Objective

The objective of this lab is to gain experience with image segmentation through both thresholding and region growing techniques and to become familiar with the programming tools for both global and point operations.

Overview

In this lab you will explore the use of several automatic thresholding techniques as well as region growing for segmenting a set of test images. A list of requirements for the lab report is given in the checkout section. The lab report is due electronically via Blackboard one week after your lab section.

1. Lab Preparation

   1. Make a working directory for this Lab

      mkdir lab3
      cd lab3

   2. Copy (link) to the files needed for this lab

      /classes/ece547/lab3.setup
      ls
      vl

2. Peakiness Detection (vtpeak)

The command vtpeak implements an automatic thresholding algorithm that detects the two main peaks in the histogram and selects the threshold from the minimum histogram value between these peaks. For this section of the lab explore its performance on a number of images. Do the following:

   1. Compile vtpeak.c using vcc.
   2. Test vtpeak on the image mp.vx

      (a) Display the image using vview and then display the image histogram using Tools → Image Histogram. Alternatively, you may display the histogram from the command line using

         vplot -h mp.vx
(b) Compute and display a thresholded version of the image using vtpeak

    ./vtpeak d=10 -v mp.vx of=mpp.vx

(c) Explore the effect, if any, of the d= parameter of vtpeak. For convenience, to accomplish this, you can reuse the above command with the same output file name and then just double click on the file name in vview to display the new image.

3. Repeat the above procedure for the other test images: facsimile, map, shtl.vx

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>vplot -h</td>
<td>compute and display the histogram of an image using the gnu plotting package</td>
</tr>
<tr>
<td>vtpeak [-v][d=]</td>
<td>automatically threshold an image</td>
</tr>
<tr>
<td></td>
<td>the -v (verbose) option prints the selected threshold</td>
</tr>
</tbody>
</table>

3. Iterative Threshold Selection (vits)

Write a program called vits.c that implements an algorithm similar to the Iterative Threshold Selection Algorithm 6.2 in Sonka et.al. as outlined below.

1. Start with the template program vtpeak.c (only modify the part that processes the data)

    cp vtpeak.c vits.c
    gedit (or your favorite text editor) vits.c
    vcc vits.c -o vits

2. Debug your program with small test images THEN test with larger images.

   Note the following in developing your program:

   1. Get an initial threshold thresh from the command line if specified
      or
      Compute the average gray value of image and use it as an initial threshold thresh

   2. Consider the two image regions R1 and R2 that are determined by thresh. (i.e. pixels above thresh and pixels below thresh) Compute the average gray values (avg1, avg2) for R1 and R2. Note that the number of pixels in R1 or R2 may be zero.

   3. Compute a new threshold using:

       thresh = (avg1 + avg2)/2

   4. Repeat 2-4 until avg1 and avg2 do not change between successive iterations

   5. Apply the threshold to the image (values above thresh set to 255 otherwise 0)

      Note that you will have to save copies of avg1 and avg2 (old1, old2) between iterations to see if they have changed.
      Also, to prevent infinite loops during development, you may want to limit the number of iterations using a loop counter.
4. Adaptive Thresholding

In adaptive thresholding an image may be decomposed into a number of patches for which a separate threshold is computed.

The command `vpatch` decomposes an image into a sequence of overlapping patches. `vquilt` with appropriate parameters will reassemble the patches back into an image. For example, consider the command:

```
vpatch p=64 l=0 mp.vx | vits | vquilt -p of=out.vx
```

This will decompose the `mp.vx` image into non-overlapping patches of size 64x64, threshold each patch, and reconstruct the image by quilting the patches together.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vpatch</code></td>
<td>Decompose and image into a set of overlapping patches</td>
</tr>
<tr>
<td><code>[p=]</code></td>
<td></td>
</tr>
<tr>
<td><code>[l=]</code></td>
<td></td>
</tr>
<tr>
<td><code>vquilt</code></td>
<td>Construct an image from patches</td>
</tr>
<tr>
<td><code>-p</code></td>
<td></td>
</tr>
<tr>
<td><code>of=out.vx</code></td>
<td>the p= parameter specifies that the images were created using vpatch</td>
</tr>
</tbody>
</table>

For this section, explore different patch sizes and overlap amounts for the `map` and `mp.vx` test images with the `vtpeak` program.

Parameter Optimization

When developing a computer vision algorithm there are typically a number of parameters that need to be optimized to achieve the best algorithm performance. Optimal parameter settings may be determined by thoroughly evaluating the algorithm on a representative set of training images with a large selection of different parameter settings. A script that will execute an algorithm with a set of different parameters is very useful in this situation. `patcheval` is an example script to evaluate the vtpeak algorithm with different adaptive thresholding settings (patch and overlap values) for vpatch. Review the patcheval script in `vview` by double clicking on the file name. The set of parameter settings to be explored are set in the lines following the "looping" comments and may be changed by editing. There will be more about writing scripts in Lab 4.

Evaluate the adaptive vtpeak method for the parameters patch size and overlap for the `map` image by running the script patcheval on the map image. Identify the best parameters for this image by reviewing the images in the file `map.sq`.

5. Region Growing (vgrow)

To introduce the concept of region growing we will develop a simple region growing algorithm based on the recursive connected component labeling algorithm developed in lab 2 called `cclabel`.

Recall that in `cclabel` we used the recursive procedure `setLabel` to identify all the connected object pixels and set the label for that object. We had two criteria for calling `setLabel` on a pixel:

1. the pixel had not yet been labeled
2. the pixel was part of the object (i.e. non-zero)
Note that the algorithm worked with the above criteria because the imagery we used contained neatly segmented objects (i.e. distinct non-zero object pixels and zero valued background pixels). (Consider what would happen if we applied cclabel to a gray-scale, non-segmented image that contained no zero pixel values.)

If we change the second criteria above from simply checking for non-zero pixels to one that checks if the pixel is within a range of values, we can use the algorithm to identify all the connected pixels that are within a given range of values. Optionally, instead of sequentially numbering each region, we can use a pixel value that is representative of the region, (e.g. the value of the first pixel in the region), as the output “label”. With the above changes we can effectively “grow regions” of similar pixels.

Start by copying cclabel.c to vgrow.c. The suggested algorithm for the vgrow program is as follows:

1. Add a new input parameter, ",r=" , to vgrow. This is an integer value that will set the region pixel range. Assign this input value, (or default), to a global variable range.

2. Add a new flag, ",-p ", to vgrow to select the labeling scheme. When this flag is set, the label value will be the value of the first pixel in the region. Otherwise the regions are numbered sequentially.

3. Initialize im and tm as in cclabel. Note that tm and im should be global variables.

4. Scan the input image, tm, if the image pixel is not labeled (i.e. the output image, im, is zero for this pixel), then
   (a) set the global variable first to the pixel value.
   (b) if the ",-p " flag is set, then call setlabel(j, i, first) else call setlabel with the next sequential label number. Note that you must go back to label number 1 after using label number 255.

5. Repeat step 4 for all unlabeled pixels.

The suggested procedure for recursive function setlabel(x, y, l) is

1. Set the output image pixel at (x, y) to l.

2. For each of the four neighbors of (x, y), check the following three criteria in the order given:
   (a) if the image pixel, tm, is non-zero
      (since we only have zero pixels at the image border, this check prevents us from leaving the bounds of im)
   (b) and the pixel is unlabeled
   (c) and the pixel is in range
      (in range means that the difference between this pixel and the first pixel for this region is less than range.)
      then call setlabel for this neighbor.

For this section, you will experiment with different range values for region growing on several images. Some things to remember:
1. Pixels with a value of zero are treated specially as not belonging to any region. When you create small test images, in general, they should not contain any zeros since such pixels will not be labeled.

2. If you have used sequential labeling then you may need to window to see the different regions. In vdview the “auto” window setting scales the window to the range of available pixel values and will make the regions visible.

3. It is sometimes easier to visualize the segments of the image using “false” coloring; that is to map a random color to each grey level in the image (providing sharp color contrast between adjacent grey level values). A color index image with a “random” look-up-table may be created with the command “vcmap If=rand”. Try this out on a region image with sequential color labels.

4. If there are more than 255 regions, the sequential label numbers will “roll over” and repeat themselves. (this is where the "-p" flag may be useful.)

5. You may run out of stack space if you attempt to vgrow very large regions within a large full size image (like mp.vx). In this case you must ”tighten” the range to produce smaller regions if possible.

Now do the following:

1. execute vgrow -p on nb.vx and determine the range value that results in the “best” segmentation. Repeat for shtl.vx

2. execute the sobel edge operator on the shtl.vx image to produce an edge detected image

   vsobel shtl.vx of=edge.vx

   Now execute vgrow (without "-p") on edge.vx and determine the range value that results in the best segmentation. Use vpix to make the image printable.

Checkout

To complete this lab see the lab instructor who will go through several parts of the lab with you then you must write up a lab report which will be due in one week; i.e. you should turn it in via Blackboard before the START of your next lab section. It is not expected that you should complete all parts of this lab by the end of the lab section; you are expected to complete and test the programs in your own time. Your Lab report should contain at least the following:

1. A cover page giving the report title and your name (you can use either your name or your course id to identify yourself). period).

2. Your observations on each of the images tested in section 2 that indicate strengths and weaknesses of the Peakiness Algorithm.

3. A written description of how your Iterative Threshold program, vits, developed in section 3, works.

4. The listing of your program vits.c.
5. A copy of the typescript that shows your \texttt{vits} program working for a small test image. This typescript must show both the input image and the output image.

6. A page with image figures and comments that shows that your program works on a full size image (show several result images).

7. A brief description of how the process of Adaptive Thresholding works.

8. Your observations on the results of the Adaptive Thresholding algorithms on \texttt{map} and \texttt{mp.vx} as described in section 4.

9. A written description of how your \textit{Region Growing} program, \texttt{vgrow}, developed in section 5, works.

10. The listing of your program \texttt{vgrow.c}.

11. A copy of the typescript that shows that your program \texttt{vgrow} works for a small test image.

12. A page with image figures and comments that shows that your program \texttt{vgrow} works on the images \texttt{nb.vx} and \texttt{shtl.vx}. Indicate the \textit{range} value used for each.

13. A page with image figures and comments showing the segmented \texttt{shtl} image after executing \texttt{vgrow} on the edge detected image. Describe, in general terms, how this segmented image was generated.
Template Program

The program vlmax.c is a template program that should be used for future image processing programs. It is similar to vtemp.c except that it also handles files containing a number of different images.

```c
#ifndef __VLMAX_H__
#define __VLMAX_H__

#define VX_MAX_IMAGES 100

/* vlmax: Compute local max operation on byte images */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    int i, j;

    /* input image structure */
    int im;

    /* temp image structure */
    VisXimage_t tm;

    /* parse the command line */
    VXparse(argc, argv, par);

    /* open input file */
    VXin = VXopen(IVAL, 0);

    /* open the output file */
    VXout = VXopen(OVAL, 1);

    while((VXlist = VXptr = VXreadframe(VXin)) != VXNIL) /* every frame */
    {
        VXfupdate(VXout, VXin); /* update global constants */
        /* find next byte image */
        while (VXNIL != (VXptr = VXfind(VXptr, VX_PBYTE)))
        {
            VXsetimage(&im, VXptr, VXin); /* initialize input structure */

            /* temp image copy with border */
            VXembedimage(&tm, &im, 1,1,1,1);
            for (i = im.ylo; i <= im.yhi; i++) /* compute the function */
            {
                ...
            }
        }
    }

    return 0;
}
```

---

Note: The code provided is a template program for computing local max operations on byte images. It includes the necessary headers, parses command-line arguments, opens input and output files, reads frames from the input file, updates global constants, finds byte images, initializes input structures, copies images with borders, and computes the function for each image. The function implementation is not shown here and would typically involve image processing operations specific to the task at hand.
for (j = im.xlo; j <= im.xhi; j++)
{
    im.u[i][j] = MAX(tm.u[i][j],
                  MAX(tm.u[i][j+1],
                      MAX(tm.u[i+1][j],
                          MAX(tm.u[i][j-1], tm.u[i-1][j]))));
}

VXresetimage(&tm); /* free the tm image structure */

/**************** End of the Application specific section **************/

VXresetimage(&im); /* free the im image structure */
VXptr = VXptr->next; /* move to the next image */
} /* end of every image section */
VXwriteframe(VXout,VXlist); /* write frame */
VXdellist(VXlist); /* delete the frame */
} /* end of every frame section */
VXclose(VXin); /* close files */
VXclose(VXout);
exit(0);