Histogram, Thresholding and Image Centroid Tutorial

Author: Bill Green (2002)
HOME EMAIL

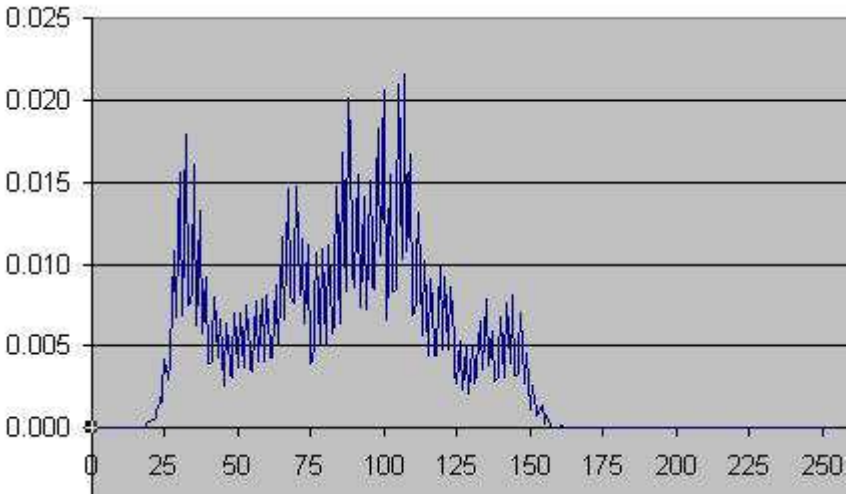
This tutorial assumes the reader knows:
(1) Data is stored left to right and bottom to top in a BMP.
(2) How to develop source code to read BMP header and info header (i.e. width, height & # of colors).
(3) How to develop source code to read raster data

INTRODUCTION

This tutorial will teach you how to:
(1) Output pixel values in histogram format to a text file.
(2) Obtain a sufficient threshold value based on the most dominant pixel value.
(3) Binarize an image based on the threshold value.
(4) Obtain the image centroid location from the binarized image.

Thresholding is an image processing technique for converting a grayscale or color image to a binary image based upon a threshold value. If a pixel in the image has an intensity value less than the threshold value, the corresponding pixel in the resultant image is set to black. Otherwise, if the pixel intensity value is greater than or equal to the threshold intensity, the resulting pixel is set to white. Thus, creating a binarized image, or an image with only 2 colors, black (0) and white (255). Image thresholding is very useful for keeping the significant part of an image and getting rid of the unimportant part or noise. This holds true under the assumption that a reasonable threshold value is chosen.

Now the question becomes, “Well how do I choose a reasonable threshold value?” That is the problem we will tackle using a histogram. We’ve all seen, or at least heard the word histogram before. In our case, our histogram is going to consist of the vertical axis being a ratio between a pixel intensity value of “x” to the total number of pixels in an image. The horizontal axis will contain the pixel value “x”. A grayscale image and its corresponding histogram are seen below to clear up any misconceptions.



Looking at the picture on the left, one can see that there are no white pixel intensity values (255). And we

can see that the histogram coincides with this observation. So rather than using trial-and-error to select an accurate threshold value, one can see from the histogram above that the most dominant pixel value, or the one occurring most throughout the image, is 107. So if we select a threshold value of 107 and make every pixel with an intensity below the threshold black, and every pixel with an intensity above the threshold white, we come up with the following binarized image:

Thus far, we have binarized an image based on what we know is an accurate threshold value in *words*. So let's try looking at a bit of code. First, lets look at my histogram program that will output histogram values to a text file.

To be compiled with Turbo C

Note: download hist.zip rather than cutting and pasting from below.

```
#include (stdio.h)
#include (stdlib.h)
#include (math.h)

/*----------STRUCTURES-----------*/
typedef struct {int rows; int cols; char *data;} sImage;

/*--------PROTOTYPES------------*/
long getImageInfo(FILE*, long, int);

int main(int argc, char* argv[])
{
  FILE                  *histogramData, *bmpInput;
  sImage                originalImage;
  unsigned char         someChar;
  unsigned char         *pChar;
  int                   r, c, i;
  int                   grayValue, nColors;
  long int              iHist[256];
  float                 hist[256];
  long int              totalNumberOfPixels;

  someChar = '0';
  pChar = &someChar;

  if(argc < 2)
  {
    printf("Usage: %s bmpInput.bmp\n", argv[1]);
    exit(0);
  }
  printf("Reading filename %s\n", argv[1]);

  /*-----DECLARE INPUT AND OUTPUT FILES-------*/
  bmpInput = fopen(argv[1], "rb");
  histogramData = fopen("histData.txt", "w");

  fseek(bmpInput, 0L, SEEK_END);

  /*------READ INPUT BMP DATA------*/
  originalImage.cols = (int)getImageInfo(bmpInput, 18, 4);
  originalImage.rows = (int)getImageInfo(bmpInput, 22, 4);
  nColors = getImageInfo(bmpInput, 46, 4);
```

```
  /*------INITIALIZE ARRAY------*/
  for(i=0; i<=255; i++) iHist[i] = 0;
  for(i=0; i<=255; i++) hist[i] = 0;
  totalNumberOfPixels = 0;

  fseek(bmpInput, (54 + 4*nColors), SEEK_SET);

  for(r=0; r<=originalImage.rows-1; r++)
  {
    for(c=0; c<=originalImage.cols-1; c++)
    {
      fread(pChar, sizeof(char), 1, bmpInput);
      grayValue = *pChar;
      iHist[grayValue] = iHist[grayValue] + 1;
      totalNumberOfPixels++;
    }
  }

  printf("Total # of pixels: %ld\n", totalNumberOfPixels);
  for(i=0; i<=255; i++)
  {
    hist[i] = (float)iHist[i]/(float)totalNumberOfPixels;
    fprintf(histogramData, "%d\t%f\n", i, hist[i]);
  }
}
```

EXPLANATION

The histogram code operates by first reading the grayscale value at the first entry and coming up with a pixel intensity between 0 and 255. It increments the total number of pixels and then it will then move on to the next row, column entry until it finishes reading all the raster data. However, while it's reading each entry, if it picks up a pixel intensity value more than once it will increment that particular value. For example, suppose there are 4 occurrences of a pixel value of 255 in an image that has a total number of pixels equal to 400.

```
grayValue = 255
iHist[255] = iHist[255] + 1 = 1
```

The first occurrence of 255 means that there have been no values read so far of iHist[255]. This means that iHist[255] is equal to zero. So moving down to the second line, we see that iHist[255] = iHist[255] + 1 = 0 + 1 = 1. It then exits the loop, moves to the next entry and does the process over again until we get to another pixel with a value of 255:

```
grayValue = 255
iHist[255] = iHist[255] + 1 = 2
```

This time iHist[255] is equal to one from haven already read in one occurrence of a 255 pixel value. So now iHist[255] = 1 + 1 = 2. The next 255 pixel value it comes to will make:

```
iHist[255] = 2 + 1 = 3
```

And finally the last occurrence of 255 will make iHist[255] = 4. After it is finished reading the rest of the values and counting them, it will then proceed to the next loop (the hist[i] loop). This loop takes the number of occurrences of a particular pixel and divides by the total number of pixels yielding a ratio. This ratio is the vertical axis of our histogram. The horizontal axis is the corresponding pixel values from 0 to 255. These values are then outputted to a ASCII text file.

So once we have selected an appropriate threshold value, we are now ready to go ahead and binarize our grayscale image. The code below looks incomplete because I left out what I have covered in my other tutorials. To threshold an image, you have to first prompt the user for a threshold value, read the pixel intensities and compare them to the specified threshold value. I did this using the following code:

To be compiled with Turbo C

Note: download bin.zip rather than cutting and pasting from below.

```
printf("Enter threshold value between 0 & 255\n");
scanf("%d", &thresholdValue);

fread(pChar, sizeof(char), 1, bmpInput);
/*-CONVERT PIXEL TO BLACK OR WHITE BASED ON THRESHOLD VALUE-*/
for(i=0; i<=vectorSize - 1; i++)
  {
    if(*pChar < thresholdValue) *(binaryImage.data + i) = 0;
    else *(binaryImage.data + i) = 255;
    fwrite((binaryImage.data + i), sizeof(char), 1, binaryOutput);
  }
```

Each time a pixel intensity value is read, the value is compared to the threshold value entered by the user. It is then made black or white and written to the “binary.bmp” file. Then the file pointer is incremented by “i” telling it to move to the next entry and begin reading.

We have completed the first 3 steps of the tutorial, so the location of the image centroid is all that is left. To find the centroid of an image, the image first has to be binarized. The centroid program will then calculate the centroid based on where the majority of the black pixels are located. If you have a completely black 20 x 20 BMP, then the centroid would be located in the exact center of that square. However, if you have an image like the one below, the centroid would be located in the center of the black square located in the top left corner.

To be compiled with Turbo C

Note: download cent.zip rather than cutting and pasting from below.

```
#include (stdio.h)
#include (stdlib.h)
#include (math.h)

typedef struct{float row; float col; float area;} imageCentroid;
typedef struct{int rows; int cols; unsigned char *data;} sImage;

long getImageInfo(FILE*, long, int);

int main(int argc, char* argv[])
{
  FILE                  *bmpInput;
  sImage                originalImage;
  unsigned char         someChar;
  unsigned char         *pChar;
  imageCentroid         ic;
  int                   r, c, nColors;

  someChar = '0';
  pChar = &someChar;

  if(argc < 2)
  {
    printf("Usage: %s bmpInput.bmp\n", argv[0]);
    exit(0);
```

```
  }
  printf("Reading filename %s\n", argv[1]);

  bmpInput = fopen(argv[1], "rb");
  fseek(bmpInput, 0L, SEEK_END);

  ic.row = ic.col = ic.area = 0.0;

  /*-------GET INPUT BMP DATA---------*/
  originalImage.cols = getImageInfo(bmpInput, 18, 4);
  originalImage.rows = getImageInfo(bmpInput, 22, 4);
  nColors = getImageInfo(bmpInput, 46, 4);

  fseek(bmpInput, (54 + 4*nColors), SEEK_SET);

  for(r=0; r<=originalImage.rows-1; r++)
  {
    for(c=0; c<=originalImage.cols-1; c++)
    {
      fread(pChar, sizeof(char), 1, bmpInput);
      if(*pChar == 0.0)
      {
        ic.row = ic.row + (originalImage.rows-1) - r;
        ic.col = ic.col + c;
        ic.area = ic.area + 1.0;
      }
    }
  }

  printf("Sum of black row pixels = %f\n", ic.row);
  printf("Sum of black col pixels = %f\n", ic.col);

  ic.row = ic.row/ic.area;
  ic.col = ic.col/ic.area;

  printf("Centroid location:\n");
  printf("row = %f\n", ic.row);
  printf("column = %f\n", ic.col);
}
```

Note the (originalImage.rows - 1) term is inserted to generate centroid location values in normal matrix form, that is the top left corner would be (0, 0). Assuming the code above is applied to a 20x20 binary BMP and the first black pixel the code comes across is located at *row 8, column 10* (going from bottom to top). Then the centroid code above would operate as follows:

```
ic.row = ic.row + (originalImage.rows - 1) - r
ic.row =   0    +        19                - 8
ic.row = 11

ic.col = ic.col + c
ic.col =   0    + 10
ic.col = 10
```

Now assuming the next black pixel comes at the entry of *row 8, column 11*. The next set of data would be:

```
ic.row = ic.row + (originalImage.rows - 1) - r
ic.row =   11   +        19                - 8
ic.row = 22

ic.col = ic.col + c
ic.col =   10   + 11
ic.col = 21
```

The code continues on this way and adds up all the rows and columns that contain a black pixel. It takes the sum of all the row entries and divides by the area to get the *row location* of the centroid. For example, if the 2 black pixels from above were the only 2 in the entire image, then the sum of the row entries would

be equal to 33. You would then take 33 divided by the area (2 in this case) to get the row location. Similarly, it takes the sum of all the column entries and divides by the area to obtain the *column location* of the centroid. The area calculation is based on the principle that one black pixel has an area of 1. So every time the code enters the loop, the area is incremented by 1.

You are visitor number: