```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include <stdlib.h>


typedef struct THR {
    int B;
    int x;
} THR;


using namespace std;
using namespace cv;


void thresholding (const char *);
int findBackground(int *, int);
void drawHistogram(IplImage *, int *);
IplImage * simpleThresholding(IplImage *, int);
IplImage * getBinImage(IplImage *, unsigned);


int main(int argc, char ** argv) {
    const char * imgName = argv[1];
    thresholding(imgName);
    return 0;
}



/// ////////////////////////////////////////////////////////////////
/// ////////////////////////////////////////////////////////////////
void thresholding(const char * imgName) {

    ///                ORIGINAL IMAGE UPLOAD
    IplImage * img = cvLoadImage(imgName, CV_LOAD_IMAGE_GRAYSCALE);

    ///                       HISTOGRAM
    int dep = 256;
    int hist[dep];
    drawHistogram(img, hist);


    ///                COMPUTE THE BACKGROUND = B
    int B = findBackground(hist, img->height * img->width);

    ///                 COMPUTE THE THRESHOLD = t
    int t = 10;

    ///                 FINAL THRESHOLD T = B + t
    int T = t + B;

    ///                    USE T TO THRESHOLD
    IplImage * bin;
    bin = simpleThresholding(img, T);


```

```
59
60      ///                             SAVE
61      cvSaveImage("Binary_img.png", bin);
62
63
64      ///                            CLEAN-UP
65      cvReleaseImage(&img);
66      cvReleaseImage(&bin);
67
68  }
69
70
71  /// ////////////////////////////////////////////////////////////////////
72  /// ////////////////////////////////////////////////////////////////////
73
74
75      /**************************************************************************
76      * Just a support function to 'drawHistogram', 'drawHistChannels', and     *
77      * 'getHistogram'.                                                         *
78      **************************************************************************/
79  void func(uchar c, int * h) {
80      unsigned x = c;
81      (*(h + x))++;
82  }
83
84
85  /// ////////////////////////////////////////////////////////////////////
86  /// ////////////////////////////////////////////////////////////////////
87
88
89  void drawHistogram(IplImage * gray, int hist []) {
90      int st = 4;       // To separate lines in the histograms, for better look ;)
91      int dep = 256;
92          /// Initialize histograms
93      for (int i = 0; i < dep; i++)
94          hist[i] = 0;
95
96          /// Calculate histogram
97      for(int i = 0; i < gray->height; i++) {
98          char * ptr = gray->imageData + i*gray->widthStep;
99          for(int j = 0; j < gray->width; j++) {
100                 func((uchar)(*ptr), hist);
101                 ptr++;
102         }
103     }
104
105         /// Create image for the histogram
106     IplImage * his = cvCreateImage(cvSize(st*dep,600), IPL_DEPTH_8U, 1);
107     cvSet(his, 0);                  // Initialize image (all black)
108     his->origin = IPL_ORIGIN_BL;    // Set the origin in the bottom left corner
109
110     //cvNamedWindow("histogram", 2);         // Create window to contain the image
111
112         /// Draw the histogram - 2 different styles
113     for (int i = 0; i < dep; i++)
114         if (hist[i] != 0)
115             cvLine(his, cvPoint(i*st, 0), cvPoint(i*st, hist[i]/10), 150, 1, 4);
116
```

```
117     cvSaveImage("Histogram.png", his);
118     //cvShowImage("histogram", his);
119 
120     cvWaitKey(0);
121 
122     cvReleaseImage(&his);
123 }
124 
125 
126 /// ////////////////////////////////////////////////////////////////////////
127 /// ////////////////////////////////////////////////////////////////////////
128 
129 
130 int findBackground(int hist[], int tot) {
131     bool fmin = true, fmax = true;
132     int m, M;
133 
134 
135     for (int i = 0; fmin || fmax; i++) {
136         if (fmin)
137             if(hist[i]){
138                 m = i;
139                 fmin = false;
140             }
141         if (fmax)
142             if(hist[255 - i]){
143                 M = 255 - i;
144                 fmax = false;
145             }
146     }
147 
148 
149     THR res_t;
150     res_t.B = m + (M - m) / 16;
151 
152     int frac = 45, cnt = 0;
153 
154     for(int i = m; i <= res_t.B; i++) {
155         cnt += hist[i];
156     }
157 
158     if( (res_t.x = ((cnt * 100) / tot)) == frac)
159         return res_t.B;
160     else if (res_t.x < frac) {
161         while(res_t.x < frac) {
162             res_t.B++;
163             cnt += hist[res_t.B];
164             res_t.x = (cnt * 100) / tot;
165         }
166         THR t1, t2;
167         t1.B = res_t.B;
168         t1.x = res_t.x;
169         t2.B = res_t.B - 1;
170         t2.x = ((cnt - hist[t2.B])* 100) / tot;
171         if (abs(t2.x - frac) < abs(t1.x - frac) ) {
172             printf("\nPercentile: %d%%    ,    Background: %d\n", t2.x, t2.B);
173             return t2.B;
174         }
```

```
175
176         else {
177             printf("\nPercentile: %d%%    ,    Background: %d\n", t1.x, t1.B);
178             return t1.B;
179         }
180
181     }
182     else { // x > frac
183         while(res_t.x > frac) {
184             res_t.B--;
185             cnt -= hist[res_t.B];
186             res_t.x = (cnt * 100) / tot;
187         }
188         THR t1, t2;
189         t1.B = res_t.B;
190         t1.x = res_t.x;
191         t2.B = res_t.B + 1;
192         t2.x = ((cnt + hist[t2.B])* 100) / tot;
193         if (abs(t2.x - frac) < abs(t1.x - frac) ) {
194             printf("\nPercentile: %d%%    ,    Background: %d\n", t2.x, t2.B);
195             return t2.B;
196         }
197         else {
198             printf("\nPercentile: %d%%    ,    Background: %d\n", t1.x, t1.B);
199             return t1.B;
200         }
201     }
202 }
203
204
205
206 /// ////////////////////////////////////////////////////////////////////////
207 /// ////////////////////////////////////////////////////////////////////////
208
209
210
211 IplImage * simpleThresholding(IplImage * G, int th) {
212     IplImage * bin;
213
214     bin = getBinImage(G, th);
215     cvNamedWindow("Binary Image", 0);
216     cvShowImage("Binary Image", bin);
217
218     return bin;
219 }
220
221
222 /// ////////////////////////////////////////////////////////////////////////////
223 /// ////////////////////////////////////////////////////////////////////////////
224
225
226 IplImage * getBinImage(IplImage * img, unsigned th) {
227     IplImage * I = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
228     for(int i = 0; i < img->height; i++) {
229         char * ptr = img->imageData + i*img->widthStep;
230         char * p = I->imageData + i*I->widthStep;
231         for(int j = 0; j < img->width; j++) {
232                 if ((uchar)(*ptr) >= (uchar)th)
```

```
233                     *p = 255;
234                 else
235                     *p = 0;
236             ptr++;
237             p++;
238         }
239     }
240     return I;
241 }
```