Dithering

Max Tirdatov

Seminar: PostScript, WS 2022/23

RWTH Aachen University

Motivation

Problem statement

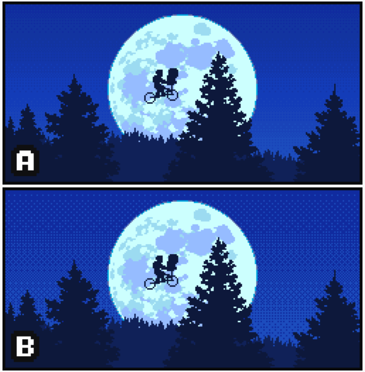
- Color depth is the number of bits available for color information per pixel
- When the color depth is reduced, how can the original color be imitated?
- Even these days, the question often comes up in printing, computer graphics and digital art

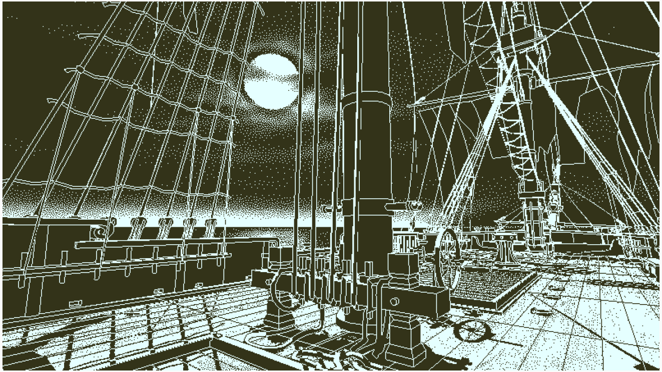
Computer graphics

- In the mid-1990s: 216 web-safe colors
- GIF only supports 8 bits per pixel

Source: `https://commons.wikimedia.org/wiki/File:GIFPAL.png`

Digital art





Source: https://pixelparmesan.com/dithering-for-pixel-artists/, https://obradinn.com/

Naive approaches

Fixed threshold

- The goal is to convert an 8 bpp grayscale image to a 1 bpp image with only pure black (0) and pure white (1)
- Assume the shades of gray are values between 0 and 1
- Idea: compare each pixel with a fixed threshold value

$$\texttt{output} = \begin{cases} 0, & \texttt{input} < 0.5 \\ 1, & \text{otherwise} \end{cases}$$

$\rightarrow$

- Detail is missing, no illusion of color depth

Source: Dietmar Rabich, CC BY-SA 4.0

https://commons.wikimedia.org/wiki/File:San_Francisco_(CA,_USA),_Golden_Gate_Bridge_--_2022_--_3023_(bw).jpg

Color banding

- Generalization: a threshold $t \in [0, 1]$ which specifies a color level between the nearest available ones to compare against
- PostScript uses a similar approach (discussed later)
- Example with linear interpolation between nearest levels A, B:

$$\texttt{output} = \begin{cases} A, & \texttt{input} < (1-t)A + tB \\ B, & \text{otherwise} \end{cases}$$

$\rightarrow$

- Using a fixed threshold results in clearly visible color bands

Random dithering

- Idea: use a random threshold for each pixel

$$\texttt{output} = \begin{cases} 0, & \texttt{input} < \texttt{random()} \\ 1, & \text{otherwise} \end{cases}$$

$\rightarrow$

- 30%-gray pixels are quantized to white about 30% of the time
- Can we do better than that?

Ordered dithering

Ordered dithering

- Idea: threshold pattern as an m-by-n matrix M
- Threshold now depends on pixel coordinates x, y:

$$\texttt{output} = \begin{cases} 0, & \texttt{input} < M_{x \bmod m,\, y \bmod n} \\ 1, & \text{otherwise} \end{cases}$$

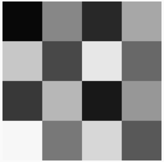
- Threshold values should be uniformly distributed to keep the probabilistic property of random dithering
- The matrix is to be chosen with care

Bayer matrix

- Recursive definition:

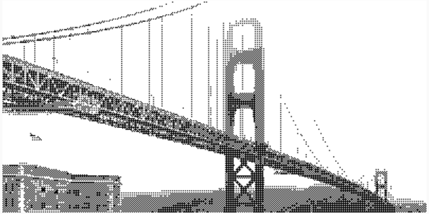
$$D^2 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}, \quad D^{2n} = \begin{bmatrix} 4D^n + D^2_{00} & 4D^n + D^2_{01} \\ 4D^n + D^2_{10} & 4D^n + D^2_{11} \end{bmatrix} = \begin{bmatrix} 4D^n + 0 & 4D^n + 2 \\ 4D^n + 3 & 4D^n + 1 \end{bmatrix}$$

- D^n is an n-by-n matrix filled with all integers from 0 to $n^2 - 1$
- $M^n := \frac{1}{n^2} \cdot D^n$ can be used as a threshold map
- Example: $M^4 = \dfrac{1}{16} \cdot \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$

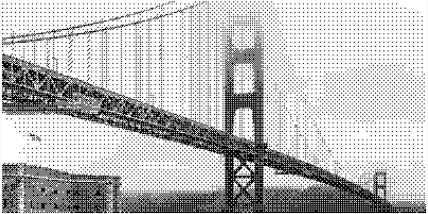


Bayer dithering

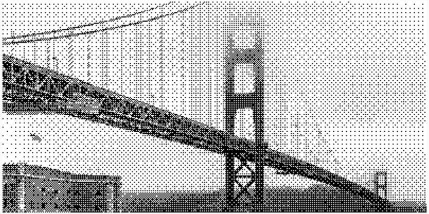
Original



Dithered with M^2



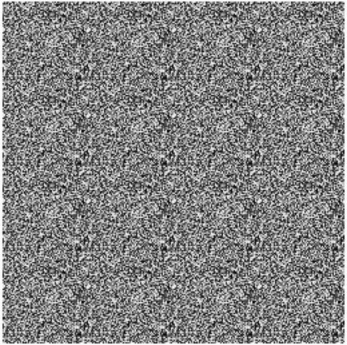
Dithered with M^4



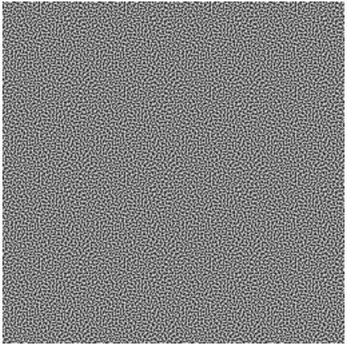
Dithered with M^8

Blue noise

- Like in blue light, higher frequencies have higher intensities
- Can be tiled seamlessly for that reason



White noise tiling



Blue noise tiling

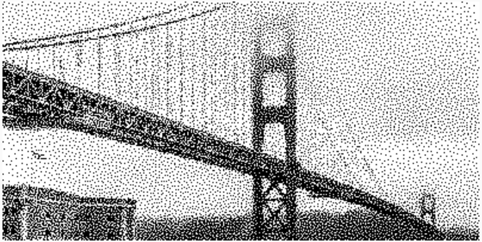
- Isotropic: frequency decomposition does not depend on the viewing angle

Source: http://momentsingraphics.de/BlueNoise.html

Blue noise dithering

- Void-and-cluster method can be used to generate blue noise
- The key part is finding areas in a 1bpp image where dot density is highest (clusters) or lowest (voids), then removing or adding dots from/to that area
- Needs to be precomputed since generation takes time

$\rightarrow$



Dithering with a 64×64 threshold map generated by void-and-cluster method

Error diffusion

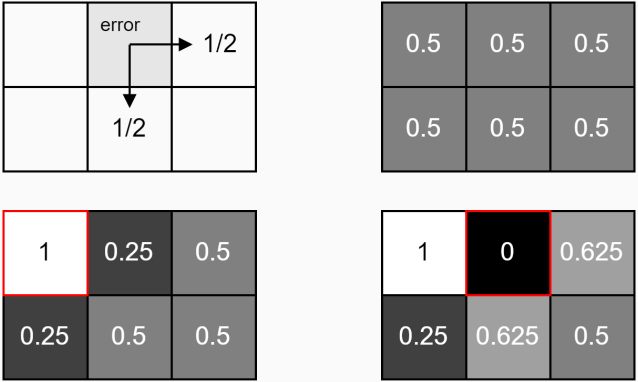
Error diffusion

- Rounding to the nearest available color level results in quantization error:

$$\texttt{error} = \texttt{input} - \texttt{output}$$

- Idea: spread the error to neighboring pixels to compensate for it later
- Thus, if a pixel is quantized to a darker shade, its neighbors are more likely to be quantized to a lighter shade, and vice versa
- In its simplest form: add the error to the pixel to the right of the current one
- More involved distributions produce better results

Example



error: -0.5

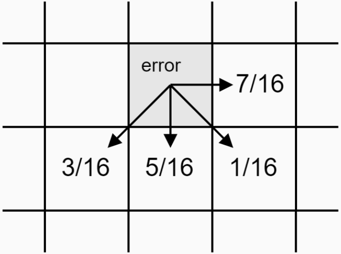
error: 0.25

- Parallelization is not straightforward

Floyd–Steinberg dithering

for each y from top to bottom **do**
  for each x from left to right **do**
    $oldpixel := pixels[y][x]$
    $newpixel := \text{findNearestPaletteColor}(oldpixel)$
    $pixel[y][x] := newpixel$
    $error := oldpixel - newpixel$

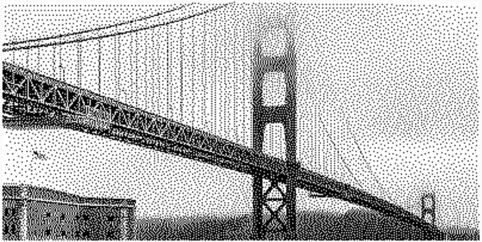
    $pixel[y][x+1] \mathrel{+}= \frac{7}{16} \cdot error$
    $pixel[y+1][x-1] \mathrel{+}= \frac{3}{16} \cdot error$
    $pixel[y+1][x] \mathrel{+}= \frac{5}{16} \cdot error$
    $pixel[y+1][x+1] \mathrel{+}= \frac{1}{16} \cdot error$
  end for
end for



- Intermediate pixel values exceeding the valid range need to be handled correctly
- Care needs to be taken at image edges

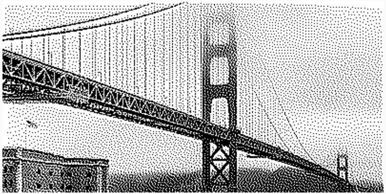
Floyd–Steinberg dithering

$\rightarrow$

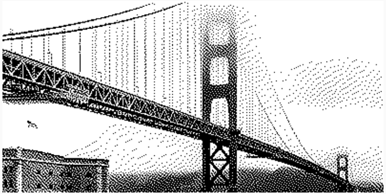


Other methods

- Floyd–Steinberg: $\frac{1}{16} \cdot \begin{pmatrix} & * & 7 \\ 3 & 5 & 1 \end{pmatrix}$
- Jarvis–Judice–Ninke: $\frac{1}{48} \cdot \begin{pmatrix} & & * & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{pmatrix}$
- Atkinson: $\frac{1}{8} \cdot \begin{pmatrix} & * & 1 & 1 \\ 1 & 1 & 1 & \\ & 1 & & \end{pmatrix}$
- And many more...



Jarvis–Judice–Ninke

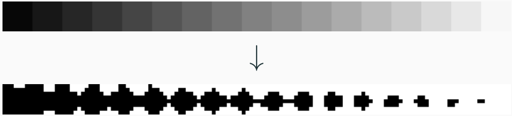


Atkinson

Halftone

Halftone

- Traditionally, variously sized round dots of ink were used to produce different color shades
- The dots can be approximated with bitmaps:



- This is PostScript's first supported approach
- Output resolution must be higher in order to preserve detail

Source: `https://commons.wikimedia.org/wiki/File:Statues_of_Abraham_Lincoln_(1915)_(14597861910).jpg`

PostScript halftone dictionaries

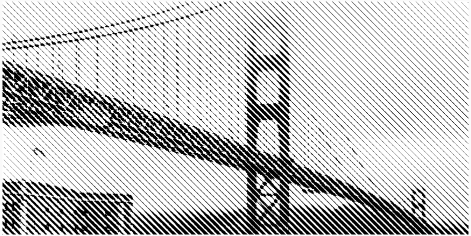
- Dictionary syntax: << $\texttt{key}_1$ $\texttt{value}_1$ $\texttt{key}_2$ $\texttt{value}_2$ ... $\texttt{key}_n$ $\texttt{value}_n$ >>
- Halftone dictionaries are used to configure halftone screen parameters
  - Retrieve current halftone dictionary: – **currenthalftone** *dict*
  - Set a new one: *dict* **sethalftone** –
- Several types of halftone dictionaries are available

Type 1 halftone dictionaries

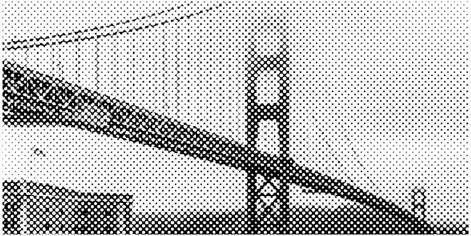
- Defines a single halftone screen by a frequency, angle and spot function
- The screen is made up of cells, each covering a certain number of device pixels
- Frequency determines the number of cell lines per inch (lpi)
- Each pixel has coordinates within its cell's coordinate system, where the range for both x and y is from -1.0 to $+1.0$
- The coordinates are passed to the spot function which outputs a number between -1.0 and $+1.0$
- The output determines how soon the pixel turns white as the cell's gray level varies from black to white

Simple spot functions

- Line screen
  - Measure the distance from the x axis
  - As a function: $f(x, y) = y^2$
  - `{exch pop dup mul}`



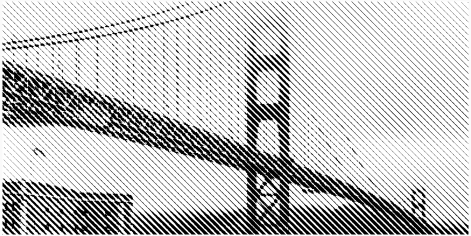
- Round dot screen
  - Measure the distance from the origin
  - As a function: $f(x, y) = \frac{x^2+y^2}{2}$
  - `{dup mul exch dup mul add 2 div}`



Code

Code to produce the line screen example from previous slide:

```
<<
  /BeginPage {
    <<
      /HalftoneType 1
      /Frequency 18
      /Angle 45
      /SpotFunction {exch pop dup mul}
    >> sethalftone
  }
>> setpagedevice
```



Type 3 halftone dictionaries

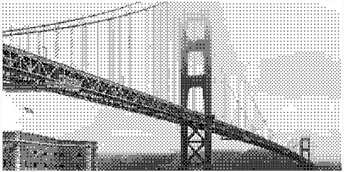
- Defines a single halftone screen by a threshold array containing 8-bit sample values taken from a string
- Enables implementation of fixed threshold, random and ordered dithering
- 8-bit threshold values instead of $t \in [0, 1]$
- Stored in a string that is essentially a byte array:

$$D^4 = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \xrightarrow{\times 16} \begin{bmatrix} 0 & 128 & 32 & 160 \\ 192 & 64 & 224 & 96 \\ 48 & 176 & 16 & 144 \\ 240 & 112 & 208 & 80 \end{bmatrix} \rightarrow \underbrace{\texttt{<008020A0C040E06030B01090F070D050>}}_{\text{PostScript hexadecimal string}}$$

Code

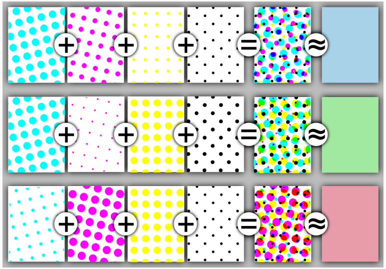
Code to dither with the 4-by-4 Bayer matrix:

```
<<
  /BeginPage {
    <<
      /HalftoneType 3
      /Width 4
      /Height 4
      /Thresholds <008020A0C040E06030B01090F070D050>
    >> sethalftone
  }
>> setpagedevice
```



Type 5 halftone dictionaries

- Defines an arbitrary number of halftone screens, one for each color component
- Keys are names of color components, such as `/Cyan`, `/Magenta`, `/Yellow` and `/Black` for CMYK space
- Values are halftone dictionaries of other types



Source: https://commons.wikimedia.org/wiki/File:Halftoningcolor.svg

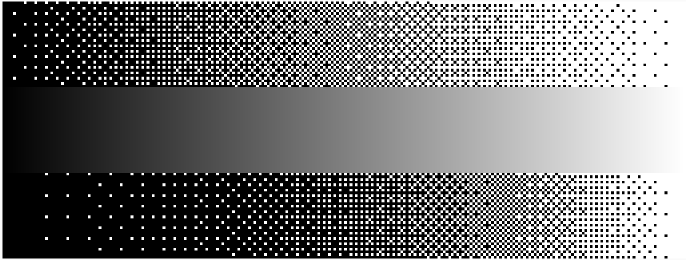
Transfer functions

- A transfer function can be used to adjust pixel values before halftone is applied
- Can be specified via optional `/TransferFunction` key
- Useful for gamma correction: $C_{\text{linear}} = \begin{cases} \dfrac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \leq 0.04045 \\ \left(\dfrac{C_{\text{srgb}} + 0.055}{1.055}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$

No transfer function:

Original:

With gamma correction:



Questions?

Comparison

Ordered dithering	Error diffusion
⊕ Parallelization is straightforward	⊖ Parallelization is challenging
⊕ Suitable for animations	⊖ Too unpredictable
⊖ Tends to blur images	⊕ Tends to enhance edges, making text more readable

Source: `https://commons.wikimedia.org/wiki/File:Signs_in_Gibraltar_Dec_2004.jpg` (CC BY-SA 3.0)

Ghostscript output devices

- Halftone dictionary is only consulted when the output color depth is not sufficient
- Color depth differs across output devices
- Use `-sDEVICE` and `-sOutputFile` options to set output device and file
- Some devices like `pngmonod` ignore the halftone dictionary and apply error diffusion instead
- Images produced by `pngmono`, `pngmonod` and `png256` devices have been used throughout this presentation
- Full list available at Ghostscript website

Adobe Systems Inc.
PostScript Language Reference.
Addison-Wesley Publishing Company, 3 edition, 1999.

Artifex Software Inc.
Details of Ghostscript Output Devices.
https://ghostscript.com/docs/9.54.0/Devices.htm.

Surma.
Ditherpunk — The article I wish I had about monochrome image dithering.
https://surma.dev/things/ditherpunk/.

John F. Jarvis, Charlie Judice, and William H. Ninke.
A survey of techniques for the display of continuous tone pictures on bilevel displays.
Computer Graphics and Image Processing, 5:13–40, 1976.

Robert Ulichney.
Void-and-cluster method for dither array generation.
In *Electronic imaging*, 1993.

Wikipedia contributors.
Wikipedia articles: *Dither, Ordered dithering, Error diffusion, Floyd–Steinberg dithering, Atkinson dithering, Halftone*.
https://en.wikipedia.org/.