VARIATIONS ON A DITHER ALGORITHM

Markus PINS and Hermann HILD

Institut für Betriebs- und Dialogsysteme
University of Karlsruhe
Karlsruhe, West-Germany

Mapping continuous-tone pictures into digital halftone pictures, i.e. 0/1-pictures, for printing purposes is a well explored technique. In this paper, one of these algorithms, the two-dimensional error-diffusion algorithm is extended to color pictures and animated pictures. The color picture algorithm is superior to existing algorithms by considering extreme color values as well as adjacent color values. The animation algorithm eliminates the noise created by the correct but varying pixel patterns generated by applying a single picture dithering algorithm on every frame. The power of the algorithms is demonstrated by experiments carried out on synthetic images generated by ray tracing.

1 INTRODUCTION

Dither algorithms are required to display continuous-tone pictures on graphic devices with less gray levels or color levels than contained in the original picture. They are particulary helpful for obtaining a high quality display on devices using a color table that permits the choice of colors from a large palette. For example, modern workstations have frame buffers with 8 planes per pixel, allowing the reproduction of 256 out of up to more than 16 million colors. Usual rendering and shading algorithms of e.g. 3-d mechanical CAD-systems make only poor use of these abilities. Rendering with more colors and then applying a dither algorithm yields considerably better results.
There are several algorithms for dithering pictures, e.g. *Dithering with a dither matrix* [ES86], the *Floyd-Steinberg Algorithm* [FS75], *Constant-Level-Thresholding and Two-dimensional-error-diffusion* [Stu82] or the *Dot Diffusion Algorithm* described by [Knu87]. Comparing these algorithms, the *two-dimensional error diffusion dither algorithm* described by [Stu82] yields very good results and turns out to be easy implementable. A sketch of this algorithm is given in section 2. Although originally developed for black-and-white dithering, it is adaptable to several situations where dithering is required. In section 3 of this paper, an adaption of the *two-dimensional error diffusion algorithm* for color images is given. Dithering of color images usually consists of two phases. In the first phase, a suitable color table is chosen. There are several approaches known for this step, e.g. popularity algorithm [Hec82], the median cut algorithm [Hec82], or octree quantization [GP88]. A first weakness of these algorithms is that colors occurring rarely in the original images might be eliminated in the dithered image. A second weakness is that adjacent colors that might be useful when modeling continuous changes between color levels might not be available in the color table. These two weaknesses are eliminated by the color table algorithm of section 3.1. Section 3.2 presents the color version of the two-dimensional error diffusion algorithm, and examples of its applications with the optimized color table.
Todays workstations are capable to display a sequence of several dozens of bitmapped pictures of a size of, say, 200×200, at a rate of 16 images or more per second. This is sufficient to give the observer the impression of continuous motion. If the movie is initially given in graytone, an immediate way of dithering is to use one of the many existing dither algorithms to map each graytone picture independently into the corresponding binary picture. However, an undesirable side effect may occur in the form of a noisy flickering. This is due to the fact that similar areas in successive graytone pictures may be mapped into pixel-patterns in the binary picture which are not similar at all. Thus

every single binary picture has patterns which correctly represent the corresponding gray values in the graytone pictures, but in the moving sequence these patterns may completely change over time, causing remarkable noise.
Unfortunately this effect cannot be demonstrated without a bitmap display, nor can one visually figure out changing patterns in subsequent pictures. We denote the difference-picture of two binary pictures $B1$ and $B2$ as the picture in which a pixel is set if and only if the values of the pixels at the two corresponding positions in $B1$ and $B2$ differ. Difference-pictures allow some conclusions on the behavior of the pictures: the more set pixels in the difference-picture, the more change and therefore flickering will be in the sequence. Of course changing pixels disturb more or less, depending on their context in the picture. Changing pixels do not at all disturb areas where genuine changes in the graytone pictures take place. Figure 1 shows two successive pictures of a raytraced picture sequence. The only moving parts in the sequence are the wings and their shadows. The background and the tower remain the same patterns over the whole sequence.

Figure 1: Subsequent pictures from the sequence *windmill*

.

Each of the pictures is correctly dithered. The different pixel-patterns which cause the flickering cannot be recognized by a visual comparison.

Nevertheless the patterns in the background are different in each binary picture, as indicated by the difference-picture in figure 2.
Up to now, there seems to be no solution to this problem in literature. In section 4, an adaption of the two-dimensional error diffusion algorithm is given, reducing the problem of flickering. This adaption can be seen as a 3-d extension of error diffusion.

2 DITHERING WITH CONSTANT-LEVEL THRESHOLDING AND TWO-DIMENSIONAL ERROR-DIFFUSION

Dithering with *Constant-Level Thresholding and Two-dimensional error-diffusion* as described by [Stu82] is an extension of the *Floyd-Steinberg-Algorithm* [FS75]. At every position (i, j) of a picture P, a carried error value resulting from the weighted average of previously computed errors is added. The range of picture elements comprises the values $[Graylevel_{min}, Graylevel_{max}]$. The errorcarry is computed by a weighted errorfilter. The weight coefficients are chosen to be 2^n to improve computational efficiency. A point in the bitmap is inserted if the graylevel value in the picture P at

Figure 2: The resulting difference-picture

It demonstrates that there are no major areas which have the same pixel-patterns.

position (i,j) including $ErrorCarry(i,j)$ is less or equal the threshold $\frac{Graylevel_{max} - Graylevel_{min}}{2}$, i.e.

$$Bitmap(i,j) = \begin{cases} 0 & if\ P(i,j) + ErrorCarry(i,j) > \frac{Graylevel_{max} - Graylevel_{min}}{2} \\ 1 & if\ P(i,j) + ErrorCarry(i,j) \leq \frac{Graylevel_{max} - Graylevel_{min}}{2}. \end{cases}$$

The value $ErrorCarry$ results from the weighted average of previously computed errors,

$$ErrorCarry(i,j) = \frac{\sum_{(k,l) \in Area} Error(i+k, j+l) * Weight(k,l)}{\sum_{(k,l) \in Area} Weight(k,l)}.$$

The weight function *Weight* and the environment *Area* of summation are defined by

```
          Weight                                     Area

 .    .    .    .    .    .    .
 .    1    2    4    2    1    .          { (-2, -2), (-2, -1), ...  (-2, 2),
 .    2    4    8    4    2    .            (-1, -2), (-1, -1), ...  (-1, 2),
 .    4    8  (i,j)  .    .    .            ( 0, -2), ( 0, -1) }
 .    .    .    .    .    .    .
```

The values of *ErrorCarry* depend on the function *Error*. *Error* itself depends on whether a point in the *Bitmap* has been inserted or not.
At position (i,j), we want to approximate the graylevel value $P(i,j) + ErrorCarry(i,j)$. If a point at this position is inserted in the *Bitmap* and $P(i,j) + ErrorCarry(i,j) > Graylevel_{min}$, the dithered picture is too dark by the amount of $P(i,j) + ErrorCarry(i,j) - Graylevel_{min}$. If the point at position (i,j) is not inserted, the dithered picture is too bright by the amount of $P(i,j) + ErrorCarry(i,j) - Graylevel_{max}$. Hence

$$Error(i,j) = \begin{cases} P(i,j) + ErrorCarry(i,j) - Graylevel_{min} & ,if\ Bitmap(i,j) = 0 \\ P(i,j) + ErrorCarry(i,j) - Graylevel_{max} & ,if\ Bitmap(i,j) = 1. \end{cases}$$

When dithering images with homogeneous background it is recommendable to superpose the image with a random noise function to avoid regular patterns. The random noise can be obtained by using a random number generator, creating random numbers in the range from $-0.05 * \frac{Graylevel_{max} - Graylevel_{min}}{2}$ to $+0.05 * \frac{Graylevel_{max} - Graylevel_{min}}{2}$. Generating numbers with a bigger amplitude leads to falsifications of the image. These random numbers are added to the image values, i.e.

$$Bitmap(i,j) = \begin{cases} 0 & if\ P(i,j) + ErrorCarry(i,j) + noise > \frac{Graylevel_{max} - Graylevel_{min}}{2} \\ 1 & if\ P(i,j) + ErrorCarry(i,j) + noise \leq \frac{Graylevel_{max} - Graylevel_{min}}{2}. \end{cases}$$

3 DITHERING OF COLOR IMAGES

Color images are $N * M$ rectangular arrays of pixel, separated in a red, green and blue color value for each pixel. The color components are usually represented by 8 bits, which means numbers in the range $[0, 255]$ or 256 different graylevel values. As explained in the introduction, color reduction consists of two phases: choosing the color table (i.e. quantization), and replacing the colors in the picture by the values in the color table (i.e. dithering).

3.1 Quantization

First, a color table C with K different colors must be chosen. (K is the number of colors, the frame buffer can reproduce. We use $K = 256$).
There are different possibilities to choose a color table. The *Standard Algorithm* is to subdivide the RGB-color-space by a grid structure into boxes, and fill the color table with the representatives of each box. Because the human eye cannot distinguish the blue color levels as well as the red or green color levels, the red and green axis will be divided into eight pieces, the blue one into four pieces. Now $8 * 8 * 4 = 256$ colors are available. The quality of the dithered picture is normaly low. The *Popularity Algorithm* chooses a color table by evaluating the K most frequently occurring colors in the image. However, when dithering pictures with a lot of different colors, the algorithm delivers no satisfying results. A recent method is the *Octree Quantization* [GP88]. The first K different colors of the image are used as initial entries to the color table. Reading the image sequentially, at each new color not contained in the color table two very near neighbours of the now existing $K+1$ colors are merged and replaced by the weighted mean. This step is repeated for every new color, until the image is completely processed.
Our approach is based on the *Median Cut Algorithm* [Hec82] which subdivides the RGB-color space into K rectangular boxes. At every step, the box with the most entries is divided along its longest axis into two new boxes, i.e.

```
Shrink_RGB_Colorspace;
For i = 1 to K do
  Split_Box_with_most_Colorentries;
  Shrink_both_boxes
od.
```

Procedure *Shrink* computes the minimal rectangular box q' to a given box q, which includes all entries of q. *Split* divides a box along the longest dimension such that each box contains approximately the same number of entries.
The algorithm starts with the full color space and terminates after K steps. The K colors are selected in such a way, that every color represents approximately the same number of pixels in the image. The representative $q.c$ of the box q is computed as weighted average of all color entries in q. q contains the colors $q.c_i$ occuring $q.s_i$ times, $i = 1 \cdots n$. n is the number of different colors in q.

$$q.s = \sum_{i=1}^{n} q.s_i, \qquad q.c = \frac{1}{q.s} * \sum_{i=1}^{n} q.s_i * q.c_i$$

The *Median Cut Algorithm* is a definite improvement over the *Popularity Algorithm*, however the color selection can be improved. First, extremal colors, occurring rarely and lying far away from the color table entries are not considered by the algorithm. Thus colors must be chosen from a color table, having a big distance to the original color value. Another weakness is that only one color is possibly chosen from a cluster of colors. For continuous gradations of colors it might happen that representative colors are chosen lying far away from the cluster. This is unsatisfactory, since the human eye is very sensitive to small gradations.
The idea of the following is to compute a bigger color table with K' colors, ($K' > K$) and select the final color table entries out of these colors, prefering the *extreme* and the *clustered* color values.

Prefering extreme color values

First, K' colors must be computed, for example with the *Median-Cut-Algorithm*. ($K' := 2 * K$ produces good results). These K' colors are inserted into a list l which contains the color values $q_i.c$ and the cumulated occurrences $q_i.s$ for every box $q_i, i = 1 \cdots K'$. In addition the distance criterion p will be introduced as an element of l. p is initialized with the number of entries $q_i.s$ in q_i. After building up the list l, the color value c with the biggest criterion value p is inserted into the color table and erased in l. The other criterion values of all elements in l are multiplied by the distance-factor $A = \frac{r}{1+r}$. r is the distance between chosen color c and the current color in the list l. Doing this the criterion value for colors lying near c will be reduced more than the value for colors far away from c. Because the distances to the chosen color are all multiplied by every p, that color will be chosen next which guarantees the best relation between occurrences and distance to all other chosen colors. This step will be repeated K times until all colors of the color table are computed.

```
for i = 1 to K do
  c = Choose_Color_from_List_with_maximal_p;
  ColorTable[i] := c;
  (* Erase c in List l of intermediate Colors and decrement K'  *)
  l = l \ c;
  K' = K' - 1;
  (* Compute distance d between Colors c and l[j].c and multiply
     l[j].p with A = r / (1 + r)                                *)
  for j = 1 to K' do
    r = d(c, l[j].c);
    A = r / (1 + r)
    l[j].p = l[j].p * A
  od
od
```

Prefering colors in clusters

A different possibility is to merge two color values lying close to each other to a new color value. Merging is weighted according to the occurrences of both color values.
The algorithm starts again with the *Median-Cut-Algorithm* which computes K' colors and inserts them in the list l. In addition, the flag *merge* initialized with FALSE is introduced as element of l.

```
repeat
  (* Choose the two Colors l[i].c and l[j].c out of l which
     have minimal distance dist and both are not merged yet    *)
  choose_nearest_not_merged_colors(dist, i, j);
  (* If the distance is lower than D, the colors i and j will
     be merged, l will be updated and the number of colors K'
     will be decremented                                        *)
  if (dist <= D)
    new_color.p = l[i].p + l[j].p;
    new_color.c = (l[i].p * l[i].c + l[j].p * l[j].c) / new_color.p;
    new_color.merge = TRUE;
    eliminate_from_list(i, j);
    add_to_list(new_color);
    K' = K' - 1
   fi
  until (K' <= K) or (dist > D);
```

This algorithm has the property that already merged color values may not be merged again. The constant D in the conditional statement denoting the maximal distance between to different color values can be determined freely. Experiments show that $5 \leq D \leq 10$ produces best results. Using this improvement the probability increases that color gradations in images have a smoother appearence.

Correcting the brightness of the image

Psychophysical research has shown that the perceptual experiences of the human eye are nonlinearly to the physical event. This means that the human eye perceives a linear increase in brightness as nonlinear, shown in figure 3 [Mur86].

Figure 3: Perceived brightness as a function of light intensity

When considering this effect evaluating the distance function d produces better results. Light intensity of color values are computed by converting the RGB-modell into the HLS-Color-Modell [ES86]. It consists of three components *Hue*, *Lightness* and *Saturation* (figure 4). *Hue* is an angle between 0 and 360 degree, *Lightness* is the height in the HLS-color-cone along the axis, normalized to the range $[0, 1]$ and *Saturation* is the distance between the color value and the axis.
Taking into consideration the human eyes perception sensitivity leads to the $HL'S$-modell derived from HLS, with

$$L' = m(L), \qquad \forall x \in L: \quad m(x) = -0.6x^2 + 1.6x$$

3.2 Dithering

The key operation of dithering is finding a closest color in the color table for a given color in the picture. A symmetrical grid with cell length d_{box} is used, enclosing the complete color space. Each grid cell contains a list of representatives which are potential nearest neighbors. A potential neighbor

Figure 4: HLS-color-cone

is a color value out of the color table if the distance to the middle of the grid cell g_{box} is smaller than d_{col},

$$d_{col} = d_{box} + \min_{\forall c \in C}(d(c, g_{box}))$$

In practice one should not evaluate the entries for all grid cells, since a lot of them are not used. It is better to compute only the needed cells and their representatives [Hec82]. Instead of a grid structure one can use a more flexible space dividing structure, for example an *Octree* [JT80] or a *Grid File* [HNS84].

Based on this closest color procedure, the modified constant level thresholding and two-dimensional error diffusion algorithm works as follows. At each point $P_{red,green,blue}(i,j)$, the corrected new value $P'_{red,green,blue}(i,j)$, representing the sum from chosen color value, errorcarry, and noise function must be computed,

$$P'_{red,green,blue}(i,j) = P_{red,green,blue}(i,j) + Scale(P(i,j)) * ErrorCarry_{red,green,blue}(i,j).$$

The errorcarry value for each picture element results from the weighted average of previously computed errors. The errorfilter has the same size and the same weights as described in the first section. The function $Scale(r,g,b)$ returns a value between 0 and 1 depending on the actual color value (r, g, b) and the color values of the color table,

$$Scale(r,g,b) = \min\left(\frac{1}{3}, \frac{1}{Number\ of\ Entries\ in\ G(r,g,b)}\right)$$

This function is used because the whole errorcarry should not be superposed to only one pixel. Dithering color images without this restriction, areas with similar colors will be mapped to areas with disturbing patterns. The function *Scale* depends on the distribution of all color values of the color table in color space. Evaluating the errorcarry for *extreme* colors,

$$ErrorCarry_{red,green,blue}(i,j) = \frac{\sum_{(k,l)\in A} Error_{red,green,blue}(i+k, j+l) * Weight(k,l)}{\sum_{(k,l)\in A} Weight(k,l)}$$

the value of the *Scale*-function can be chosen bigger than computing the errorcarry for clustered colors. This is important if homogeneous areas of the image should be dithered.

Having computed $P'(i,j)$ the nearest entry in the color table must be searched. It is inserted in the dithered image at position (i,j). The resulting error $Error_{red,green,blue}(i,j)$ depends on the color value *col* and on $P'(i,j)$,

$$\begin{aligned} Error_{red,green,blue}(i,j) &= P'_{red,green,blue}(i,j) - col_{red,green,blue} - \\ &\quad \Big(1 - Scale(P(i,j))\Big) * ErrorCarry_{red,green,blue}(i,j) \end{aligned}$$

To avoid disturbing patterns in dithered images this algorithm can be improved by adding a function *noise*. This function depends on the distribution of colors in the color table too. The maximal amplitude of the noise function must be smaller than the average distance between all color representatives in a grid cell $G(r,g,b)$,

$$|noise(r,g,b)| \leq \frac{d_{box} * \sqrt{3}}{\sqrt[3]{Number\ of\ Entries\ in\ G(r,g,b)}}$$

To evaluate the final color value $P'_{red,green,blue}(i,j)$, we use

$$\begin{aligned} P'_{red,green,blue}(i,j) &= P_{red,green,blue}(i,j) + noise(P(i,j)) + \\ &\quad Scale(P(i,j)) * ErrorCarry_{red,green,blue}(i,j) \end{aligned}$$

3.3 Results

The table below compares the explained methods for color table evaluation in terms of execution time and color table quality. The quality criterion is the variance $\overline{d}$ of the distances between the color values of the image and the nearest color value in the color table,

$$\overline{d} = \frac{1}{M * N} * \sum_{i=1}^{N} \sum_{j=1}^{M} \min_{k=1..K} (d(c_k, P(i,j))^2)$$

$$d(c_1, c_2) = (c_{1.r} - c_{2.r})^2 + (c_{1.g} - c_{2.g})^2 + (c_{1.b} - c_{2.b})^2,$$

with c_k the k-th color in the color table C, and $P(i,j)$ the value of image P at position (i,j). Both images were quantized to $K = 256$ different colors, with $K' = 512$.

	Computational Costs	Picture 1 (8216 colors)	Picture 2 (74480 colors)
Standard Algorithm	$O(1)$	$\overline{d} = 669.9053$	$\overline{d} = 442.3563$
Popularity Algorithm	$O(N * M * log_2(K))$	$\overline{d} = 6.7597$	$\overline{d} = 116.8340$
Median Cut Algorithm	$O(N * M * log_2(K))$	$\overline{d} = 8.3895$	$\overline{d} = 44.1677$
Median Cut Algorithm prefering extreme colors	$O(N * M * log_2(K') + K * K')$	$\overline{d} = 6.6931$	$\overline{d} = 27.1358$
Median Cut Algorithm prefering extreme and clustered colors	$O(N * M * log_2(K) + K * K' + K' * log_2(K') + K * log_2(K'))$	$\overline{d} = 5.6031$	$\overline{d} = 26.1835$

To elucidate the results, the second picture is shown in three versions. Figure 1 shows the original picture with 24 bits per pixel. In figure 2 the color table was computed with the **Median Cut Algorithm**. The color table for figure 3 was chosen by the **Extended Median Cut Algorithm**.

4 DIGITAL HALFTONING OF PICTURE SEQUENCES

4.1 Noise In Digital Halftone Sequences

The example shown in the introduction was dithered with one out of many possible algorithms. However, this behavior is shown by all algorithms that represent similar areas in the graytone pictures

by different pixel-patterns in the binary pictures. Typical representatives are the *Floyd-Steinberg Algorithm* [FS75], the *Dot Diffusion Algorithm* [Knu87], or the *Constant-Level-Thresholding and Two-dimensional error-diffusion* as described by [Stu82]. For this type of algorithm, a straightforward solution to achieve non-flickering sequences is slightly shifting the pixels set in the actual binary picture in order to achieve pixel-patterns as close as possible to those of the previous picture. As a boundary condition, the number and distances of the pixel movements is to be minimized. Experiments show that even if every pixel is shifted by at most *one* position, more than two third of the pixels can be made to match. However, the shift by just one position essentially spoils the picture, contours become less defined and regular pixel-patterns become irregular. Furthermore, the movement by only one position is not enough to remedy the flickering problem.
Another class of dither algorithms place the pixels independently of their environment in the graytone or binary picture by creating always the same pixel-patterns from the same graytone-areas. If these algorithms additionally show a certain steadyness, i.e. similar gray areas are mapped into similar pixel-patterns, then they indeed create sequences without the disturbing flickering. The *ordered-dither* algorithms, for example dithering with a *dither matrix* [ES86] [Knu87], satisfy these properties, and they indeed create sequences without flickering. The problem of these algorithms is that they do not generate pictures of a quality as high as those algorithms which are suffering from the flickering problem. The following modification of the error diffusion algorithm shows how to maintain the high quality of the dithered pictures as well as reducing the flickering effect.

4.2 Modification Of The Error-Diffusion Algorithm

The principle of the error diffusion algorithm is to take an error carry into account in the further process of deciding whether a pixel is to be set or not. This is extended to picture sequences by facilitating the decision for a pixel to be set if there is a set pixel in the precedent picture and vice versa. The goal is to achieve the best matching pixel-patterns in two subsequent pictures, expressed by the Hamming-distance between two pictures.
This can easily be realized. We enlarge the probability for an unset pixel by lowering the threshold by a constant *Delta* if there is an unset pixel in the previous binary picture. If the corresponding pixel in the previous picture is set, we raise the threshold by adding *Delta*, thus lowering the chance that a pixel will not be set. In details,

$$Bitmap(i,j,t) = \begin{cases} 0 & if\ P(i,j,t) + ErrorCarry(i,j,t) + Delta(i,j,t-1) > \frac{Graylevel_{max} - Graylevel_{min}}{2} \\ 1 & if\ P(i,j,t) + ErrorCarry(i,j,t) + Delta(i,j,t-1) \leq \frac{Graylevel_{max} - Graylevel_{min}}{2} \end{cases}$$

with

$$Delta(i,j,t) = \begin{cases} -Delta & if\ Bitmap(i,j,t) = 1 \\ Delta & if\ Bitmap(i,j,t) = 0 \end{cases}$$

This definition of *Bitmap* is justified by the observation that neither constant raising nor lowering or varying the threshold will affect the quality of the dithered binary pictures except for very local areas. Further, setting pixels according to the previous picture rather than to the needs of the actual picture increase the errors made, but even in the worst case the $ErrorCarry$ at any position (i,j) is bound,

$$(*) \quad |ErrorCarry(i,j)| \leq \frac{Graylevel_{max} - Graylevel_{min}}{2} + Delta.$$

Figure 5 demonstrates the result of the application of this algorithm. The value for *Delta* was chosen at 15% of the range of the gray values. The picture shows that already this relatively small *Delta* is able to keep almost all pixels at their previous positions. A visual judgement of the moving sequence shows that the problem of flickering is sufficiently solved.
In figure 5, structures of previous pictures can be realized in the actual picture. The only differences in the corresponding *difference-picture* are at the moving parts of the picture, the wings of the mill and their shadows. (See figure 5). The motion blur effect thus introduced might be useful in case of computer generated graphics minimizing the effect of temporal aliasing. Nevertheless, in the next section, a modification of the algorithm is presented eliminating this effect.

Figure 5: Picture out of the sequence *windmill*

The left picture is dithered with the refinement just described. At the wings one can clearly see the remainders of previous pictures. The only differences in the right difference-picture are at the moving parts of the picture, the wings and their shadows.

4.3 The Change-Picture Technique

The first modification that is suggested now is to apply the pattern-keeping threshold manipulation only in regions where no or only minor changes happened between the previous and the actual graytone picture. A measure for the changes between to subsequent graytone pictures is given by

$$Change(i,j,t) = |P(i,j,t) - P(i,j,t-1)|$$

This yields the following formula for the value of $Bitmap(i,j,t)$,

$$Bitmap(i,j,t) = \begin{cases} 0 & if\ P(i,j,t) + ErrorCarry(i,j,t) + Delta(i,j,t-1) > \frac{Graylevel_{max} - Graylevel_{min}}{2} \\ 1 & if\ P(i,j,t) + ErrorCarry(i,j,t) + Delta(i,j,t-1) \leq \frac{Graylevel_{max} - Graylevel_{min}}{2} \end{cases}$$

with

$$Delta(i,j,t) = \begin{cases} -w(Change(i,j,t)) * (Graylevel_{max} - Graylevel_{min}) & if\ Bitmap(i,j,t) = 1 \\ w(Change(i,j,t)) * (Graylevel_{max} - Graylevel_{min}) & if\ Bitmap(i,j,t) = 0 \end{cases}$$

The function w determines to what extent the algorithm tries to copy the values of the pixels in the previous picture. The goal is to keep patterns with a maximum extent if no change takes place at a position (i,j) (i.e. $Change(i,j,t) = 0$). With increasing changes, w should become smaller. Some concrete values for w are shown in figure 6.

Unfortunately, this modification is not sufficient to overcome the *structure-keeping* property of the algorithm. It still can be observed that it tends to generate patterns which are similar to structures in previous pictures. This can be explained by a certain *inertia* of the error diffusion algorithm in general. (Spatial) changes in the gray level within a graytone picture may affect a larger area in the binary picture than the exactly corresponding positions in the graytone picture. In figure 1, there are no set pixels at all within the facets of the lower-left wing. Instead of white holes one should expect the same patterns as in the surrounding background, since the background in the graytone picture is the same over the whole area of this wing. These holes are the areas where the wrong old structures occur: the algorithm tries to keep the pixel-patterns in the facets of the wing, since the gray level of this areas did not change. By doing so, the undesired holes are kept. The disturbing results can be observed in figure 5. The problem was caused by the shadow-like disturbances created by the error diffusion algorithm. It should not be tried to keep pixel-patterns fixed in such areas.

```
         Delta
      +
100%  +
      +

      .
      .

      +
 15%  *
      +
      +
      +
  5%  +              *
      +
      +--------------+--------------+---------- ....  --*---->
     (0             13                                 255)
 Graylevel_min                                     Graylevel_max
```

Figure 6: Weighting function w

Therefore *safety-zone* is now built around all areas changing in time, by expanding the borders of this areas. Applying this dilatation to $Change$ yields

$$Change'_k(i,j,t) = \max_{i-k \le n \le i+k,\, j-k \le m \le j+k} (Change(n,m,t)),$$

where k determines the size of the dilatation-filter.
Making the threshold manipulation dependent on $Change'$ instead of $Change$ excludes the critical areas around changing structures from the areas where the pixel-patterns are tried to be kept fixed. Figure 7 shows a picture out of a sequence dithered with the additional application of the dilatation. The graytone (*CHANGE*)-picture was dilated with a filter of size $k = 7$. Besides some casual disturbances in a pixel-pattern there are no more structures of previous pictures.

4.4 Application

The proposed algorithm showed to be a helpful tool to dither non-flickering animations. The responsibility of the function w is to determine the amount of influence of the previous picture. For the windmill-example we achieved the best results with values for w as presented above. However, depending on the image content other values of w might improve the result.
Memory and execution time requirements are linear in the size of the images. Approximately 10 seconds on a *SUN 3-50* are required for a 200×200 image.
Since the resulting pixel-patterns are always the result of a compromise between the optimal pixel-pattern for the actual picture and the pattern of the previous picture, some casual irregularities in the patterns are unavoidable. When the algorithm comes from such an area into an area with a given pattern, it suddenly has to adopt to this pattern. These are critical areas. To our experience these disturbances are compensated by the benefit of non-flickering sequences.

Acknowledgements

The authors would like to express their appreciation to Prof. Heinrich Müller for reviewing this manuscript, to Christoph Rothe, who painted the diagrams and to Achim Stößer, who has designed the windmill-sequence and the pictorial examples in section 2.

Figure 7: Picture out of the sequence, which is now dithered with all refinements.

The threshold manipulation is dependent on the dilated graytone-change-picture. Old structures of previous pictures are no longer recognizable. Taking a closer view some minor irregularities in the pixel-patterns along the way the wing moved may be realized.

References

[ES86] J. Encarnacao and W. Straßer. *Computer Graphics.* Oldenbourg-Verlag, München, 1986.

[FS75] R.W. Floyd and L. Steinberg. An adaptive algorithm for spatial gray scale. *Int. Symp. Dig. Tech. Papers*, 36, 1975.

[GP88] M. Gervautz and W. Purgathofer. A simple method for color quantization: Octree quantization. *Proc. Computer Graphics International'88*, Springer Verlag Berlin:219–231, 1988.

[Hec82] P. Heckbert. Color image quantization for frame buffer display. *Computer Graphics*, 16(3):297–305, Juli 1982.

[HNS84] Hinterberger, Nievergelt, and Sevcik. The grid file, an adaptable, symetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, März 1984.

[JT80] C.L. Jakson and S.L. Tanimoto. Octrees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[Knu87] D.E. Knuth. Digital halftones by dot diffusion. *ACM Transactions on Graphics*, 6(4):245–273, Oktober 1987.

[Mur86] G.M. Murch. Human factors of color displays. *Eurographic Seminars*, Advances in Computer Graphics II:1–27, 1986.

[Stu82] P. Stucki. Image processing for documentation. *Fachberichte und Referate*, Textverarbeitung und Bürosysteme(13):245–282, 1982.