Image segmentation for stone size inspection

Jui-Pin Hsu

Chiou-Shann Fuh

Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, ROC

ABSTRACT

Object size inspection is an important task and has various applications in computer vision, for example, automatic control stone-breaking machines.

In this paper, an algorithm is proposed for image segmentation on size inspection of almost round stones with strong textures or almost no textures. We use one camera and multiple light sources at difference positions to take one image when each of the light sources is on. Then we compute the image differences and threshold them to extract edges. We will explain, step by step, picture taking, edge extraction, noise removal, and edge gap filling. Experimental results will be presented. Through various experiments, we find our algorithm robust on various stones and under noise.

Keywords: image segmentation, image difference, curve fitting, edge extraction, edge detection, gap filling, size inspection

1 INTRODUCTION

In real world, some stone-breaking machine has two rollers to squeeze-and-break stones. Using the machine, we should adjust the space between the two rollers properly, according to the sizes of the stones. On one hand, if large stones come, we should adjust the space wider, or the rollers' lifetime will be shortened; on the other hand, if small stones come, we should adjust the space narrower, or the stones will slip through, as shown in Figure 1. Hence, segmentation on stone images for size inspection is important if we want to automatically control the machine.

If stones have some high contrast textures, a traditional gradient edge detector will find many false edges. If stones have almost no textures, stereo image technique will fail on matching corresponding points.[1] Similarly, a traditional gradient edge detector will fail on finding edges between two stones with almost the same brightness. That is our motivation to use multiple light sources at different positions and image differences to extract edges.

After edges are extracted, we will remove the noises to get a clearer edge image. Due to some inevitable problems (explained later), the edges will not be all linked. General edge linking techniques[2] seem not able to solve the problem. Snake algorithm[3] may fill the gaps but will be slow. We will propose a fast algorithm to fill the edge gaps.

 0-8194-1858-7/95/$6.00

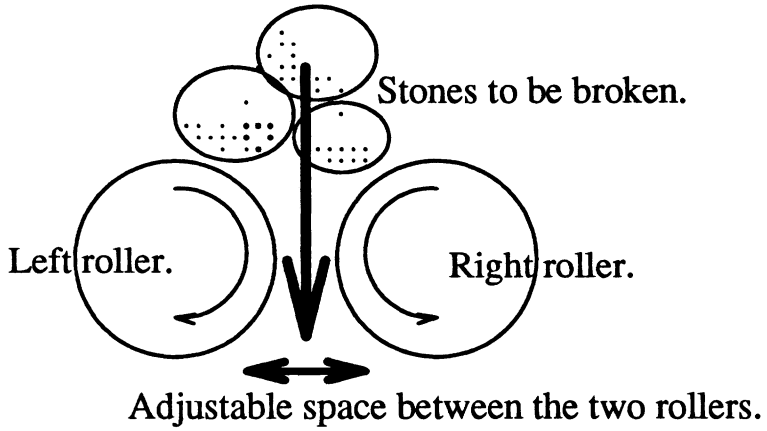

Figure 1: Side view of rollers of a stone-breaking machine.

2 SETUP FOR PHOTOGRAPHING

We are to take four images with four light sources at different positions. Figure 2 shows the setup with light source at east (E) of the camera. Other light source positions are similar, and at north (N), south (S), and west (W) respectively. The four source images are shown in Figure 3, which is taken by CCD camera with lens of focal length 25mm, The resolution is 470H×512W, and the distance between lens and the stones is about 62cm. Each light source is about 7cm to 8cm away from the center of the lens. Sizes of stones are about 5cm×5cm×2cm to 8cm×6cm×5cm.

3 ALGORITHM AND EXPERIMENTAL RESULTS

3.1 Absolute image difference

Let the four images taken be N, S, E, and W, corresponding to the images mentioned in last section. Then we compute $||N - S||$, $||E - W||$, where $||\bullet||$ stands for absolute value.

As we can expect, the absolute image difference $||N - S||$ will have larger values at the horizontal edges of the stones, but smaller values at other directions and non-edge pixels, since the shadows will differ at horizontal edges. Similarly, $||E - W||$ has the same effect at vertical edges. Figure 4 shows the histogram equalized absolute image differences for viewing.

3.2 Global and local thresholdings

A single global thresholding, which causes large gaps in edge image, seems not sufficient to extract edges properly. We combine a local thresholding and a global thresholding to extract edges.

By local thresholding, we mean that a pixel must have intensity $\alpha\%$ higher than the average intensity of its local 7×7 window area to pass the threshold. The percentage α is set to 20 in our experiment. Applying local thresholding on $||N - S||$ and $||E - W||$, we get two images $LBIN_{ns}$ and $LBINew$, as show in Figure 5.

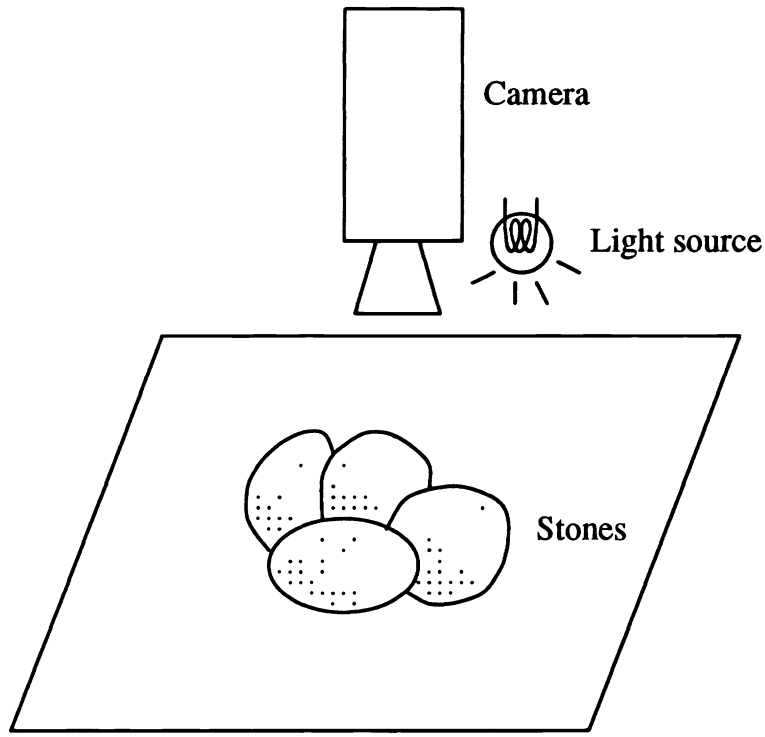

Figure 2: Global view of the camera setup.

By global thresholding, we binarize images, with threshold at a little higher percentage ($\beta\%$) than the expected percentage ($\gamma\%$) of edge pixels. The percentage γ is typically about 15, and β is set to 22. That is, the brightest 22% pixels will be marked white. Applying global thresholding to $||N - S||$ and $||E - W||$, we get $GBIN_{ns}$, $GBIN_{ew}$. is shown in Figure 6.

To combine local thrsholding and global thresholding, we compute $BIN_{ns} = LBIN_{ns}\ AND\ GBIN_{ns}$, and $BIN_{ew} = LBIN_{ew}\ AND\ GBIN_{ew}$, as shown in Figure 7.

We rely highly on local thresholding, which extracts edges effectively. However, local thresholding is not stable when the intensity is small, since a little change in value will make large change in percentage. Global thresholding help it to eliminating false edges. On the other point of view, global thresholding is good except it will find false edges on the bright belts area caused by direct reflectance of light source. Since intensities in the bright belts' area are usually high and do not vary much, local shresholding will reject them from edges.

To gather both N-S and E-W edge information, we compute $EDGE_{noisy} = BIN_{ns}\ OR\ BIN_{ew}$, as shown in Figure 8.

3.3 Noise removal for edge image

After the thresholdings, the edge image has noises of small spots. For noise removal, we use connected component algorithm on $EDGE_{noisy}$ to filter out small white components less than 40 pixels and small black components less than 400 pixels. The result image, $EDGE$, is shown in Figure 9. We assumed that stones whose areas are smaller than 400 pixels do not exist. Practically, the stones of such small sizes are unimportant and thus ignorable.

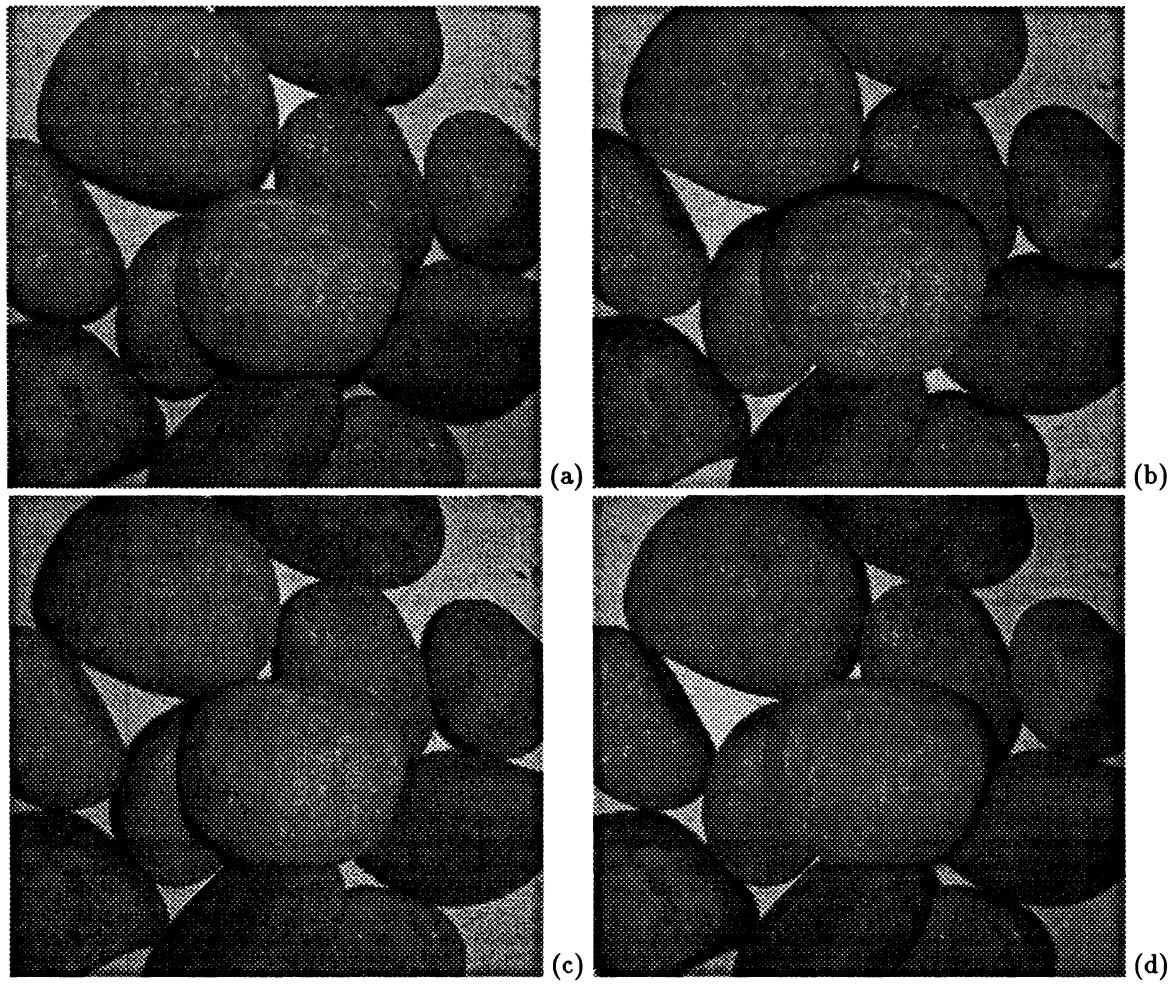

Figure 3: Source images with light sources at four directions (a) N, (b) S, (c) E, and (d) W of the camera, with resolution 470H×512W.

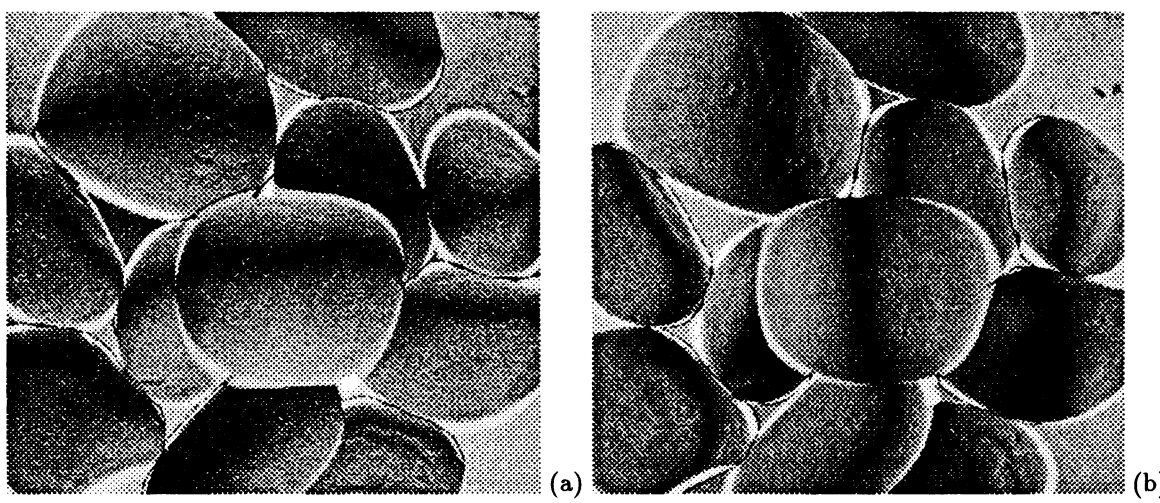

Figure 4: Histogram equalized absolute image differences for viewing: (a) $He(\|N - S\|)$. (b) $He(\|E - W\|)$. $He(\bullet)$ stands for histogram equalization.

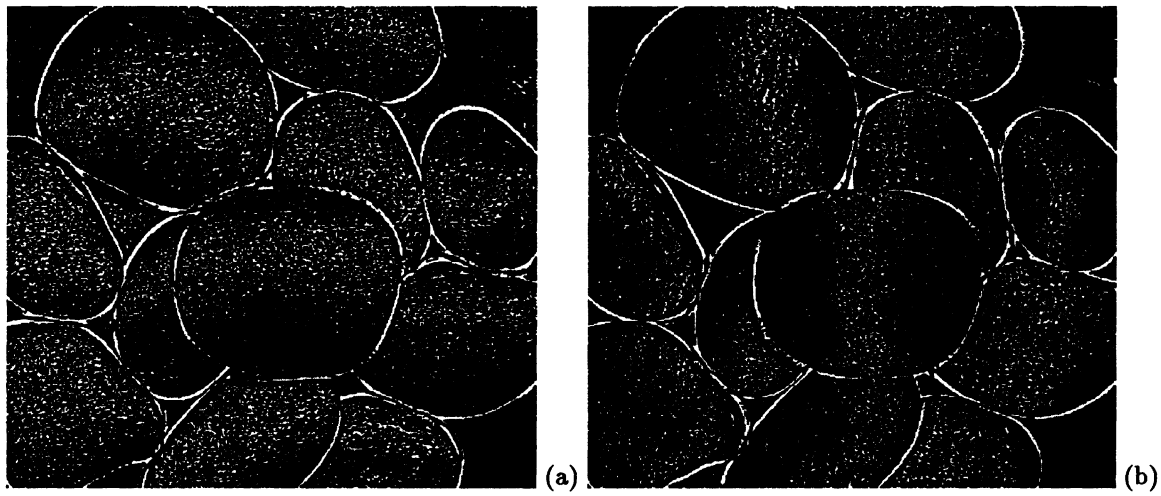

Figure 5: Local thresholding: Pixels 20% brighter than average brightness in the local 7×7 area will pass the threshold and be marked as white. (a) applied to $||N - S||$, and (b) applied to $||E - W||$.

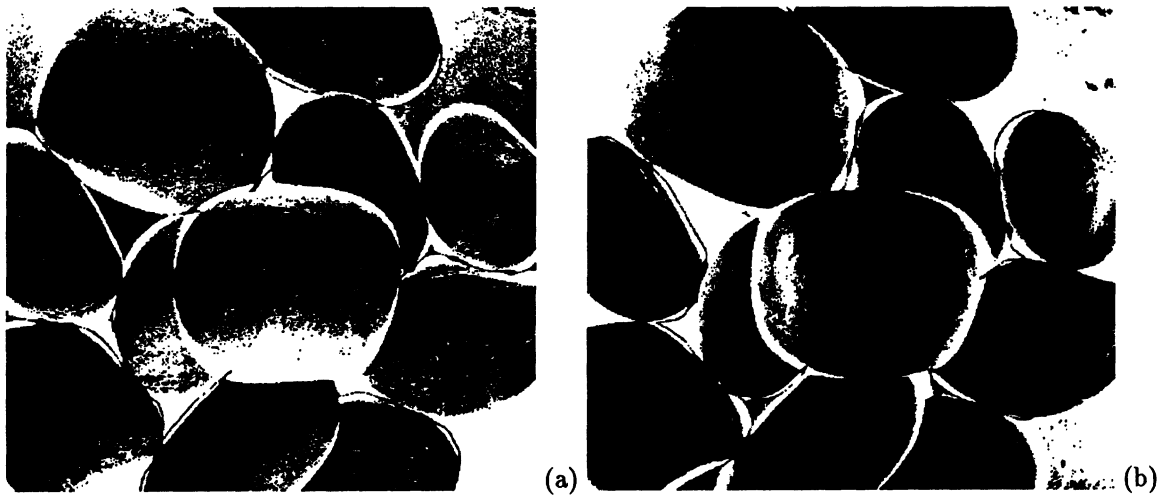

Figure 6: Global thresholding: The brightest 22% pixels will be marked white. (a) applied to $||N - S||$, and (b) applied to $||E - W||$.

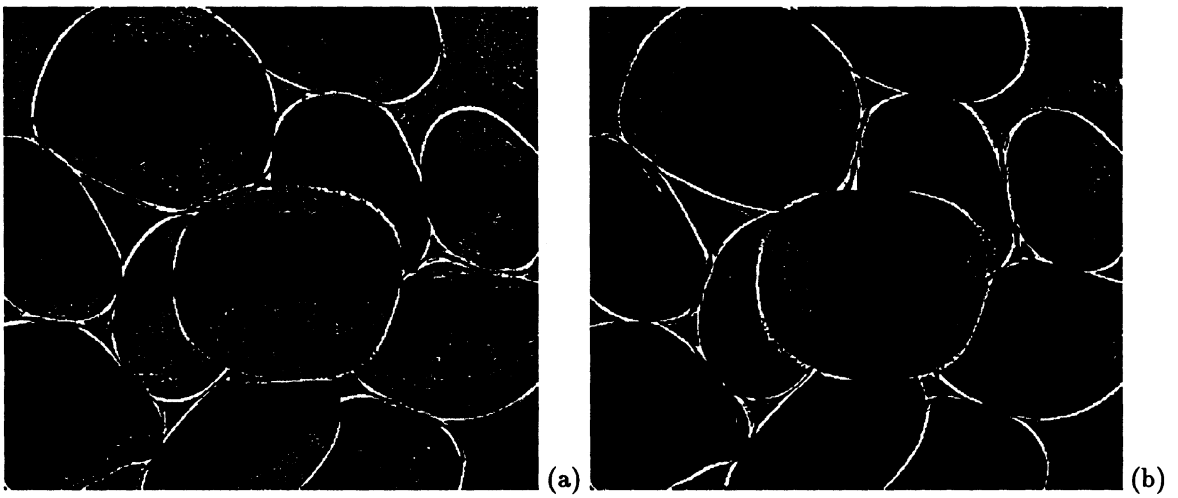

Figure 7: Combination of local thresholding and globle one: Pixels pass both local and global thresholds will be marked white and otherwise black. (a) BIN_{ns}, binary-AND of Figure 5a and Figure 6a, (b) BIN_{ew}, binary-AND of Figure 5b and Figure 6b.

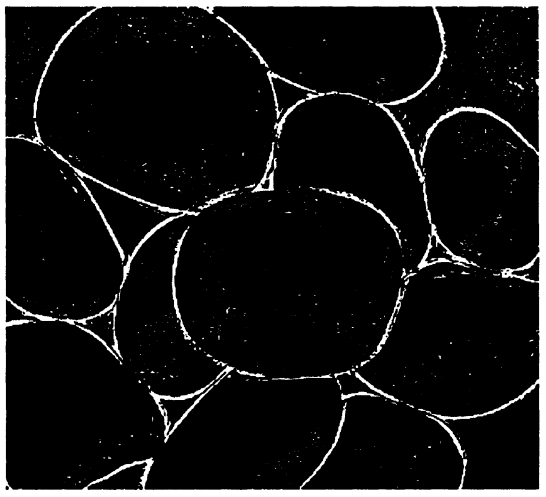

Figure 8: $EDGE_{noisy}$, gathering N-S and E-W information: binary-OR of Figure 7a and Figure 7b.

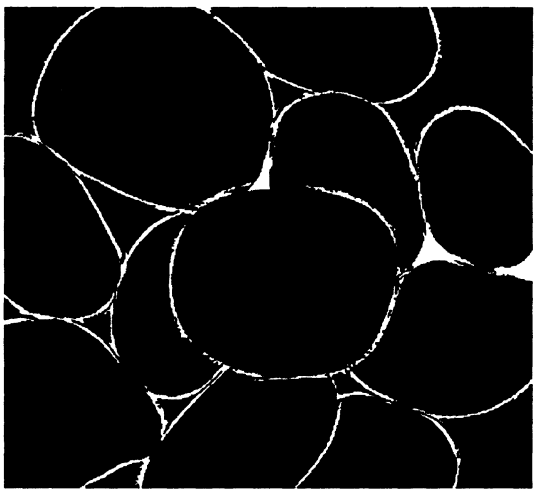

Figure 9: $EDGE$, noise removed edge image: White connected components less than 40 pixels and black connected components less than 400 pixels in $EDGE_{noisy}$ are filtered out.

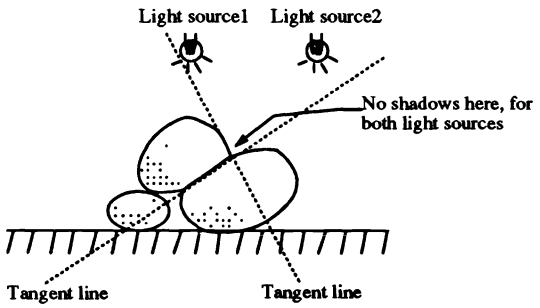

Figure 10: No shadow for image taken under each of the two light sources at the same sides of both tangent lines, so no significant difference on the edge.

3.4 Preprocessing for edge boundary tracing and gap filling

The edge image has many gaps. Figure 10 shows one of the reason why edge gap forms. We will trace the edge boundaries and fill the edge gaps.

Before edge boundary tracing, we dilate each edge pixels in $EDGE$ to all its eight neighbors, called $EDGE_{thick}$, as shown in Figure 11. Then we find all boundary pixels in $EDGE_{thick}$, called $EDGE_{boundary}$, as shown in Figure 12. Adopting dilation is to prevent edge boundary from being mistraced, as explained in Figure 13.

3.5 Edge boundary tracing and gap filling

We will show how to trace $EDGE_{boundary}$, and to use modified 2D Lagrange polynomial curve fitting, for extrapolation to fill the gaps.

First, we find the two end pixels of each piece of edge boundary $p(t)$, as the beginning pixel $p(1)$ and the ending pixel $p(N)$ for tracing. The number N denotes the total number of pixels in $p(t)$. For cyclic piece $p(t)$, $p(1) = p(N)$, and $N-1$ is the total number of pixels in the piece. Figure 14 illustrates a piece of edge boundary in $EDGE_{boundary}$.

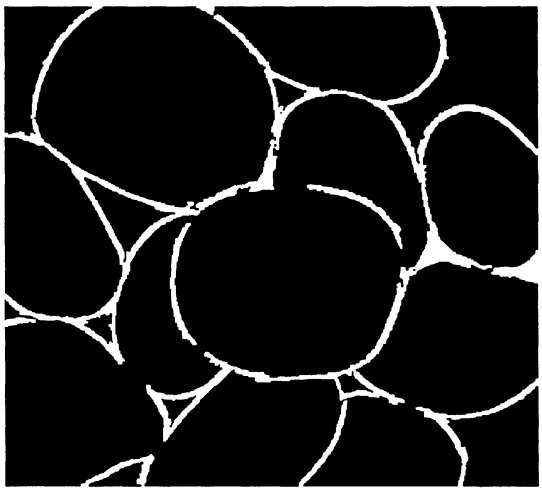

Figure 11: $EDGE_{thick}$, dilation of $EDGE$: preventing edge boundary from mistracing, see text and Figure 13.

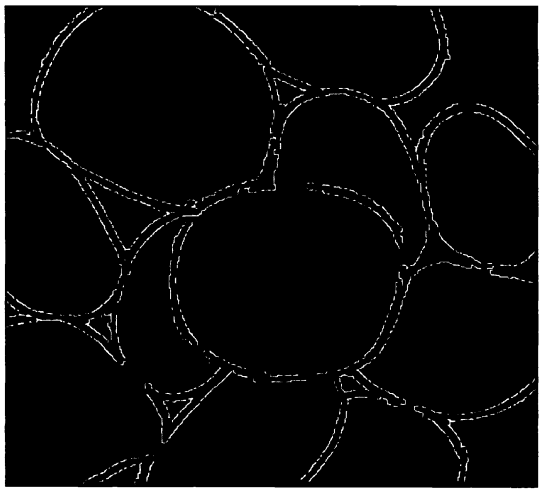

Figure 12: $EDGE_{boundary}$, boundary of $EDGE_{thick}$: to be traced.

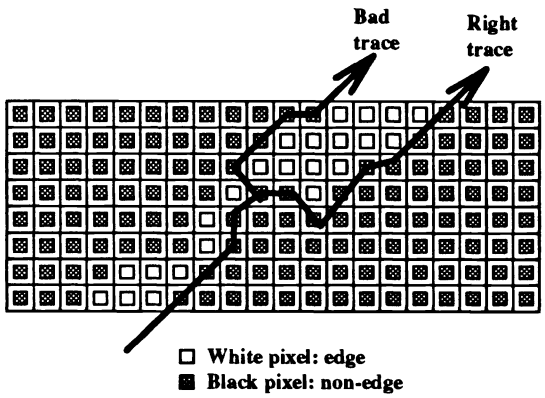

Figure 13: Mistrace of edge boundary: the arrow in the right shows the right trace, and the arrow in the left shows a bad trace.

Figure 14: A boundary piece in $EDGE_{boundary}$.

Second, we determine a step size s, for piecewise and imbricated fitting a piece of edge boundary $p(t)$, by the following algorithm:

{fit the last piecewise curve}
$curve\_fit(p(N-s),p(N-s\ div\ 2),p(N))$;

{fit the remaining piecewise curves}
for i:= 1 to $N-s$ step $s\ div\ 2$ do
$curve\_fit(p(i),p(i+s\ div\ 2),p(i+s))$;

The $curve\_fit(p(i),p(j),p(k))$ procedure is to compute the polynomial curve fitting of the three pixels p(i),p(j),p(k). The polynomial curve $c(t)$ computed by the procedure $curve\_fit(p(i),p(j),p(k))$ is defined as the following:

$$c(t) = \frac{(t-j)(t-k)}{(i-j)(i-k)}p(i) + \frac{(t-i)(t-k)}{(j-i)(j-k)}p(j) + \frac{(t-i)(t-j)}{(k-i)(k-j)}p(k)$$

One can easily check that $c(i) = p(i)$, $c(j) = p(j)$, and $c(k) = p(k)$. Thus $c(t)$ pass the three pixels. s is set to 30 in our experiment.

Third, we extrapolate the computed curves by ext pixels to fill the gaps. Parameter ext is set to 18 in our experiment. Figure 15 shows the result, called $CURVE\_FIT$. Note that only the extrapolation part use curve fitting. Pixels on original piece of curves do not use curve fitting but are drown originally. Except that, when $p(i)$, $p(j)$, $p(k)$ form a sharp angle ($< 130°$), we don't fit the piece. Those pieces are the U-turns (see Figure 14). Curve fittings on them are nonsense.

3.6 Recall of edge image and small spot noise removal

Extrapolation only links boundaries of edges. So, we fill $EDGE$ onto the result $CURVE\_FIT$ by binary-OR, called $EDGE_{fill}$, as shown in Figure 16. Then we filter out the small black connected component less than 400 pixels in $EDGE_{fill}$ to gain a better edge image, called $EDGE_{better}$, as illustrated in Figure 17.

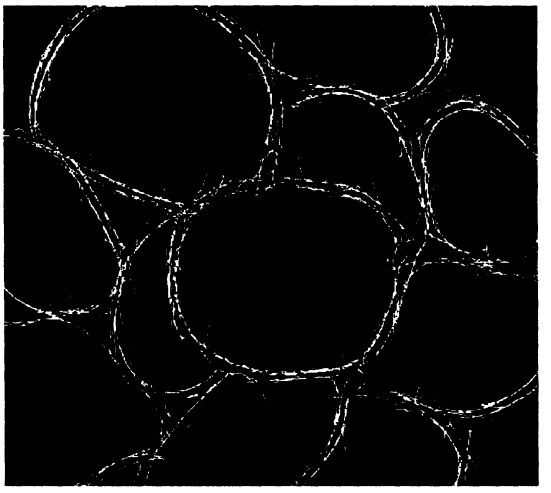

Figure 15: $CURVE\_FIT$, piecewise curve fitting of $EDGE_{boundary}$. modified 2D Lagrange polynomial is used. (see text)

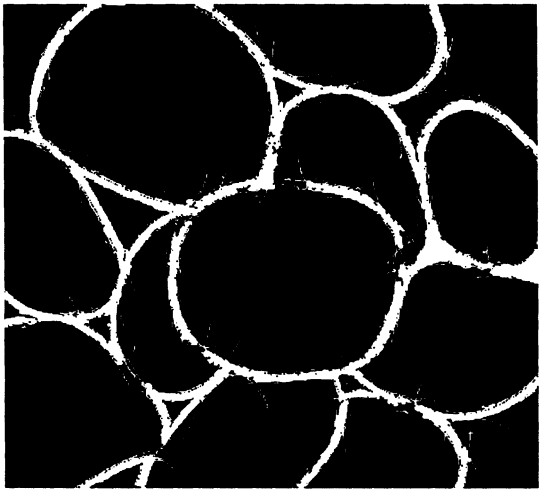

Figure 16: $EDGE_{fill}$, filling the original edge image EDGE onto $CURVE\_FIT$.

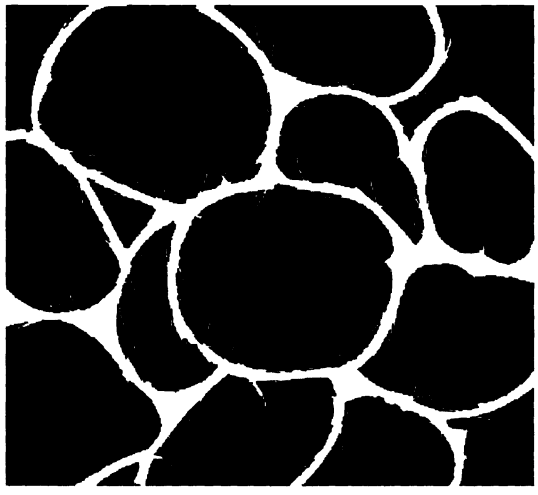

Figure 17: $EDGE_{better}$, small black connected component less than 400 pixels removed from $EDGE_{fill}$.

3.7 Unwanted curve segment removal

Extrapolation generates many small noisy curve segment. We will clear them in last result image as the followings.

Define $n(p)$ to be the number of white pixels in the local 9×9 slice window. Define boolean function $b(p)$ = "if black pixels in the local 9×9 slice window of p are all in the same black connected component". Then we follow the algorithm:

for all white pixels p in image do
    if ($n(p) < 40$) and ($b(p)$=TRUE) then
        assign p to be black.

Note that function $b(p)$ requires a connected component algorithm, to mark different components. Test on $b(p)$ is necessary, since we may have some one-pixel-wide edges. Figure 18 shows the result, and we complete the whole segmentation.

3.8 Performance

The algorithm above is implemented on Sun Sparc 10 machine. Although the program is not optimized, it can still complete the work in three minutes. Practically, it should be done in seven seconds, including stone size computing, which is not included in this paper. Yet, the time can be shortened by better machine, optimized program and perhaps program of parallel version with parallel computer.

4 CONCLUSION AND FUTURE WORK

We proposed an algorithm, including picture taking, edge extraction, noise removal, edge gap filling, for stone image segmentation. Our key idea is to use image differences to be able to process both high-textured and almost non-textured stones. Practically, stones can indeed have high textures or almost no textures. At edge gap filling, we

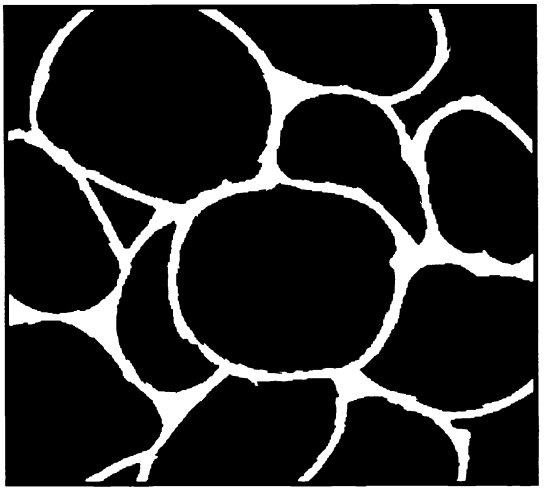

Figure 18: The complete segmentation result, noisy curve segment removed from $EDGE_{better}$.

make use of the modified 2D Lagrange polynomial to do piecewise curve fitting to fast fill the edge gaps.

We assumed that small stones are unimportant, and ignorable. If they are to be considered, higher resolution camera has to be used. If sizes of stones differ very much, higher resolution camera with large field of view has to be used. But in case of controlling stone-breaking machines, that is not necessary, since preventing rollers from being broken is more important than avoiding stones from slipping through. So stones smaller than a certain size can be ignored.

The future work is to compute the sizes of the stones, after the segmentation. The polynomial curve fitting information may be useful in computing sizes of stones.

Acknowledgements

This research work was supported by National Science Council of Taiwan, ROC, under NSC Grants NSC 83-0422-E-002-010 and NSC 84-2212-E-002-046, and by Cho-Chang Tsung Foundation of Education under Grant 84-S-26.

5 REFERENCES

[1] R. M. Haralick and L. G. Shapiro, *"Computer and Robot Vision,"* Vol. 2, pp. 357-362, Addison Wesley, Reading, MA, 1993.

[2] R. C. Gonzalez and R. E. Woods, *"Digital Image Processing,"* Addison-Wesley, Reading, MA, 1992.

[3] F. Leymarie and M. D. Levine, *"Simulating the Grassfire Transform Using an Active Contour Model,"* IEEE transaction on Pattern Analysis and Machine Intelligence, Vol. 14, No. 1, pp. 56-75, January 1992.