ECE 420

Lecture 5

Feb 18 2019

Now Entering

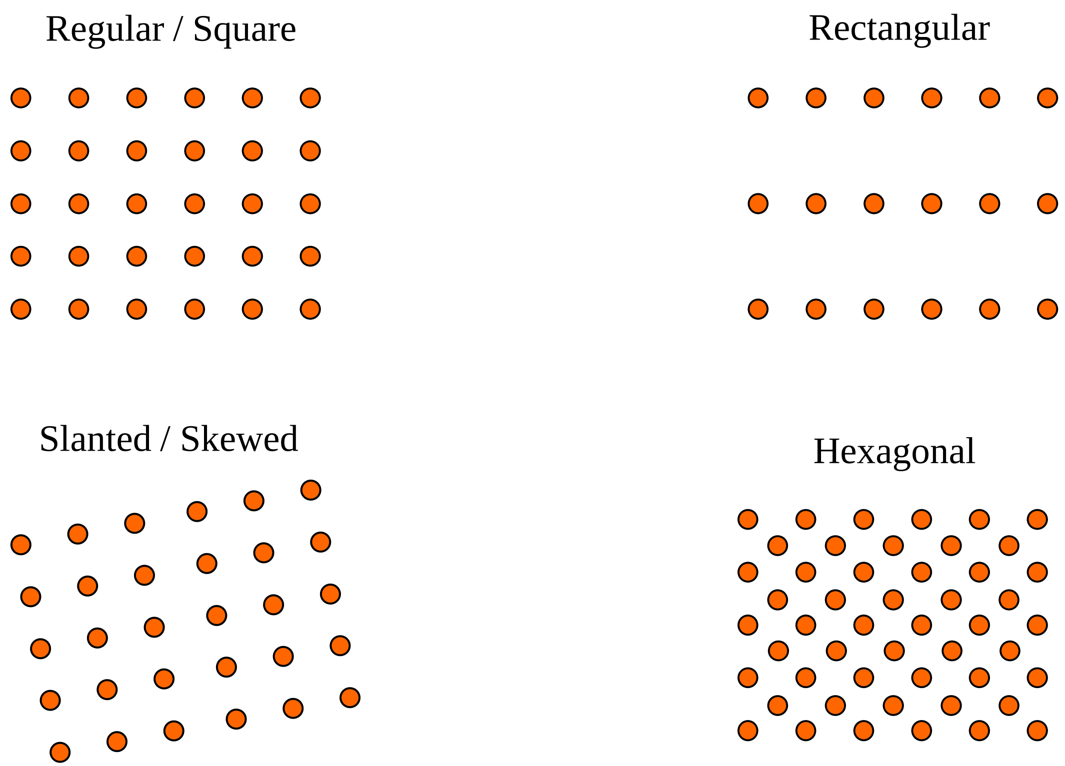
The Second Dimension!

2D Signal Processing

- Manipulation of samples of a bivariate function
  - $f(x, y) \rightarrow f[n, m]$ or $f[i, j]$ or $x[n, m]$ or ???
  - Unfortunate namespace collision with usual function names x, y
  - Pick whatever you wish but just be consistent with your notation!
- All the usual 1D operations/theorems apply as usual along each dimension/axis
  - Sampling
  - Filtering
  - Rate changing
  - Interpolation
  - …
- There are also true 2D extensions of the above!

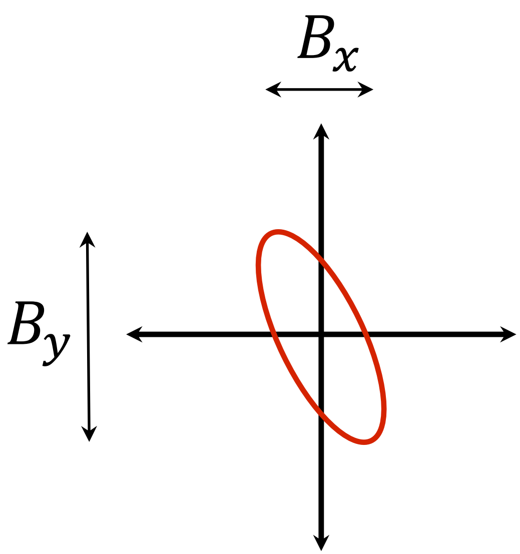
2D Sampling

- Many more options in sampling patterns

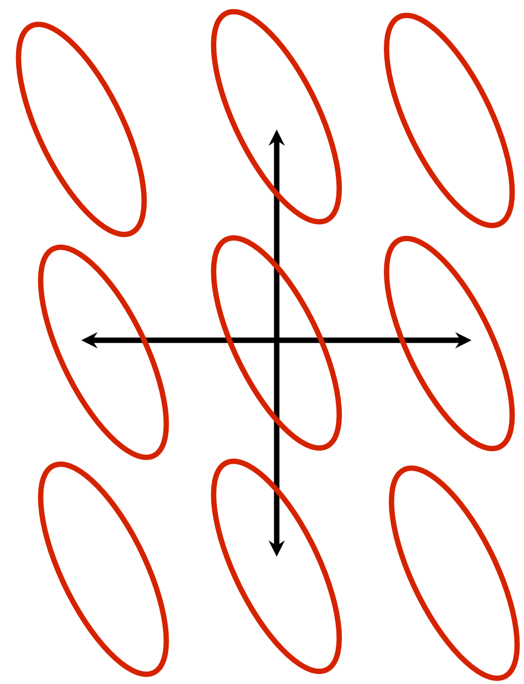


Sampling and 2D Fourier Transforms

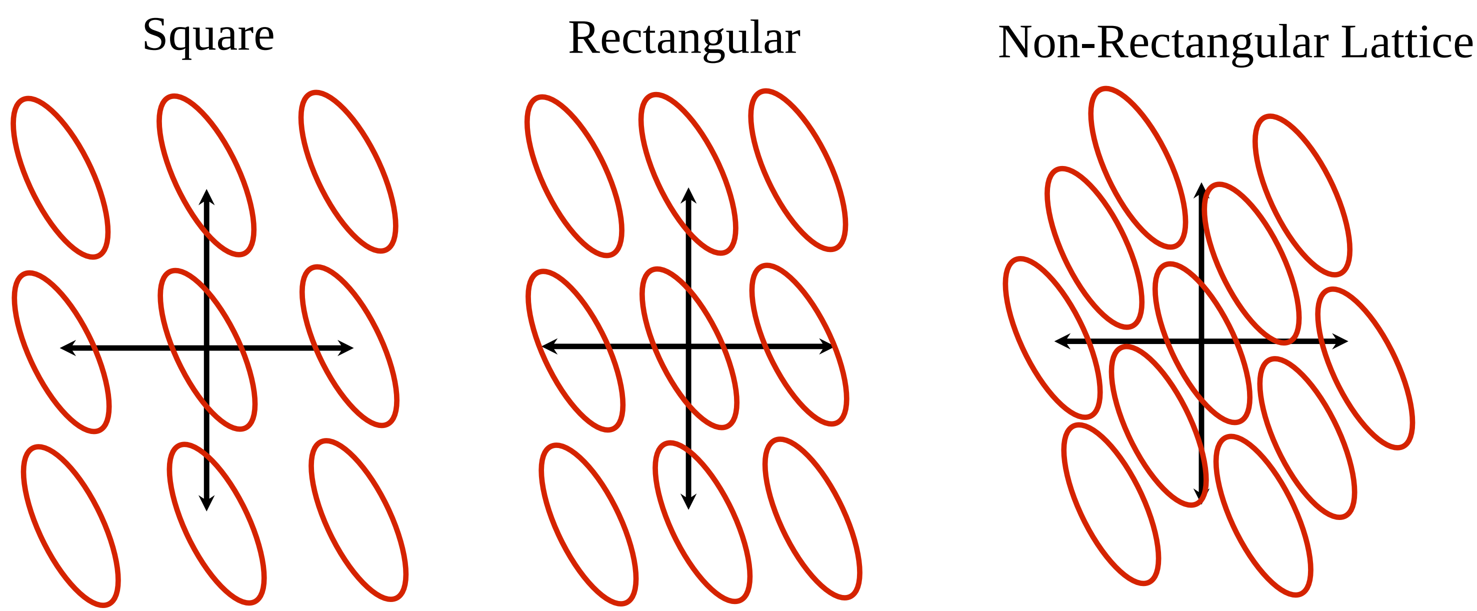
- 2D Fourier transform is cascade of transforms along each dimension
  - $G(\Omega_x, \Omega_y) = F_y\{\, F_x\{\, g(x, y)\,\}\,\}$
- The essential support of $G(\Omega_x, \Omega_y)$ determines how it must be sampled



Square sampling based on $\max(B_x, B_y)$



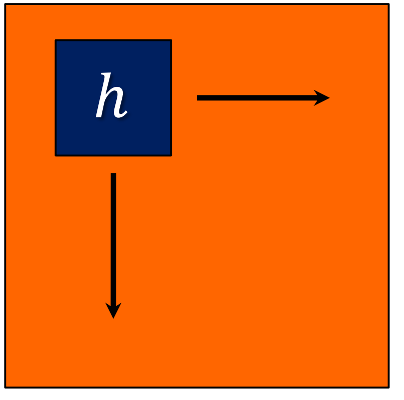
Sampling and 2D Fourier Transforms



- The better packed the spectra, the more efficient our sampling scheme is
  - Sampling efficiency = fewer samples = more performant algorithms!
- However, processing on non-rectangular lattices can be tricky
  - Unless there is an application-specific reason to do so, probably will be working with uniform sampling in both directions

2D Convolution

- Recall 1-D convolution
  - $y[n] = \sum_m x[m]h[n-m]$
- 2D Convolution is a multi-dimensional generalization of this
  - $y[i,j] = \sum_m \sum_n x[m,n]h[i-m,j-n]$
- h typically referred to as the filter kernel
- Same idea of a weighted average of elements over a sliding window
- Applications usually have entire 2D samples available, so "non-causal" h are typical
  - Centered, zero phase filters are also common



Separable Convolution

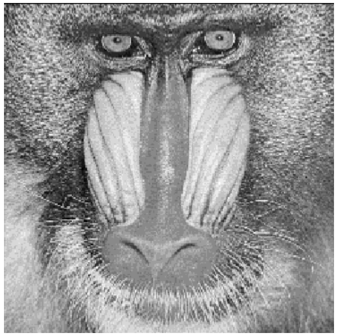
- h is defined as separable if it can be factored
  - $h[n,m] = h_x[n]h_y[m]$
- Rewrite convolution as cascade of 1D convolutions
  - $y[i,j] = \sum_m \sum_n x[m,n]h[i-m,j-n]$

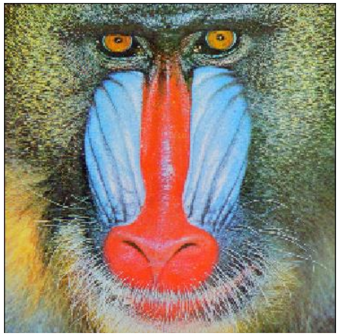
$$= \sum_m h_x[i-m] \sum_n x[m,n]h_y[j-n]$$

- Why is this advantageous? Computation! For an NxN filter
  - 2D convolution is N^2 multiply-accumulates
  - Separable convolution is 2 x 1D convolution for $2N$ multiply-accumulates

Image Processing

- Particular (exciting!) application of 2D signal processing is image processing
- Many different file types: BMP, JPG, PNG, TIF (among others)
- Typical images are
  - binary (0/1)
  - grayscale (single intensity value)
  - color (e.g. 3-channel RGB image, 4-channel RGBa)



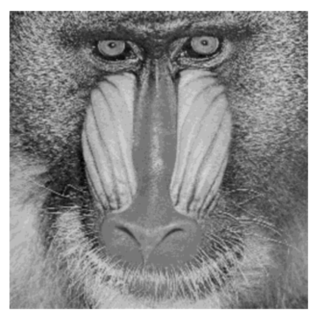


‘Test’ images

- A number of classical images used in image processing papers
- Allow for quantitative and subjective comparison of different algorithms

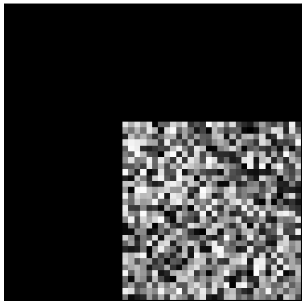


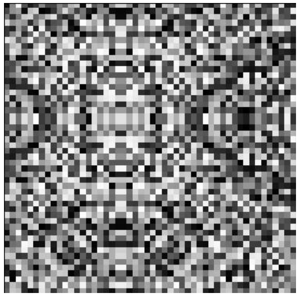





Boundary Condition Handling

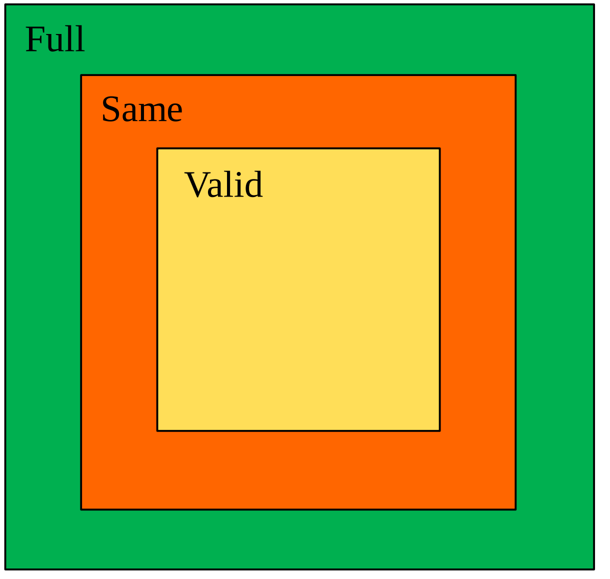
- Just as with 1D signals, we have to consider how to handle boundary conditions
  - How does signal behave outside sampled boundaries?
- Zero padding
  - Might not be a good option when working with images
- Constant extension
- Mirror extension
- Wrap-around extension
- Filter normalization
- How to pick? Mainly what makes 'sense' for your application





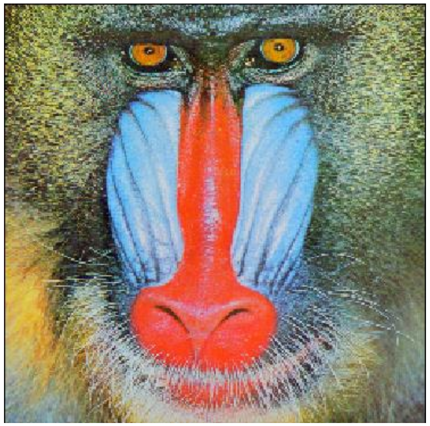
Convolution Output Domain

- Different varieties of output sets for 2D convolution
- Valid - region where h does not go outside image boundary. Output size $N - K + 1$.
- Same - same size as input image, requires handling of elements outside of image. Output size N.
- Full - expanded output image by size of h, usually assumes zeros outside image, useful for ‘overlap-add’ type image block processing. Output size $N + K - 1$

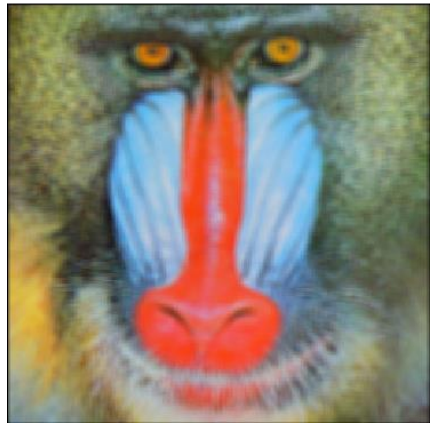


Examples of Filters

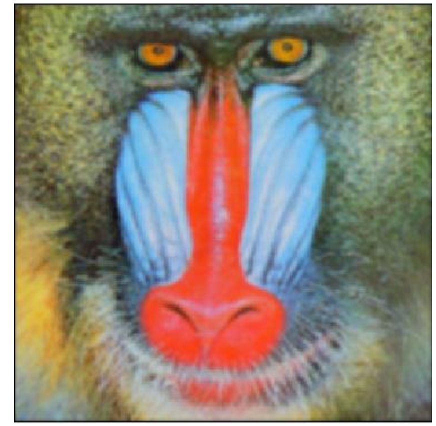
Original



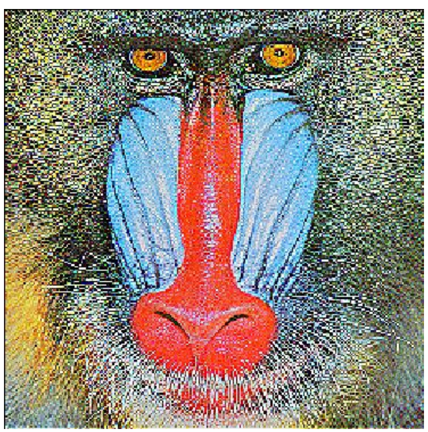
Average



Gaussian

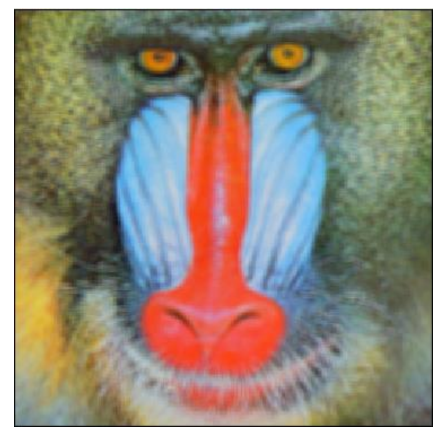


Sharpen



Median

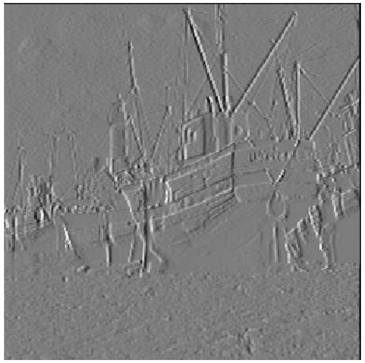
Trimmed Mean



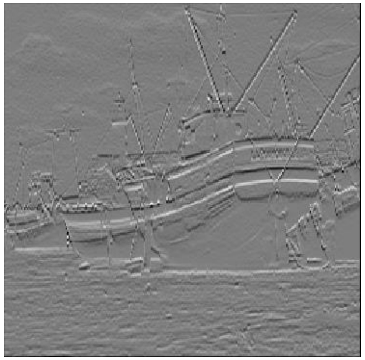
Examples of Filters

- Numerical derivative filters

Original



∂x



∂y

Edge Map

Image Data Types

- Image processing provides some unique numerical processing challenges
- Bit depth (or dynamic range) of input (and likely output) spaces
  - Binary: 0/1
  - 8-Bit: 0-255
  - 16-Bit: 0-65535
  - Most images use 8-bit representation
- Integer values over that interval
- Very easy for algorithms to
  - Exceed dynamic range of data type
  - Use too narrow an interval of dynamic range and suffer degradation due to quantization noise

Processing Pixel Values

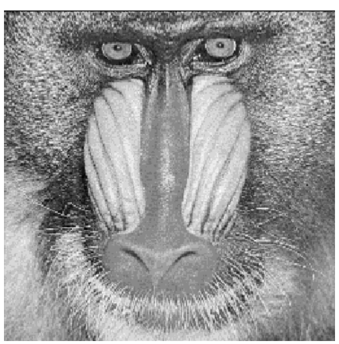
- Option 1: Keep in native data type
  - Similar difficultly to implementing fixed point algorithms
  - Unsigned datatypes can yield unexpected mathematical evaluations
    - a = 50, b = 30
    - 2a - 4b = 236 ?!?   4a + 3b = 34 ?!?
  - Can exceed maximum/minimum representable value if not careful
  - For convolution operations, can keep in native datatype without worry if all filter coefficients $h[n,m] \geq 0$ and $\sum_{n,m} h[n,m] \leq 1$
  - Need careful analysis of algorithm to ensure proper operation
    - Intermediate values can suffer from this same problem!

Processing Pixel Values

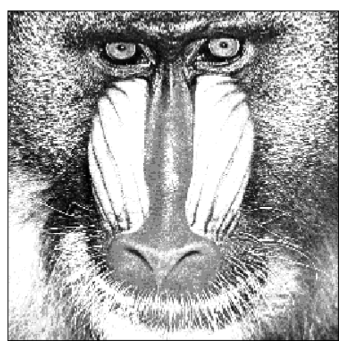
- Option 2: Temporarily convert to working data type and convert back for output
  - Work with a more ‘natural’ domain (signed integers, floating point) so reduced impact on algorithm itself
    - For floating point, can map to [0, 1) and abstract algorithm with respect to input data type
  - Introduces cost of performing type conversions
  - Working domain datatype typically larger datatype, so more memory required
    - Throughput of operations on larger datatype typically lower as well
  - Still may have problems with output elements exceeding representable range

Conversion of Output Pixels

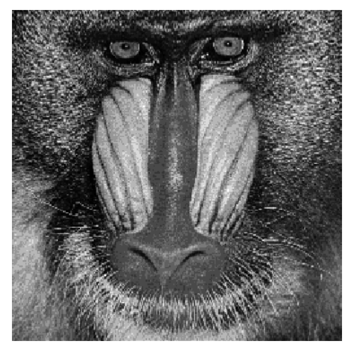
- Ideally our algorithm ‘behaves’ and keeps the output range within the dynamic range of the image pixel $[0, 2^B - 1]$
- If not, we have to map into that range in order to have a valid output image
- Some options: Clip/saturate, rescale/map, wrap-around (examples below with transform $\mathrm{f}^2/100$)



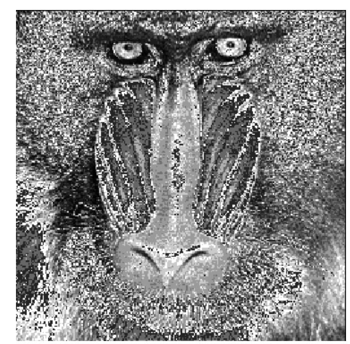
Original



Clamp / Saturate



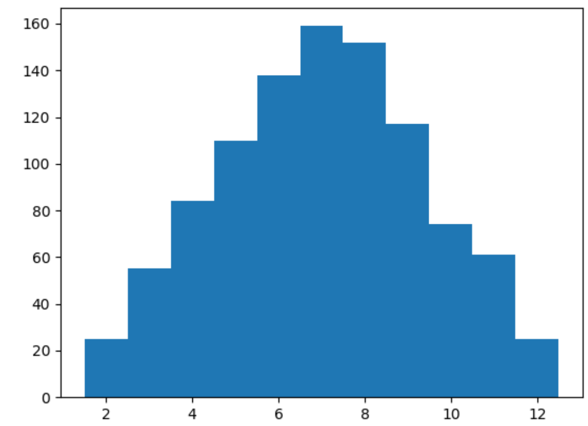
Rescale



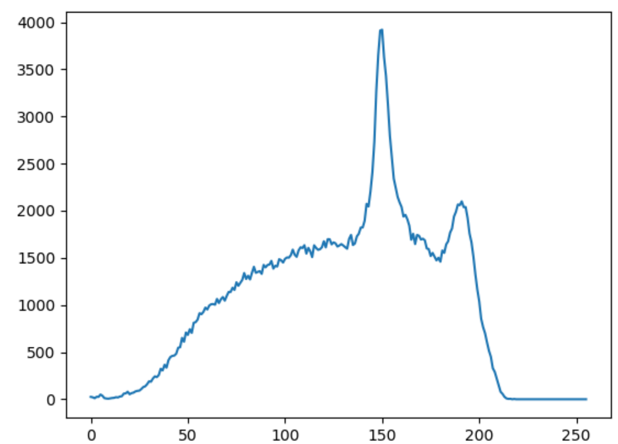
Wrap

Histograms

- Histogram represents distribution of numerical data
- Each bin denotes a particular value / outcome (or range of values/outcomes)
- Number assigned to a bin is the count of observed occurrences of values for that bin
- For images, perform analysis over all pixels in the image
  - Creates statistical distribution of pixel intensities
  - Spatial information is discarded



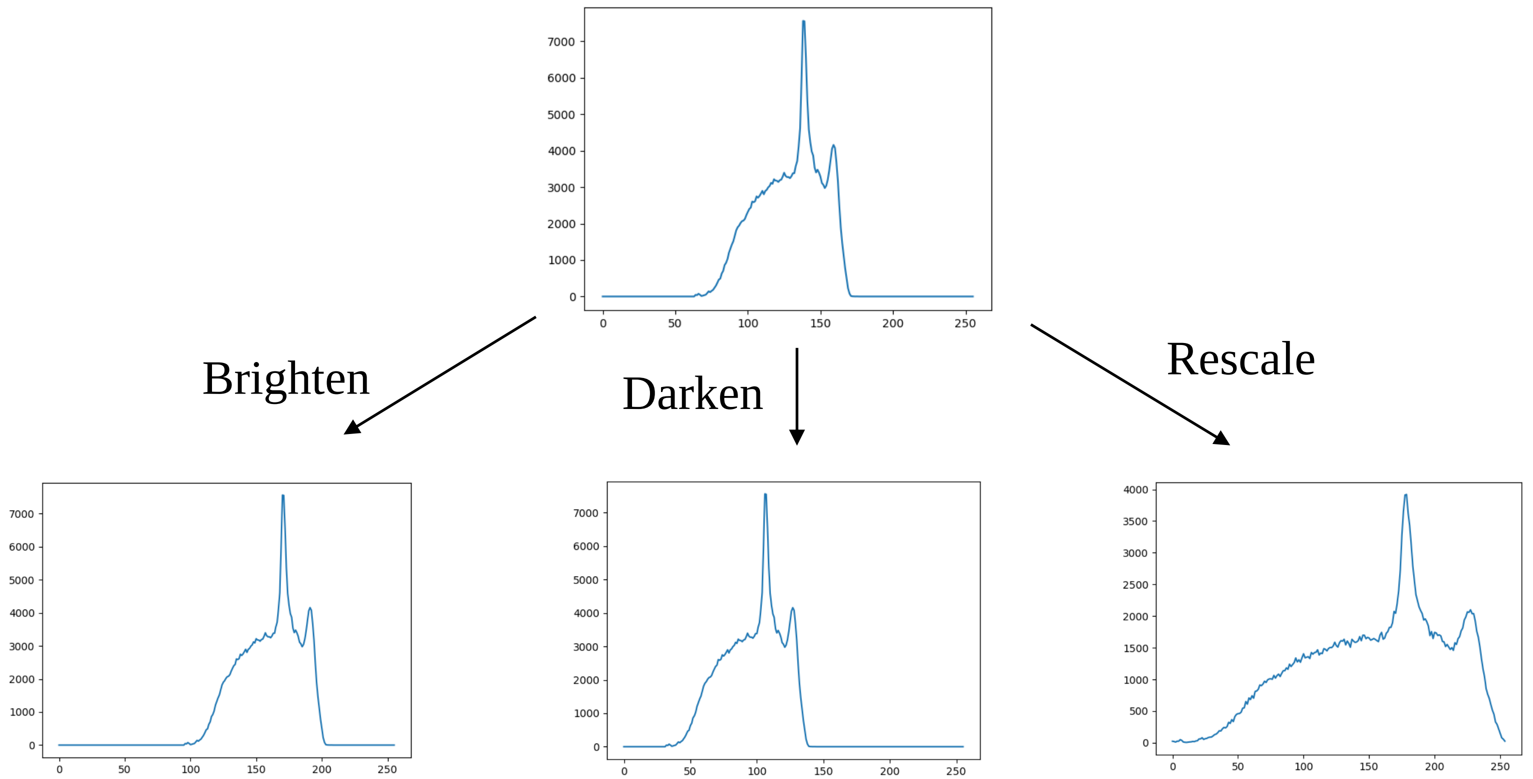
Histogram of 1,000 Dice Rolls



Histogram of 8-bit gray Mandrill

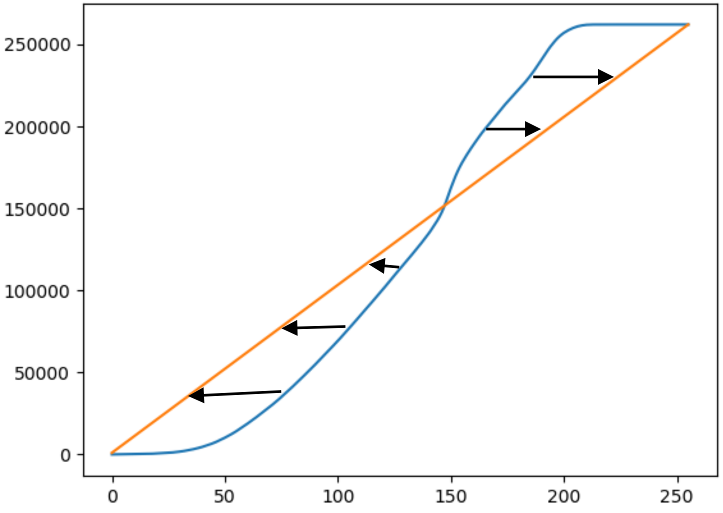
Histogram Manipulation

- Manipulate pixel values to achieve desired modified histogram distribution

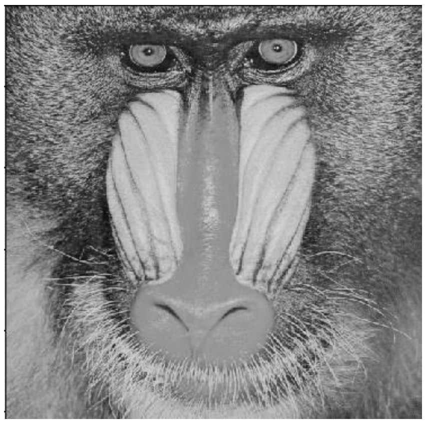


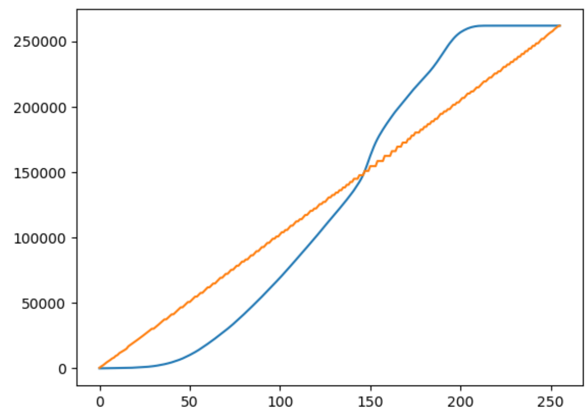
Histogram Equalization

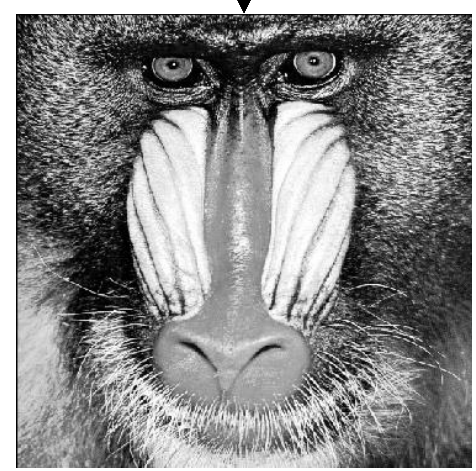
- Histogram manipulation to leverage entire dynamic range of pixel values
- Define the cumulative distribution function
  - $C[x] = \sum_{t=0}^{x} h[t]$
- Determine a warping function that maps pixel values in the input distribution to pixel values in the output distribution
  - $x_2 = W(x_1) \approx C_2^{-1}(C_1[x_1])$
- For a linear distribution in C_2
  - $W(x_1) = \frac{C_1[x_1] - C_1[x_{min}]}{N^2 - C_1[x_{min}]} (2^B - 1)$



Histogram Equalization

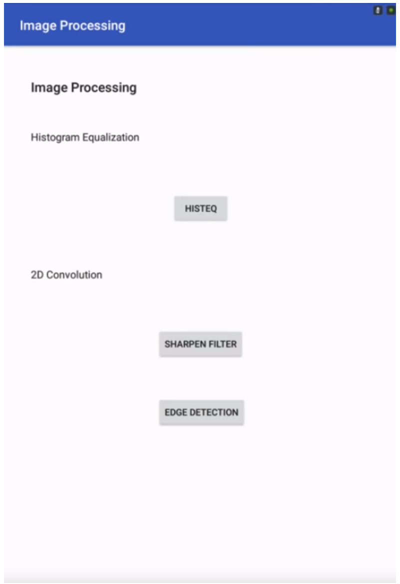


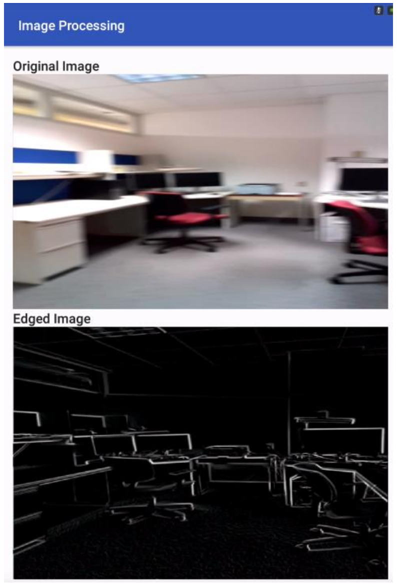




Lab 6 Overview

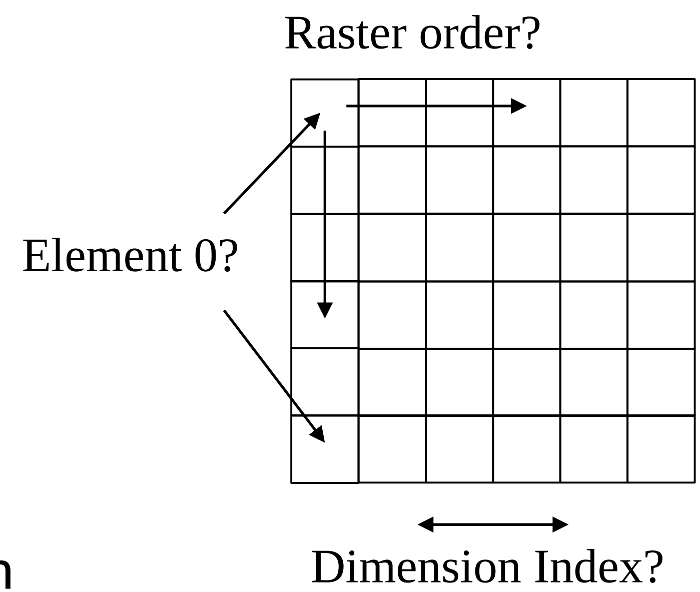
- Implementation of real-time histogram equalization and 2D convolution
- Not both at the same time, user selectable
- Different filters can be chosen, can assume a 3x3 kernel



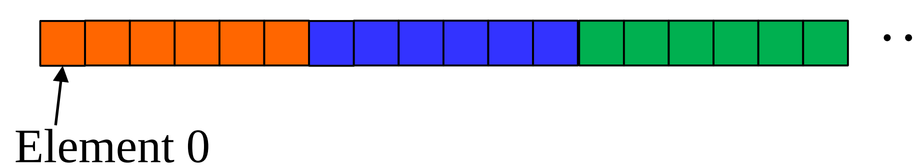


Working with Image Data

- Most high level languages define array/image objects
  - Simplifies algorithm implementation with explicit multi-dimensional indexing
  - Complicates algorithm implementation with possibly non-intuitive indexing conventions
  - Recommend create a small image with some landmark pixels to understand convention

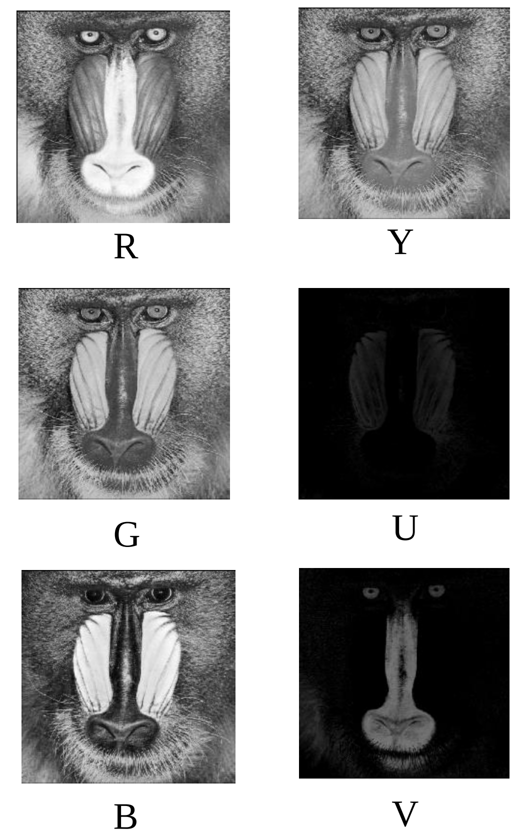


- Lower level languages use a flat buffer
  - Explicit index calculations
  - $offset = x + y * width$
  - Careful for buffer overrun!



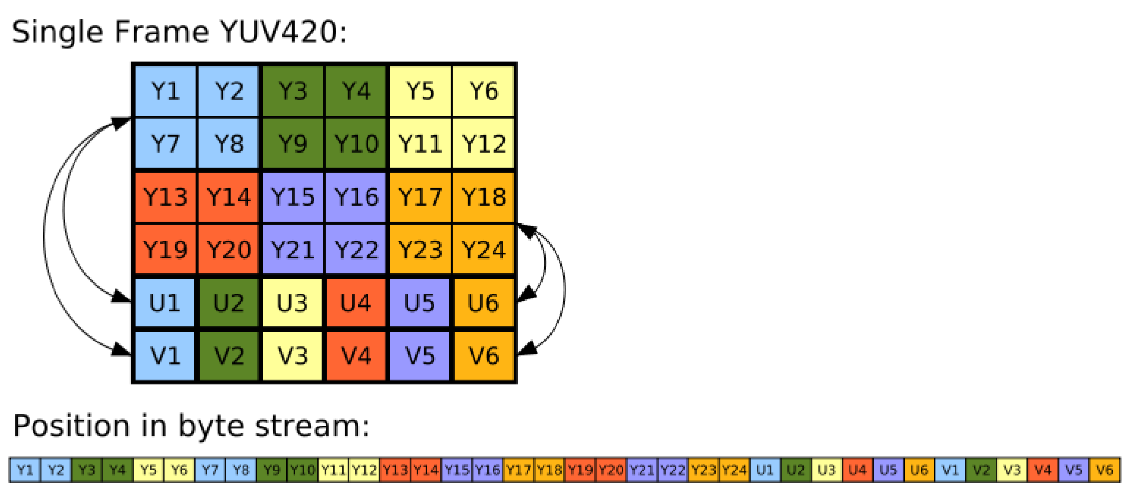
RGB vs. YUV

- Color images are broken down into different bands of information
- "Traditional" representation is RGB
  - One channel each for red, green, and blue respectively
- Another common representation is YUV
  - Y - luminance
  - U,V - chrominance
- YUV provides a perception-based encoding
  - RGB mostly distributes information among all channels
  - YUV concentrates most information in Y channel



Android Handling of Color

- YUV420 encoding
  - U, V channels sampled at half the rate in x/y dimensions
  - U, V channels follow Y channel
- Since Y carries most of the information, we will just manipulate the Y channel alone (grayscale intensity map)
- Leaving U, V alone automatically 'recolors' the pixels



Many, Many Other Image Manipulations

- Segmentation
- Morphological operations
- Compositing
- Warping
- Rotation
- Denoising
- Compression
- Classification / Identification
- Feature Extraction

This week

- Lab 5: Pitch Synthesizer Quiz/Demo
- Lab 6: Image Processor (Histogram and Filtering)
- Assigned Project Lab Proposals Due March 1