

A Numerical Approach for Snake Models and Implementation with an FPGA Architecture

Ali Ahmadi, Hans Jürgen Mattausch, and Tetsushi Koide
Research Center for Nanodevices and Systems, Hiroshima University
1-4-2 Kagamiyama, Higashi-Hiroshima, 739-8527, Japan
Tel: +81-82-424-6265 Fax: +81-82-422-7185
Email: ahmadi@sxsys.hiroshima-u.ac.jp

Abstract—We propose a hardware implementation for active contour models (snakes) using a numerical algorithm and an FPGA architecture, which can be used in moving object detection tasks. A digital implementation of discrete snake models is introduced and an FPGA implementation is proposed based on use of a number of parallel cell units for snake points and a main controller for all control tasks and main memory access. Each cell unit contains three parallel processing elements (PE) including three internal small memories for updating each snake point within a neighborhood of pixels. Using this parallel architecture we could obtain a snake generation time of 16ms in VGA image size (640×480 pixels) which was used for motion detection in video samples of 24 fps. The performance results are very encouraging however the system can be still improved by using more parallel processing units and simplifying some complicated instructions.

Keywords—snake models, moving objects, FPGA, VLSI, active contour.

I. INTRODUCTION

Active contour models (so called Snakes) have been widely used as an interesting approach in many applications including motion tracking, segmentation, shape modeling, and edge detection. Snakes were first introduced by Kass [1] as a means to combine low-level processing with higher-level knowledge. The algorithm was based on energy minimization of the snake as a framework where low-level information (such as image gradient or image intensity) can be combined with higher-level information (such as shape, continuity of the contour or user interactivity). The snake's tracking process is basically performed by using an initial curve and moving it to the target object under the influence of internal forces within the curve and external forces

derived from the image data. A number of approaches have been already proposed to improve the accuracy and speed of original snake model by using some new methods such as dynamic programming [3], modification of external force (balloon model) [4], multi-resolution algorithm [8], dual active contours [5], and local minimization of energy [2].

However, snake models suffer from some problems like difficulty of initialization or multiple local minima appeared in the snake updating process, still they are considered as robust and attractive models for use in many kind of applications.

A substantial hardware implementation for snake models has not yet been achieved in the literature. In this paper, first we describe a numerical approach to snake models based on the work of Williams & Shah [2] and then propose a discrete snake model hardware (DSMH) implementation with field programmable gate array (FPGA). The model design is based on use of a number of parallel cell units for snake points and a main controller for all control tasks as well as the main memory access. Each cell unit contains three parallel processing elements (PE) including three internal small memories for updating each snake point within a neighborhood of pixels. Using this parallel architecture for hardware design leads to a very satisfactory process speed in tracking of moving object in videos with a high frame rate.

The property of reconfigurability in FPGA makes it possible to design the same hardware for multi-task applications. Therefore, some other image processing steps which usually come after motion detection, like feature extraction and object recognition can be done by the same hardware as well, which is subject of our future works.

II. NUMERICAL SNAKE MODELS

A. Description

By numerical snake we mean a snake with a number of discrete points (snaxels) each updating in an iterative process, which is suitable for a digital implementation. A traditional snake [1] is defined as a curve $\mathbf{v}(s)=[x(s), y(s)]$, $s \in [0,1]$, that moves through the spatial domain of an image to minimize the energy functional

$$E_{snake} = \int_0^1 \{E_{int}(\mathbf{v}(s)) + E_{img}(\mathbf{v}(s))\} ds \quad (1)$$

with

$$E_{int} = \frac{1}{2} [\alpha(s) |\mathbf{v}_s(s)|^2 + \beta(s) |\mathbf{v}_{ss}(s)|^2] \quad (2)$$

$$E_{ext} = -\gamma |\nabla I(x, y)|^2 \quad (3)$$

where α and β are the weighting parameters that control the snake's tension and rigidity, respectively. \mathbf{v}_s and \mathbf{v}_{ss} denote the first and second derivatives of $\mathbf{v}(s)$ to s , ∇I is the gradient of pixel intensity, and γ is a user selected scalar parameter.

Williams & Shah [2] proposed a greedy algorithm for local minimization of snake points which proceeds by examining a search space around each point. The algorithm is similar to Kass model [1] but deals particularly with three energy functions referring to continuity, curvature, and image energy as follows:

$$E = \int_0^1 (\alpha(s)E_{con} + \beta(s)E_{cur} + \gamma(s)E_{img}) ds \quad (4)$$

where E_{con} , E_{cur} , and E_{img} are considered as

$$E_{cont} = d_{av} - |v_i - v_{i-1}| \quad (5)$$

$$E_{curv} = |v_{i-1} - 2v_i + v_{i+1}|^2 \quad (6)$$

$$E_{img} = (min - mag) / (max - min) \quad (7)$$

Here, v_i is a main snake point and v_{i-1} and v_{i+1} are its neighbors, d_{av} is the average distance between snake points, mag is the gradient magnitude of the current pixel, min and max are the minimum and maximum gradient values in each neighborhood. The algorithm process speed is $O(nm)$ which is much faster than Kass model with $O(nm^3)$.

Our numerical approach is very similar to Williams & Shah as we use a neighborhood of pixels around each snaxel for energy minimization, but differs in the curvature energy term. As the curvature energy should

be decreased in the snake corner points, we distinguish between normal snaxels and corner snaxels by using a dynamic value for energy weight β in (4). In this way, the curvature energy in the corner points is kept under a maximum level. Besides this, due to speed concerns the updating snaxels are done in parallel, so the average distance between snaxels (d_{av}) is updated once all snaxels get to new positions in each iteration.

B. Approach

A block diagram of algorithm used for DSMH is shown in Fig. 1. The input image is first read as an array of $m \times n$ pixels. A numerical two-dimensional gradient function is used to calculate the gradient of image through pixels intensity value. Next, snaxels are

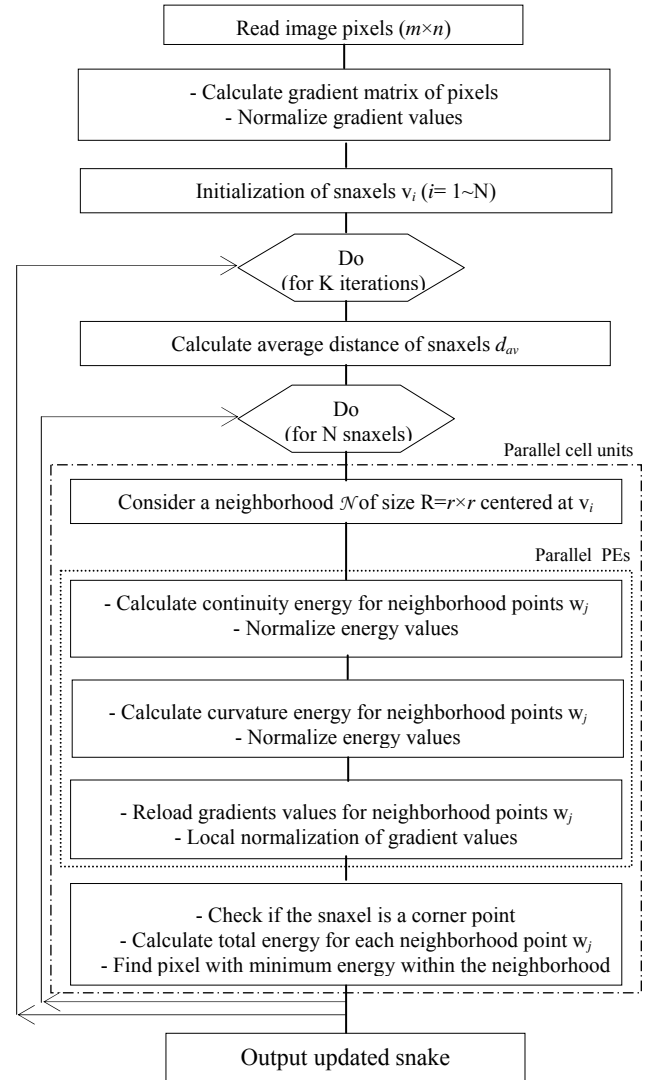


Figure 1. A block diagram of snake adaptation steps. Parallel processes are marked in dashed-line blocks.

initialized and saved in the memory. If this is a middle frame of a video sequence, we use the snake which is already generated for pervious frame, as initial snake. The main loop then starts to update the snaxels through a parallel process. For each snaxel a neighborhood pixels of size $r \times r$ is considered as search area for energy minimizing and three energy functions, that is, continuity energy, curvature energy, and image energy are determined and normalized for all pixels of the neighborhood. The energy terms are weighted with three parameters depending on the image conditions. If the snaxel is a corner point the parameter for curvature energy is changed and takes smaller value. The weighted energies are then summed to give the total energy for each neighborhood pixel. Finally, the pixel with minimum energy is selected as the target and current snaxel is replaced with it. This updating process is continued for K iterations until there is no change in the snake shape, that is, once snake catches the desired image object.

It is worth attention that in this approach updating all snaxels as well as calculating different energy functions for each snaxel neighborhood pixel are achieved in parallel, which makes the algorithm extremely fast. The first initialization of snaxels is done manually by user or can be done by the system using some prior knowledge about image texture or moving objects.

III. HARDWARE IMPLEMENTATION

Snake models may look as complicated parametrical image processing algorithm not suitable for hardware implementation. However, due to discrete characteristic of our approach besides the parallel processes taken place in each iteration, it is possible to implement a hardware in both VLSI and reconfigurable circuits such as FPGA chips. Here, we use FPGA architecture for implementing DSMH. The reconfigurability of FPGA can be used to make the hardware flexible enough for doing different tasks like object detection and object recognition with the same hardware. The FPGA pipeline capability to implement pipeline structure makes hardware sharing of cell units for several snaxels possible. The Verliog hardware description language (Verilog-HDL) is used for design hardware and the circuits are generated automatically by Synopsys FPGA compiler from high-level Verilog design. The hardware is designed such as embodies the main features of numerical snakes developed by software programs like accuracy and robustness in catching the target object,

and also to overcome some snake's shortcomings like low detection speed.

The new generations of FPGA have several millions of gates [6], on chip SRAM memory, and are very fast re-programmable without any need to physical changes in circuits. This feature allows the same hardware to be used for several different tasks keeping power consumption, space and cost down to a minimum and the processing speed at a maximum level.

A simple schematic of DSMH is depicted in Fig.2. As can be seen, the system contains two core blocks: *main controller* and *cell units*. The main controller could be thought of as a micro programmed controller which controls timing of loading and storing data input and output. It reads input data pixels as an array of $m \times n$ in the main memory, takes initialized snake points and coefficient values and saves them in the two side memories. The gradient of image pixels as well as the average distance between snaxels are also calculated by the main controller before the cells operations are started. The controller's tasks can be listed in detail as follows:

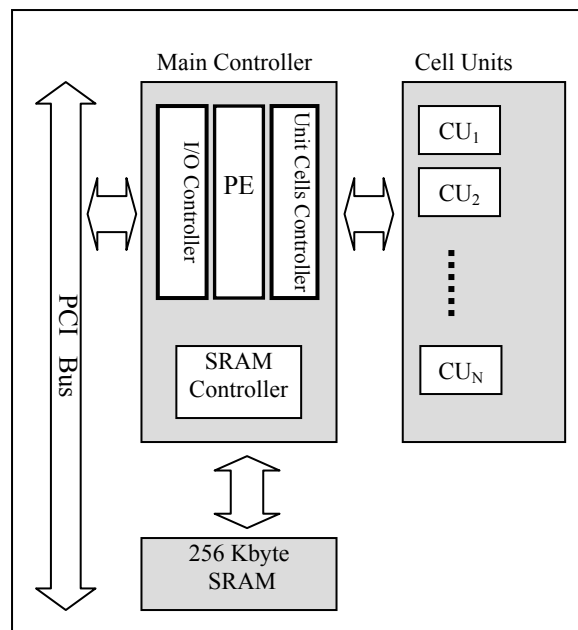


Figure 2. A simple schematic of DSMH.

- Load input image pixels in the main memory
- Load initialized snaxels (v_N) in the snaxel memory
- Load energy parameters (α , β , and γ)
- Determine gradient of pixels and save in the memory
- Determine average distance of snaxels (d_{av})
- Pass gradient and parameter values to cell units

- Fire and control the parallel operation of cell units
- Save updated snaxels in each iteration
- Stop the updating process
- Write the final snaxels to PCI

A 256 KByte SRAM memory is used as the main memory of system corresponding to the VGA image size (640×480).

Cell units are small processing units operating parallel to update snake points. Each cell encloses one controller, some processing elements (PE), three internal small memories, and some additional registers. The main PEs inside each cell are working in parallel. Also, the number of cell units can be taken as many as the number of snake points which leads to the full parallel performance of the system. A schematic of each cell unit is shown if Fig.3. Each cell receives following data as input: W_{nbh} (matrix of neighborhood pixels(7×7)), $v_{i,i-1,i+1}$ (current and neighbor snaxels), d_{av} (average distance between snaxels), g_{th} (threshold value for gradient normalization), G_{nbh} (matrix of gradient values of neighborhood pixels (7×7)), α, β, γ (coefficients for continuity energy E_{con} , curvature energy E_{cur} , image energy E_{img} , respectively).

Three parallel PEs, each containing an internal memory (7×7 bytes size) calculate three types of energy for all the neighborhood pixels. Theses energies are weighted with coefficients and summed to give the total energy value. The pixel with

minimum energy is selected as the output. The functionality of PEs are described below:

PE1:

$$E_{con} = d_{av}^2 - |w_j - v_{i-1}|^2 \quad (8)$$

$$E_{con} = E_{con} / \max(E_{con}) \quad (9)$$

PE2:

$$E_{cur} = |v_{i-1} - 2w_j + v_{i+1}|^2 \quad (10)$$

$$E_{cur} = E_{cur} / \max(E_{cur}) \quad (11)$$

PE3:

$$E_{img} = (G_{min} - G) / (G_{max} - G_{min}) \quad (12)$$

Variables are already described above. The input data W_{nbh} , $v_{i,i-1,i+1}$, and d_{av} are updated by the main controller at the end of each loop. The input gradient matrix G_{nbh} is updated for each new image frame. It should be noticed that we use Euclidean distance for calculating the neighborhood pixels distance as well as the average distance between snaxels.

IV. PERFORMANCE ANALYSIS

The estimated size and speed of the synthesized DSMH is described below. A clock cycle of 50 MHz is used for the system. Considering the functionality of parallel PEs in cell units which already described in Section III ((8)-(12)), the number of instructions for PE1, PE2, and PE3 (Fig. 3) is determined as 8R, 9R, and 3R, respectively, where R is the neighborhood size (7×7=49).

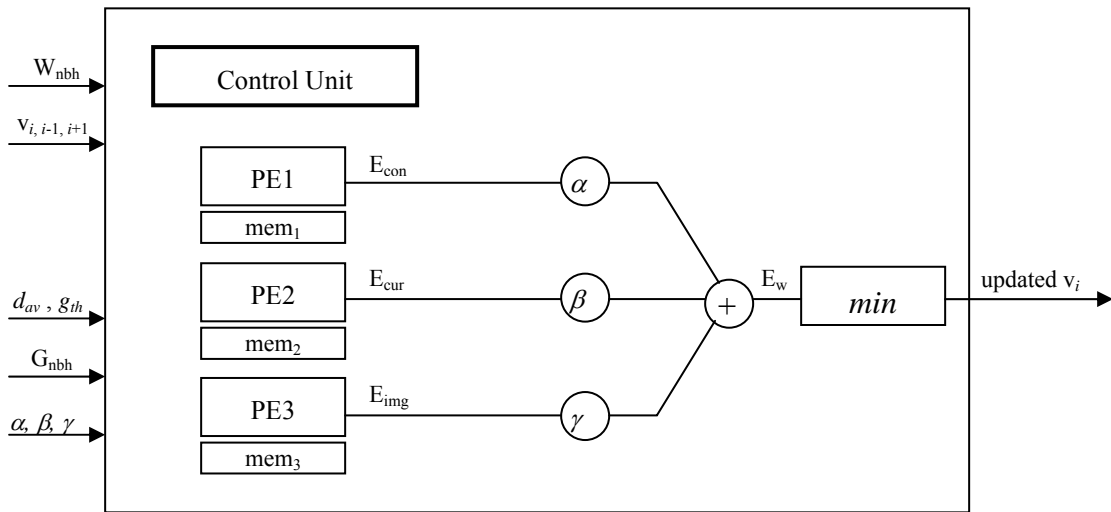


Figure 3. A schematic of DSMH cell unit. \odot = multiply signal by A. \oplus = add signals. Descriptions of variables and parameters are given in the text.

As these are parallel processing units, we only consider the longest time, i.e., $9R$. Adding this to the other sequential processes, the total number of instructions in each cell unit is determined as $(9+6)R$. As for the main controller, there are two series of instructions: The first is loading the image pixels in the memory and gradient calculation which can be considered as $2S$ where S is the number of image pixels. The second is for updating d_{av} in each iteration which is $5N+1$. The total instructions required is then $N((9+6)R + (5N+1))$.

It should be noticed that snake points are updated simultaneously since we exploit N parallel cell units for N snaxels. Accordingly, the total clock cycles necessary for complete snake adaptation process is obtained as

$$T = K(15R + 5N + 1) + 2S \quad (13)$$

Taking a clock of 50 MHz, N as 30, and maximum iteration as $K=200$, for VGA images (640×480 pixels) we will have $T \approx 16$ ms. For an input video with frame rate $\text{fps}=20$ the interval between frames is $d_i=50\text{ms}$. So we can see that the snake adaptation time for each frame is very satisfactory as $T \ll d_i$.

The system can be used for color images as we save the image pixels in format of 8-bit data (256-color or simplified RGB). In order to examine the system performance, a software simulator for DSMH written in Visual C language and some sample videos of moving objects were applied. The simulated time for snake adaptation in different video samples are reported in Table 1. Figure 4 shows a typical example of object tracking by snakes, as well.

Since there has been no other work on snakes hardware implementation, it is very hard to have an actual comparison on results. The current performance results are very encouraging specially in sense of process speed. However, still there are some problems to be solved for example the distance measure currently we use in snaxel updating is Euclid distance which is very area and power consuming and makes the cell unit design complicated. Also some conventional shortcomings of snake models are still remaining like first initialization and multiple local minima, which prevent the system to be robust enough in tracking all kinds of objects.

The system throughput can still be enhanced by using more parallel operations and simplifying complicated instructions like multiplier and divider which are very time and area consuming.

Table 1. Estimated processing time of DSMH for different input samples.

Video Sample	Iteration K	Average snake adaptation time (ms)
Moving balls VGA gray	70	13
Moving balls VGA 256-color	95	14
Human object VGA gray	120	15
Human object VGA 256-color	135	16

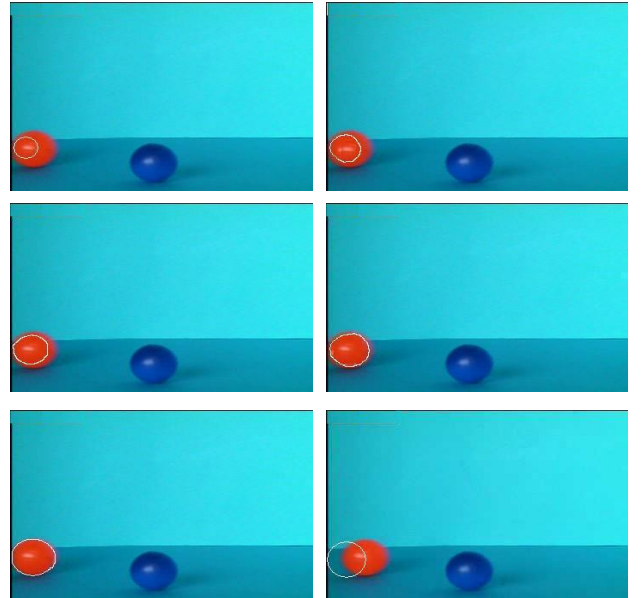


Figure 4. A typical example of object tracking by snakes for two moving balls.

V. CONCLUSION

We proposed a hardware implementation for discrete snake models using an FPGA architecture. The design is based on use of a number of parallel cell units for snake points and some parallel processing elements within each cell unit for updating each snaxel. Using this parallel architecture yields to a very acceptable snake adaptation time as 16 ms. The system still can be improved by employing some additional algorithms like pipeline construction method for reducing number of PEs and optimizing the area consumption. The reconfigurability of FPGA can be applied as an advantage for improving the system as a multi-task hardware to be used in succeeding steps of image processing like object recognition and feature extraction. The system also has the potential to be

implemented in a VLSI architecture as a parallel cell net of independent cell units with a higher speed and minimum size which is subject of our future works.

REFERENCES

[1] M. Kass, A. Witkin, and D. Terzopoulos, Snake: Active Contour Model, *Int'l Journal of Computer Vision*, Vol 1, pp. 321-331, 1987.
[2] D. Williams and M. Shah. A Fast Algorithm for Active Contours and Curvature Estimation, *CVGIP: Image Understanding*, Vol. 55, No. 1, pp. 14-26, Jan. 1992.
[3] A. A. Amini, T. E. Weymouth, and R. C. Jain, Using Dynamic Programming for Solving Variational Problems in

Vision, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 12, No. 9, pp. 855-867, Sep. 1990.

[4] L. D. Cohen and I. Cohen, Finite-Element Methods for Active Contour Models and Balloons for 2D and 3D Images, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 15, No. 11, pp. 1,131-1,147, Nov. 1993.

[5] S. R. Gunn, M. S., Nixon, A Robust Snake Implementation; A Dual Active Contour, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 19, No. 1, pp. 63-68, Jan. 1997.

[6] Xilinx technology website: <http://www.xilinx.com/>

[7] J. Waldemark, Implementation of A Pulse Coupled Neural Network in FPGA, *Int'l Journal of Neural Systems*, Vol. 10, No. 3, pp. 171-177, June 2000.

[8] B. Leroy, I.Herlin, and L.D. Cohen, Multi-resolution algorithms for active contour models, *Proceeding of 12th Int'l Conf. on Analysis and Optimization of Systems*, pp. 58-65, June 1996.