Object Pair Recognition for Calculating Angle of Illumination

by Group 4: Arjun Jauhari (aj526) and Tae Jong (Eric) Jung (tj93)

Abstract

This project outlines a method of recognizing and segmenting an object-shadow pair, originally intended for calculating the angle of illumination. We attempt to tackle this segmentation task by composing different simple algorithms together. We design an algorithm using region growing and adaptive multi-level thresholding to segment an object-shadow pair, and we use a visionX program vrdiff to evaluate and compute statistics on the accuracy of the algorithm. We gathered simplistic images with single image to test our algorithm first, and planned to test it further on more complicated images, but didn't have enough time. We were able to get workable segmentation algorithm, good enough, with adjustments, to be used for calculating angle of illumination for future work.

1. Introduction

Object-shadow detection has been an interest for computer vision for a long time. For temporal tracking, it is often beneficial to identify and remove shadows, as they have similar temporal behavior as the object, and therefore are likely to be classified as such, even if it is not. Our main motivation behind this project was to enable calculating the angle of illumination, which could provide a useful metadata to the photograph.

There are many methods and approaches to detecting shadows. Lalonde et al. [2] uses learning based approach, on the assumption that material consisting the "ground" of the setting is often limited. Later, Lalonde et al. [4] uses probabilistic approach to approximate the probability of shadow existing.

We employ a region growing algorithm, maintaining the base color, to first segment the images, and perform adaptive automatic thresholding with the base color values. Before we expand further on our approach, we first explore the uses of object-shadow pair recognition and expand on previous work done on this subject.

1.1 Background

Main motivation behind this specific type of segmentation is to calculate angle of illumination. By obtaining clear object-shadow pair, we can use analytic methods to infer angle of illumination from the relative angles of the object and the shadow. However, actual identifying task has been left for future work, as we wanted to make sure that the object-shadow pair segmentation worked well enough before moving on to application.

We combine two simple techniques to segment an object-shadow pair. First, we use region growing to segment the input image into regions, then we adaptively threshold the regions using histogram information.

Once we have solid segmentation function and can reliably obtain object-shadow pairs, we can use this to compute various different things. To calculate the angle of illumination, one could potentially find the relative angle between the object and the shadow, calculate the corresponding ratio of length of the two, and using these information and simple trigonometry.

Being able to calculate angle of illumination could be very helpful. For one, it could enable a crude sundial digitally, or it could be used to adjust solar panels for maximum energy generation. Being able to do this reliably could be another small step towards environment-aware AI.

1.2 Previous Work

Major part of this project is to identify object-shadow pairs. Lot of work has been done on shadow removal for efficient object tracking over the past decade, and the first step in shadow removal is to detect shadows reliably. Therefore all this work becomes relevant for this project.

So what is a shadow? How it gets generated? What are its features or properties? All these questions needs to be answered to write an efficient algorithm for shadow detection. Below paragraph aims to answer these questions.

A shadow occurs when an object partially or totally occludes direct light from a source of illumination. Shadows can be divided into two classes: self and cast shadows. A self shadow occurs in the portion of an object which is not illuminated by direct light. A cast shadow is the area projected by the object in the direction of direct light. We are mainly interested in cast shadow as that is necessary and sufficient to detect the angle of illumination.

Some features of shadows that can help its detection are:

Intensity: Generally the pixels under shadow are darker as compared to other background pixels. This can be used to get a rough estimate of shadow region.

Geometry: This can be used to split the object and its shadow by calculating its center of gravity and orientation through moments.

Chromacity: The chromacity of pixel remain same with or without shadow but they become darker. This property can be used if working on color images. Furthermore, this feature requires a comparison between image with shadow and without shadow. Therefore, it might not be suitable for this project as the stated comparison might not be possible because of unavailability of image without shadow.

There are many other shadow features like physical properties, textures, etc. which help in detection but most of them require a comparison with reference background image without shadow/object which might not be possible in the case of still images. Therefore, intensity and geometry are the only suitable features to use for single image shadow detection as they can be implemented directly using input frame. But these features alone are not guaranteed to result in a good shadow detection algorithm. This restriction makes the shadow detection an extremely challenging problem in single images. Next, we review the recent work done in literature in regard with this problem.

Review of recent papers on shadow detection:

- Shadow detection using paired regions is proposed by Guo et al. [1], where they compare two regions of the image that are likely to be of the same material. First, they find pairs of regions that are likely to be of same material and determine whether they have the same illumination condition. Next they make a graph using these region pairs together with the region-based appearance features. Node in graph encodes region appearance and edge indicated whether the nodes with same surface are under same or different illumination. Finally, the regions are classified into shadow or non-shadow by doing graph-cut inference on the graph generated in previous steps. Relevant contribution of this paper with respect to this project, (1) a new method for detecting shadows using a relational graph of paired regions, (2) quantitative evaluation of shadow detection with comparison to existing work, (3) image dataset to test shadow removal.

- Lalonde et al. [2] in their paper “Detecting Ground Shadows in Outdoor Consumer Photographs”, focus on shadows cast by objects onto the ground plane. They work around the assumption that the types of materials constituting the ground in typical outdoor scenes are relatively limited, most commonly including concrete, asphalt, grass, mud, stone, brick, etc. which is reasonable enough. Therefore with these assumptions, the appearances of shadows on the ground are not as widely

varying as the shadows everywhere in the scene. They take learning based approach from a set of labelled images of real world scenes. Further, they divide the whole approach into 3 stages (1) they exploit local information around edges in the image. This is done by computing a set of shadow sensitive features that include the ratios of brightness and color filter responses at different scales and orientations on both sides of the edge. These features are then used with a trained decision tree classifier to detect whether an edge is a shadow or not, (2) they enforce a grouping of the shadow edges using a Conditional Random Field (CRF) to create longer contours, (3) they incorporate a global scene layout descriptor with their CRF, such as the 3-way ground-vertical surface-sky classifier by Hoiem et. al [3].

- Lalonde et al. [4] in their work in "Estimating Natural Illumination from a Single Outdoor Image" propose a method for estimating the likely illumination conditions of the scene. In particular, they compute the probability distribution over the sun position and visibility. They take probabilistic approach, because the information for precise estimate of illumination condition from single image vary from image to image. The method relies on a combination of weak cues that can be extracted from different portions of the image: the sky, the vertical surfaces, and the ground. The appearance of the sky can tell if it's clear or overcast. On a clear day, the sky might give some weak indication about the sun position. The presence of shadows on the ground plane can inform about sun visibility, while the direction of shadows cast by vertical structures can tell about sun direction. Since each of these cues by itself is rather weak and unreliable, they combine the information obtained from these weak cues together with a prior computed over a set of 6 million Internet photographs while doing estimation on new image.

- Slavador et al. [5] in their paper "Cast shadow segmentation using invariant color features" propose an image understanding system for the detection of cast shadows which exploits shadow's spectral and geometric properties. The analysis is organized in three stages. A hypothesis about the presence of a shadow is first generated on the basis of an initial and simple evidence, i.e. shadows normally darken the surface upon which they are cast. The validity of this hypothesis is further verified on each detected region by making use of hypotheses on color invariance and geometric properties of shadows. Finally, an information integration stage confirms or rejects the initial hypothesis for every detected region. The system is conceived to work with uncalibrated images from different sources and with objects of different nature.

1.3 Overview

We will now describe the algorithm used to segment the images, followed by experiments designed to test the segmentation accuracy. Data set and the procedure of experiments will be detailed, followed by results, discussion on the results, and finally conclusion.

2. Methods

We used modular designing approach by dividing the project into two parts: shadow detection and periphery calculation. Main part of this project will be shadow detection. There are many researches, as can be seen from above, that relates to this, and there are myriads of methods suggested for different situations. We decided to limit our project to single images, not choosing to do any image sequences or temporal analysis. The reason is that we might not have a good moving object to use as a reference all the time, rending our application useless. Most of our time will be devoted to implementing a good shadow detector, as the periphery calculation should not be too intensive, once we have a good object-shadow pair.

Below we describe the developed algorithm and the experiments conducted to evaluate the performance of the algorithm.

2.1 Algorithm

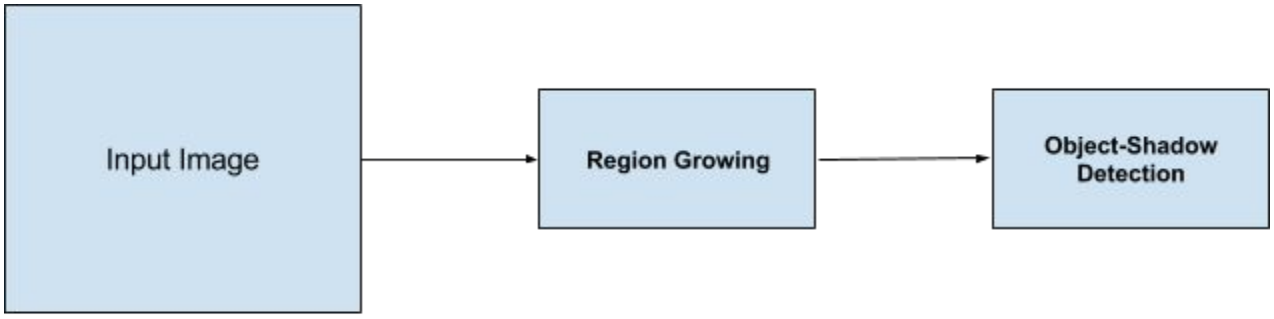


Figure 1: Algorithm Overview

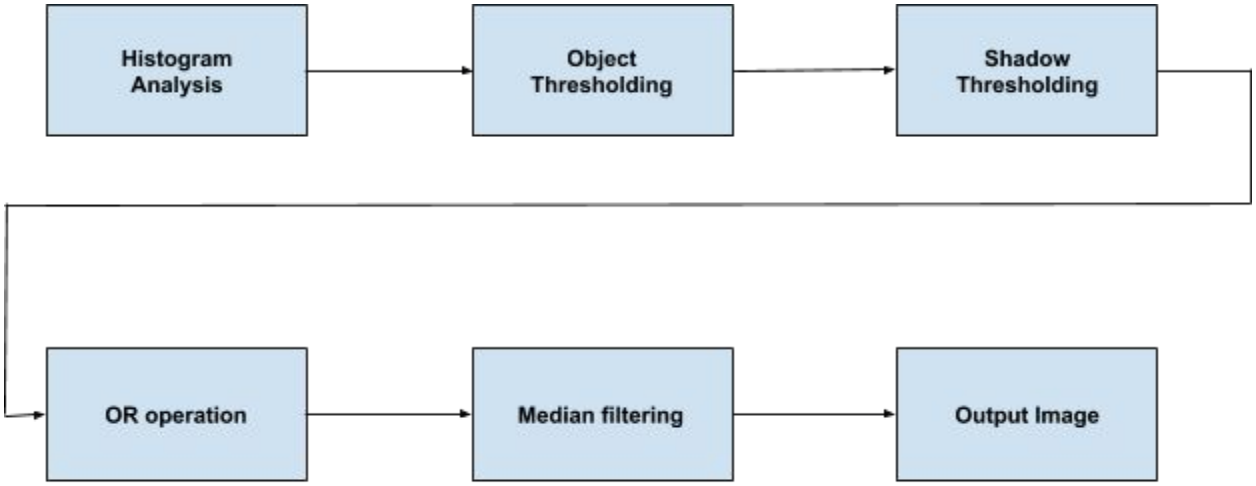


Figure 2: Object-Shadow Detection

Our algorithm first do region growing on input image followed by object-shadow detection as shown in Figure 1 above. Object-shadow detection is the key part of the overall algorithm which is explained in detail below.

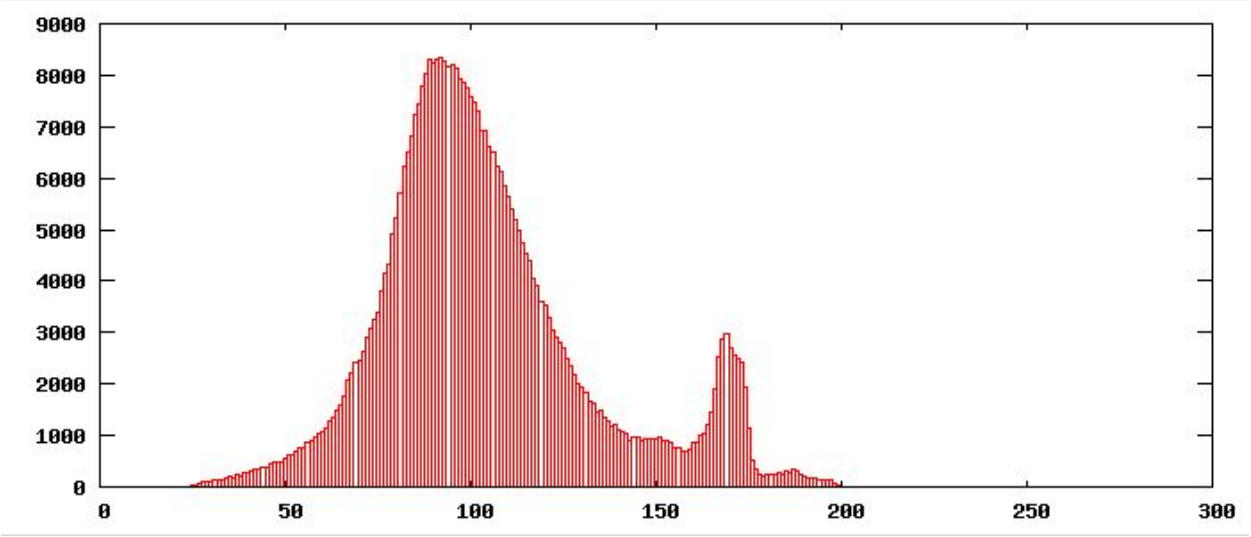


Figure 3: Original histogram

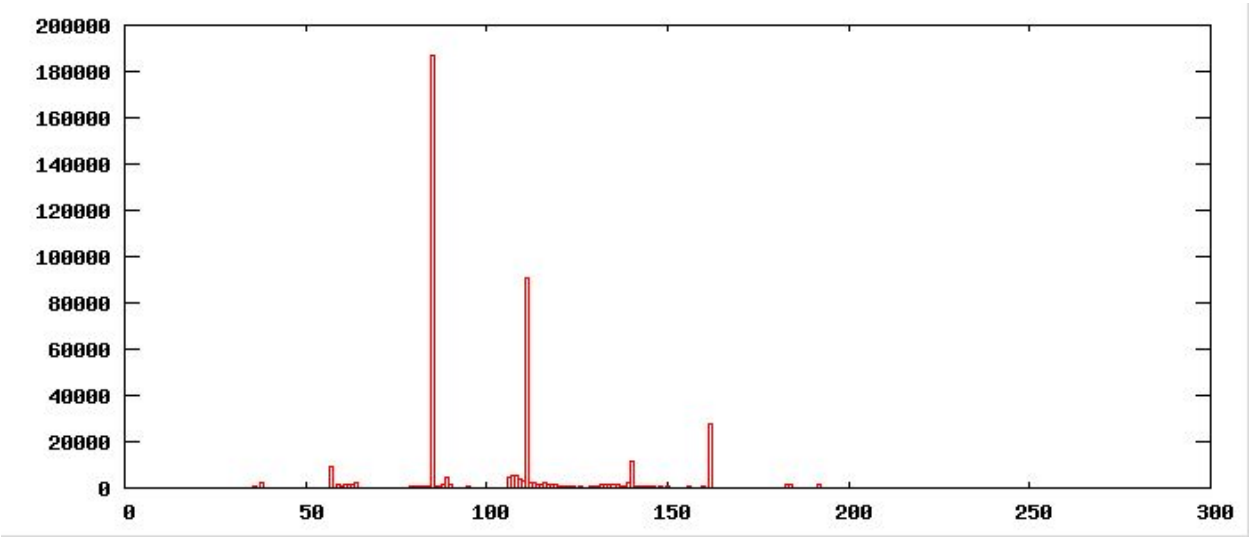


Figure 4: Histogram after Region Growing

Region growing divides the image into 3 major regions: Object, Shadow, and Background. For example, Figure 3 above shows a histogram of original image and Figure 4 shows the histogram after Region Growing. It can be seen that after region growing the histogram splits into 3 major regions. Now the output of region growing is used to do object shadow segmentation.

We now explain each of the block in Figure 2 above in detail -

Histogram Analysis: In this step we search for a cutoff value on number of pixels which we use to find the number of significant peaks in the histogram. Any histogram bin whose value is more than cutoff value is classified as peak. This search is done based on following constraints -

1. Number of peaks needs to be between bounded between maximum number of peaks and minimum number of peaks.
2. Distance between two peaks has to be greater than user specified constant(dist).

Below is pseudo-code for the same -

ALGORITHM

1: Initialize cutoff value
2: Calculate number of peaks above cutoff
3:
if (# of peaks) > maxPeaks
    reduce cutoff by 50% and goto STEP 2
else if (# of peaks) < minPeaks
    increase cutoff by 10% and goto STEP 2
end if
4:
if (distance between two consecutive peaks) < min distance
    increase cutoff by 10% and goto STEP 2
end if
5: **return** cutoff value

Object Thresholding: After we have found peaks from Histogram Analysis step above, we use those peaks to find the peak belonging to object. We assume that the rightmost peak would be that of object since object tends to be most bright in the image. With this assumption we threshold the image with threshold set between rightmost two peaks.

Shadow Thresholding: This part is similar to above object thresholding part. Here we assume shadow peak to be the leftmost peak. Thresholding is down with threshold set between the leftmost two peaks.

OR: In this step we OR the two images we got from above object and shadow thresholding steps.

Median filtering: We do median filtering on the OR-ed image using VisionX inbuilt median filter function. This is done mainly to remove noise from the segmented image.This filtered image is the output of our algorithm.

2.2 Experiment

In this section we describe the experiments conducted to find the best possible parameters for our algorithm so that it gives best output on various types of images.

The parameter to be tuned are -

1. threshold for region growing
2. Maximum number of peaks
3. distance between peaks

2.2.1 Hypothesis and evaluation function

The three parameters mentioned above affect the output in different ways. First parameter above governs how many different region the image will be divided into which is very important as we don't want too many regions neither do we want too few. Ideal number of regions would be 3 i.e one for object, shadow and background each. But in real scenarios that might not be possible always. Hence proper tuning of this parameter is necessary.

Second and third parameters are also very important as they regulates the number of valid peaks we find after region growing. The quality of threshold for object or shadow thresholding directly depends on

this. Improper setting of these parameter will result in wrong threshold and the resulting object shadow segmentation will be not as expected.

Our evaluation framework is quite simple. Since we are comparing segmented images, we use annotated image set as ground truth and use function vrdiff to evaluate multiple different measures(including dice similarity coefficient and fraction of pixels correct) of the two image similarities and differences, which we can use to determine the quality of segmentation algorithm.

For instance, if A is annotated ground truth and B is the output of algorithm then Dice coefficient is calculated as -

$$DC = \frac{2*|A \cap B|}{|A|+|B|}$$

2.2.2 Documented Data Set

Our image dataset consist of 20 hand picked diverse images which represents common real world scenarios. Originally the image were colored but we decided to work on gray-scale images as we don't use color information in our object-shadow detection algorithm. So we converted colored images into grayscale images using VisionX. Some of the grayscale images from dataset are shown below -

Image 1: A dog standing on sand

Image 2: Hydrant

Image 3: A man on road

Image 4: Multiple people on ground

We annotated all the images in the dataset using VisionX which serves as the ground truth. Following is a sample image and its object-shadow boundary marked from dataset:

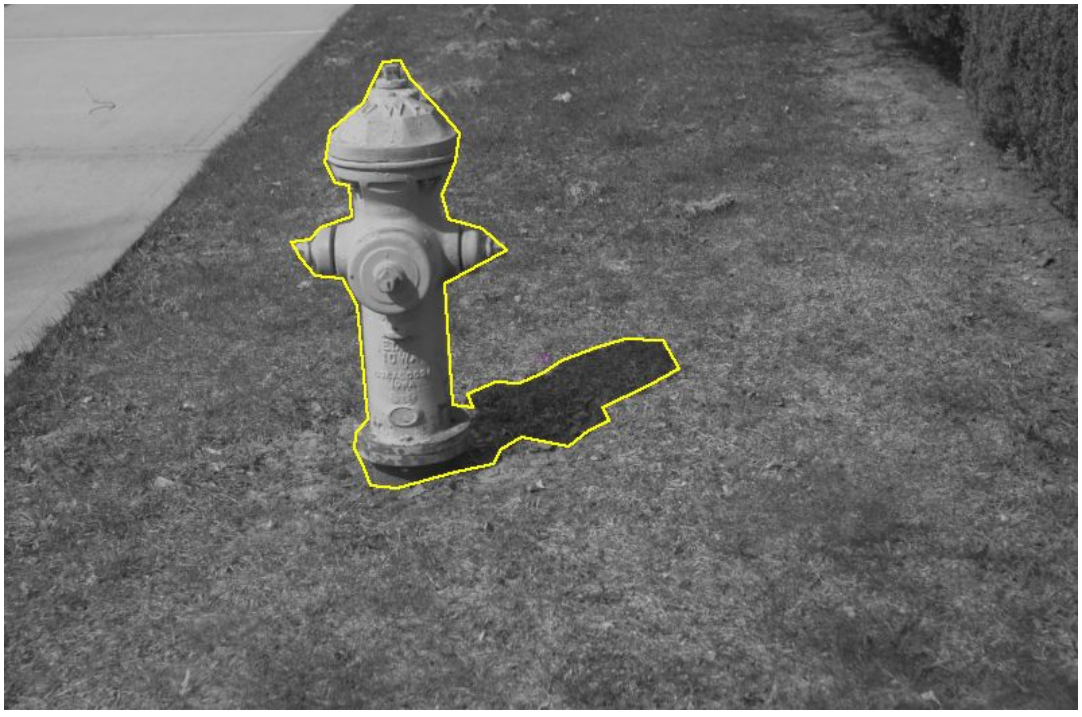


Image 5: Annotated image with object-shadow marked

2.2.3 Experiment procedure

As explained above the critical parameter for this algorithm are -

1. Threshold for region growing
2. Maximum number of peaks
3. distance between peaks

This experiment aims to find the values for these parameter so that the algorithm scales to variety of images. We divide our dataset into training and testing sets. In our training set we kept 5 images which appropriately captures the diversity of our overall dataset.

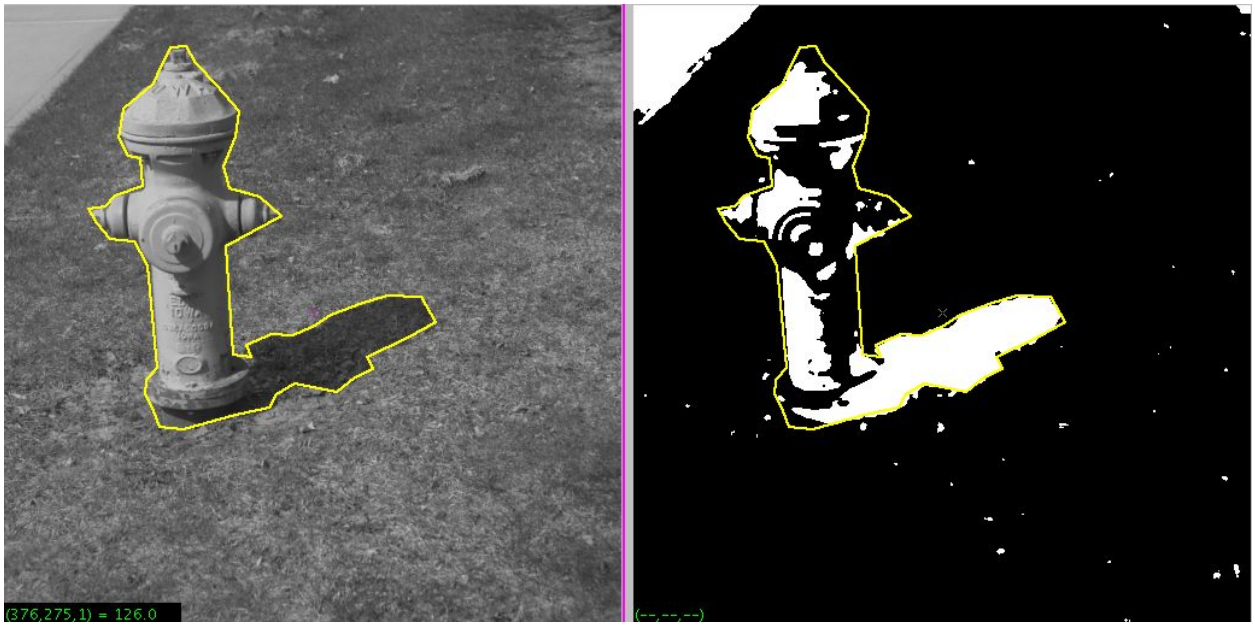
We sweep parameter over specified range and study their effect on both the number of valid peaks found and Dice coefficient value obtained between output image and annotated image. To run this experiment we used script experiment.sh(included below). This scripts sweeps over various parameter values in a loop and call required functions to output number of valid peaks and Dice coefficient.

Set of values evaluated for each parameters -

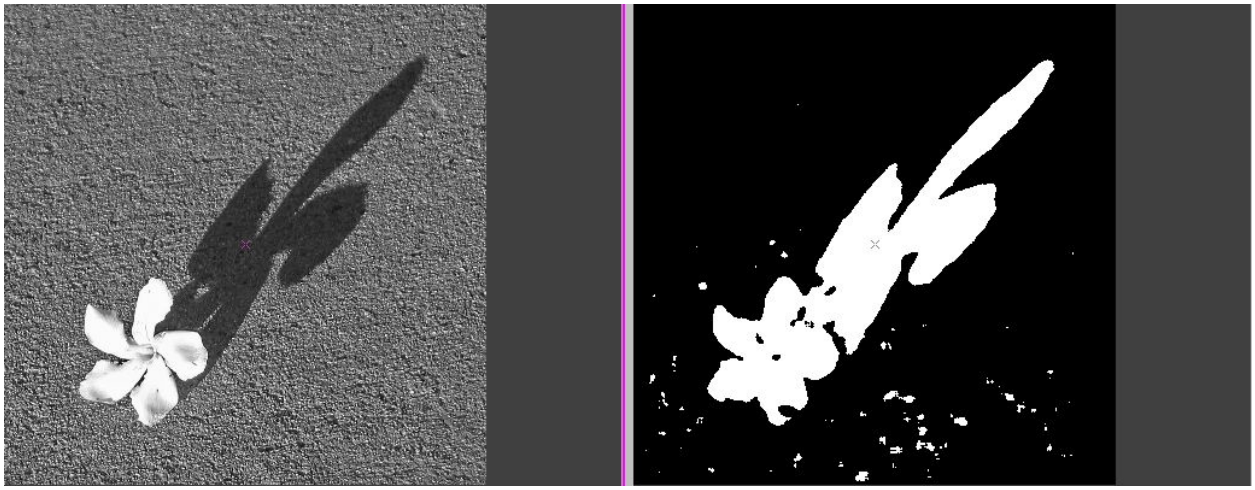
1. Threshold for region growing: Swept from 15 to 30 with step size of 5
2. Maximum number of peaks: Swept from 7 to 10 with step size of 1
3. distance between peaks: Swept from 5 to 14 with step size of 3

3. Results

In this section we show the results obtained from our object-shadow detection algorithm.
On left are the original images and on right are the output images from algorithm.

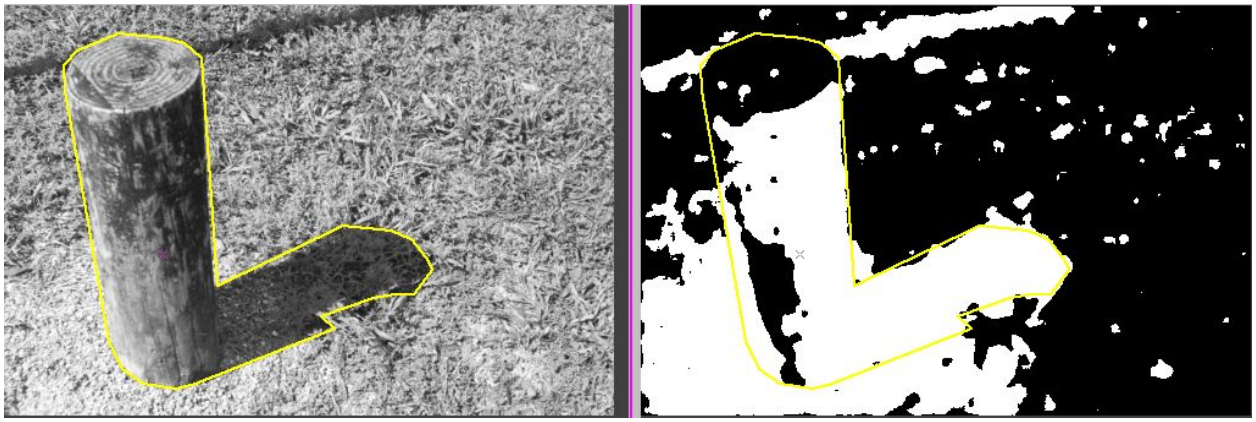


Result Image 1



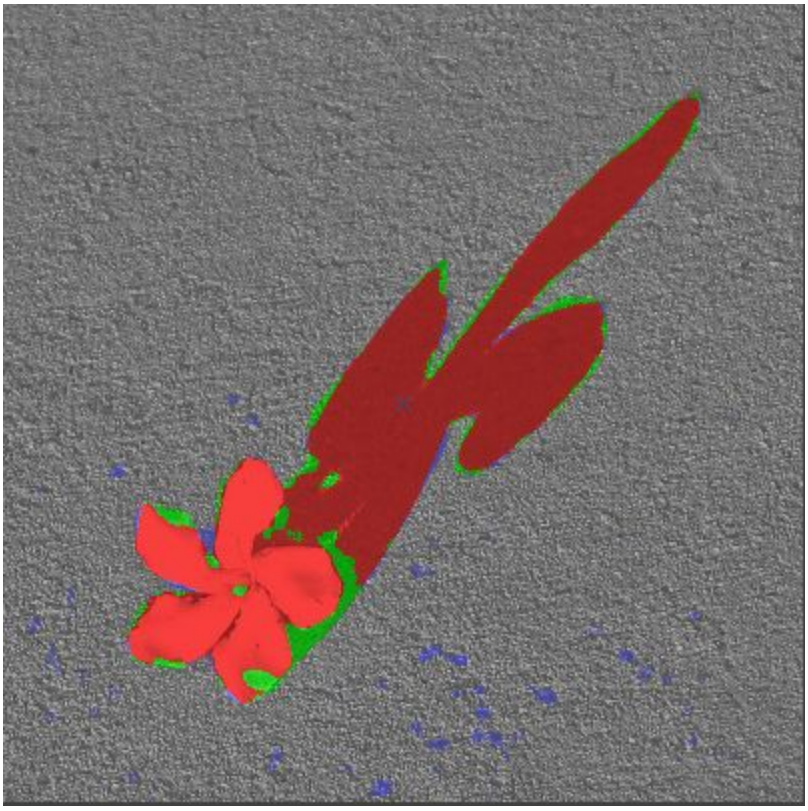
Result Image 2

Result Image 3

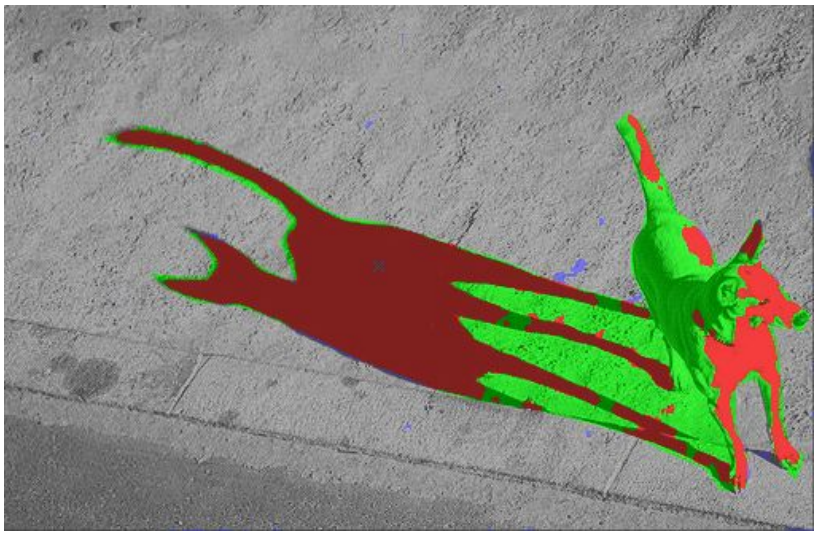


Result Image 4

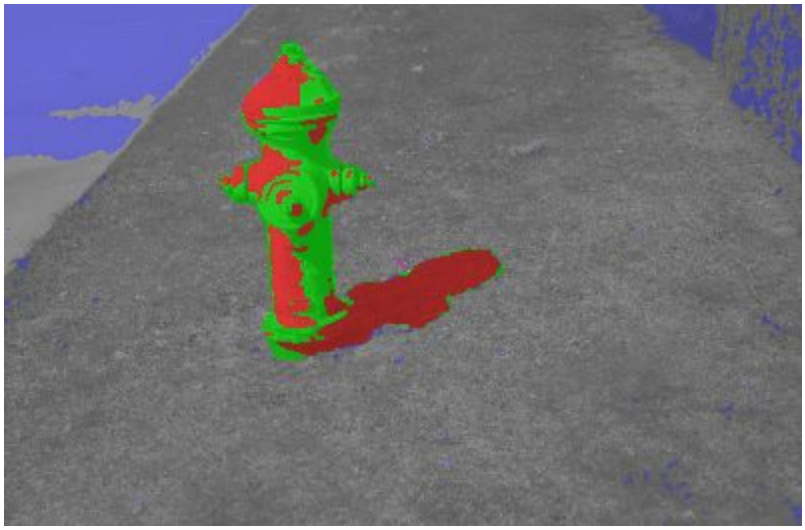
Below are the resulting image after comparison between ground truth and algorithm output using vrdiff function of VisionX.



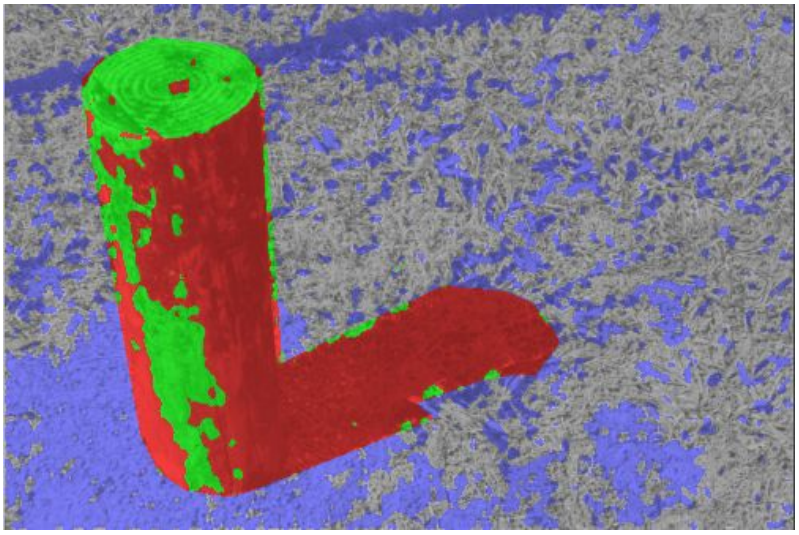
Comparison 1: Flower



Comparison 2: Dog



Comparison 3: Hydrant



Comparison 4: Wooden Post

Table 1: Evaluation Table

Image number	DSC	Sensitivity	Specificity
Comparison 1: Flower	0.85	0.98	0.96
Comparison 2: Dog	0.44	0.96	0.74
Comparison 3: Hydrant	0.29	0.32	0.88
Comparison 4: Wooden Post	0.37	0.39	0.62

Table 1 summarizes the Dice Coefficient, Sensitivity and Specificity as obtained from the comparison of annotated image and algorithm output.

3.1 Evaluation Metrics:

Given annotated ground truth A and segmented image B, we define following metrics, which are all calculated by vrdiff

Dice Similarity Coefficient(DSC): Indicates how "similar" two objects are.

$$DSC = \frac{2*|A \cap B|}{|A|+|B|}$$

Sensitivity: Indicates % of object correctly segmented.

$$Sensitivity = \frac{|A \cap B|}{|A|}$$

Specificity: Indicates % of background correctly segmented.

$$Specificity = \frac{\sim|A \cap B|}{\sim|A|}$$

4. Discussion

Out of 4 examples we have listed, the flower is segmented almost perfectly, with all percentages nearing 100%. In the case of dog, the majority of the error comes from specificity, where the background between shadow of the legs are annotated as part of object-region pair, whereas the algorithm identified it (correctly, in some sense) as background rather than continuous part of the shadow. In the case of hydrant, we have a huge section on the top left corner that has been identified as an object, due to how our thresholding works. Wooden post was the hardest to segment, with low scores all around. Main reason for this was that the pixel difference between object and background was not significant enough. Overall the experiment results were within our expectations. We expected that the performance of our algorithm might be limited in cases where there is not much pixel difference between object and background. The metrics shown above also tells a similar story.

As shown by each example chosen, the algorithm still has a lot of room for improvement. However, we were able to achieve this much using composition of two simple algorithms. This made us realize that keeping things simple is the key, directly jumping to complex algorithms is not the best strategy to solve a complex problem. Generally, combining simple algorithms together can solve major part of a problem as we demonstrated in our project.

For future work, on the application side, there can be work done to use this segmentation algorithm for its intended use, to calculate the angle of illumination. On improving the algorithm side, perhaps one can perform closing with different parameters to completely fill the gaps between the edge regions.

5. Conclusion

We have finished implementing and experimenting an object-shadow pair segmentation algorithm using multiple techniques layered together. We can see that simple threshold based region growing coupled with multiple adaptive thresholding can produce acceptable quality of segmentation that we can use for the main purpose, of calculating angle of illumination. Future and intended work include using the segmented image to calculate the location of source of illumination, perhaps using Hough transform or calculation based on region moments. In terms of segmentation itself, future work can include denoising, as currently, there are a lot of stranded pixels that does not belong in the main object that are classified as either object or shadow.

References

1. R.Q.Guo, Q.Y.Dai, D.Hoiem, “Single-image shadow detection and removal using paired regions,” IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp: 2033~2040.

2. Lalonde, Jean-François., Efros, Alexei.A., and Narasimhan, Srinivasa.G. “Detecting ground shadows in outdoor consumer photographs.” In European Conference on Computer Vision. 2010

3. Hoiem, D., Efros, A.A., Hebert, M. “Recovering surface layout from an image.” International Journal of Computer Vision 75 (2007)

4. J.-F. Lalonde, A.A. Efros and S.G. Narasimhan "Estimating Natural Illumination from a Single Outdoor Image", Proc. IEEE Int'l Conf. Computer Vision (ICCV), 2009, pp.183 -190

5. E. Salvador, A. Cavallaro, and T. Ebrahimi. “Cast shadow segmentation using invariant color features.” Computer Vision and Image Understanding, 95(2):238-259, August 2004.

6. N. Al-Najdawi, H. Bez, J. Singhai, E. Edirisinghe. “A survey of cast shadow detection algorithms.” Pattern Recognit. Lett., 33 (6) (2012), pp. 752–764

7. A. Sanin, C. Sanderson and B. C. Lovell "Shadow detection: A survey and comparative evaluation of recent methods", PR, vol. 45, no. 4, pp.1684 -1695 2012

Program Documentation

NAME

vgrow - performs region growing using fixed threshold

SYNOPSIS

vgrow if=<input_image> of=<output_image> [r=<range>] [-p]

DESCRIPTION

Performs recursive region growing by iterating over each pixel. The program iterates over each pixels, and if not labeled, calls a recursive function on the location. Depending on the -p flag, we use either the location pixel value or a unique region number and label all connected areas that fall within the range of the location pixel value, by checking each 4-connected neighbors, and if unlabeled and pixel within the range, label the region and call the recursive function again. Once all recursive function returns, all connection regions have been labeled, and we can resume iterating.

OPTIONS

r = <range>	range of pixel values allowed to be classified as same region
-p	region labeling scheme. Include to set region labels to reflect pixel values, exclude to set region labels using a monotonic identifying number

AUTHOR

Arjun Jauhari and Eric Jung

NAME

vtobjshadow - performs thresholding between two most significant histogram peaks

SYNOPSIS

vtobjshadow if=<input_image> of=<output_image> [of2=<output_image2>] [d=<min_dist>] [-s] [-v]

DESCRIPTION

Performs adaptive thresholding to find two thresholds to segment the image, one for objectand one for shadow. It can output the total paired image by -s flag, and both object and shadow image separately otherwise.

OPTIONS

d = <min_dist>	minimum distance between peaks of histogram
-v	verbose flag. Include to print out the procedure
-s	create single segmented image. Include to OR object and exclude with parameter of2 to output object and shadow image separately; of=object, of2=shadow
of2 = <output_image2>	Used optionally when -s flag is not included, to output shadow segmentation

AUTHOR

Arjun Jauhari and Eric Jung

Program Code

A. vgrow.c

```
01 /*****************************************************************/
02 /* vgrow  Compute region growing on a single byte image   */
03 /*****************************************************************/
04
05 #include "VisXV4.h"         /* VisionX structure include file    */
06 #include "Vutil.h"          /* VisionX utility header files      */
07
08 VXparam_t par[] =            /* command line structure           */
09 { /* prefix, value,   description                          */
10 {  "if=",0,    " input file  vtemp: local max filter "},
11 {  "of=",0,    " output file "},
12 {  "r=", 0,    " range value "},
13 {  "-p", 0,    " labeling scheme information "},
14 {  0,    0,    0}  /* list termination */
15 };
16
17 #define  IVAL   par[0].val
18 #define  OVAL   par[1].val
19 #define  RANGE  par[2].val
20 #define  PFLAG  par[3].val
21
22 void setlabel(int, int, int);
23 Vfstruct (im);                    /* i/o image structure         */
24 Vfstruct (tm);                    /* temp image structure        */
25 int first;
26 int range;                  /* range for region            */
27
28 main(argc, argv)
29 int argc;
30 char *argv[];
31 {
32    int          y,x;                    /* index counters              */
33    int          L = 1;                  /* index counters              */
34    VXparse(&argc, &argv, par);    /* parse the command line      */
35
36    range = 10;                    /* default dist */
37    if (RANGE) range = atoi(RANGE);  /* if d= was specified, get value */
38
39    Vfread(&im, IVAL);                /* read image file               */
40    Vfembed(&tm, &im, 1,1,1,1);    /* image structure with border  */
41    if ( im.type != VX_PBYTE ) {   /* check image format          */
42          fprintf(stderr, "vtemp: no byte image data in input file\n");
43          exit(-1);
44    }
45
46    /* Zeroing out the output file pixels im */
47    for (y = im.ylo ; y <= im.yhi ; y++) {
48          for (x = im.xlo; x <= im.xhi; x++)  {
```

```
49          im.u[y][x] = 0;
50          }
51    }
52
53    for (y = im.ylo ; y <= im.yhi ; y++) {
54        for (x = im.xlo; x <= im.xhi; x++)  {
55          if (tm.u[y][x] != 0) {
56                if (im.u[y][x] == 0) {
57                      first = tm.u[y][x];
58                      if (PFLAG) {
59                      setlabel(x,y,first);
60                    }
61                      else {
62                      setlabel(x,y,L);
63                      L = (L % 255) + 1;
64                      }
65                }
66          }
67          }
68    }
69    Vfwrite(&im, OVAL);            /* write image file                    */
70    exit(0);
71 }
72
73 /* function to compute the local maximum */
74 void setlabel(int x, int y, int L)
75 {
76    im.u[y][x] = L;
77    if (tm.u[y][x+1] != 0 && im.u[y][x+1] == 0 && abs(tm.u[y][x+1] -
                                                          first) <= range)
78          setlabel(x+1,y,L);
79    if (tm.u[y][x-1] != 0 && im.u[y][x-1] == 0 && abs(tm.u[y][x-1] -
                                                          first) <= range)
80          setlabel(x-1,y,L);
81    if (tm.u[y+1][x] != 0 && im.u[y+1][x] == 0 && abs(tm.u[y+1][x] -
                                                          first) <= range)
82          setlabel(x,y+1,L);
83    if (tm.u[y-1][x] != 0 && im.u[y-1][x] == 0 && abs(tm.u[y-1][x] -
                                                          first) <= range)
84          setlabel(x,y-1,L);
85 }
```

B. vtobjshadow.c

```
/*****************************************************************/
/* vtpeak:    Threshold image between two most sig. hgram peaks      */
/*****************************************************************/

#include "VisXV4.h"         /* VisionX structure include file    */
#include "Vutil.h"          /* VisionX utility header files      */

VXparam_t par[] =           /* command line structure            */
{
{  "if=",   0,   " input file, vtpeak: threshold between hgram peaks"},
{  "of=",   0,   " output file "},
{  "of2=",  0,   " output file "},
{  "d=",0,   " min mindist between hgram peaks (default 10)"},
{  "-v",0,   "(verbose) print threshold information"},
{  "-s",0,   " creates a single obj-shadow segemented image"},
{   0,  0,   0} /* list termination */
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  OVAL2  par[2].val
#define  DVAL   par[3].val
#define  VFLAG  par[4].val
#define  SFLAG  par[5].val

main(argc, argv)
int argc;
char *argv[];
{

  Vfstruct (im);                /* input image structure       */
  Vfstruct (im2);                /* input image structure            */
  int y,x;                      /* index counters              */
  int i,j;

  int hist[256];                /* histogram bins              */
  int thresh;                   /* threshold                   */
  int maxbin;                   /* maximum histogram bin       */
  int nxtbin;                   /* second maximum bin          */
  int minbin;                   /* minumim histogram bin       */
  int maxa, maxb;               /* second max bin above/below maxbin   */
  int npixels=0;               /* total number of pixels */
  int normthresh;
  int niter;
  int npeaks;
  int peaks[256];            /* peaks array             */
  int flag;
  int rerun;
```

```
048   int dist;
049   int mindist;
050   int shadowth, objth;
051
052
053   VXparse(&argc, &argv, par);    /* parse the command line      */
054
055   mindist = 7;                   /* default mindist */
056   if (DVAL) mindist = atoi(DVAL);  /* if d= was specified, get value */
057   if (mindist < 0 || mindist > 255) {
058   fprintf(stderr, "d= must be between 0 and 255\nUsing d=10\n");
059     mindist = 7;
060   }
061
062   while ( Vfread( &im, IVAL) ) {
063         if ( im.type != VX_PBYTE ) {
064               fprintf (stderr, "error: image not byte type\n");
065               exit (1);
066         }
067
068         /* make a copy */
069         Vfread( &im2, IVAL);
070
071         /* clear the histogram */
072         for (i = 0; i < 256; i++) hist[i] = 0;
073
074         /* compute the histogram */
075         for (y = im.ylo; y <= im.yhi; y++) {
076         for (x = im.xlo; x <= im.xhi; x++) {
077                     hist[im.u[y][x]]++;
078                     npixels++;
079         }
080         }
081
082         /* compute the threshold */
083         normthresh = floor(0.001*npixels);   // 0.1% of total pixels
084         flag = 1;
085         rerun = 0;
086         niter = 0;
087         while (flag) {
088         niter++;
089         /* Find peaks above threshold */
090         npeaks = 0;
091         for (i = 0; i <= 255; i++) {
092                     if (hist[i] > normthresh) {
093                           peaks[npeaks] = i;
094                           npeaks++;
095                     }
```

```
096         }
097
098         if (npeaks < 3) {
099               /* reduce thresh and continue */
100                     flag = 1;
101                     normthresh /= 2;   // reduce 50%
102                     continue;
103         } else if (npeaks > 8) {
104               /* increase thresh and continue */
105                     flag = 1;
106                     normthresh *= 1.1;
107                     continue;
108         }
109
110         for (j = 1; j<npeaks; j++) {
111                     dist = peaks[j] - peaks[j-1];
112                     if (dist<mindist) {
113                     /* inc threshold and continue */
114                           flag = 1;
115                           normthresh *= 1.1;  // increase 10%
116                           rerun = 1;
117                           break;
118                     }
119         }
120
121         if (rerun) {
122                     rerun = 0;
123                     continue;
124         }
125
126         flag = 0;
127         }
128
129         if(VFLAG)
130               fprintf(stderr, "niter = %d normth = %d npeaks = %d\n",
131                                 niter, normthresh, npeaks);
132
133         /* Pick leftmost(shadow) rightmost(object) peaks */
134         shadowth = floor((peaks[0] + peaks[1])/2);
135         objth = floor((peaks[npeaks-1] + peaks[npeaks-2])/2);
136
137         /* Adding Non-linearity: dont want too low or too high thresh */
138         if (shadowth < 15)
139               shadowth += 10;
140         if (objth > 240)
141               objth -= 10;
142
143         if(VFLAG)
```

```
144                fprintf(stderr, "shadowth = %d objth = %d\n",
145                                shadowth, objth);
146
147          /* apply the threshold(shadow) */
148          for (y = im.ylo; y <= im.yhi; y++) {
149          for (x = im.xlo; x <= im.xhi; x++) {
150                if (im.u[y][x] >= shadowth) im.u[y][x] = 0;
151                else                        im.u[y][x] = 255;
152          }
153          }
154
155          /* apply the threshold(obj) */
156          for (y = im2.ylo; y <= im2.yhi; y++) {
157              for (x = im2.xlo; x <= im2.xhi; x++) {
158                if (im2.u[y][x] >= objth) im2.u[y][x] = 255;
159                else                      im2.u[y][x] = 0;
160          }
161          }
162
163          if (SFLAG) {
164          /* ORing the two images */
165          for (y = im.ylo; y <= im.yhi; y++) {
166                      for (x = im.xlo; x <= im.xhi; x++) {
167                            if (im.u[y][x] || im2.u[y][x]) {
168                            im.u[y][x] = 255;
169                      }
170                      }
171          }
172          }
173
174          Vfwrite( &im,  OVAL);
175          if (!SFLAG)
176          Vfwrite( &im2, OVAL2);
177     } /* end of every frame section */
178     exit(0);
179 }
```