1. Problem

In this homework, we perform the Otsu’s algorithm to segment out the interest region form a color image of the Lake Tahoe. Then extract the contour of the lake.

2. Otsu’s algorithm in color space

a. Split the R, G, and B channels

Since the input is a color image, while the original Otsu’s algorithm is implemented on a grayscale image, we first need to split the 3 channels out of the color image. This is done by calling cvSplit from openCV.

b. Find out the best segmentation thresholds for the three channels respectively using Otsu’s method.

For a single channel, the optimal threshold \( k^* \) that maximizes the between-class variance, \( \sigma_B^2 \), is selected. In this way, the two classes separated have the most significant distinguishes.

We compute \( \sigma_B^2 \) using the formulas below. First, define \( L \) to be the number of gray levels the image is represented in. The number of pixels at level \( I \) is donated by \( n_i \). The total number of pixels is \( N \). The probability that a pixel is in gray level \( i \) is,

\[
p_i = \frac{n_i}{N}.
\]

The total mean level of the image is,

\[
\mu_T = \sum_{i=1}^{L} i \cdot p_i
\]

If level \( k \) is the threshold, the between-class variance can be expressed as

\[
\sigma_B^2 = \omega_0 (\mu_0 - \mu_T)^2 + \omega_1 (\mu_1 - \mu_T)^2
\]

\[
= \omega_0 \omega_1 (\mu_1 - \mu_0)^2
\]

where

\[
\omega_0 = \sum_{i=1}^{k} p_i, \quad \omega_1 = 1 - \omega_0, \quad \mu_0 = \frac{\sum_{i=1}^{k} i \cdot p_i}{\omega_0}, \quad \mu_1 = \frac{\sum_{i=k+1}^{L} i \cdot p_i}{\omega_1}
\]

Now, we construct the zeroth and first-order moment of the histogram up to level \( k \).

\[
m_0(k) = \sum_{i=1}^{k} p_i, \quad m_1(k) = \sum_{i=1}^{k} i \cdot p_i
\]

So that,

\[
\omega_0 = m_0(k), \quad \omega_1 = 1 - m_0(k), \quad 
\mu_0 = \frac{m_1(k)}{m_0(k)}, \quad \mu_1 = \frac{\mu_T - m_1(k)}{1 - m_0(k)}
\]

Substitute all the parameters into \( \sigma_B^2 \), then we have,
\[ \sigma_B^2 = m_0(k)(1 - m_0(k)) \left[ \frac{\mu_T - m_1(k)}{1 - m_0(k)} - \frac{m_1(k)}{m_0(k)} \right]^2 \]

The above equation is calculated for every gray level \( k, k \in [1, \ldots, L] \). We choose \( k \) as the one whose \( \sigma_B^2 \) is the maximum among all the values.

c. **Merge the results of three channels together.**
For our given image, if all of the R, G, and B values of a pixel are smaller than their corresponding threshold, it is a pixel belongs to the lake region. Otherwise, it is the background. Since the lake is blue, the result of B channel is the closest to the merged result.

d. **Extract the contour of the lake.**
From step c, we are able to get a binary mask, which indicates the lake (pixel value 0) and the background (pixel value 255). We perform a 4-neighborhood border detector to capture the contour. The contour must belong to the lake, and connect to the background in 4-neighborhood.
So the strategy is:

for \( i = 1: \text{number of pixels in the image} \)
    if (the value of pixel \( i \) is 0 &\& at least one of the 4-neighborhood has pixel value 255)
        pixel \( i \) is on the contour;

However, we detect a lot of noise pixels, the single dots, which should not belong to the contour. We further remove those noise pixels by checking whether the pixel is connected to other pixels in the lake region, and remove the single dots.
3. The segmented results with the binary image and the contour.

(a) The original image

(b) The lake region only

(c) The binary image with the lake region. Merged results of the three channels

(d) The contour of the lake (black line)
(e) R channel binary image
(f) G channel binary image
(g) B channel binary image
4. Code

```cpp
#include <cv.h>
#include <highgui.h>
#include <iostream>
#include <opencv2/core/core.hpp>
using namespace cv;
using namespace std;
#include "Otsu.h"

int main(int argc, char** argv)
{
    char* inImagename;
    inImagename = argv[1];
    IplImage* inImage = cvLoadImage(inImagename,1);
    int i,j,k;
    if(!inImage)
    {
        cout<<"Could not load image file!"<<endl;
        return 1;
    }

    IplImage* lake_region = cvCreateImage(cvGetSize(inImage),IPL_DEPTH_8U,3);
    cvCopy(inImage, lake_region, NULL);
    uchar* data_region = (uchar *)lake_region->imageData;

    IplImage* lake_contour = cvCreateImage(cvGetSize(inImage),IPL_DEPTH_8U,3);
    cvCopy(inImage, lake_contour, NULL);
    cvZero(lake_contour);
    uchar* data_contour = (uchar *)lake_contour->imageData;

    // split image into 3 color plans
    IplImage* r = cvCreateImage(cvGetSize(inImage), IPL_DEPTH_8U, 1);
    IplImage* g = cvCreateImage(cvGetSize(inImage), IPL_DEPTH_8U, 1);
    IplImage* b = cvCreateImage(cvGetSize(inImage), IPL_DEPTH_8U, 1);
    cvSplit(inImage, r,g,b,NULL);

    // compute the otsu threshold
    int threshR, threshG, threshB;
    threshR = Otsu(r);
    threshG = Otsu(g);
    threshB = Otsu(b);

    // merge the three color plane together
    uchar* datar = (uchar*)r->imageData;
    uchar* datag = (uchar*)g->imageData;
    uchar* datab = (uchar*)b->imageData;
    int width = inImage->width;
    int height = inImage->height;
    int step = r->widthStep;
    int Step = inImage->widthStep;

    // creat the binary mask and merge the 3 filtered color plane together
    IplImage* binary = cvCreateImage(cvGetSize(inImage), IPL_DEPTH_8U, 1);
    uchar* databinary = (uchar*)binary->imageData;
```
cvZero(binary);

for (i=0;i<width;i++)
{
    for (j=0;j<height;j++)
    {
        //Merge the three channel together
        if (datar[j*step+i]>thresHr || datag[j*step+i]>thresHg ||
            datab[j*step+i]>thresHb)
        {
            databinary[j*step+i] = 255;
            for (k=0;k<3;k++)
                data_region[j*Step+i*3+k] = 0;
        }
        //segment the three channels
        if (datar[j*step+i]>thresHr)
            datar[j*step+i]=255;
        else
            datar[j*step+i]=0;
        if (datag[j*step+i]>thresHg)
            datag[j*step+i]=255;
        else
            datag[j*step+i]=0;
        if (datab[j*step+i]>thresHb)
            datab[j*step+i]=255;
        else
            datab[j*step+i]=0;
    }
}

//save the binary image for the three channels
char rname [128];
sprintf(rname, "r.png");
cvSaveImage(rname,r);
char gname [128];
sprintf(gname, "g.png");
cvSaveImage(gname,g);
char bname [128];
sprintf(bname, "b.png");
cvSaveImage(bname, b);

//save the output image
char binaryname [128];
sprintf(binaryname, "binary.png");
cvSaveImage(binaryname, binary); // cvReleaseImage(&binary);
char outname1 [128];
sprintf(outname1, "lake_region.png");
cvSaveImage(outname1, lake_region); // cvReleaseImage(&inImage);
// cvNamedWindow("originalx",CV_WINDOW_AUTOSIZE);
// cvShowImage("originalx",binary);
// cvWaitKey(-1);

//extract the contour of the image
for (i=0;i<width;i++)
{...
for(j=0;j<height;j++)
{
    if(databinary[j*step+i]==0 )
    {
        //check the 4-neighborhood
        {
            {
                for(k=0;k<3;k++)
                {
                    data_contour[j*Step+i*3+k] = 0;
                }
            }
        }
    }
}

char outname2 [128];
sprintf(outname2, "lake_contour.png");
cvSaveImage(outname2, lake_contour);
cvReleaseImage(&inImage);

#include <cv.h>
#include <highgui.h>
#include <iostream>
#include <opencv2/core/core.hpp>
using namespace cv;
using namespace std;

int Otsu(IplImage *image)
{
    int i,j;
    float width = image->width;
    float height = image->height;
    int numBins = 256;
    float range[] = {0, 255};
    float *ranges[] = { range };
    CvHistogram *hist = cvCreateHist(1, &numBins, CV_HIST_ARRAY, ranges, 1);
    cvClearHist(hist);

    // compute the histogram of the image
    cvCalcHist(image, hist, 0, 0);
// compute the uT
float uT=0;
for(j=0;j<256;j++)
{
    float histValue = cvQueryHistValue_1D(hist,j);
    uT = uT+histValue*(float)j/width/height;
}

// compute zeroth and first-order moment
float m0, m1;
float maxVb=0;
float Vb;
int threshH = 0;
for (i = 1;i<255;i++)
{
    m0 =0;
    m1 = 0;
    Vb = 0;
    for(j=0;j<i+1;j++)
    {
        float histValue = cvQueryHistValue_1D(hist,j);
        m0 = m0+histValue/width/height;
        m1 = m1+histValue*(float)j/width/height;
    }
    Vb = m0*(1-m0)*pow((uT-m1)/(1-m0)-m1/m0,2);
    if (Vb>maxVb)
    {
        maxVb = Vb;
        threshH = j;
    }
}
return threshH;