On Memory-constrained Algorithm for Contour Representation of Binary Image

Satoshi Konno (0910752)

School of Information Science,
Japan Advanced Institute of Science and Technology

August 10, 2013

Keywords: algorithm, memory-constrained algorithm, image recognition, edge detection.

Recently, demands for small memory algorithm are increasing in many computer applications and services such as embedded and cloud computing areas because any further improvement for performance and memory space of computers are not expected. The memory of embedded computers are small traditionally, and the memory of server computers is not enough to handle big data such as cloud computing problems.

In smartphone application markets such as Google Play and AppStore[2], the applications using image recognition are very popular. The image recognition libraries such as OpenCV[3] are developing strenuously. Image recognition is very important field for the smartphone markets.

In this paper we improve a contour representation algorithm for image recognition [4] to run more fast with more small memory. The algorithm inputs a binary image, then it outputs boundaries of connected components as edge polygons. We define the output edge polygons as "contour representation."

First, we have implemented the algorithm simply using C++ programing language on general personal computer. To apply the algorithm for actual

[2]Google Play : https://play.google.com/, AppStore : http://www.apple.com/itunes/

[3]OpenCV : http://opencv.org/

[4]Tetsuo Asano, Sergey Bereg, Lilian Buzer, David Kirkpatrick : "Binary Image Processing with Limited Storage", IEICE technical report, Theoretical foundations of Computing 110(37), pp31-38, 2010.

images such as photograph, we have to convert the image into the binary image. To create the binary image, we convert the image of RGB color space into Lab color space, then we convert a image pixel into a binary value, true or false, using the L* value of Lab color space. We have implemented the algorithm successfully, but we found that we have to select the adequate L* value to get the good output result. To select the adequate L* value, we suggest an algorithm to find the adequate value quickly. Additionally we found some preprocessing effects are effective for the output result. For example, mosaic effect which replaces some block pixel values into an average value of the pixel values is very effective for the algorithm. We found that the preprocessing effects are effective, but we didn't validate the effects in more detail because the effects aren't essential problem in this paper.

Next, we checked the simple implementation of the algorithm with C++ programming language in more detail. Then we found that implementing the algorithm is a bit complex because it contains many branch conditions such as 'if' or 'switch' expressions to find edge polygons of binary image. To implement the algorithm, we have to use many the branch conditions with the nesting. The complex branch conditions affect the performance because the branch conditions are implemented statically when the code is compiled. For example, the implementation performance might be slow when the input image has many horizontal edges and the implementation branches the vertical edge condition at first. The branches are emanated from the current search direction of movement and the surrounding pixel patterns, and there are 32 kinds of the combination. We will implement the algorithm into parallel computing platforms such as OpenCL[5], we discussed to eliminate the branch conditions to execute the algorithm in more parallel.

To improve the simple implementation, we suggest two new extended data models for binary image to eliminate the complex branch conditions in the search routine which decide a next search edge. First, we define a new extended pixel index model which has extra pixels in the neighborhood of the original pixel index model. And then, we define a new edge index model based on the extended pixel model (Figure1). The edge index model

[5]OpenCL : http://www.khronos.org/opencl/

assigns a specific index for the major edges which are referred in the search algorithm. In the simple implementation, we have to use three temporary variables, a current x, y position and a current search direction, in the main loop of the search algorithm. However we can reduce the temporary variables into a three tuple of a variable and a current edge index, using the edge index model. The edge index includes the information on three variables, and we can compute the three variables from a edge index. We define a search difference array to next edge indexes using the edge index model to decide the next search edge index from the current edge index. To decide the next search edge index, use the search difference array with the current edge index and the surround pixel pattern index of the edge pixel. Using the search difference array, we can eliminate the complex branch conditions in the simple implementation.

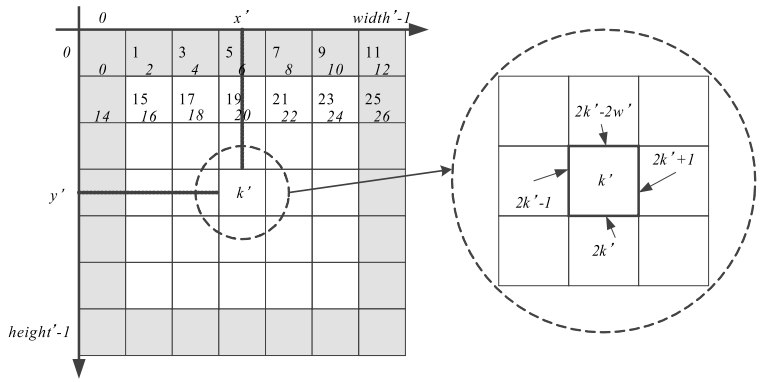

Figure 1: Edge Index Model

The original algorithm needs 52 steps for deciding a next search edge. Using the edge index model with the next search difference array, we could reduce the steps into 5 steps, then we can implement the algorithm very simply to eliminate the complex branch conditions. We checked the performance on the same computer with C++ programming language too. We prepared some square image models for the performance test to uniform the search directions. As a result, it is 1.5 times faster than the simple implementation. To eliminate the complex branch conditions, we can process any input images which has random edges in a flat time. Additionally, we think that we can implement the algorithm into the parallel computing platforms more easily because we reduced the temporary variables into

only a variable and eliminated the complex branch conditions.

Finally, We suggest a correction algorithm for the output result. The output edge polygons using the edge detection algorithm are bumpy. We suggest an algorithm which smooths the output edges. The algorithm is based on a famous reduction algorithm[6]. Using the algorithm, we can reduce the extra edges in the bumpy polygons, but we found that some edge polygons are converted into a simple line which has only two vertices. We consider that the simple lines are removable because the simple line has no area. Accordingly, we can get more simple output edge polygons to remove the simple lines from the reduction result.

[6]D. Douglas, T. Peucker : "Algorithms for the reduction of the number of points required for represent a digitised line or its caricature.", Canadian Cartographer 10, pp112-122, 1973.