Lab3

SEGMENTATION, THRESHOLDING AND REGION GROWING

Xinyi (Blanca) Bai

Cornell University
Electrical and Computer Engineering

Table of Contents

1. Introduction

In this lab I gain experience with the image segmentation through
both thresholding and region growing techniques. And lab 3 helps us become familiar with the programming tools for both global and point operations. In section 2, I exercise on peakiness detection algorithm. In Section 3, Iterative threshold selection algorithm is implemented. In section 4, I exercise on adaptive thresholding algorithm and explore for different patch sizes and overlap amounts for testes images, and compare them with vtpeak program. In section 5, region growing algorithm is implemented, the all descriptions and source codes are in this report.

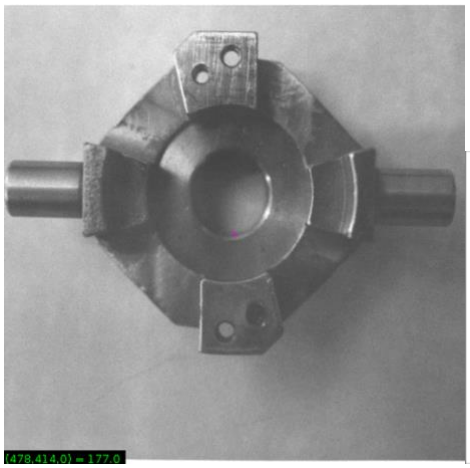
2. Peakiness Detection (vtpeak)

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.

2.1 Observations of testes images

2.1.1mp.vx

```
1. vcc vtpeak.c - o vtpeak
2. vtpeak mp.vx of = vtpeakmp.vx
```



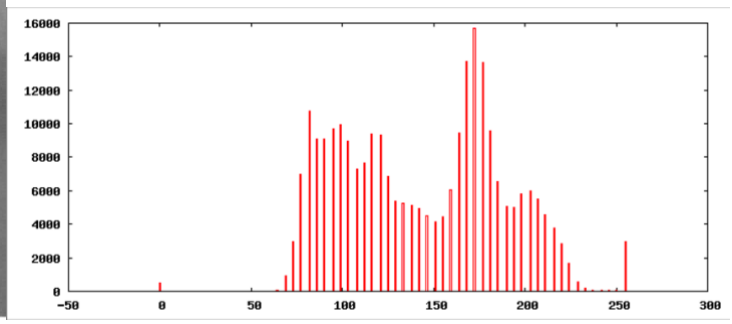


Figure 1: input mp.vx(left) Figure 2: Histogram of input mp.vx(right)

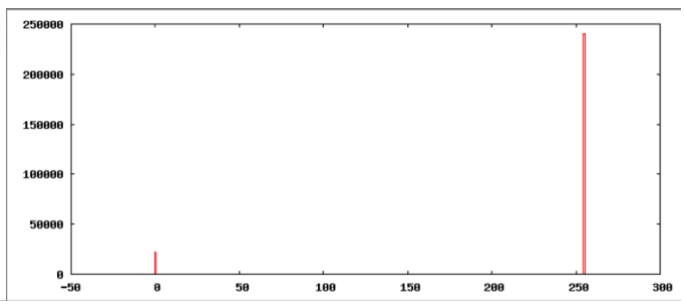


Figure 3: output vtpeakmp.vx(left) Figure 4: Histogram of output vtpeakmp.vx (right)

```
1. vtpeak d = 10 - v mp.vx of = mpp.vx
2. maxbin = 172 nxtbin = 82 thresh = 83
```

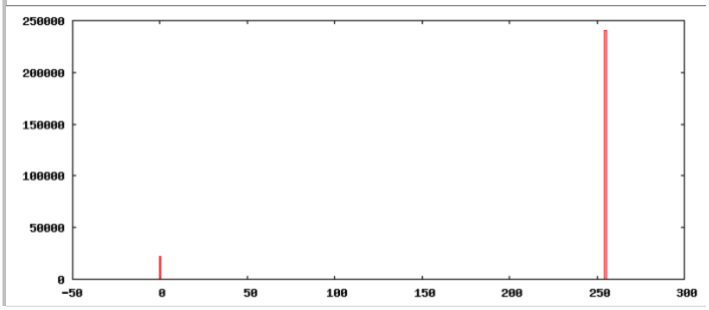


Figure 5: output mpp.vx(left) Figure 6: Histogram of output mpp.vx (right)

The threshold is 83, d = 10.

2.1.2 facsimile

```
1. vtpeak d = 10 - v facsimile of = vtpeakd10facsimile.vx
2. maxbin = 240 nxtbin = 0 thresh = 1
```

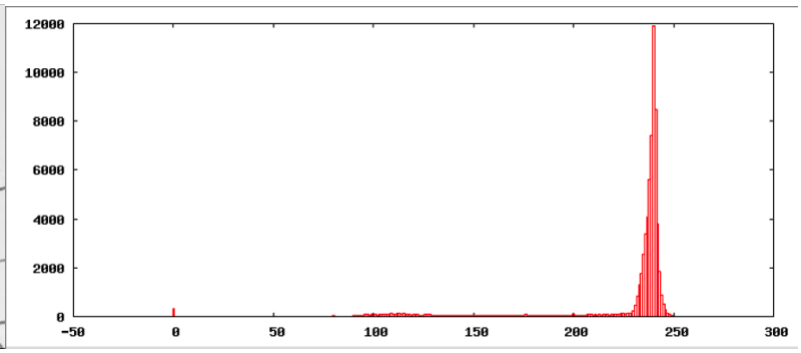


Figure 7: input facsimile(left) Figure 8: Histogram of input facsimile (right)

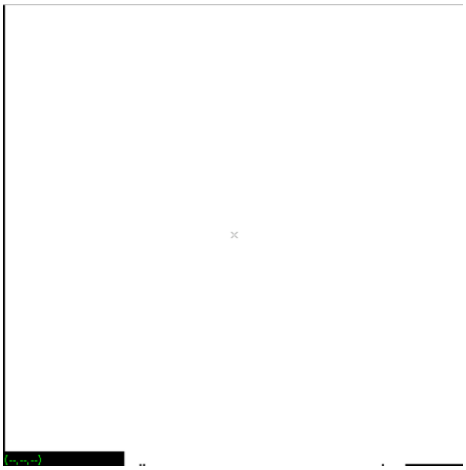


Figure 9: output vtpeak10facsimile.vx

The threshold is 1, d = 10. So the result is almost a blank, because of the error threshold.

So in order to get a better result, I change the d to 5.

```
1. vtpeak d = 5 - v facsimile of = vtpeakd5facsimile.vx
2. maxbin = 240 nxtbin = 235 thresh = 235
```

```
view vtpeakd5facsimile.vx
```

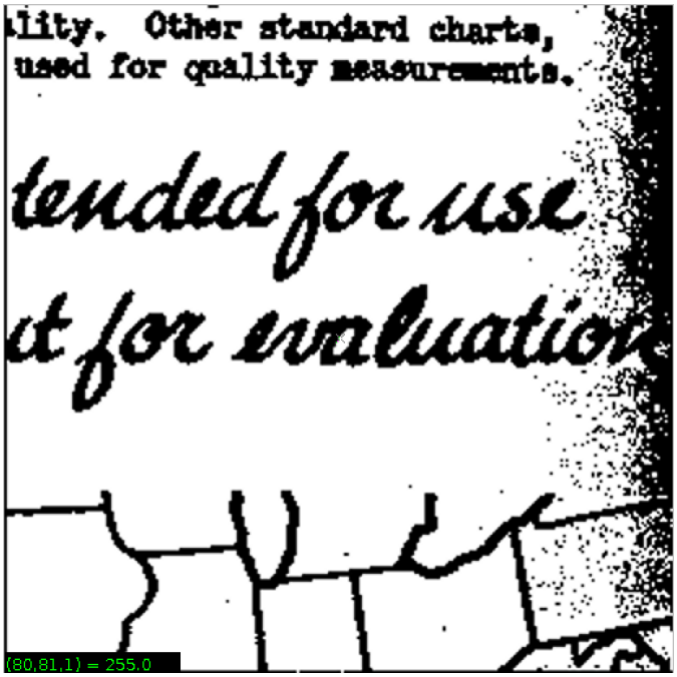


Figure 10: output vtpeak5facsimile.vx

The threshold is 235, d = 5. So the result is better than d = 10.

2.1.3 map.vx

```
1. vtpeak d = 10 - v map of = vtpeakd10map.vx
2. maxbin = 127 nxtbin = 115 thresh = 122
```

Figure 11: input map

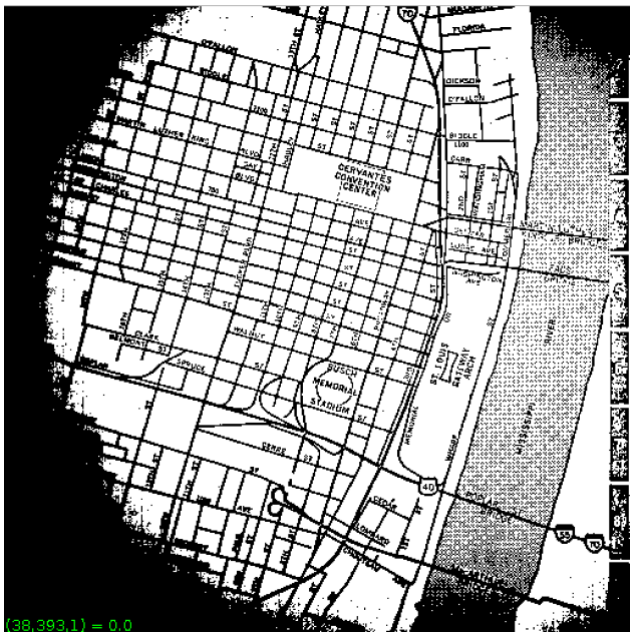


Figure 12: output vtpeakd10map.vx

The threshold is 122, d = 10.

2.1.4 shtl.vx

```
1. vtpeak d = 10 - v shtl.vx of = vtpeakd10shtl.vx
2. maxbin = 148 nxtbin = 158 thresh = 158
```

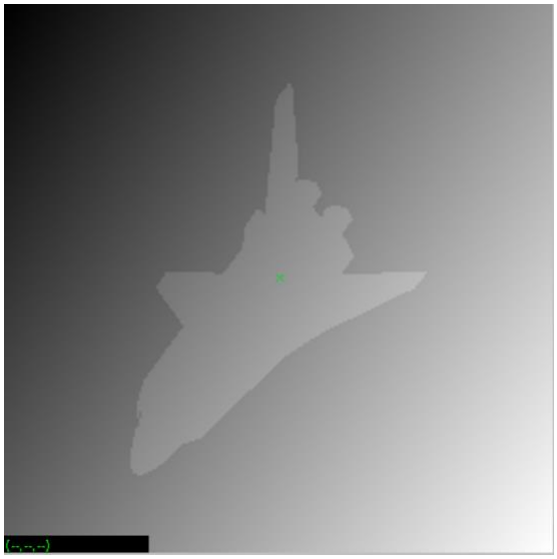


Figure 13: input shtl.vx

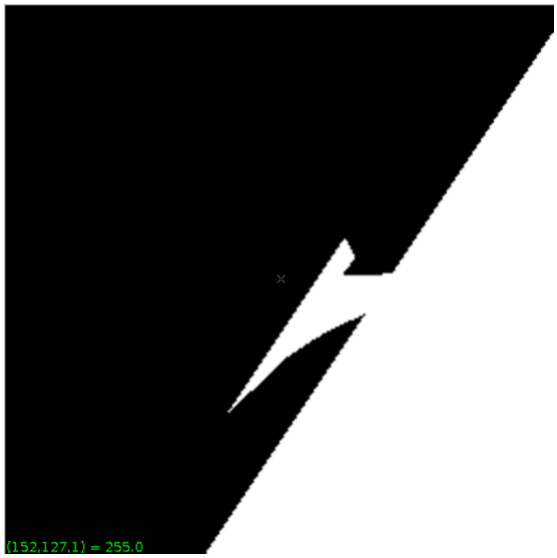


Figure 14: output vtpeakd10shtl.vx

The threshold is 158, d = 10.

```
vtpeak d = 1 - v shtl.vx of = vtpeakd1shtl.vx
maxbin = 148 nxtbin = 149 thresh = 148
```

```
view vtpeakd1shtl.vx
```

Figure 15: output vtpeakd1shtl.vx

The threshold is 148, d = 1.

For shtl.vx, although the d is different, we find the threshold is almost the same, the result is not good, because of the gradient background.

2.2 Strength and weakness of vtpeak

Strength: vtpeak is detrmined by a small amount of pixels, so it is simple to implement and fast.

Weakness: it doesn't work on gradient background image, such as shtl.vx. And vtpeak is not robust for noise, because noise is the highest peak. Vtpeak will be more robust by smoothing the histogram.

3. Iterative Threshold Selection (vits)

3.1 Implementation

1) Compute the average gray value of image (avg0)and use it as an initial threshold thresh
2) Consider the two image regions Above and Below, compute the average gray values (avgabove, avgbelow) for Above and Below. If the number of pixels in Above or Below is zero, then set the avgabove or avgbelow be 0.
3) Compute a new threshold using: thresh = (avgabove + avgbelow)/2
4) Repeat 2-3 steps until avgabove and avgbelow do not change between successive iterations
5) Apply the thresh to the image, set values above thresh as 255, below thresh as 0.

3.2 Source code

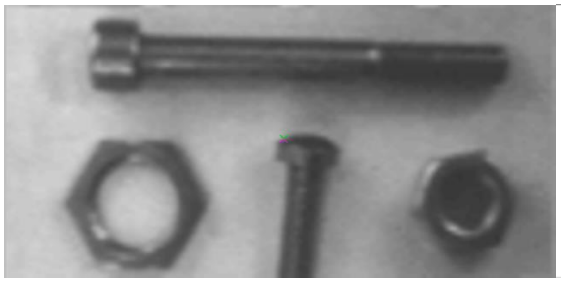
```
1.  //Compute the average gray value of image  //and use it as an initial threshold thresh
2.  int sum = 0;
3.  int pixnum = 0;
4.  for (y = im.ylo; y <= im.yhi; y++) {
5.      for (x = im.xlo; x <= im.xhi; x++) {
6.          sum = sum + im.u[y][x];
7.          pixnum++;
8.      }
9.  }
10. int avg0 = sum / pixnum;
11. int thresh = avg0;
12. int sumabove = 0;
13. int sumbelow = 0;
14. int pixnumabove = 0;
15. int pixnumbelow = 0;
16. int avgabove = 0;
17. int avgbelow = 0;
18. int oldavgabove = 1;
19. int oldavgbelow = 0;
20. while (avgabove != oldavgabove && avgbelow != oldavgbelow) {
21.     oldavgabove = avgabove;
22.     oldavgbelow = avgbelow;
23.     for (y = im.ylo; y <= im.yhi; y++) {
24.         for (x = im.xlo; x <= im.xhi; x++) {
25.             if (im.u[y][x] > thresh) {
26.                 sumabove = sumabove + im.u[y][x];
27.                 pixnumabove++;
28.             }
29.             if (im.u[y][x] < thresh) {
30.                 sumbelow = sumbelow + im.u[y][x];
31.                 pixnumbelow++;
32.             }
33.         }
34.     }
35.     if (pixnumabove == 0) avgabove = 0;
36.     if (pixnumabove == 0) avgbelow = 0;
37.     else {
38.         avgabove = sumabove / pixnumabove;
39.         avgbelow = sumbelow / pixnumbelow;
40.     }
41. }
42. thresh = (avgabove + avgbelow) / 2; /* apply the threshold */
43. for (y = im.ylo; y <= im.yhi; y++) {
44.     for (x = im.xlo; x <= im.xhi; x++) {
```

```
45.         if (im.u[y][x] >= thresh) im.u[y][x] = 255;
46.         else im.u[y][x] = 0;
47.     }
48. }
```

3.3 Results on small test image

3.3.1 nb.vx

```
1. vcc vits.c - o vits
2. vits nb.vx of = vitsnb.vx
```



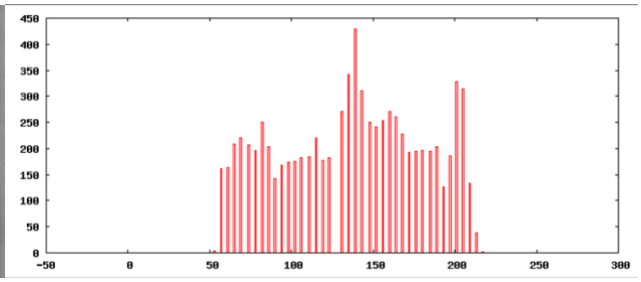


Figure 16: input nb.vx(left) Figure 17: Histogram of input nb.vx(right)

Figure 18: output vitsnb.vx

3.4 Results on full-size test images

3.4.1 mp.vx

```
1. vits mp.vx of = vitsmp.vx
2. vpix - neg mp.vx | vcapt c = "vits input mp.vx" of = vitsinputcaptionmp.vx
3. vxport - png vitsinputcaptionmp.vx
```

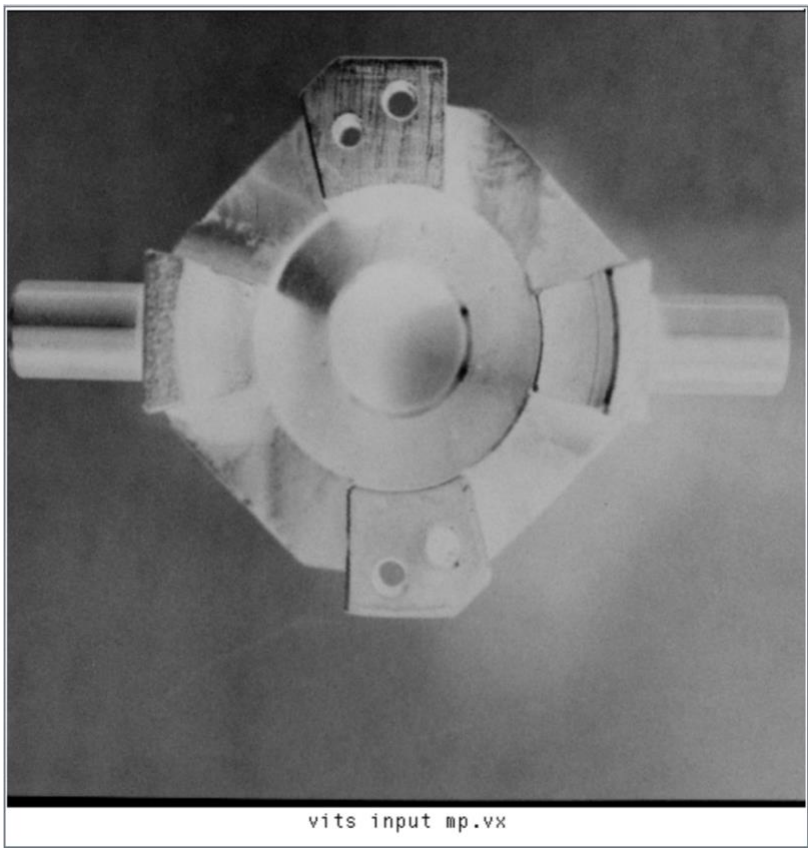


Figure 19: input mp.vx

```
vpix - neg vitsmp.vx | vcapt c = "vits output mp" of = vitsoutputcaptionmp.vx
vxport - png vitsoutputcaptionmp.vx
```

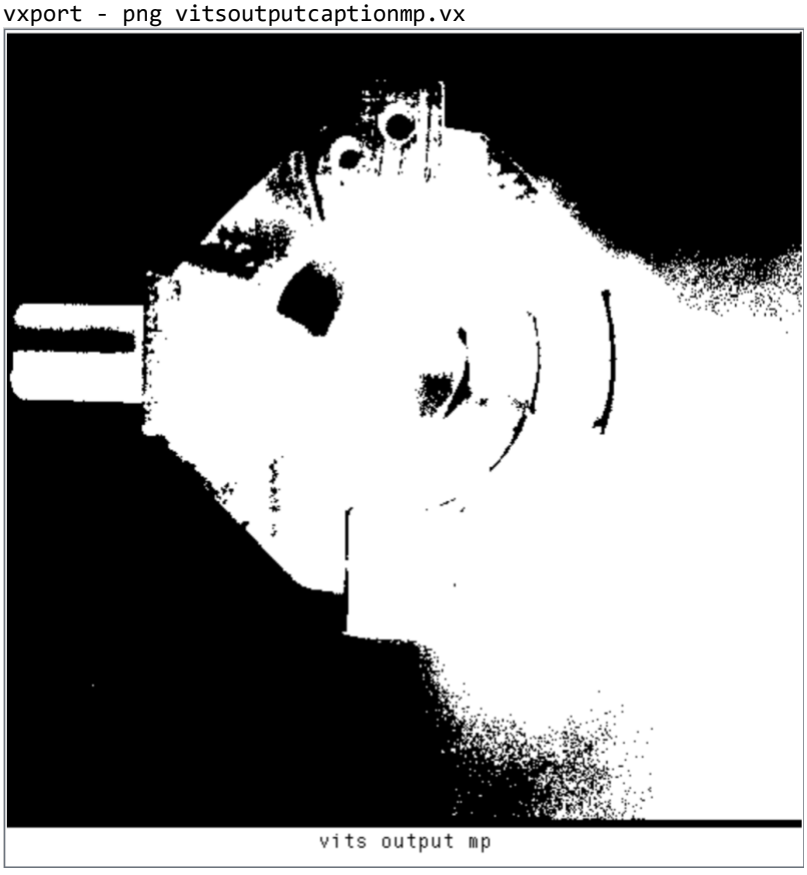


Figure 20: output vitsmp.vx

3.4.2 map

```
1. vits map of = vitsmap.vx
```

Figure 21: input map.vx

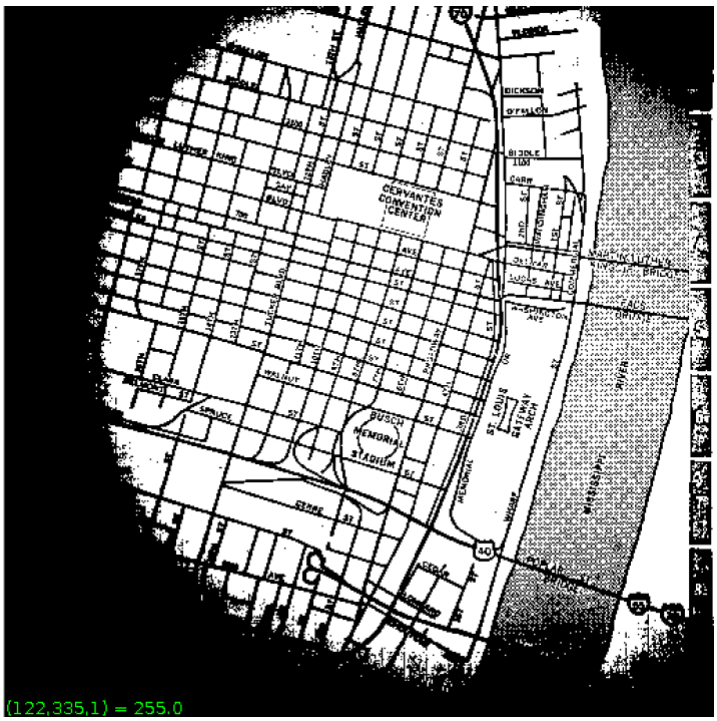


Figure 22: output vitsmap.vx

3.5 Analysis on vits

The result of vits on map is better than vtpeak method. Because in Iterative threshold, all pixels are considered in determining the threshold.

4. Adaptive Threshold

4.1 How does it work?

1) Paritition image into several local regions (patches).
2) Determine a threshold for each region.
3) Interpolate threshold for each pixel.

4.2 Results on map

```
1. vpatch p = 64 l = 0 map | vtpeak | vquilt - p of = vtpeak64l0outmap.vx
```

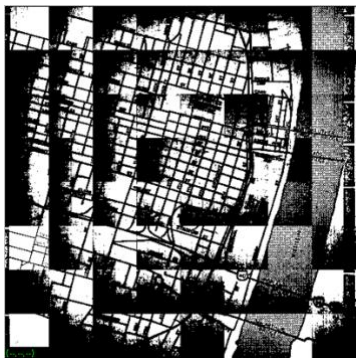


Figure 23: output vtpeak64l0outmap.vx

non-overlapping patches of size 64x64.

```
1. vpatch p = 64 l = 5 map | vtpeak | vquilt - p of = vtpeak64l5outmap.vx
```

Figure 24: output vtpeak64l5outmap.vx

5-overlapping patches of size 64x64.

```
1. vpatch p = 64 l = 10 map | vtpeak | vquilt - p of = vtpeak64l10outmap.vx
```

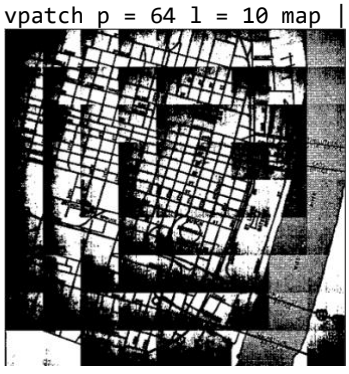


Figure 25: output vtpeak64l10outmap.vx

10-overlapping patches of size 64x64.

```
1. vpatch p = 30 l = 0 map | vtpeak | vquilt - p of = vtpeak30l0outmap.vx
```

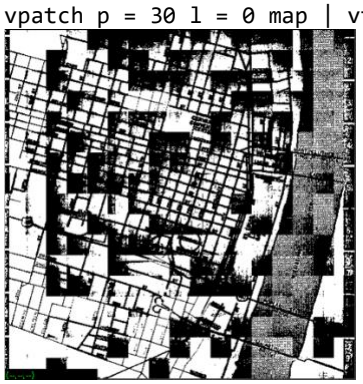


Figure 26: output vtpeak30l0outmap.vx non-overlapping patches of size 30x30.

```
1. vpatch p = 80 l = 5 map | vtpeak | vquilt - p of = vtpeak80l5outmap.vx
```

Figure 27: output vtpeak80l5outmap.vx 5-overlapping patches of size 80x80.

4.3 Results on mp.vx

```
1. vpatch p = 64 l = 0 mp.vx | vits | vquilt - p of = vitsp64l0mpout.vx
```

Figure 28: output vist64l0outmap.vx non-overlapping patches of size 64x64.

```
1. vpatch p = 64 l = 0 mp.vx | vtpeak | vquilt - p of = vtpeak64l0outmp.vx
```

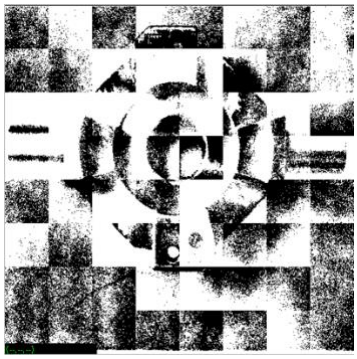


Figure 29: output vtpeak64l0outmp.vx non-overlapping patches of size 64x64.

```
vpatch p = 80 l = 5 mp.vx | vtpeak | vquilt - p of = vtpeak80l5outmp.vx
```

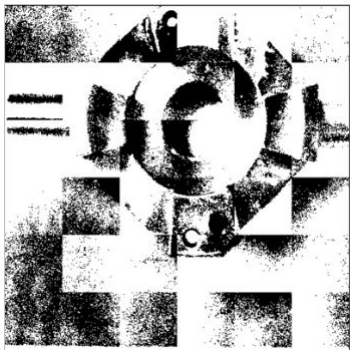


Figure 30: output vtpeak80l5outmp.vx 5-overlapping patches of size 80x80.

```
vpatch p = 30 l = 0 mp.vx | vtpeak | vquilt - p of = vtpeak30l0outmp.vx
```

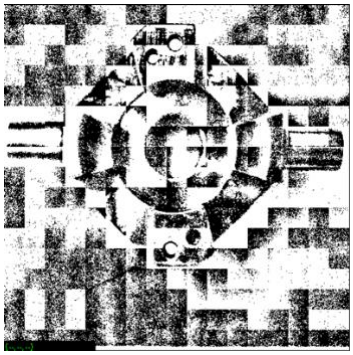


Figure 31: output vtpeak30l0outmp.vx non-overlapping patches of size 30x30.

```
vpatch p = 50 l = 0 mp.vx | vtpeak | vquilt - p of = vtpeak50l0outmp.vx
```

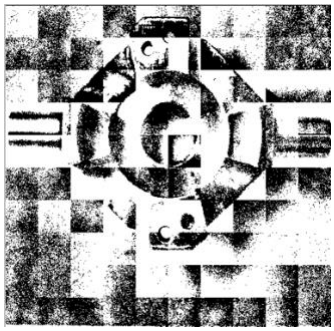


Figure 32: output vtpeak50l0outmp.vx
non-overlapping patches of size 50x50.

```
1. vpatch p = 64 l = 10 mp.vx | vtpeak | vquilt - p of = vtpeak64l10outmp.vx
```

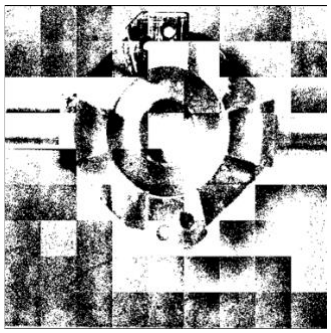


Figure 33: output vtpeak64l10outmp.vx
10-overlapping patches of size 64x64.

```
1. vpatch p = 64 l = 5 mp.vx | vtpeak | vquilt - p of = vtpeak64l5outmp.vx
```

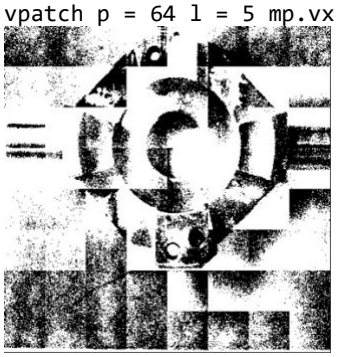


Figure 34: output vtpeak64l5outmap.vx
5-overlapping patches of size 64x64.

```
1. vpatch p = 64 l = 0 mp.vx | vtpeak | vquilt - p of = vtpeak64l0outmp.vx
```

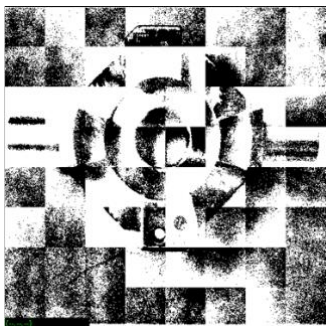


Figure 35: output vtpeak64l0outmp.vx

non-overlapping patches of size 64x64.

5. Region Growing (vgrow)

5.1 Implementation

1) Embed the image im into the image tm with a one pixel border.

   NOTE: Declare the im, vm and tm as global variable. In this way you do not need to pass them as arguments to your subprocedure setlabel

2) Clear the output (label) image, im set all pixel value in im equals 0, after clear. Then copy the image im as vm to avoid “Segmentation default” error, then use the image vm for the label (output) image
3) Traverse the input image, im.
4) If the object pixel is not labeled (i.e. the output image, vm, is zero for this pixel) and the tm is non-zero, the pixel is in range, then set h equals this pixel value in tm, then call the recursive function setlabel(y, x, h)
5) Repeat the steps 3, 4 and 6 for all unlabeled object pixels.
6) Change the value r to see the difference on region growing results.

For recursive function setlabel(y, x, h) is

1) Set the output image pixel at (y, x) to the label h, (vm(y, x) = h), set the input parameter range r equal to any number, I set r = 20
2) If the pixel above the given pixel is an object pixel, it is not labeled (vm(y, x+1)==0), and the range is smaller than r, then call setlabel(y, x+1, h)
3) If the pixel below the given pixel is an object pixel, it is not labeled (vm(y, x-1)==0), and the range is smaller than r, then call setlabel(y, x-1, h)
4) If the pixel left of the given pixel is an object pixel, it is not labeled (vm(y-1, x)==0), and the range is smaller than r, then call setlabel(y-1, x, h)
5) If the pixel right of the given pixel is an object pixel, it is not labeled (vm(y+1, x)==0), and the range is smaller than r, then call setlabel(y+1, x, h)

5.2 Source code

```
Vfread( & im, IVAL); /* read image file              */
Vfembed( & tm, & im, 1, 1, 1, 1); /* image structure with border  */
Vfembed( & vm, & im, 1, 1, 1, 1); /* preset all pixels in im to 0 implies unlabled */
for (y = im.ylo; y <= im.yhi; y++) {
    for (x = im.xlo; x <= im.xhi; x++) {
        vm.u[y][x] = 0;
    }
} /* when meet object and check isunlabled in im, then use lable function to lable it    */

for (y = im.ylo; y <= im.yhi; y++) {
    for (x = im.ylo; x <= im.xhi; x++) {
        if (tm.u[y][x] != 0 && vm.u[y][x] == 0) {
            h = tm.u[y][x];
            setlabel(y, x, h);
        }
    }
```

```
}
for (y = im.ylo; y <= im.yhi; y++) {
    for (x = im.ylo; x <= im.xhi; x++) {
        im.u[y][x] = vm.u[y][x];
    }
}
Vfwrite( & im, OVAL); /* write image file                */
exit(0);
} /* setlabel(y, x, h) function here   */
void setlabel(int y, int x, int h) {
    int r = 20;
    vm.u[y][x] = h;
    if (vm.u[y][x + 1] == 0 && tm.u[y][x + 1] != 0 && abs(tm.u[y][x + 1] - tm.u[y][x]) < r)
    {
        setlabel(y, x + 1, h);
    }
    if (vm.u[y][x - 1] == 0 && tm.u[y][x - 1] != 0 && abs(tm.u[y][x - 1] - tm.u[y][x]) < r)
    {
        setlabel(y, x - 1, h);
    }
    if (vm.u[y - 1][x] == 0 && tm.u[y - 1][x] != 0 && abs(tm.u[y - 1][x] - tm.u[y][x]) < r)
    {
        setlabel(y - 1, x, h);
    }
    if (vm.u[y + 1][x] == 0 && tm.u[y + 1][x] != 0 && abs(tm.u[y + 1][x] - tm.u[y][x]) < r)
    {
        setlabel(y + 1, x, h);
    }
}
```

5.3 Results on small test image

5.3.1 facsimile

```
vgrow facsimile of = vgrowfasixmile20
```

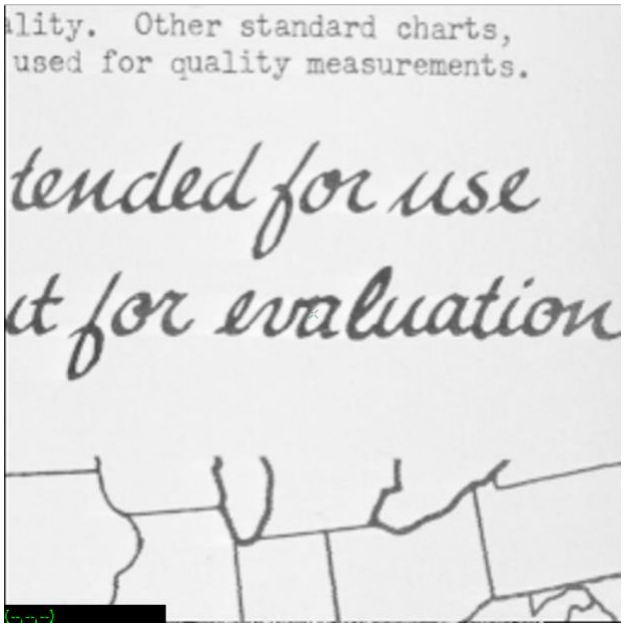


Figure 36: input facsimile

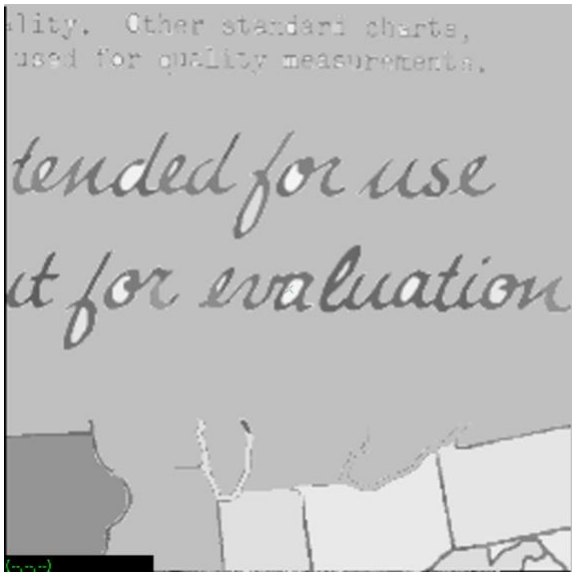


Figure 37: output vgrowfacsimile20

The range is 20.

5.4 Results on nb.vx and shtl.vx with range value

5.4.1 nb.vx

```
1. vgrow nb.vx of = vgrownb10.vx
```

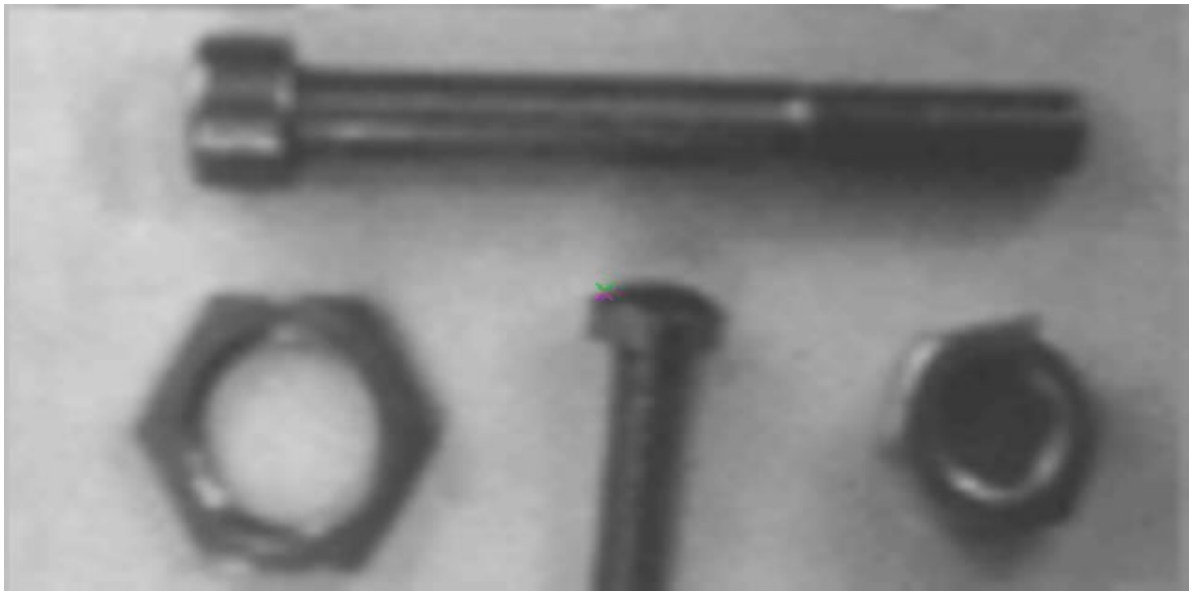


Figure 38: input nb.vx

Figure 39: output vgrownb10.vx

The range is 10.

Figure 40: output vgrownb5.vx

The range is 5.

5.4.2 map

```
1. vgrow map of = vgrowmap10.vx
```

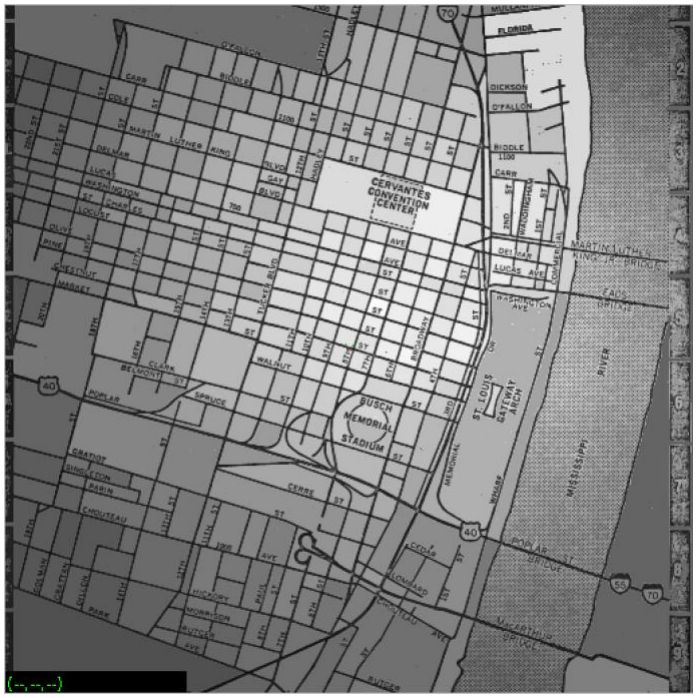


Figure 41: output vgrowmap10.vx

The range is 10.

5.5 Results on the segmented shtl image

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

5.5.1 shtl.vx dege.vx

```
1. vsobel shtl.vx of = edge.vx
```

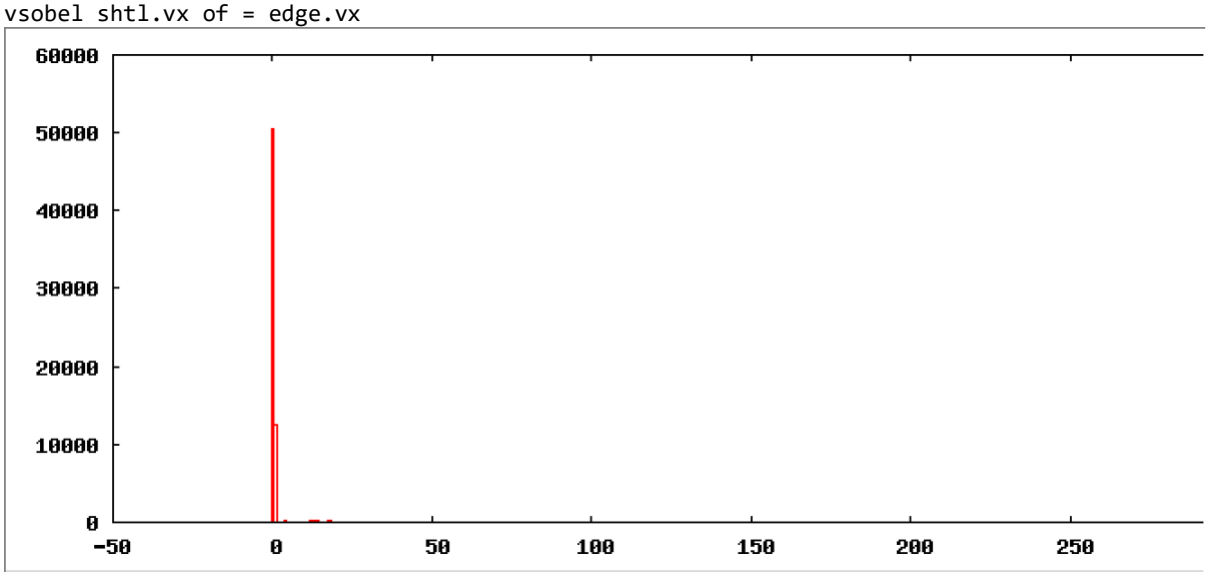


Figure 42: Histogram of edge.vx

```
1. vgrow edge.vx of = vgrowedge5.vx
```

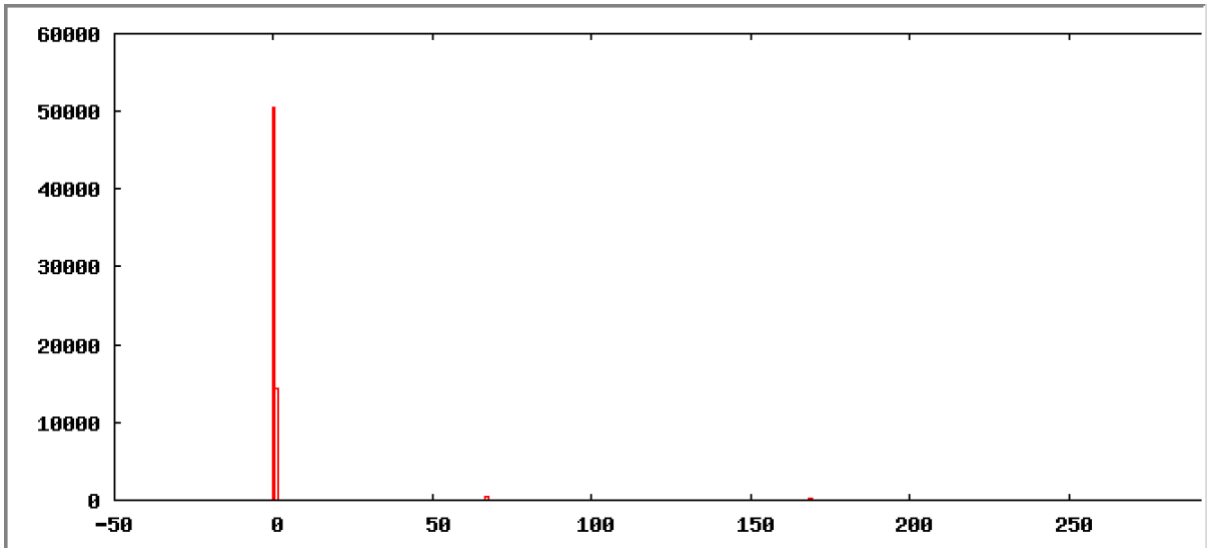


Figure 43: Histogram of vgrowedge5.vx

I tried range as 5, 10, 15, 50, and find the vgrow results on the segmented shtl image arethe same as the input segmented shtl image.
The segmented image is generated by the edge operator on the shtl.vx.