CSI 4133 Computer Methods in Picture Processing and Analysis

Fall 2024

Pengcheng Xi, Ph.D.

Outline

- Introduction to dithering
- Types of dithering
- Color quantization
- Applications of dithering
- Pros and cons
- Visual examples
- Software implementations

Image quantization

image quantized to
8 greyscales

image quantized to
4 greyscales

image quantized to
2 greyscales

Why dithering

- Essential in image processing when displaying or printing images on devices that have a limited color range, or when reducing color depth to save storage space without losing too much image quality
- Without dithering, visible contours can be detected between two levels. Our visual system is particularly sensitive to this.
- By adding random noise to the original image, we break up the contouring. The quantization noise itself remains evenly distributed about the entire image.
- Purpose:
  - The darker area will contain more black than white
  - The light area will contain more white than black

What is dithering

- As a **dictionary** term
  - To *dither* (dithers, dithering, dithered)
    - To be uncertain or unable to make a decision about doing something
    - To do something nervously
- As a **technical** term
  - A form of noise, or 'erroneous' signal or data which is added to sample data for the purpose of minimizing quantization error
  - Dither is routinely used in processing of both digital audio and digital video data
- A technique used to simulate color depth in images by strategically arranging pixels.
- Useful in displaying images on devices with limited color palettes

Types of dithering

- Random dithering
  - The simplest form, where noise is added randomly to pixels.
  - Creates a more “organic” look, but with less control over detail
- Ordered dithering
  - **Bayer** matrix or *ordered dithering*, where a threshold matrix determines which pixels are adjusted.
  - More structured and creates repetitive patterns
- Error diffusion dithering
  - **Floyd-Steinberg** dithering, where quantization error is distributed to neighboring pixels to maintain a smoother image appearance

Color quantization

- How color quantization works with dithering to approximate colors that aren't available in a limited color palette
- Applications:
  - Printing
  - Computer graphics
  - Image processing in low-color displays
  - Web design for compressing images while preserving details

Pros and cons

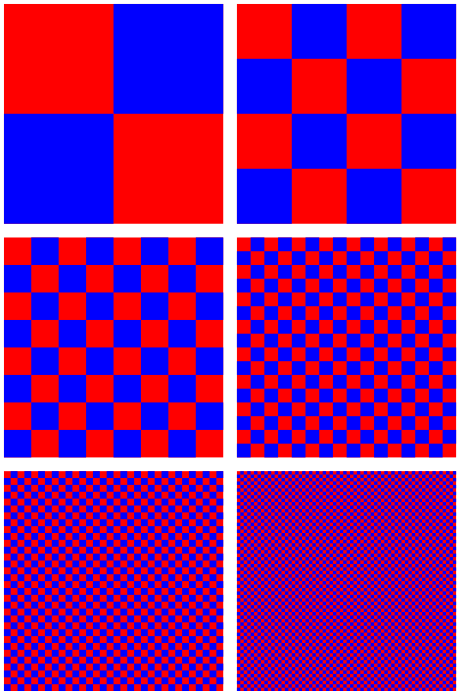
- Advantages
  - Enhanced visual quality on limited color devices, lower file sizes
- Disadvantages
  - Can introduce noise and may produce artifacts in some cases

Dithering

- How to approximate 4.8 using numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9?
  - Truncate (4.8) = 4
  - Round (4.8) = 5
  - Dithering -> randomly approximate it by 20% 4 and 80% 5

Dithering

- An illustration of dithering
  - Red and blue are the only colors used but, as the red and blue squares are made smaller, the patch appears magenta.



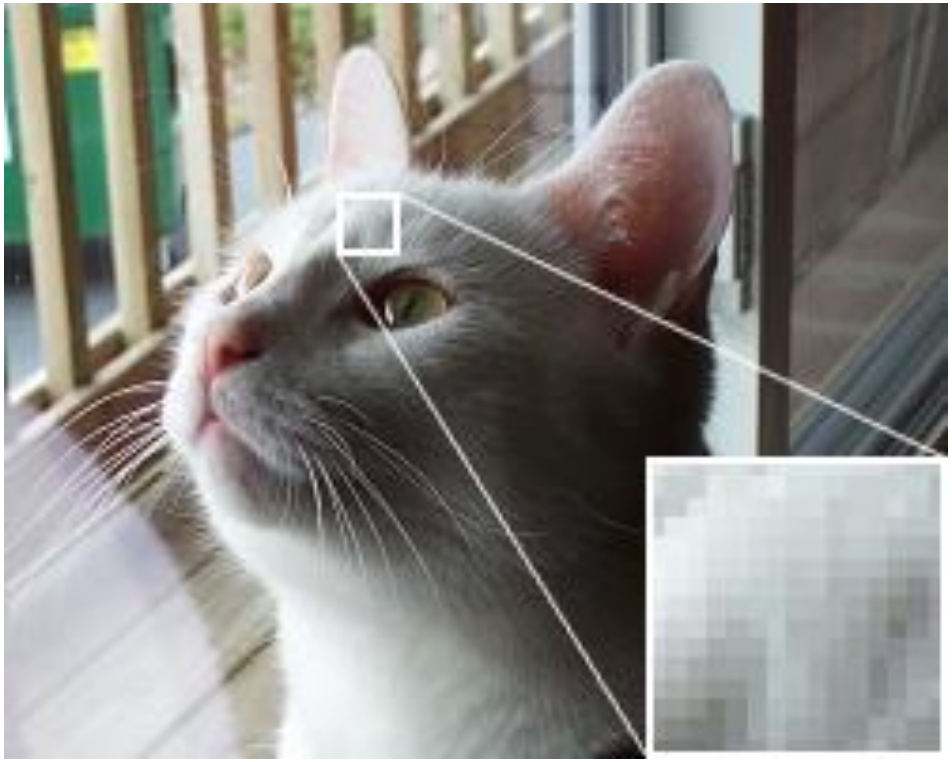
https://en.wikipedia.org/wiki/Dither#/media/File:Dithering_example_red_blue.svg

Dithering in an image

- **Objective**: to create the *illusion* of new colors and shade of gray by varying a pattern of dots (sometimes called *halftoning*), i.e., to reduce the number of gray levels from Q to Q'.
  - Recap on *halftoning*: the primary purpose of half-tone screening is to reproduce images with a wide range of tones using only a few colors

Dithering

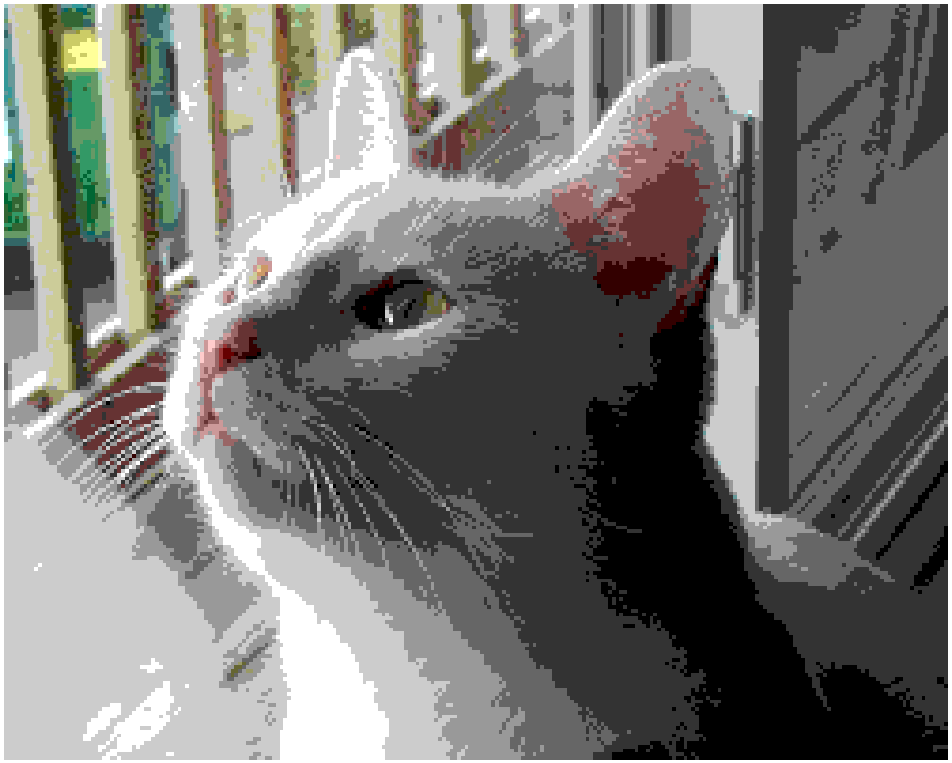
- Original photo
  - Note the smoothness in the detail
- 24-bit color (2^{24} = 16,777,216 colors)



http://en.wikipedia.org/wiki/Dithering

Dithering

- Using the web-safe color (216 color) palette
  - With no dithering applied
  - Note the large flat areas and loss of detail



http://en.wikipedia.org/wiki/Dithering

Web safe colors emerged during the early era of the internet; a standardized palette of 216 colors that displayed consistently across all major browsers.

Dithering

- Using the web-safe color palette
  - With **Floyd-Steinberg dithering**
  - Note that even though the same palette is used
  - The dithering gives a better representation of the original

http://en.wikipedia.org/wiki/Dithering

Dithering

- Reducing the color depth of an image can often have significant visual side-effects
  - If the original image is a photograph, it is likely to have thousands, or even millions of distinct colors
  - The process of constraining the available colors to a specific color palette effectively throws away a certain amount of color information

Dithering

- In a dithered image, colors not available in the palette are approximated by a diffusion of colored pixels from within the available palette.
- If the original pixel colors are simply transformed into the closest available color from the palette
  - Typically, this approach results in flat areas and a loss of detail, and may produce patches of color that are significantly different from the original
- Dithering helps to reduce color banding and flatness

Dithering in an image

- Human eyes perceive the diffusion as a mixture of the colors in it
- Dithering is analogous to the halftone technique used in printing
  - Dithered images, particularly those with relatively few colors, can often be distinguished by a characteristic graininess, or speckled appearance.

Dithering applications

- One common application of dithering is to more accurately display graphics containing a greater range of colors than what the hardware is capable of showing
  - E.g., to display a photographic image containing millions of colors on video hardware that is only capable of showing 256 colors at a time

Dithering applications

- For situations in which the graphic file format is the limiting factor
  - The commonly-used GIF format is restricted to the use of 256 or fewer colors in many graphics editing software
  - Images in other file formats, such as PNG, may also have such a restriction imposed on them for the sake of a reduction in file size

Dithering algorithms

- **Thresholding** (average dithering)
  - Each pixel value is compared against a fixed threshold
- **Random dithering**
  - Each pixel value is compared against a random threshold, resulting in a staticky image
- **Ordered dithering**
  - Dithers using a fixed pattern. For every pixel in the image, the value of the pattern at the corresponding location is used as a threshold. E.g., **Bayer**
- **Error diffusion dithering**
  - Diffuse the quantization error to neighboring pixels. E.g., **Floyd-Steinberg**

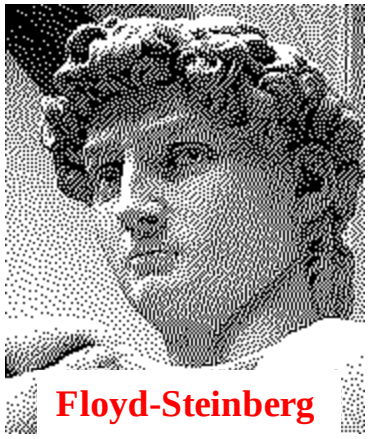

(Original)


Threshold


Random


Halftone


Bayer


Floyd-Steinberg

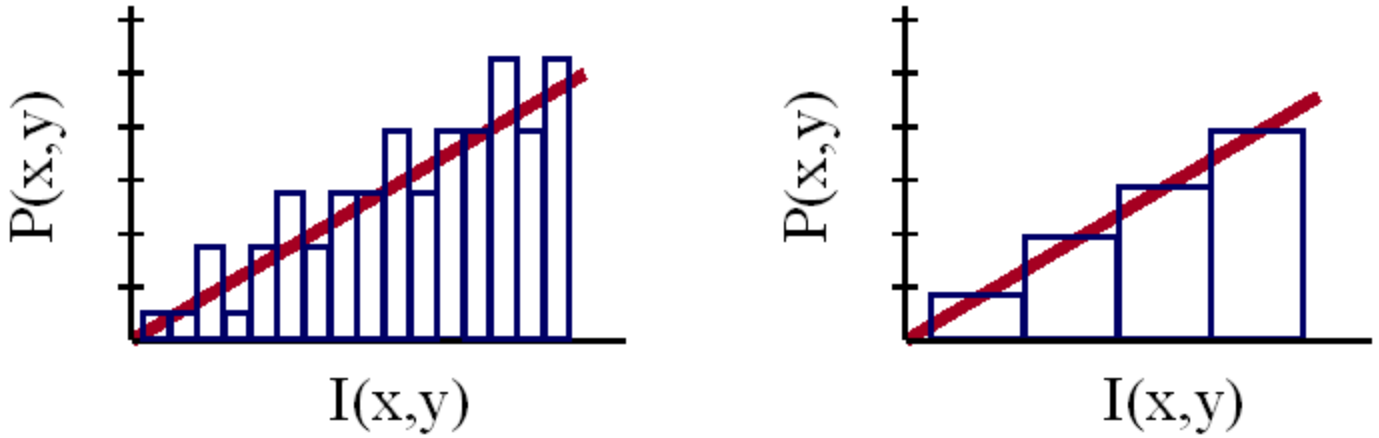
21

The unlabelled images are those from other error-diffusion dithering algorithms

http://en.wikipedia.org/wiki/Dithering

Random dither

- Randomize quantization errors
  - Errors appear as noise



$$P(x, y) = \text{trunc}(I(x, y) + \text{noise}(x,y) + 0.5)$$

http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/dither/

Random dither

Original
(8 bits)

Uniform
Quantization
(1 bit)

Random
Dither
(1 bit)

http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/dither/

Ordered dither

- Pseudo-random quantization errors
  - Matrix stores pattern of thresholds

```
i = x mod n
j = y mod n
e = I(x,y) - trunc(I(x,y))
if (e > D(i,j))
        P(x,y) = ceil(I(x, y))
else
        P(x,y) = floor(I(x,y))
```

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/dither/

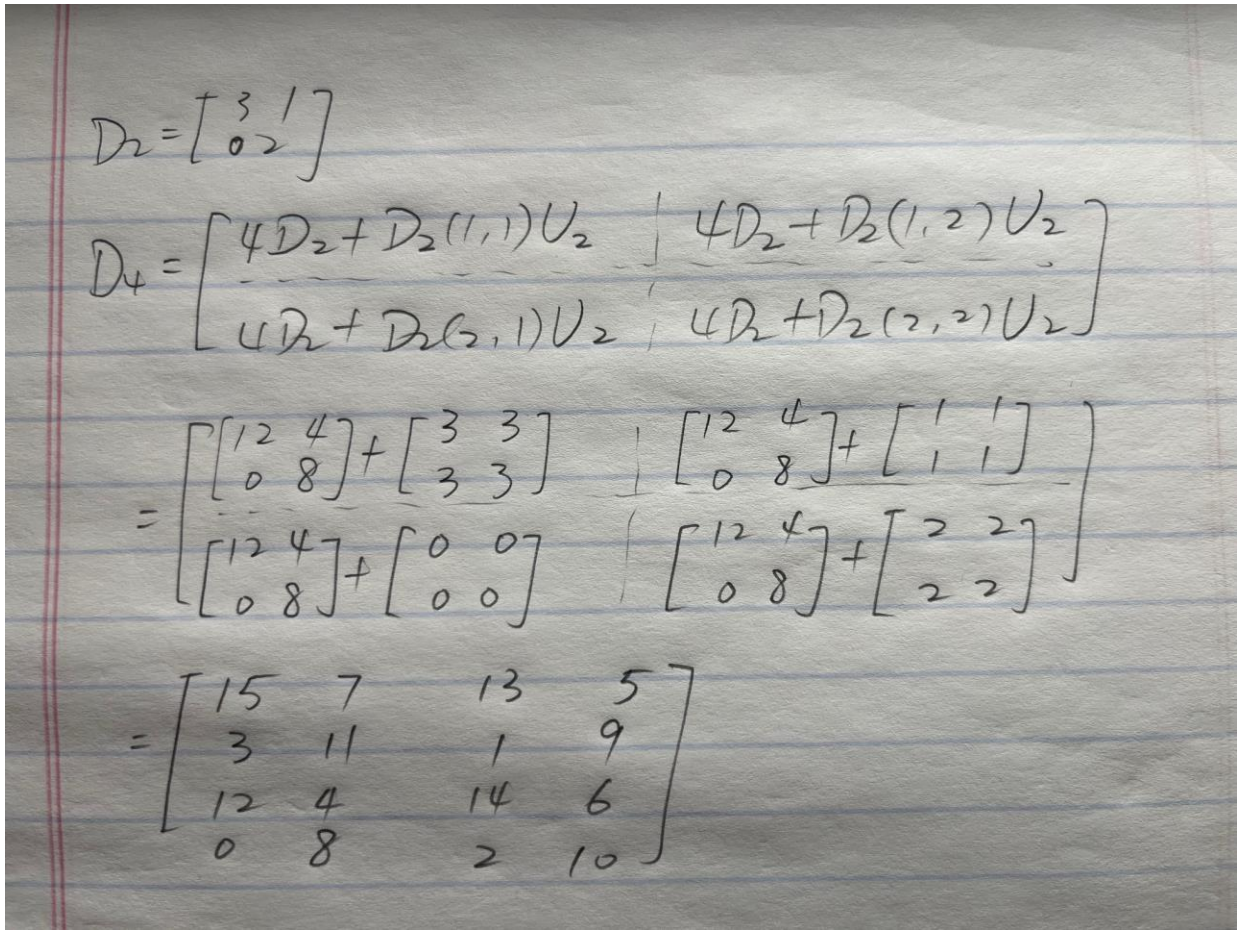
Ordered dither

- Bayer's ordered dither matrix

$$D_n = \begin{bmatrix} 4D_{n/2} + D_2(1,1)U_{n/2} & 4D_{n/2} + D_2(1,2)U_{n/2} \\ 4D_{n/2} + D_2(2,1)U_{n/2} & 4D_{n/2} + D_2(2,2)U_{n/2} \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{bmatrix}$$

Bayer's ordered dither matrix



The dither matrix needs to be scaled depending on the quantization factor

Ordered Dithering for binary values

$$D = \begin{bmatrix} 0 & 128 \\ 192 & 64 \end{bmatrix}$$

Replicate D to image size

$$D_2 = \begin{bmatrix} 0 & 128 & 32 & 160 \\ 192 & 64 & 224 & 96 \\ 48 & 176 & 16 & 144 \\ 240 & 112 & 208 & 80 \end{bmatrix}$$

$$p(i,j) = \begin{cases} 1 & \text{if } x(i,j) > d(i,j) \\ 0 & \text{if } x(i,j) \leq d(i,j) \end{cases}$$

d(i,j) is the matrix obtained by replicating D

```
x = imread('newborn.jpg'); % size of 256x256

D=[0 128;192 64];
r=repmat(D,128,128); % size of 256x256
x2=x>r; imshow(x2)

D2=[0 128 32 160;192 64 224 96; 48 176 16 144;240
  112 208 80];
r2=repmat(D2,64,64);
x4=x>r2; imshow(x4)
```

Ordered Dithering for more output grey values

Example: four output levels 0, 1, 2 and 3.

Step 1: First quantize by dividing grey value x(i,j) by 85

`q(i,j) = [(x(i,j)/85]` (because 255/3 =85)

Step 2: Suppose now that our replicated dither matrix d(i,j) is scaled so that its values are in the range 0 – 85

e.g., if the dither matrix values are in the range of [0, 15], they need to be scaled up to be in [0, 85]

Step 3: The final value p(i,j) is defined as

$$p(i,j) = q(i,j) + \begin{cases} 1 & \text{if } x(i,j) - 85q(i,j) > d(i,j) \\ 0 & \text{if } x(i,j) - 85q(i,j) \leq d(i,j) \end{cases}$$

Matlab

```
x = imread('newborn.jpg');
%size 256x256

% Dither to 4 grey levels
D=[0 56;84 28];
r=repmat(D,128,128);
x=double(x);
q=floor(x/85); %ceil(255/3)=85
x4=q+(x-85*q>r);
subplot(121); imshow(uint8(85*x4));

% Dither to 8 grey levels
clear r x4
D=[0 24;36 12];
r=repmat(D,128,128);
x=double(x);
q=floor(x/37); %ceil(255/7)=37
x4=q+(x-37*q>r);
subplot(122); imshow(uint8(37*x4));
```

Dithering to 4 output greyscales

Dithering to 8 output greyscales

Quantization vs. Dithering

image quantized to
8 greyscales

image quantized to
4 greyscales

image quantized to
2 greyscales

Error diffusion

- Quantization error: the difference between the original gray value and the quantized value
- If quantized to 0 and 255, intensity value closer to the center (128) will have higher error

Error diffusion (cont.)

- **Goal**: spread this error over neighboring pixels
- **Method**: sweeping from left to right, top to down, for each pixel
  - Perform quantization
  - Calculate the quantization error

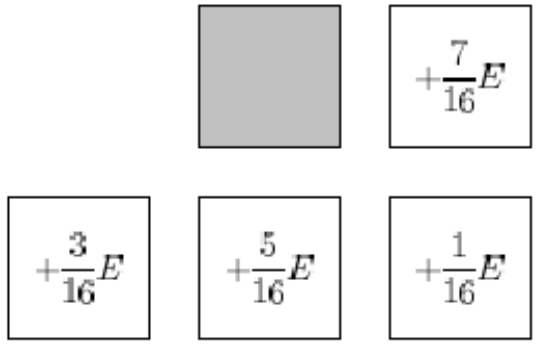
$$E = \begin{cases} p(i,j) & \text{if } p(i,j) < 128 \\ p(i,j) - 255 & \text{if } p(i,j) \geq 128 \end{cases}$$

  - Spread the error to the right and below neighbors

Dithering algorithms

- **Floyd-Steinberg** dithering algorithm
  - It minimizes visual artifacts through an error-diffusion process
  - The Floyd-Steinberg algorithm typically produces images that more closely represent the original than simpler dithering algorithms

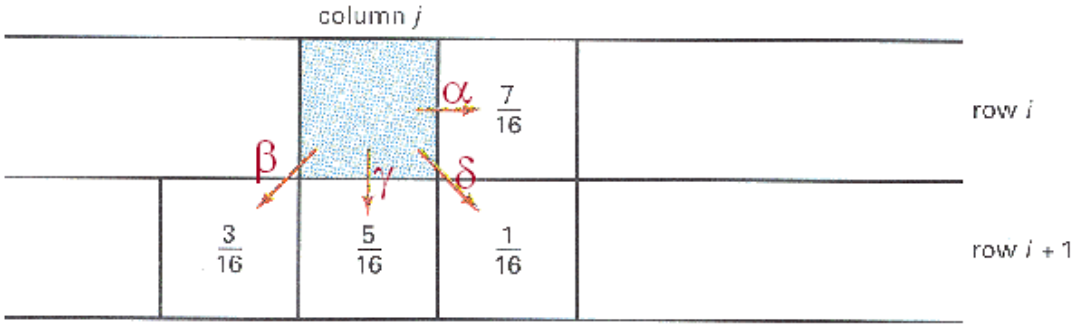
Error diffusion



- The process is implemented by adding the weighted error to the neighboring pixels.
- A number of different weighting schemes have been proposed
- The **Floyd and Steinberg filter**
  - Weights must be *normalized* in actual implementation
  - The ordering of processing pixels is called *raster ordering*:
    - From left to right and then from top to bottom

Error diffusion dither

- Spread quantization error over neighbour pixels
  - Error dispersed to pixels right and below



$$\alpha + \beta + \gamma + \delta = 1.0$$

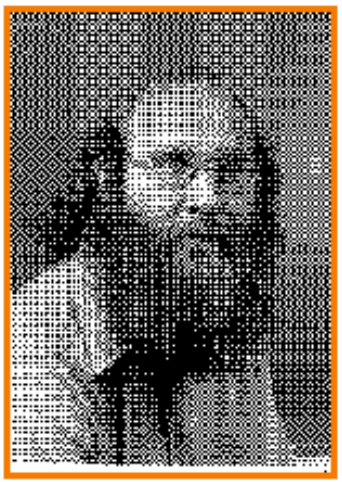
Figure 14.42 from H&B

http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/dither/

Error diffusion dither

Original
(8 bits)

Random
Dither
(1 bit)



Ordered
Dither
(1 bit)

Floyd-Steinberg
Dither
(1 bit)

http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/dither/

Error diffusion – Floyd & Steinberg algorithm

- For each pixel *p(i,j)*, perform the following sequence of steps:
  - Perform quantization
  - Calculate quantization error

$$E = \begin{cases} p(i,j) & \text{if } p(i,j) < 128 \\ p(i,j) - 255 & \text{if } p(i,j) >= 128 \end{cases}$$

  - Spread this pixel error to the right and below according to the table

Note: quantization error needs to be computed based on requirements e.g., if reducing the number of intensity levels from 128 to 2, we will use 64 as the threshold

MATLAB code

```
img_size = size(img);
x_max = img_size(1);
y_max = img_size(2);

img = double(img);

for x = 1:x_max
    for y = 1:y_max
        % Read pixel value
        oldpixel = img(x,y);

        % check if more than 25%
        if (oldpixel > 192)
            newpixel = 255;
        elseif (oldpixel > 128)
            newpixel = 170;
        elseif (oldpixel > 64)
            newpixel = 85;
        else
            newpixel = 0;
        end
        % set pixel value into new image
        image_floyd(x,y) = newpixel;

        %calculate error
        quant_error = double(oldpixel - newpixel);

        % check boundaries and accumulate error
        if(x + 1 <= x_max)
            img(x+1,y) = double((img(x+1,y) + 7/16 * quant_error));
        end
        if ((x-1 > 0) && (y+1 < y_max))
            img(x-1,y+1) = double((img(x-1,y+1) + 3/16 * quant_error));
        end
        if(y + 1 <= y_max)
            img(x,y+1) = double((img(x,y+1) + 5/16 * quant_error));
        end
        if((x + 1 <= x_max) && (y + 1 <= y_max))
            img(x+1,y+1) = double((img(x+1,y+1) + 1/16 * quant_error));
        end
    end
end

image = image_floyd;
```

Error diffusion – Floyd & Steinberg algorithm

- Several points to note
  - The error is spread to pixels before quantization is performed on them. Thus the error diffusion will affect the quantization level of those pixels
  - Once a pixel is quantized, its values will not be affected, as the error diffusion only affects pixels to the right and below
  - To implement this algorithm, we need to embed the image in a larger array of zeros, so that the indices do not go outside the bounds of our array

Matlab **`dither`** function

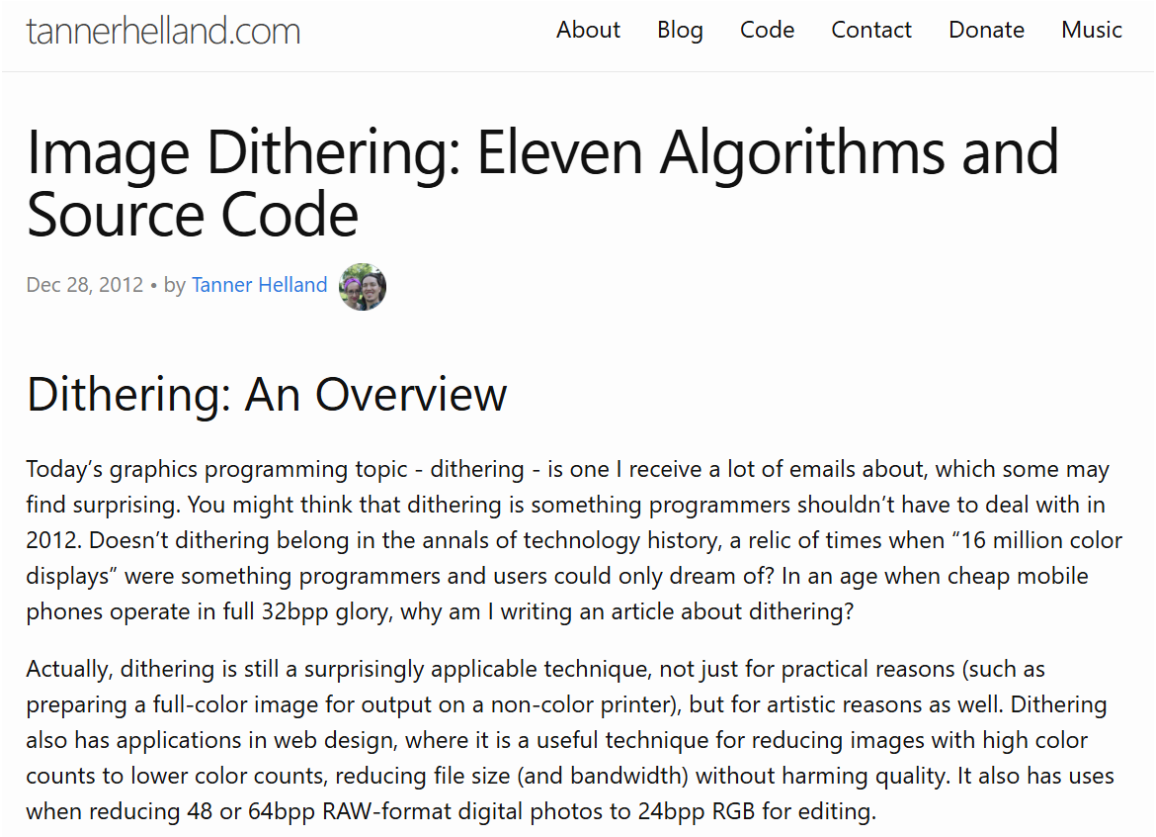
- Matlab **`dither`** function implements the Floyd-Steinberg error diffusion

Example:

```
im = imread('cameraman.tif');
imshow(im)
im2= dither(im);
figure; imshow(im2)
```

Useful resource

- https://tannerhelland.com/2012/12/28/dithering-eleven-algorithms-source-code.html



tannerhelland.com

About Blog Code Contact Donate Music

Image Dithering: Eleven Algorithms and Source Code

Dec 28, 2012 • by Tanner Helland

Dithering: An Overview

Today's graphics programming topic - dithering - is one I receive a lot of emails about, which some may find surprising. You might think that dithering is something programmers shouldn't have to deal with in 2012. Doesn't dithering belong in the annals of technology history, a relic of times when "16 million color displays" were something programmers and users could only dream of? In an age when cheap mobile phones operate in full 32bpp glory, why am I writing an article about dithering?

Actually, dithering is still a surprisingly applicable technique, not just for practical reasons (such as preparing a full-color image for output on a non-color printer), but for artistic reasons as well. Dithering also has applications in web design, where it is a useful technique for reducing images with high color counts to lower color counts, reducing file size (and bandwidth) without harming quality. It also has uses when reducing 48 or 64bpp RAW-format digital photos to 24bpp RGB for editing.

Dither (ordered dither)

- [Question 1]
  - You want to reduce the number of intensities in an image from 512 to 16 using a dither matrix of size 4x4. Compute the required dither matrix and explain how this one would be used to reduce the number of gray levels.

Answer

- Step 1: understanding the dither matrix for 16 intensity levels
  - Here is a Bayer matrix scaled from 0 to 15: $M = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$
  - This matrix is typically used to threshold an image with 16 possible levels (0-15) in a way that distributes errors visually across the pixel grid

Answer (cont.)

- Step 2: compute scaling intervals
  - Since we want 16 intensity levels, we divide the 512-level range into 15 equal intervals. Each level corresponds to ceil(511/15)=35 intensity values.

Answer (cont.)

- Step 3: using the dither matrix to quantize the image
  - For each pixel value x, we compute q=floor(x/35)
    - q will be in a range of (0, 14)
    - q is also called current base level (will be updated later)
  - **Scale dither matrix if necessary**
  - Update the pixel value x with:
    - $x' = q + (x - 35q > M)$
    - x' will be in a range of (0, 15) -> this is what we wanted

Dither (Floyd-Steinberg)

- [Question 2] Convert the following image to be black and white image using Floyd-Steinberg dithering algorithm

120	**130**	**140**
150	**160**	**170**
180	**190**	**200**

Dither (Floyd-Steinberg)

- The general process is traversing each pixel in the image from left to right, top to bottom
- We apply dithering to the center pixel (160) in this example
- Step 1: thresholding:
  - The threshold we use is 128, which is the midpoint between black (0) and white (255)
  - The center pixel has a value of 160, which is greater than 128, therefore we set the pixel value to 255 (white)

Dither (Floyd-Steinberg)

- Step 2: calculate quantization error:
  - Quant_error = 160 – 255 = -95
- Step 3: distribute the error to neighbors:
  - Right neighbor (170):
    - 170+(-95*7/16)
  - Bottom-left neighbor (180):
    - 180+(-95*3/16)
  - Bottom neighbor (190):
    - 190+(-95*5/16)
  - Bottom-right neighbor (200):
    - 200+(-95*1/16)

Dither (Floyd-Steinberg)

- Step 4: update pixel grid after each step
- Step 5: clip the image pixel values to be either 0 (black) or 255 (white)