effect**

The elimination effects are now applied to any segment or object of a photo randomly. Since we did not consider the nature of the eliminated object, in some scenarios, for instance, the removal of eye may produce scary photo for users.

## **9.4 Not all images can be segmented**

If a user supplies an image that can not be segmented or can only be segmented to a few obvious components, no image can be generated. In this case, our program will display an error message to encourage user to provide another photo.

## Chapter 10 Difficulties

---

We both have limited knowledge of image processing. What we first face is the problem that the project relies heavily on image processing. It turns out that we used quite a long time to carry out the image processing foundation.

The second difficulty that we encounter is that we have to figure out the way to go by applying appropriated algorithms. It seems that only very limited numbers of projects are of similar nature as this one, which means references available are somehow limited. So, as we are starting from zero, we have to carefully define the problem, have brain storm and implement the system with a lot of trial-and-error.

Since the images used in PhotoHunt could be complicated; and the judgment of result of changes on image is subjective, a generic solution is hard to derive. For any of the algorithms which are developed by us or the others, it may work well under one condition but it fails in another environment. A better solution might be applying several algorithms in order to cope with different conditions; however, it is still hardly to say it could works everywhere.

## Chapter 11 Project Progress

---

Date	Progress
September	<ul style="list-style-type: none"> <li>· Carry out background research on PhotoHunt</li> <li>· Set goal and objectives for this semester</li> <li>· Learn and Got ourselves familiar with image processing</li> <li>· Study the basic data structure and operation provided by OpenCV with C++</li> <li>· Write some testing program to analyze different segmentation approach</li> </ul>
October	<ul style="list-style-type: none"> <li>· Study the usage of MFC to constructed the Semi-Automatic PhotoHunt Generation program</li> <li>· Design and test different elimination algorithms</li> <li>· Implement the elimination to the Semi-Automatic PhotoHunt Generation Engine.</li> </ul>
November	<ul style="list-style-type: none"> <li>· Make improvement to the existing elimination algorithm</li> <li>· Learn Flash action script and PHP for the development of the Game Engine</li> <li>· Build the Game Engine for uploading images and invoking the generation function provided the Image Generation Engine.</li> <li>· Prepare report and presentation</li> </ul>

## **Chapter 12 Future Works**

---

### **12.1 Continues to improve the segmentation algorithm**

The algorithm of segmentation is not works perfectly in some cases. A bad segmented area can downgrade the quality of the effect. Also, the current segmentation left some noise on the edge of the segmented area. We would like to improve the accuracy of the algorithm in order to obtain a better result.

### **12.2 Building up a global view of the images**

Currently the elimination algorithm only considers a few pixels which are surrounding the segmented area, and performs the color manipulation based on such insufficient information. For a better elimination, a larger area should be considered; even a global view is needed, and the relationship between each segment will helps. What we will try in the next semester, is to try for working on extracting the information carried by images and apply to the effect for PhotoHunt.

### **12.3 Introduce more effects**

We only focus on the elimination effect in this semester, since it is on the top list. However, elimination is not only effect to today's' PhotoHunt Game. In the next semester, we hope to enrich the image generation engine by introducing more effects including duplication, image transformation.

### **12.4 Adding features to the Web-Based game engine**

The web-base game engine, which is also an important part in our system, is a great application that involves our image generation engine. More features, such as versus mode and difficulty levels adjustment, will be added to the game engine to uplift the entertainment.

## Acknowledgement

---




We would like to thank our final year project supervisor, Professor Michael R. Lyu, who gave us unlimited support and invaluable advice. We really want to thank him for his kindness and support.

We would also like to thank Mr. Edward Yau, for he gave a lot of useful ideas and technical help in our project. He enriched our project by providing us with a lot of information we need.



## Appendix 1 Testing Data & Result

A1 Images generated by the testing Program to view the result of Gaussian filtering and Median Filtering Function

Median Filtering	Neighbor Size			
	0	3	5	7
				
	Filter Size			
	0	3	5	7
Gaussian Filtering				
	Sigma (Filter Size=)			
	1	2	3	
				

A2 Data generated by threshold segmentation testing  
program(ThresSeg.cpp)

Index		Index		Index		Index	
0	168	64	324	128	738	192	3999
1	141	65	375	129	804	193	3990
2	369	66	351	130	822	194	3942
3	168	67	333	131	873	195	4149
4	273	68	420	132	924	196	4269
5	258	69	393	133	921	197	4185
6	261	70	411	134	993	198	3756
7	186	71	498	135	1008	199	3498
8	168	72	498	136	1041	200	3147
9	144	73	687	137	1191	201	2889
10	144	74	786	138	1701	202	2760
11	138	75	786	139	2181	203	2985
12	111	76	762	140	2103	204	3255
13	141	77	987	141	1785	205	3735
14	141	78	993	142	1707	206	4224
15	147	79	1023	143	1692	207	5076
16	138	80	1137	144	1725	208	4584
17	126	81	1164	145	1668	209	4818
18	183	82	1068	146	1599	210	5325
19	135	83	1092	147	1491	211	5394
20	117	84	999	148	1743	212	5124
21	111	85	978	149	1674	213	4209
22	123	86	1155	150	1695	214	3429
23	114	87	1059	151	1704	215	2802
24	171	88	1197	152	1587	216	2043
25	150	89	1284	153	1623	217	1278
26	171	90	1365	154	1686	218	1224
27	162	91	1518	155	1533	219	1227
28	201	92	1635	156	1677	220	1425
29	207	93	1728	157	1539	221	1545
30	192	94	1947	158	1533	222	1707

31	156	95	1797	159	1503	223	2085
32	195	96	2094	160	1608	224	2022
33	249	97	1980	161	1518	225	2232
34	198	98	1941	162	1758	226	2493
35	240	99	2139	163	2082	227	3204
36	216	100	1995	164	2256	228	3210
37	240	101	2040	165	2295	229	4509
38	207	102	2235	166	2097	230	5739
39	255	103	2802	167	1935	231	5748
40	288	104	3771	168	2076	232	7299
41	222	105	5433	169	2316	233	10278
42	180	106	8838	170	2496	234	9498
43	252	107	14103	171	2607	235	9927
44	237	108	9255	172	2292	236	11163
45	198	109	5181	173	2370	237	14217
46	213	110	3030	174	2109	238	19956
47	168	111	1914	175	2157	239	57777
48	204	112	1302	176	2118	240	67809
49	186	113	996	177	2250	241	38247
50	216	114	858	178	2178	242	52335
51	150	115	690	179	1965	243	12675
52	198	116	711	180	2058	244	7743
53	219	117	669	181	2145	245	6747
54	192	118	687	182	2250	246	4809
55	225	119	558	183	2742	247	4758
56	231	120	708	184	3045	248	2349
57	216	121	699	185	3402	249	3009
58	246	122	768	186	3270	250	426
59	315	123	768	187	3207	251	345
60	282	124	690	188	3279	252	318
61	261	125	807	189	3699	253	531
62	288	126	723	190	3624	254	192
63	258	127	825	191	3954	255	465

A3 Original and Segmented image by Thresholding



Original Image1  
Threshold computed= 133



Segmented Image 1



Original Image1  
Threshold computed= 86

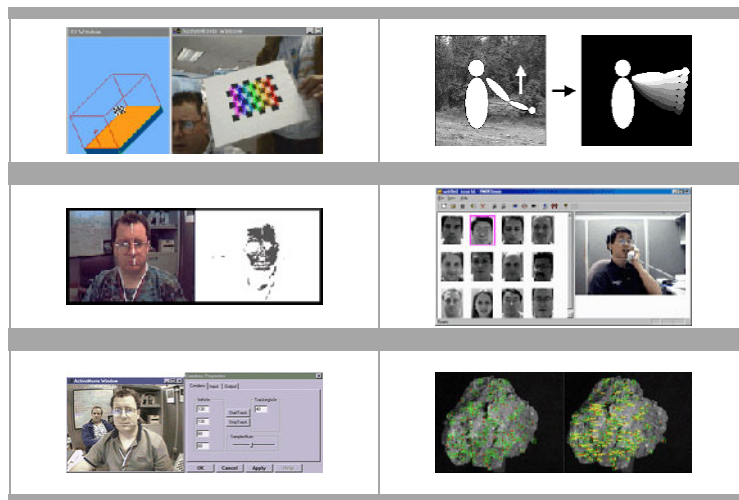


Segmented Image 2

## Appendix 2 OpenCV

---

OpenCV[10] is the Open Source Computer Vision Library that mainly aimed for real time computer vision. It implemented various types of tool to complete image interpolation. Other than the primitives such as binarization, filtering, image statistics, OpenCV is also a high-level library implementing algorithms for Human-Computer Interaction (HCI); Object Identification, Segmentation and Recognition; Face Recognition; Gesture Recognition; Motion Tracking, Ego Motion, Motion Understanding; Structure From Motion (SFM); and Mobile Robotics..



## OpenCV Overview

OpenCV comprise of 5 main classes, namely CXCORE, CV, HIGHGUI, CVAUX and CVCAM to provide programming functions for different aspect in image interpolation area.

Library area:

Chapter	Contents
Image functions	Creation, allocation, destruction of images. Fast pixel access macros.
Data Structures	Static types and dynamic storage.
Contour Processing	Finding, displaying, manipulation, and simplification of image contours.
Geometry	Line and ellipse fitting. Convex hull. Contour analysis.
Features	1st & 2nd Image Derivatives. Lines: Canny, Hough. Corners: Finding, tracking.
Image Statistics	In region of interest: Count, Mean, STD, Min, Max, Norm, Moments, Hu Moments.
Image Pyramids	Power of 2. Color/texture segmentation.
Morphology	Erode, dilate, open, close. Gradient, top-hat, black-hat.
Background Differencing	Accumulate images and squared images. Running averages.
Distance Transform	Distance Transform
Thresholding	Binary, inverse binary, truncated, to zero, to zero inverse.
Flood Fill	4 and 8 connected
Camera Calibration	Intrinsic and extrinsic, Rodrigues, un-distortion, Finding checkerboard calibration pattern

View Morphing	8 point algorithm, Epipolar alignment of images
Motion Templates	Overlaying silhouettes: motion history image, gradient and weighted global motion.
CAMSHIFT	Mean shift algorithm and variant
Active Contours	Snakes
Optical Flow	HS, L-K, BM and L-K in pyramid.
Estimators	Kalman and Condensation.
POSIT	6DOF model based estimate from 1 2D view.
Histogram (recognition)	Manipulation, comparison, backprojection. Earth Mover's Distance (EMD).
Gesture Recognition	Stereo based: Finding hand, hand mask. Image homography, bounding box.
Matrix	Matrix Math: SVD, inverse, cross-product, Mahalanobis, eigen values and vectors. Perspective projection.
Eigen Objects	Calc Cov Matrix, Calc Eigen objects, decomp. coeffs. Decomposition and projection.
embedded HMMs	Create, destroy, observation vectors, DCT, Viterbi Segmentation, training and test.
Drawing Primitives	Line, rectangle, circle, ellipse, polygon. Text on images.
System Functions	Load optimized code. Get processor info.
Utility	Abs difference. Template matching. Pixel order<->Plane order. Convert Scale. Sampling lines. Bi-linear interpolation. ArcTan, sqrt, inv-sqrt, reciprocal. CartToPolar, Exp, Log. Random numbs. Set image. K-Means.

## **OpenCV in Our Project**

OpenCV facilitate our projects by providing various types of image processing functions. With the use of openCV, it streamlined our testing and implementation process.

However, just as the conclusion in paper[12], openCV was not adequately commented and documented for the use of non-research community. Although it provides many useful resources, it always take time for us to understand the data structure and the relationship between each function. Moreover, when we encountered difficulties and tried to raise a question in the official yahoo group, the replies are not actually providing a solution but to raise other non-relevant questions. Therefore, understanding a single function sometime may require us lots of research and trial-and-error behind.



## Reference

---

- [1] Liquidfuse, <http://www.liquidfuse.com/photohunt-play.htm>
- [2] Power Hunt, <http://www.wahon.net/~johnson/PhotoHunt/index.htm>
- [3] Critical Seeker 4.1,  
[http://www.filedudes.com/Critical\\_Seeker-screenshot-12753.html](http://www.filedudes.com/Critical_Seeker-screenshot-12753.html)
- [4] Wikipedia, [http://en.wikipedia.org/wiki/Image\\_processing](http://en.wikipedia.org/wiki/Image_processing)
- [5] Canny, J., "A Computational Approach to Edge Detection", IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, November 1986
- [6] Roland Wilson, Michael Spann, "Image Segmentation and Uncertainty", Research Studies Press Ltd, 1988.
- [7] E H Adelson, C H Anderson, J R Bergen, P J Burt, and J M Ogden.  
"Pyramid methods in image processing." RCA Engineer, 29:33--41, 1984.
- [8] Bun, P J.. Hang, T. H., Rosenfeld, A. "Segmentation and Estimation of Image Region Properties through Cooperative Hierarchical Computation." IEEE Transactions on System. Man. and Cyhematics. Vol. SMC-II.No. I2,December 1981.
- [9] "Open Source Computer Vision Library" <http://www.intel.com/research/mrl/>
- [10] Intel Corporation, "Reference Manual of OpenCV"
- [11] Kosir, A.; Tasic, J.F.; "Pyramid segmentation parameters estimation based on image total variation" EUROCON 2003. Computer as a Tool. The IEEE Region. 8 22-24 Sept. 2003

- [12] Cavallaro, A., "Image analysis and computer vision for undergraduates",  
Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP  
'05). IEEE International Conference, March 18-23 2005
- [13] Qingcang Yu; Cheng, H.H.; Cheng, W.W.; Xiaodong Zhou, "Interactive open  
architecture computer vision" Tools with Artificial Intelligence, 2003.  
Proceedings. 15th IEEE International Conference , 3-5 Nov. 2003
- [14] How to obtain parameters from HTML to SWF  
<http://www.jiagao.net/archive/1104378756.php>