Assignment 3

5140379043 盛思远

Part 1 source code

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>

using namespace cv;
using namespace std;

const int MAX_GRAY_VALUE = 256;
const int MIN_GRAY = 90;//remove the effect of background

int main(int argc, char** argv)
{
    IplImage* image = cvLoadImage("cherry.png", 0);//load image with gray model

    //draw histogram and show it
    int bins = 256;
    int hist_size[] = { bins };
    float range[] = { 0, 256 };
    const float* ranges[] = { range };
    MatND hist;//hist data
    int channels[] = { 0 };
    Mat gray(image, 0);
    //calculate hist and store the data into dist
    calcHist(&gray, 1, channels, Mat(), // do not use mask
        hist, 1, hist_size, ranges,
        true, // the histogram is uniform
        false);
    double max_val = 0;
    //minMaxLoc(hist, 0, &max_val, 0, 0);//calculate the max of histogram(max num
of occurences)
    for (int i = MIN_GRAY; i < MAX_GRAY_VALUE; i++)
    {
        if (hist.at<float>(i) > max_val)
        {
            max_val = hist.at<float>(i);
        }
    }
    int scale = 2;//width of hist
    int hist_height = 256;//height of hist
```

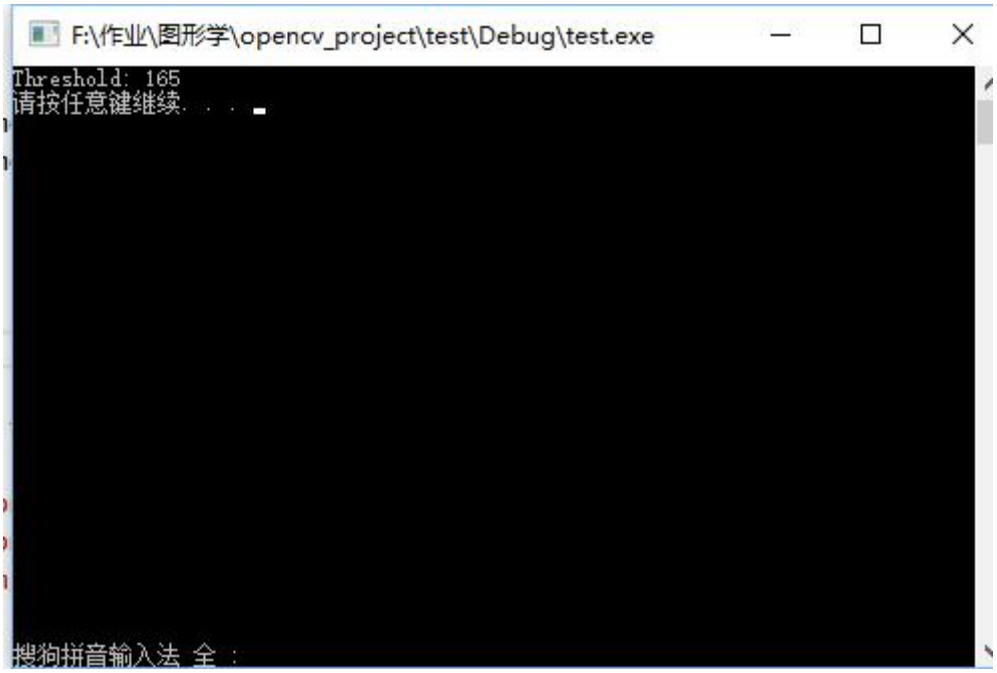
```
    Mat hist_img = Mat::zeros(hist_height, bins*scale, CV_8UC3);//image of
histogram
    //load the data from hist to image
    for (int i = MIN_GRAY; i<bins; i++)
    {
        float bin_val = hist.at<float>(i);
        int intensity = cvRound(bin_val*hist_height / max_val);  //要绘制的高度
        rectangle(hist_img, Point(i*scale, hist_height - 1),
            Point((i + 1)*scale - 1, hist_height - intensity),
            CV_RGB(255, 255, 255));
    }

    double histogram[MAX_GRAY_VALUE];//histogram for probability of gray value
    for (int i = 0; i < MAX_GRAY_VALUE; i++)//initialize the histogram
    {
        histogram[i] = 0;
    }

    //update the histogram
    for (int row = 0; row < image->height; row++)
    {
        uchar* ptr = (uchar*)image->imageData + row*image->widthStep;
        for (int col = 0; col < image->width; col++)
        {
            int temp_gray_value = ptr[col];
            histogram[temp_gray_value]++;
        }
    }

    //normalization
    int totalpoints = 0;
    for (int i = MIN_GRAY; i < MAX_GRAY_VALUE; i++)
    {
        totalpoints += histogram[i];
    }
    for (int i = MIN_GRAY; i < MAX_GRAY_VALUE; i++)
    {
        histogram[i] = histogram[i] / totalpoints;
    }

    //w0:percentage of foreground
    //w1:percentage of background
    //u0:average gray value of foreground
    //u1:average gray value of background
```

```
    //u:global average gray value
    //tips:w0 + w1 = 1 and u0 + u1 = u = sum(histogram[i]*i)
    double w0 = 0, w1 = 0, u0 = 0, u1 = 0, u = 0;;
    for (int i = MIN_GRAY; i < MAX_GRAY_VALUE; i++)//initialize u
    {
        u = u + i*histogram[i];
    }

    //search optimal threshold values
    int threshold = -1;
    double max_variance = -1;
    for (int index = MIN_GRAY; index < MAX_GRAY_VALUE; index++)
    {
        w0 = w0 + histogram[index];//update percentage of foreground
        w1 = 1 - w0;//update percentage of background
        u0 = u0 + histogram[index] * index;//update average gray value of foreground
        u1 = u - u0;//update average gray value of background

        double temp_variance = w0*(u0 - u)*(u0 - u) + w1*(u1 - u)*(u1 - u);
        if (temp_variance > max_variance)
        {
            max_variance = temp_variance;
            threshold = index;
        }
    }

    //create foreground and background pictures
    IplImage* foreground_image = cvCloneImage(image);
    IplImage* background_image = cvCloneImage(image);
    for (int row = 0; row < foreground_image->height; row++)
    {
        uchar* foreground_ptr = (uchar*)foreground_image->imageData +
row*foreground_image->widthStep;
        uchar* background_ptr = (uchar*)background_image->imageData +
row*background_image->widthStep;
        for (int col = 0; col < foreground_image->width; col++)
        {
            int temp_gray_value = foreground_ptr[col];
            if (temp_gray_value > MIN_GRAY)
            {
                if (temp_gray_value > threshold)
                {
                    foreground_ptr[col] = 255;
                    background_ptr[col] = 0;
```

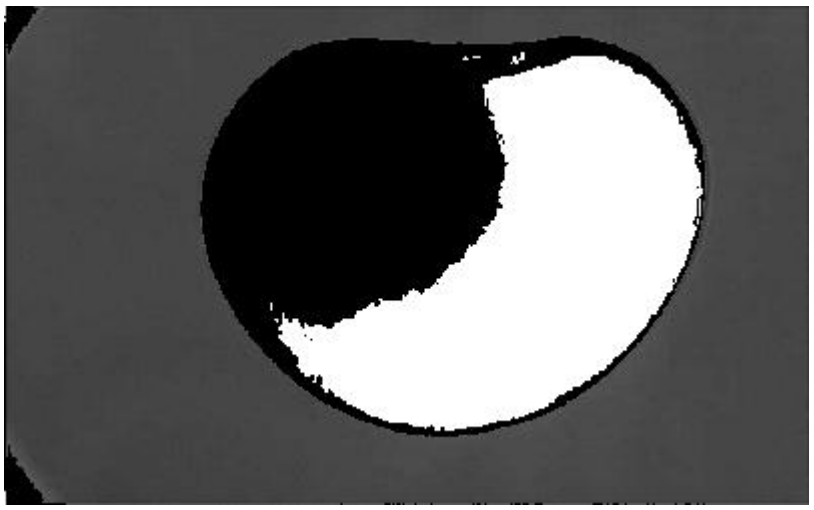
```
                }
                else
                {
                    foreground_ptr[col] = 0;
                    background_ptr[col] = 255;
                }
            }
        }
    }

    //output the threshold
    cout << "Threshold: " << threshold << endl;

    cvSaveImage("foreground.png", foreground_image);
    cvSaveImage("background.png", background_image);
    cvSaveImage("histogram.png", &IplImage(hist_img));

    cvReleaseImage(&image);
    cvReleaseImage(&foreground_image);
    cvReleaseImage(&background_image);
    system("pause");
}
```

Part 2 output

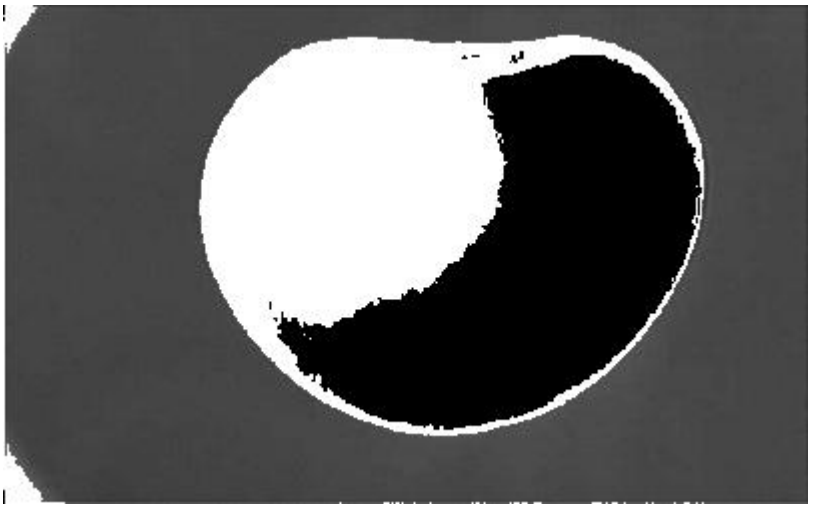
(1)Threshold



(2)Foreground



(3)Background



(4)Histogram