A Dynamic Communication Algorithm for Digital Halftoning

Philip F. Hingston
School of Computer & Information Sci,
Edith Cowan University
Western Australia 6050
email: p.hingston@ecu.edu.au

R. Lyndon While
Dept of Computer Sci & Software Eng,
The University of Western Australia
Western Australia 6009
email: lyndon@cs.uwa.edu.au

Abstract

We present a novel algorithm for digital halftoning. The algorithm combines a technique based on error diffusion with the use of a cost function to determine termination. Its chief advantages include the use of randomness to avoid visual artifacts in the binary image and its amenability to parallel execution. The algorithm is a member of the class of "dynamic communication algorithms" which make novel use of dynamically-routed messages to structure the execution of a program.

Keywords: digital halftoning, image analysis, parallel algorithms, algorithm design.

1 Introduction

We propose a new algorithm for digital halftoning which belongs to the recently-discovered class of "dynamic communication algorithms", or DCAs[1]. Algorithms in this class are designed specifically to exploit the communication capabilities of (logically) fully-interconnected memory-passing architectures with large numbers of processors. Dynamic communication algorithms differ from traditional parallel algorithms in that they use communication in a constructive manner to minimise the computational requirements of applications. The communication pattern of a DCA is determined at run-time using data local to the sending processors. This allows processors to derive information (beyond that contained in the message itself) from the mere existence (or, indeed, non-existence) of messages. The propagation of messages as the algorithm proceeds leads to an exponential growth in such derived information. Previous studies of the class of dynamic communication algorithms have resulted in new algorithms in a diverse range of important application areas. DCAs can be emulated efficiently on sequential architectures using offset-addressing to simulate communication between processors, an operation for which all stock hardware is heavily optimised.

The remainder of the paper is structured as follows. Section 2 introduces the problem of digital halftoning and describes some previous algorithms. Section 3 introduces the class of dynamic communication algorithms in more detail. Section 4 describes our new algorithm, Algorithm Υ. Section 5 illustrates the results of applying Algorithm Υ to some sample images and also discusses a hybrid of Algorithm Υ with an existing algorithm to combine their best features. Section 6 concludes the paper.

[1]This class of algorithms has previously been called "communication-intensive, massively-parallel algorithms".

2 Digital Halftoning

The digital halftoning problem may be described as that of approximating a greyscale image with a bi-level black-and-white image. This is important both for displaying images on monochrome screens and for printing them as hard copy. The aim of halftoning is to produce a binary image that is "visually similar" to the greyscale original. The interpretation of whether two images are visually similar is necessarily a subjective one, different people having differing opinions on the relative merits of the results of halftoning algorithms. The level of "similarity" required will also depend on the expected use of the final image. Other, more objective properties of an algorithm include some measure of the "difference" between the original and final images, the performance of the algorithm on sequential machines and its amenity to parallel execution. In this paper we consider only the case where the images are represented by pixel arrays of the same dimensions.

Many halftoning algorithms are designed to match the average intensity in the neighbourhood of a pixel in the binary and greyscale images. As many of these methods tend to produce a blurred effect, edge enhancement methods may be used to sharpen the resulting images. In this paper we prefer not to implement edge enhancement so that the basic algorithms may be more readily compared.

One of the most widely used methods is ordered dither [Bayer, 1973]. This uses a small $n \times n$ *dither matrix* of threshold values, D, to tile the image. Pixel P_{ij} is set to white if the corresponding greyscale value is greater than $D[i \bmod n][j \bmod n]$ and to black otherwise. Ordered dither is highly parallel, as each pixel in the image is treated separately. Unfortunately, this simple method produces annoying visual artifacts, such as contouring and patterning, especially in large uniform areas of an image.

The Floyd-Steinberg algorithm [Floyd and Steinberg, 1976] and the JJN algorithm [Jarvis et al., 1976] are examples of *error diffusion* techniques. These algorithms perform a single linear pass over the image: each pixel is set to either black or white and the resulting error is distributed to neighbouring pixels. These methods sacrifice parallelism in order to overcome some of the aliasing effects of ordered dither. However, they introduce some artifacts of their own, such as snake-like patterning in some areas of an image.

An elaboration on error diffusion is the use of approximations of space-filling curves, such as Hilbert curves, rather than linear scans [Witten and Neal, 1982, Cole, 1991, Velho and Gomes, 1991,

Wyvill and McNaughton, 1991]. Once again, these methods are inherently serial in nature.

Knuth [Knuth, 1987] introduces the method of *dot diffusion*, which attempts to combine the advantages of ordered dither and error diffusion. The image is tiled with a fixed matrix as in ordered dither, and the image pixels are assigned classes based on the matrix value. The pixels in a class are then handled simultaneously, with errors from one class being distributed to neighbouring pixels with higher matrix numbers. The algorithm is more parallel than the error diffusion methods but still suffers from visual artifacts.

The *space diffusion* method [Zhang and Webber, 1993] combines dot diffusion with the use of space-filling curves. In this method, the image is tiled along an approximation to a space-filling curve rather than a linear scan. This aims to preserve the parallel nature of dot diffusion whilst improving its visual attractiveness.

Recently, researchers have cast the problem in terms of finding a binary image that minimises a suitable cost function. Gotsman's Algorithm HT [Gotsman, 1993] uses a cost function that matches average intensities in the original and final images. The term "average" is used here to mean a weighted average, i.e. a low-pass filter. Algorithm HT performs multiple linear scans over the binary image, at each step changing the pixel if doing so reduces the value of the cost function. The process continues until no further improvement is found. This algorithm has the virtue that it can be adapted for halftoning sequences of images from an animation with little temporal aliasing and very good compressibility. To do this, the binary image from the previous frame is used as an initial image to be refined by the algorithm for the next frame in the sequence. It is not clear to what extent this algorithm could be parallelised.

Geist, Reynolds and Suggs [Geist et al., 1993] use a cost function that includes a term measuring how well average intensities match plus a term that reflects spectral information. Their algorithm uses a neural network or simulated annealing to search for minima of the cost function. While this method produces impressive images, it is very computationally expensive and therefore most suited for production of high quality images where execution speed is not a concern. On the other hand, it can be implemented as a parallel algorithm.

3 Dynamic Communication Algorithms

Programming parallel computers is notoriously difficult. Factors contributing to this difficulty include the complexity of concurrency, the need for explicit resource management and the current diversity of parallel machine models. One important question is how to partition a program so that it makes good use of the resources available for its execution: this is commonly known as load-balancing. The standard approach to load-balancing is to statically divide up the work to be performed into a number of tasks, each of which is then assigned a particular processor for its execution. Inevitably, as a result of this division, results calculated on one processor will be needed by a task on another processor. Communication is thus required to send results from where they are calculated to where they are needed. This communication is purely an overhead of the load-balancing algorithm: because its source and destination are determined statically by the division of work, it conveys no information other than the value contained within it. We can compare this type of communication to that which results from a manager assigning a task to a worker: at the completion of the task, the worker passes the results back to the manager.

But we know from human interaction that there are other forms of communication, ones which carry implicit information additional to their actual content. We identify two forms of such communication, which we illustrate by examples.

In the first case, imagine a lottery in which there is one prize of known size. The entrants know which day the winner is to be notified: therefore simply to receive a letter from the lottery organisers on that day will be enough to convey the good news, irrespective of the content of the letter. The letter might in fact be empty! Moreover, and often more importantly, not to receive a letter on that day will be enough to convey the bad news to the losers: they receive information *without any communication taking place.* This situation corresponds to processors sending dynamic communications to each other in synchronous fashion: the mere existence or non-existence of a message in a particular cycle can impart information to the destination (or non-destination!) processor.

In the second case, imagine a worker who is up for promotion receiving a dinner invitation from the Head of his company. It may be possible for the worker to infer potential good news about his future with the company simply from the presence of such an invitation, without such news being explicitly mentioned by the Head. This situation corresponds to processors inferring information from dynamic communications by knowing how the destination was decided, and using that knowledge to deduce the values of data local to the sending processor.

The distinguishing feature of these forms of communication is that their destinations are not determined statically by the structure of the situation, they are determined dynamically using information which comes to light after that structure is created. This means that the receiver (or non-receiver) of the communication can infer something about the data or intentions of the sender, beyond what is contained in the actual message. The communication has ceased to be simply an overhead: it has become an important part of the situation. This allows us to construct new algorithms which are based around their communications in an entirely novel fashion.

Previous studies of the class of dynamic communication algorithms have resulted in new algorithms in a diverse range of important application areas, including sorting [Sharp and Cripps, 1989], tessellation [Sharp, 1990], fractal image generation [Sharp and Cripps, 1991], pattern recognition [Sharp and While, 1993b] and pitch detection in speech [Sharp and While, 1993a]. In all of these areas, a DCA has been discovered which out-performs previously-known algorithms in some respect.

DCAs can also be emulated efficiently on sequential architectures. The operation of a parallel machine is modelled by an array of records, one representing each processor. Point-to-point communications are modelled by array-indexing operations and broadcast operations are modelled by assignments to global variables. Note that offset-addressing is an operation for which all stock processors are heavily optimised: this means that, often, emulated DCAs out-perform other styles of algorithms on sequential machines.

4 Algorithm Υ

Algorithm Υ combines the technique of error diffusion with the use of a cost function to determine termination. Assume the following definitions.

P_{ij} is a pixel, $1 \leq i \leq Imax$, $1 \leq j \leq Jmax$

G_{ij} is the original greyscale value of P_{ij}

C_{ij} is the current binary value of P_{ij}

N_{ij} is the set of pixels in the neighbourhood of P_{ij}

Using these we define the following quantities.

$$N = |\ \mathsf{N}_{ij}\ |$$

$$\mathsf{B}_{ij} = \{\mathsf{P}_{i'j'} \mid \mathsf{P}_{i'j'} \in \mathsf{N}_{ij} \wedge \mathsf{C}_{i'j'} = 0\}$$

B_{ij} is the set of black pixels in the neighbourhood of P_{ij}

$$\mathsf{W}_{ij} = \{\mathsf{P}_{i'j'} \mid \mathsf{P}_{i'j'} \in \mathsf{N}_{ij} \wedge \mathsf{C}_{i'j'} = 1\}$$

W_{ij} is the set of white pixels in the neighbourhood of P_{ij}.

Note that $\mathsf{B}_{ij} \cup \mathsf{W}_{ij} = \mathsf{N}_{ij}$

$$\mathsf{E}_{ij} = \sum_{\mathsf{P}_{i'j'} \in \mathsf{N}_{ij}} \mathsf{C}_{i'j'}$$

E_{ij} is the "estimated" current intensity value in the neighbourhood of P_{ij}.

Note that $\mathsf{E}_{ij} = |\ \mathsf{W}_{ij}\ |$

We first describe the basic outline of Algorithm Υ, to highlight the dynamic properties of its communication structure. We then go on to discuss the options available to make the outline concrete.

We assume a machine which has at least as many processors as there are pixels in the greyscale image. We assign one processor to each pixel in the image. The processors execute the following steps synchronously.

1. The processor calculates T_{ij}. T_{ij} is the "target" intensity value in the neighbourhood of P_{ij}. Section 4.1 describes the possible definitions of T_{ij}.
2. The processor generates the initial value of C_{ij}. It generates a random real r, $0 \leq r \leq 1$, and sets C_{ij} to white if $r < \mathsf{G}_{ij}$ and to black otherwise.
3. The processor calculates the initial value of E_{ij}.
4. The processors iterate through the following steps synchronously until termination.
   (a) If $\mathsf{E}_{ij} \gg \mathsf{T}_{ij}$, the processor randomly chooses some pixels in W_{ij} and sends them messages requesting that they turn black. Otherwise, if $\mathsf{E}_{ij} \ll \mathsf{T}_{ij}$, the processor randomly chooses some pixels in B_{ij} and sends them messages requesting that they turn white. Otherwise the processor is "happy" and communicates this to the other processors.
   (b) If enough processors are happy, the algorithm terminates and the processor returns the value C_{ij} as its result. Otherwise, if the processor receives enough messages requesting a change of colour, it changes the value of C_{ij} and sends a message to each pixel in N_{ij} informing it of the change[2].
   (c) The processor updates the value of E_{ij} with the information from incoming messages.

Algorithm Υ involves communication at every step except Step 2 and Step 4(c). At Steps 1, 3 and 4(b), the processor sends a message to each pixel in N_{ij}. Assuming that the definition of N_{ij} does not vary during one execution of the algorithm, the pattern of these communications is determined statically. At Step 4(a) the processor sends messages to some pixels randomly chosen from N_{ij}. Randomness is important here as it ensures that systematic patterns of noise do not arise in the binary image. The destinations of these messages cannot therefore be known in advance: they are determined dynamically by the random-number generator.

The three issues which are not specified in the above description of Algorithm Υ are the definition of T_{ij} and how the processors determine whether they are "happy", how unhappy processors communicate their desire for change to one another and how happy processors indicate their happiness and co-operate to detect termination.

4.1 The definition of T_{ij} and the pursuit of happiness

T_{ij} is the "target" intensity value in the neighbourhood of P_{ij}. There are at least three possible definitions of T_{ij}.

1. $\mathsf{T}_{ij} = N \times \mathsf{G}_{ij}$
2. $\mathsf{T}_{ij} = \sum_{\mathsf{P}_{i'j'} \in \mathsf{N}_{ij}} \mathsf{G}_{i'j'}$
3. $\mathsf{T}_{ij} = \sum_{\mathsf{P}_{i'j'} \in \mathsf{N}_{ij}} f(i,i',j,j') \times \mathsf{G}_{i'j'}$

The first definition makes T_{ij} depend only on the value of G_{ij}, i.e. the intensity in the centre of the neighbourhood. The second definition makes T_{ij} depend on all of the values $\mathsf{G}_{i'j'}$ in the neighbourhood. This definition tends to make the final binary image slightly more "blurred" than does the first definition. The third possibility is a generalisation of the second, where T_{ij} is defined to be a weighted sum of the values $\mathsf{G}_{i'j'}$ in the neighbourhood, the weighting being specified by the function f. We have conducted no experiments with this definition of T_{ij} as yet.

A processor determines whether it is happy by comparing the values of E_{ij} and T_{ij}. The strictest definition of happiness is

$$\mathsf{E}_{ij} \gg \mathsf{T}_{ij} \quad \text{iff} \quad \mathsf{E}_{ij} - \mathsf{T}_{ij} > 0.5$$

$$\mathsf{E}_{ij} \ll \mathsf{T}_{ij} \quad \text{iff} \quad \mathsf{T}_{ij} - \mathsf{E}_{ij} > 0.5$$

$$\mathsf{Happy}(\ \mathsf{P}_{ij}\ ) \quad \text{iff} \quad \mathsf{E}_{ij} \not\gg \mathsf{T}_{ij} \wedge \mathsf{E}_{ij} \not\ll \mathsf{T}_{ij}$$

Thus P_{ij} is happy if E_{ij} is within 0.5 of T_{ij}, i.e. if E_{ij} is the best possible estimate of T_{ij}.

There is a further enhancement available using this definition of happiness. From the initial value of T_{ij}, the processor can uniquely determine the integer number of pixels for which it is aiming. For example, if the initial value of $\mathsf{T}_{ij} = 4.3$, the processor will be happy only if $\mathsf{E}_{ij} = 4$. The processor can therefore round the value of T_{ij} to the nearest integer at Step 1 of the algorithm. The principal effect of this rounding is to make the values C_{ij} converge more quickly.

[2] We assume that $\mathsf{P}_{ij} \in \mathsf{N}_{i'j'} \Rightarrow \mathsf{P}_{i'j'} \in \mathsf{N}_{ij}$.

However, this definition of happiness may in fact be too deterministic. The effect of such a strict definition is similar in some ways to thresholding over a neighbourhood instead of over a single pixel: areas in the greyscale image with continuously varying intensity tend to be divided up in the binary image into distinct regions with integer intensities.

We overcome this problem by relaxing the definition of happiness somewhat. Each processor determines the initial value of T_{ij} according to whichever of the above definitions is in use. It then uses the fractional part of T_{ij} to decide randomly whether to round T_{ij} down or up to the nearest integer. For example, if the initial value of T_{ij} is 4.3, the processor generates a random real r, $0 \leq r \leq 1$, and sets T_{ij} to 5 if $r < 0.3$ and to 4 otherwise. This technique gives the advantage of good convergence which arises from the use of integer targets, whilst avoiding the disadvantage of neighbourhood thresholding which arises from a totally deterministic definition of happiness.

So, to recap, the definition of T_{ij}, $\gg$, $\ll$ and Happy which we have (so far) found which give the best results in Algorithm Υ is

$$\begin{aligned} \mathsf{T}_{ij} &= \lceil\, \mathsf{t}_{ij} \,\rceil, \text{ iff } r < \mathsf{t}_{ij} - \lfloor\, \mathsf{t}_{ij} \,\rfloor \\ &= \lfloor\, \mathsf{t}_{ij} \,\rfloor, \text{ otherwise} \\ &\quad \text{where } \mathsf{t}_{ij} = N \times \mathsf{G}_{ij} \end{aligned}$$

$$\mathsf{E}_{ij} \gg \mathsf{T}_{ij} \quad \text{iff} \quad \mathsf{E}_{ij} > \mathsf{T}_{ij}$$

$$\mathsf{E}_{ij} \ll \mathsf{T}_{ij} \quad \text{iff} \quad \mathsf{E}_{ij} < \mathsf{T}_{ij}$$

$$\mathsf{Happy}(\,\mathsf{P}_{ij}\,) \quad \text{iff} \quad \mathsf{E}_{ij} = \mathsf{T}_{ij}$$

where r is a random real with $0 \leq r \leq 1$.

4.2 Communication for change

An unhappy processor tries to increase its own happiness by convincing some processors in its neighbourhood to change their pixel colours. If it convinces too few of its neighbours, it will take too long to achieve happiness. If it convinces too many, it will remain unhappy, its intensity alternating between being too high and being too low. Striking a balance between these extremes is therefore critical to the correct functioning of the algorithm.

We have constructed a simple voting scheme that balances the (potentially) conflicting desires of processors whose neighbourhoods overlap. In this scheme, a processor changes its pixel's colour if a majority of its neighbours send messages requesting it to change. Thus global happiness will increase.

There are two cases to consider where processors are unhappy. The first case is where $\mathsf{E}_{ij} \gg \mathsf{T}_{ij}$, i.e. the processor considers its neighbourhood to be too white. The second case is where $\mathsf{E}_{ij} \ll \mathsf{T}_{ij}$, i.e. the processor considers its neighbourhood to be too black.

Case 1: $\mathsf{E}_{ij} \gg \mathsf{T}_{ij}$

This is the case where there are too many white pixels in P_{ij}'s neighbourhood. P_{ij} calculates a probability, p, and sends a message requesting change to each processor in W_{ij} with probability p.

Assuming that every neighbour of the receiving processor chooses the same p, we can calculate a value for p so that the expected number of neighbours that will change is neither too high nor too low. Although this assumption is not strictly valid, it has the virtue that the calculation can be done locally so that there is no communication overhead. The calculated value of p works well in practice.

The calculation is as follows. The unhappy processor wants $\mathsf{E}_{ij} - \mathsf{T}_{ij}$ neighbours to turn black on average, i.e. it wants

$$\begin{aligned} &Prob(\text{White neighbour } \mathsf{P}_{i'j'} \text{ turns black}) \\ &= \frac{\mathsf{E}_{ij} - \mathsf{T}_{ij}}{\mathsf{E}_{ij}} \end{aligned} \tag{1}$$

But, by supposition,

$$\begin{aligned} &Prob(\text{White neighbour } \mathsf{P}_{i'j'} \text{ turns black}) \\ &= Prob(\mathsf{P}_{i'j'} \text{ receives} > \tfrac{N}{2} \text{ requests}) \end{aligned} \tag{2}$$

We can calculate

$$\begin{aligned} &Prob(\mathsf{P}_{i'j'} \text{ receives } k \text{ requests}) \\ &= \binom{N}{k} \times p^k \times (1-p)^{N-k} \end{aligned} \tag{3}$$

so, combining equations (2) and (3), we get

$$\begin{aligned} &Prob(\text{White neighbour } \mathsf{P}_{i'j'} \text{ turns black}) \\ &= \sum_{k=\frac{N}{2}+1}^{N} \binom{N}{k} \times p^k \times (1-p)^{N-k} \\ &= 1 - \sum_{k=0}^{\frac{N}{2}} \binom{N}{k} \times p^k \times (1-p)^{N-k} \\ &= 1 - B(p; \frac{N}{2}, N) \end{aligned} \tag{4}$$

where $B(p; k, n)$ is the cumulative density function of the binomial distribution. Thus we see from equations (1) and (4) that P_{ij} should choose p such that

$$\begin{aligned} B(p; \frac{N}{2}, N) &= 1 - \frac{\mathsf{E}_{ij} - \mathsf{T}_{ij}}{\mathsf{E}_{ij}} \\ &= \frac{\mathsf{T}_{ij}}{\mathsf{E}_{ij}} \end{aligned} \tag{5}$$

Case 2: $\mathsf{E}_{ij} \ll \mathsf{T}_{ij}$

This is the case where there are too many black pixels in P_{ij}'s neighbourhood. The calculation is similar to that of Case 1: P_{ij} should choose p such that

$$B(p; \frac{N}{2}, N) = \frac{N - \mathsf{T}_{ij}}{N - \mathsf{E}_{ij}} \tag{6}$$

4.3 The detection of termination

Algorithm Υ iterates until "enough processors" are happy with the state of their neighbourhood. This statement leaves open two questions: how does a processor indicate its happiness to other processors, and how many processors is "enough".

4.3.1 Communication for termination

Algorithm Υ, as described so far, is an extremely well-distributed algorithm: indeed, that is one of its principal strengths. All communication is between processors in the same neighbourhood: as a consequence, the execution time of the algorithm is independent of the number of processors participating (given one processor per pixel).

The detection of termination, however, clearly requires a global decision about the state of the processors. On a machine with a combining network

[Gottlieb et al., 1983], this would present no problem: the processors could concurrently increment a global variable H if they were happy, and the happiness level could be determined by examining the value of H. On a machine supporting combining operations, the value of H could be accumulated by this simple technique in time independent of the number of processors.

On a machine without combining, the fastest way to calculate the value of H is to connect the processors into a tree structure and accumulate the values from the processors at the leaves up to the "central processor" at the root of the tree. This accumulates the value of H in $O(\log n)$ time. The central processor then broadcasts its decision via the same route.

The detection of termination is clearly the most expensive aspect of Algorithm Υ, in terms of complexity. It is therefore beneficial to execute several iterations of the main algorithm between calculating values of H. The actual number of iterations will depend on the number of processors participating in the execution. It may also be varied dynamically: as more processors become happy with their state, it may be beneficial to test for global happiness more often.

Note, finally, that in a sequential implementation of Algorithm Υ, the simple technique of accumulating a global value H will suffice.

4.3.2 How much happiness is enough?

In general, it will not be possible for all processors to be happy. A simple termination criterion is to stop if the value of H remains unchanged for several iterations. In practice, we have found that such a stage is always reached. However, a safer option is to stop if the value of H fails to improve by some small, predetermined amount over several iterations. Although this may result in premature stopping in some cases, in practice true convergence may require several hundred further iterations, and it is all but impossible to tell the difference in the final images.

5 Results

In this section we present some examples showing the binary images obtained by applying Algorithms Υ and HT and the Floyd-Steinberg algorithm to two test images. We have chosen a natural image, a portrait of Albert Einstein, and a synthetic image, a circle shaded with smoothly varying intensities, to highlight the strengths and weaknesses of the three algorithms. For Algorithms Υ and HT, 3×3 neighbourhoods were used and the intensity for each neighbourhood was calculated using a simple (unweighted) average.

5.1 Circle

This image is 256×256 pixels, containing a circle with intensity smoothly varying from 0.0 to 1.0 around the circle. The background intensity is set at 0.5.

Figure 1 shows that the Floyd-Steinberg algorithm produces distinct distracting regions shaded with regular textured patterns. The shape of the circle is entirely lost near the right hand edge of the image. Algorithm HT does much better, but exhibits a faint but noticeable artifact—there are definite jumps in intensity at several points around the circle. This is most noticeable on the left of the circle, in the areas of maximum and minimum intensity. Algorithm HT renders these as solid white and solid black respectively. This is because the best approximation to the proper intensity for each individual neighbourhood is indeed a solid colour. However, the global effect of these locally correct decisions is wrong, and the human visual system is sensitive to this error.

Algorithm Υ is free from such artifacts and the slightly grainy overall effect is not unpleasant.

5.2 Einstein

This image is 487×345 pixels, containing a 5 bit greyscale picture of Albert Einstein. Figure 2 shows the original image. Note that the dithering process used to print Figure 2 is from a different class to the halftoning algorithms that we are considering here: the greyscale and binary pixel arrays have different dimensions.

Figure 3 shows that the Floyd-Steinberg algorithm produces a result with some texturing evident, and a rather unnatural, patchy facial appearance. Figures 4 and 5 show that Algorithms HT and Υ produce more natural results, with Algorithm HT perhaps having an edge in preserving detail (for example in the pattern on the chair on which Professor Einstein is seated). This extra detail is the tradeoff for the aliasing problems seen in the circle image.

5.3 Combining HT and Υ

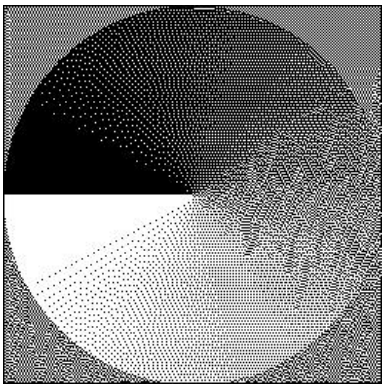
Algorithm Υ progresses ever more slowly as it nears convergence. Typically, one thousand iterations are required for full convergence. This would still be very fast in a fully parallel implementation, but in a serial implementation it is slower than other algorithms (except, we expect, the neural network or simulated annealing approaches). By contrast, Algorithm HT reliably converges in about a dozen iterations.

As well as the aliasing problem mentioned earlier, Algorithm HT has the problem that it can only find an image that represents a local minimum of its cost function. It cannot find a solution with a lower cost if finding that solution requires changing more than one pixel at one step. Algorithm Υ, however, is not so restricted, and can even sometimes take a step that temporarily produces an inferior image, allowing a more thorough coverage of the search space. This ability arises from the use of randomness in the search, as in simulated annealing.

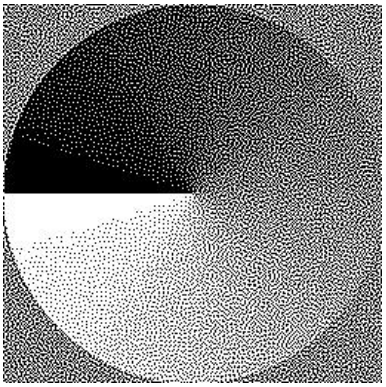
In this study, we have not necessarily been concerned to produce the best possible quality picture, but only to demonstrate the feasibility of the DCA approach to solve a computer graphics problem that is inherently a large-scale parallel problem. Nonetheless, we were led to wonder whether we could combine the fast convergence of Algorithm HT with the better searching of Algorithm Υ.

The images in Figure 6 were produced by a program combining Algorithm HT and Algorithm Υ. The program ran Algorithm HT to convergence, then, using the resulting image as the initial image, ran algorithm Υ for several iterations. The resulting image was then used as the initial image for Algorithm HT, and this cycle was repeated until no further lowering of the value of the cost function was obtained.

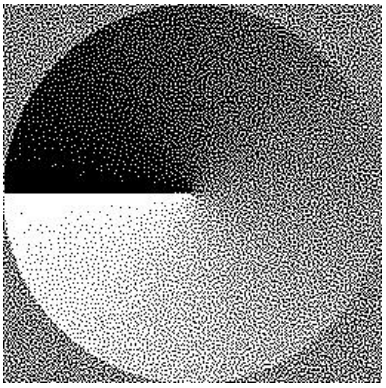
This reduced the cost for the circle from 2284 to 1950 (15% lower), and for the Einstein picture from 11471 to 9380 (10% lower). Notice in Figure 6(a) that the aliasing problem is accentuated, as we would expect in a lower cost solution.



1(a): Floyd-Steinberg



1(b): HT



1(c): Υ

Figure 1: The continuously-varying circle

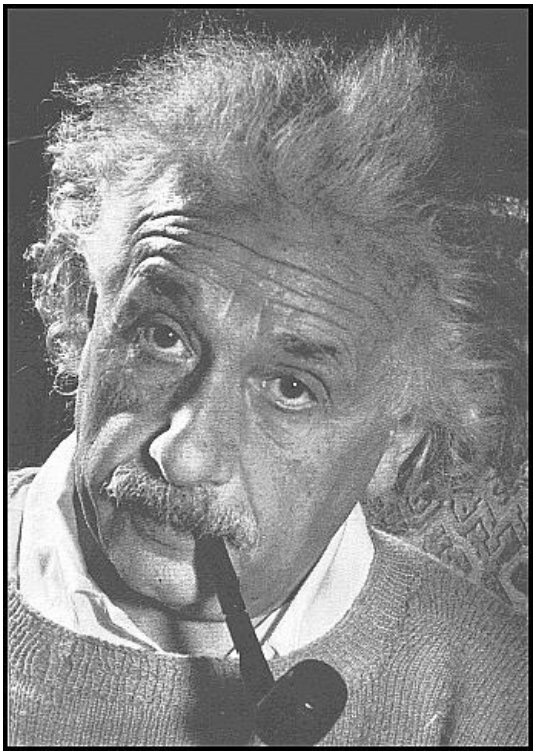


Figure 2: Albert Einstein original

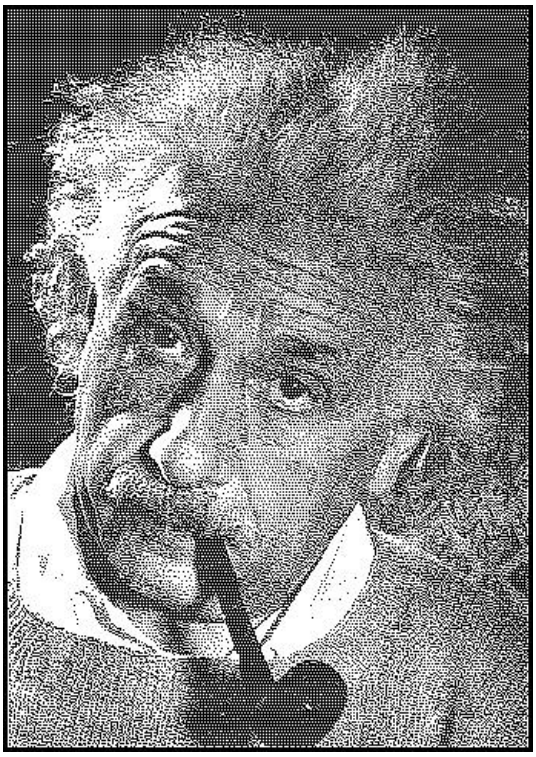


Figure 3: Einstein with Floyd-Steinberg

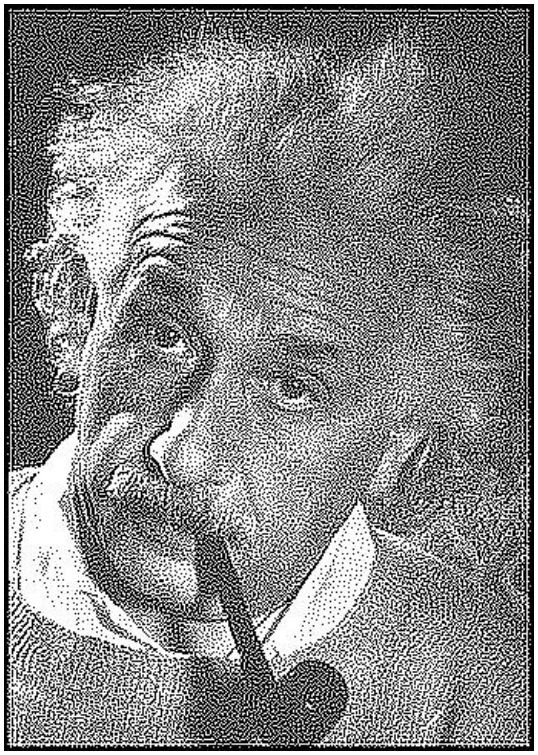


Figure 4: Einstein with HT

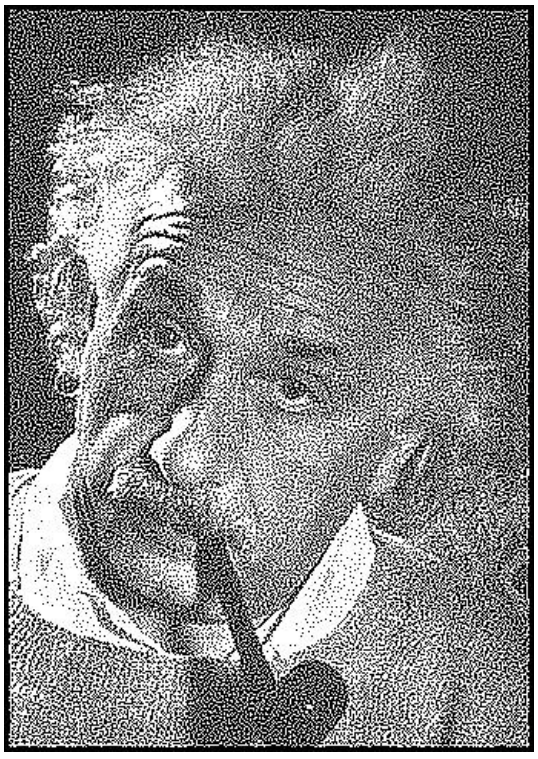
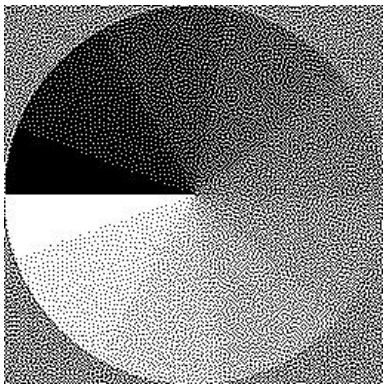
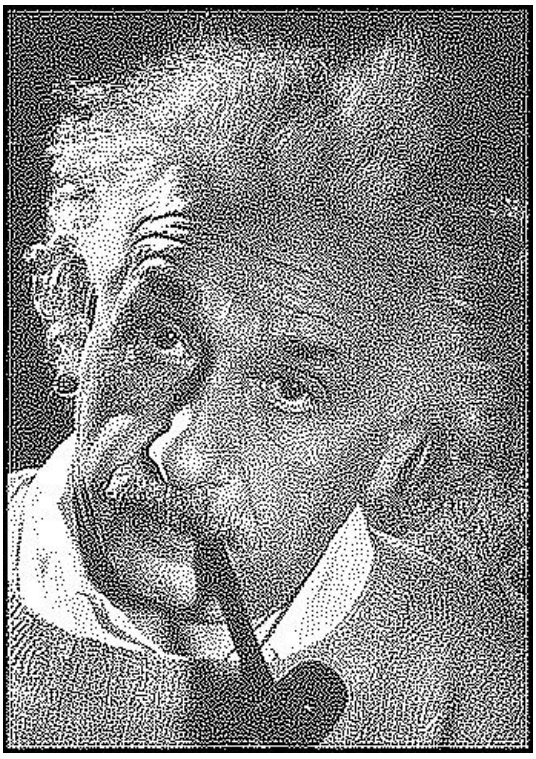


Figure 5: Einstein with Υ



6(a): circle



6(b): Einstein

Figure 6: Images from the combined algorithm

6 Conclusions

We have described a new parallel algorithm for digital halftoning. Algorithm Υ is based on the technique of error diffusion, whereby a pixel which is not happy with the state of its neighbourhood requests one or more of its neighbours to change colour to increase its happiness level. The algorithm also uses a cost function to determine when it should terminate: a simple count of the number of pixels which are happy.

The figures in Section 5 show that Algorithm Υ is competitive with the best of previous halftoning algorithms. The images produced by Algorithm Υ are free from the texturing and patterning produced by ordered dither, error diffusion and dot diffusion methods. Algorithm Υ also avoids the thresholding problem exhibited by Algorithm HT. Being an iterative algorithm, it should share Algorithm HT's advantages for animation. The algorithm is highly parallel and uses randomness to avoid becoming stuck at solutions that are only locally optimal.

Algorithm Υ is a member of a novel class of algorithms called "dynamic communication algorithms". Algorithms in this class differ from other parallel algorithms in that the pattern of their communication is determined at run-time using data local to the source processor. This permits the constructive use of communication to reduce the computational requirements of algorithms. Dynamic communication algorithms have previously been discovered in a diverse range of important application areas, including sorting, tessellation of the plane, fractal image generation, pattern recognition and pitch detection in speech. We expect that such algorithms will increase in importance as the use of massively-parallel machines based on message-passing becomes more widespread.

Acknowledgement

We would like to acknowledge the support received from the Digital Equipment Corporation through their External Research Programme.

References

[Bayer, 1973] BAYER, B.E. (1973). *An Optimum Method for Two-level Rendition of Continuous-tone Pictures.* IEEE International Conference on Communication, New York, 26-11–26-15.

[Cole, 1991] COLE, A.J. (1991). *Halftoning without Dither or Edge Enhancement.* The Visual Computer 7:232–46.

[Floyd and Steinberg, 1976] FLOYD, R. AND STEINBERG, L. (1976). *An Adaptive Algorithm for Spacial Greyscale.* SID 17:75–7.

[Geist et al., 1993] GEIST, R., REYNOLDS, R. AND SUGGS, D. (1993). *A Markovian Framework for Digital Halftoning.* Transactions on Graphics 12(2):136–59.

[Gotsman, 1993] GOTSMAN, C. (1993). *Halftoning of Image Sequences.* The Visual Computer 9:255–66.

[Gottlieb et al., 1983] GOTTLIEB, A., LUBACHEVSKY, B.D. AND RUDOLPH, L. (1983). *Basic Techniques for the Efficient Co-ordination of Very Large Numbers of Co-operating Sequential Processors*, ACM Transactions on Programming Languages and Systems, Vol. 5, No. 2, pp. 164–89.

[Jarvis et al., 1976] JARVIS, J., JUDICE, C. AND NINKE, W. (1976). *A Survey of Techniques for the Display of Continuous-tone Pictures on Bi-level Displays.* Computer Graphics and Image Processing 5:13–40.

[Knuth, 1987] KNUTH, D.E. (1987). *Digital Halftones by Dot Diffusion.* ACM Transactions on Graphics 6(4), 245–73.

[Sharp, 1990] SHARP, D.W.N. (1990). *Functional Language Program Transformation for Parallel Computer Architectures.* University of London doctoral thesis.

[Sharp and Cripps, 1989] SHARP, D.W.N. AND CRIPPS, M.D. (1989). *A Parallel Implementation Strategy for Quicksort.* International Symposium on Computer Architecture and Digital Signal Processing, Hong Kong, Vol. 1.

[Sharp and Cripps, 1991] SHARP, D.W.N. AND CRIPPS, M.D. (1991). *Parallel Algorithms that Solve Problems by Communication.* 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, 87–94.

[Sharp and While, 1993a] SHARP, D.W.N. AND WHILE, R.L. (1993). *Determining the Pitch Period of Speech using no Multiplications.* 18^{th} International Conference on Acoustics, Speech and Signal Processing, Minneapolis, Minnesota.

[Sharp and While, 1993b] SHARP, D.W.N. AND WHILE, R.L. (1993). *Pattern Recognition using Fractals.* 22^{nd} International Conference on Parallel Processing, St. Charles, Illinois, III:82–9.

[Velho and Gomes, 1991] VELHO, L. AND GOMES, J. (1991). *Digital Halftoning with Space-filling Curves.* Computer Graphics 25(4):81–90.

[Witten and Neal, 1982] WITTEN, I.H. AND NEAL, R.M. (1982). *Using Peano Curves for Bi-level Display of Continuous-tone Images.* IEEE Computer Graphics Applications 2(3):47–52.

[Wyvill and McNaughton, 1991] WYVILL, G. AND MCNAUGHTON, C. (1991). *Three Plus Five Makes Eight: a Simplified Approach to Halftoning.* Computer Graphics International, New York, Springer Verlag, 379–92.

[Zhang and Webber, 1993] ZHANG, Y. AND WEBBER, R. (1993). *Space Diffusion: an Improved Parallel Halftoning Technique Using Space-filling Curves.* Proceedings of SIGGRAPH 93, Anaheim, California. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, New York, 305–12.