MVA '92 IAPR Workshop on Machine Vision Applications Dec. 7-9,1992, Tokyo

EXTRACTION OF COLOR REGION BOUNDARIES

Stephen M. Blackburn [1]
Department of Computer Science
Australian National University

GPO Box 4 Canberra ACT 2601
Australia

ABSTRACT

The literature on region analysis is dominated by algorithms and applications for binary images. In contrast, region analysis of color images is a relatively unexplored field. This paper presents some new research that strengthens the notational support for boundary encoding of color image regions, offers a sequential algorithm for extracting color region boundaries and outlines a VLSI architecture for the extraction of color region boundaries in real-time.

I. BOUNDARY REPRESENTATION

A. Binary Images

The literature describes a number of notations for representing boundaries in binary images. Most rely on the fact that since only two colors are present, one may be considered background. Only the boundaries of the foreground regions need be defined; the background regions are defined by implication. The notations may describe either the location of the pixels on the perimeter of each region, or alternatively the *cracks* between the boundary pixels.

B. Color Images

In contrast, the notations for color regions are not so well developed. The most common notation is a simple extension of the most basic binary image notation. Each region in the image is described in terms of the pixels lying on its perimeter. However, this notation has disadvantages. It introduces a redundancy factor of two into the description. Each boundary is shared by two regions, which separately describe the same boundary.

This description also does not adequately represent the physical properties of a boundary. For example, if one wanted to simplify a complex boundary [1], the boundary descriptions of *both* bounding regions would need to be simplified. Moreover, the boundaries would need to be simplified consistently otherwise aliasing could arise between the regions.

Alternatively, each boundary may be described in terms of the *cracks* between two regions. Thus adjacent regions may use that common description, thereby avoiding the redundancy of describing each boundary twice. Also, if the boundary were simplified, the changes to the boundary would by default be reflected in the shapes of both regions. Such a notation has already been proposed [2], but not in the context of raster scan analysis.

The price of this notation is additional complexity in the data structures. Instead of each region being described in terms of a single boundary, it is now described by n *sub-boundaries*, where n is the effective number of neighbours that region has. Each sub-boundary separates exactly two regions, and is linked to exactly two other sub-boundaries at each end – one for each of the two regions the sub-boundary bounds. The exception is in the case where a region's boundary consists of precisely one sub-boundary.

As boundary approximation and image compression are two of the key objectives of this project, the crack notation has been developed.

II. A SEQUENTIAL ALGORITHM

A. Background

The only algorithm described in the literature for the extraction of color region boundaries is a simple boundary tracing algorithm [3]. This requires the entire image to be in RAM, and its completion time is dependent on image complexity.

Many binary image boundary extraction algorithms use a sliding window to analyse pixel relationships. Typically a 2×2 pixel window is slid over the image in a raster-scan fashion. As the raster-scan progresses boundaries of regions are propagated down the image via the application of three basic rules: initiation, propagation and joining.

Since only a limited number of pixel relationships may occur in such a window, a look-up table may be used to establish which of the rules are to be applied in any instance. The propagation and joining of the boundaries can be implemented dynamically through the use of regular data structures.

It is the sequential nature of such binary image algorithms that allows real-time hardware implementations [4]. However, in order to perform boundary extraction for color images in real-time, a sequential algorithm suitable for *color* images must be developed.

B. Analysis of a 2×2 pixel, n-color window

A 2×2 pixel, n-color window may contain some part of either one, two, three or four different regions. For

[1] Work done while the author was at the VLSI and Systems Technology Laboratory, University of New South Wales.

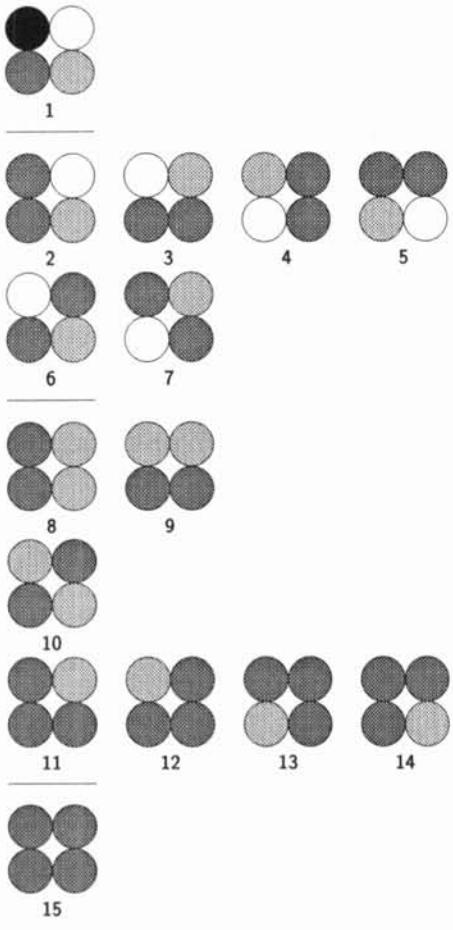

Figure 1: Possible arrangements of regions in a 2×2 pixel, n-color window.

each of these cases, there exist one or more possible arrangements of the regions (see Figure 1).

In total, there are seven equivalence classes, each appearing as a row in Figure 1. Each class has one, two or four members depending on its behaviour subject to rotation and reflection. Evidently there are only fifteen possible arrangements of regions in a 2×2 window.

The fact that only fifteen different situations can arise allows for the construction of a relatively simple sequential algorithm based on a small look-up table. Note that binary algorithms such as [5] rely on a total of sixteen arrangements; windows 8–15 in Figure 1 and their eight inverses.

The arrangements in Figure 1 are the basis for a raster-scan algorithm suitable for color images. All that remains is to establish a set of propagation rules for each of the fifteen cases. These rules may reflect either of the common notations used to describe the boundaries of color regions. In this paper, the crack-based notation will be used because of its generality.

C. Propagation rules

In addition to the three rules used in binary image algorithms (initiation, propagation and joining), a fourth rule, *termination* is added. Such a rule is not necessary in the binary context as there is no concept of boundaries terminating.

	Boundary			
State	North	South	East	West
1	T	I	I	T
2	T	I	I	
3	T		I	T
4	T	I		T
5		I	I	T
6	J	I	I	J
7	P	P	P	P
8	P	P		
9			P	P
10	P	P	P	P
11	P		P	
12	J			J
13		P		P
14		I	I	
15				

Table 1: Rules for boundary propagation for each of the states depicted in Figure 1.

Table 1 illustrates how the rules may be applied to the fifteen possible states depicted in Figure 1. Such a table will be used as the basis for a hardware implementation of the algorithm.

III. A Suitable VLSI Architecture

A. Motivation for a Real-Time System

The idea of boundary encoding images for image transmission is not new [6]. Boundary encoding may also be used as a tool for image analysis [7]. In either situation, there are applications where real-time boundary encoding is essential. This may be achieved in one of two ways: applying a standard algorithm to an appropriately powerful computer or building specialised hardware for the purpose. Given the magnitude of the problem at hand, the latter approach has been taken.

B. System Requirements

- 30 frames per second.
- Filtered and classified, 16 bit color image input.
- 512×512 pixel image.
- Continuous output of region boundaries.
- A latency of not more than 80msec.

C. Allocating space for growing data structures

The greatest difficulty with building real-time boundary extraction hardware is allocating space for boundaries as they grow. The length of a boundary may vary from a few bits to a few Kbits. This and the need to store partially built boundaries ensures that such hardware is not easy to design. Furthermore, boundaries are curves composed of one or more max-points and min-points, which means that when a raster-scan of an image is taken, various monotonically increasing or decreasing portions of curves (referred to as *queues* in [4]) that were previously assumed to be unassociated will eventually join.

If the resultant boundary is to be represented by a single contiguous list of bit sequences, as is normally

the practice, then a problem arises as to where bit sequences for a particular queue will be situated in the complete boundary. Since queues are typically numerous and small, linking them with pointers is likely to be exceedingly expensive.

One solution to the problem is to take two passes of the image [4]. In the first raster scan, data is collected on the length of each of the boundaries and its constituent queues. Before the next pass is made the data gathered in the first pass is analysed and used to produce a table indicating where each boundary and each queue is to be written in memory. When the second pass is made, boundary codes are generated and placed in the appropriate memory locations.

The two-pass approach underlies the architecture developed here. The problems posed by the relative complexity of color image boundary descriptions have meant the architecture presented here is in most other aspects quite different from that proposed in [4].

D. A pipelined architecture

A four-stage pipelined architecture is proposed. Stage one will take a raster scan of the image (see Figure 2) and produce a queue length table (Table 2). Stage two will analyse the queue length table and produce a memory segmentation table (Table 3). Stage three will take a second raster-scan and produce boundary codes, using the memory segmentation table to place them contiguously in memory. The fourth stage will output the boundary codes sequentially.

Such an architecture relies on an input image having no more than N queues. Given that the input images have been filtered and classified, choosing N as 64K is reasonable for a 512×512 pixel image [4].

D.1 Stage One

This stage establishes the memory requirements of each queue and the relationships between connected queues, writing the information to a large table implemented in RAM.

Table 2 indicates the data content of the table. For each queue, there are entries for queue length, the queue to which it is connected at its *lower* end, the sub-boundary to which it is connected associated with the region on its *lower* side, the sub-boundary to which it is connected associated with the region on its *upper* side, the color on its *lower* side and its *apparent* region number, as noted during the first raster scan. Note that there is an implicit connection between left and right queue pairs, hence the need for only one *Next Q* field.

The depth of the table is dictated by the maximum allowable number of queues (64K). Such a table would be implemented with 3×32×64Kbit static RAM. At the initiation and termination of each queue, the processor must write data to the queue length table. In the worst case (windows 1 and 6 in Figure 1) this involves 4 writes in one pixel-time. Given the 127$n$sec pixel time, a memory cycle time less than 31$n$sec is required. This is well within the limits of current static memory technology. The necessary speed is achieved by use of four I/O buffers and a degree of internal pipelining.

The processor for this stage will consist primarily of microcode controlled by the fifteen possible window states. The microcode will be responsible for the writing of data to the table. A line-store will hold pixel and queue data for the current image line.

D.2 Stage Two

During this stage, the queue length table is traversed and memory segmentation data produced. The goal is to traverse every sub-boundary from end to end, allocating memory so that the boundary codes for each sub-boundary may be written contiguously during stage three.

The procedure only involves translating region number and queue length data, so one physical table may be used by both stages one and two.

The approach taken to translating the data is to start at the first queue in the table and then follow the boundary of the region on its lower side (the region with color equal to *Color A*) until returning to the original queue. All sub-boundaries in the region boundary are traversed from end to end, except the sub-boundary to which the original queue belongs.

Whenever a queue is encountered with *Color A* equal to the color of the region being traversed (*Color A* of the original queue), the region number for that queue is updated and the queue is flagged as having had a *Color A* traversal. The region number for every such queue on a particular region boundary will thus be made consistent.

As the raster scan progresses, consistent region numbers can be propagated to all pixels in each region. The effect of this is a solution to another image processing problem, *connected-components labelling.*

The procedure is continued until every region boundary has been traversed once (all sub-boundaries are traversed exactly twice).

This algorithm may be implemented in hardware as a finite state machine. Two read and one write buffers are used to achieve the speeds required. The second read buffer is used to pre-fetch the next queue data. In the worst case the table traversal will take $8N$ clock cycles for N queues. With N=64K, this means 512K clock cycles, exactly one half-frame delay.

D.3 Stage Three

The hardware requirements for stage three are very similar to those for stage one. A second raster-scan of the image is made, during which boundary codes are generated. Data is read from the memory segmentation table and used to write boundary codes to their appropriate locations in memory. Note that the boundary codes are written in near-random order. If it were desirable, region labelled pixels could be output as well as the region boundary codes. Microcoded control logic very similar to that used for stage one reads queue data from the memory segmentation table.

D.4 Stage Four

Stage four outputs the boundary codes in sequential order. If the boundary code memory is implemented as an odd and even pair of fast static RAM chips, boundary codes can be output in pairs, effectively doubling output speed. Under such a scheme, existing technology will allow a worst case 1M boundary codes to be output in under 14msec, a .4 frame delay.

D.5 Integration

The similarity of the first and third stages, along with the simplicity of the state machines required for the second and fourth stages, can be exploited to produce one single-chip multi-function processor. Such a processor can be built with a 1micron CMOS process. The volume of data to be passed from stage to stage suggests that the pixel stream should be interleaved between three processors frame-by-frame, each processor processing a single frame through each of the four stages (Figure 4).

A guaranteed latency of less than two frames allows the use of just three processors to complete the four-stage pipeline.

IV. Conclusion

The extraction of region boundaries in real-time has a number of potential applications. This paper has outlined a notation for representing color region boundaries, a raster scan algorithm for boundary extraction and an architecture suitable for implementing in a VLSI pipelined processor.

References

[1] K. Wall and P. E. Danielsson, "A fast sequential method for polygonal approximation of digitized curves," *Computer Vision, Graphics, and Image Processing*, vol. 28, pp. 220–227, 1984.

[2] A. R. Hanson and E. M. Riseman, "Segmentation of natural scenes," in *Computer Vision Systems* (A. R. Hanson and E. M. Riseman, eds.), pp. 129–163, Academic Press, 1978.

[3] A. K. Jain, *Fundamentals of Digital Image Processing.* Prentice-Hall, 1989.

[4] J. M. Apffel, K. W. Current, J. L. C. Sanz, and A. K. Jain, "An architecture for region boundary extraction in raster scan images suitable for VLSI implementation," *Machine Vision and Applications*, vol. 2, pp. 193–214, 1989.

[5] P. E. Danielsson, "Encoding of binary images by raster-chain-coding of cracks," in *6th International Conference on Pattern Recognition*, pp. 335–338, IEEE, 1982.

[6] D. N. Graham, "Image transmission by two dimensional contour coding," *Procedings of the IEEE*, vol. 55, pp. 336–346, 1967.

[7] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, pp. 32–46, 1985.

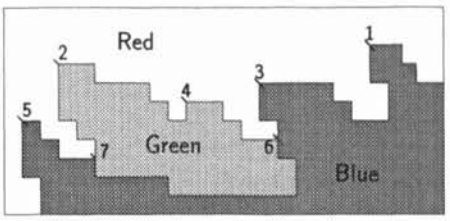

Figure 2: Three regions showing boundaries, sub-boundaries and max-points numbered in raster-scan order.

Q	*Len.*	*Next Q*	*Next s-b A*	*Next s-b B*	*Color A*	*Reg. Num*
1L	5	3R			Blue	1
1R	7	·	·	·	Blue	1
2L	7		7L	5R	Green	2
2R	10	4L			Green	2
3L	4		6L	4R	Blue	3
3R	9	1L			Blue	3
4L	1	2R			Green	4
4R	7		6L	3L	Green	4
5L	5	·	·	·	Blue	5
5R	6		7L	2L	Blue	5
6L	4	7L	3L	4R	Blue	6
6R	–	–	–	–	–	–
7L	12	6L	5R	2L	Green	7
7R	–	–	–	–	–	–

Table 2: Queue data for Figure 2 after the table building phase.

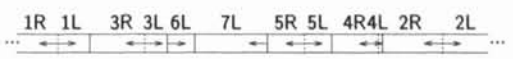

Figure 3: Segmentation of memory resulting from application of the segmentation algorithm to regions in Figure 2.

Q	*Q Start*	*Dir.*	*Next Q*	*Next s-b A*	*Next s-b B*	*Color A*	*Reg. Num*
1L	7	→	3R			Blue	1
1R	6	←	·	·	·	Blue	1
2L	74	→		7L	5R	Green	2
2R	73	←	4L			Green	2
3L	22	→		6R	4R	Blue	1
3R	21	←	1L			Blue	1
4L	62	→	2R			Green	2
4R	61	←		6R	3L	Green	2
5L	48	→	·	·	·	Blue	1
5R	47	←		7L	2L	Blue	1
6L	26	→	7L	3L	4R	Blue	1
6R	–	–	–	–	–	–	–
7L	41	←	6L	5R	2L	Green	2
7R	–	–	–	–	–	–	–

Table 3: Queue data for Figure 2 after memory segmentation and region labelling phase.

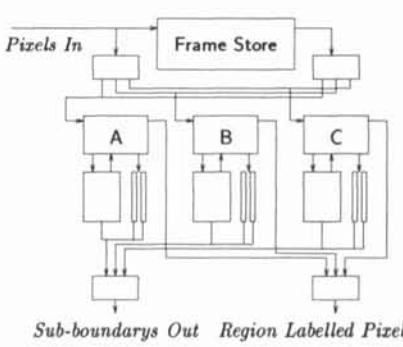

Figure 4: Pipelined boundary extraction architecture.