# Halftoning Soft Cores for Low-Cost Digital Displays

Muhammet Erkoç
Computer Engineering Department
Boğaziçi University
Bebek 34342 ISTANBUL, TURKEY
muhammet.erkoc@boun.edu.tr

Arda Yurdakul
Computer Engineering Department
Boğaziçi University
Bebek 34342 ISTANBUL, TURKEY
yurdakul@boun.edu.tr

Abstract-This paper presents hardware design of a family of soft cores that improve image quality on low cost digital displaying devices with limited color palette. The two-element half-toning method proposed for gray-scale images [1] has been extended for color images. It also supports real-time video streams. Experimental results on Xilinx Spartan 3E1800 FPGAs show that frame rate for 512x512 video streams is more than 150fps.

## I. INTRODUCTION

Digital halftoning is a method that has been initially designed for producing gray scale images by using only black and white dots [2]. Human eye blends halftone dots into smooth tones. Hence, different scales of gray can be obtained by merely changing the rate of black dots over a white surface or vice versa.

There have been several methods in the literature of half-toning. Methods incorporating error-diffusion are preferred due to their low computational complexity and good visual quality. Floyd Steinberg's method [3] is the first algorithm that combines a fairly good overall visual performance with a few operations. In this method, the grey value of an input pixel is compared against a palette which has different levels of gray. The closest value from the palette is selected as the output. The error between the input and the output signals is distributed to the 4 neighbors which have not been processed yet. The distribution coefficients are 7/16, 3/16, 5/16 and 1/16. However, the fixed coefficients cause visually unpleasant artifacts like worm effects in extreme gray levels, regular patches in some levels of gray, discontinuities between structured and unstructured areas.

Several variations have been proposed over Floyd Steinberg method. Among them, the ones that reduce computations are the most attractive for low-cost portable devices with reduced resources. The method proposed by Ostromoukhov [4] uses only three unprocessed neighbours to distribute error. By making some assumptions to reduce the artifacts, it has generated different coefficients for different intensity levels. The quality of the halftoned images is appreciably good in the test images. However its major drawback is the requirement of a memory space for saving different coefficients for three directions for each gray level. If 8-bit quantization is used for each tone, then the memory has to hold becomes 256x3=768 different coefficients. Ostromoukhov's assumptions have been improved by incorporating a human vision filter in [1]. As a result, the error distribution direction has been reduced to two. Moreover, only one distribution coefficient for each tone is sufficient because the sum of two coefficients is unity. As a result, 256 memory locations is sufficient for an 8-bit system. Its performance PSNR is worse than Ostromoukhov's three-element method. However its visual performance is quite good and the artifacts are significantly reduced.

The soft cores presented in this paper process real time color images in low-cost digital displays by using the method proposed in [1]. Three different IPs[1], intellectual properties, are implemented for different type of systems. The first one is for 12-bit color LCDs which are widely used on low-cost cellular phones. Second core is for 8-bit color LCDs which are used in GPS devices. The last one is for 4-bit color LCDs which are used in game console TFT's. Cores for 12-bit and 8-bit color LCDs are almost same but the 4-bit color one is slightly different from the others due to possible different palette selections. The common feature of all cores is their being independent from the size of the image. The cores have been implemented on a low-cost FPGA (Xilinx Spartan 3E1800) and the experimental results show that the frame rates are quite good: It is more than 150fps for 512x512, more than 2500fps for 128x128 streams. As a result, this core can be used for mobile TV applications on low-cost cellular phones.

The organization of the paper can be described as follows: The next section gives implementation details of the soft cores. Section III presents the resource utilization and performance characteristics of the cores when they are synthesized on the FPGAs. Section IV presents experimental results and the last section concludes the work.

## II. IMPLEMENTATION

The method proposed in [1] uses serpentine (snake) tracing as shown in Fig 1. In this method, first line is processed from left to right after this line next line is processed right to left in the reverse order. The basic design flow for all the cores are the same as shown in Fig 2. This flow is explained in detail for 12-bit color LCDs which are the common color-

[1] IP = Intellectual Property

limited displaying devices on use. The other cores are obtained by making small the variations on the core.

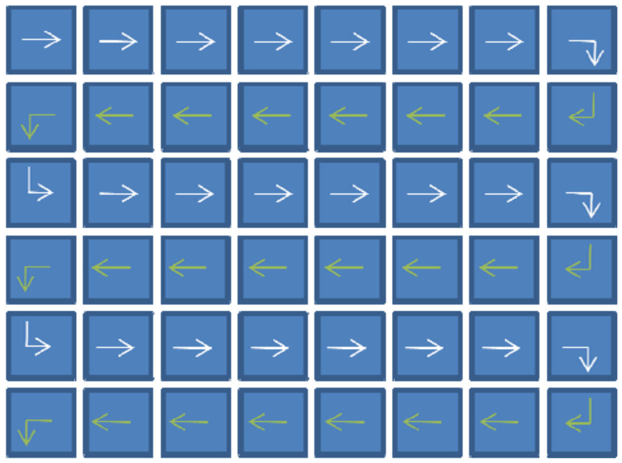


FIGURE 1: SERPENTINE (SNAKE) TRACING.

### 1) CORE FOR 12-BIT COLOR LCDS

The design consists of three different modules. *Palette Module* searches the palette to find the best matching color with the color of the current pixel. *Line Memory Module* is used for data manipulations in the memory. To*p Source Module* is used for replacing the current input pixel with the color obtained from the Palette Module and calculating new update values for upcoming pixels.

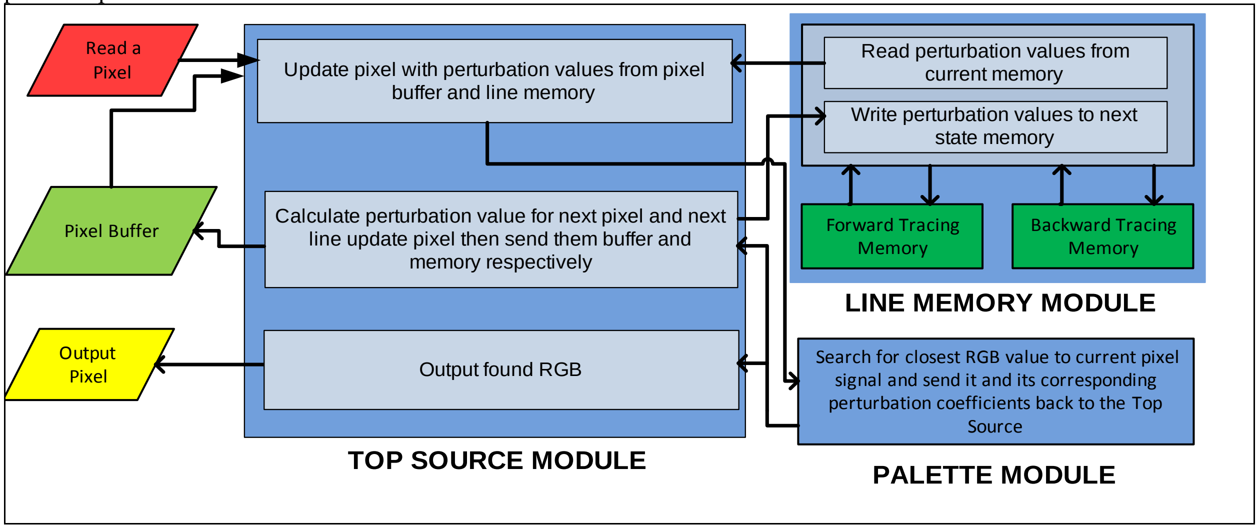
#### a) TOP SOURCE MODULE

This module processes the input pixel when clock is rising. At the beginning of the each line, perturbation value due to previous pixel in the same line is reset to 0. An end of line (EOL) signal is utilized to recognize each line in the image since we developed our core for different image sizes. EOL is also used so as to determine the direction of tracing. A register is used in order to solve this problem. After processing a line is finished, the content of the register is complemented to show the change of direction.

Perturbation values for next pixel in the same line, $P_{i,j+1}$ in the Fig. 3, are calculated when the appropriate RGB values and error values for current pixel are found. They are calculated by multiplication of 8-bit difference (error) value and 4-bit coefficient value and rounding this 12-bit result value to a 8-bit perturbation value. Updated current pixel, shown as $P_{i,j}$ in the Fig 3, is also calculated by adding or subtracting previously calculated perturbation values, due to previous pixels, shown as $P_{i,j-1}$ and $P_{i-1,j-1}$ in the Fig3.

Perturbation values for next line $P_{i+1,j-1}$ pixel are calculated by multiplying 8-bit error value and 4-bit two's complement of coefficient and rounding this 12-bit result to 8-bit perturbation value for $P_{i+1,j-1}$. This complement and rounding operations are done because coefficients are between 0 and 1 and their sum is also one. Complement operation provides producing the other coefficient 1-c. Rounding provides a coefficient between 0 and 1. After the calculation of perturbation values, they are written to the Line Memory

There are three combinational components in the top module. One of them is used for sending 24-bit updated current pixel and receiving the output RGB values from 12-bit color set, corresponding coefficients and 24-bit error value**.** The other two modules are for two memories. The first one is for reading current perturbation values from previous line, the other one is for writing next-line


Read a Pixel
Update pixel with perturbation values from pixel buffer and line memory
Read perturbation values from current memory
Write perturbation values to next state memory
Pixel Buffer
Calculate perturbation value for next pixel and next line update pixel then send them buffer and memory respectively
Forward Tracing Memory
Backward Tracing Memory
LINE MEMORY MODULE
Output Pixel
Output found RGB
Search for closest RGB value to current pixel signal and send it and its corresponding perturbation coefficients back to the Top Source
TOP SOURCE MODULE
PALETTE MODULE
FIGURE 2: DESIGN FLOW FOR ALL THREE CORES.

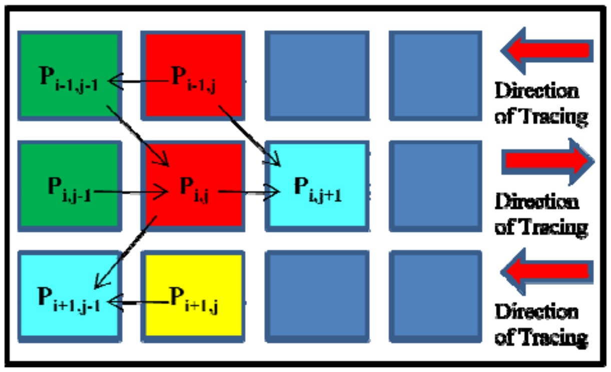


FIGURE 3: UPDATING NEIGHBORING PIXELS IN THREE SUBSEQUENT LINES.

perturbation values. Writing or reading a location is decided according to the position register.

### b) PALETTE MODULE

*Palette Module* finds closest color value from the palette for the current input pixel. In the 12-bit color palette, four bits are used for each color component independently. This means that for each color component, our palette has 16 entries. As a result, the depth of the total color palette is 16x3=48. Consequently, we used a comparison based search algorithm for R, G and B values independently. Thus the closest R, G and B values selected from the palette produces the color closest to the pixel. Though the IP is synchronized on the rising edges of the system clock, the color search on the palette is carried out on the falling-edges of the clock so as to get rid of hazards. Concurrently, perturbation coefficients are extracted. We need absolute error values and signs of them because in the case of negative error value, meaning subtraction from input pixel, we should know whether subtraction or addition is required. Also corrected RGB values, which will be sent to the palette, are limited within the interval (0,255). This restriction also prevents overflows.

### c) LINE MEMORY MODULE

Memory module is created to write 27-bit, 24-bit for correction values and 3-bit for signs of this correction values, into RAM or read 27-bit from RAM. Read or write option is determined by a *cn*, stands for current next, signal that arrange which line you are currently processing and which memory should be in write or read mode currently. Fig.4 shows how line memory module works. "RAMB16_S9_S18" type Ram was used in the design which is capable of processing 512 pixels in one line but it can be easily increased or decreased at design time. RAM memory allocation is done as follows: G and B perturbation values are written into the first 16-bit of current location via *B* port of RAM, R perturbation value is written next to the them via *A* port and their sign bits are written into their corresponding parity bits.

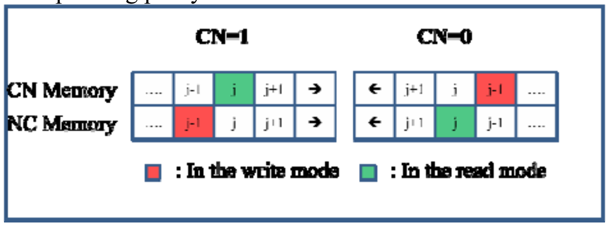


FIGURE 4: BASIC PROCESS OF LINE MEMORY.

### 2) 8-BIT COLOR

This design is almost the same as the design which was described above by 12-bit color design. The only difference between them is a smaller comparison unit and a palette, because in the 8-bit color palette 3, 3, 2-bit is used for representing R, G, B respectively. As a result, the depth of the total palette is 8+8+4=20.

### 3) 4-BIT COLOR

This design is slightly different from the other two because design cannot be done by finding RGB values independently. In the 4-bit color palette there can be different choices for better cover of all colors. Our 4-bit color representation can be seen on Table I. In this case, minimum mean-squared-error has to be calculated for each comparison. However, in the 4-bit palette 16 comparisons are enough for finding closest RGB value from palette. The same amount of comparison is done for just finding closest RGB color in the palette in the 12-bit color design.

TABLE I. 4-BIT COLOR PALETTE

| COLORS | R | G | B | $R_H$ | $G_H$ | $B_H$ |
|---|---|---|---|---|---|---|
| ⬛ | 0 | 0 | 0 | 0 | 0 | 0 |
| 🟦 | 0 | 0 | 170 | 0 | 0 | AA |
| 🟩 | 0 | 170 | 0 | 0 | AA | 0 |
| 🟦 | 0 | 170 | 170 | 0 | AA | AA |
| 🟥 | 170 | 0 | 0 | AA | 0 | 0 |
| 🟪 | 170 | 0 | 170 | AA | 0 | AA |
| 🟫 | 170 | 0 | 0 | AA | 0 | 0 |
| ⬜ | 170 | 170 | 170 | AA | AA | AA |
| ⬛ | 85 | 85 | 85 | 55 | 55 | 55 |
| 🟦 | 85 | 85 | 255 | 55 | 55 | FF |
| 🟩 | 85 | 255 | 85 | 55 | FF | 55 |
| 🟦 | 85 | 255 | 255 | 55 | FF | FF |
| 🟥 | 255 | 85 | 85 | FF | 55 | 55 |
| 🟪 | 255 | 85 | 255 | FF | 55 | FF |
| 🟨 | 255 | 255 | 85 | FF | FF | 55 |
| ⬜ | 255 | 255 | 255 | FF | FF | FF |

## III. DESIGN ANALYSIS

4-bit, 8-bit and 12-bit color designs are implemented on Xilinx Spartan-3A 1800 DSP FPGA board. Some macro and speed design statistics are given in the Table II Memory size of design is not mentioned here because memory size is depending on number of pixels used in the LCD. We simulated our placed and routed design on ModelSimSE. We also processed some images of different types and results are shown in the Table III. These images have 100x100 resolution.

Speed statistics shows that our design allows a high rate of video streaming (> 150 fps for 512x512) at a low system clock frequency (<50MHz). Thus, halftoning algorithm is not a limit for video streaming.

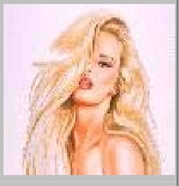
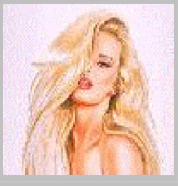
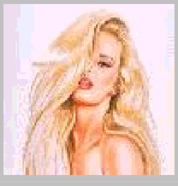
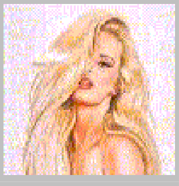
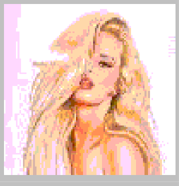
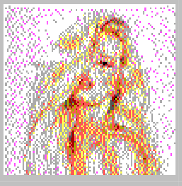
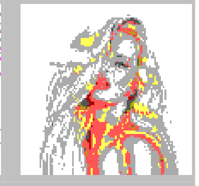
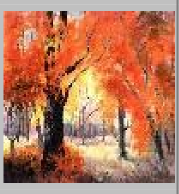
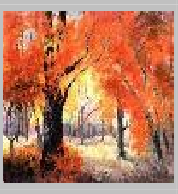
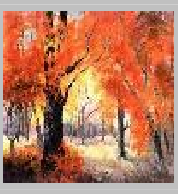
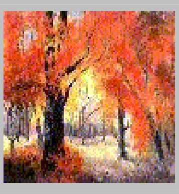
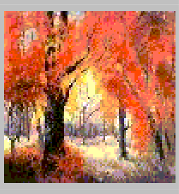
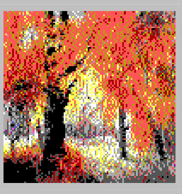
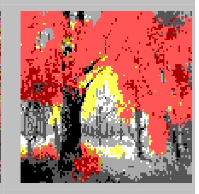
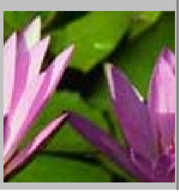
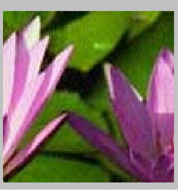
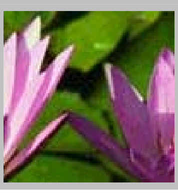
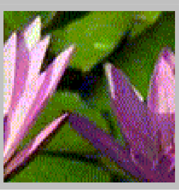
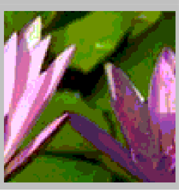
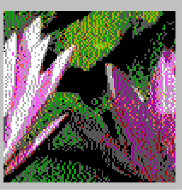
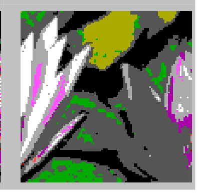
## IV. EXPERIMENTS

S-CIELab differences of 12-bit images from 24-bit original images are calculated by device independent human vision perception coefficients. Results show that rms differences of halftoned images are generally less than that of non-halftoned 12-bit images. Specifically, 9 of 13 halftoned images have less error than non-halftoned ones. Even 4 of 13 have bigger error but their error is diffused between pixels. However, in the 12-bit

TABLE II. MACRO AND SPEED STATSTICS OF 3 DIFFERENT DESIGNS

| Design | Macro Statistics | | | | | | Speed Statistics | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Slice Percentag e | Number of Multiplier s | Number of Adders and Subtractor s | Number of Counters | Number of Registers | Number of Comparat ors | Minimum Period | Maximum Frequency | Minimum Input Arrival Time Before Clock | Maximum Output Required Time After Clock | Maximum FPS for 128x128 Resolution Video Streaming | Maximum FPS for 512x512 Resolution Video Streaming |
| 4-bit color | *%3* | *9* | *92* | *1* | *25* | *4* | *22.725 ns* | *43.985 MHz* | *12.057 ns* | *5.248 ns* | *2685* | *167* |
| 8-bit color | %4 | 6 | 85 | 1 | 24 | 31 | 20.679 ns | 48.359 MHz | 13.512 ns | 5.390 ns | 2951 | 184 |
| 12-bit color | %6 | 6 | 113 | 1 | 24 | 87 | 24.016 ns | 41.638 MHz | 13.134 ns | 5.271 ns | 2541 | 158,83 |

TABLE III. HALFTONING EXAMPLES

| Original Image | Halftoned 12-bit Image | Non-halftoned 12-bit Image | Halftoned 8-bit Image | Non-halftoned 8-bit Image | Halftoned 4-bit Image | Non-halftoned 4-bit Image |
|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

non-halftoned images some pixels have very high error. Hence halftoned images seem smoother even if they have grater S-CIELab error.

In our psychological experiments, we randomly selected 20 volunteers from the university population and asked their opinions on five different images (see Table V) of totally different scenes, with the resolution of 512x512, 473x662, 506x388, 512x512 and 100x100. We showed three versions of these images, respectively original, halftoned and regular 12-bit color image, to our volunteers, then allowed them to see previous images again and look anyway they want within the daylight environment via 15 inch regular computer LCD. During experiment people are asked to answer two questions. “Which 12-bit color is better?” and “How would you grade halftoned image as a choice of ‘perfect (=5)’, ‘good (=4)’, ‘better than the regular 12-bit (=3)’, ‘no difference (=2)’ and ‘worse than the regular one (=1)’ ”. We used regular 24-bit color LCD instead of aimed 12-bit color LCD in order to avoid from 12-bit color devices’ own processing algorithm.

TABLE IV. RESULTS OF PSYCHOPHYSICAL EXPERIMENT

| Image | Halftoning Preference | Number of Grades Over 5 | | | | | Average Grade Over 5 | Preference of Halftoning |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | | |
| Lena | 17 | 3 | 2 | 7 | 6 | 2 | 3,1 | 85% |
| Iceberg | 20 | 0 | 1 | 3 | 6 | 10 | 4,25 | 100% |
| Airplane | 19 | 1 | 4 | 6 | 7 | 2 | 3,25 | 95% |
| Bird | 18 | 2 | 5 | 6 | 4 | 3 | 3,05 | 90% |
| Manga | 20 | 0 | 2 | 4 | 8 | 6 | 3,9 | 100% |
| **Average** | **18,8** | **1,2** | **2,8** | **5,2** | **6,2** | **4,6** | **3,51** | **94%** |

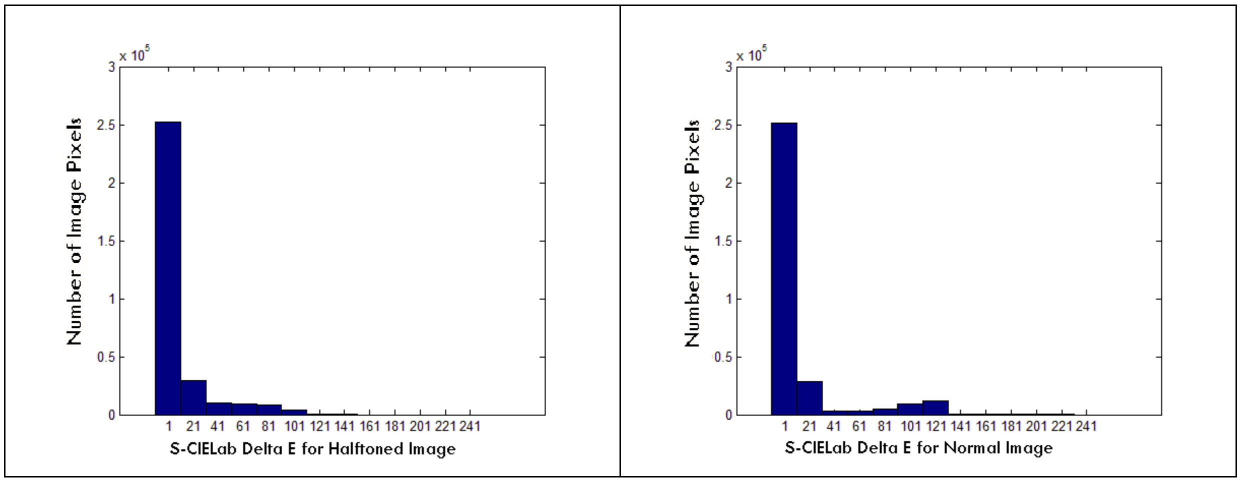


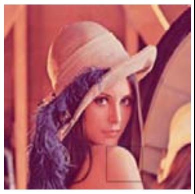
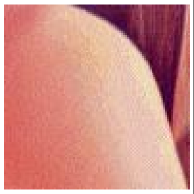
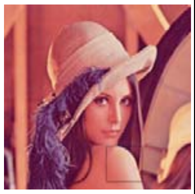
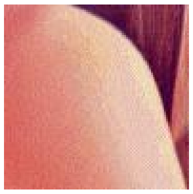
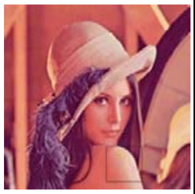
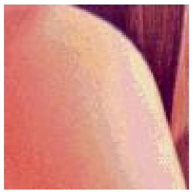
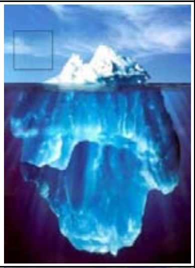
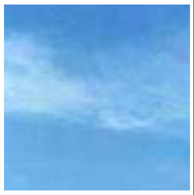
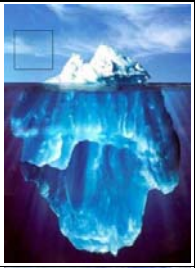
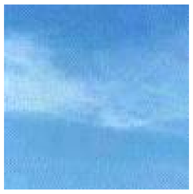
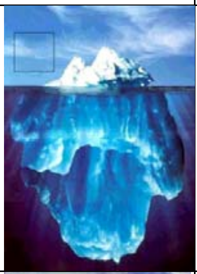
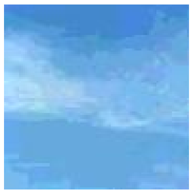
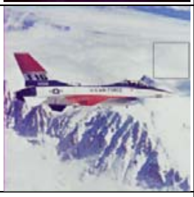
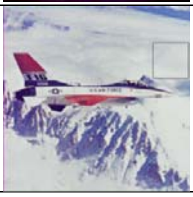
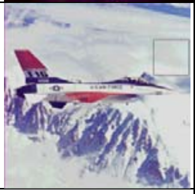
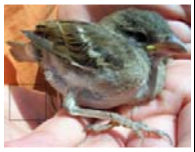
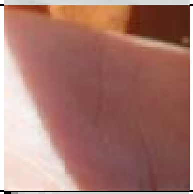
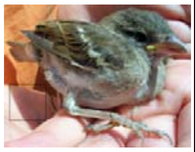
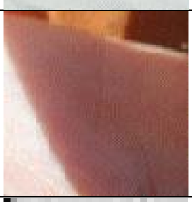
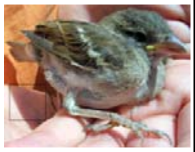
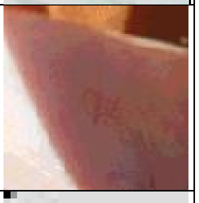
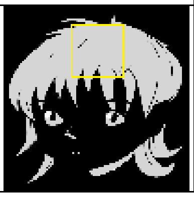
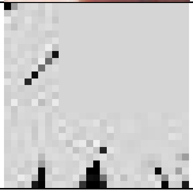
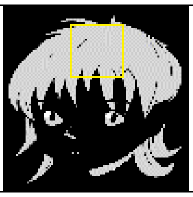
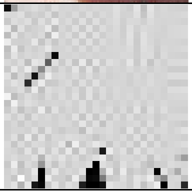
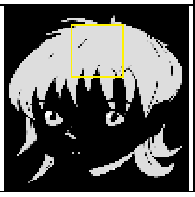
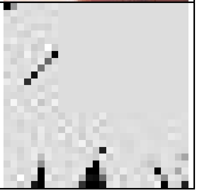
FIGURE 5: S-CIELAB ERROR OF 12-BIT HALFTONED AND NON-HALFTONED IMAGES

The statistics are shown in the Table IV. These results show us people generally prefer halftoned image to non-halftoned one in 12-bit color case. The average grade our halftoned images as 3,51 over 5. These statistics shows that halftoned images are regarded better than non-halftoned 12-bit images.

## V. CONCLUSION

In this paper, we designed soft cores for low-cost digital displaying devices by using an algorithm with low computational complexity. We have carried out both psychological experiments and scientific calculations to show that the algorithm gives good results in color images. We have also demonstrated that our core can be used in processing real time video applications.

TABLE V. IMAGES OF THE PSYCHOPHYSICAL EXPERIMENT

| Original Resolutio n | Original Image | Marked Section | 12-bit Color Halftoned Image | Marked Section | 12-bit Color Image | Marked Section |
|---|---|---|---|---|---|---|
| **Lenna 512x512** |  |  |  |  |  |  |
| **Iceberg 473x662** |  |  |  |  |  |  |
| **Airplane 512x512** |  | |  | |  | |
| **Bird 506x388** |  |  |  |  |  |  |
| **Manga 100x100** |  |  |  |  |  |  |

## REFERENCES

[1] J.M. Guo and J.H Chen, "High Efficiency Error Diffusion with Two-Element Error Kernel," *Proc. of IEEE Int. Conf. on Image Proc.*, pp. 1501-1504, 8-11 Oct. 2006.

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

[3] R.W. Floyd and L. Steinberg. An adaptive algorithm for spatial grey scale. *Proc. Soc. Inf. Display*, 17:75–77, 1976.

[4] V. Ostromoukhov, "A simple and efficient error-diffusion algorithm. Computer Graphics," *Proc. of SIGGRAPH 2001*, 567-572, 2001.

[5] H. Le Borgne, N. Guyader, A. Gudrin-Dugui and J. Hirault, "Clasification of Images : Ica Filters vs Human Perception," *Proc. of IEEE Int. Symp. on Signal Proc. and Its App.*, vol. 2, pp. 251-254, 1-4 July 2003.