PATTERN RECOGNITION

By Lijia Wu

Prepared Under the Direction of Dr. Henry Chang

School of Engineering
Northwestern Polytechnic University
47671 Westinghouse Dr., Fremont, CA 94539

13-Feb-13

ACKNOWLEDGEMENTS

I am heartily thankful to the contributions of the following groups and individuals to the development of my thesis.

I would like to express my deepest appreciation to Dr. Henry Chang, who is really selfless to give us the opportunities to practice and dedicating to give us guidance. His encouragement and support from initial to the final level enabled me to develop an understanding of the subject.

In addition, I would like to thank Mr. Paul Cao, who patiently helps me solve difficulties I confronted during the project. I am extremely grateful for your assistance and suggestions throughout the whole time.

Besides, a thank you to all project group members for the mutual encouragements. Hope we will fulfill ourselves all the way learning and practicing.

Last but not least, a special thanks to my family members for their support, love and encouragement to achieve goals in my life.

ABSTRACT

Pattern recognition is not a new field of research. Actually, theories and techniques about it have developed for a long time. While with the fast advancement of computer architecture, machine learning, and computer vision, computational complexity is possible to be dealt with and more and more new ways of thinking are brought into the research of pattern recognition. The research introduces a simplified method of pattern recognition, and some outstanding applications are included.

In existing researches there are many techniques which are available for extracting information from images. However, in daily life we will come across different types of images, which need different handling method to be recognized. In this study we will come across a set of easy-understood processes for clearly isolated-object images, especially for circle or square objects.

Like real-world image processing, we will recommend histogram, thresholding, and connectivity analysis for whole pattern recognition process.

The method in this paper has been proved to be useful for large samples and has a little deviation for small image samples. Further studies with large size samples are required to demonstrate significance.

Although no technical significance was in this study, it is still encouraging to figure out a fundamental pattern recognition technique for beginners.

CONTENTS

ILLUSTRATIONS

1. OVERVIEW

1.1. Pattern Recognition Concept

When the techniques of image processing were developed in the 1960s, the cost of processing was fairly high with the computing equipment of that era. That changed in the 1970s, when digital image processing proliferated as cheaper computers and dedicated hardware became available. With the fast computers and signal processors available in the 2000s, digital image processing has become the most common form of image processing and generally, is used because it is not only the most versatile method, but also the cheapest.

Image pattern recognition, also known as image processing, means methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information. It recognizes elements of an image through processes.

1.2. The Objectives

The goal of image pattern recognition is to distinguish or recognize the objects in an image, to make objects visualizable.

1.3. Recognition Method

The classical problem in image processing is that of determining whether or not the image data contains some specific object, feature, or activity.

Here, we assume that an image has been converted into a 2-D array consisting of different integers, which represent different levels of pixel grey. We recognize an image mainly through following four steps:

- Convert image pixel array into binary array, set background pixels to zero as boundary. We need to use histogram-based methods to find a proper threshold to separate the matrix into two classes. For example, foreground to 1, background to 0, or object groups to non-zero, background to zero. We'll discuss how to generate a histogram and introduce histogram-based ways for thresholding in the following chapters.

- After got the corresponding binary array, start to analyze pixel connectivity and try to isolate objects. We'll discuss connectivity analysis in chapter 4.

- At last, based on some formulas, try to recognize object shapes(circular or square).

2. HISTOGRAM

2.1. Introduction

Histogram is first introduced by Karl Pearson in Statistics as a graphical representation showing a visual impression of distribution of data, or the frequency of data distribution.

Usually, a histogram consists of adjacent rectangle bars. The heights of the bars represent the frequency density of intervals while the intervals represent the value of data. Figure 1 shows an example of simple histogram.

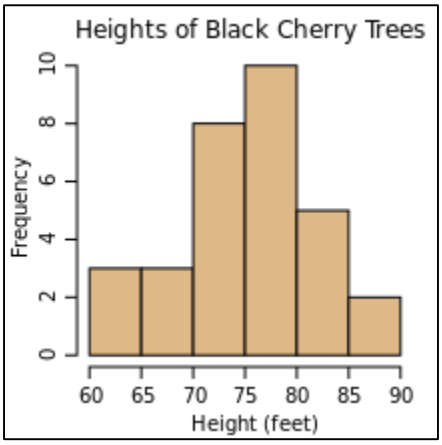

Figure 1 Heights of Black Cherry Trees

2.2. Algorithm

Using histogram is a fast and simple way to generate the frequencies of values of a 2-D array in programming. It uses array elements as indexes to another array's elements.

- Assume the range of 2D array value is 0(min) to 255(max), define a histogram 1D array with 256(max – min + 1) values.

```
int image[N][N]={
    {1,3,5,7,9,3,4,4,5,6},
    {1,3,5,7,9,3,4,4,5,6},
    {1,3,5,7,9,3,4,4,5,6},
    {1,3,5,20,25,4,33,5,6,4},
    {1,3,5,20,35,4,32,5,6,4},
    {1,3,5,7,3,9,23,5,6,4},
    {1,3,5,21,25,27,23,5,6,4},
    {1,3,5,7,9,3,4,4,5,6},
    {1,3,5,7,9,3,4,4,5,6},
    {1,3,5,7,9,3,4,4,5,6}
   };
/* Histogram of the image data */
int hist[max]={0};
```

- Increment histogram[i] when finding a i from the 2D array(i is the 2D array value)

```
for(i = 0; i < N; i++)
  {
        for(j = 0; j < N; j++)
        {
              hist[image[i][j]]++;
        }
  }
```

- Example output

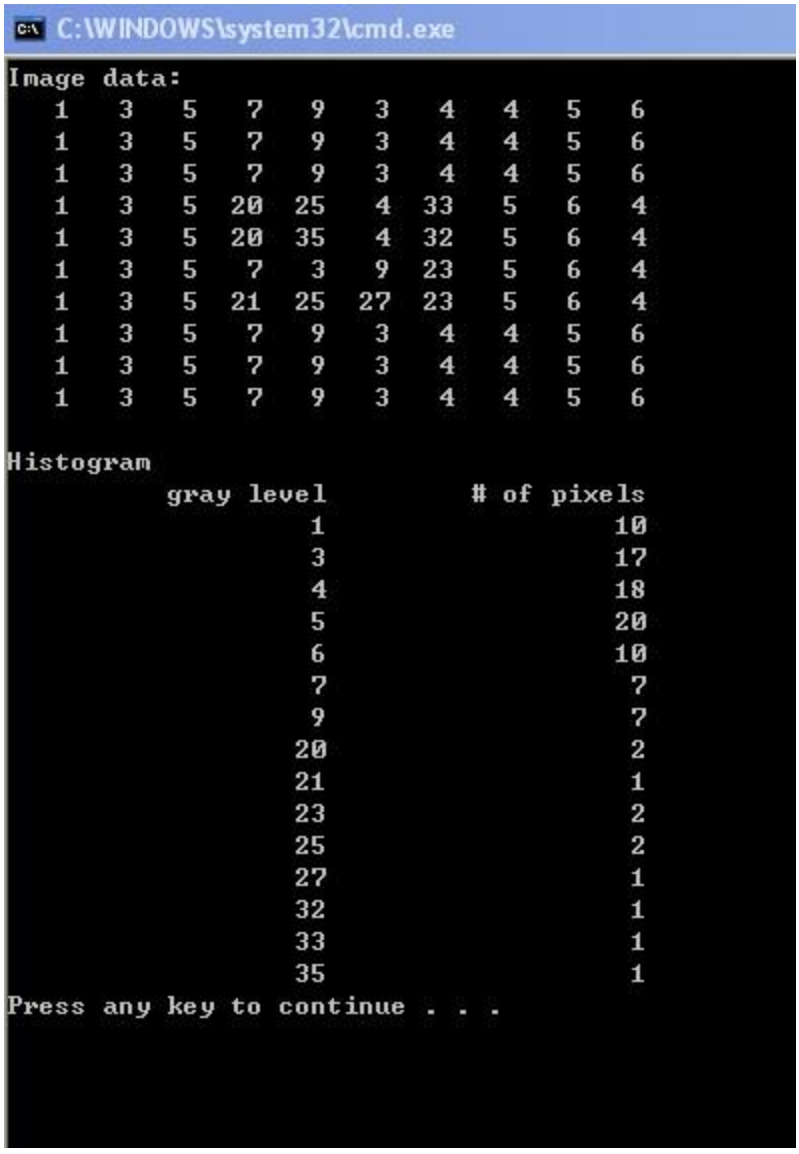

Figure 2 Histogram output

3. THRESHOLD

3.1. Introduction

After generating the histogram of a 2-D array, in order to separate objects from an image, we need to determine the border of pixels, that is to find a threshold as the separation symbol for foreground and background or more other group pixels.

There are many histogram-based ways to determine the threshold, such as thresholding(image process), balanced histogram thresholding and Otsu's method.

Here, we used Otsu's method.

3.2. Algorithm

The algorithm assumes that the image to be thresholded contains two classes of pixels or bi-modal histogram (e.g. foreground and background) then calculates the optimum threshold separating those two classes so that their combined spread (intra-class variance) is minimal(Nobuyuki Otsu, 1979).

The formula is: **$g(t) = \omega0(t)\ \omega1(t)\ (\mu0(t) - \mu1(t))\ ^\wedge\ 2$**

t: the threshold separating 1D array value to background and foreground classes
g(t): variance of these classes
w0(t): the occurrence rate of value less than t in whole array
w(1t): the occurrence rate of value greater than t in whole array
u0(t): the average of value less than t(sum(i * number of (i)) / number of (value less than t))
u1(t): the average of value greater than t(sum(i * number of (i)) / number of (value less than t))

The principle is finding the maximum g(t), then this t is the best threshold.

So here is my code:
rcfore is w0(t), rcback is w1(t), pixelCforeRate is u0(t), pixelCbackRate is u1(t)

```
for(thr = 1; thr < size; thr++)
        {
                cfore = cback = 0;
                pixelTotalCfore = pixelTotalCback = 0;
                for(i = 1; i <= thr; i++)
                {
                        //calcute foreground color's number of pixel
                        cfore += hist[i];
                        //calcute foreground color's area of pixel
                        pixelTotalCfore += i * hist[i];
                }
                for(i = thr; i < size; i++)
                {
                        //calcute background color's number of pixels
                        cback += hist[i];
```

```
                    //calcute background color's area of pixels
                    pixelTotalCback += i * hist[i];
                }

                //the number rate of foreground and background colors
                rcfore = (float)cfore / (imageRow * imageCol);
                rcback = 1 - rcfore;

                //the area rate of foreground and background colors
                if(cfore){
                    pixelCforeRate = (float) pixelTotalCfore / cfore;
                }
                else{
                    pixelCforeRate = 0;
                }

                if(cback){
                    pixelCbackRate = (float) pixelTotalCback / cback;
                }
                else{
                    pixelCbackRate = 0;
                }

                //calculate the variance
                result = rcfore * rcback * (pixelCforeRate - pixelCbackRate) * (pixelCforeRate -
pixelCbackRate);

                if(result > maxg){
                    maxg = result;
                    bestThr = thr;
                }
            }
```

Example output:

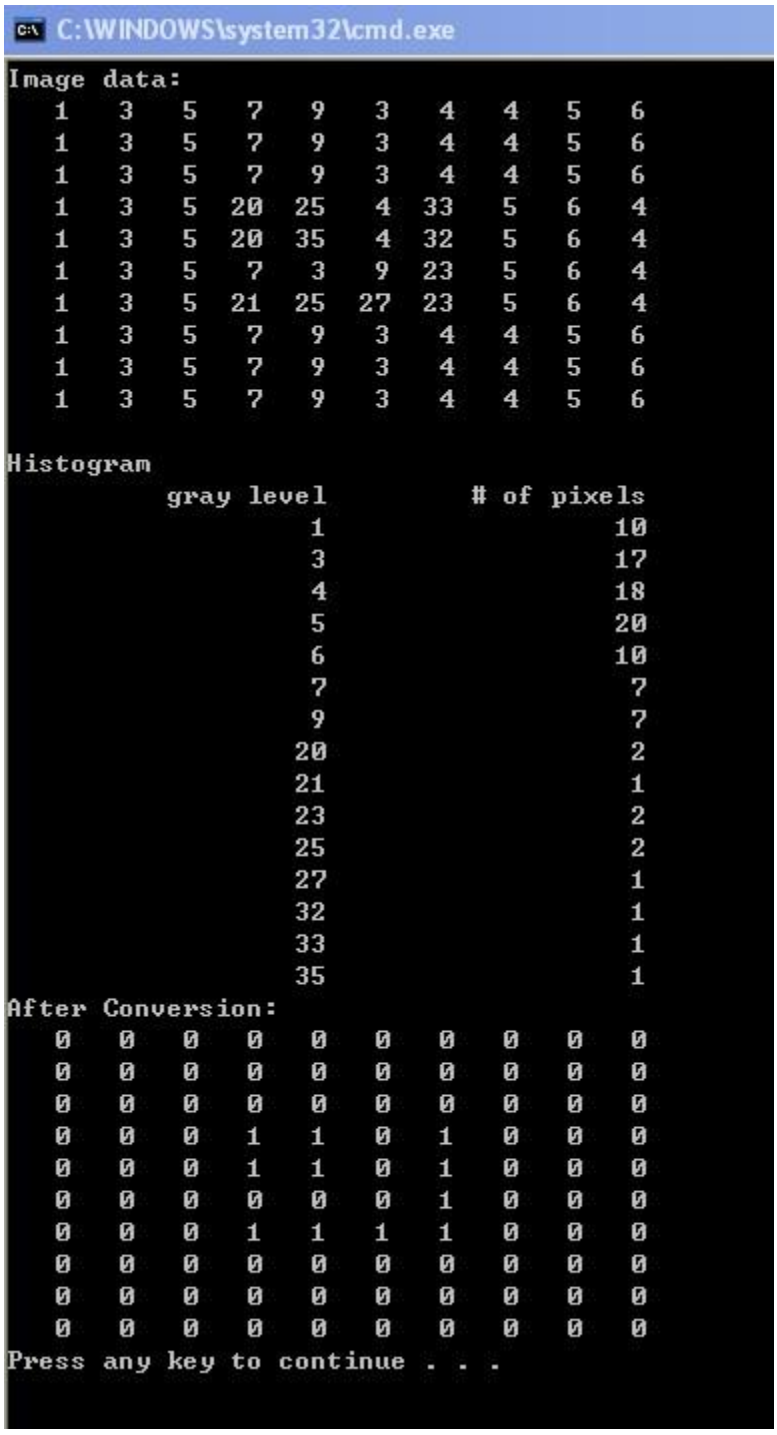

Figure 3 Binary Image Array

4. CONNECTIVITY ANALYSIS

4.1. Introduction

Connectivity analysis generates a matrix of connectivity relationships between pixel grey levels, which based on connected-component labeling.

A graph, containing vertices and connecting edges, is constructed from relevant input data. The vertices contain information required by the comparison heuristic, while the edges indicate connected 'neighbors'. Connected-component labeling traverses the graph, labeling the vertices based on the connectivity and relative values of their neighbors. Connectivity is determined by the medium; image graphs, for example, can be 4-connected or 8-connected(R. Fisher, et al, 2003).

4-connected: checking 4 nearby elements

8-connected: checking 8 nearby elements

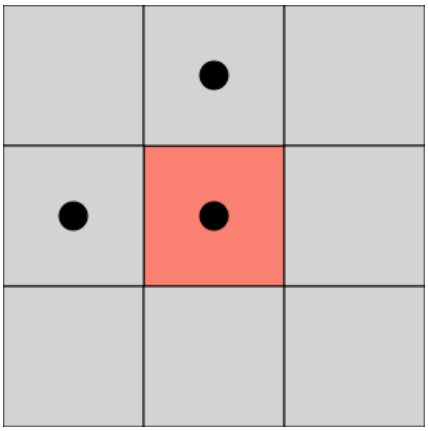

Figure 4 4-connected

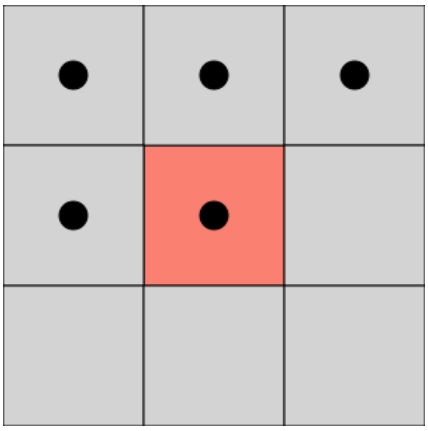

Figure 5 8-connected

Here, we based on 8-connected, checking the values of pixels that are North-East, North, North-West and West of the current pixel.

4.2. Algorithm

There are mainly two steps to do the analysis:

- From left to right, top to bottom, roughly analyze connectivity and assign labels to each pixel(check nearby 4 pixels in order: west, northwest, north and northeast). Assigned the first pixel value to the first label(that is 0 here).
  For the first row values, if current pixel value is same with the west pixel, assign west pixel's label to current pixel; or create a new label.
  For the first column, if current pixel value is same with the north pixel, assign north pixel's label to current pixel; or create a new label.
  For the last column, if current pixel value is same with the west pixel, assign west pixel's label to current pixel; or check the northwest, if same, assign the northwest's; or check the north pixel, if same, assign the north's; or create a new label;
  For other pixels, check west, northwest, north, northeast in order.

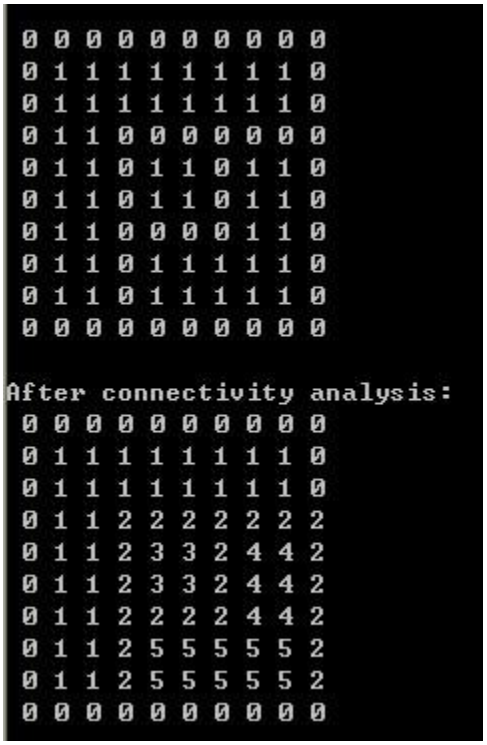

Figure 6 Connectivity step I

- From right to left, bottom to top, check if there are connected pixels with different labels(this time check nearby 8 pixels). If there are, choose the smallest label to relabel the pixels.

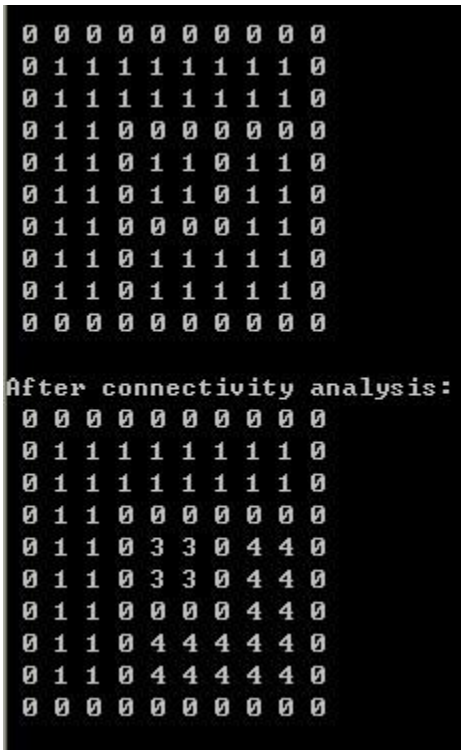

Figure 7 Connectivity step II

- check if the labels continuous, if not, make them continuous

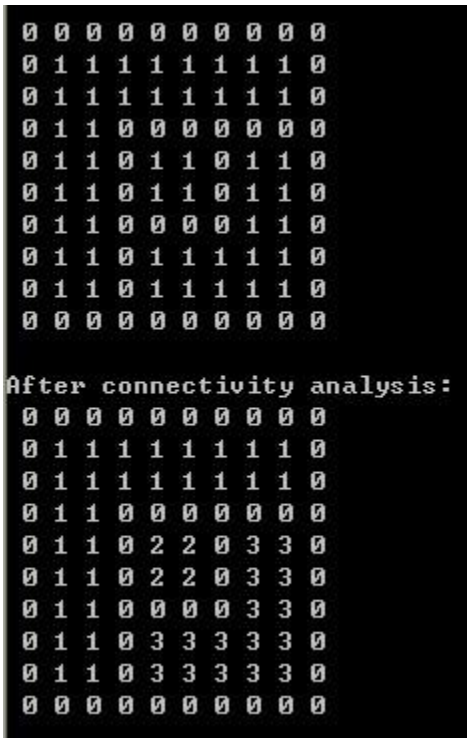

Figure 8 Connectivity step III

5. OBJECT SHAPE ANALYSIS

5.1. Introduction

Here, we based on given formula to judge object shapes by compare the formula result.

We assume that only two types of objects exist in the image, one is circular object, another is square object.

5.2. Algorithm

- Based on the algorithm of histogram, calculate the area and perimeter of each label(area means how many pixels, perimeter means the number of boundary pixels).

Check the west, north, east and south of current pixel. If current pixel is different from either one of the four directions, then mark current pixel label as a border(Tang Zhen-Jun, et al, 2006). Note: the zeroes inside red rectangles are boundary pixels.

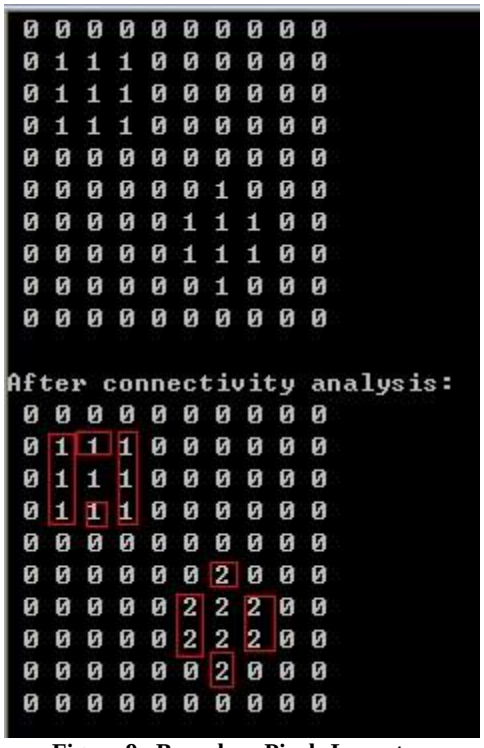

Figure 9 Boundary Pixels Layout

- According to below formula, calculate the radius of each object, here each label object.

  r = 4 * π * area[i] /(perimeter[i] * perimeter[i]); //i is the pixel labels

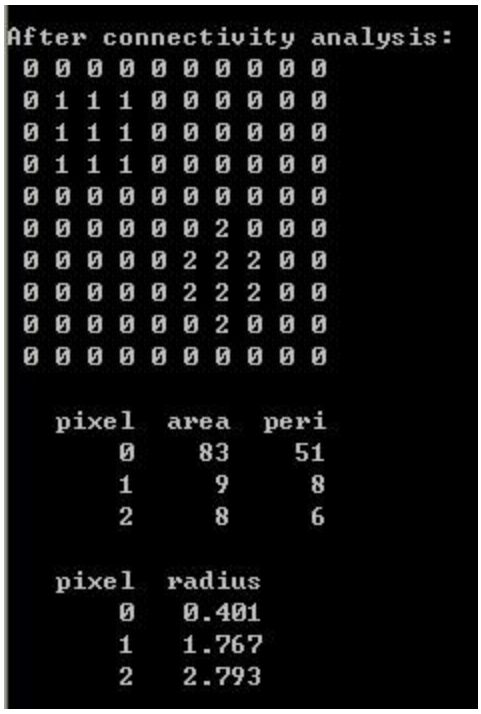

Figure 10 Radius

Theoretically, if r is close to $\pi/4$, then the object is a square; if close to 1, the object is a circle. However, because the array is too small, so there is a big variance for this result.

6. PROGRAM CODE

- **Histogram and Threshold practice**

```
#include<stdio.h>

#define max 256
#define N 10

int findThreshold(int hist[], int size, int imageRow, int imageCol);
void convertBin(int image[][N], int row, int thr);

void main()
{
  /* Image data */
  int image[N][N]={
      {1,3,5,7,9,3,4,4,5,6},
      {1,3,5,7,9,3,4,4,5,6},
      {1,3,5,7,9,3,4,4,5,6},
      {1,3,5,20,25,4,33,5,6,4},
      {1,3,5,20,35,4,32,5,6,4},
      {1,3,5,7,3,9,23,5,6,4},
      {1,3,5,21,25,27,23,5,6,4},
      {1,3,5,7,9,3,4,4,5,6},
      {1,3,5,7,9,3,4,4,5,6},
      {1,3,5,7,9,3,4,4,5,6}
     };
  /* Histogram of the image data */
  int hist[max]={0};
  int i, j;
  int bestThr;

  /* The code size of calculating a histogram from
   * an image data is less than 5 lines
   */
   for(i = 0; i < N; i++)
      {
                for(j = 0; j < N; j++)
                {
                        hist[image[i][j]]++;
                }
      }

      //print data and histogram
      printf("Image data:\n");
      for(i = 0; i < N; i++)
      {
                for(j = 0; j < N; j++)
                {
                        printf("%4d", image[i][j]);
                }
                printf("\n");
      }
      printf("\nHistogram\n%20s%20s\n", "gray level", "# of pixels");
      for(i = 0; i < max; i++)
      {
                if(hist[i]){
                        printf("%20d%20d\n", i, hist[i]);
                }
      }
```

```
	bestThr = findThreshold(hist, max, N, N);

	//convert 2D array to 0 and 1 based on the threshold
	convertBin(image, N, bestThr);

}

int findThreshold(int hist[], int size, int imageRow, int imageCol)
{
	int thr, bestThr, i;
	int cfore, cback;
	int pixelTotalCfore, pixelTotalCback;
	float rcfore, rcback;
	float pixelCforeRate, pixelCbackRate;
	float result;
	float maxg = 0;

	for(thr = 1; thr < size; thr++)
	{
		cfore = cback = 0;
		pixelTotalCfore = pixelTotalCback = 0;
		for(i = 1; i <= thr; i++)
		{
			//calcute foreground color's number of pixel
			cfore += hist[i];
			//calcute foreground color's area of pixel
			pixelTotalCfore += i * hist[i];
		}
		for(i = thr; i < size; i++)
		{
			//calcute background color's number of pixels
			cback += hist[i];
			//calcute background color's area of pixels
			pixelTotalCback += i * hist[i];
		}

		//the number rate of foreground and background colors
		rcfore = (float)cfore / (imageRow * imageCol);
		rcback = 1 - rcfore;

		//the area rate of foreground and background colors
		if(cfore){
			pixelCforeRate = (float) pixelTotalCfore / cfore;
		}
		else{
			pixelCforeRate = 0;
		}

		if(cback){
			pixelCbackRate = (float) pixelTotalCback / cback;
		}
		else{
			pixelCbackRate = 0;
		}

		//calculate the variance
		result = rcfore * rcback * (pixelCforeRate - pixelCbackRate) * (pixelCforeRate - pixelCbackRate);

		if(result > maxg){
```

```
                        maxg = result;
                        bestThr = thr;
                }
        }

        return bestThr;
}

void convertBin(int image[][N], int row, int thr)
{
        int i, j;
        for(i = 0; i < row; i++)
        {
                for(j = 0; j < N; j++)
                {
                        if(image[i][j] < thr){
                                image[i][j] = 0;
                        }
                        else{
                                image[i][j] = 1;
                        }
                }
        }

        printf("After Conversion:\n");
        for(i = 0; i < N; i++)
        {
                for(j = 0; j < N; j++)
                {
                        printf("%4d", image[i][j]);
                }
                printf("\n");
        }
}
```

- **Connectivity Analysis Practice**

```
void connectAnalysis(int a[][N], int b[][N], int size)
{
        int i, j, p, k=0;   //k is the label, starting from 0
        int c[N*N];
        int found;

        //first pass of connectivity analysis to assign labels
        for(i = 0; i < N; i++)
        {
                for(j = 0; j < N; j++)
                {
                        //tag a label to the first value of image array
                        if(i == 0 && j == 0){
                                b[i][j] = k++;
                        }

                        //check the west of the first row pixels, if equal, assign the same label
                        //to the current pixel; or assign a new label
                        else if(i == 0 && j > 0){
                                if(a[i][j] == a[i][j-1]){
                                        b[i][j] = b[i][j-1];
                                }
                                else{
```

```
                                        b[i][j] = k++;
                                }
                        }

                        //check the north and northeast of the first column pixels
                        //if equal to either, assign the same label to current pixel
                        //or, create new label for current pixel
                        else if(j == 0){
                                if(a[i][j] == a[i-1][j]){
                                        b[i][j] = b[i-1][j];
                                }
                                else if(a[i][j] == a[i-1][j+1]){
                                        b[i][j] = b[i-1][j+1];
                                }
                                else{
                                        b[i][j] = k++;
                                }
                        }

                        //check the west, northwest and north of the last column pixels
                        else if(j == N-1){
                                if(a[i][j] == a[i][j-1]){
                                        b[i][j] = b[i][j-1];
                                }
                                else if(a[i][j] == a[i-1][j-1]){
                                        b[i][j] = b[i-1][j-1];
                                }
                                else if(a[i][j] == a[i-1][j]){
                                        b[i][j] = b[i-1][j];
                                }
                                else{
                                        b[i][j] = k++;
                                }
                        }

                        //other pixels have full neighbors, that is west, northwest, north, northeast
                        else{
                                if(a[i][j] == a[i][j-1]){
                                        b[i][j] = b[i][j-1];
                                }
                                else if(a[i][j] == a[i-1][j-1]){
                                        b[i][j] = b[i-1][j-1];
                                }
                                else if(a[i][j] == a[i-1][j]){
                                        b[i][j] = b[i-1][j];
                                }
                                else if(a[i][j] == a[i-1][j+1]){
                                        b[i][j] = b[i-1][j+1];
                                }
                                else{
                                        b[i][j] = k++;
                                }
                        }
                }
}

//second pass of connectivity analysis to modify labels
for(i = N-1; i >= 0; i--)
{
        for(j = N-1; j >= 0; j--)
```

```
{
		if(i == 0 && j == 0){
			b[i][j] = b[i][j];
		}
		else if(i == 0 && j == N-1){
			if(a[i][j] == a[i][j-1] && b[i][j] != b[i][j-1]){
				b[i][j] = findMin(b[i][j], b[i][j-1]);
				b[i][j-1] = b[i][j];
			}
			if(a[i][j] == a[i+1][j-1] && b[i][j] != b[i+1][j-1]){
				b[i][j] = findMin(b[i][j], b[i+1][j-1]);
				b[i+1][j-1] = b[i][j];
			}
			if(a[i][j] == a[i+1][j] && b[i][j] != b[i+1][j]){
				b[i][j] = findMin(b[i][j], b[i+1][j]);
				b[i+1][j] = b[i][j];
			}
		}
		else if(i == N-1 && j == 0){
			if(a[i][j] == a[i-1][j] && b[i][j] != b[i-1][j]){
				b[i][j] = findMin(b[i][j], b[i-1][j]);
				b[i-1][j] = b[i][j];
			}
			if(a[i][j] == a[i-1][j+1] && b[i][j] != b[i-1][j+1]){
				b[i][j] = findMin(b[i][j], b[i-1][j+1]);
				b[i-1][j+1] = b[i][j];
			}
			if(a[i][j] == a[i][j+1] && b[i][j] != b[i][j+1]){
				b[i][j] = findMin(b[i][j], b[i][j+1]);
				b[i][j+1] = b[i][j];
			}
		}
		else if(i == N-1 && j == N-1){
			if(a[i][j] == a[i][j-1] && b[i][j] != b[i][j-1]){
				b[i][j] = findMin(b[i][j], b[i][j-1]);
				b[i][j-1] = b[i][j];
			}
			if(a[i][j] == a[i-1][j-1] && b[i][j] != b[i-1][j-1]){
				b[i][j] = findMin(b[i][j], b[i-1][j-1]);
				b[i-1][j-1] = b[i][j];
			}
			if(a[i][j] == a[i-1][j] && b[i][j] != b[i-1][j]){
				b[i][j] = findMin(b[i][j], b[i-1][j]);
				b[i-1][j] = b[i][j];
			}
		}
		else if(i == 0){
			if(a[i][j] == a[i][j-1] && b[i][j] != b[i][j-1]){
				b[i][j] = findMin(b[i][j], b[i][j-1]);
				b[i][j-1] = b[i][j];
			}
			if(a[i][j] == a[i][j+1] && b[i][j] != b[i][j+1]){
				b[i][j] = findMin(b[i][j], b[i][j+1]);
				b[i][j+1] = b[i][j];
			}
			if(a[i][j] == a[i+1][j-1] && b[i][j] != b[i+1][j-1]){
				b[i][j] = findMin(b[i][j], b[i+1][j-1]);
				b[i+1][j-1] = b[i][j];
			}
			if(a[i][j] == a[i+1][j] && b[i][j] != b[i+1][j]){
```

```
                        b[i][j] = findMin(b[i][j], b[i+1][j]);
                        b[i+1][j] = b[i][j];
                }
                if(a[i][j] == a[i+1][j+1] && b[i][j] != b[i+1][j+1]){
                        b[i][j] = findMin(b[i][j], b[i+1][j+1]);
                        b[i+1][j+1] = b[i][j];
                }
        }
        else if(i == N-1){
                if(a[i][j] == a[i][j-1] && b[i][j] != b[i][j-1]){
                        b[i][j] = findMin(b[i][j], b[i][j-1]);
                        b[i][j-1] = b[i][j];
                }
                if(a[i][j] == a[i][j+1] && b[i][j] != b[i][j+1]){
                        b[i][j] = findMin(b[i][j], b[i][j+1]);
                        b[i][j+1] = b[i][j];
                }
                if(a[i][j] == a[i-1][j-1] && b[i][j] != b[i-1][j-1]){
                        b[i][j] = findMin(b[i][j], b[i-1][j-1]);
                        b[i-1][j-1] = b[i][j];
                }
                if(a[i][j] == a[i-1][j] && b[i][j] != b[i-1][j]){
                        b[i][j] = findMin(b[i][j], b[i-1][j]);
                        b[i-1][j] = b[i][j];
                }
                if(a[i][j] == a[i-1][j+1] && b[i][j] != b[i-1][j+1]){
                        b[i][j] = findMin(b[i][j], b[i-1][j+1]);
                        b[i-1][j+1] = b[i][j];
                }
        }
        else if(j == 0){
                if(a[i][j] == a[i-1][j] && b[i][j] != b[i-1][j]){
                        b[i][j] = findMin(b[i][j], b[i-1][j]);
                        b[i-1][j] = b[i][j];
                }
                if(a[i][j] == a[i+1][j] && b[i][j] != b[i+1][j]){
                        b[i][j] = findMin(b[i][j], b[i+1][j]);
                        b[i+1][j] = b[i][j];
                }
                if(a[i][j] == a[i-1][j+1] && b[i][j] != b[i-1][j+1]){
                        b[i][j] = findMin(b[i][j], b[i-1][j+1]);
                        b[i-1][j+1] = b[i][j];
                }
                if(a[i][j] == a[i][j+1] && b[i][j] != b[i][j+1]){
                        b[i][j] = findMin(b[i][j], b[i][j+1]);
                        b[i][j+1] = b[i][j];
                }
                if(a[i][j] == a[i+1][j+1] && b[i][j] != b[i+1][j+1]){
                        b[i][j] = findMin(b[i][j], b[i+1][j+1]);
                        b[i+1][j+1] = b[i][j];
                }
        }
        else if(j == N-1){
                if(a[i][j] == a[i-1][j] && b[i][j] != b[i-1][j]){
                        b[i][j] = findMin(b[i][j], b[i-1][j]);
                        b[i-1][j] = b[i][j];
                }
                if(a[i][j] == a[i+1][j] && b[i][j] != b[i+1][j]){
                        b[i][j] = findMin(b[i][j], b[i+1][j]);
                        b[i+1][j] = b[i][j];
```

```
                        }
                        if(a[i][j] == a[i-1][j-1] && b[i][j] != b[i-1][j-1]){
                                b[i][j] = findMin(b[i][j], b[i-1][j-1]);
                                b[i-1][j-1] = b[i][j];
                        }
                        if(a[i][j] == a[i][j-1] && b[i][j] != b[i][j-1]){
                                b[i][j] = findMin(b[i][j], b[i][j-1]);
                                b[i][j-1] = b[i][j];
                        }
                        if(a[i][j] == a[i+1][j-1] && b[i][j] != b[i+1][j-1]){
                                b[i][j] = findMin(b[i][j], b[i+1][j-1]);
                                b[i+1][j-1] = b[i][j];
                        }
                }
                else{
                        if(a[i][j] == a[i-1][j] && b[i][j] != b[i-1][j]){
                                b[i][j] = findMin(b[i][j], b[i-1][j]);
                                b[i-1][j] = b[i][j];
                        }
                        if(a[i][j] == a[i+1][j] && b[i][j] != b[i+1][j]){
                                b[i][j] = findMin(b[i][j], b[i+1][j]);
                                b[i+1][j] = b[i][j];
                        }
                        if(a[i][j] == a[i-1][j-1] && b[i][j] != b[i-1][j-1]){
                                b[i][j] = findMin(b[i][j], b[i-1][j-1]);
                                b[i-1][j-1] = b[i][j];
                        }
                        if(a[i][j] == a[i][j-1] && b[i][j] != b[i][j-1]){
                                b[i][j] = findMin(b[i][j], b[i][j-1]);
                                b[i][j-1] = b[i][j];
                        }
                        if(a[i][j] == a[i+1][j-1] && b[i][j] != b[i+1][j-1]){
                                b[i][j] = findMin(b[i][j], b[i+1][j-1]);
                                b[i+1][j-1] = b[i][j];
                        }
                        if(a[i][j] == a[i-1][j+1] && b[i][j] != b[i-1][j+1]){
                                b[i][j] = findMin(b[i][j], b[i-1][j+1]);
                                b[i-1][j+1] = b[i][j];
                        }
                        if(a[i][j] == a[i][j+1] && b[i][j] != b[i][j+1]){
                                b[i][j] = findMin(b[i][j], b[i][j+1]);
                                b[i][j+1] = b[i][j];
                        }
                        if(a[i][j] == a[i+1][j+1] && b[i][j] != b[i+1][j+1]){
                                b[i][j] = findMin(b[i][j], b[i+1][j+1]);
                                b[i+1][j+1] = b[i][j];
                        }
                }
        }
}

//third pass to make labels continuous
k = 0;
for(i = 0; i < N; i++)
{
        for(j = 0; j < N; j++)
        {
                if(i == 0 && j == 0){
                        c[k] = b[i][j];
                }
```

```
                        else{
                                found = 0;
                                for(p = 0; p <= k; p++)
                                {
                                        if(b[i][j] == c[p]){
                                                found = 1;
                                        }
                                }
                                if(!found){
                                        c[++k] = b[i][j];
                                }
                        }
                }
        }

        for(i = 0; i < N; i++)
        {
                for(j = 0; j < N; j++)
                {
                        for(p = 0; p <= k; p++)
                        {
                                if(b[i][j] == c[p]){
                                        b[i][j] = p;
                                }
                        }
                }
        }
}
```

- **Image Shape Recognition**

```
void  imageShape(int b[][N], int size)
{
        int area[N] = {0}, perimeter[N] = {0};
        int i, j;
        double r;

        for(i = 0; i < N; i++)
        {
                for(j = 0; j < N; j++)
                {
                        area[b[i][j]]++;
                        if(!i || i == N-1 || !j || j==N-1){
                                perimeter[b[i][j]]++;
                        }
                        else{
                                if(b[i][j] != b[i][j-1] || b[i][j] != b[i-1][j] || b[i][j] != b[i][j+1] || b[i][j] !=
b[i+1][j]){
                                        perimeter[b[i][j]]++;
                                }
                        }
                }
        }

        printf("%8s%6s%6s\n", "pixel", "area", "peri");
        for(i = 0; i < N; i++)
        {
                if(area[i]){
                        printf("%8d%6d%6d\n", i, area[i], perimeter[i]);
```

```
                }
        }

        printf("\n%8s%8s\n", "pixel", "radius");
        for(i = 0; i < N; i++)
        {
                if(area[i] && perimeter[i]){
                        r = 4 * PI * area[i] /(perimeter[i] * perimeter[i]);
                        printf("%8d%8.3f\n", i, r);
                }
        }
        printf("\n");
}
```

BIBLIOGRAPHY

Nobuyuki Otsu. *A threshold selection method from gray-level histograms*. Systems, Man and Cybernetics, IEEE Transactions on. January, 1979.
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04310076

R. Fisher, S. Perkins, A. Walker and E. Wolfart. *Connected Component Labeling*. 2003.
http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm

Tang Zhen-Jun, Zhang Xian-Quan. *A New Boundary Extraction Algorithm In A Binary Image*. Control & Automation, 10th, 2006.
http://lib.cqvip.com/read/detail.aspx?ID=23001575

EXTERNAL LINKS

- http://en.wikipedia.org/wiki/Histogram
- http://en.wikipedia.org/wiki/Otsu%27s_Method
- http://en.wikipedia.org/wiki/Connected-component_labeling
- http://ftp.utcluj.ro/pub/users/nedevschi/IP/IP_Labs_2011/ipl_06e.pdf
- http://en.wikipedia.org/wiki/Digital_image_processing