

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

Abstract

Many tedious tasks can now be done by computers automatically. As the processing power of computers and mobile devices improves, these devices can now perform many demanding computation tasks that were considered impractical a few years ago.

In this project, we built a system that can automatically generate one or more PhotoHunt images from a given original. These images are derived from the original for the PhotoHunt game, in which a user is asked to identify the differences between the derived images.

The system we built is based on various image processing techniques. It is highly flexible that can generate an image into two derivatives in less than a second. In order to test the system and obtain the feedback from players, we have built an online application. We are encouraged to find that this online game has recorded more than 100 visitors per day since the launch.

Table of Contents

Abstract	2
Chapter 1 Introduction	7
1.1 Motivation	7
1.2 Project Objectives.....	8
Chapter 2 Analysis of the PhotoHunt.....	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
Chapter 5 Image Analysis		40
5.1	Parameters	41
5.1.1	Object and Background Average	42
5.1.2	Object Standard Deviation.....	43
5.1.3	Background Mode	43
5.1.4	Object and Background Neighbor Difference	43
5.2	Parameter Usages and Applications	45
5.2.1	Screening out segment with color that are too similar to background	45
5.2.2	Deciding the type of modification to be applied	48
5.2.3	Providing suggestion on replacement color for elimination.....	49
Chapter 6 Image Generation Engine		50
6.1	Processing Flow.....	51
6.2	Segmentation module	53
6.3	Elimination Module.....	54
6.4	Color Change Module	54
6.5	Object Appending Modules	55
6.6	Image Warping Modules.....	55
6.7	Smoothing Image.....	55
Chapter 7 Elimination Algorithms		56
7.1	Direct Copy Algorithm	56
7.1.1	Testing Result	56
7.2	Horizontal Gradient Algorithm.....	58
7.2.1	Test Result	59
7.3	Nearest Boundary Algorithm.....	61

7.3.1	Testing Result	62
7.4	Enhanced Nearest Boundary Algorithm	64
7.4.1	Testing Result	64
7.5	Hybrid Elimination Algorithm.....	66
7.5.1	Testing Result	68
7.6	Conclusion	69
Chapter 8	Image Warping.....	70
8.1	Image Warping Algorithm	70
8.1.1	Transformation Equation	71
8.2	Image Warping Results.....	75
Chapter 9	Appending Object	77
9.1	Appending object algorithm	77
9.2	Appending object result.....	78
9.3	Limitation	79
Chapter 10	Semi-Automatic Generation Program	80
10.1	Introduction	80
10.2	Interface	80
10.3	Implementation.....	81
10.3.1	Flow Chart	82
10.3.2	Operations.....	84
Chapter 11	Game Engine	87
11.1	Overview	88
11.1.1	Processing User Requests.....	88
11.1.2	Game Playing	89
11.2	Front-end	89
11.2.1	Main Menu	90
11.2.2	Single Player.....	90
11.2.3	Multiple Player	94

11.2.4 Challenge Mode.....	96
11.3 Backend	97
11.4 Database	98
Chapter 12 Evaluation	99
12.4 Acceptance Rate	99
12.5 Processing Time.....	100
12.6 Image Quality	101
Chapter 13 Problems and Solutions	104
13.1 Segmented area found noise and distortion.....	104
13.2 No Global Information between the segments	104
13.3 Not all images can be segmented	105
13.4 Copyright Issue of User Upload Image	105
Chapter 15 Contribution of work	108
15.1 Preparation.....	108
15.2 Semester One.....	108
15.3 Semester Two.....	109
15.4 Conclusion.....	110
Appendix 1 Testing Data & Result	112
Appendix 2 Analysis data.....	116
Appendix 3 Data Dictionary of Database	117
Appendix 4 Evaluation Data	119
Appendix 5 Internal Test	120
Reference.....	126

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.

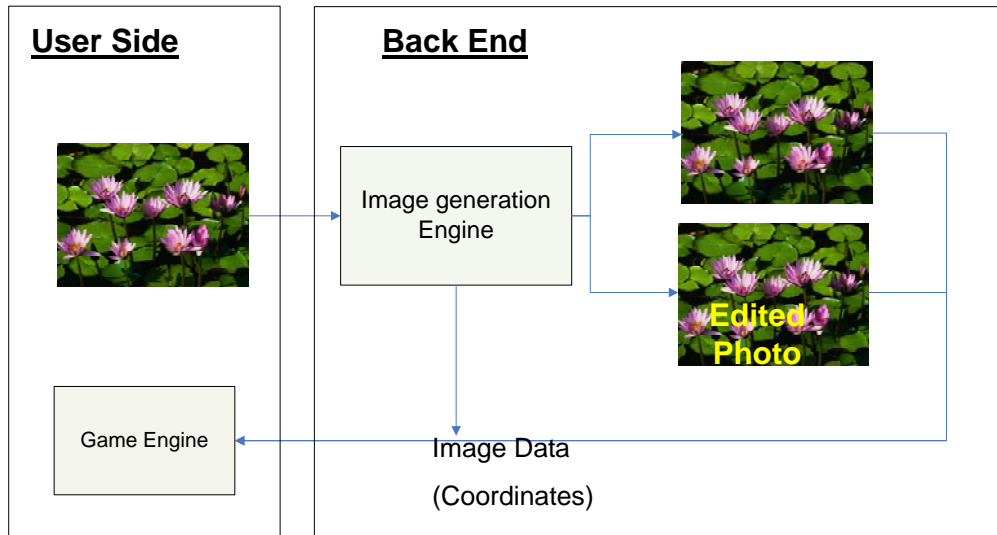
There is always an alternative, since the tastes are varying from people to people. Despite the well-known computer games that are advertised everywhere, in reality, the simple games, still holds their position. Moreover, the simplicities of these games provided them high flexibility to implement in 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 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 during the year.

Chapter 2 Analysis of the PhotoHunt

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 the very beginning.

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.

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

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

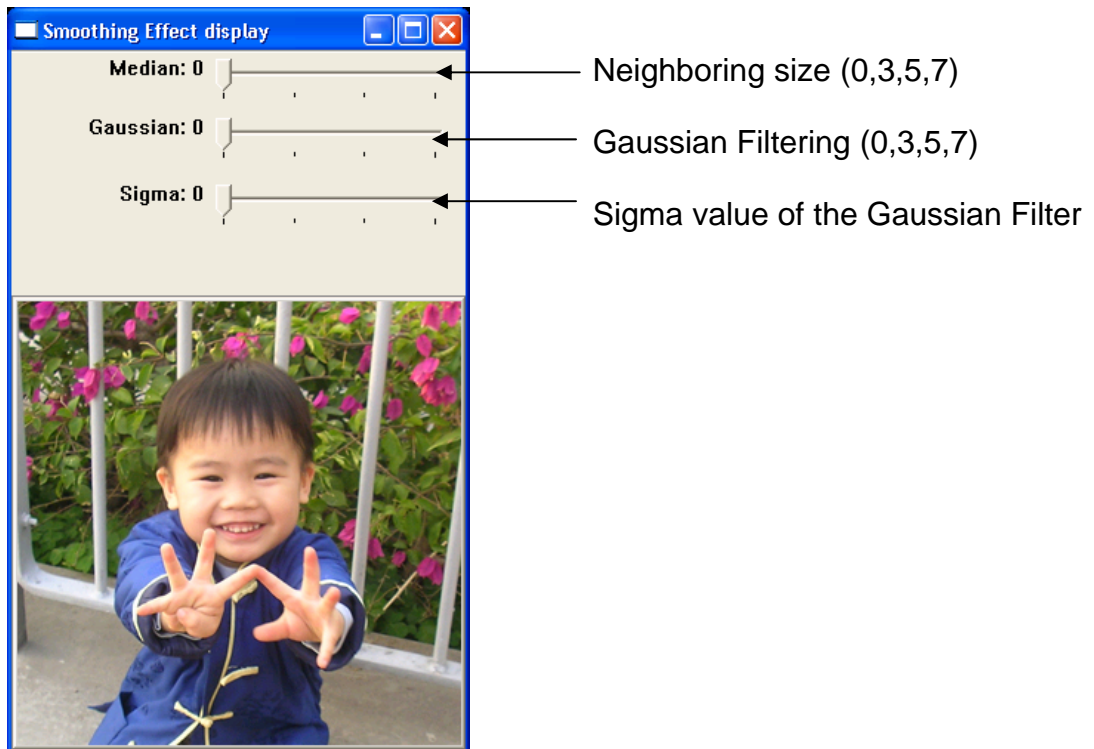
$$S(x, y) = \text{Image}(x, y) \cdot G(x, y)$$

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

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.

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

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

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

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

$c(x, y)$ = 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

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

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

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

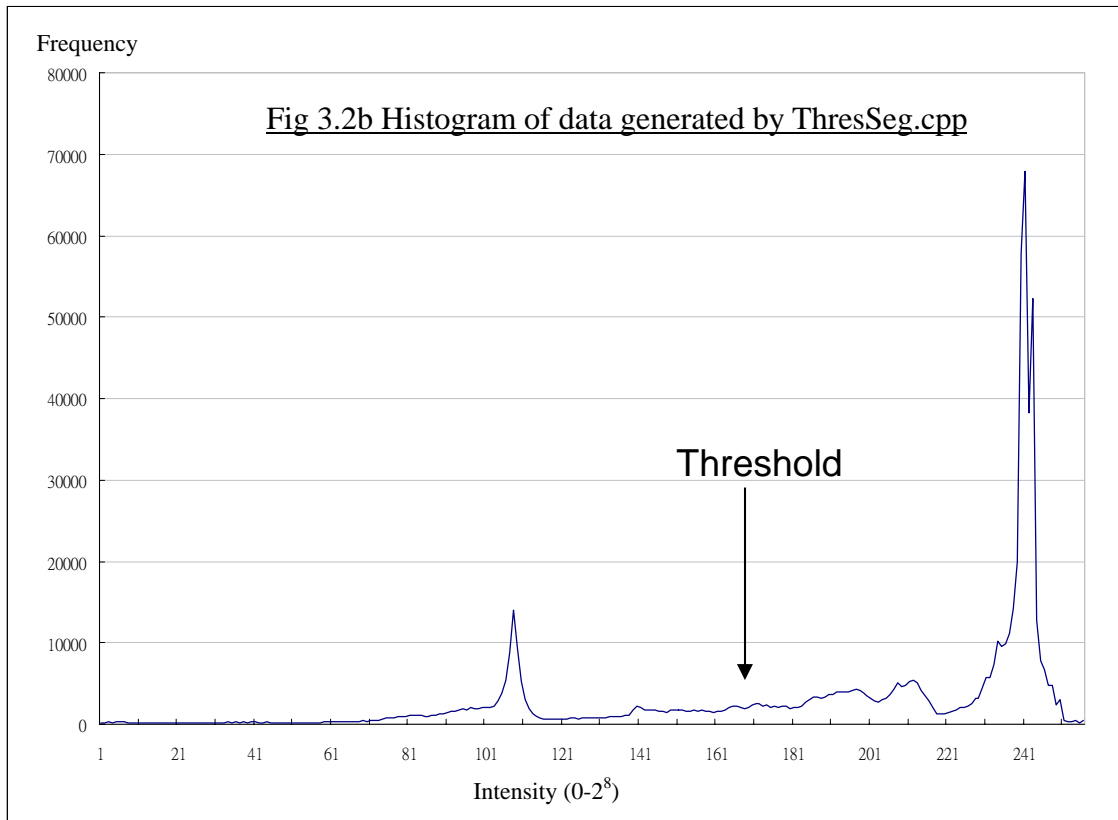
(i.e. the mean of the pixel label as background in the i-th iteration)



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

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

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, θ

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

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

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.

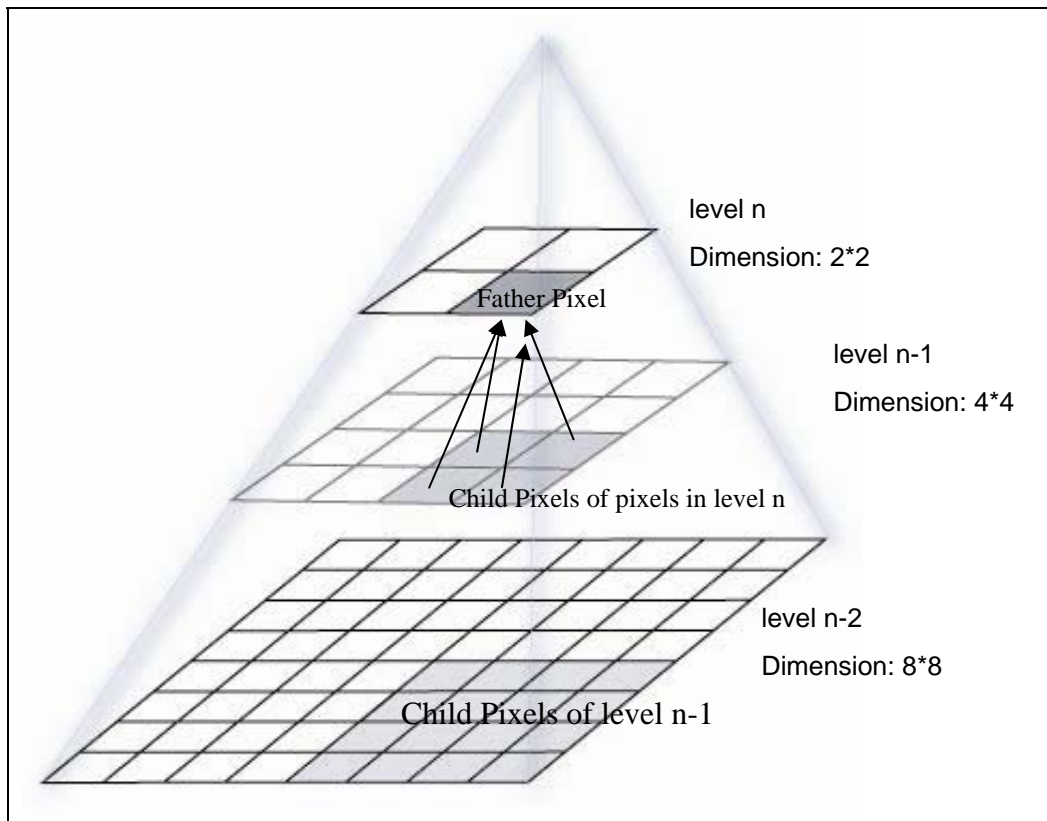


Fig 4.1 This diagram shows the graphical representation of the Image Pyramid.

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

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

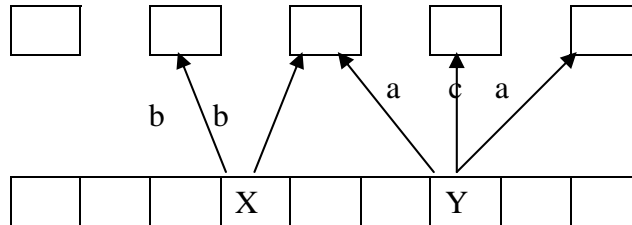
c) Symmetric

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

d) Equal contribution

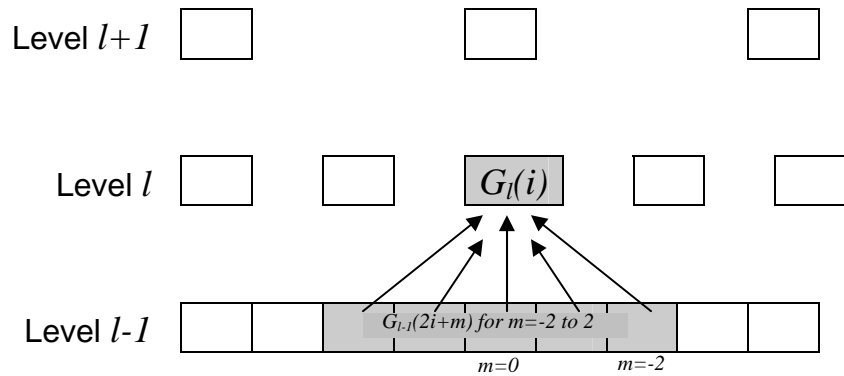
Each pixel should have equal contribution to the father pixel in the upper level.

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.

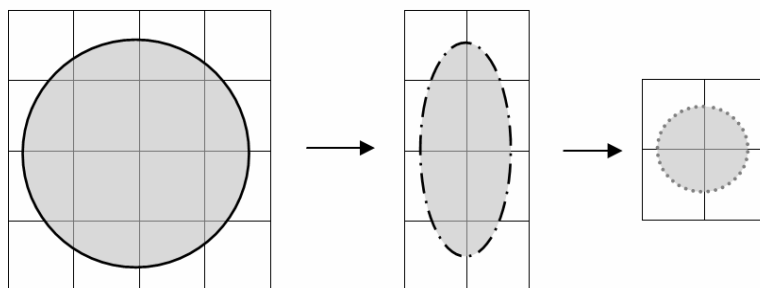


Fig 4.2 Applying reduction twice to get the father level

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 4.3 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)

1	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 4.3a 5x5 Gaussian Matrix

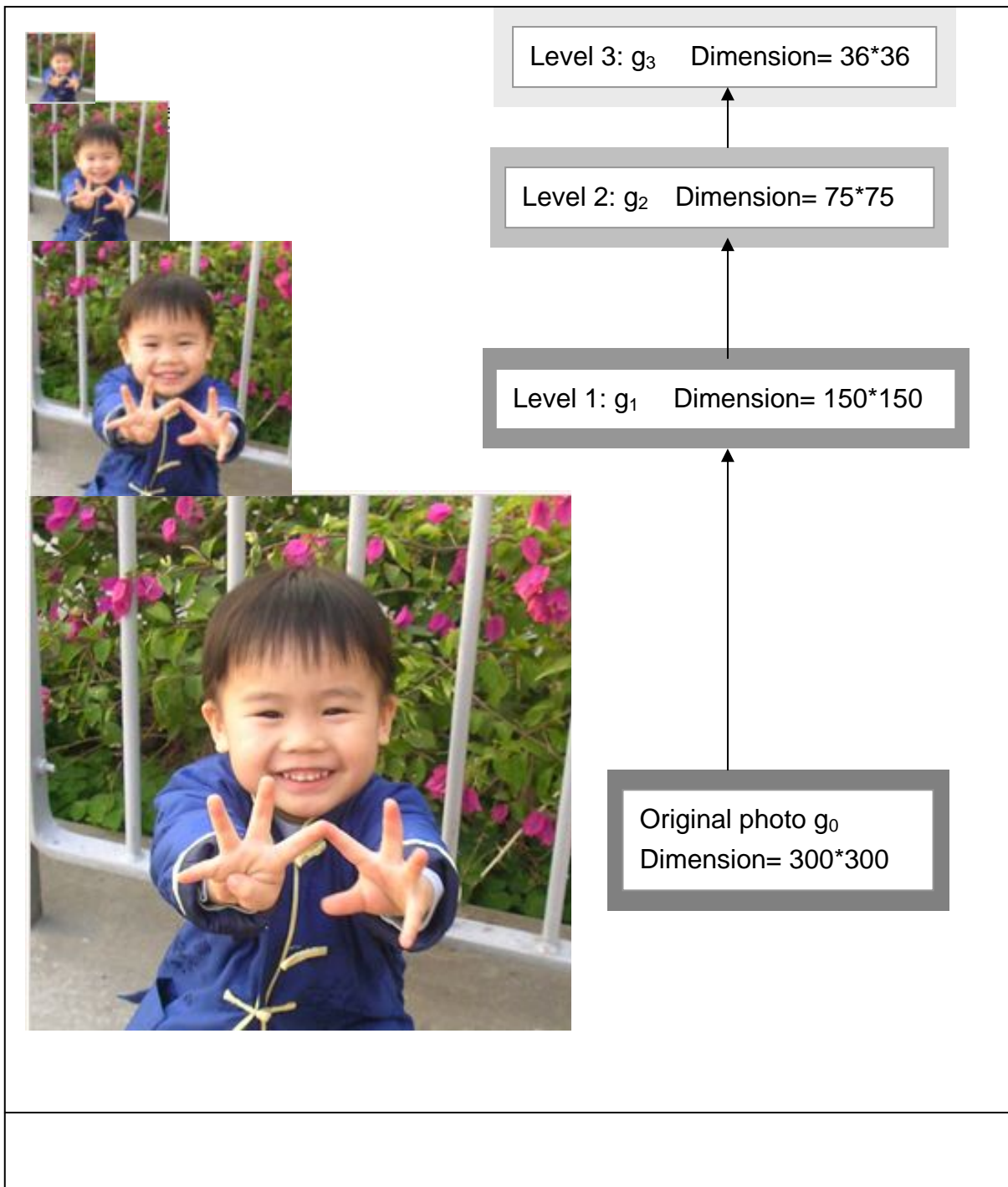


Fig 4.3 Generation of Gaussian Pyramid

4.3 Segmentation by pyramid-linking

After the Guassian 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.

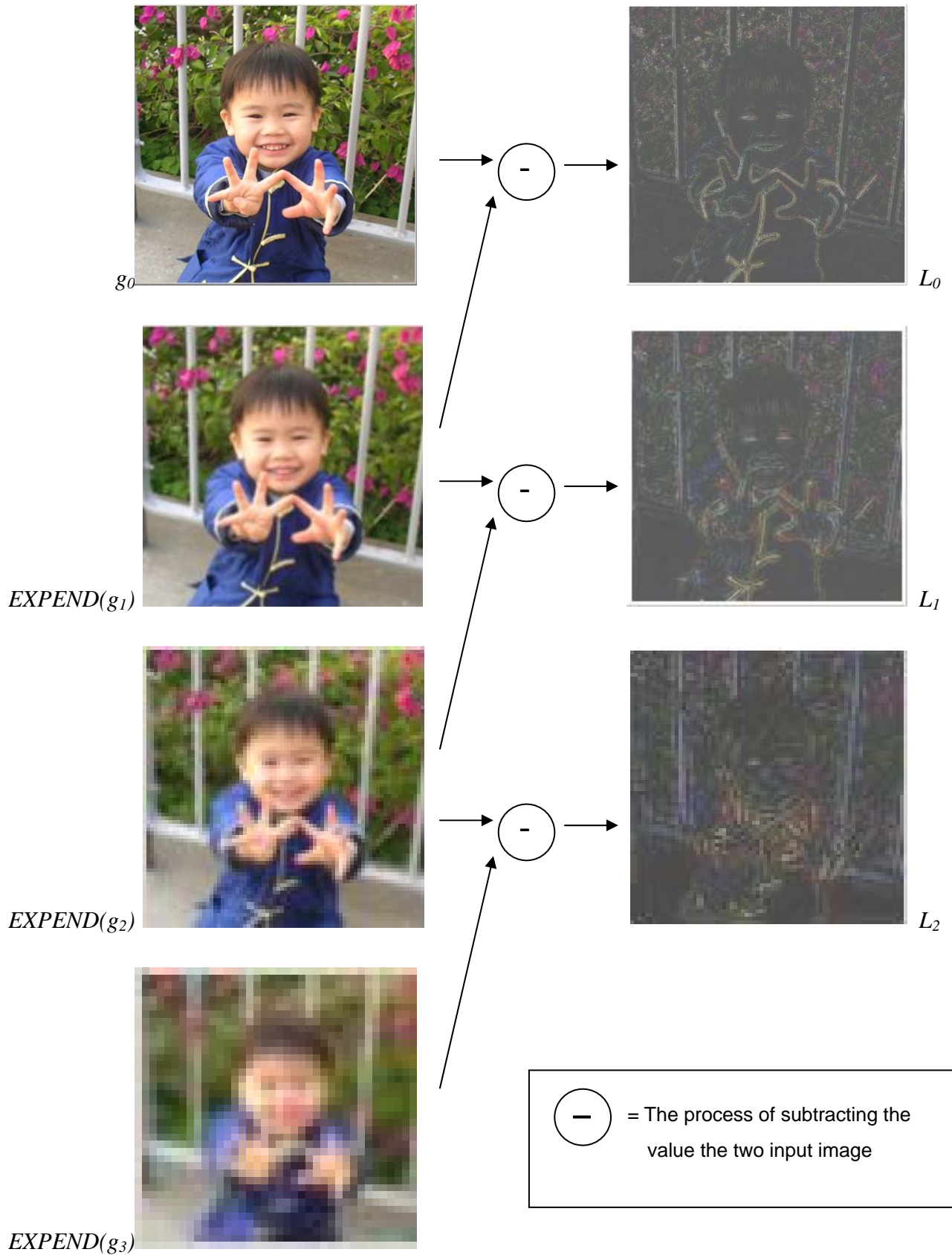


Fig 4.4 shows the Generation of Laplacian Pyramid with the aid of OpenCV

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.

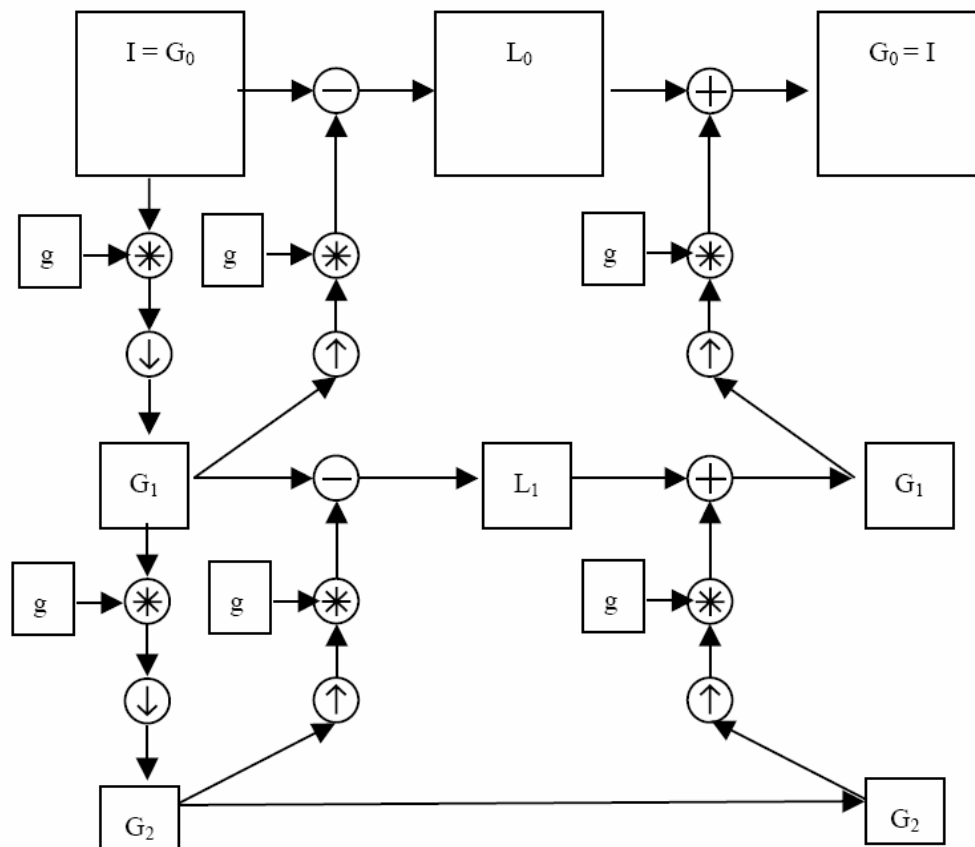


Fig 4.2 shows a Three-level Gaussian and Laplacian Pyramid

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 then 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 then the level L that specified, the local property of pixels on level L will be clustered to obtain the desired number of segments. (The Threshold2 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


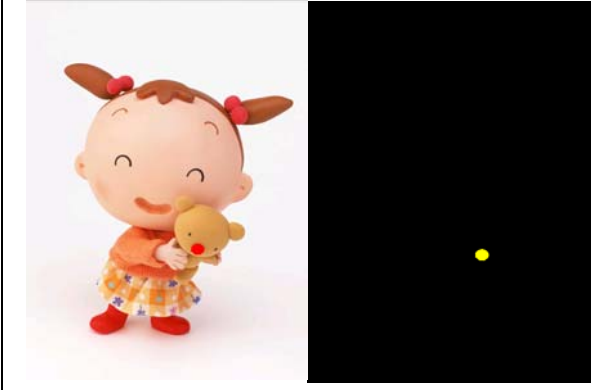



The second parameter restricted the variance of the value of pixel within the same region in the adjacent level. Any segment A and B belong to the same cluster if $d(A,B) < T_2$.

4.7.3 Estimation of the Parameter

The paper [11] suggested that the threshold1 is less significant than Threshold2 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 threshold1 and 50 to 70 for threshold2 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 Analysis

At the beginning of first semester, we have put our focuses on finding an optimal segmentation method. We found and tried out different segmentation strategies such as Thresholding, Pyramid Segmentation and Edge detection in an attempt to attain a method that can be fit to all general images. However, due to the fact that real images consist of multiple layers of stochastic element, such as texture, random point, line, curve, graph, region which are impractical to recognize by one single method, we have made a decision to shift the direction of our investigation. Instead of searching for good segmentation method, we are now searching for good segment.

This change of direction is based on the sense that we are not required to segment the whole image. The game property of PhotoHunt allows us to get only a few segments from one image since five differences are needed to be created. Therefore, any segmentation method, even for those sometimes return undesirable outputs, is considered as feasible as long as it provides five output segments after the screening process of the Segment analysis module.

The main purpose of image analysis module is to screen out the undesirable segments among the outputs created by the segmentation module. This screening process is controlled by several parameters which reflect the color properties of the segment and its surrounding.

This module also mimics the decision making process of human in the traditional

PhotoHunt Game. With the control parameters computed in the analysis, it can give suggestion on type of change (e.g. Elimination, Color change) to be made to the segment. The module also provides segment information for the later use in the elimination module.

This image analysis module mainly focuses on the investigation of color, intensity and brightness of the region of interest. It studies the color similarity to ensure significant color change to the region and screen out obviously bad region. It is admitted that the module is unable to completely sort away all inappropriate segments, yet it considerably improves the generation result. The module enhances the engine capability to achieve our rule of “Not obvious yet discoverable”.

5.1 Parameters

We have designed four different parameters to determine the property of the segment. In order to understand and calculate these parameters, we will first go over some of the graphics representation first.

The Segmentation Module output a segment by a reference diagram (Black and Yellow shown in the middle diagram below). Then, the engine will make use of this reference to generate a minimum-area box engulfing the segment. All the pixels in the box are categorized into three groups: object, background and offset.

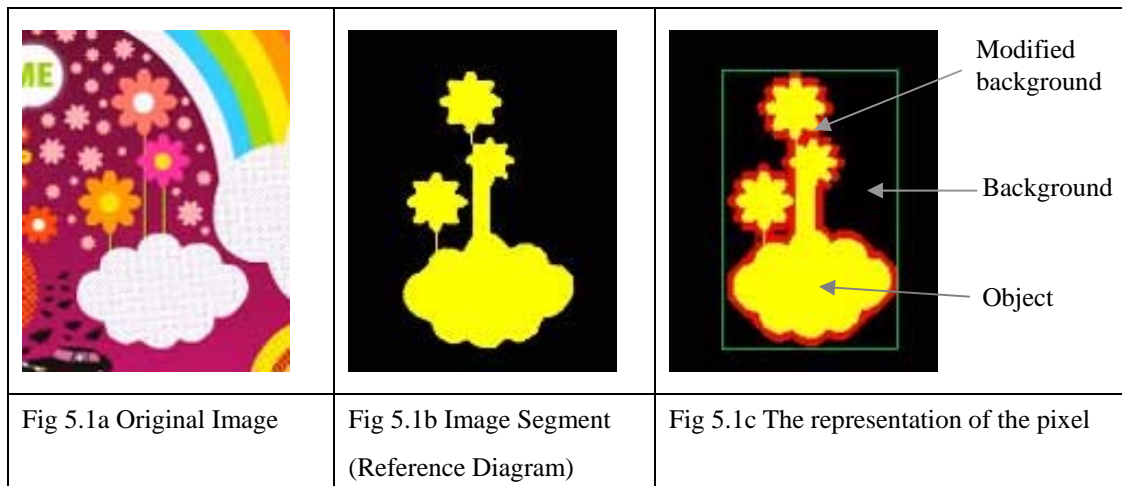
Object: All pixels in the returned segment are regard as an object pixel.

(As shown in yellow in below image)

Background: In the initial design, all the pixels in the box excluding the object are considered as background.

Modified background: for more accurate result, the engine generates an offset by expanding the segment by 2 pixels. All the background in the below content refer to this offset.

(As shown in red in below image)



5.1.1 Object and Background Average

The average brightness characteristic of a selected region is calculated as the sample mean. For the further computation of variance, we will use RGB color model here. Since the color we are considering is in 3-dimensional, we applied two approaches to find out the sample mean.

(1) We calculate the magnitude of the property vector, and then compute the sample mean of the Euclidean distance.

Pitfall of this approach:

Since the magnitude of the vector is given by

$$\sqrt{r^2 + g^2 + b^2} \quad \text{where } r, g, b \text{ are the property of red green and blue channel respectively}$$

It may happen that two totally different vectors result in the same magnitude,

e.g. the vector $[255 \ 0 \ 0]^T$ (Blue) and $[0 \ 255 \ 0]^T$ (green) give the same magnitude= 255. This approach can only provide the average intensity of the segment, but ignore the RGB color model. This ignorance of the RGB color space may affect the result.

(2) We calculate the sample mean separately, thus three sample means for three channels. According to [15], this sample mean is given by

$$\mu = \frac{1}{n} \sum_{(x,y) \in R} c(x,y) \quad \text{for} \quad \begin{array}{l} R: \text{the region of interest(object/background)} \\ n: \text{the total number of pixel in } R \\ c(x,y): \text{characteristics at the point}(x,y) \end{array}$$

In order to alleviate the error caused by noise, we have modified the equation to $\mu = \frac{8}{n} \sum_{(x,y) \in R} c(x,y) / 8$. Six means, three for background and three for object will be worked out and proceeded to compute the standard deviation.

5.1.2 Object Standard Deviation

The sample Standard Deviation of the region R with n pixel is given by:

$$\sigma = \frac{1}{n} \sum_{(x,y) \in R} d_{x,y} \quad \text{where } d_{x,y} \text{ is the Euclidean distance of the property vector}$$

5.1.3 Background Mode

It refers to the statistic mode of the region of interest, which indicates the majority of color in the background.

5.1.4 Object and Background Neighbor Difference

The neighbor difference is computed by comparing the difference between the 4-neighbors (see fig 5.2a) of a pixel. The difference, D, of region R is given

by:

$$D_R = \frac{1}{4n} \left(\sum_{(x,y) \in R} P_{left}(x,y) + \sum_{(x,y) \in R} P_{right}(x,y) + \sum_{(x,y) \in R} P_{up}(x,y) + \sum_{(x,y) \in R} P_{down}(x,y) \right)$$

where

$$P_{neighbor}(x,y) = \begin{cases} 1 & \text{if } colorDiff_{neighbor} \leq \Delta \quad neighbor \in \{left, right, up, down\} \\ 0 & \text{otherwise} \end{cases}$$

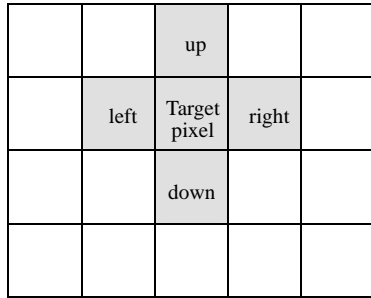
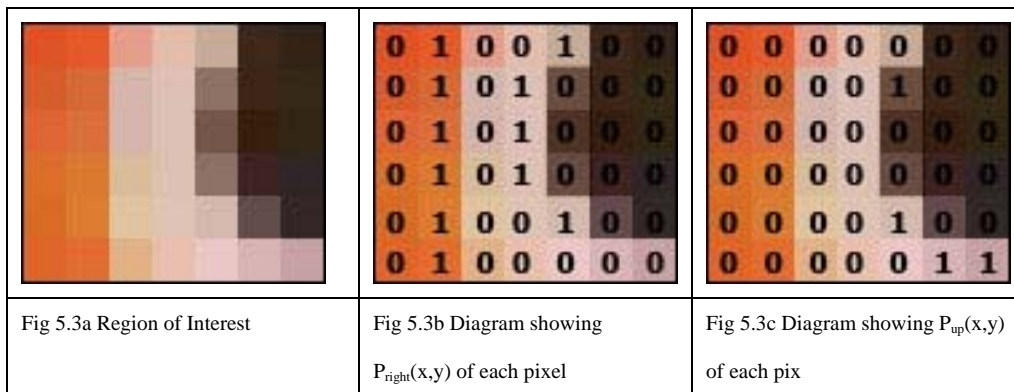


Fig 5.2a

Graphical Representation of the four neighbors of a target pixel

The color difference will be calculated using the method mentioned in [16]. The engine first converts the RGB color space to LAB color space which is more accurate in quantifying color difference, and work out the color difference as given by:

$$colorDiff_{neighbor} = \sqrt{(L_{x,y} - L_{neighbor})^2 + (a_{x,y} - a_{neighbor})^2 + (b_{x,y} - b_{neighbor})^2}$$



5.2 Parameter Usages and Applications

The previous chapter introduced all the necessary parameters for the screening and decision making process. After all the calculation of those parameters, the engine perform the following tasks:

- (1) Screening out segment with color that are too similar to background
- (2) Deciding the type of modification to be applied
- (3) Providing suggestion on replacement color for elimination

5.2.1 Screening out segment with color that are too similar to background










Due to the enormous variation of brightness and contrast of real world image, not all the segments that come from the segmentation module are suitable and proper. We assume that when the segment and its surrounding are similar in color and brightness, they are come from the same object and hence this is not an appropriate segment.

In order to check the similarity of the segment and its surrounding, the engine will compare the average color of object and background. With the use of the sample mean, the engine will decide whether to reject the segment.

$$Segment = \begin{cases} reject & \text{if } |ObjM_b - BgM_b| + |ObjM_g - BgM_g| + |ObjM_r - BgM_r| < \Delta \\ accept & \text{otherwise} \end{cases}$$

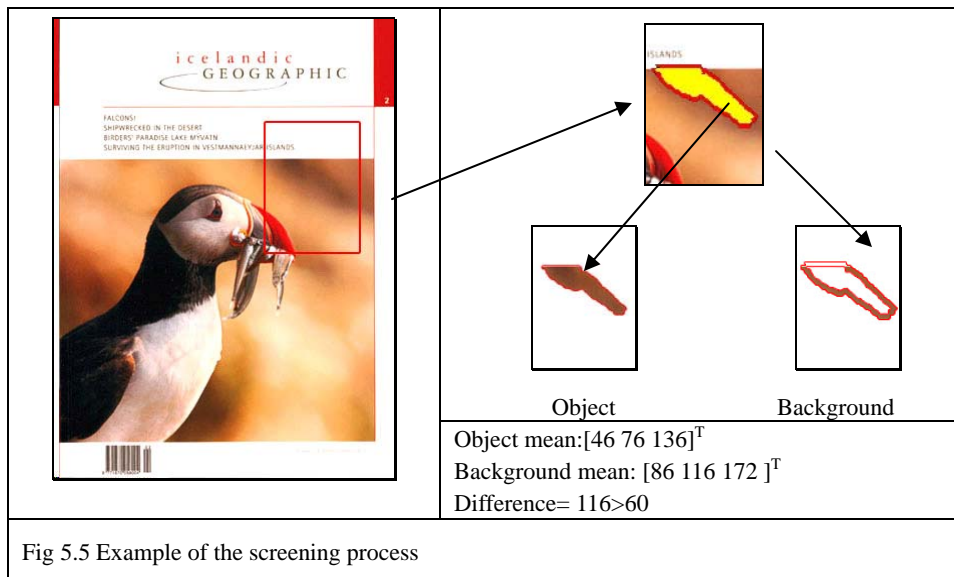
$ObjM_c$ is the mean of object in c channel

BgM_c is the mean of background in c channel (refer to 9.1.2 for details)

Example and data for the segment screening process			
			
	Fig 5.4a Segment1	Fig 5.4b Segment2	
			
	Fig 5.4c Segment3	Fig 5.4d Segment4	
Fig 5.4 Segment of original image and its segments			
	$ObjM : [66 \ 72 \ 72]^T$ $BgMe : [220 \ 231 \ 228]^T$ Diff : 475 > 60 Result : <u>Accept</u>		$ObjM : [68 \ 191 \ 200]^T$ $BgMe : [38 \ 58 \ 79]^T$ Diff : 284 > 60 Result : <u>Accept</u>
	$ObjM : [217 \ 92 \ 76]^T$ $BgMe : [207 \ 81 \ 65]^T$ Diff : 32 < 60 Result : <u>Reject</u>		$ObjM : [3 \ 83 \ 148]^T$ $BgMe : [10 \ 99 \ 163]^T$ Diff : 32 < 60 Result : <u>Reject</u>

Pit fall of comparing the object and background average and modification:

Since we are computing the sample mean of the background, any extreme data, even in a small portion, can notably amplify the effect on the value. This will lead to the result that sometimes obviously desirable region pass the checking. Below gives a practical example:



This is observed that the white color at the top increased the mean vector of the background, which forced the region to pass the test. However the majority of background was in brown, the difference is recorded merely due to the white leakage. This segment, which is detected because of the brightness different, should be rejected.

So as to improve the checking process, we replace the mean by the mode of background in the comparison. The new condition rule becomes:

$$Segment = \begin{cases} reject & \text{if } |ObjM_b - BgMode_b| + |ObjM_g - BgMode_g| + |ObjM_r - BgMode_r| < \Delta \\ accept & \text{otherwise} \end{cases}$$

$ObjM_c$ is the mean of object in c channel

$BgMode_c$ is the mode of background in c channel (refer to 9.1.2 for details)

Then above result give a new set of data:

Object mean: [46 76 136]^T

Background Mode: [56 88 152]^T

Difference= 36 < 60

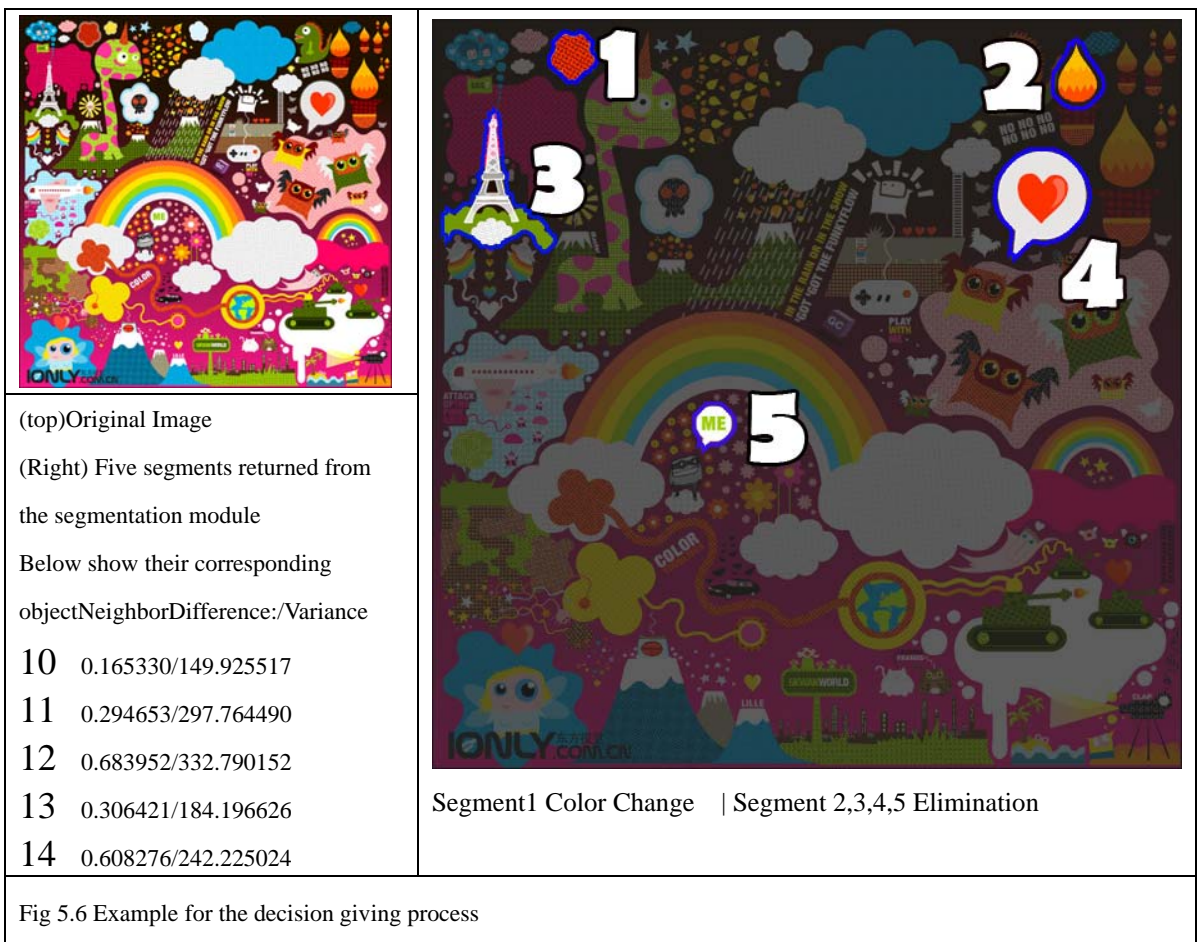
Thus, the segment is *rejected* in this new condition.

5.2.2 Deciding the type of modification to be applied

The engine classifies two types of modification only; they are color change and elimination. The classification is achieved base on the rule that color change must be applied to a monotonic colored object, i.e. with only one dominate color.

The color singularity is checked using the Object Neighbor Difference and Variance. Both variables give smaller value for a lighter variation of color within the region. The Object Neighbor Difference indicated the complexity of the images. On the other hand, object variance was a sign of the difference of color value. Combining these two indicators, we formed the rule:

$$Segment = \begin{cases} colorChange & \text{if } ObjectNeighbor \leq \Delta_1 \text{ and } Variance \leq \Delta_3 \\ Elimination & \text{otherwise} \end{cases}$$



5.2.3 Providing suggestion on replacement color for elimination

The module will hand over the following parameters to the elimination module to enhance the elimination effect:

(1) **Background Neighbor Difference**

This is the control used in the elimination module to determine the color to choose to replace the existing one.

(2) **Background Mode**

This mode enables the elimination to be carried out by using the color that looks alike the background.

(3) **Background Mean**

This is used when the background color is a texture or non-single colored.

A more detail description on the usage of these variables and the program flow will be shown in the Elimination Module chapters.

Chapter 6 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.

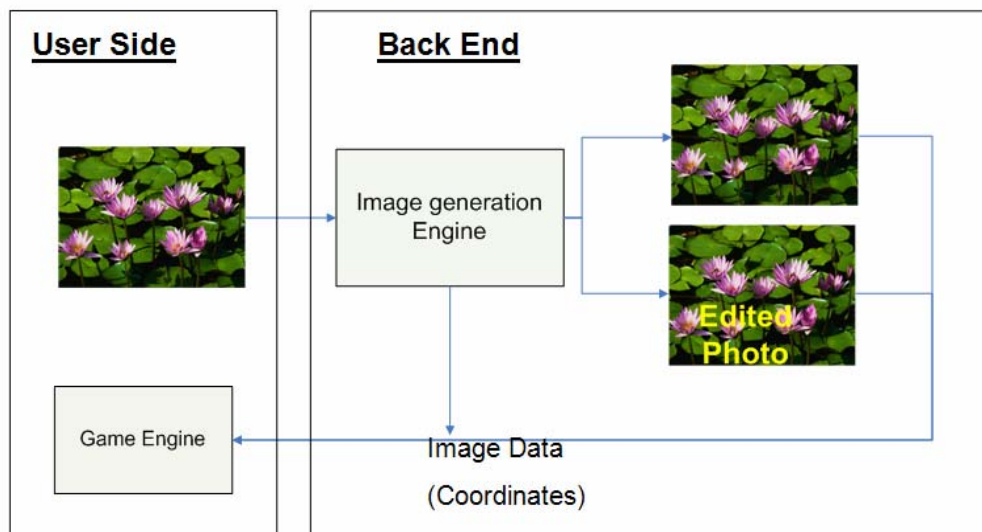


Fig 6.1 Flow Diagram of Automatic PhotoHunt generation

6.1 Processing Flow

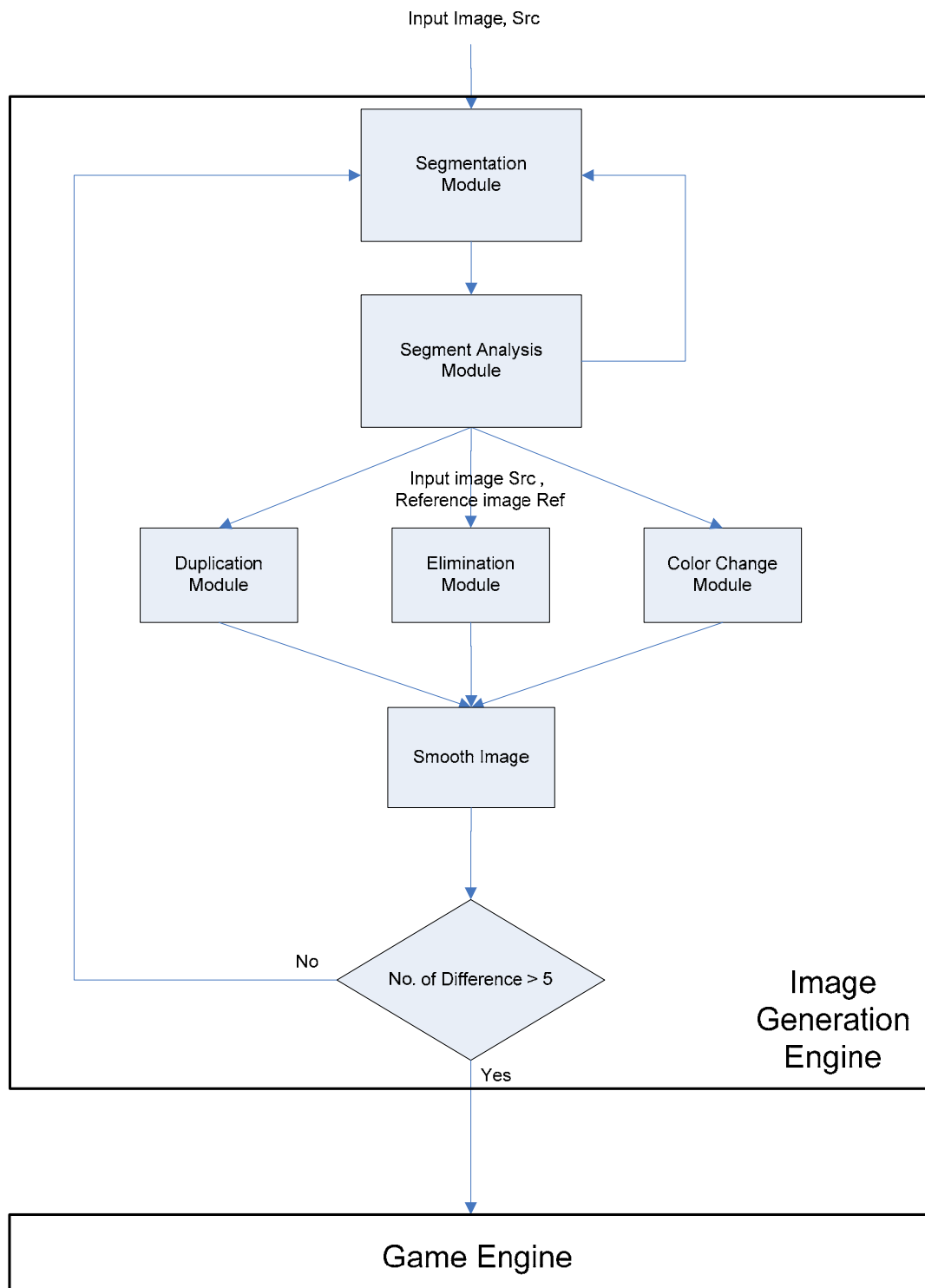


Fig 6.1 Flow Diagram for the Image Generation Engine

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.

6.2 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 .

6.3 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.

6.4 Color Change Module

The main function of this module is to perform the color change effect. 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

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

Dominant color segment: segment of more than 75% dominant color pixel

6.5 Object Appending Modules

This module would add up something to the image. What added up in the image is several predefined images, such as dates which usually appears in photos.

6.6 Image Warping Modules

This module is the process of manipulating an image such that a selected segmented area in the image have been significantly distorted. Our algorithm is using the forward mapping approaches which is a common technique for distortion.

6.7 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 7 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.

7.1 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 $I = 0, 1, 2, \dots$

7.1.1 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 7.3 and Fig. 7.4).. 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. 7.1 and Fig. 7.2). In conclusion, although this algorithm is simple, it is not a generic solution to our elimination process.



Figure 7.1



Figure 7.2

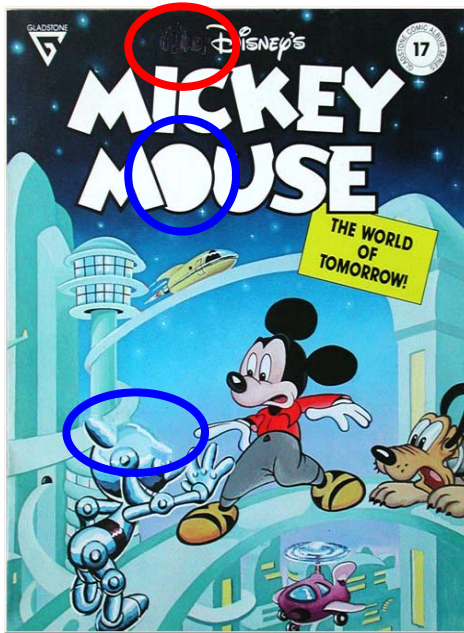


Fig 7.3

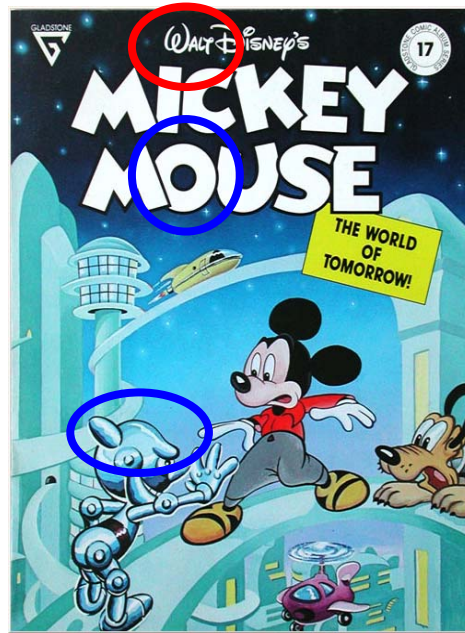


Fig 7.4

7.2 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. 7.5 shows the pixel N which has two horizontal boundaries P and Q.

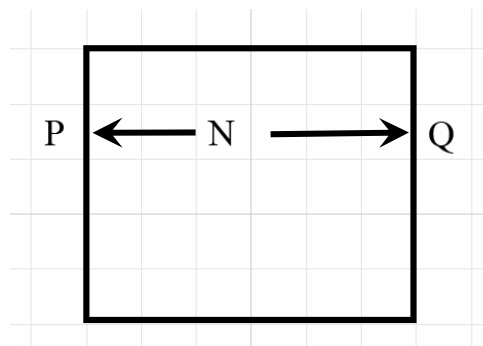


Fig 7.5

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 7.6

7.2.1 Test Result

This algorithm works even worse than the Direct Copy algorithm in most of the cases as shown on Fig. 7.7 and Fig. 7.8.

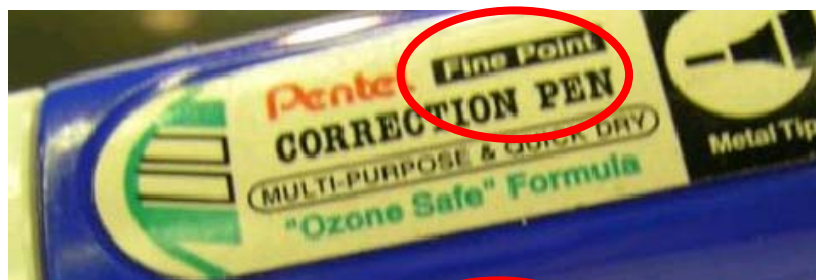


Fig. 7.7

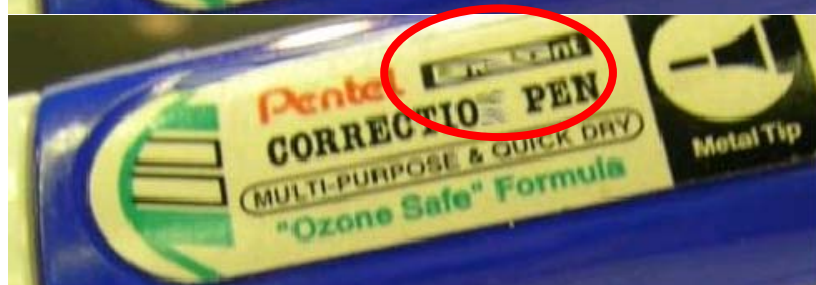


Fig. 7.8

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

The Fig 7.9 shows the noise at the boundaries of image. (The picture is produced by direct copy algorithm to show the effect)

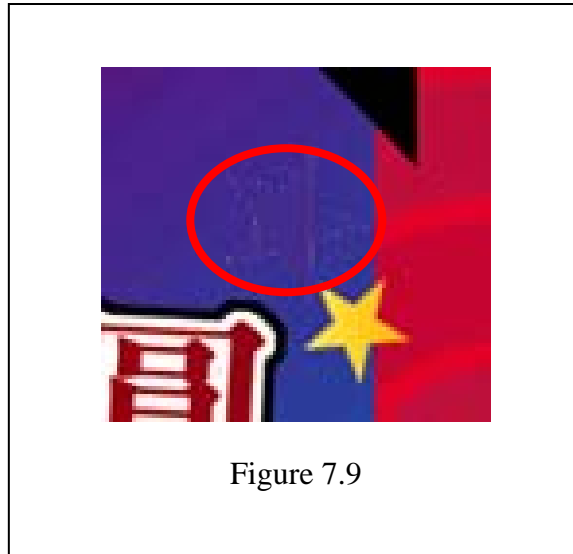


Figure 7.9

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. 7.10, 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. 7.10

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.

7.3 Nearest Boundary Algorithm

In order to determine correct color to be filled and solve the overlapping problem which shown on Fig. 7.10 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. 7.11)

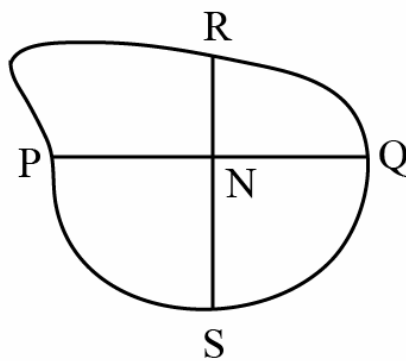


Fig 7.11

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

7.3.1 Testing Result

The result is similar to the one from Direct Copy algorithm. As we can see, in Fig. 7.12 and Fig. 7.13, some results are closed to the man-made image, but some are unacceptable.

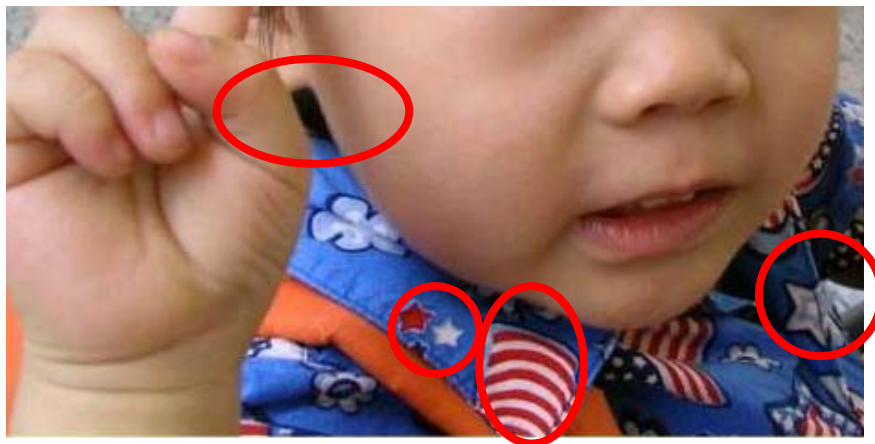


Fig 7.12



Fig 7.13



Fig 7.14



Fig 7.15

It is obvious that this algorithm has its strength in elimination. For instance, in Fig. 7.12 and Fig. 7.13, the eliminated region could even simulate the texture of the surrounding areas. However, it still needs improvement to increase its generality.

7.4 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

7.4.1 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 7.16



Fig 7.17



Fig 7.18

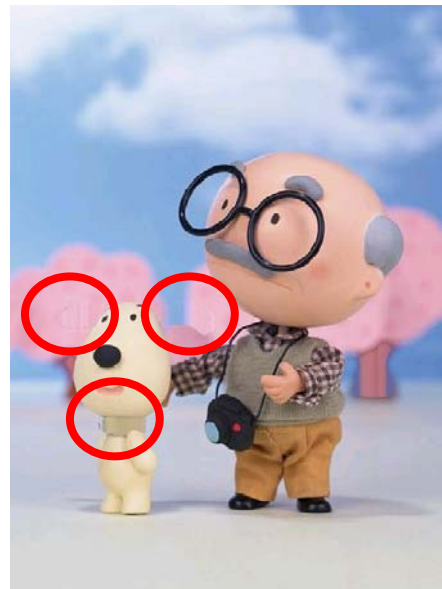


Fig 7.19



7.5 Hybrid Elimination Algorithm

We observed that there are no any elimination algorithms that can work alone to always generate good results. For an elimination algorithm to be optimized in generic problems, we introduced the hybrid elimination algorithm. This algorithm makes use of statistic data that came from the image analysis module to carry out the elimination process. These data provides more information about the surroundings and the segment's property. With the use of these data, a mixture of the previous elimination algorithms will be applied to ensure a more generic solution.

The algorithm requires the following information:

1. Background Mode
2. Background Neighbor Difference
3. Background Mean

The engine first check the background neighbor difference(2). If the background

neighbor difference is below threshold, λ , the background is classified as single colored. In this occasion, the object will be replaced by the color of background which is defined in the background mode(1).

If it is above threshold, the object is assumed to be in a textured background. Then, the engine will check the four areas around the segment(fig xx) and compute the mean of the each part.

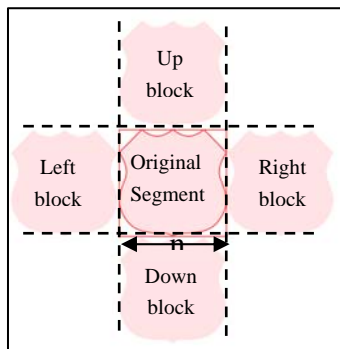


Fig xx the four area around the segment

In order to copy the similar texture from the surrounding to produce the disappear effect, the computed mean is compare against the background mean(3). The part with the most similar texture, that gives a smallest difference, will be selected to replace the segment using the direct copy algorithm. The pseudo code is given in fig 7.20.

```

Function hybridElimination(BgMode,BgNeighborDiff, BgMean)
  If BgNeighborDiff <  $\lambda$ 
    for each pixel in the bounding box
      if (x,y) is in the segment
        dst(x,y)=BgMode
      endif
    endfor
  else
    compute the meanLeft, meanRight, meanUp, meanDown
    minDiff= min(|BgMean-meanLeft|, |BgMean-meanRight|, |BgMean-meanUp|
      , |BgMean-meanDown| )
    for each pixel in the bounding box
      if (x,y) is in the segment
        if BgMean-meanLeft= minDiff
          dst(x-n, y)=obj(x,y); //copy from left
        if BgMean-meanRight= minDiff
          dst(x+n, y)=obj(x,y); //copy from right
        if BgMean-meanUp= minDiff
          dst(x, y-n)=obj(x,y); //copy from up
        if BgMean-meanDown= minDiff
          dst(x, y+n)=obj(x,y); //copy from down
        endif
      endif
    endfor
  endif

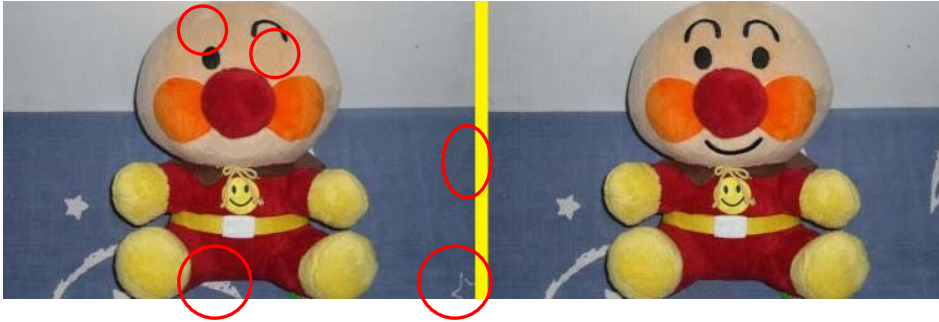
```

Fig 7.20 Pseudo code for Hybrid Elimination Algorithm

7.5.1 Testing Result

To mimic the complicated human thinking process, it should involve more conditional branches and rules. This algorithm, which takes more information into account, constructs a more desirable modified image under most of the circumstance.

The following give some examples generated from this algorithm, red circle indicating $BgNeighborDiff < \lambda$ and red circle indicating $BgNeighborDiff > \lambda$:



7.6 Conclusion

Among all the five algorithms, the Hybrid Elimination algorithm generally gives more adequate result over the others. This is due the handling of information acquired from the original image.

However, the latest algorithm has several limitations. Only considering four surrounding block might not be sufficient to determine the color to be filled. Also, the mean comparison provides limited pattern recognition, which is not 100% accurate.

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

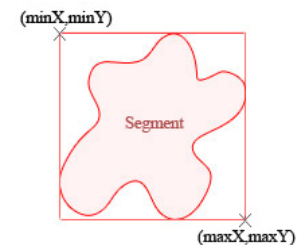
Chapter 8 Image Warping

The main function of this module is to apply the geometric transformation to a selected area of an image. There are different types of transformation, from simple operation like scaling, rotation to more complex operations such as warping and morphing. Among all these transformation, we choose to apply warping to the region on interest. The reason is that as we are not transforming the whole image, it generates less distortion between the transformed area and its surrounding. It also looks more natural in generic object.

In the following parts, we will outline the Image Warping methodology as well as some generated result to show the functionality of these module in a more concrete approach.

8.1 Image Warping Algorithm

The only input to this module is the reference diagram indicating the segment. To apply the transformation, the irregular-shaped segment is first given a rectangular bounding box. Also, the co-ordinates of the left top corner and right bottom corners, namely (minx, minY) and



(maxX,maxY) respectively, are checked for the computation of the coefficient in a transformation equation.

This module implements the image warping by employing the forward mapping approach. It iterates over each pixel (x,y) in the original image and compute a corresponding new position (x', y') , given by a pair of transformation equations

$x' = T_x(x, y)$ and $y' = T_y(x, y)$. The pseudo code for this forward mapping is

given below:

```

for (int x=minX ; x<maxX; x++)
    for( int y=minY; y<maxY;y++)
        If (x,y) is in the segment
            newX= T(x,y); //map the pixel to new location
            newY=T(x,y);
            destinationImg(newX,newY)=srcImg(x,y)
        endif
    endfor
endfor

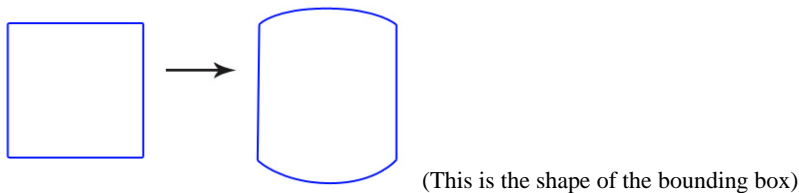
```

In a quadratic warp, the transformation can be express in polynomial in this way

$$x' = a_0x^2 + a_1y^2 + a_2xy + a_3x + a_4y + a_5$$

$$y' = b_0x^2 + b_1y^2 + b_2xy + b_3x + b_4y + b_5$$

In stead of the 2-D quadratic warp, the module transforms the original region to below form:



8.1.1 Transformation Equation

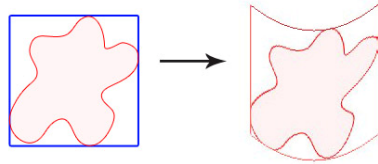
(A) Transformation along X-direction, $T_x(x,y)$

Since the transformation is only performed along y-direction, the x-coordinate of the region remain unchanged. Thus, $T_x(x,y): x'=x$

(B) Transformation along Y-direction, $T_y(x,y)$

To estimate the coefficient of the transformation equation, the complete transformation is decomposed into two parts, quadratic and linear.

B1. The segment will go through a quadratic stretch before it is transformed to the final shape. This intermediate transformation will have the following output.



This transformation is obtained by giving the y direction a displacement of value that varies as x changed. We set the equation

$$y' = y + \Delta_y \quad \text{and} \quad \Delta_y = b_0x^2 + b_1x + b_2$$

When moving along the x-axis, the engine will need to obtain a maximum at the middle of the segment (i.e. $x = (\max X + \min X)/2$) and no change at both sides ($y=0$). As the quadratic should have $\max X$ and $\min X$ as the root, the equation become:

$$(x - \max X)(x - \min X) = 0$$

$$x^2 - (\min X + \max X)x + \max X \min X = 0$$

It gives $\Delta_y = w(x^2 - (\min X + \max X)x + \max X \min X)$.

If the maximum displacement is set to be Δ_{\max} pixels, substitute

$(\frac{\max X + \min X}{2}, \Delta_{\max})$ to the equation

$$\Delta_{\max} = w\left(\left(\frac{\max X + \min X}{2}\right)^2 - (\min X + \max X)\left(\frac{\max X + \min X}{2}\right) + \max X \min X\right)$$

$$w = \frac{\Delta_{\max}}{\left(\left(\frac{\max X + \min X}{2}\right)^2 - (\min X + \max X)\left(\frac{\max X + \min X}{2}\right) + \max X \min X\right)}$$

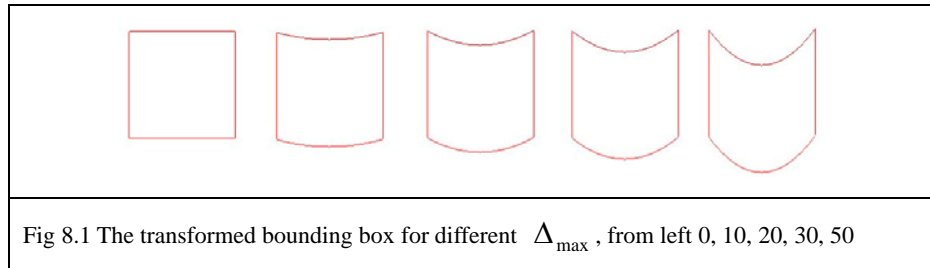
$$w = \frac{4\Delta_{\max}}{-(\max X + \min X)^2 + 4\max X \min X}$$

Thus, the equation for the quadratics is:

$$y' = y + wx^2 - w(\min X + \max X)x + w \max X \min X$$

where $w = \frac{4\Delta_{\max}}{-(\max X + \min X)^2 + 4 \max X \min X}$

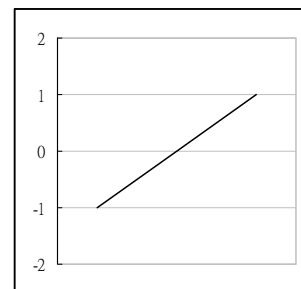
for a given Δ_{\max} that indicating the curvature of the transformation.



This quadratic transformation only gives a 1-direction displacement to move the pixels downward by the given magnitude. To achieve the final desirable effect, another weight, representing a directional vector, is designed to multiplied to the displacement, Δ_y .

B2. This transformation divided the image into two, upper half and lower half. The transformation is designed so produce the effect that, all the pixels above the upper half would have a negative displacement, and vice versa. A linear transformation is employed so as to reduce the distortion effect which makes the modification less natural. The linear equation represents the ratio of the displacement in different value of y . This ratio, r , should have the following property:

1. When $y = \max Y$, $r = 1$
2. When $y = \min Y$, $r = -1$
3. When $y = \frac{\max Y + \min Y}{2}$, $r = 0$



We than can use the above data to form a linear equation in term of y to compute the ratio r . The transformation is given by:

$$\frac{1 - (-1)}{\max Y - \min Y} = \frac{r - 1}{y - \max Y}$$

Simplify to:

$$r = \frac{2y - \max Y - \min Y}{\max Y - \min Y}$$

Combining B1 and B2:

We get the required transformation function, $T_y(x,y)$, by multiplying the ratio to the displace, i.e.

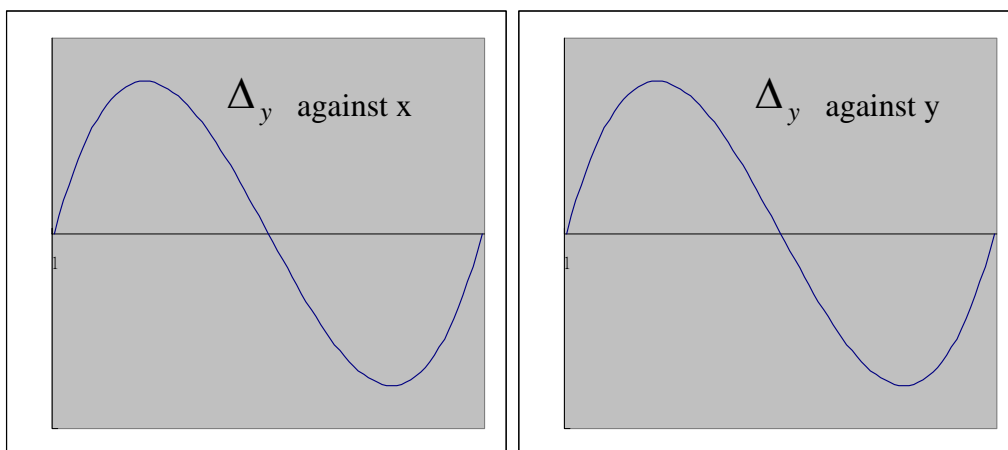
$$y' = y + wrx^2 - wr(\min X + \max X)x + wr \max X \min X$$

By expanding it, we get:

$$y' = y + \frac{w(\max Y + \min Y)}{\max Y - \min Y} x^2 - \frac{2w}{\max Y - \min Y} x^2 y + \frac{2w(\min X + \max X)}{\max Y - \min Y} xy + \frac{w(\max Y + \min Y)(\min X + \max X)}{\max Y - \min Y} x + \frac{2w(\max X \min X)}{\max Y - \min Y} y + \frac{w(\max Y + \min Y)}{\max Y - \min Y}$$

where $w = \frac{4\Delta_{\max}}{-(\max X + \min X)^2 + 4 \max X \min X}$

To visualize the effect, we plot the diagram of Δ_y against x and against y. The combined equation gives the following graph:



We have also tried to convert this transformation to a 2-directional one by replacing x by y and y and x. However, it is observed that one direction warping

result can be found in the real word more easily.

Fig 10.2 The transformation equation

$$x' = x$$

$$y' = y + \frac{w(\max Y + \min Y)}{\max Y - \min Y} x^2 - \frac{2w}{\max Y - \min Y} x^2 y + \frac{2w(\min X + \max X)}{\max Y - \min Y} xy + \frac{w(\max Y + \min Y)(\min X + \max X)}{\max Y - \min Y} x + \frac{2w(\max X \min X)}{\max Y - \min Y} y + \frac{w(\max Y + \min Y)}{\max Y - \min Y}$$

$$\text{where } w = \frac{4\Delta_{\max}}{-(\max X + \min X)^2 + 4 \max X \min X}$$

8.2 Image Warping Results

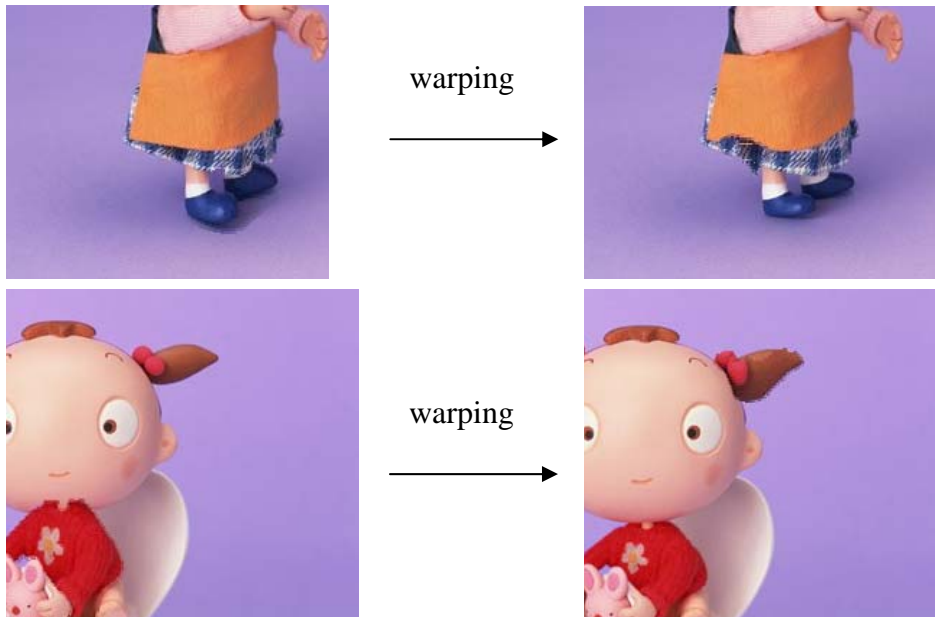
Below show some of the output after the image warping,



It can be seen that the eye right eye of the cat is warped.



There are several different created by warping in the picture:



The head of the man is warped.

Chapter 9 Appending Object

The main function of this module is to append an object from our database to the original image. In traditional PhotoHunt, adding objects are based on the knowledge of the image. For example, glasses are sometimes added to men, but never being added to a building. As the content of the input image is unlimited, it is impossible to give a general recognition to all types of object. Thus, the engine cannot perform the object recognition task as human; the selection of object to be added is carried out in a random bias. However, all the objects in our database is an generic one, which mean they can be apply to any kind of pictures.

9.1 Appending object algorithm

All the stored objects in the database are in a single color background. To append them to the original image, the objects is copied pixel by pixel. Below show the pseudo code:

```

Function appendObject(Img originalImg, Img appendImg, color bgcolor)
for (int x ; x<objectwidth; x++)
    for (int y; y<objectheight; y++)
        if obj(x,y) is not of background color
            originalImg(x+displacement X,y+ displacementY)=appending(x,y)
        endif
    endfor
endfor

end function appendObj

```

9.2 Appending object result

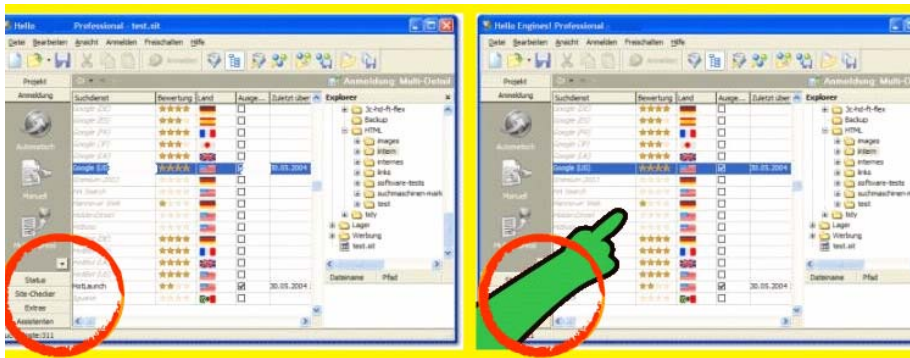
(1) Adding date to the image



(2) Adding pointing finger



(3) Adding monster hand



Some sample images in the database:



9.3 Limitation

The adding of object is quite obvious in nature. However, for the reason that adding the object increased the entertainment level and the variation of modification, this module still worth to sustain.

As intensive player of the game may trained to detect these appending object effect, the database of the object should be large enough. Also, the probability of appending an object is set to a very small value, so as to minimize the chance for a single player to play the same effect twice.

Chapter 10 Semi-Automatic Generation Program

10.1 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.

10.2 Interface

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

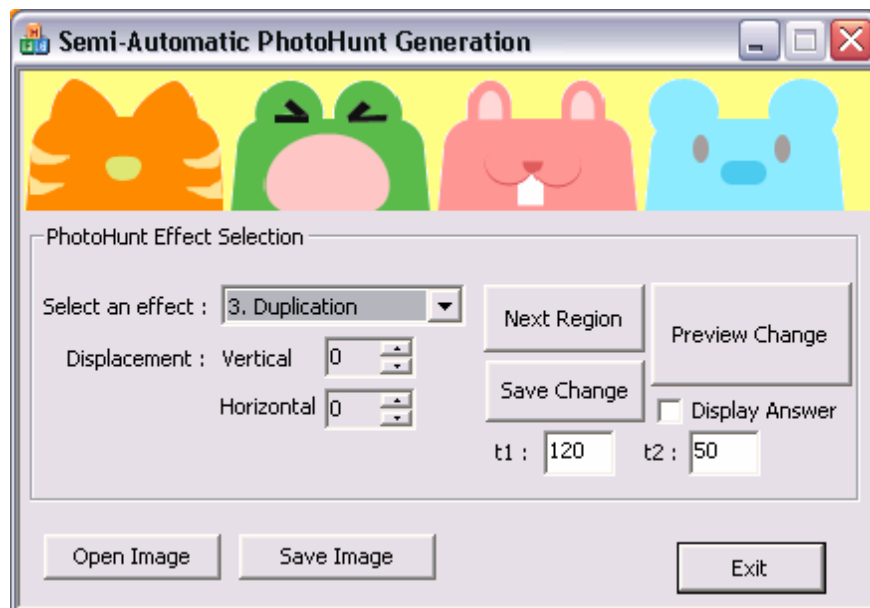


Fig. 10.1

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 : <ol style="list-style-type: none"> 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

10.3 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

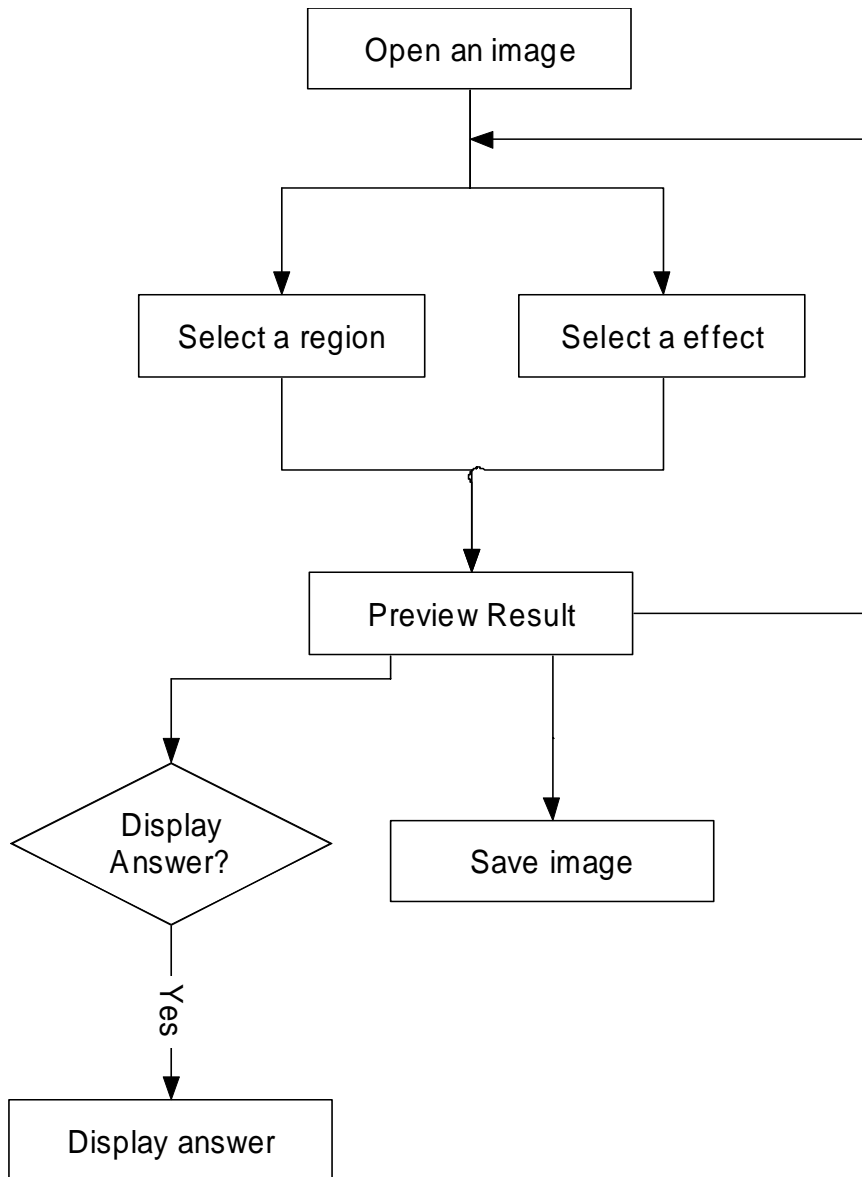
IGE while the IGE will give the results back to SAGE to display.

10.3.1 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.



10.3.2 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. 10.2

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.

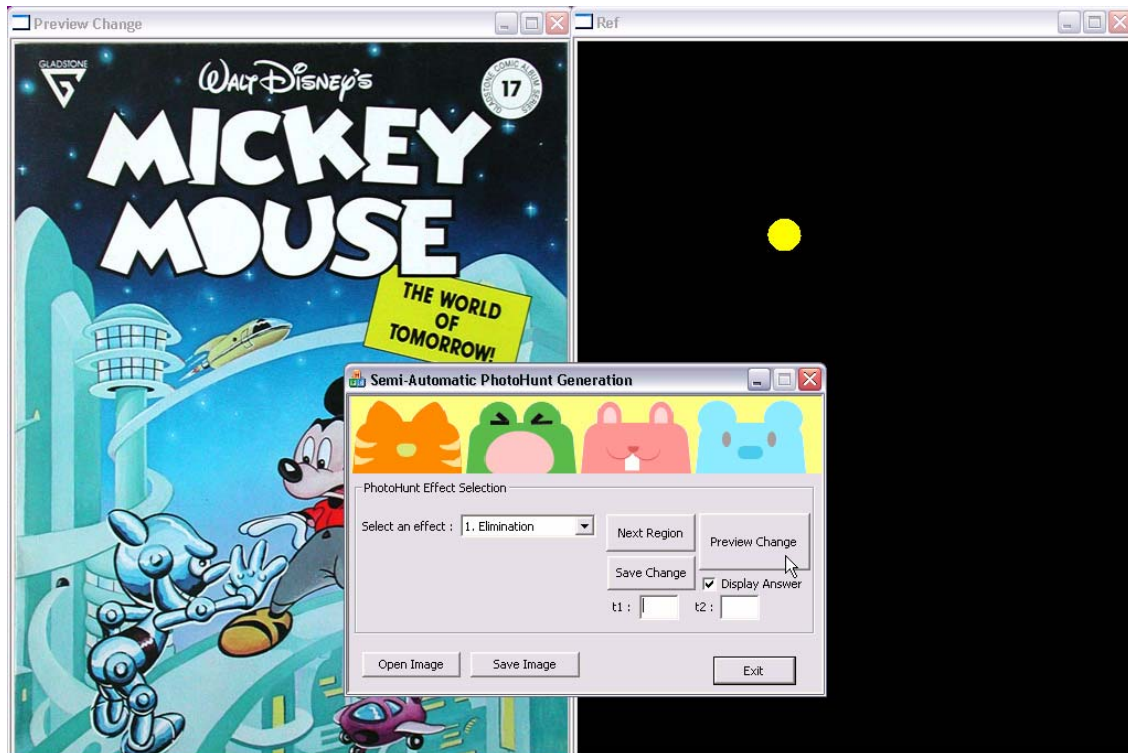


Fig. 10.3

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.

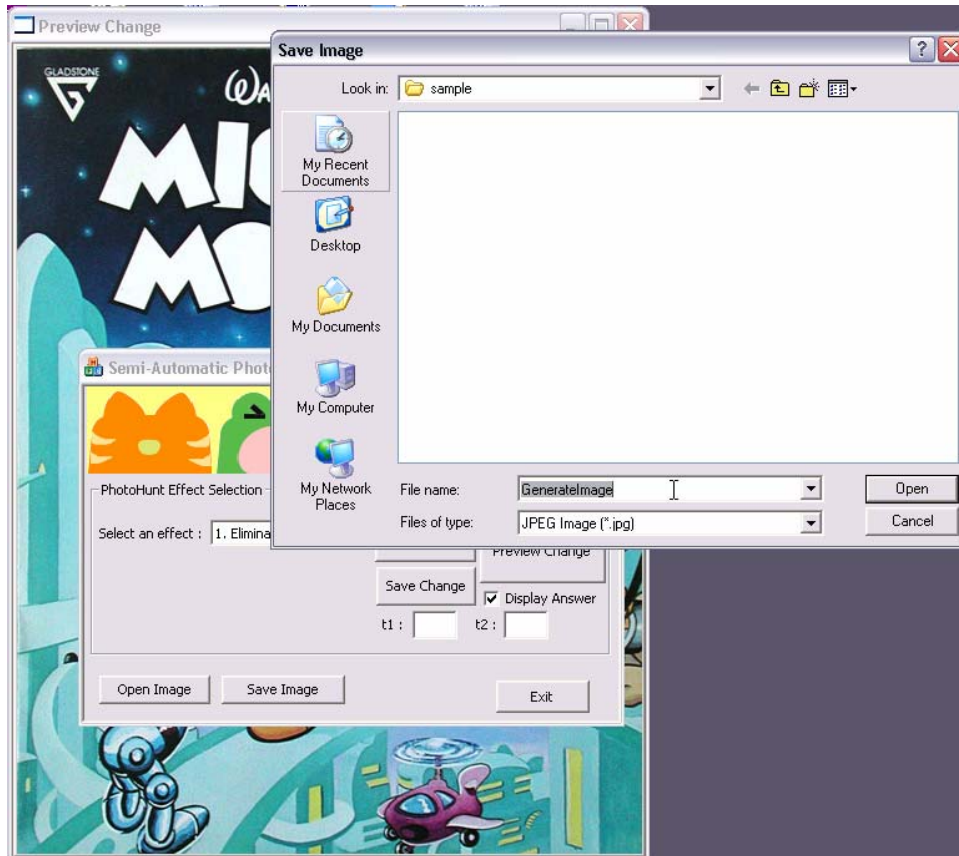


Fig. 10.4

Chapter 11 Game Engine

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.

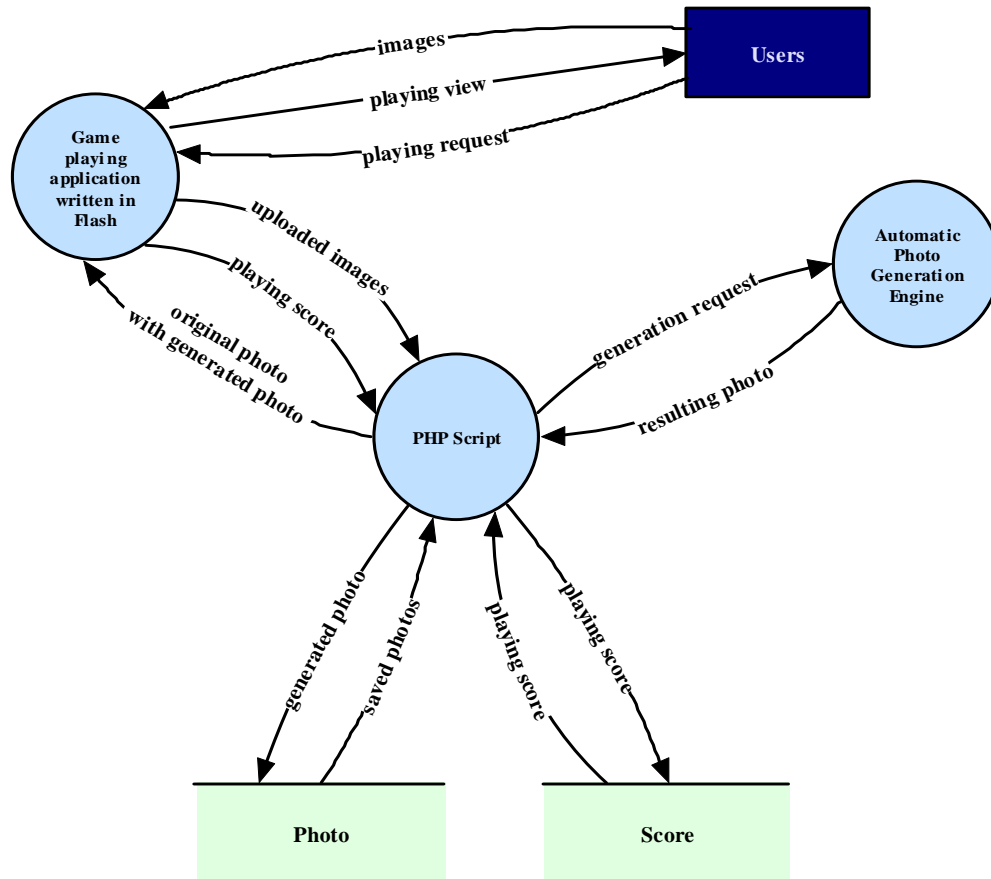
This Game Engine comprised of two parts. One is the game interface which is implemented by Flash. It is the only part that could be seen by users. Through the interface, users could upload an arbitrary image. Two generated images will then return to the interface. The game is then ready to play.

Apart from visible component, the backend is supported by a PHP script behind. It handles all the operations of files while uploading such as saving the image into proper location, recording scores and processing multiplayer match. 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 and challenge mode. In the following sections, a brief description of the engine will be given.

11.1 Overview

The following diagram shows the overview of the engine :



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.

11.1.1 Processing User Requests

When the PHP script receives 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 of the records and send the images back to the flash application. Once again, the flash application displays the images exactly the same way as above.

11.1.2 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.

11.2 Front-end

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

11.2.1 Main Menu

The following image shows the front page of game (Fig 11.1). There are four important buttons in the middle, including “Single Player”, “Multiple Player”, “Challenge Mode” and “About”. Players should entering the corresponding sections by clicking on any one of the button.



11.2.2 Single Player

Fig 11.1

Fig 11.2 will be shown when user has clicked the “Single Player” button in the main menu. A green box popped up and it contains few boxes. The upper white box allows the user to enter an identifier as the player name. The button located in the middle, which is labeled “start”, could started the game with random image picking mode by only one click.

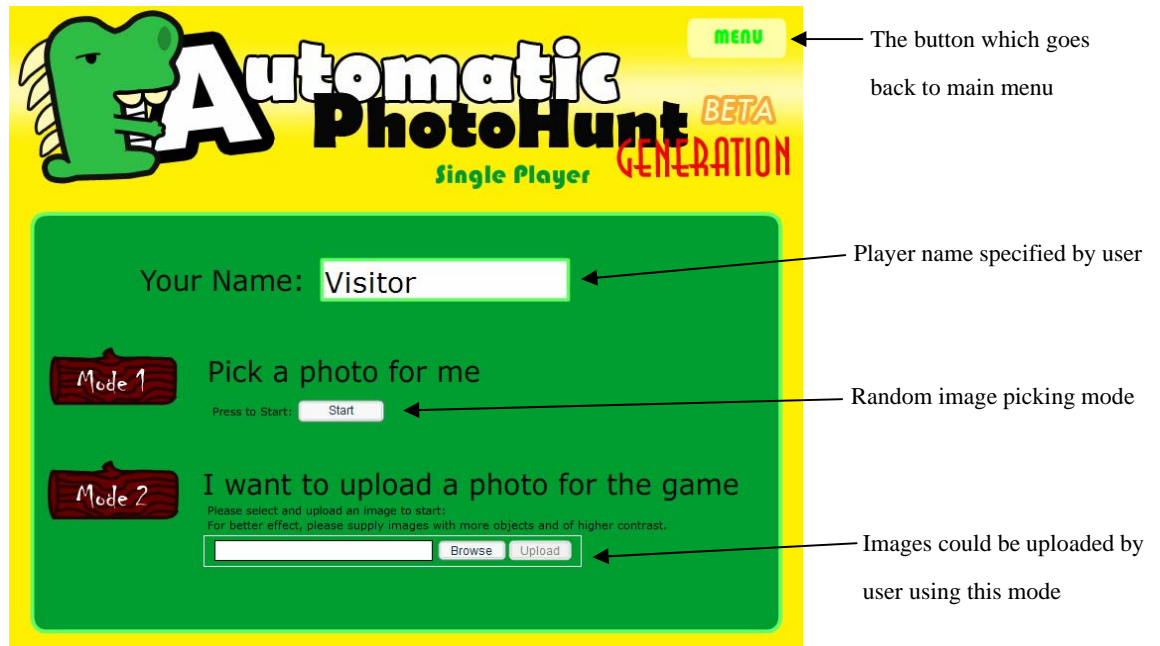


Fig 11.2

The bottom white box associated with two buttons, namely “Browse” and “Upload” as shown in Fig 11.3. Images could be uploaded by clicking these buttons. The “Upload” button is disabled by default.



Fig 11.3

When the “Browse” button is pressed, a Select-File-To-Upload window will appear (Fig 11.4). User can then select image from the box. Only JPEG format is supported in current stage.

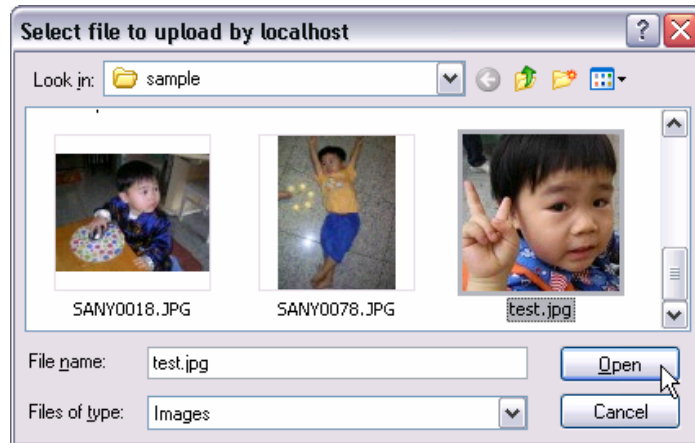


Fig 11.4

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. (Fig 11.5)



Fig 11.5

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. (Fig 11.6)



Fig 11.6

No matter mode 1 or mode 2 has been chosen, the game engine jumps to the next stage, and starts the game (Fig 11.7). Two generated images show up in parallel. Player clicks on the differences of the images will result in a red circle indicating the player is just doing right. There are some more things that come with the images. Score, Timing bar and two special buttons (Time, Tips) are placed above the generated images, while a counter which indicated the number of differences has not yet been discovered in the bottom of the screen.

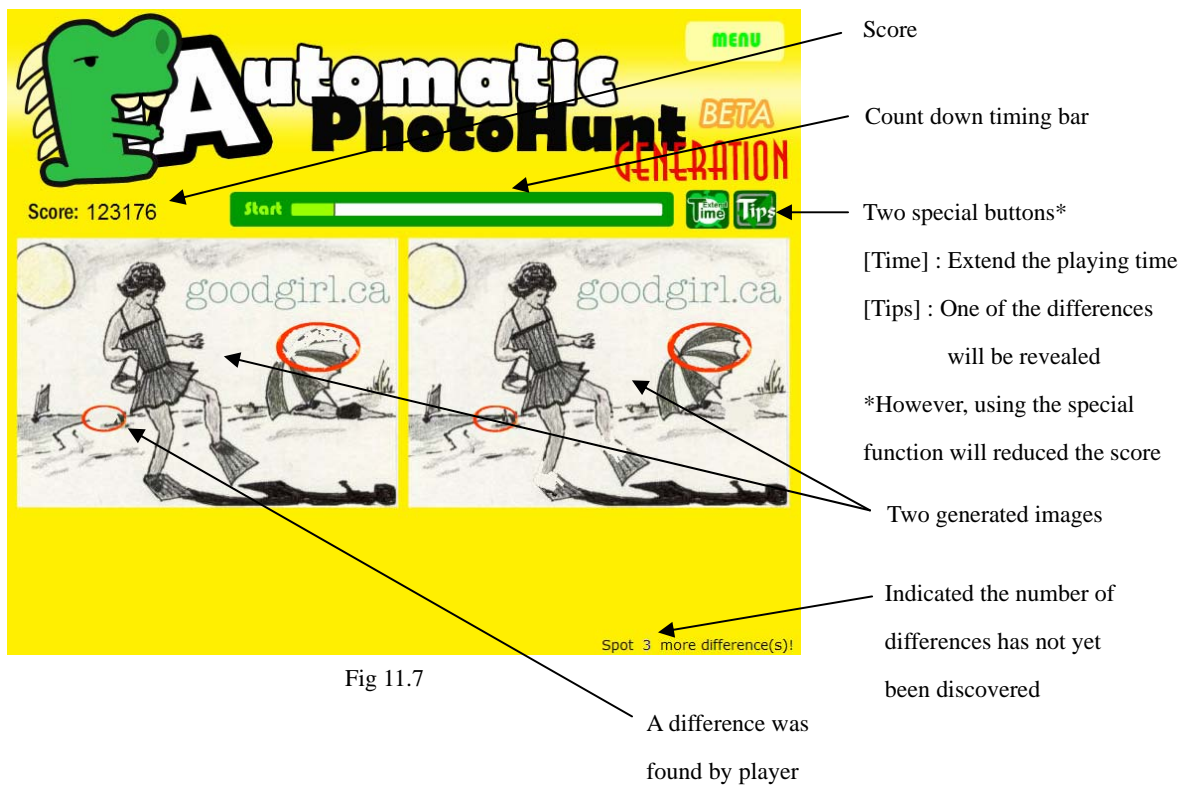


Fig 11.7

Whether the time is running out or all differences have been exposed, a summary window will be shown, and tell the performance. Players could also give comments on the generated images in the box provided in the left bottom.

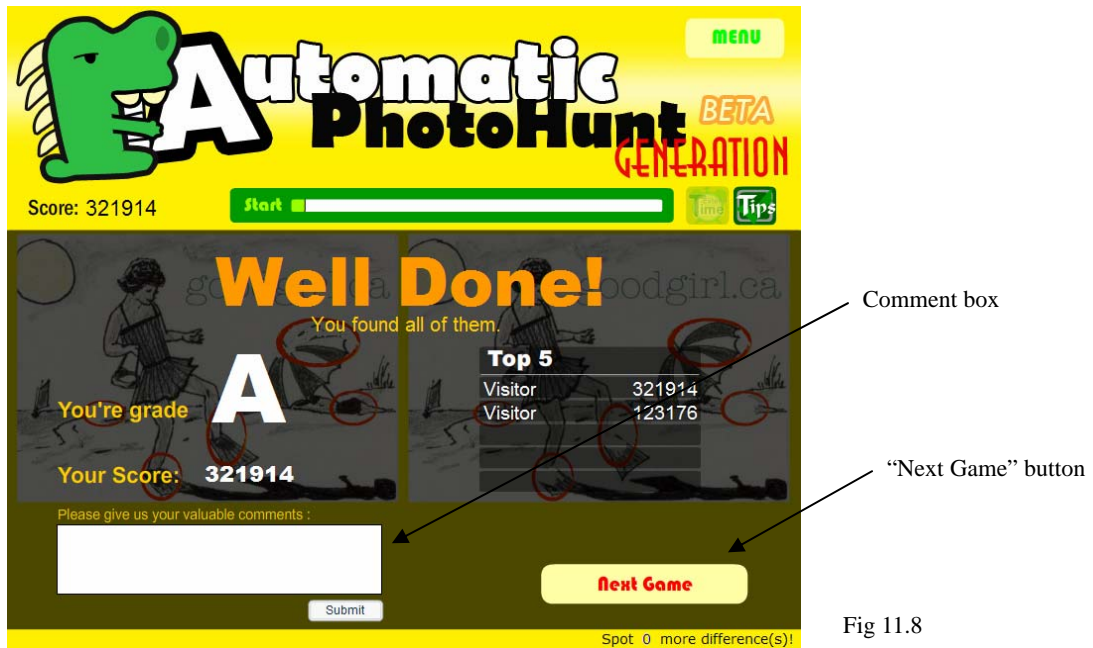


Fig 11.8

11.2.3 Multiple Player

The screen shown when the “Multiple Player” button was clicked has some differences from the single player mode, while the major parts shares a lots of common behaviors.

When a player enters the multiple player mode, the engine will automatically assign the opposite to him or her if there is some other players waiting for a match. However, if there is no player in the list, he or she needs to wait until someone comes. The following screen shot shows the waiting situation. (Fig 11.9).

The game will be started just after the engine finished the matching. The screen is almost identical to the single player mode (Fig 11.3). Several minor differences have been shown on Fig 11.10.

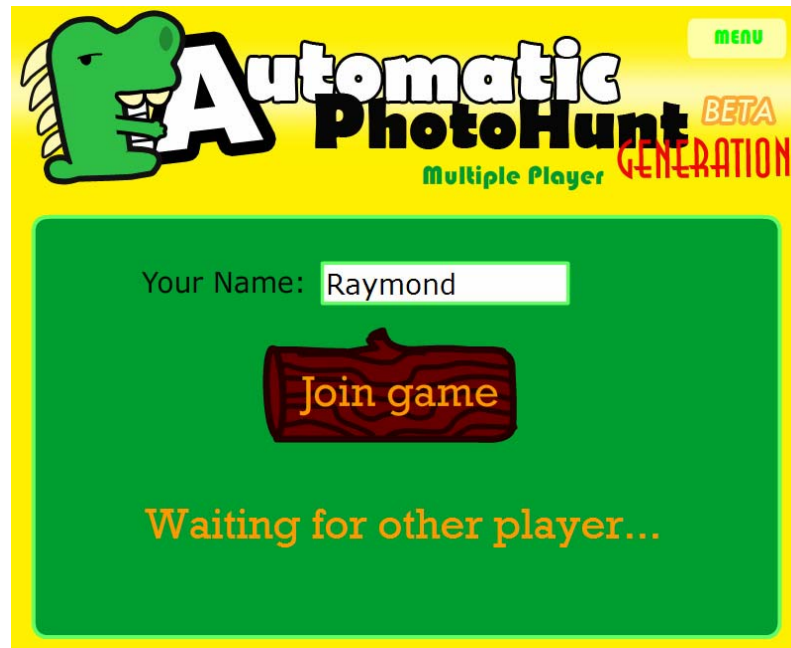


Fig 11.9

While the single player mode contains two special buttons, including “Time” and “Tips”, the multiple player mode only features the “Time” button but the number of this button is increased from one to two. There are two types of circles in the screenshot, one is in blue and another is in red. The red one illustrate the differences found by the player himself which is the same as in single player mode. The one in blue is indicated the difference found by the opposite. In the footer of the screen, player name and score of the opposite is added for the players’ reference.

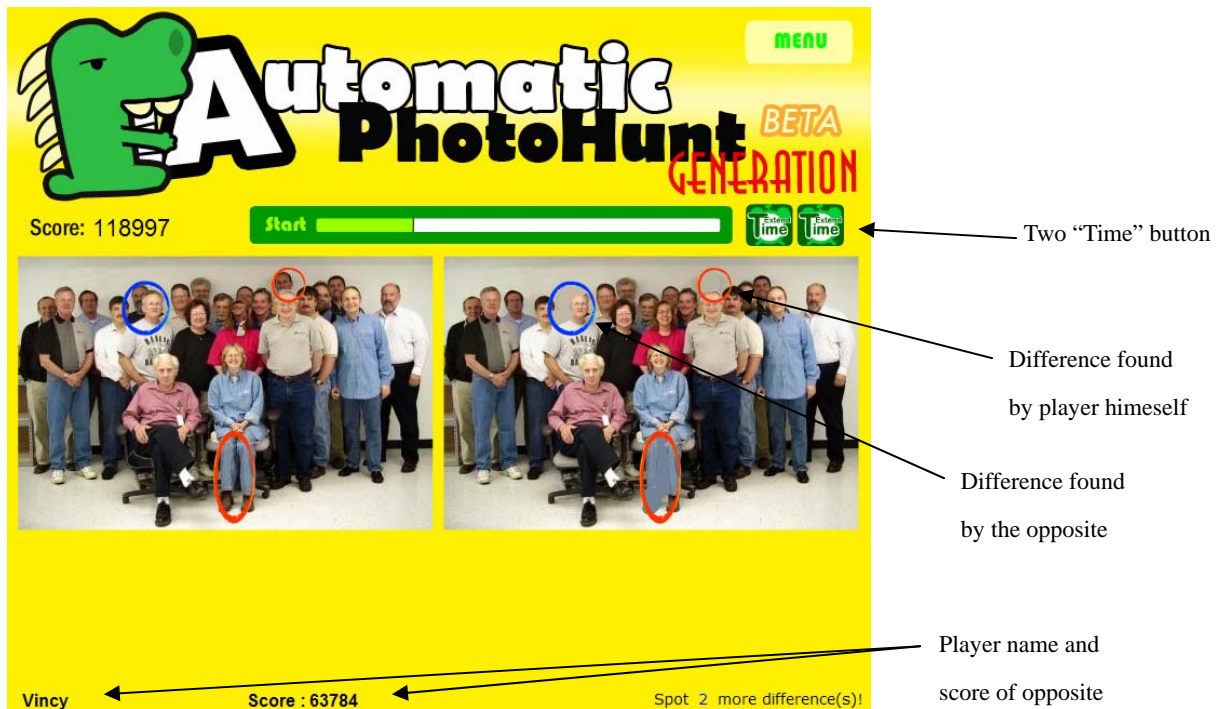
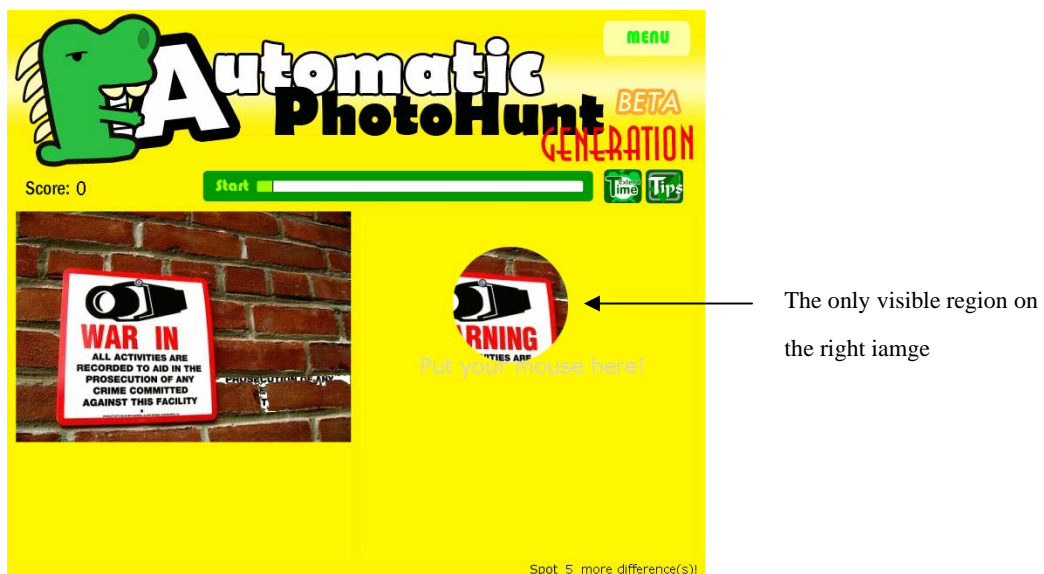


Fig 11.10

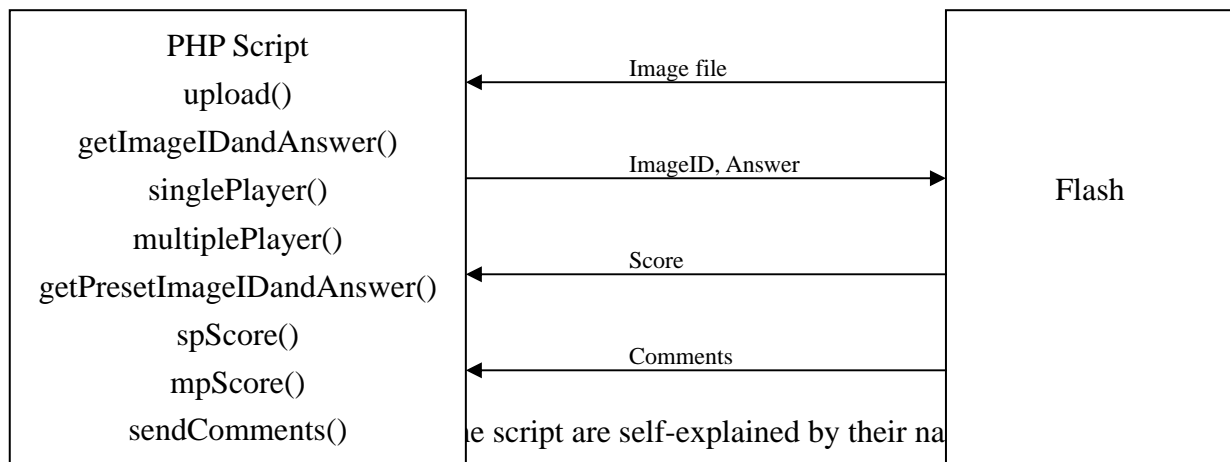
11.2.4 Challenge Mode

Based on the single player mode, challenge mode has been developed to provide more fun factors to advanced players. Instead of two images are displaying simultaneously, in challenge mode, only one images are visible. For another image, a circle region could be viewed only when the mouse cursor is moved on top of the image. Fig 11.11 give the idea of the challenge mode.



11.3 Backend

The game engine was mainly supported by an important component, the PHP script. The script provides most of the major functions, including uploading image, random image picking, score retrieving and storing, and saving comments. The following figure illustrate the relationship between the front-end and backend, which involving lots of message passing.



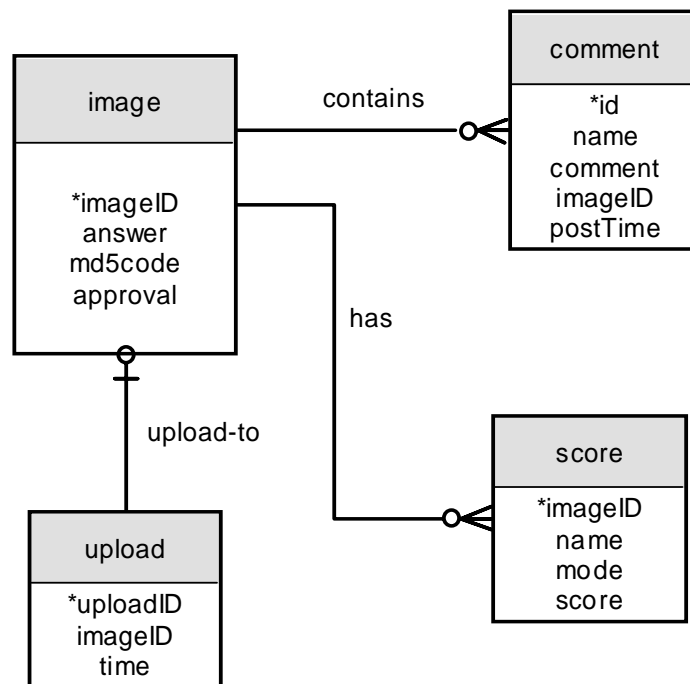
below gives a brief description for each of them.

Function Name	Return Type	Descriptions
upload	N/A	Handling upload action, generating “modified” image by executing Generation engine and save the result
getImageIDandAnswer	String	Return the result of generation to flash
singlePlayer	String	Randomly pick an existing image for playing
multiplePlayer	String	Return image that is determined by multiple player socket
getPresetImageIDandAnswer	String	Return the result of generation to flash (For admin use)
spScore	String	Saving score from single player mode
mpScore	String	Saving score from multiple player mode
sendComments	N/A	Saving comments from player

11.4 Database

The game engine involves database where stores the information of images, player scores and comments. A popular open source database, MySQL, is used to provide high performance services to the engine.

There are four tables in total in the database, namely “image”, “score”, “comment” and “upload”. The Entity Relationship Diagram and Data Dictionary have been shown below.



Entity Relationship Diagram

Appendix 3 shows the Data Dictionary of this Entity Relationship Diagram.

Chapter 12 Evaluation

While the generation engine is being developed, any outcome from the algorithm is adjusted by us only. However, that measure is intangible, which may result in deviation. The only way to quantify the quality of result is conducting a survey. We have conducted a survey, which is bundled with the internal testing. Appendix 5 shows the advertisement and responses from players. For tangible results, we tried our algorithm under several conditions to obtain the variables such as performance and some other major constrains.

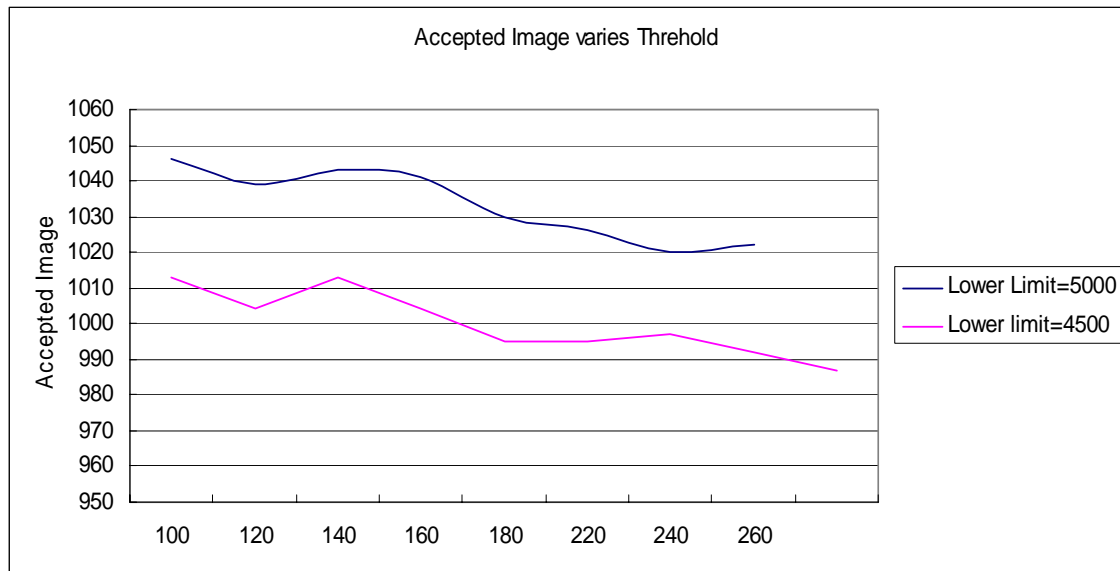
Upon the completion of the generation engine, we have evaluated its performance in order to optimize the processing time and segmentation thresholds which are resulting in different acceptance rate. We also evaluate the quality of images, which is directly affecting the result of generated images.

In the following sections, we show the evaluation result including processing time, acceptance rate, and the image quality. All of the data are obtained by using a Pentium 4 3.0G, 1G RAM machine, which is freshly rebooted with few essential background processes. Note that for the first fourth evaluations, the source data is presented in Appendix 4.

12.4 Acceptance Rate

The acceptance rate has a close relationship with the threshold. We have tried several combinations of upper limit and lower limit thresholds, and runs 1318

images to find out the optimal value of threshold while still holding an acceptable acceptance rate. The upper limit threshold is from 100 to 260 and lower limit is 5000 to 4500 in our case.



As we could see in the following line chart, a wider range of threshold implies a wider acceptance rate. However, a large lower limit means a smaller segment will be accepted that it is really difficult to recognize by human eyes. In another hand, a smaller upper limit leads to a larger segment which may be too big for photohunt.

To simplify the optimization problem, we are assuming the 75% acceptance rate is satisfactory, that is, around 988 accepted images. The corresponding x-axis is about 200 in upper limit and 4500 in lower limit.

12.5 Processing Time

The processing time evaluation is essential since user are expecting a generation

result immediately after uploading images. Thus the processing time could not be large because the user will not have patient to wait for it.

For the testing we have done in 10.1, the processing time is quite fast even with different combinations of thresholds. The fluctuation is small, from 00:14:15 to 00:17:51. As the result, we simply take an average to estimate the processing time;

$$\textit{Average Processing Time of 1318 images} = 00:15:50$$

Divide the total processing time by the number of images, we have

$$\textit{Estimated Processing Time of one image} = 0.721 \textit{ second}$$

Note that the result is obtained in a modern personal computer. We conclude that the result is adequate for application of photohunt generation.

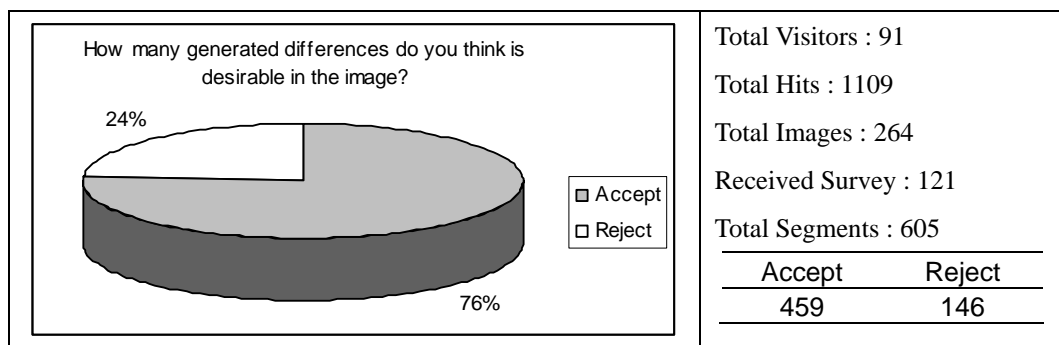
12.6 Image Quality

One of the major fundamental steps of our generation engine is to segment an image into objects that this process heavily depends on the condition of the given image. Support a blank image is provided to the engine to do the segmentation, no matter how well the algorithm is, our engine will certainly reject the image since it could not extract any of objects.

In order to identify the “good conditions” of image for the generation engine, we conducted a survey during the internal test, and concluded what characteristic a “good image” should share. In the version of internal test of the flash, a questionnaire will be popped after a player is finished a photo. It asked for “*How many generated differences do you think is desirable in the image?*” and the

answer range is from 0 to 5 since each of the generated image contains 5 differences.



The following graph shows result in this survey. The internal test comes with 264 preloaded images. At the end of the survey, the web statistic record 91 visitors and 1109 hits. We received 121 feedbacks in total, in another word, 605 segments has been rated. About 76% segments were considered as Accept, occupying 459 segments out of 605. The rest 24% segments were rejected by players, which has 146 segments.





According to survey data, we examine those “best” and “worst” generated images. An image are received most “Accept” rating is considered best image and one received most “Rejection” rating is the “Worst” image. Several characteristics of both image has been observed, which will definitely helps on deciding whether an image is suitable for the generation engine. These characteristics are

1. Many objects within the image
2. Sharp Edge of object against background
3. Less noise
4. Maybe a cartoon

Below is few examples illustrate the characteristics

Many objects	Few objects
	

Sharp Edge	Not Sharp Edge
	

Less Noise	Noisy
	

Chapter 13 Problems and Solutions

13.1 Segmented area found noise and distortion

Problem: 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.

Solution: The image analysis module screen out some of the terribly segmented regions. Also, the distortion problem of the unsorted segment is alleviated by applying the filter. However, we believed that there are still rooms for improvement.

13.2 No Global Information between the segments

Problem: 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

Solution: The surrounding of the segment takes into account by computing the mean the neighbor difference and the mode.

13.3 Not all images can be segmented

Problem: 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.

Solution: In this case of unsuccessful segmentation, our program will reject the image and display an alert message to encourage user to provide another photo. Also, the front page of the website shows some guidelines for user to follow in order to achieve the best effect from our engine.

13.4 Copyright Issue of User Upload Image

Problem: In the web-based PhotoHunt, users are free to upload any types of image. As the user process is uncontrolled, the engine sometime receives improper image such as copyrighted image, violence or sex materials. If these images are saved and shared to other user, it may arouse public concern.

Solution: We added a field “Approved” to the database, only approved image will be shared to other players. All the images uploaded will be checked for a certain period of time(less than a week), improper content will be deleted.

Chapter 14 Project Progress

Date	Progress
September 2006	<ul style="list-style-type: none"> • Carry out background research on PhotoHunt • Set goal and objectives for this semester • Learn and Got ourselves familiar with image processing • Study the basic data structure and operation provided by OpenCV with C++ • Write some testing program to analyze different segmentation approach
October 2006	<ul style="list-style-type: none"> • Study the usage of MFC to constructed the Semi-Automatic PhotoHunt Generation program • Design and test different elimination algorithms • Implement the elimination to the Semi-Automatic PhotoHunt Generation Engine.
November 2006	<ul style="list-style-type: none"> • Make improvement to the existing elimination algorithm • Learn Flash action script and PHP for the development of the Game Engine • Build the Game Engine for uploading images and invoking the generation function provided the Image Generation Engine. • Prepare report and presentation
December 2006	<ul style="list-style-type: none"> • Design and implement: <ol style="list-style-type: none"> 1. Game interface 2. Back End System for Single-player mode

January 2007	<ul style="list-style-type: none"> • Implement Multiplayer mode • Design new segmentation algorithm • Conduct testing to optimize the parameters of the existing single-player game
February 2007	<ul style="list-style-type: none"> • Implement Image analysis module • Add more features to generation engine including: <ul style="list-style-type: none"> • Image Warping • Object Appending • Implement Challenge Mode
March 2007	<ul style="list-style-type: none"> • Release preliminary version to the CSE network for internal test • Evaluate the performance of the engines • Make improvement according to feedbacks
April 2007	<ul style="list-style-type: none"> • Finalize all the engines • Release beta version for public test • Document the whole system

Chapter 15 Contribution of work

This chapter will state what I have been done and what I have learnt in the year's time. There are lots of difficulties that I have encountered; I have got many experiences and knowledge from solving the problems one by one. The following sections include the preparation, my works from semester one to semester two, then finally a conclusion.

15.1 Preparation

As we both have very limited knowledge about image processing, we have no idea of how the project started at the very beginning. We spent about one month to learn about the fundamental of image processing. During this one month, we also learnt about the MFC programming since we never do that before. To accelerated the learning curve, I was focus on the technical part of the programming while my partner concentrated on the algorithm of image processing after getting know some basic knowledge of digital image foundation.

15.2 Semester One

We have learnt that OpenCV has been a great tool for processing digital image which would definitely shorten our development time. However, the reality shows that it is not true. While it is not doubt that OpenCV is a very powerful library, the reference on web or book is very limited. I spent three weeks to get used to this library by reading some very tough documents and discussions. As my partner came with a segmentation method that could extract objects from the image, I have developed a semi-automatic program using both MFC and OpenCV. This

program made use of the segmentation algorithm with many manual controls and parameter tuning. It accelerated the development and testing time in the later time. What I have learnt here is the GUI programming skill. After the segmentation algorithm has been finished, we thought of 4 elimination algorithms. I have implemented those algorithms to the testing program and finally done a preliminary version of the generation engine. I have also made a simple front-end application to make use of the generation engine at this stage.

15.3 Semester Two

While my partner was continue to work on the generation engine with more flexible to applicable on more types of images, I have started to make a full-featured front-end application which supported by the generation engine. We decided to use Flash and PHP to do the job and worked together to finish the outlook of the game. I have no much experience on Flash such that I spent lots time on reading books and tutorial of it. I have learnt many advance technique about Flash in this semester, including the actionscript 2.0, uploading capability and the XMLSocket connection. By using such technical skills, I have successfully to create some unique features to the game including multiple player mode and challenge mode. The flow about the game is quite complicated; as I never design game before, I have got the knowledge about how to design a game to satisfy players. I have also done a revision about the database design since the game involving the storage in database. Although I am experienced in PHP programming, I got something new on this project, such as the socket programming.

15.4 Conclusion

For this project, I have paid lots of effort it but gain much more rewards. I have learnt many algorithms of image process and plenty of practical skills in design and programming. As we have done a very interest project which involves a fancy game, we felt happy by introducing it to friends and any one who want to play PhotoHunt. For a final word, I feel satisfactory in this project




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

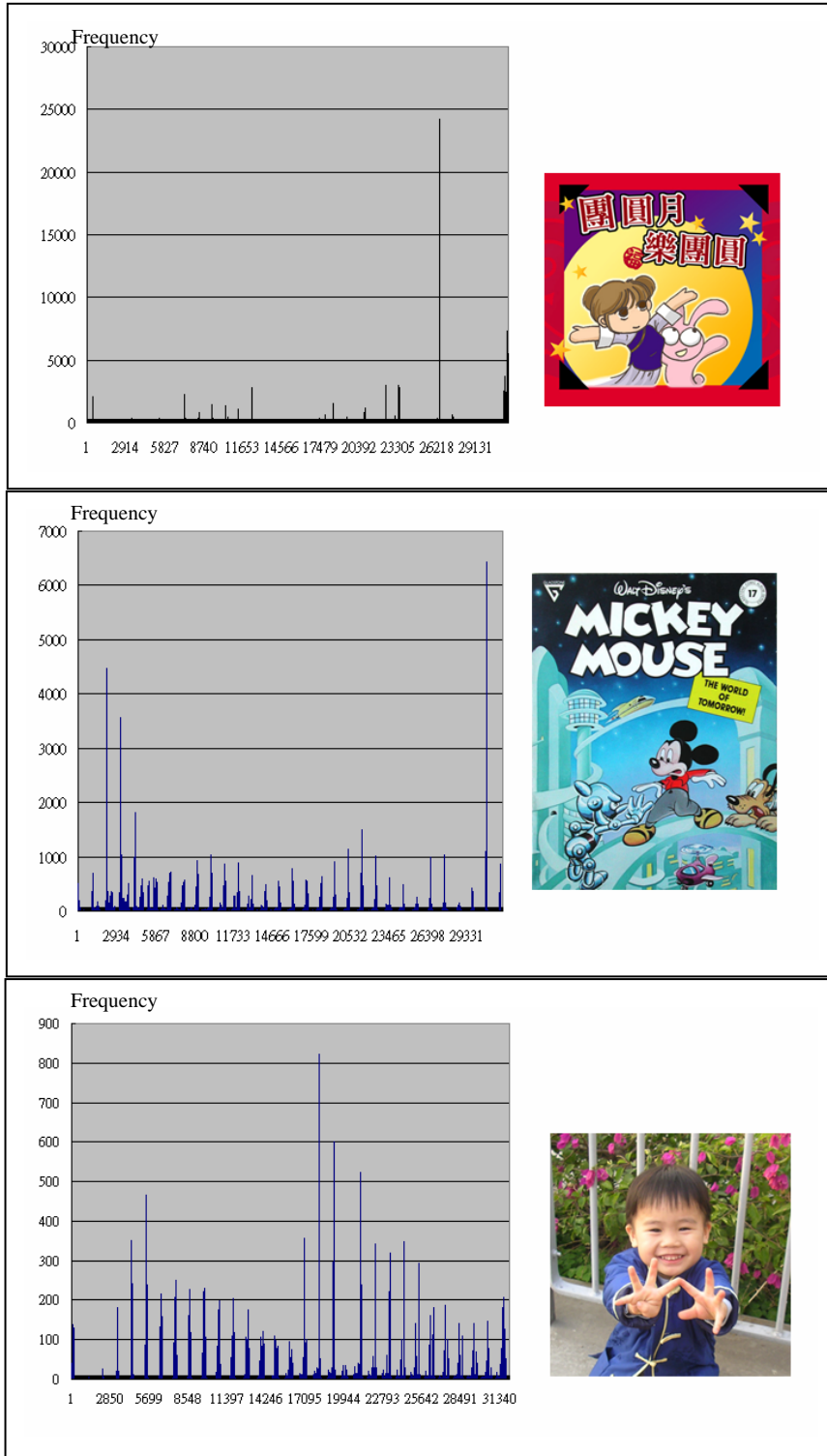


Segmented Image 2

Appendix 2 Analysis data

Below charts show the color histogram of the image.

However, the method is abandoned due to the fact that the 3-dimensional information is hashed to 2-dimension. The hashing function result in that the relationships between the channels have lost.



Appendix 3 Data Dictionary of Database

Entities	Fields	Descriptions
image	Definition	Stores image information
	Special Notes	-- --
score	Definition	Scores from players corresponding to a particular image
	Special Notes	-- --
comment	Definition	Players' comment about the game
	Special Notes	-- --
upload	Definition	Information of uploaded images
	Special Notes	This information is temporarily stored in this table, after the processing is done, the information will be transfer to the image table

Relationships	Fields	Descriptions
contains	Definition	For each image, there may be some comments come from users
	Parent Entity	image
	Child Entity	comment
	Special Notes	-- --
has	Definition	For each image, there may be some scores from players
	Parent Entity	image
	Child Entity	score
	Special Notes	-- --
upload-to	Definition	An uploaded image will have a record in entity upload, then this record will transfer to image after processing
	Parent Entity	upload
	Child Entity	image
	Special Notes	-- --


Attributes	Fields	Descriptions
imageID	Definition	Unique ID of the image
	Data Type	Character: {a-z A-Z 0-9}, length: 24
	Special Notes	-- --
answer	Definition	Answer of differences on image
	Data Type	Character: {a-z A-Z 0-9 ^ #}, length: 0-150
	Special Notes	-- --
md5code	Definition	Hash of the image file
	Data Type	Character: {a-z A-Z 0-9}, length: 32
	Special Notes	-- --
approval	Definition	Approval status of image
	Data Type	Character: {Y N}, length: 1
	Special Notes	-- --
id	Definition	Unique ID of comment
	Data Type	Integer, length: 11
	Special Notes	-- --
name	Definition	Name of player
	Data Type	Character: {a-z A-Z 0-9}, length: 0-100
	Special Notes	-- --
comment	Definition	Comments from players
	Data Type	Text
	Special Notes	-- --
postTime	Definition	Submitted time of comments
	Data Type	Timestamp: yyyy-mm-dd hh:mm:ss
	Special Notes	-- --
Time	Definition	Timestamp of uploaded image
	Data Type	Timestamp: yyyy-mm-dd hh:mm:ss
	Special Notes	-- --
mode	Definition	Playing mode of game
	Data Type	Character: {s m}, length: 1
	Special Notes	-- --
score	Definition	Score of playing game
	Data Type	Integer, length: 5
	Special Notes	-- --

Appendix 4 Evaluation Data

Lower limit	Upper limit	Accepted	%	Rejected	%	Total Image	Processing time
5000	100	1046	79.36%	272	20.64%	1318	00:14:55
5000	120	1039	78.83%	279	21.17%	1318	00:14:29
5000	140	1043	79.14%	275	20.86%	1318	00:15:07
5000	160	1041	78.98%	277	21.02%	1318	00:15:36
5000	180	1030	78.15%	288	21.85%	1318	00:16:44
5000	220	1026	77.85%	292	22.15%	1318	00:17:08
5000	240	1020	77.39%	298	22.61%	1318	00:14:57
5000	260	1022	77.54%	296	22.46%	1318	00:14:36
4500	100	1013	76.86%	305	23.14%	1318	00:14:15
4500	120	1004	76.18%	314	23.82%	1318	00:14:49
4500	140	1013	76.86%	305	23.14%	1318	00:15:20
4500	160	1004	76.18%	314	23.82%	1318	00:15:47
4500	180	995	75.49%	323	24.51%	1318	00:16:09
4500	200	995	75.49%	323	24.51%	1318	00:16:38
4500	220	997	75.64%	321	24.36%	1318	00:17:36
4500	240	992	75.27%	326	24.73%	1318	00:17:21
4500	260	987	74.89%	331	25.11%	1318	00:17:51

Appendix 5 Internal Test

The poster was posted to newsgroup that invited students to try on the game:



Help Needed- FYP Testing
Experiencing the newly Developed Online PhotoHunt Generation

We are CS students currently working on the Final Year Project "**Automatic PhotoHunt Generation**". We would like to invite all CSE staffs and students to try and test out our newly developed PhotoHunt Game. Below is the information of the game. Hope you all enjoy it!

How to start Automatic PhotoHunt Generation?
Please visit
➔ <http://photohunt.servegame.org>
***The game can only be accessed through CSE network.

What is Automatic PhotoHunt Generation (APG)?
Base on the classic PhotoHunt game, APG will instantly generate 4-5 differences for an user uploaded pictures. APG also support two newly invented modes, namely multi-player mode and challenge mode.
Multi-player Mode: Challenge your friends in this competition based Multi-player mode!
Challenge Mode: Don't miss the chance to test your well-trained skill for Advanced player


We would be grateful if you could provide your valuable opinions.

There are some feedbacks from CSE students:

Help needed on FYP Testing - Automatic PhotoHunt Generation	hlshum4	2007/3/26 下午 10:32	210KB
Help needed on FYP Testing - Automatic PhotoHunt Generation	hlshum4	2007/3/26 下午 10:33	210KB
Re: Help needed on FYP Testing - Automatic PhotoHunt Generation	柏德烈治	2007/3/27 上午 12:07	2KB
Re: Help needed on FYP Testing - Automatic PhotoHunt Generation	大廉成	2007/3/27 上午 09:38	49KB
Re: Help needed on FYP Testing - Automatic PhotoHunt Generation	柏德烈治	2007/3/27 下午 03:36	1KB
Re: Help needed on FYP Testing - Automatic PhotoHunt Generation	ming	2007/3/27 上午 11:32	546KB
Re: Help needed on FYP Testing - Automatic PhotoHunt Generation	柏德烈治	2007/3/27 下午 03:31	2KB

Front page of the released version:

Automatic PhotoHunt Generation




What is Automatic PhotoHunt Generation?
Base on the classic PhotoHunt game, APG will instantly generate 5 differences for an user uploaded pictures. APG also support two newly invented modes, namely multi-player mode and challenge mode.

Who we are?
We are two year 3 students, Raymond SHUM and Vincy To, who study Computer Science at The Chinese University of Hong Kong. This project is our FYP project which supervised by Prof. Michael R. Lyu.


How to obtain a better result?
It depends on the quality of images. Take a look of the examples :

GOOD!


Many Objects




Few Objects




Sharp Edge




Not Sharp Edge



Less Noise

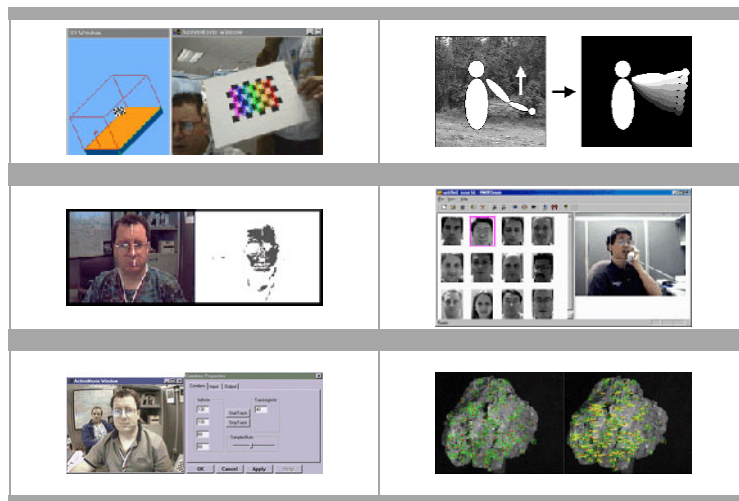


Noisy



Appendix 6 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

- [4] Wikipedia, 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>
- [15] Image Processing Fundamental,
<http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Statisti.html>
- [16] Lidija Mandic; Sonja Grgic; Mislav Grgic;” **Comparison of Color Difference Equations**”, Multimedia Signal Processing and Communications, 48th International Symposium ELMAR-2006 focused on June 2006 Page(s):107 – 110
- [17] Nick Efford, “Digital Image Processing a practical introduction using JAVA”, Addison-Wesley, 2000
- [18] Jiaya Jia and Chi-Keung Tang, “ Image Repairing: Robust Image Synthesis by Adaptive ND Tensor Voting.”. IEEE Conference on Computer Vision and Pattern Recognition (*CVPR*), June 2003, pages I: 643-650
- [19] .Pedro F. Felzenszwalb and Daniel P. Huttenlocher. “Efficient Graph-Based Image Segmentation”, International Journal of Computer Vision, Volume 59, Number 2, September 2004.

[20] PHP 5 Sockets with Flash 8

http://www.kirupa.com/developer/flash8/php5sockets_flash8.htm

[21] Lindsay W. MacDonald and M. Ronnier Luo, "COLOUR IMAGE SCIENCE",
John Wiley & Sons, LTD.

[22] I. Pitas, "Digital Image Processing Algorithms and Applications",
Wiley-Interscience.

[23] Abhay Sharma, "Understanding Color Management", Delmar Learning, Thomson
Learning, Inc.