Image Representation

What's an Image?

Image: distribution of light energy on 2D "film": E(x, y, λ, t)

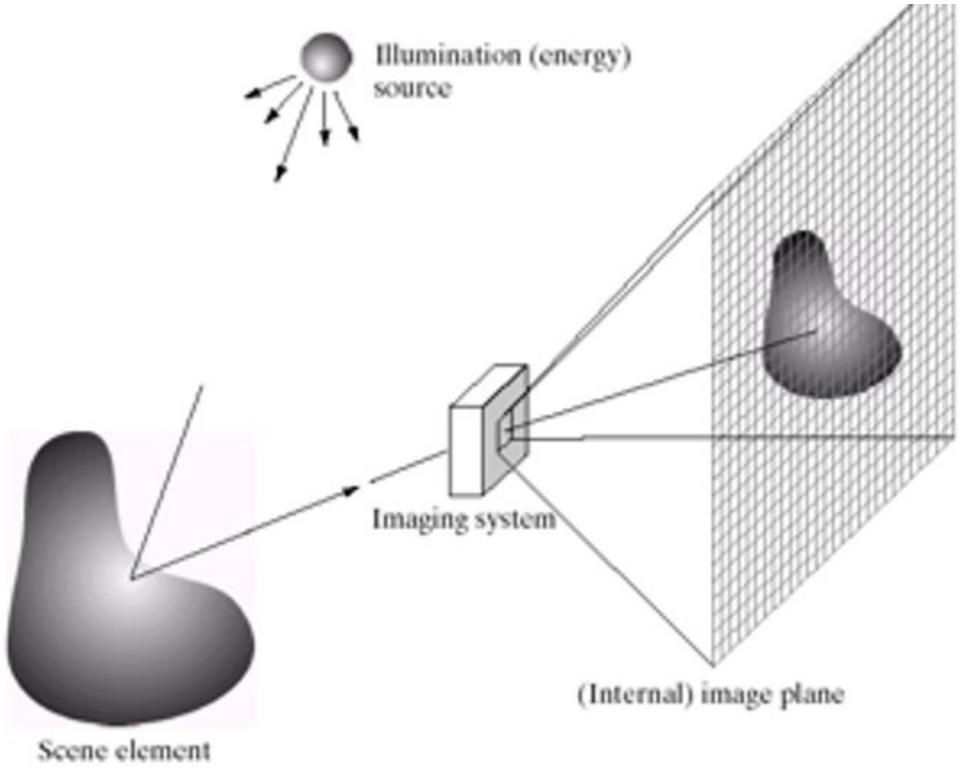
(x, y) - position

λ - wavelength (blue, green, yellow, red, violet)

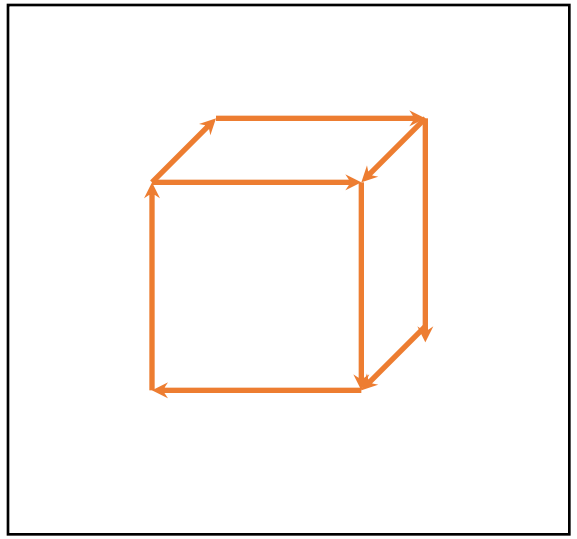
t – time

This is a *continuous* representation

- Not easily represented on a computer



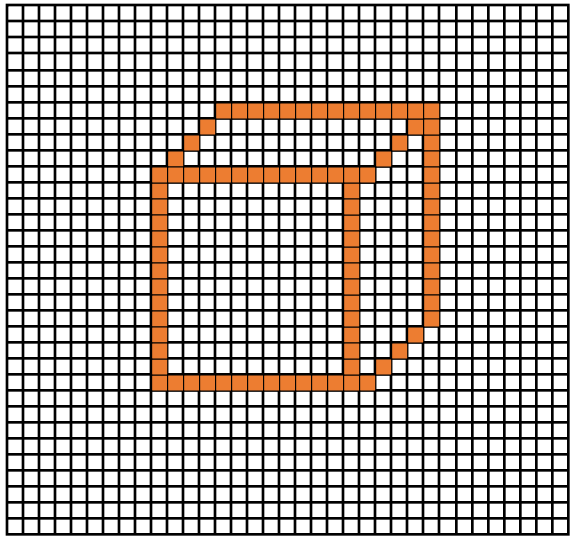
How do we represent images?



Vector Representation

+ arbitrary resolution
+ good for line drawings (text)

- may draw same point twice
- hard to do color changes



Raster Representation

+ good for color images
+ general purpose

- bounded resolution (aliasing)
- store EVERY pixel

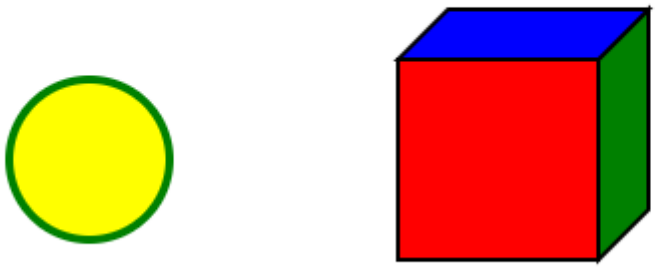
Home HTML HTML SVG Tryit: SVG circle https://www.w3schools.com/html/tryit.asp?filename=tryhtml_svg_circle

Run ›

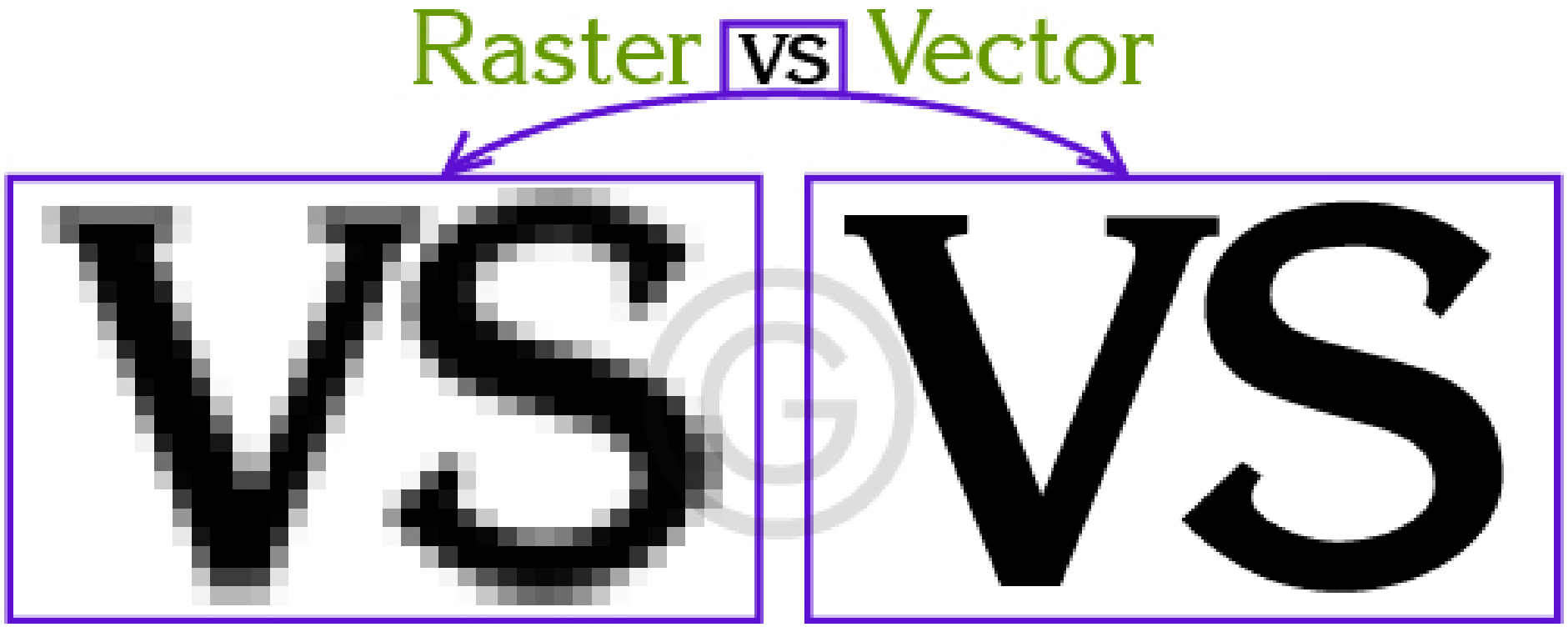
Result Size: 525 x 1565

Get your own website

```
<!DOCTYPE html>
<html>
<body>

<svg width="200" height="200">
  <circle cx="100" cy="100" r="40"
  stroke="green" stroke-width="4" fill="yellow" />
Sorry, your browser does not support inline SVG.
</svg>

<svg width="200" height="200" >
  <!-- 画前面的面 (红色) -->
  <polygon points="50,50 50,150 150,150 150,50" fill="red" stroke="black"
stroke-width="2"/>

  <!-- 画右边的面 (绿色) -->
  <polygon points="150,50 175,25 175,125 150,150" fill="green"
stroke="black" stroke-width="2"/>

  <!-- 画上面的面 (蓝色) -->
  <polygon points="50,50 150,50 175,25 75,25" fill="blue" stroke="black"
stroke-width="2"/>
</svg>

</body>
</html>
```

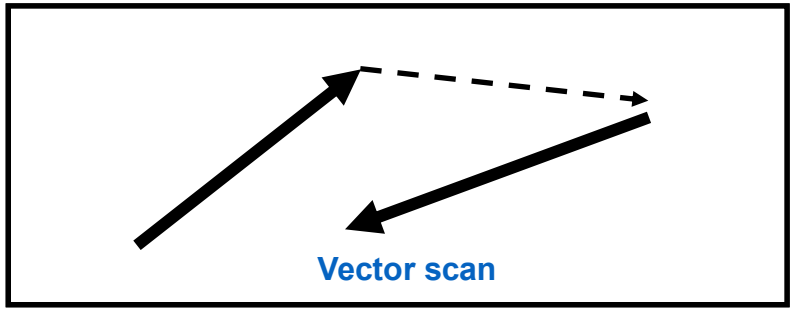


Raster *vs* Vector

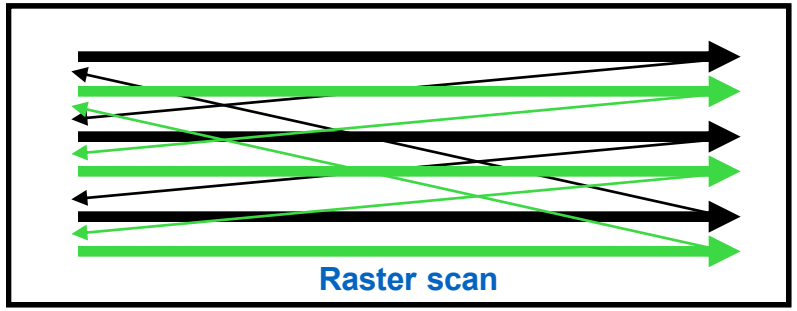


Raster *vs* Vector

- Early displays were vector displays
  - electron beam traces out line segments
  - image is a sequence of endpoints



- Raster displays (CRT TV's)
  - electron beam traces out a regular pattern: raster scan



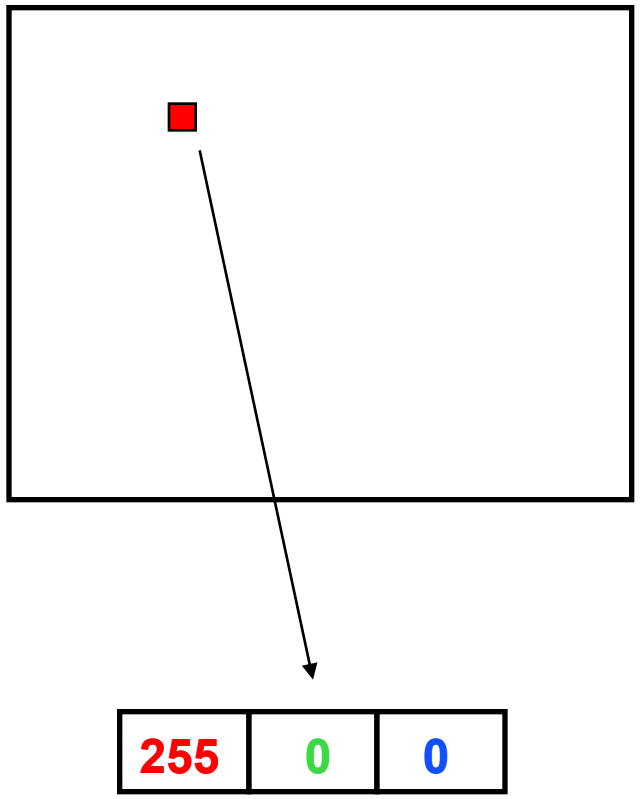
- Nowadays, (LCD's, OLED)
  - image is a raster: a 2D array of pixels

  - can display image of vector graphics by software (e.g., PDF)

Framebuffers for image processing and display

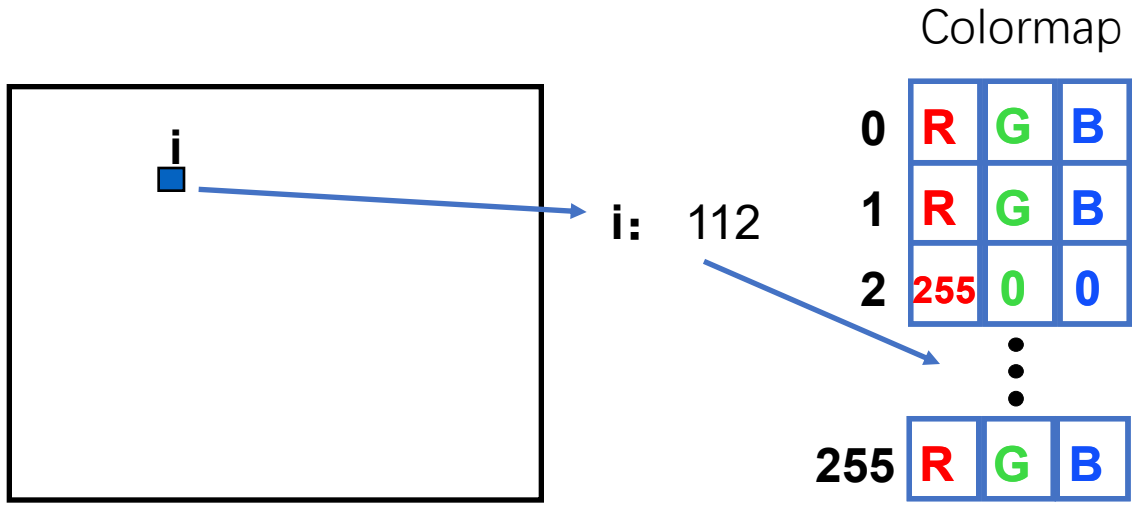
- image : a 2-D array of *pixels*.
- The value stored in each pixel controls its brightness
- The memory that stores the 2-D array of pixel values is called a *framebuffer*.
- The video hardware scans the framebuffer at ~60Hz
  - changes appear immediately
- Displays support different types of pixels
  - B&W displays: 1 bit/pixel *(bitmap)*.
  - Basic color displays: 8, 16, or 24 bits.
  - High-end displays: 96 or more bits.

Color Image Storage: Full-color (RGB)



- For 24 bit color:
  - store 8 bits each of red, green, and blue per pixel.
  - E.g. (255,0,0) is pure red, and (255, 255, 255) is white.
  - Yields 2^24 = 16 million colors.
- For 15 bit color:
  - 5 bits red + 5 green + 5 blue
- The video hardware uses the values to drive the R, G, and B guns.
- You can mix different levels of R, G, and B to get (almost) any color you want

Color Image Storage: Colormaps (LUT's)



- A single number (e.g. 8 bits) stored at each pixel.
- Used as an *index* into an array of RGB triples.
- With 8 bits per pixel, you get 256 colors of your choice
- Simple things to fill up color-maps with:
  - A gray ramp (for grayscale pictures)
  - A bunch of pre-chosen colors
  - A set of colors adaptively chosen for a given picture

Some picture file formats

BMP： Full color image

JPEG: Joint Photographic Experts Group Format

TIFF: Tagged-Image File Format

GIF: CompuServe Graphics Interchange Format

PPM: Portable PixMap Format (ASCII or binary)

EPS: Encapsulated PostScript Format (ASCII)

	BITS PER PIXEL	FILE SIZE	COMMENTS
JPEG	24	small	lossy compression
TIFF	8,24	medium	good general purpose
GIF	1,4,8	medium	popular, but 8-bit
PPM	24	big	easy to read/write
EPS	1,2,4,8,24	huge	good for printing

Others: BMP, XPM, RAS, PICT, PNG (Portable Network Graphic; transparency), OpenEXR, etc...

Some picture file formats

- Some frame buffers have 96 or more bits per pixel. What are they all for? We start with 24 bits for RGB.

- Alpha channel: an extra 8 bits per pixel, to represent "transparency." Used for digital compositing. That's 32 bits.

Image compositing

- Represent an image as layers that are composited (matted) together
- To support this, use pixel's extra *alpha* channel in addition to R, G, B
- Alpha is opacity: 0 if totally transparent, 1 if totally opaque
- Alpha is often stored as an 8 bit quantity; usually not displayed.

Image Alpha channel Result

Image compositing

- Mathematically, to composite a_2 over a_1 according to matte α

  $b = (1-\alpha) \bullet a_1 + \alpha \bullet a_2$

  α = 0 or 1 -- a hard matte, α = between 0 and 1 -- a soft matte

- Compositing is useful for photo retouching and special effects.
- Compositing is useful for translucent polygon rendering and volume rendering!

Deeper framebuffers

- Some frame buffers have 96 or more bits per pixel. What are they all for? We start with 24 bits for RGB.
  - Alpha channel: an extra 8 bits per pixel, to represent "transparency." Used for digital compositing. That's 32 bits.
  - A Z-buffer (or depth buffer), used to hold a "depth" value for each pixel. Used for hidden surface 3-D drawing. 16 bits/pixel of "z" brings the total to 48 bits.
  - Double buffering:
    - For clean-looking flicker-free real time animation.
    - Two full frame buffers (including alpha and z).
    - Only one at a time is visible—you can toggle instantly.
    - Draw into the "back buffer" (invisible), then swap.
    - Can be faked with off-screen bitmaps (slower.)
    - 2 x 48 = 96.

Image Processing

Image processing

- *Point Processing*: modify each pixel as a function of its pixel value
- *Filtering*: output is a function of the (usually) local neighborhood around the pixel
- Other topics:
  - Image transformation (resize, warp)
  - Image compression (lossless, lossy)
  - Image encoding and decoding (for efficient transmission)
  - Image segmentation
  - Image matting
  - Image understanding (going all the way to computer vision)
  - Video processing (extending to a sequence of images)
  - …

Point processing

- Input: a[x,y], Output b[x,y] = f(a[x,y])
- f transforms each pixel value separately
- Useful for contrast adjustment (recall gamma correction?)

Suppose our picture is grayscale (a.k.a. monochrome).

Let v denote pixel value, suppose it's in the range [0,1].

$f(v) = v$ identity; no change

$f(v) = 1-v$ negate an image
(black to white, white to black)

$f(v) = v^p, p<1$ brighten

$f(v) = v^p, p>1$ darken

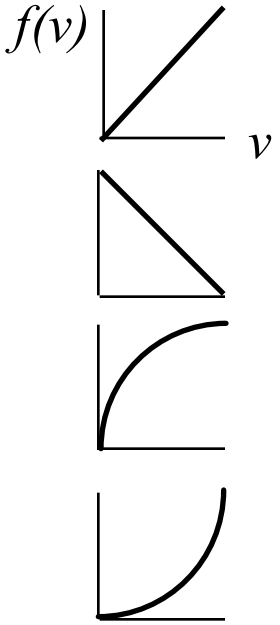


Image Processing

Dithering and quantization

Image filtering

Image inpainting

Matting

…

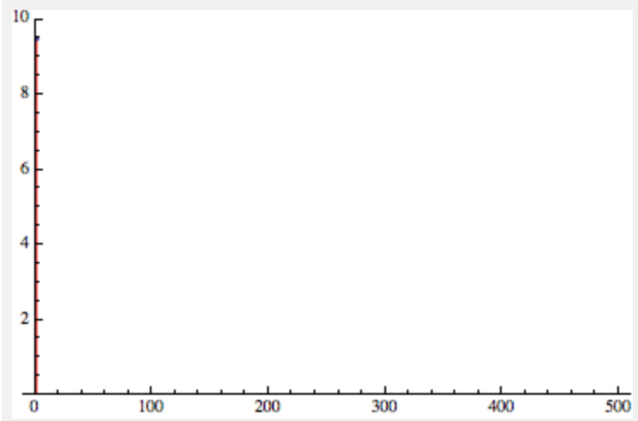
How to draw grayscale on a limited-bit screen?

If image and display have the same resolution:

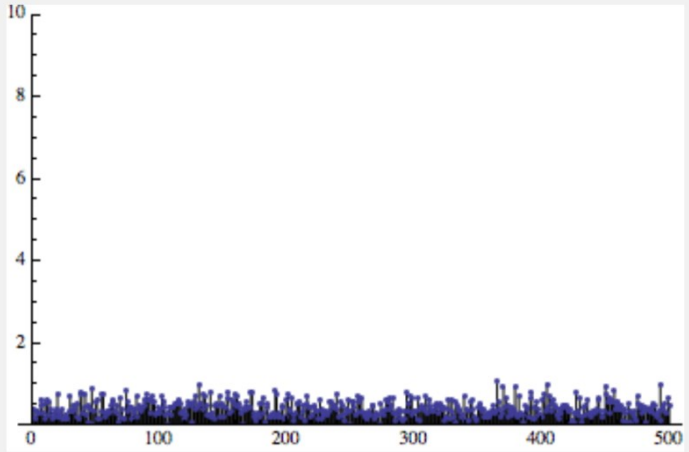
- The values of the input image's pixels are normalized in floating point format to [0, 1] (or [0,255]) with 0 (black) and 1 (or 255, white).
- Scan in raster order.
- At each pixel, draw the least-error output value (round off on 0.5 or 128)

Dithering with noise

Adding noise : Intentionally / deliberately adding some noise to signal to prevent large-scale / low resolution errors that come from quantization or under-sampling.



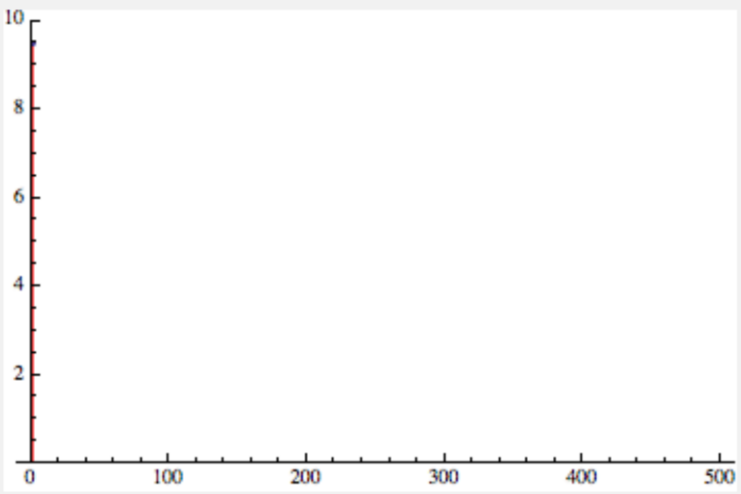
Red plot/spike = frequency spectrum of error when not using dithering (constant, no frequencies). Black – with white noise dithering.



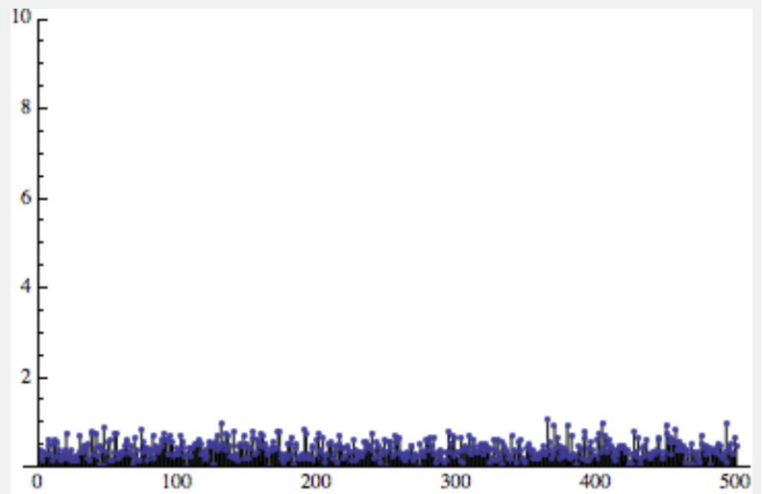
Red plot/spike = frequency spectrum of error when not using dithering (constant, no frequencies). Black – with white noise dithering.

Dithering with noise

- Adding noises to signals could decrease the systematic bias.
- Let's analyze the quantization error in the frequency domain:



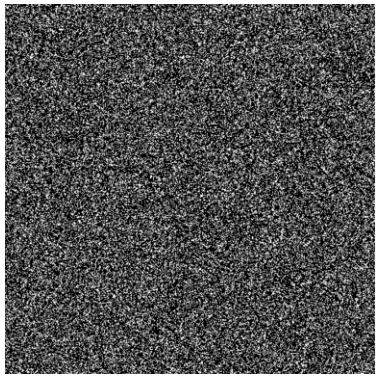
Red plot/spike = frequency spectrum of error when not using dithering (constant, no frequencies). Black – with white noise dithering.



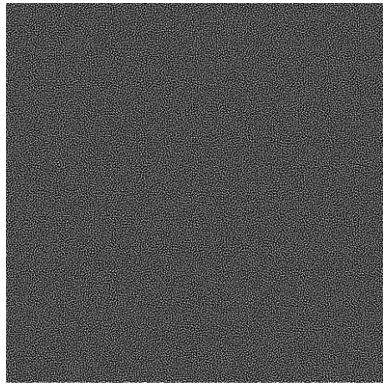
Red plot/spike = frequency spectrum of error when not using dithering (constant, no frequencies). Black – with white noise dithering.

- Before dithering, the error lies at the lowest frequency, while adding noise could decrease the low frequency error.

Dithering with white noise

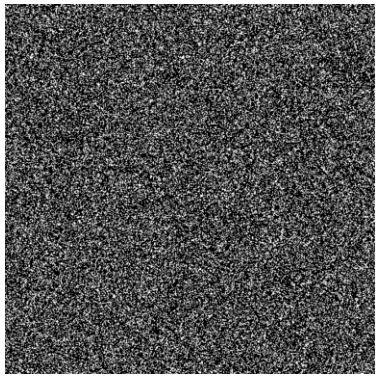


Random noise

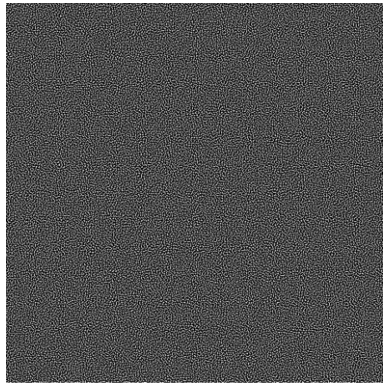


Blue noise

Dithering with blue-noise



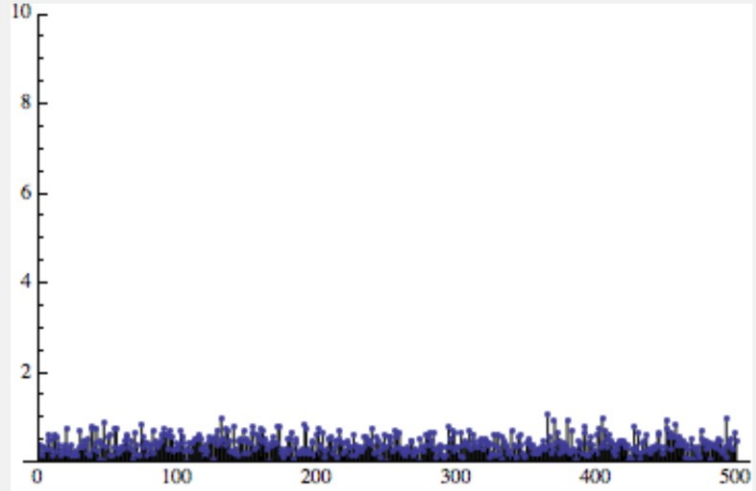
Random noise



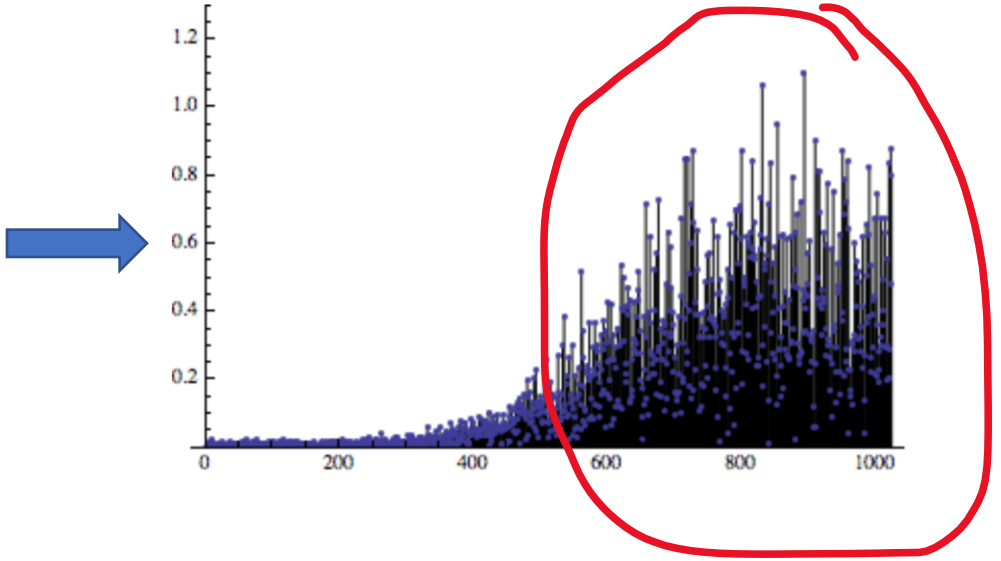
Blue noise

Human perception

- Humans perceive medium scale of detail much better than very high or very low frequencies.
- If we could add noises to make the errors at lower frequencies be smaller, we could achieve better results.



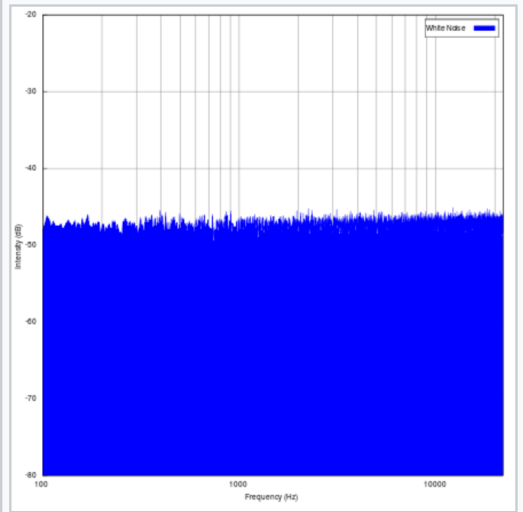
Red plot/spike = frequency spectrum of error when not using dithering (constant, no frequencies). Black – with white noise dithering.



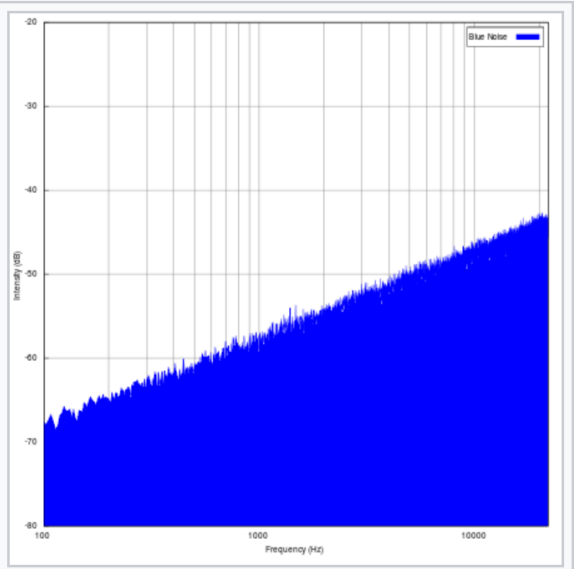
We are not good at capturing high-frequency errors

White and Blue Noise

- White noise is a signal (or process), named by analogy to white light, with a flat frequency spectrum when plotted as a linear function of frequency .
- Blue noise is also called azure noise. Blue noise's power density increases 3.01 dB per octave with increasing frequency over a finite frequency range.[5] In computer graphics, the term "blue noise" is sometimes used more loosely as any noise with minimal low frequency components and no concentrated spikes in energy.



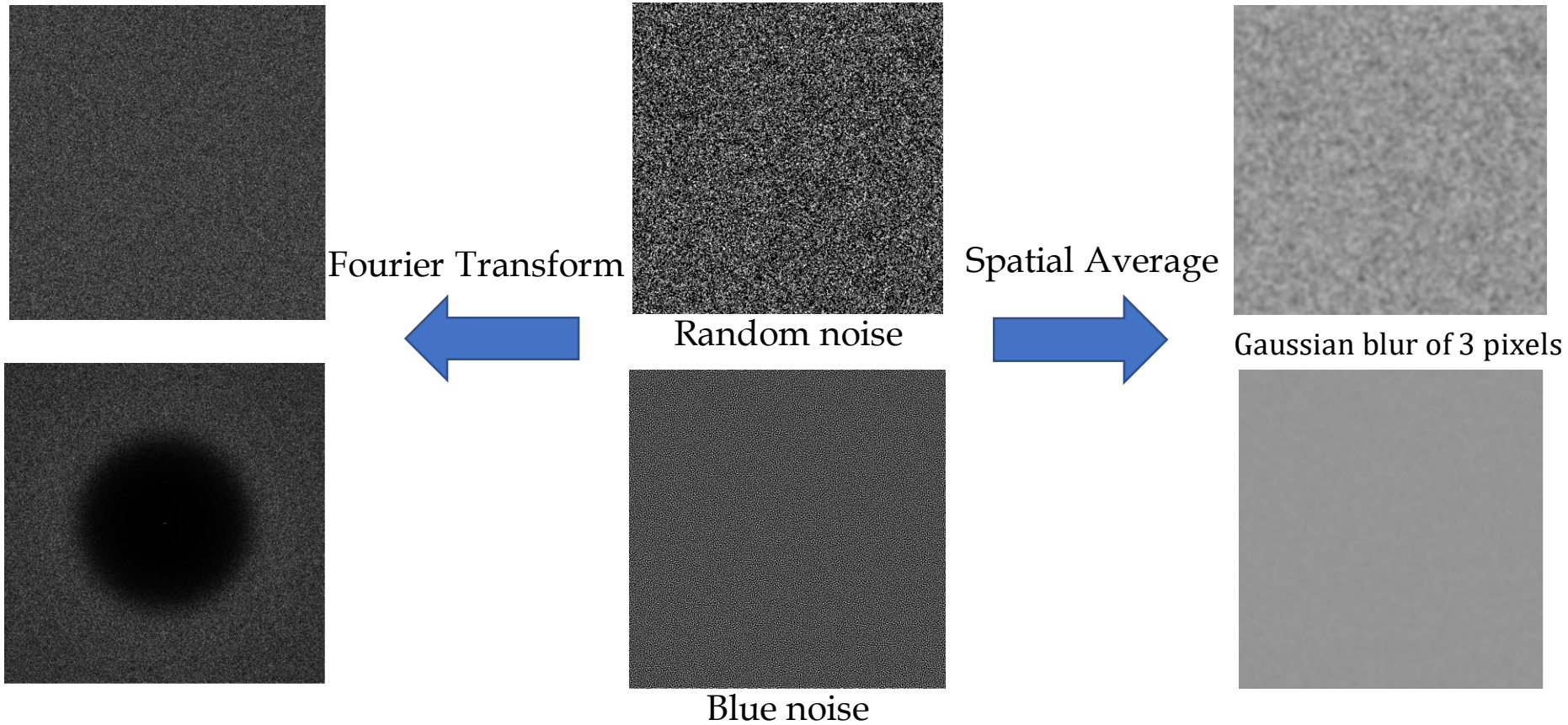
White noise spectrum. Flat power spectrum. (logarithmic frequency axis)



Blue spectrum (+3.01 dB/octave)

Blue Noise

- Blue noise is a good choice for dithering.
  - Has minimal low-frequency components
  - Signals dithered by blue noise has smaller low-frequency errors
- If we plan to draw a grey image only with white and black color...



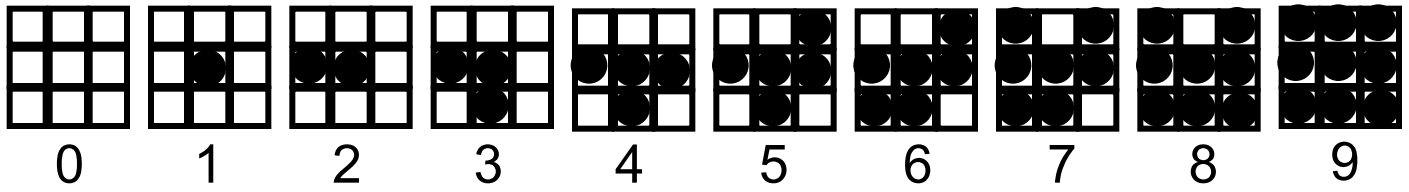
Dithering in space

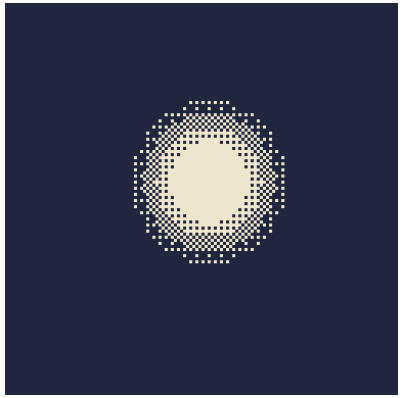
If display has higher resolution than that of the image:

- **E.g., display full color on an 8-bit screen, or print on paper using black-and-white ink (1-bit)**

- Basic idea: give up *spatial* resolution in return for greater *brightness* or *color* resolution
- The eye does *spatial averaging*, so present a pattern whose *average* color matches the color you want
- **Ordered dithering**: in the patterns below, each square is either black or white.
  - From far away, the eye sees the average brightness of each grid, not the individual squares.
  - The average brightness of each 3x3 grid depends on the number of black and white squares—you can get ten distinct brightness levels ranging from black to white.
  - To draw a grayscale picture, each input pixel is represented by an output pattern. The pattern of dots that gets drawn depends on the input pixel value.

you can also dither in time!



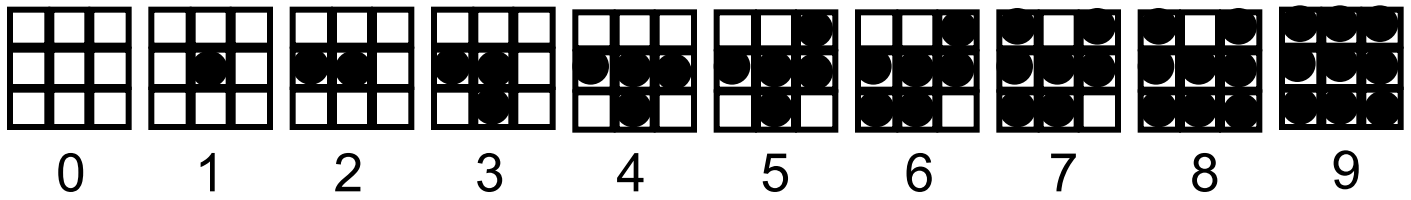


Dither in space: ordered dithering

- How do we select a good set of patterns
- Pick patterns that avoid annoying artifacts:
  - Constant-brightness regions should not have obvious stripes.
  - On many devices (e.g. laser printers) isolated pixels should be avoided.
  - Growth-sequence: pixels that are "on" at one brightness levels should remain on for all higher levels. This avoids contouring artifacts.
- The full set of dot patterns can be encoded in a single n x n *dither matrix*. Each element in the matrix is a threshold: the dot is turned on for input values greater than the threshold. A sample 3x3 dither matrix is

dither matrix

6	8	4
1	0	3
5	2	7



Dithering in space

If display has the same resolution as the image:

dither matrix M

6	8	4
1	0	3
5	2	7

- Cut the image into 3x3 Pixel Blocks
- If Pixel value > corresponding value in the dither matrix (normalized to 0-1, e.g. M/9) , set as 1, else set as 0.
- E.g. Pixels:

0.5	0.5	0.5
0.5	0.5	0.5
0.5	0.5	0.5

Quantized:

0	0	1
1	1	1
0	1	0

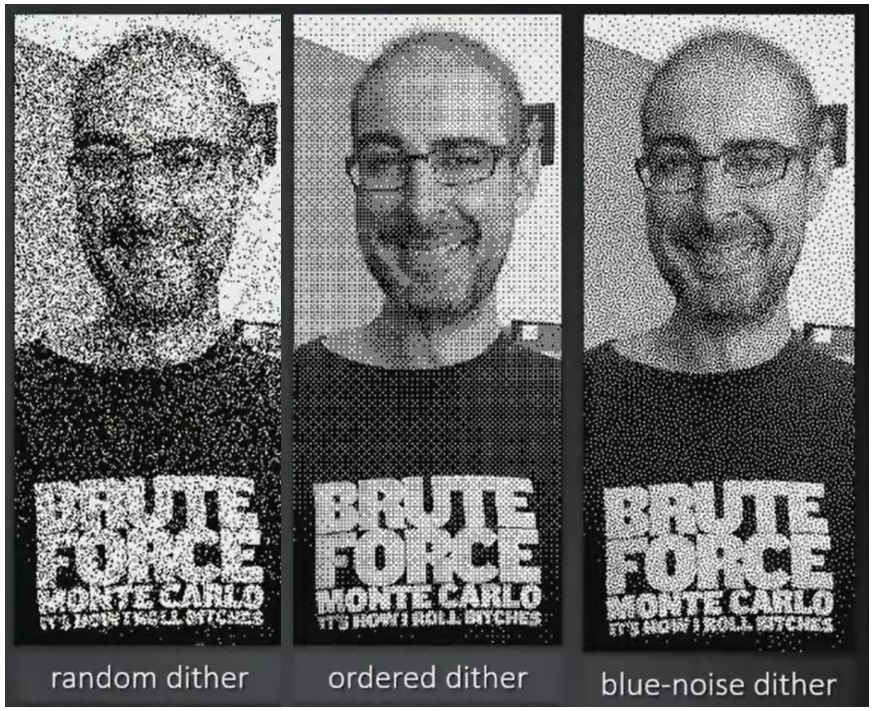
0.6	0.5	0.4
0.6	0.5	0.4
0.6	0.5	0.4

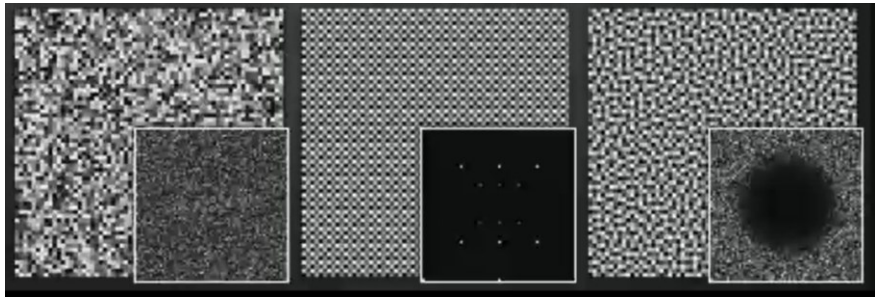
0	0	0
1	1	1
1	1	0

Ordered dithering

Comparison


BRUTE FORCE
MONTE CARLO
IT'S HOW I ROLL BITCHES
BRUTE FORCE
MONTE CARLO
IT'S HOW I ROLL BITCHES
constant (0.5)


random dither
ordered dither
blue-noise dither

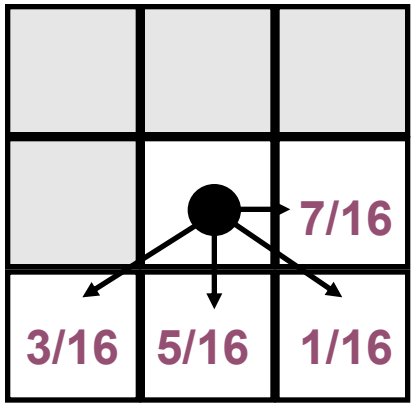


Dithering with error diffusion (same resolution)

- The values of the input image's pixels are normalized in floating point format to [1, 1] (or [0,255]) with 0 (black) and 1 (or 255, white).
- Scan in raster order.
- At each pixel, draw the least-error output value (round off on 0.5 or 128)

Floyd-Steinberg Error Diffusion

- Divide the error into 4 (uneven) chunks.
- Add the error chunks back into the input values, at the 4 neighboring pixels you haven't hit yet:
- Can alternate scan direction



Dithering with error diffusion (same resolution)

Original image

After Floyd-Steinberg dithering

Color dithering

- You can mix Red, Green, and Blue to get any color you like.
- If you have an RGB image and a 3bit display, 1 per color, you just dither R, G, and B separately.
- On an 8 bit display, you can use the color map to divide the 8 bits into three parts (3, 3, 2) for R, G, and B. (Blue gets shortchanged because we can't see blue very well.) So you get 8 levels each for R and G, and 4 for B.
- Dither R, G, and B separately (Floyd-Steinberg works fine for multi-bit output,) assemble the results into an 8-bit byte, and write to the frame buffer.
- The results generally look excellent, particularly on a high-res monitor.

More on dithering:

http://www.iro.umontreal.ca/~ostrom/publications/research.html#halftoning

Color dithering: An example

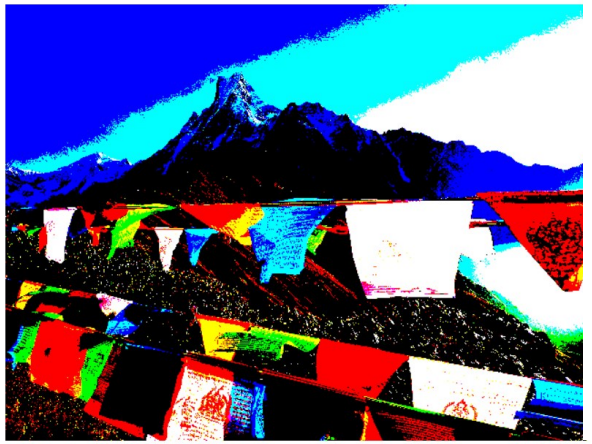
Original Image

Shown on a 8-bit monitor

After Floyd-Steinberg dithering

Color dithering: More examples

8 bits /color channel



Quantized to 1 bit /color channel

Using white noise to dither before quantization

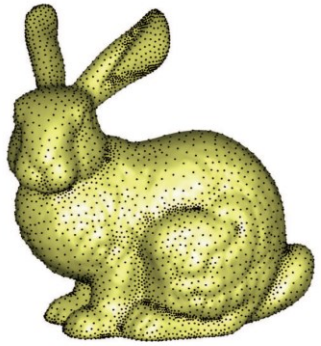
Using blue noise to dither before quantization

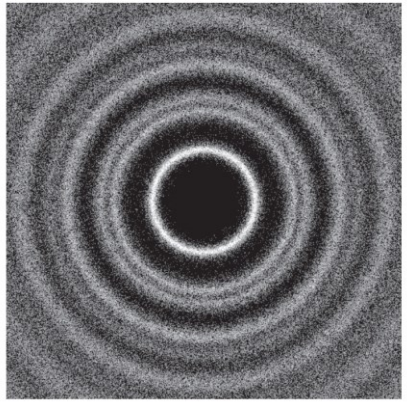
Blue Noise: a spatial view

- Blue noise has many applications in computer graphics such as geometry processing, simulation and rendering, etc., almost **anywhere sampling is done!**
- How to generate blue noise in different scenarios with different properties is a big subject! (ref. Penrose tiling, Wang tiling)

Victor Ostromoukhov

Roger Penrose





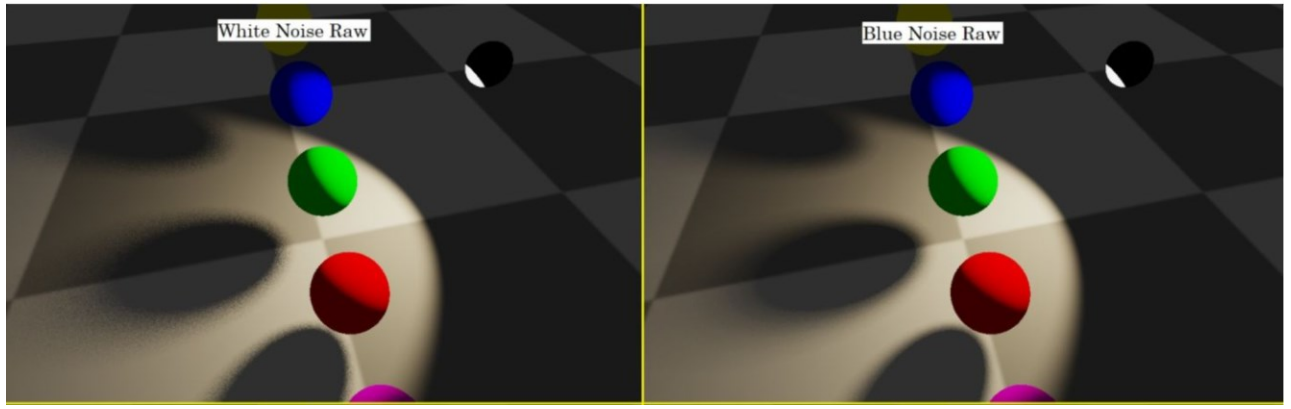


Image Processing

Dithering and quantization

Image filtering

Image inpainting

Matting

…

Image filtering: blurring

Original, 64x64 pixels

3x3 blur

5x5 blur

Examples: blurring filter

A simple blurring effect can be achieved with a 3x3 filter centered around a pixel, written explicitly:

b[x,y] = (a[x-1,y-1] + a[x,y-1] + a[x+1,y-1]
+a[x-1,y] + a[x,y] + a[x+1,y]
+a[x-1,y+1] + a[x,y+1] + a[x+1,y+1]) / 9

or as coefficient matrix:

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

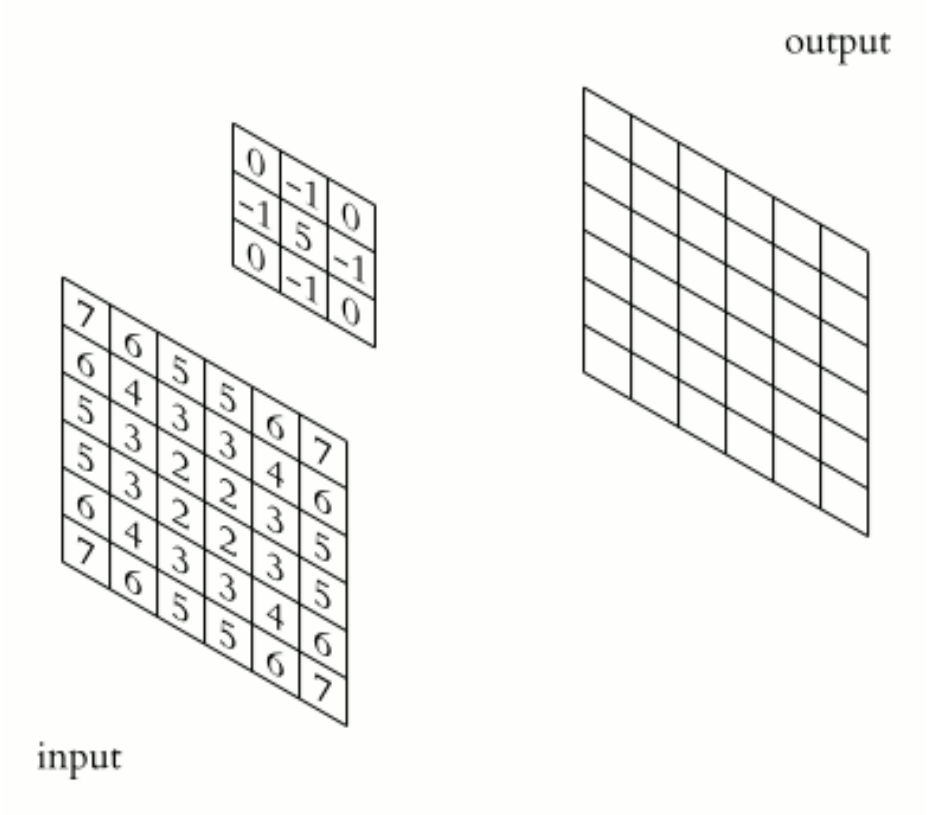
More blurring is achieved with a wider $n \times n$ filter:

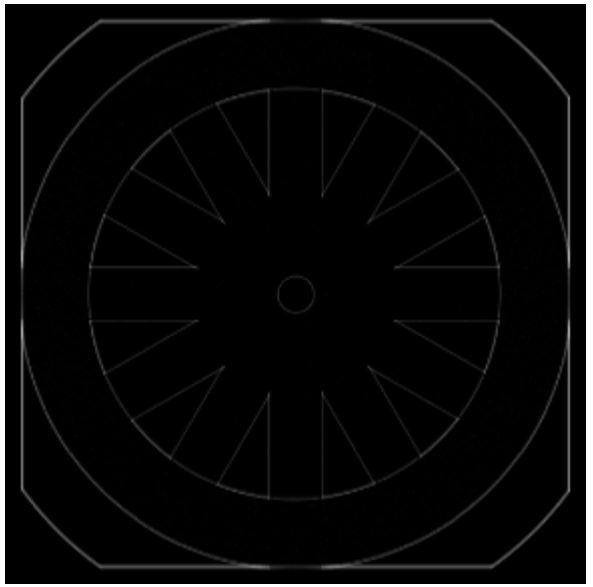
$$\frac{1}{n * n}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

Filtering and convolution

- f: input signal;
  h: output signal;
  g: filter / kernel

$$h(t) = f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$





Examples: edge filter

$$b[x,y] = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} a[u,v]h[x-u, y-v]$$

a[x,y] = input signal

b[x,y] = output signal

h[x,y] = 3x3 filter

x,y take on only integer vals

$\frac{1}{4}$ ×

1	0	-1
2	0	-2
1	0	-1

h

(0,0) at center

0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25

a

0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0

b

Image filters

- In 1-D such a simple filter can be written:

$$b[x] = \sum_{t=-\infty}^{+\infty} a[t]h[x-t]$$

where a[x] = input signal
b[x] = output signal
h[x] = filter
x takes on only integer values

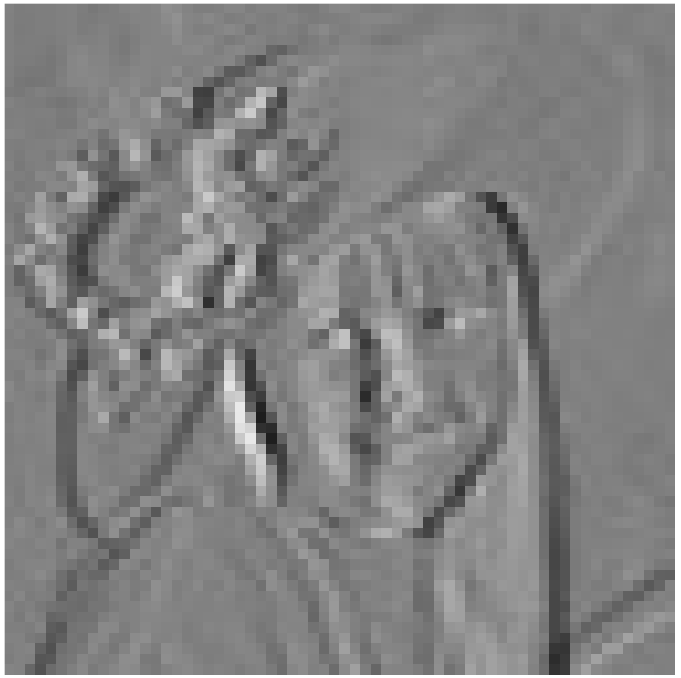
- This is convolution, written b = a ⊗ h for short. Convolution is commutative, i.e. a ⊗ h=h ⊗ a
- 2-D is similar, but with a double-summation:

$$b[x,y] = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} a[u,v]h[x-u,y-v]$$

- This class of filters is called “linear, shift-invariant”

Image filtering: edge detection

Original, 64x64 pixels



horizontal derivative

vertical derivative

Examples: edge filter

Gradient and its magnitude:

$$\nabla a = \left(\frac{\partial a}{\partial x}, \frac{\partial a}{\partial y}\right), \quad |\nabla a| = \sqrt{\left(\frac{\partial a}{\partial x}\right)^2 + \left(\frac{\partial a}{\partial y}\right)^2}$$

Sobel edge filter uses these weights (integer approximation):

$$\frac{\partial}{\partial x} \Rightarrow \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}, \qquad \frac{\partial}{\partial y} \Rightarrow \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

This is a *nonlinear filter* because of the sqrt and square operations.

Original a(x,y)

After edge filtering

$$\|\nabla a[x,y]\| = \sqrt{\left(\frac{\partial a[x,y]}{\partial x}\right)^2 + \left(\frac{\partial a[x,y]}{\partial y}\right)^2}$$

Image Processing

Dithering and quantization

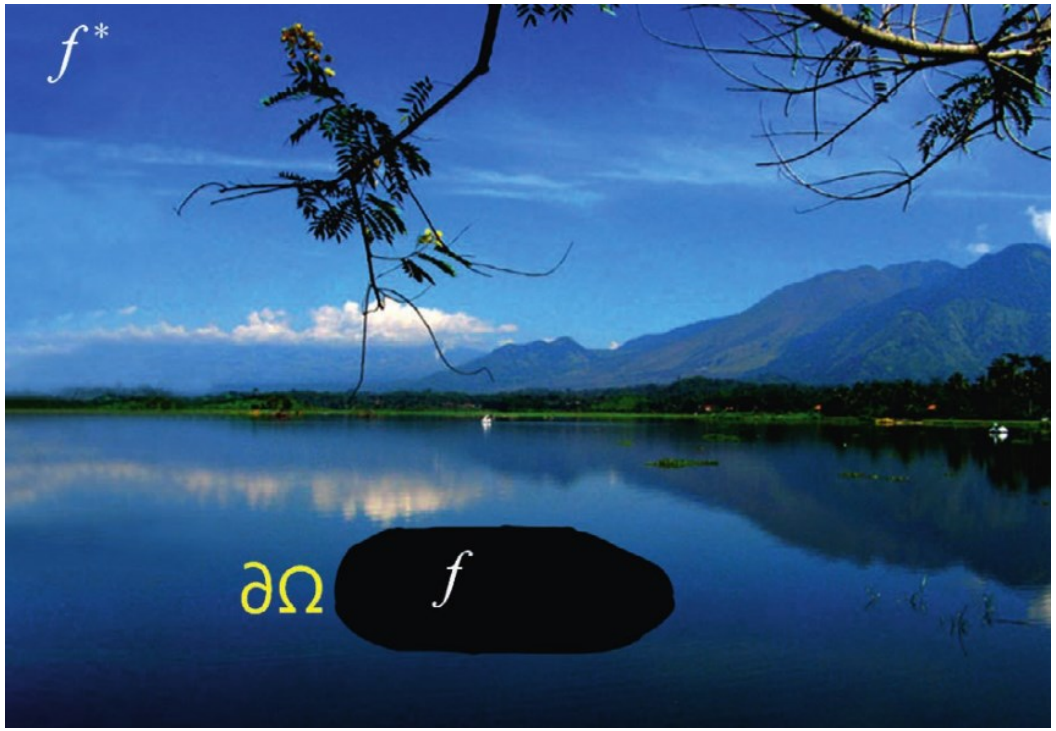
Image filtering

Image inpainting

Matting

…

Image completion (inpainting)



Input image

Output (inpainted) image

Image completion (inpainting)

- Fill in some regions of an image f^* whose information is lost, and make it as plausible as possible.
- "Plausible" means as smooth as possible.
  - Spatial locality: The neighboring parts of the same object should be similar.
  - Occam's Razor: Nothing should exist unless inducted from the remaining parts of the image.
  - Gradients are measurement for changes; a smooth region should have as small gradient as possible.

    Integrates L2 norm of the gradient over the region:

$$\arg\min_{f} \iint_{\Omega} \| \nabla f \|^2, \quad \text{s.t.} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

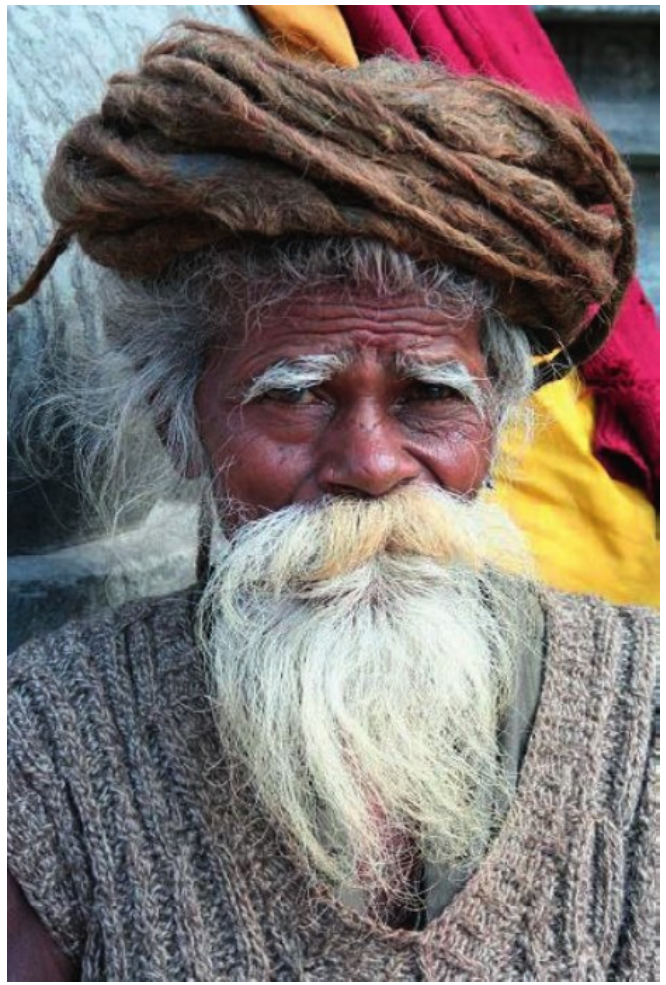
$f(x, y)$ is the pixel value at (x, y)

Laplace editing!

Image completion (inpainting)

Input image

Inpainted image



Original image

Image cloning/insertion

Image cloning/insertion

Poisson Editing

- How to "inject" some shapes or information to the region?
- Altering the optimization goal:

$$\arg\min_{f} \iint_{\Omega} \| \nabla f - \mathbf{g} \|^2 \quad \text{s.t.} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

- **g** is the function that "guides" the completion, and is usually the gradient of another picture.
- Laplace editing (Image completion) is the special case of Poisson editing where **g**=0 and the region has only one pure color.
- Details in Notes P66

Poisson Editing

- Variational interpretation

$$f^* = \arg\min_f \iint_\Omega \|\nabla f - \mathbf{g}\|^2 \quad \text{s.t} f^*|_{\partial\Omega} = f|_{\partial\Omega}$$

Euler Equation: $F_f - \frac{\partial}{\partial x} F_{f_x} - \frac{\partial}{\partial y} F_{f_y} = 0$

$$\Delta f = div(\mathbf{g}) \quad \text{s.t} f^*|_{\partial\Omega} = f|_{\partial\Omega}$$

https://en.wikipedia.org/wiki/Euler%E2%80%93Lagrange_equation

$$\min J[f] = \int_a^b F(f, f', x)\, dx \qquad \frac{\partial F}{\partial f} - \frac{d}{dx}\left(\frac{\partial F}{\partial f'}\right) = 0$$

Gradient

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Divergence

$$div(f(\mathbf{x})) = \left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right] \cdot f = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}$$

Laplacian ($\Delta = \nabla^2 = \nabla \cdot \nabla$)

$$\nabla \cdot \nabla f(\mathbf{x}) = \left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right] \cdot \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Poisson Editing

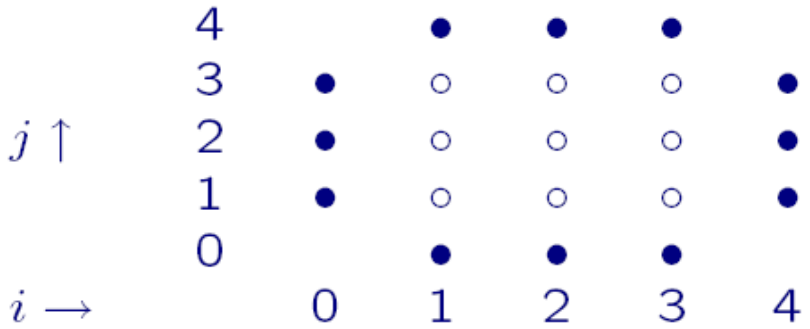
- Poisson equation

$$\Delta f = -\rho$$

$$\Delta \equiv \frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y}$$

$$\rho = \rho(x, y)$$

Continuous form



$$\Delta f = -\rho$$

$$\frac{f_{i+1,j} + f_{i-1,j} - 2f_{i,j}}{h^2} + \frac{f_{i,j+1} + f_{i,j-1} - 2f_{i,j}}{h^2} = \rho_{i,j}$$

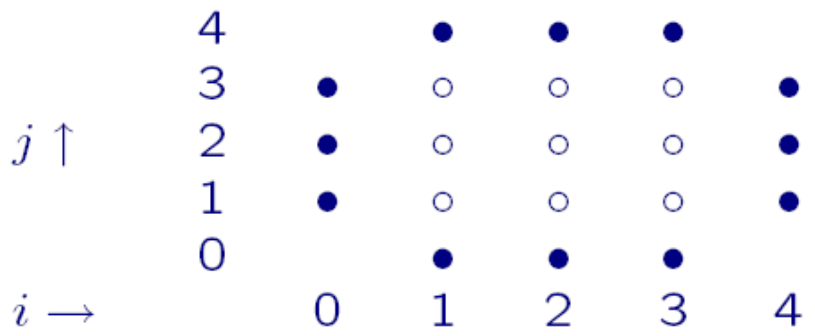
Discrete form

Poisson Editing

- Poisson equation $\Delta f = -\rho$

$$\Delta \equiv \frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y} \qquad \rho = \rho(x, y)$$

$$\frac{f_{i+1,j} + f_{i-1,j} - 2f_{i,j}}{h^2} + \frac{f_{i,j+1} + f_{i,j-1} - 2f_{i,j}}{h^2} = \rho_{i,j}$$



$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} f_{1,1} \\ f_{2,1} \\ f_{3,1} \\ f_{1,2} \\ f_{2,2} \\ f_{3,2} \\ f_{1,3} \\ f_{2,3} \\ f_{3,3} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ b_{1,3} \\ b_{2,3} \\ b_{3,3} \end{bmatrix}$$

A sparse linear system

Poisson Editing

- Solving Poisson equation → solving linear equations

- Solve $Ax = b$, where A is a sparse matrix
  - Direct method
  - Iterative method
    - Jacobi, Gauss-Seidel, SOR
  - Multi-grid method

Poisson Editing

Image Processing

Dithering and quantization

Image filtering

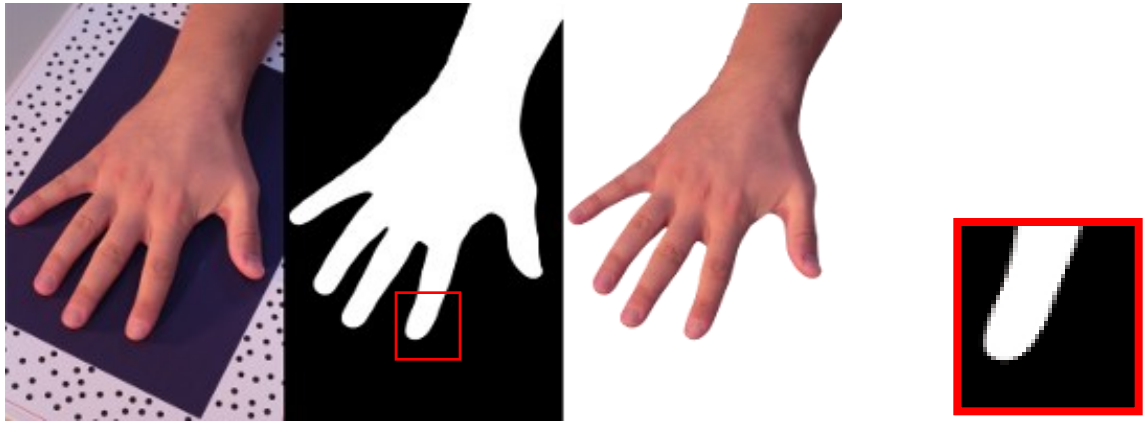
Image inpainting

Matting

…

Image processing: segmentation and matting

- Image segmentation (hard separation, ‘ 0’ or ‘1’)



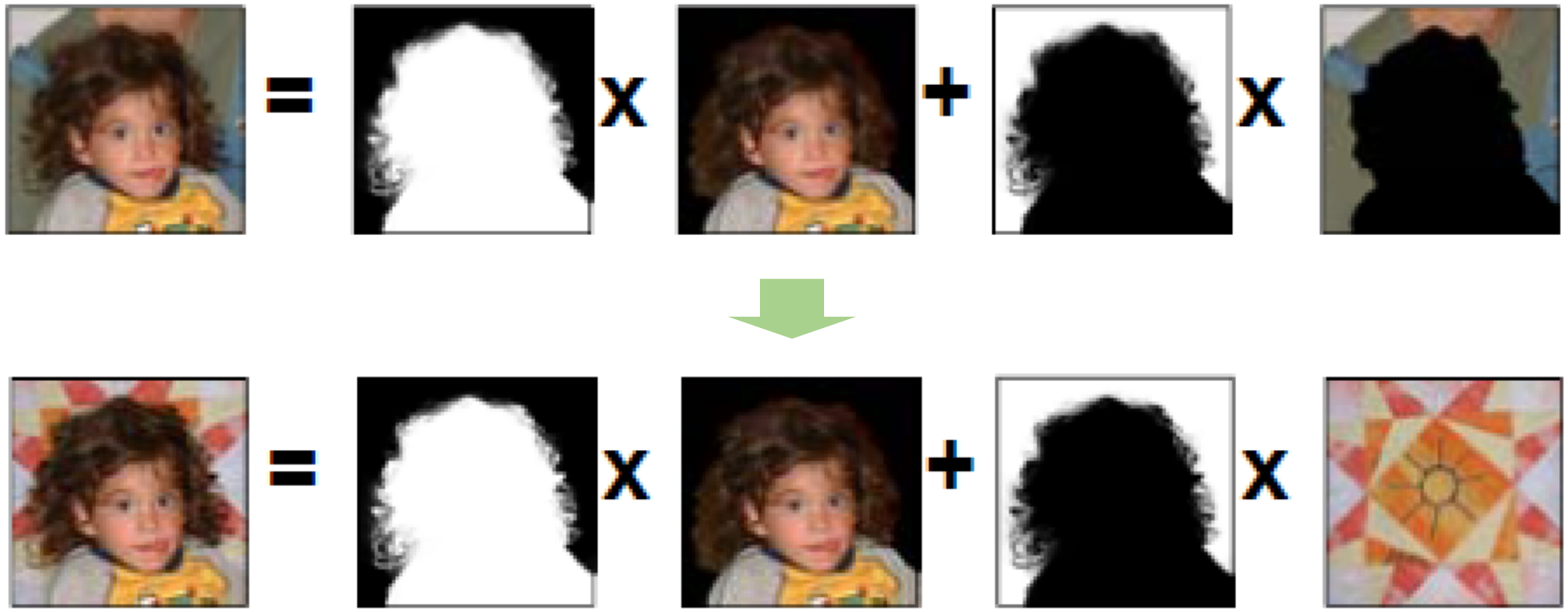
- Matting (soft segmentation)

Image matting equation

- inverse problem of image composition:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i$$

I: Image
F: Foreground
B: Background
α: Alpha-channel



Why is matting difficult?

- ill-posed problem

$$I_i = \alpha_i F_i + (1-\alpha_i) B_i$$

$$I_i^R = \alpha_i F_i^R + (1-\alpha_i) B_i^R$$

$$I_i^G = \alpha_i F_i^G + (1-\alpha_i) B_i^G$$

$$I_i^B = \alpha_i F_i^B + (1-\alpha_i) B_i^B$$

Solution 1: blue (green) screen matting

- Assume that the background only has blue (green) color and there is no blue (green) in the foreground.
- Note that the lightness could change, the background is not always (0,0,1), it could be (0,0,0.5)

$$F_b = 0, B_r = B_g = 0$$

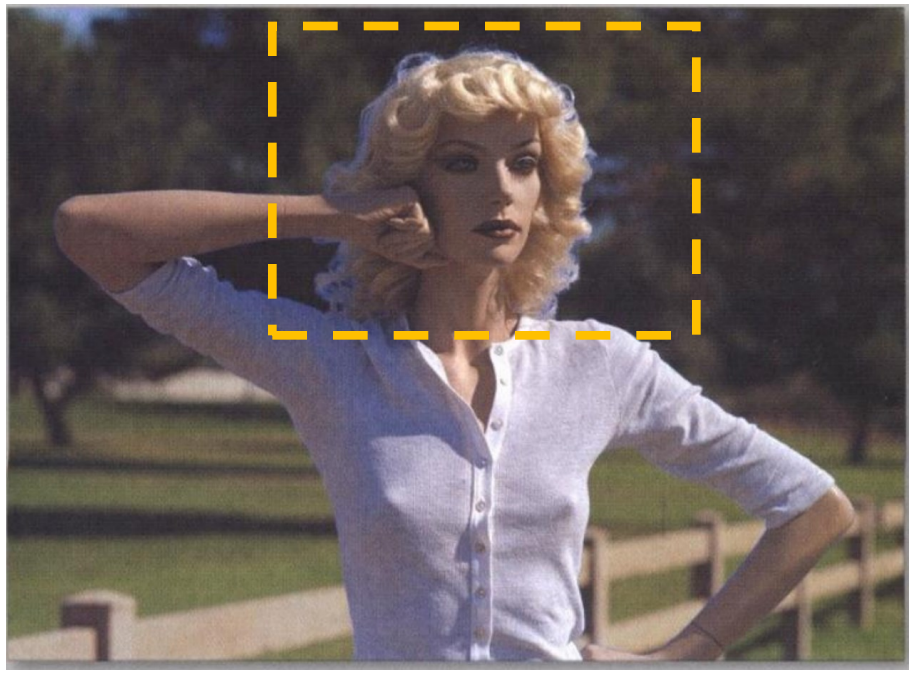
$$C_r = \alpha F_r$$

$$C_g = \alpha F_g$$

$$C_b = (1-\alpha) B_b$$

Blue screen matting

- Downsides:
  - The assumption that the foreground cannot have blue colors is restrictive.
  - Blue/Green Spilling: light reflected off the background hits the foreground, making it be blue/green



Green screen in Star Wars (2005)