Visual Computing I:

Interactive Computer Graphics and Vision

Digital Image Processing (Manipulation and Analysis)

Stefanie Zollmann and Tobias Langlotz

Last time..

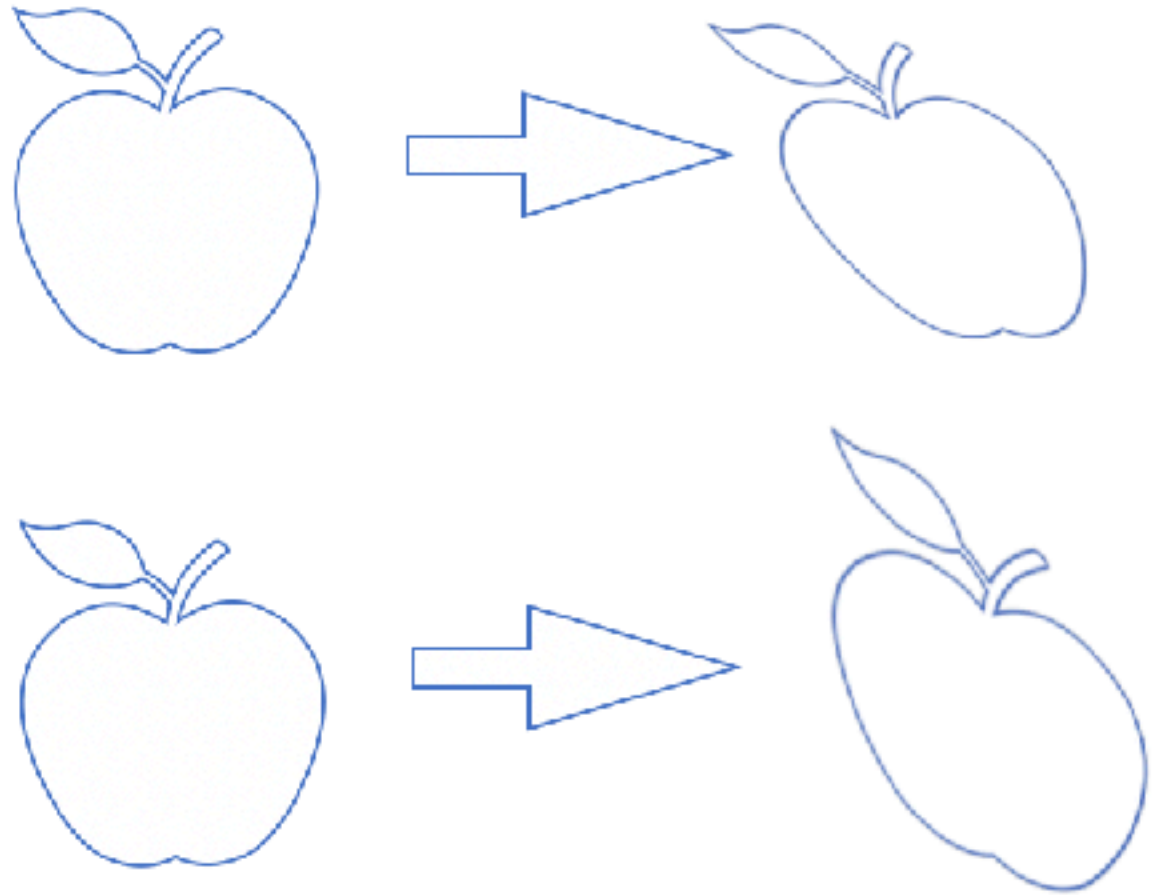
Homogenous Transforms - Shearing

- Distorts the shape by shifting one axis relative to the other
- Example: Turning a rectangle into a parallelogram
- Shearing by a factor of sh_x in x direction:

$$\begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Shearing by a factor of sh_y in your direction:

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



2D Animations

- Animation = applying transformations over time
- Common techniques:
  - Translation over time -> moving an object
  - Scaling over time -> growing/shrinking
  - Rotation over time -> spinning
- Achieved by updating transformation parameters frame by frame

2D Image Manipulation

2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

Image Luminance

Luminance

- Measures perceived “gray-level” of pixel
  - Not so good:
    - (R+G+B) / 3
    - Our example: L = 120

Luminance

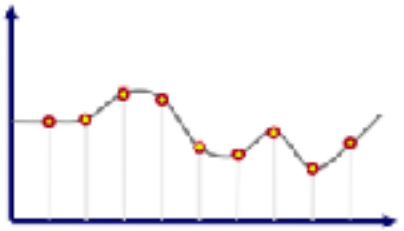
- Measures perceived “gray-level” of pixel
  - Not so good:
    - (R+G+B) / 3
    - Our example: L = 120
  - Better:
    - L = 0.30*R + 0.59*G + 0.11*B
    - Luminosity method (human perception)
    - Our example: L = 106

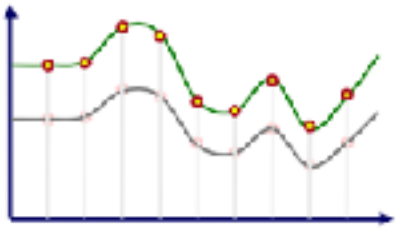
Luminance

- Measures perceived “gray-level” of pixel
  - Not so good:
    - (R+G+B) / 3
    - Our example: L = 120
  - Better:
    - L = 0.30*R + 0.59*G + 0.11*B
    - Luminosity method (human perception)
    - Our example: L = 106
- What must be done to the RGB values to make this image brighter?

Luminance

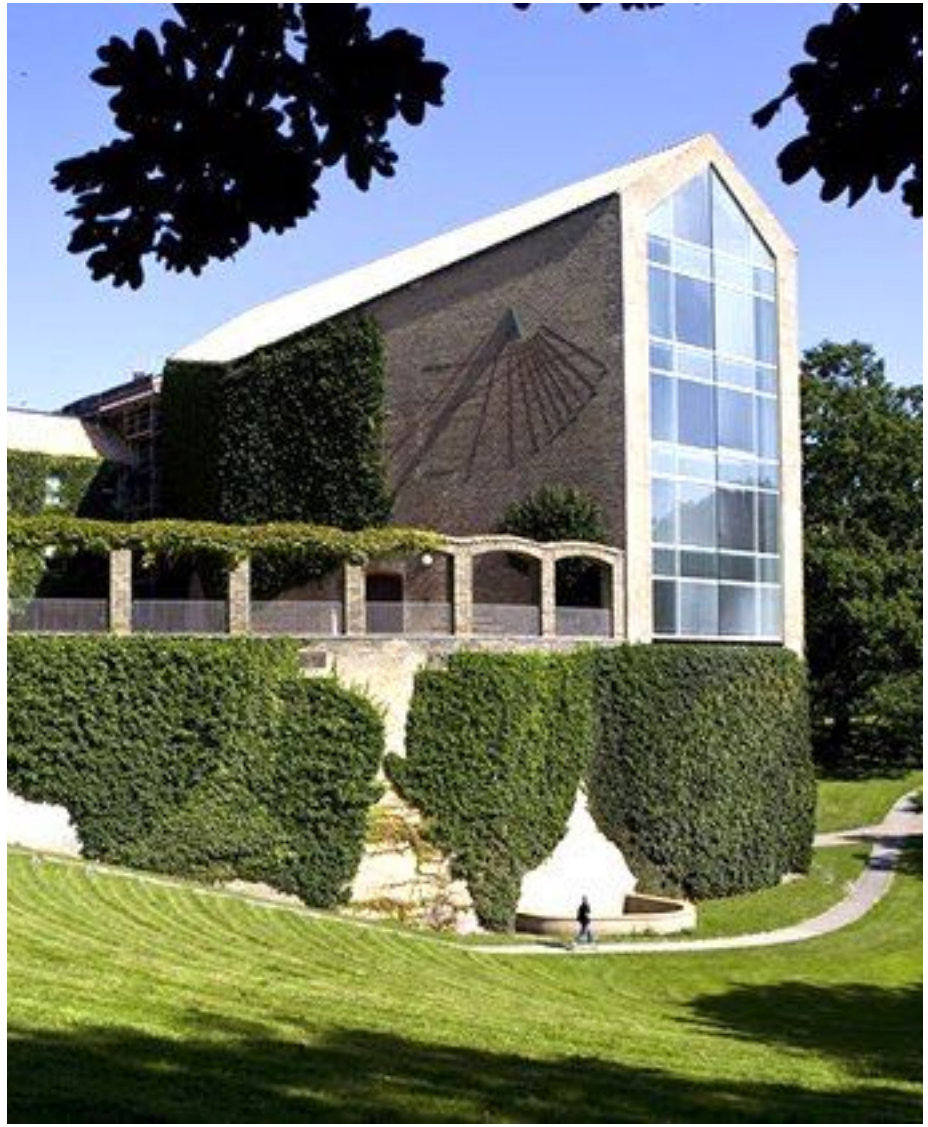
- Change luminance:
- Method 1: Convert to HSL, scale L, convert back
    - (more on this shortly…)
- Method 2: Scale R, G, and B directly
    - Multiply each of red, green, and blue by a factor
    - Must clamp to [0..1] … always
        - ([0..1] in floating point but often [0,255] for fixed point)

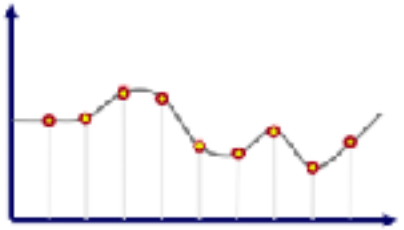


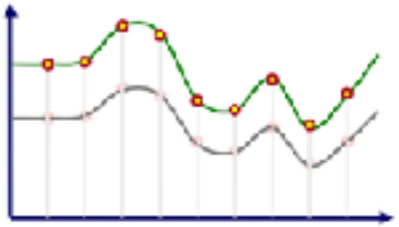


Luminance

- Change luminance:
- Method 1: Convert to HSL, scale L, convert back
    - (more on this shortly…)
- Method 2: Scale R, G, and B directly
    - Multiply each of red, green, and blue by a factor
    - Must clamp to [0..1] … always
        - ([0..1] in floating point but often [0,255] for fixed point)

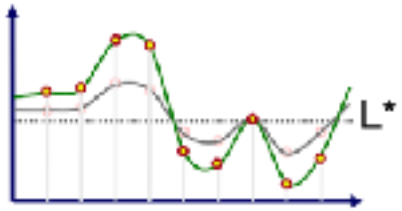






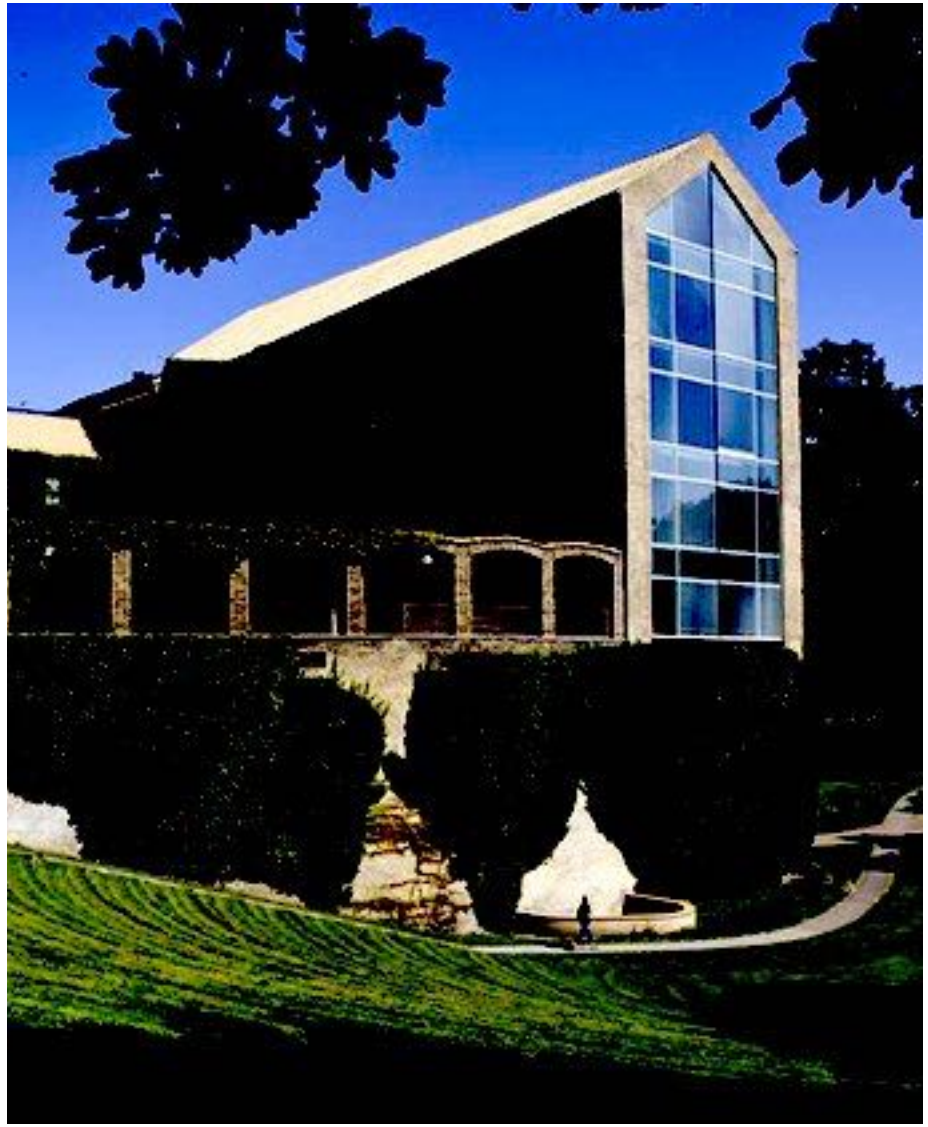
Contrast

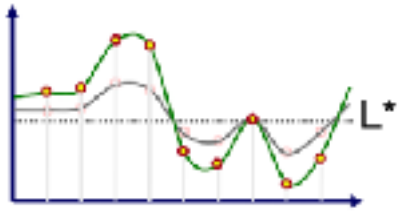
- Contrast is the difference in luminance or colour that makes an object (or its representation in an image or display)
- Simpler: Contrast is the difference in luminance or colour for each pixel from the overall mean
    - Compute mean luminance L* over whole image
    - Scale deviation from L* for each pixel



Contrast

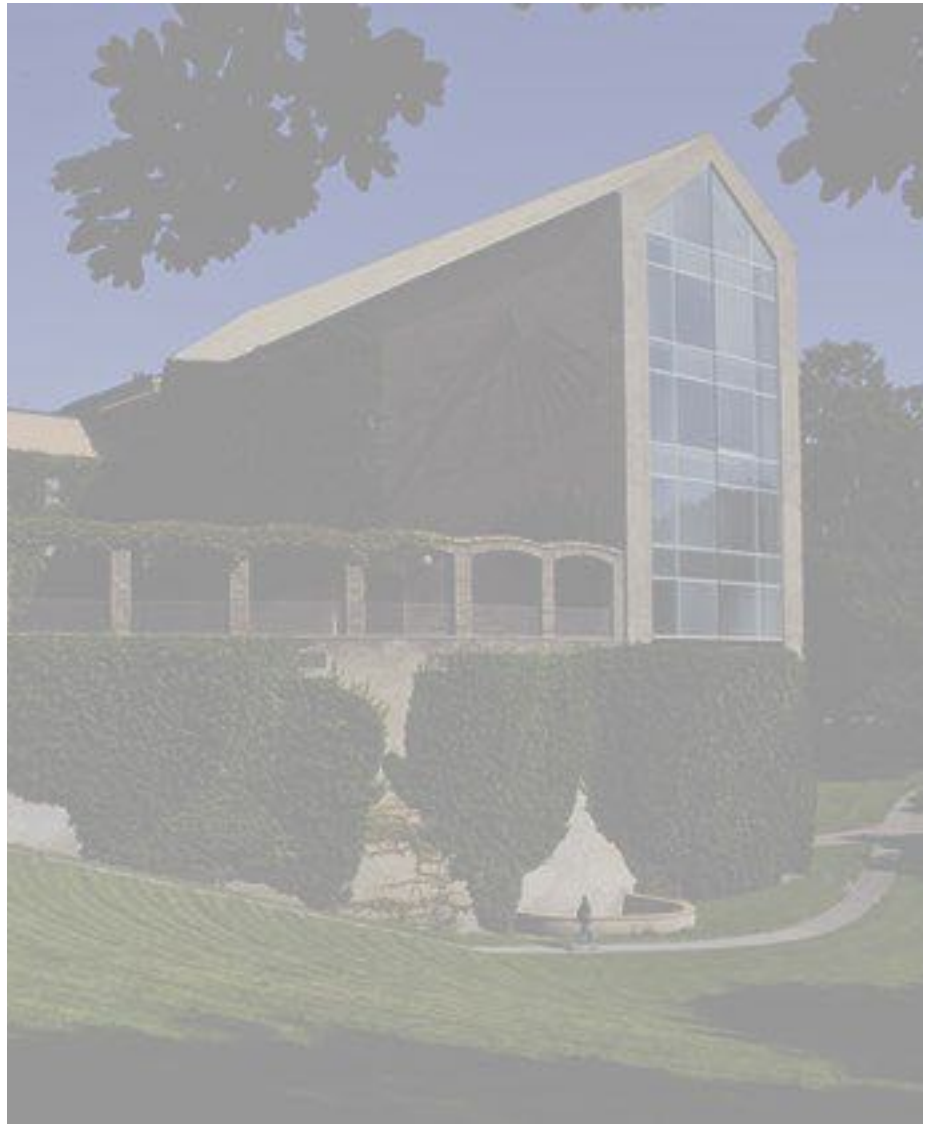
- Contrast is the difference in luminance or colour that makes an object (or its representation in an image or display)
- Simpler: Contrast is the difference in luminance or colour for each pixel from the overall mean
    - Compute mean luminance L* over whole image
    - Scale deviation from L* for each pixel

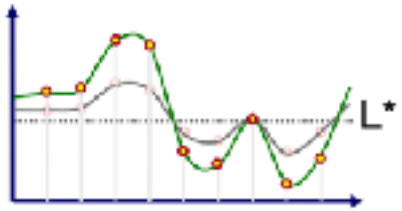




Contrast

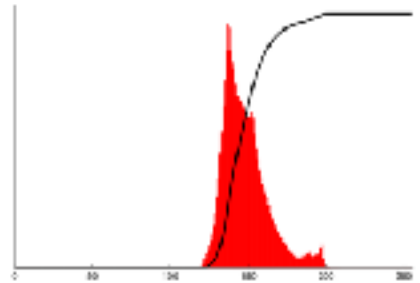
- Contrast is the difference in luminance or colour that makes an object (or its representation in an image or display)
- Simpler: Contrast is the difference in luminance or colour for each pixel from the overall mean
    - Compute mean luminance L* over whole image
    - Scale deviation from L* for each pixel

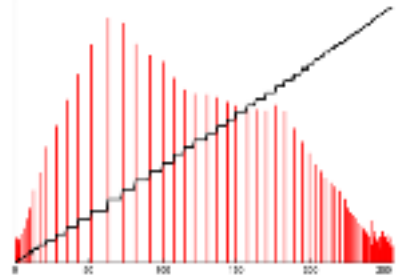




Histogram Equalization / Normalisation

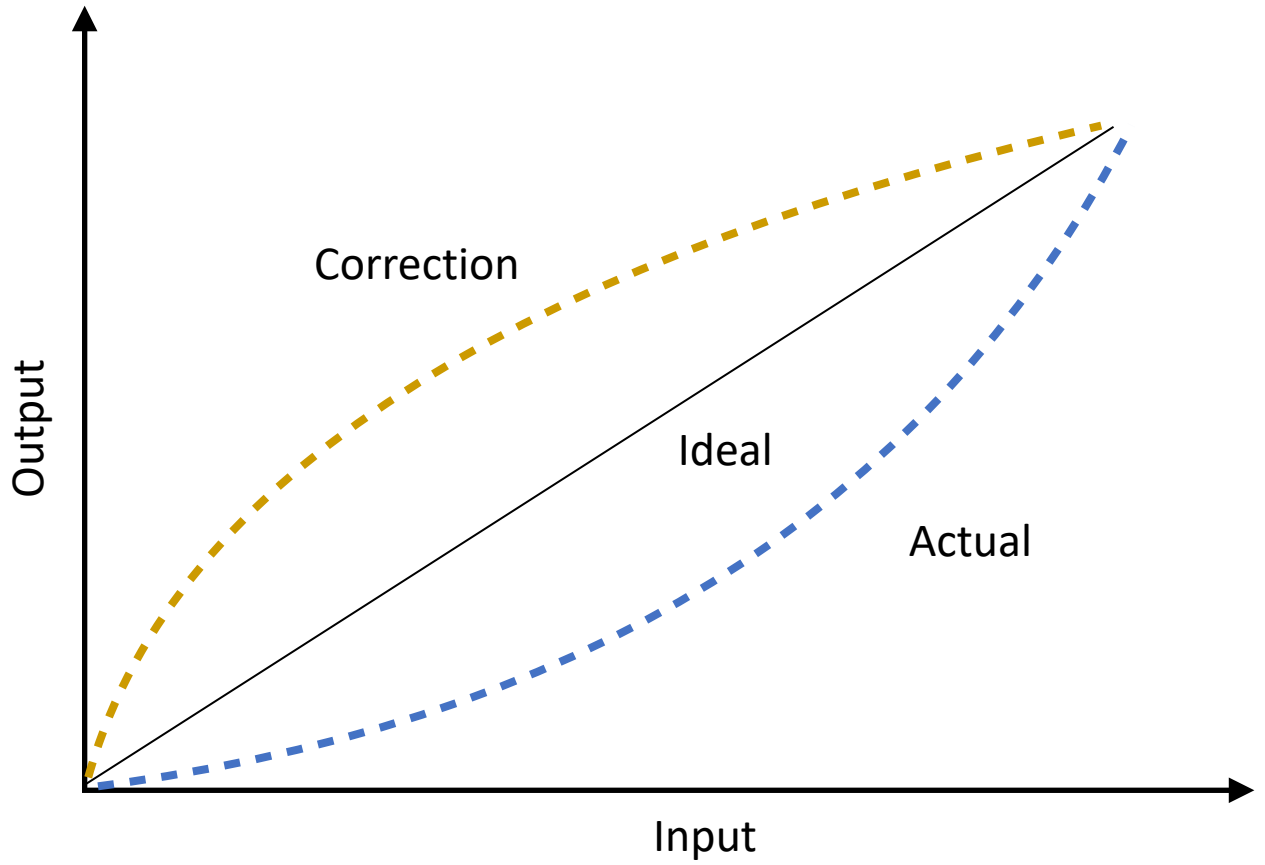
- Different form of Contrast
- Change distribution of luminance values to cover full range [0-1] / [0,255]





Gamma Correction

- Apply non-linear function to account for the difference between brightness and perceived brightness of displays
- $I_{out} = I_{in}^{\gamma}$
- γ depends on camera and monitor



2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

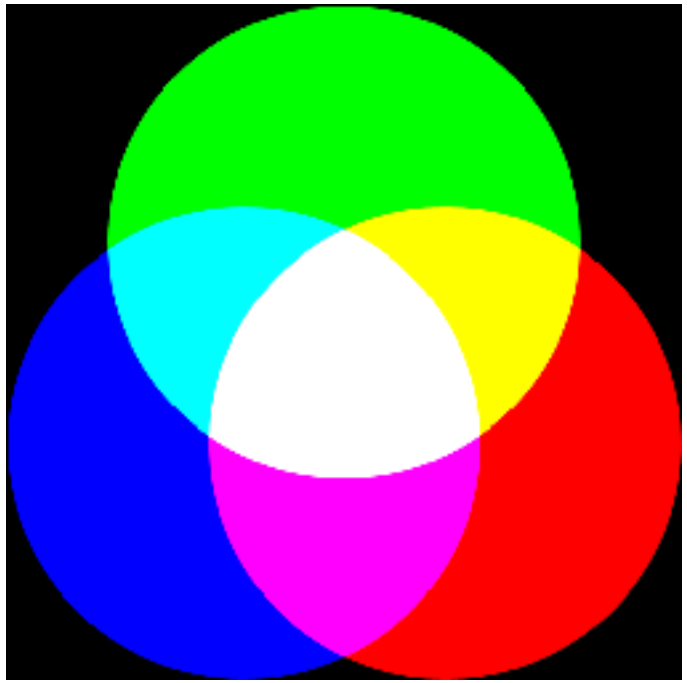
2D Image Processing

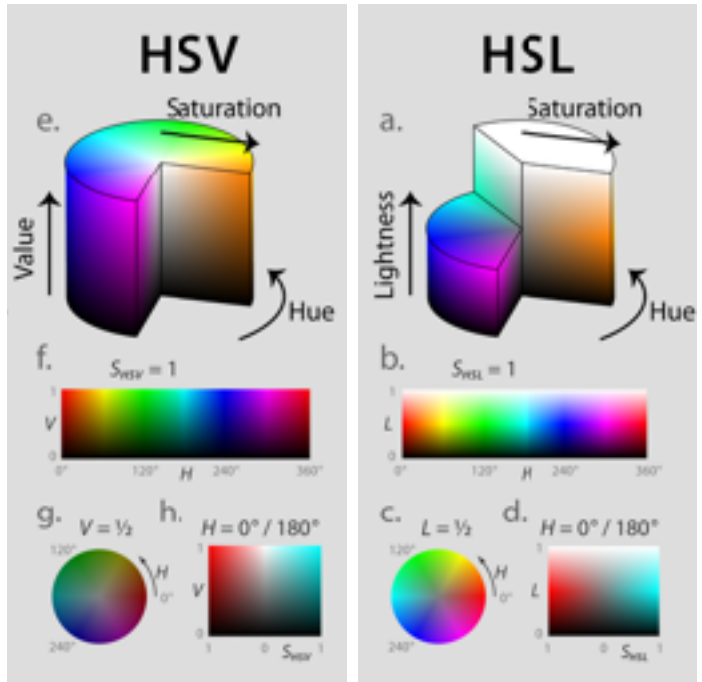
- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

Color

Colour Processing

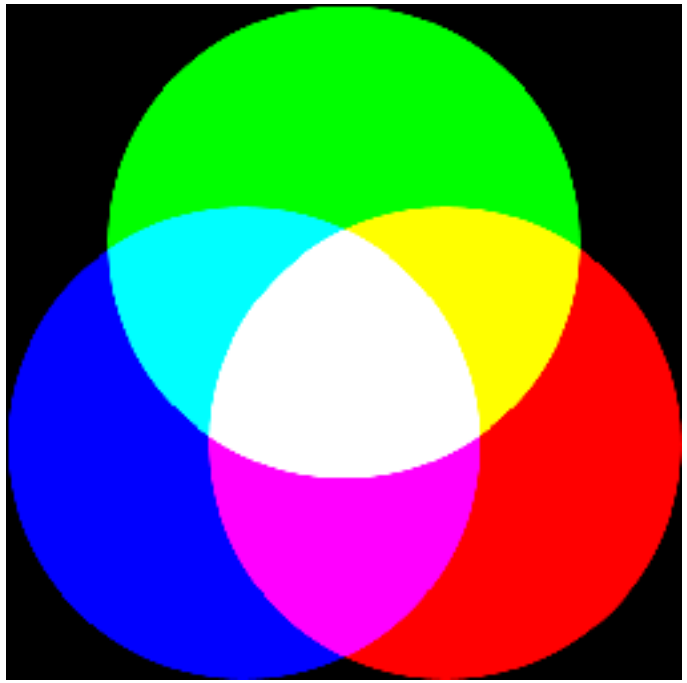
- Make use of different colour models
- Colour models (previous lec.)
  - RGB
  - CMY(K)
  - HSV
  - HSL
  - La*b*
  - Etc. (XYZ, ..)

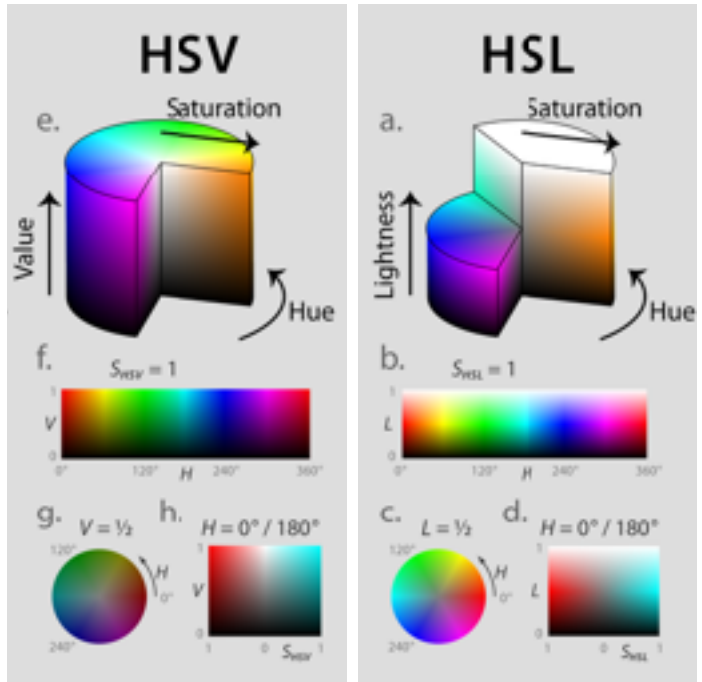




Colour Processing

- Make use of different colour models
- Colour models (previous lec.)
  - RGB
  - CMY(K)
  - HSV
  - HSL
  - La*b*
  - Etc. (XYZ, LMS..)





- Which one is good for conversion to grayscale? Saturation?

Greyscale

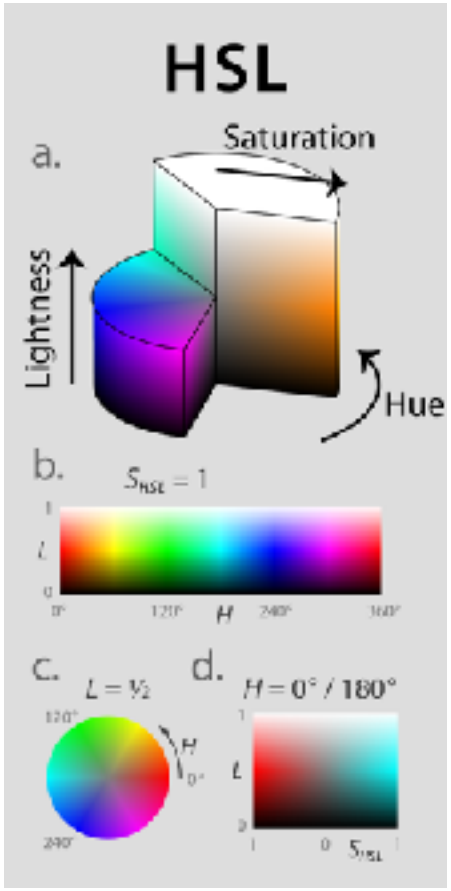
Original

Greyscale
(“black&white” photo)

Greyscale

- Convert from colour to grey-levels
  - Method 1: (not so good): R+G+B/3

Greyscale



- Convert from colour to grey-levels
  - Method 2: Convert to HSL, set S=0, convert back to RGB
  - Method 3: Set RGB of every pixel to (L,L,L)

Saturation

Original

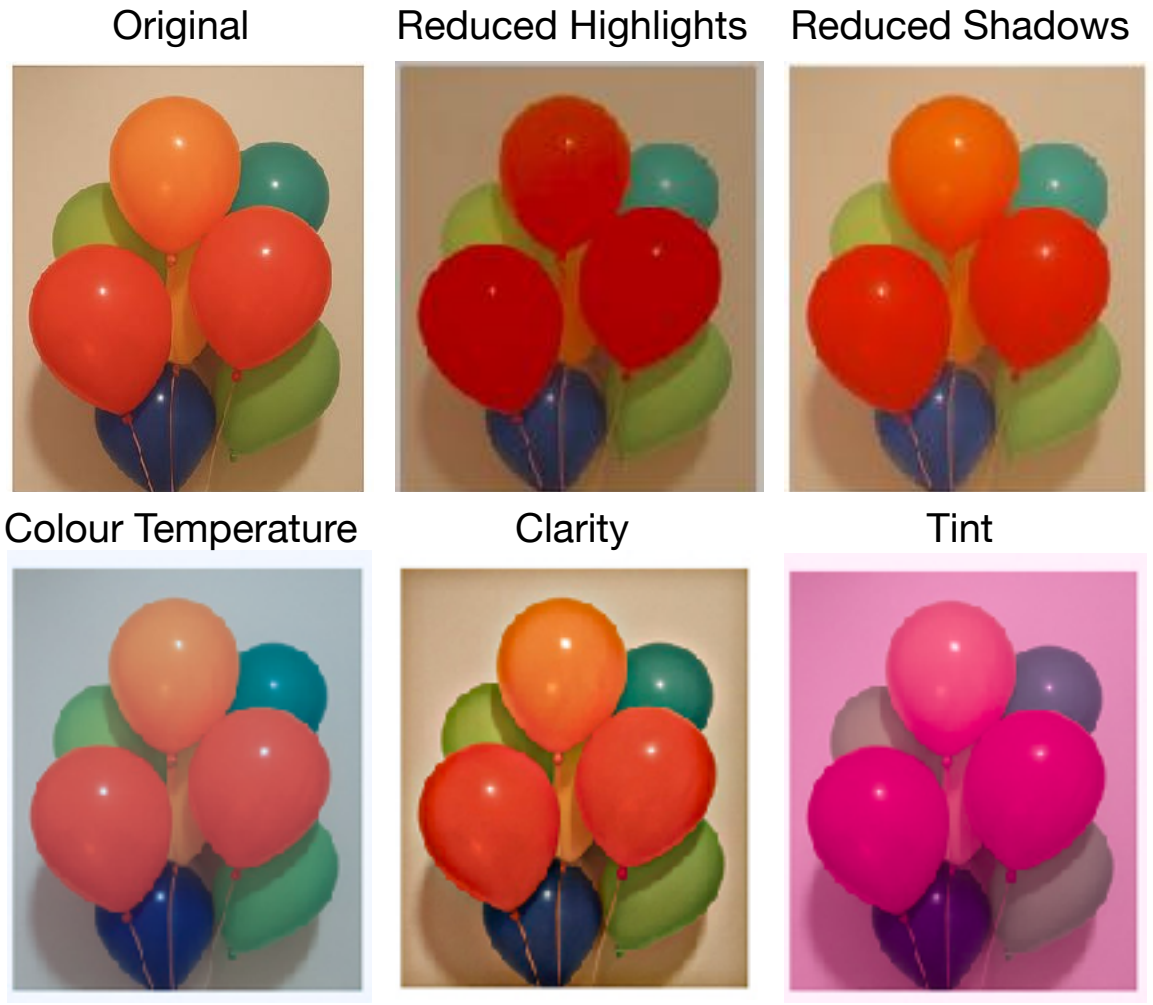
Adjusted Saturation

Saturation

- Increase/decrease color saturation of every pixel
  - Method 1: Convert to HSL, scale S, convert back
  - Method 2: R’ = L + scale * (R-L) … same for G&B

Other

- Many other colour processing methods:
  - White balancing
  - Colour Temperature
  - Tint
  - Texture
  - Clarity
  - Dehazing
  - Highlights/Shadows
  - Levels/Histogram
- General rule: Think about the best colour models to operate in



2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

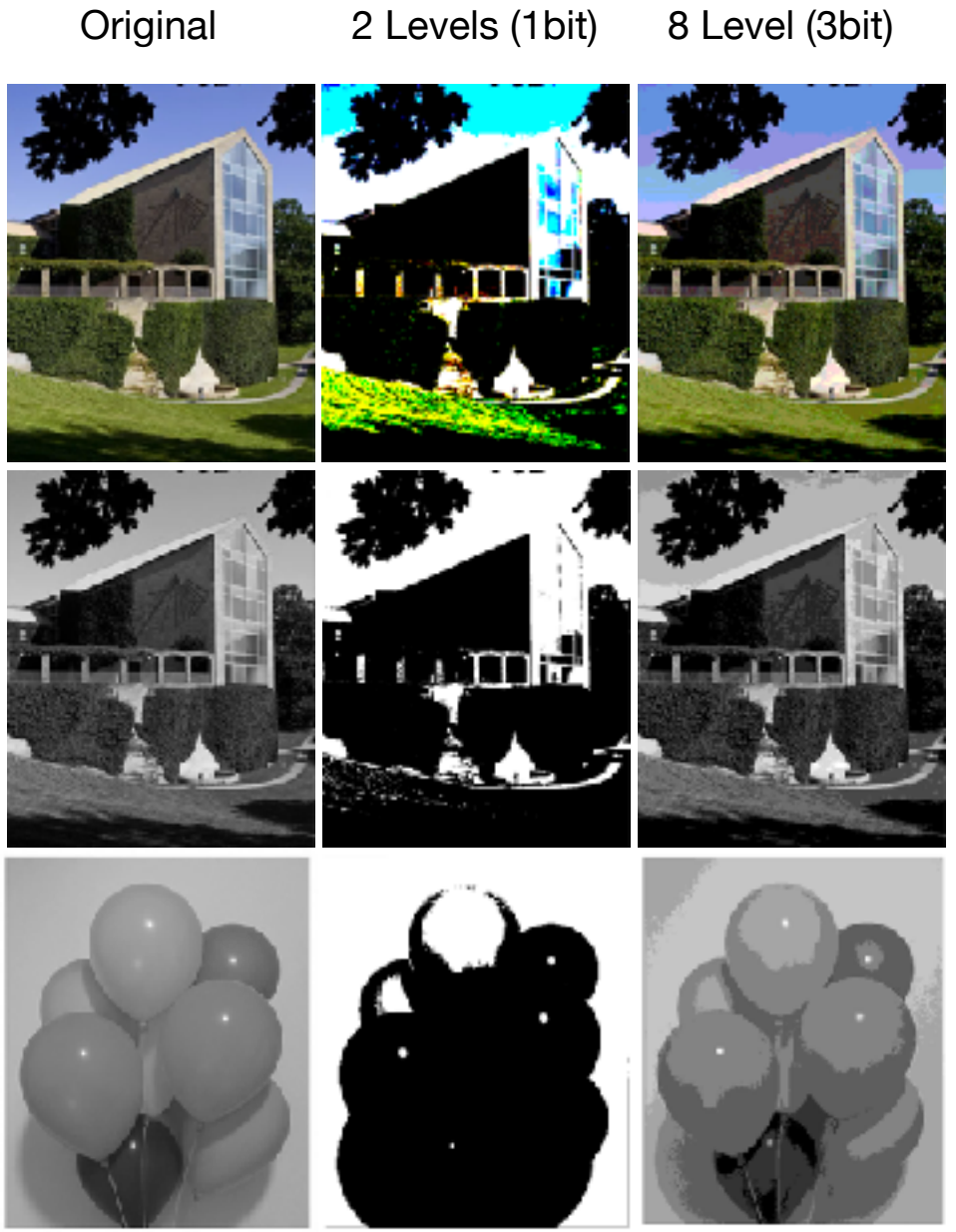
2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

Dithering

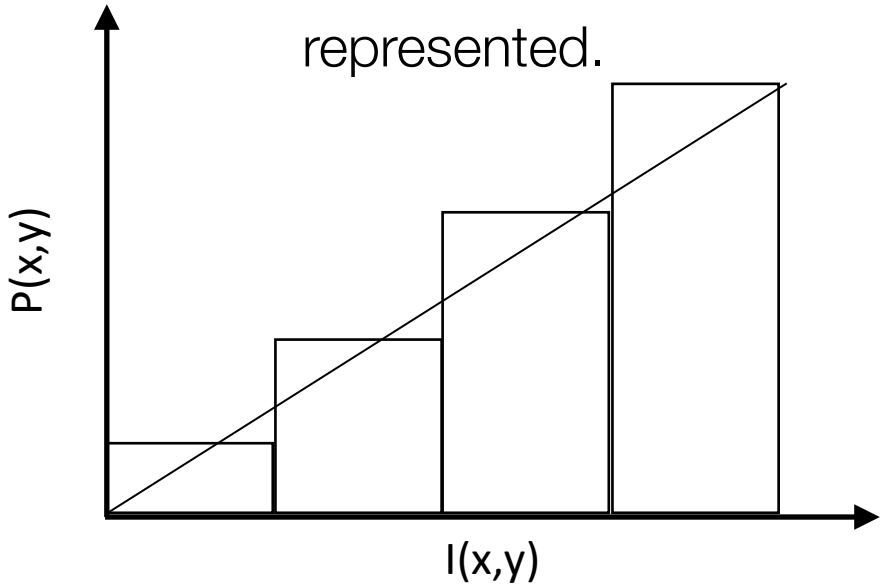
Quantization and Dithering

- Quantization:
  - Reduce intensity/colour resolution
    - Greyscale 8bit to 1bit
    - RGB 12 bit to 8 bit
    - RGB 8bit to 5/6 bits (RGB565)
  - Frame buffers have a limited number of bits per pixel
  - Physical devices have a limited dynamic range
  - Reduce image size



Quantization and Dithering

- Quantization:
  - P(x, y) = round(I(x, y))
    - Where round() chooses nearest value that can be represented.

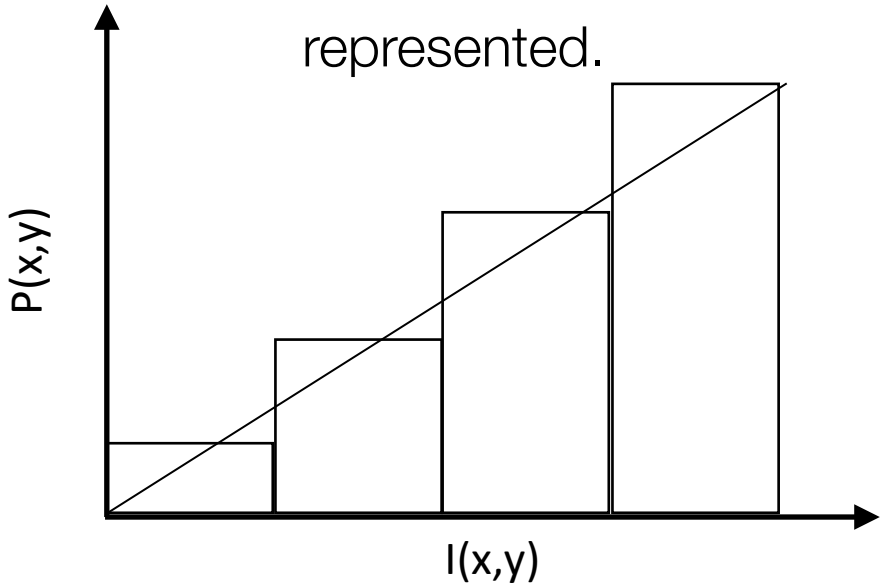


Original

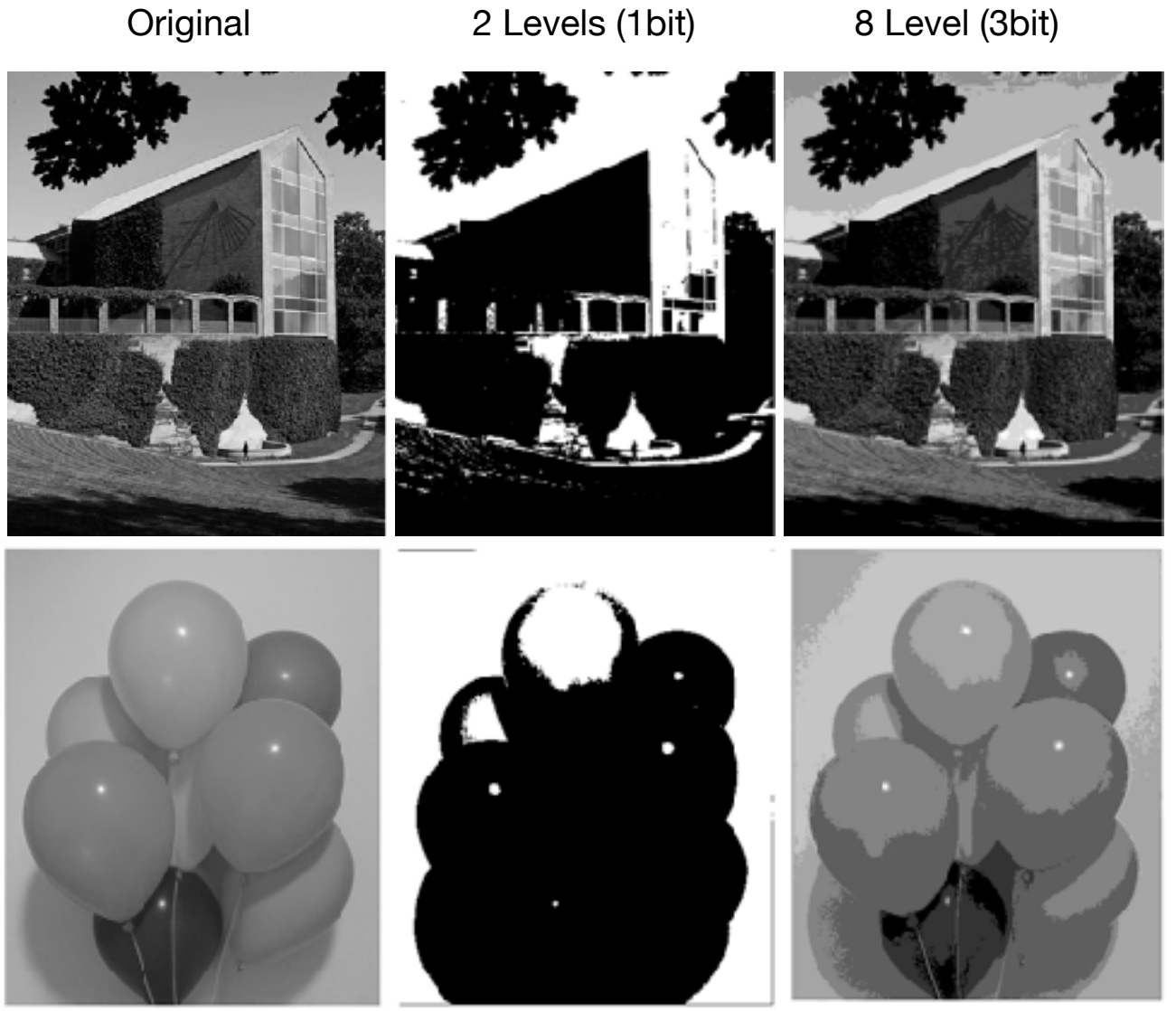
4 Levels (2 bit)

Quantization and Dithering

- Quantization:
  - P(x, y) = round(I(x, y))
    - Where round() chooses nearest value that can be represented.

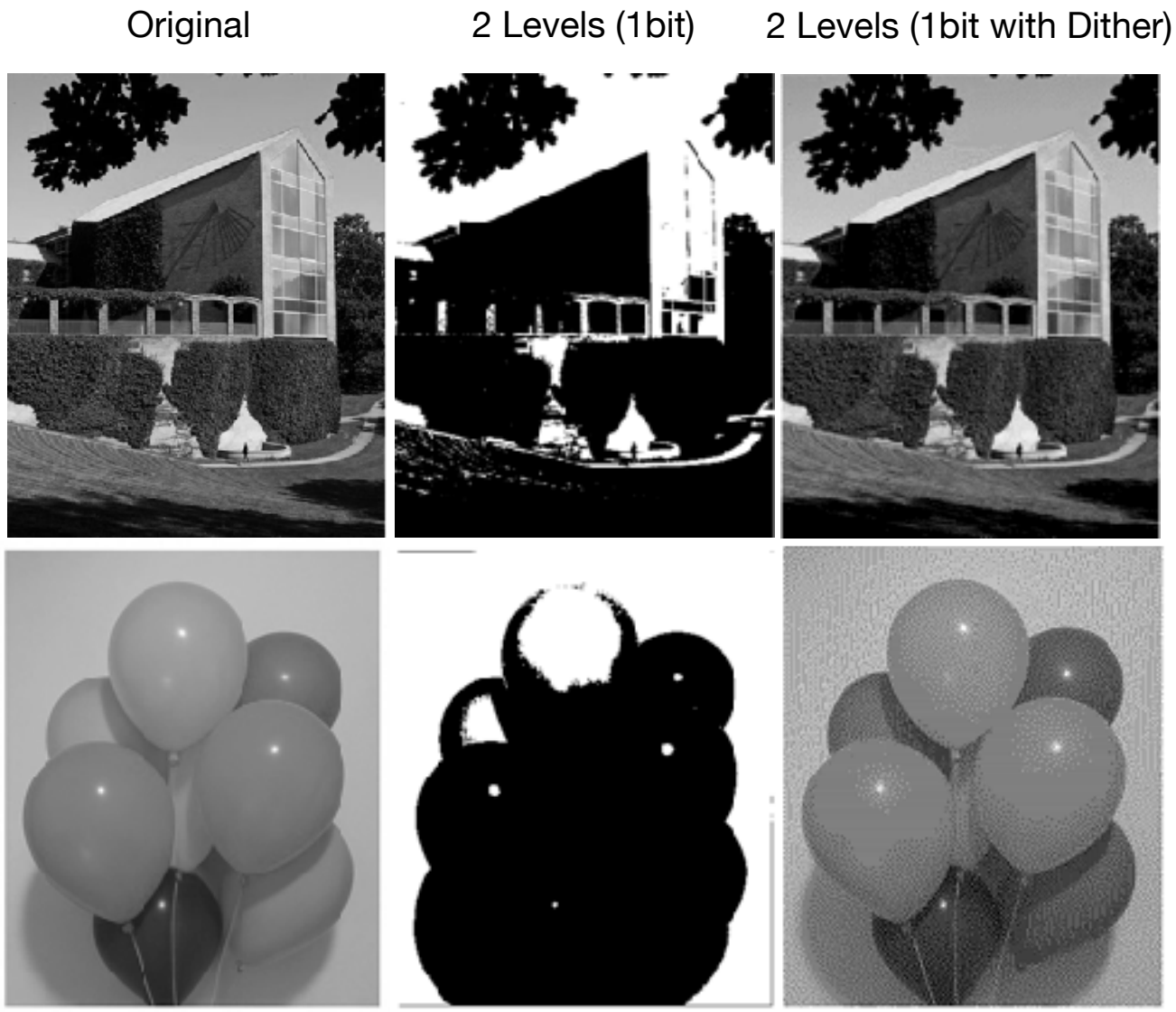


  - Contouring or Banding

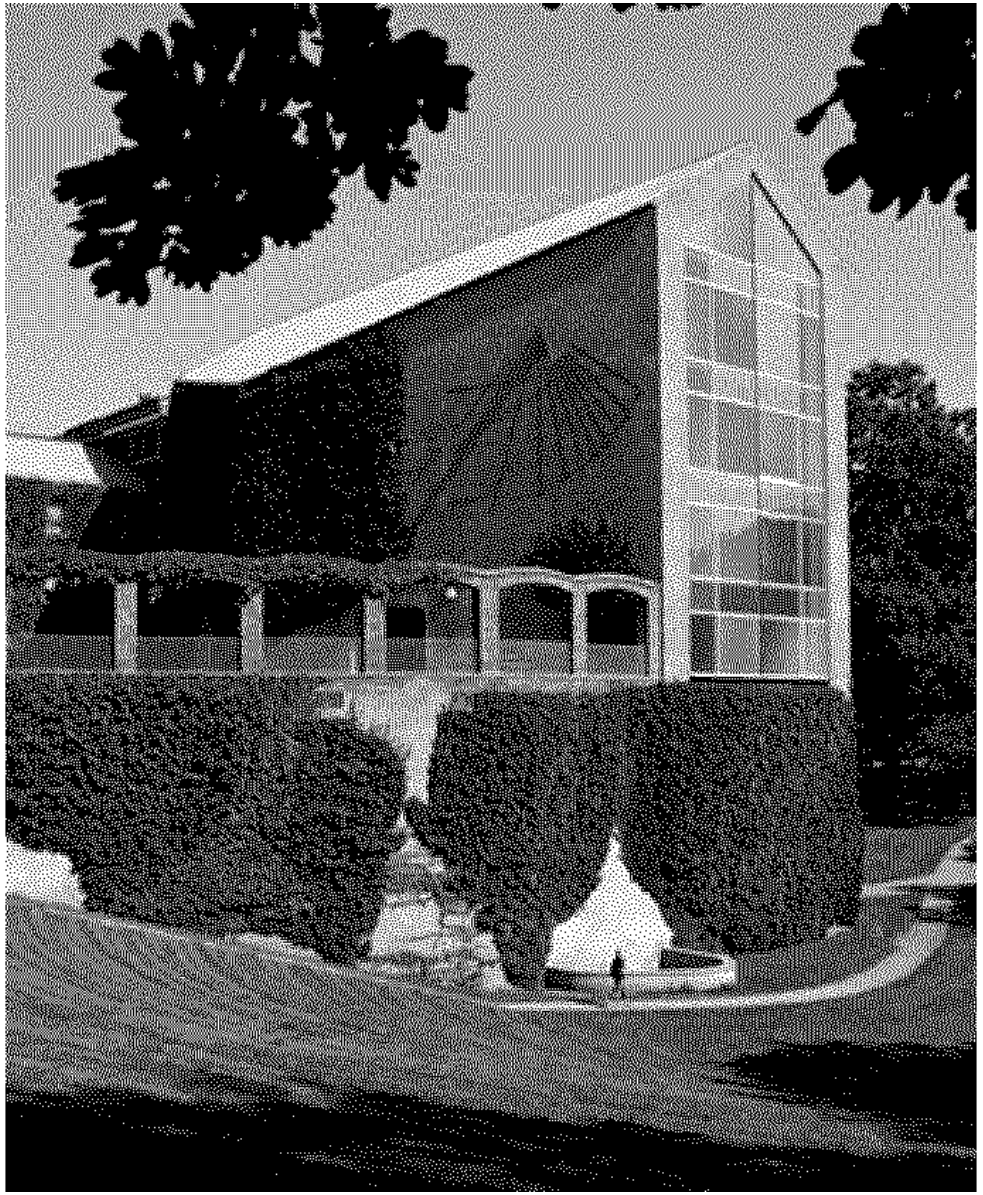


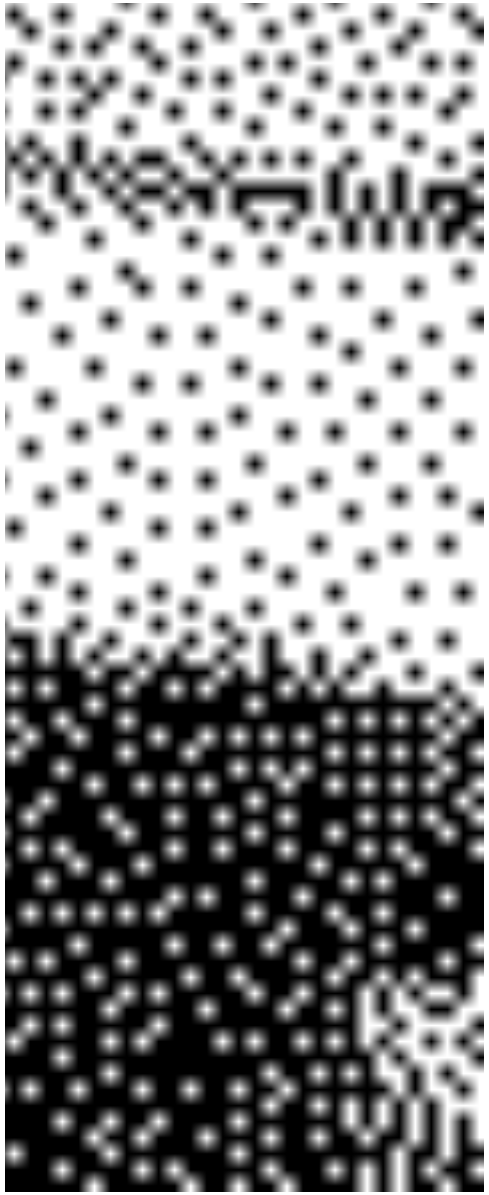
Quantization and Dithering

- Dither use noise or patterns to randomise or reduce quantisation error
- Reducing large-scale patterns (contouring or banding)

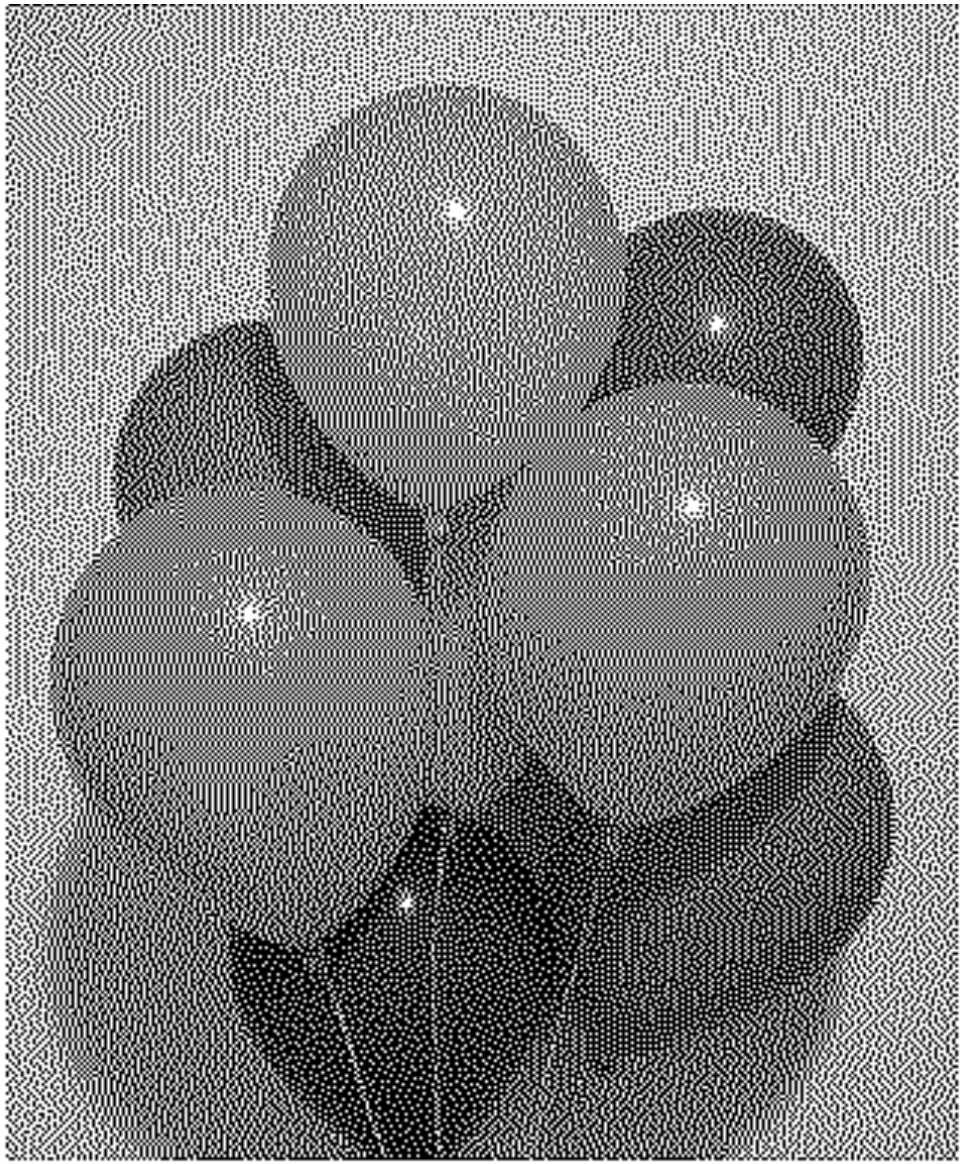


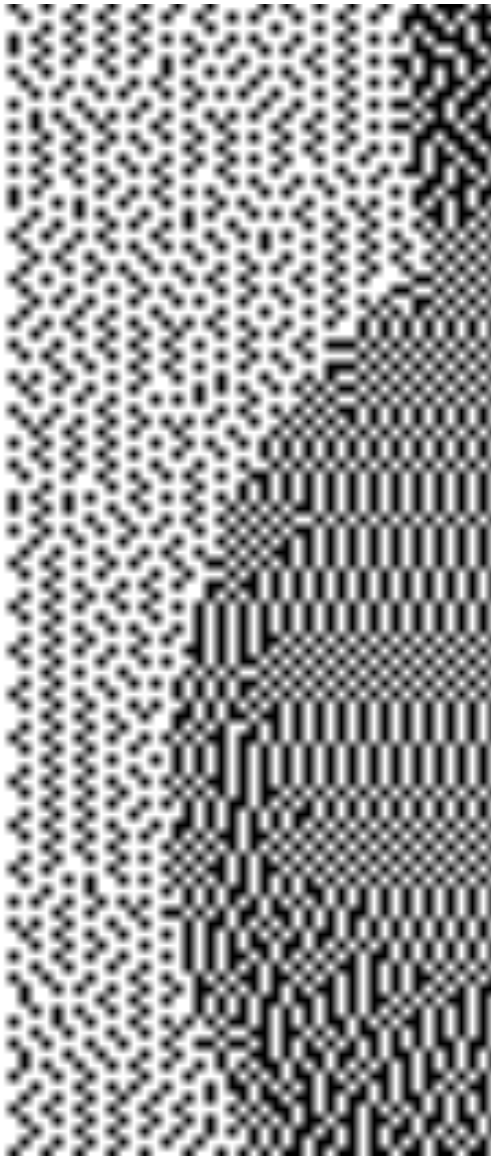
Quantization and Dithering





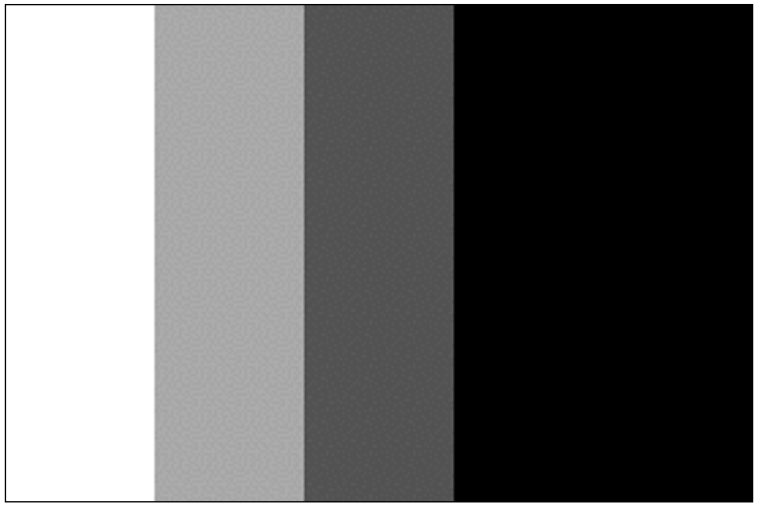
Quantization and Dithering

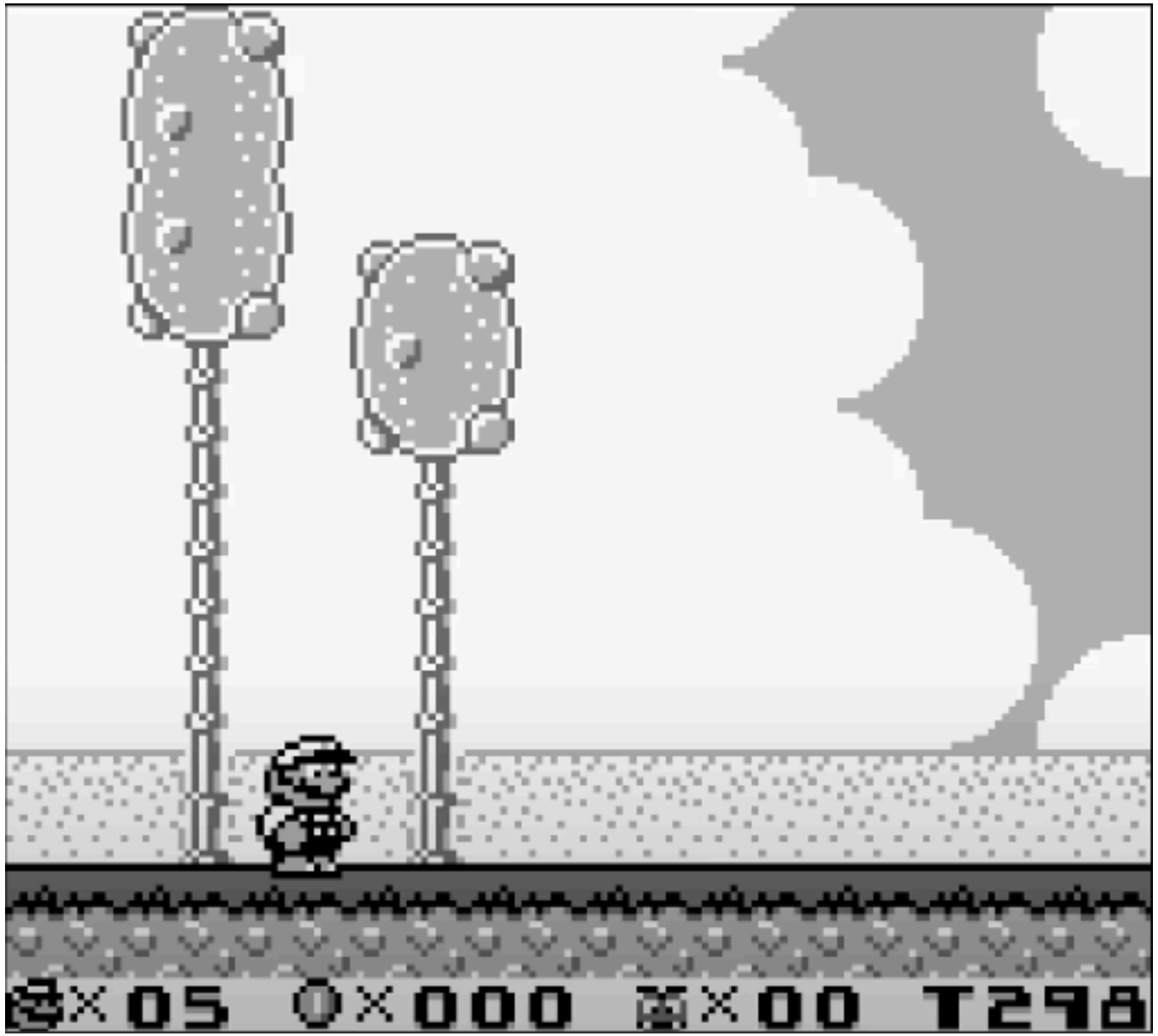




Quantization and Dithering

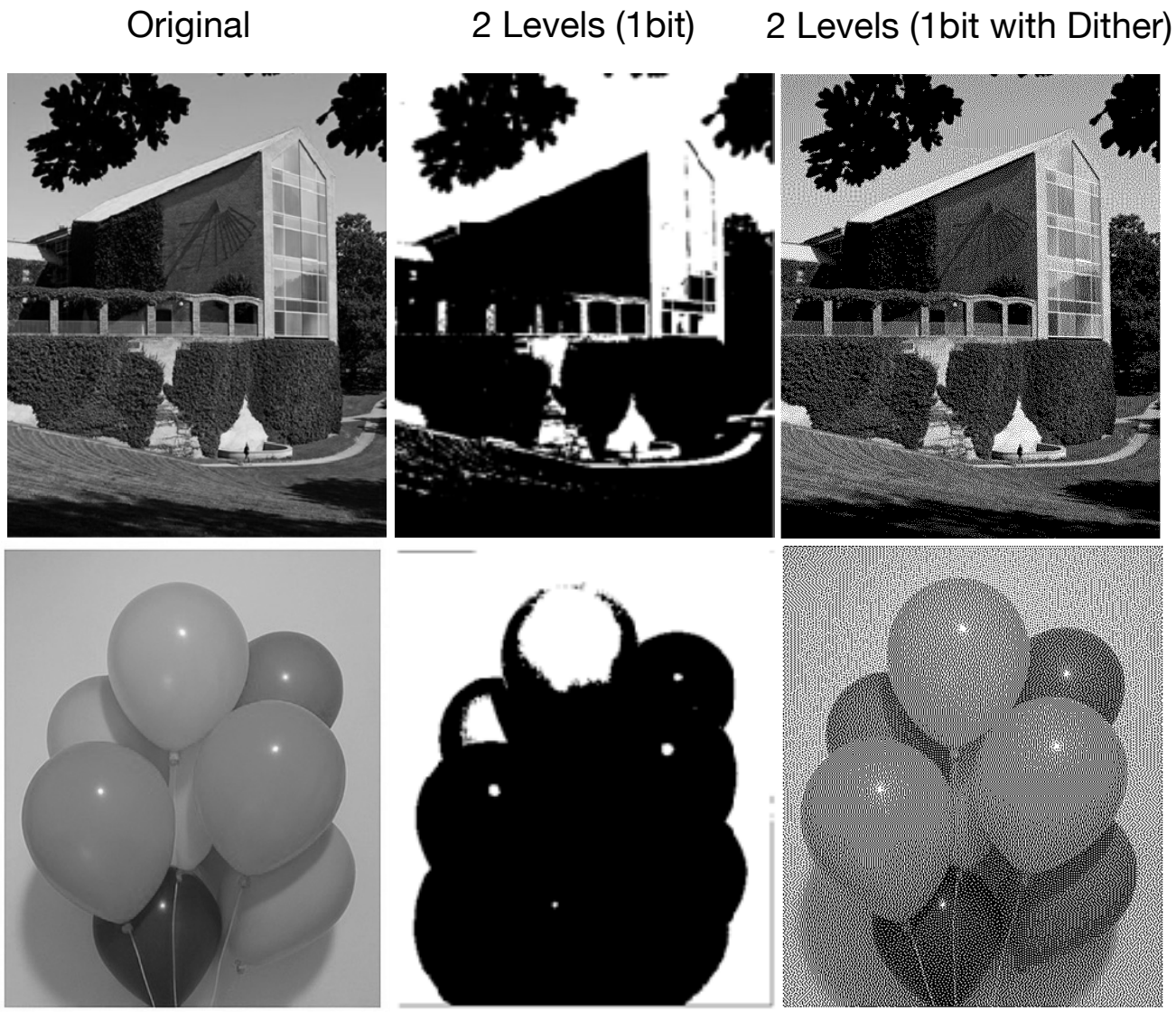
- From the 2nd Lecture
  - Super Mario World (Gamboy)
  - No 3D graphics
  - Grey scale (2 bits, 4 shades)





Quantization and Dithering

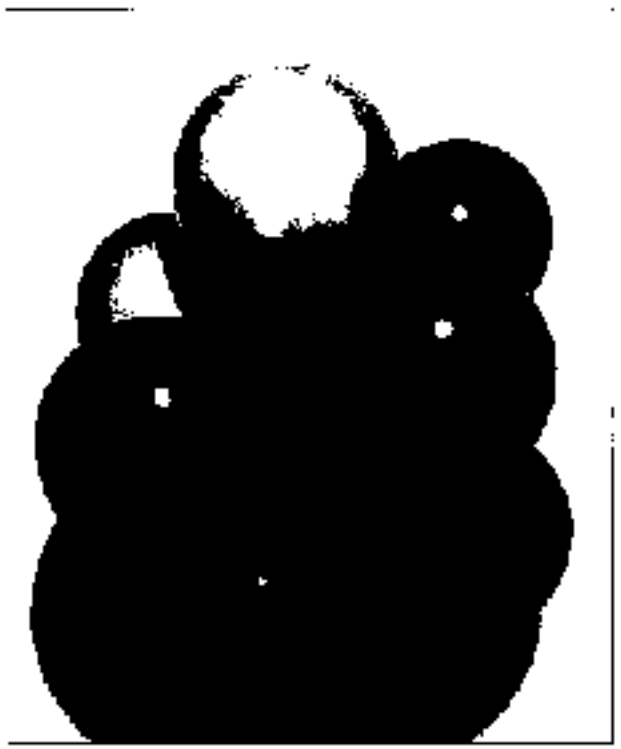
- Dither uses noise or patterns to randomise or reduce quantisation error
- Reducing large-scale patterns (contouring or banding)
- Distribute errors among pixels
  - Exploit perceptual spatial integration
  - Display greater range of perceptible intensities



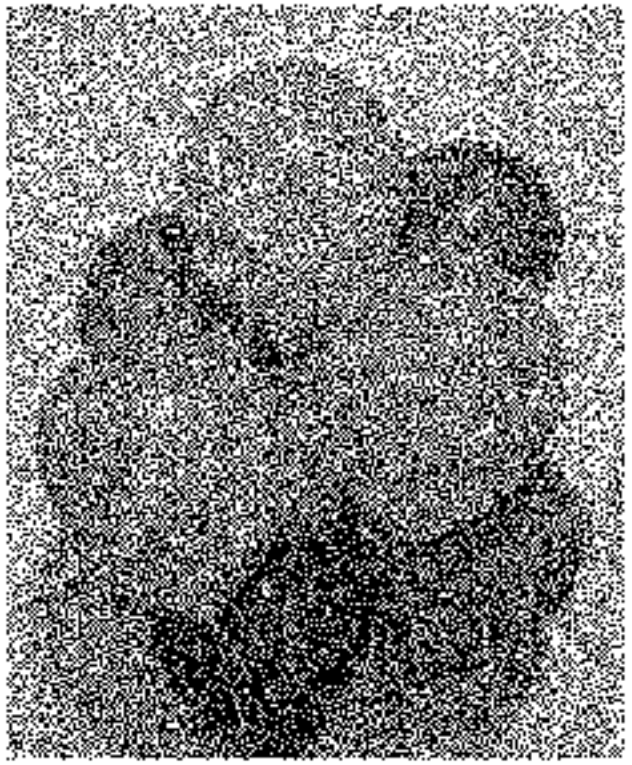
Quantization and Dithering

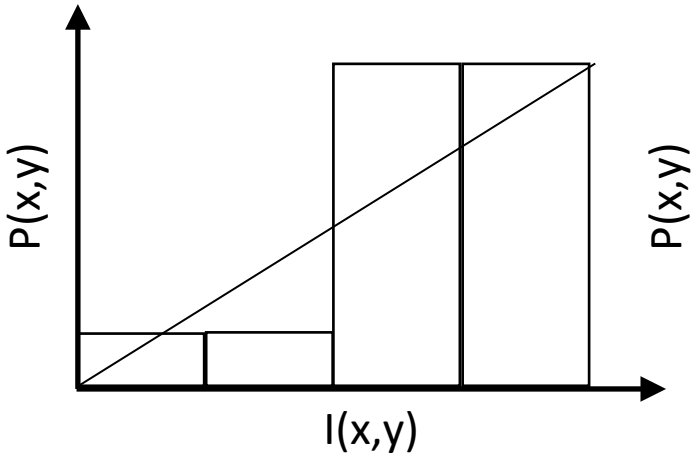
- Random Dither:
  - P(x, y) = round(I(x, y) + noise(x,y))
    - Where round() chooses nearest value that can be represented.

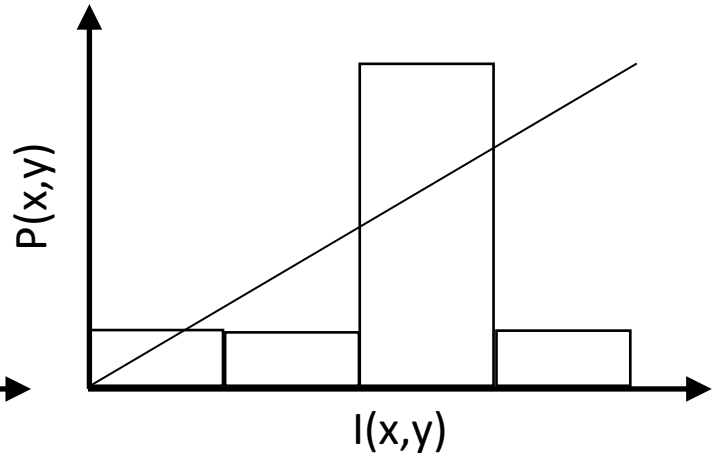
Original



2 Levels (1bit with Random Dither)



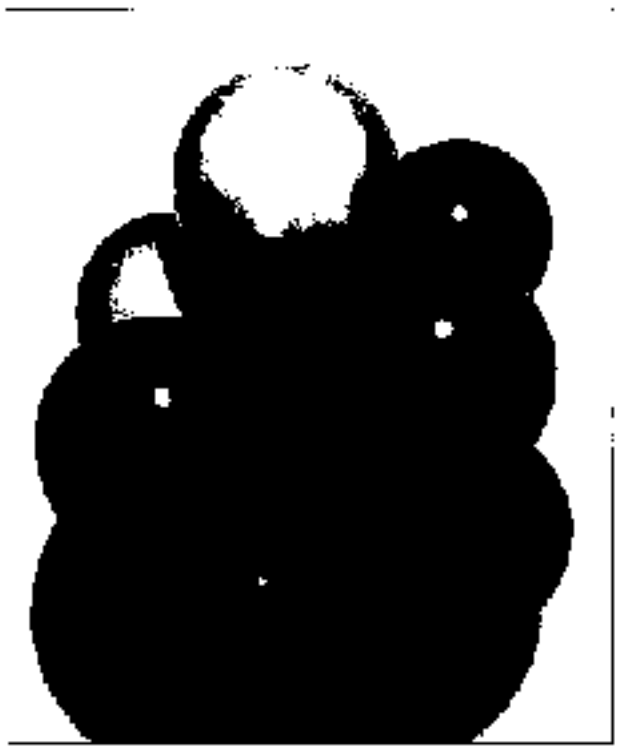




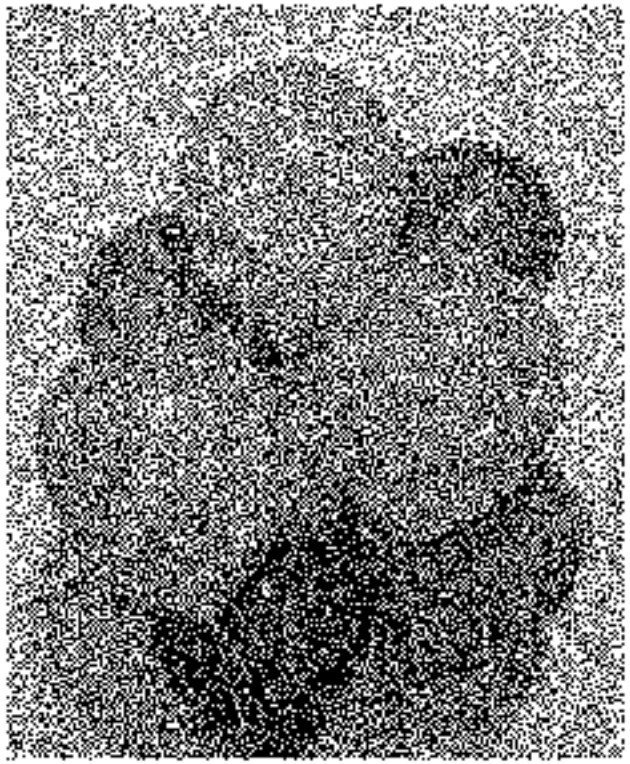
Quantization and Dithering

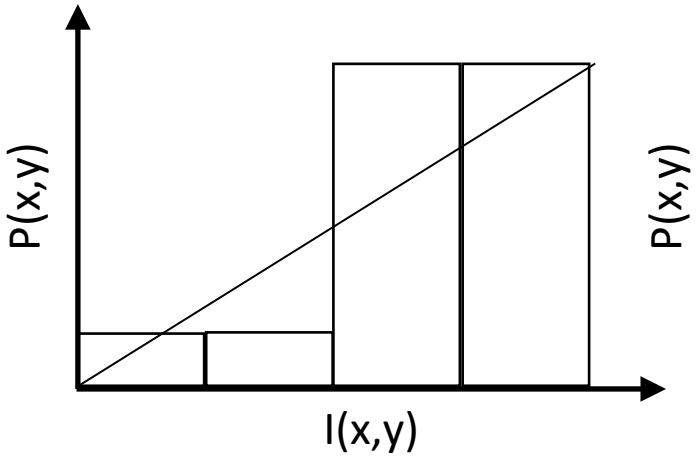
- Random Dither:
  - P(x, y) = round(I(x, y) + noise(x,y))
    - Where round() chooses nearest value that can be represented.

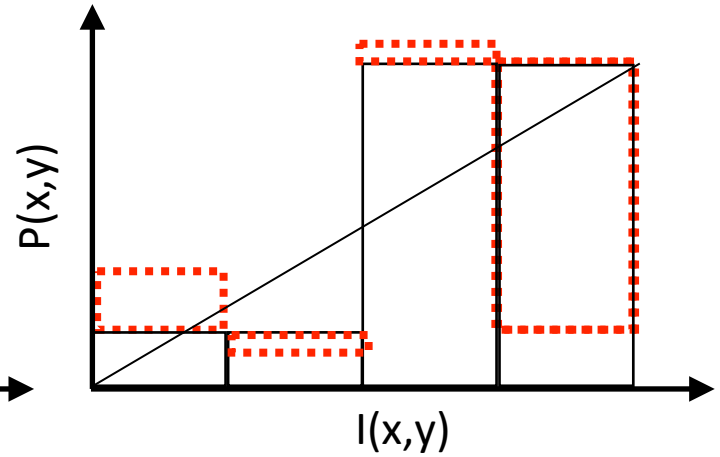
Original



2 Levels (1bit with Random Dither)



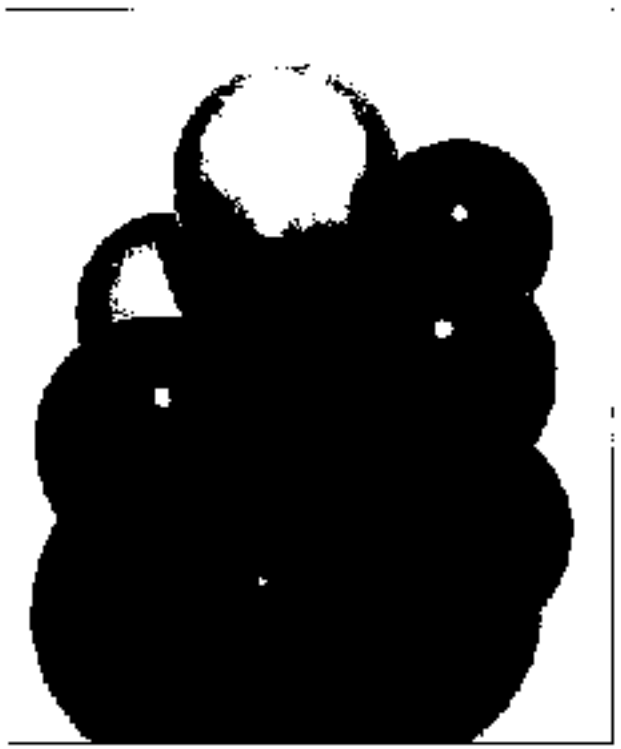




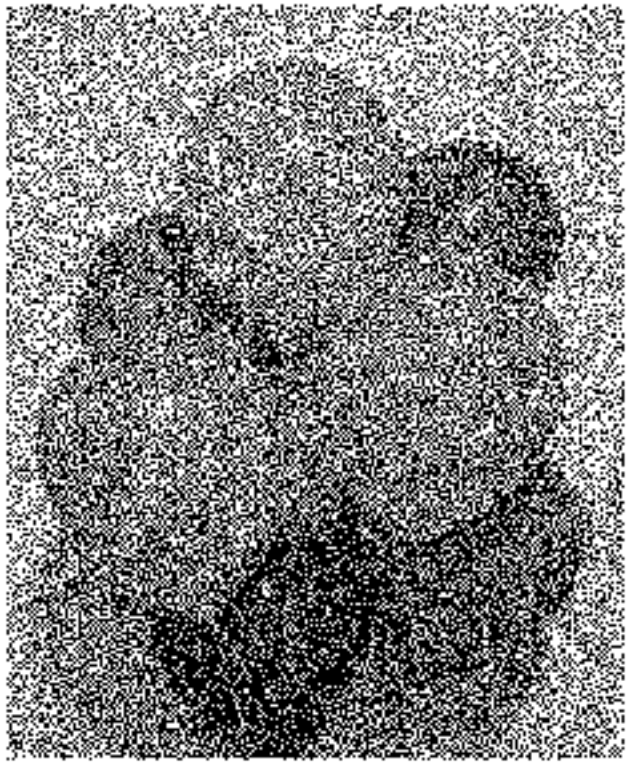
Quantization and Dithering

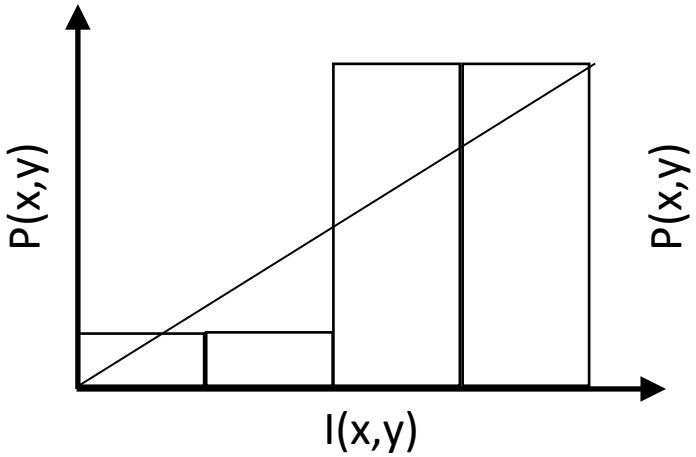
- Random Dither:
  - P(x, y) = round(I(x, y) + noise(x,y))
    - Where round() chooses nearest value that can be represented.

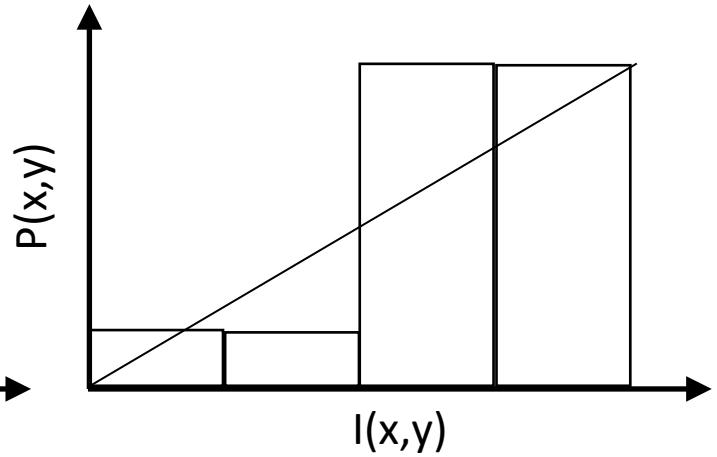
Original



2 Levels (1bit with Random Dither)



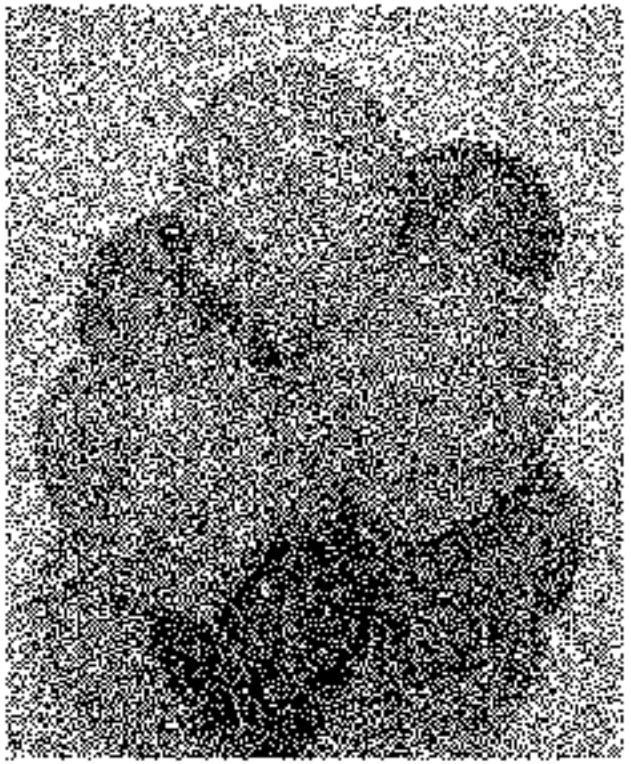




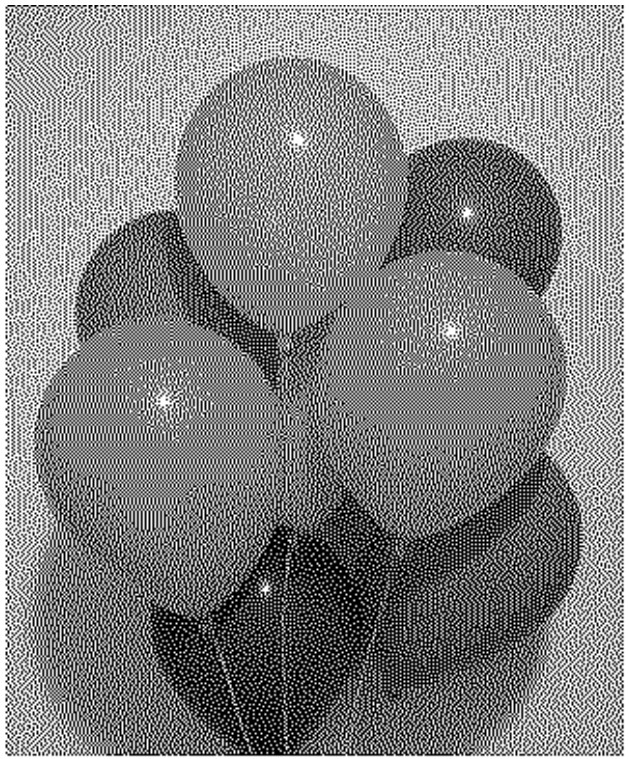
Quantization and Dithering

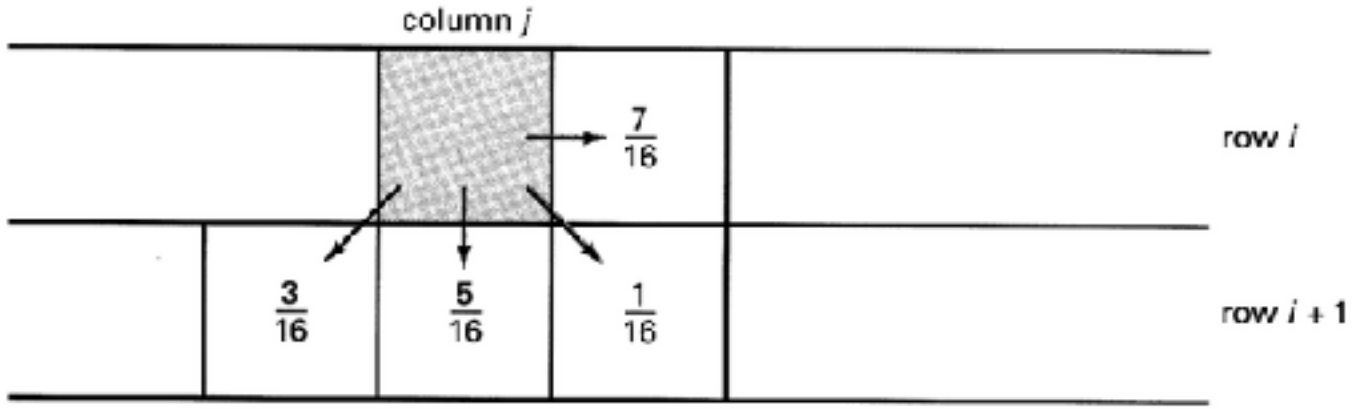
- Diffusion Dither:
  - E.g. Floyd–Steinberg dithering
  - Spread quantisation error over neighbouring pixels
    - Error dispersed to pixels right and below
    - Floyd-Steinberg weights:
    - 3/16 + 5/16 + 1/16 + 7/16 = 1.0

2 Levels (1bit with Random Dither)



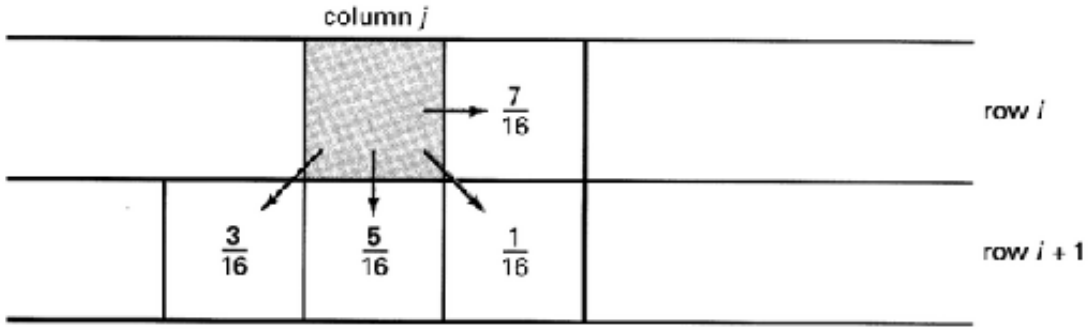
2 Levels (1bit with Diffusion Dither)





Quantization and Dithering

- Diffusion Dither:
  - E.g. Floyd–Steinberg dithering

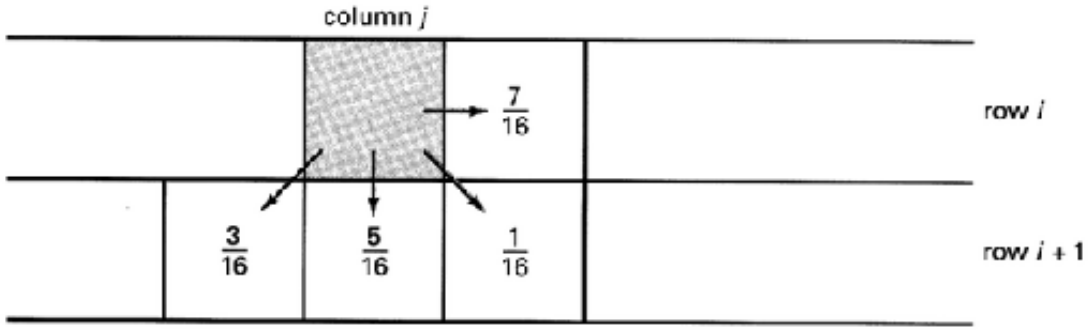


Quantisation

```
for each y from top to bottom do
    for each x from left to right do
        oldpixel = pixels[x][y]
        newpixel = find_closest_palette_color(oldpixel)
        pixels[x][y] = newpixel
        quant_error = oldpixel - newpixel
        pixels[x + 1][y    ] = pixels[x + 1][y    ] + quant_error × 7 / 16
        pixels[x - 1][y + 1] = pixels[x - 1][y + 1] + quant_error × 3 / 16
        pixels[x    ][y + 1] = pixels[x    ][y + 1] + quant_error × 5 / 16
        pixels[x + 1][y + 1] = pixels[x + 1][y + 1] + quant_error × 1 / 16
```

Quantization and Dithering

- Diffusion Dither:
  - E.g. Floyd–Steinberg dithering




```
for each y from top to bottom do
    for each x from left to right do
        oldpixel = pixels[x][y]
        newpixel = find_closest_palette_color(oldpixel)
        pixels[x][y] = newpixel
        quant_error = oldpixel - newpixel
        pixels[x + 1][y    ] = pixels[x + 1][y    ] + quant_error × 7 / 16
        pixels[x - 1][y + 1] = pixels[x - 1][y + 1] + quant_error × 3 / 16
        pixels[x    ][y + 1] = pixels[x    ][y + 1] + quant_error × 5 / 16
        pixels[x + 1][y + 1] = pixels[x + 1][y + 1] + quant_error × 1 / 16
```

Quantisation

Diffusion dithering by distributing quantisation error

Quantization and Dithering

8 colour quantisation

8 colour quantisation with Diffusion Dither

2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg

- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median

- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

Image Filters - Linear Filter

Linear Filters

Linear Filters and Image Convolutions

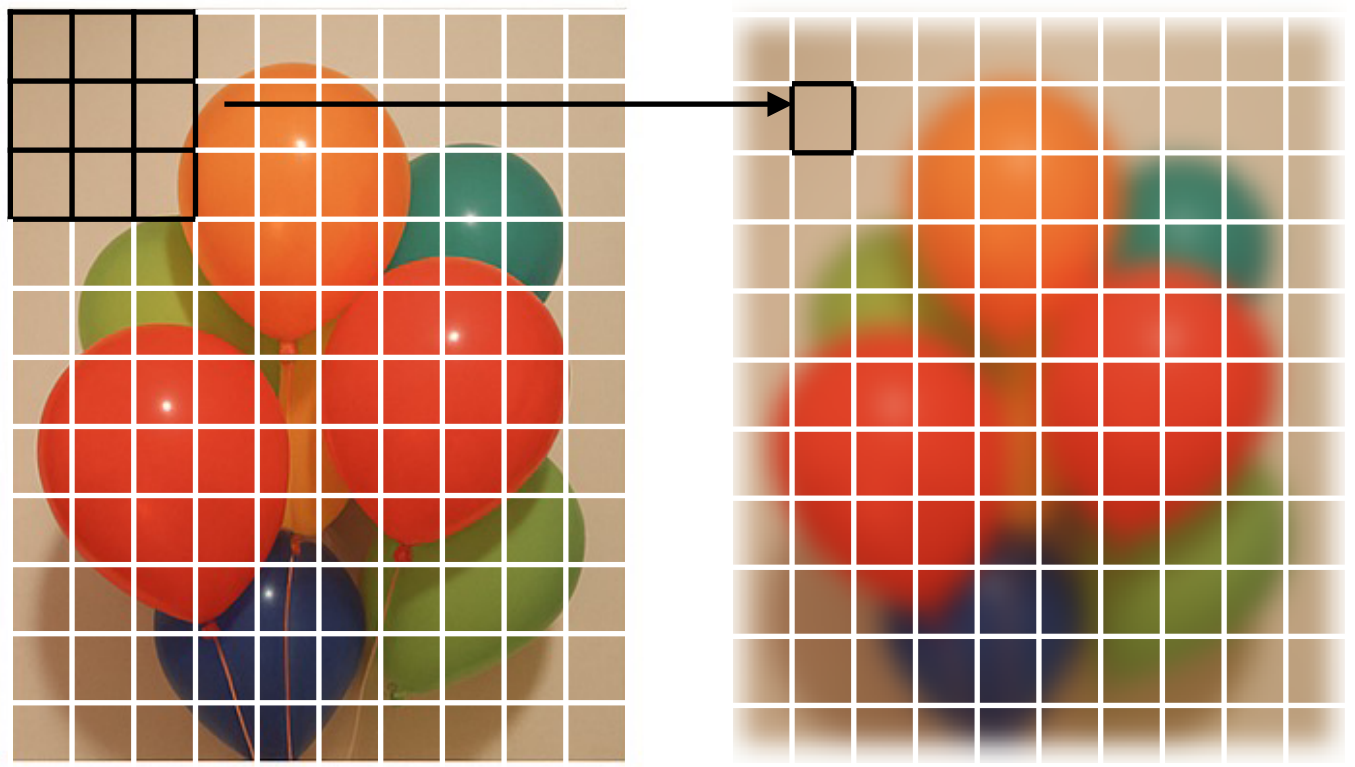
- Image filtering with linear filters
  - Construct array **R** of same size as original image **F**
  - Fill each location in **R** with a weighted sum of the pixel values of corresponding location's neighbourhood in **F**
  - Use the same weights each time
  - Different sets of weights represent different filters
  - ***Convolution*** with a (filter) ***kernel***
- **Shift-invariant:** filtered result depends on image neighbourhood but not on position of neighbourhood
- **Linear:** filtering the sum of two images is equivalent as summing the two filtered images
- **Shift-Invariant Linear Systems (SILS)**

F

R

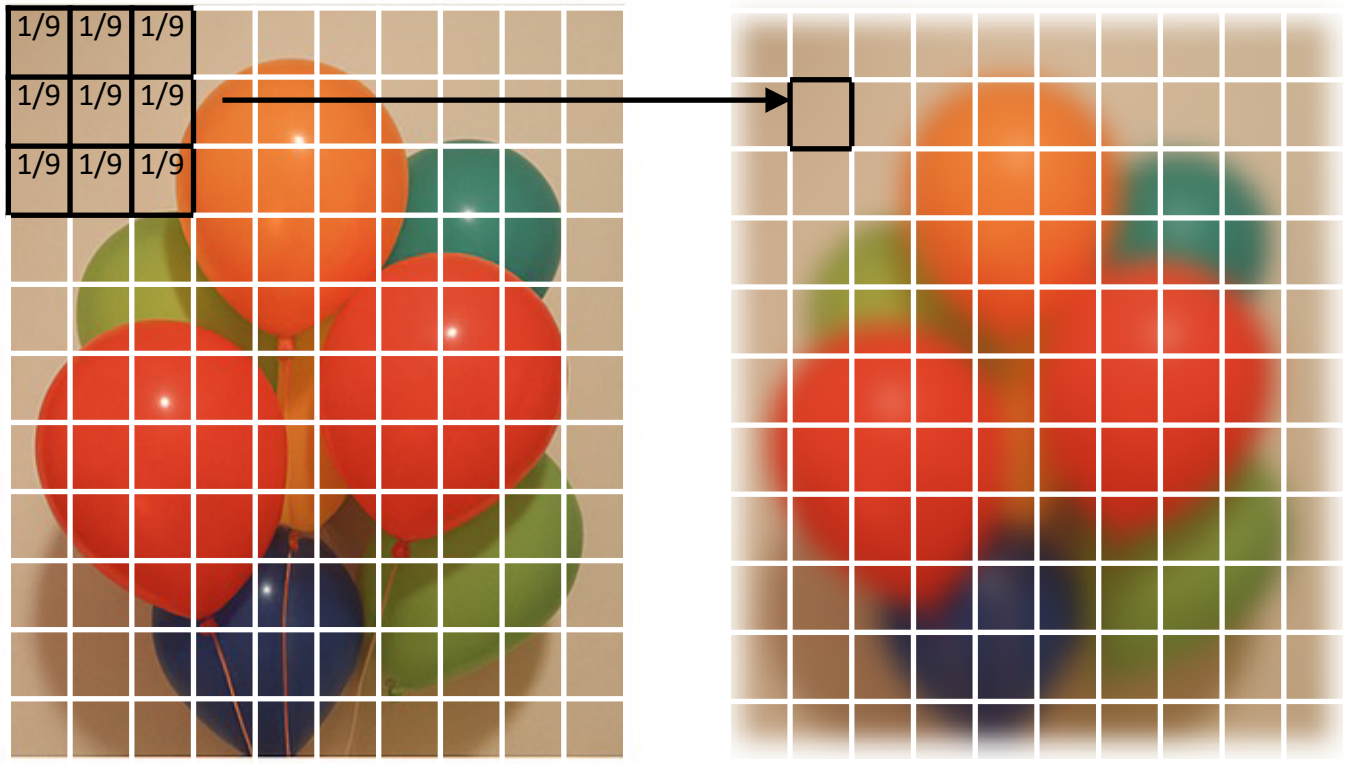
Linear Filters and Image Convolutions

- Image filtering with linear filters
  - Construct array **R** of same size as original image **F**
  - Fill each location in **R** with a weighted sum of the pixel values of corresponding location's neighbourhood in **F**
  - Use the same weights each time
  - Different sets of weights represent different filters
  - ***Convolution*** with a (filter) ***kernel***
- **Shift-invariant:** filtered result depends on image neighbourhood but not on position of neighbourhood
- **Linear:** filtering the sum of two images is equivalent as summing the two filtered images
- **Shift-Invariant Linear Systems (SILS)**



Linear Filters and Image Convolutions

- Image filtering with linear filters
  - Construct array **R** of same size as original image **F**
  - Fill each location in **R** with a weighted sum of the pixel values of corresponding location's neighbourhood in **F**
  - Use the same weights each time
  - Different sets of weights represent different filters
  - ***Convolution*** with a (filter) ***kernel***
- **Shift-invariant:** filtered result depends on image neighbourhood but not on position of neighbourhood
- **Linear:** filtering the sum of two images is equivalent as summing the two filtered images
- **Shift-Invariant Linear Systems (SILS)**

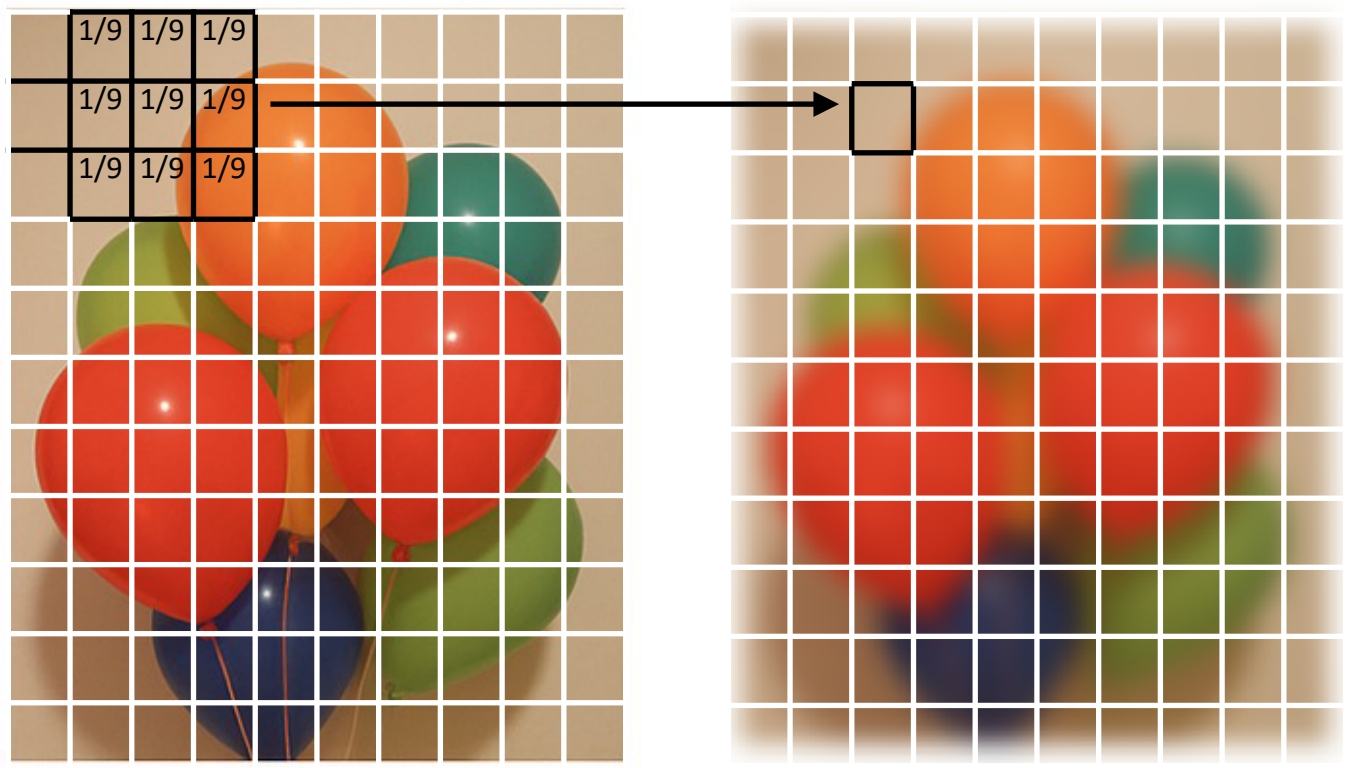


1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Kernel H

Linear Filters and Image Convolutions

- Image filtering with linear filters
  - Construct array **R** of same size as original image **F**
  - Fill each location in **R** with a weighted sum of the pixel values of corresponding location's neighbourhood in **F**
  - Use the same weights each time
  - Different sets of weights represent different filters
  - ***Convolution*** with a (filter) ***kernel***
- **Shift-invariant:** filtered result depends on image neighbourhood but not on position of neighbourhood
- **Linear:** filtering the sum of two images is equivalent as summing the two filtered images
- **Shift-Invariant Linear Systems (SILS)**



1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Kernel H

Linear Filters and Image Convolutions

- Convolution
  - The patterns of weights is usually referred to as **kernel** H
  - The process of applying H to an image F is known as **convolution**
  - Say: "*The convolution of H with F results in R*"
  - The kernel H can be defined as a continuous function or a finite matrix of discrete weights
  - What is the kernel for our local average example?
  - How large is it?
    - $(2k)^2$ weights

$$R_{i,j} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{i+k} \sum_{v=j-k}^{j+k} F_{u,v}$$

$$R_{i,j} = \sum_{u=i-k}^{i+k} \sum_{v=j-k}^{j+k} H_{i-u,j-v} \cdot F_{u,v}$$

$$R = H \otimes F$$

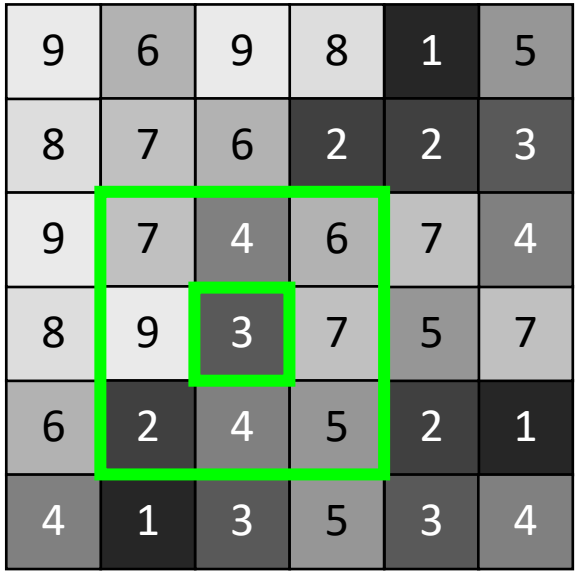
$$H_{i,j} = \frac{1}{(2k+1)^2}$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

H

Linear Filters and Image Convolutions

- Kernels used for filters
- Array of weighting values
- Typically 3×3, 5×5, etc.

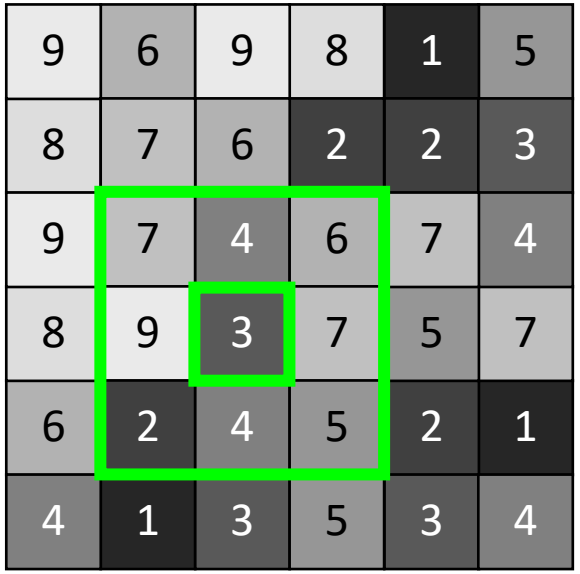


- At each pixel in the image:
  - The kernel is overlaid so that the middle number is over the current pixel
  - Each weight is multiplied by the corresponding pixel value
  - The result of the filter is the sum of these products

$$H = \frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

Linear Filters and Image Convolutions

- Kernels used for filters
- Array of weighting values
- Typically 3×3, 5×5, etc.



- Example

$$[F * H](3,4) = \frac{1}{16}(7 \times 1 + 4 \times 2 + 6 \times 1 + 9 \times 2 + 3 \times 4 + 7 \times 2 + 2 \times 1 + 4 \times 2 + 5 \times 1)$$

$$= \frac{1}{16}(7 + 8 + 6 + 18 + 12 + 14 + 2 + 8 + 5)$$

$$= 5$$

$$H = \frac{1}{16}\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

Linear Filters and Image Convolutions

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

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

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 3 & -\frac{1}{2} \\ 0 & -\frac{1}{2} & 0 \end{bmatrix}$$

Linear Filters and Image Convolutions

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Box Blur

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

Gaussian Blur

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Sharpen

$$\begin{bmatrix} 0 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 3 & -\frac{1}{2} \\ 0 & -\frac{1}{2} & 0 \end{bmatrix}$$

Linear Filters and Image Convolutions

Original

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Edge horizontal

$$\begin{matrix} -\frac{1}{2} & 0 & \frac{1}{2} \\ -1 & 0 & 1 \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{matrix}$$

Edge vertical

$$\begin{matrix} -\frac{1}{2} & -1 & -\frac{1}{2} \\ -0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} \end{matrix}$$

Edge (Sobel Operator)

Linear Filters and Image Convolutions

Original

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Edge horizontal

$$\begin{matrix} -\frac{1}{2} & 0 & \frac{1}{2} \\ -1 & 0 & 1 \\ -\frac{1}{2} & 0 & \frac{1}{2} \end{matrix}$$

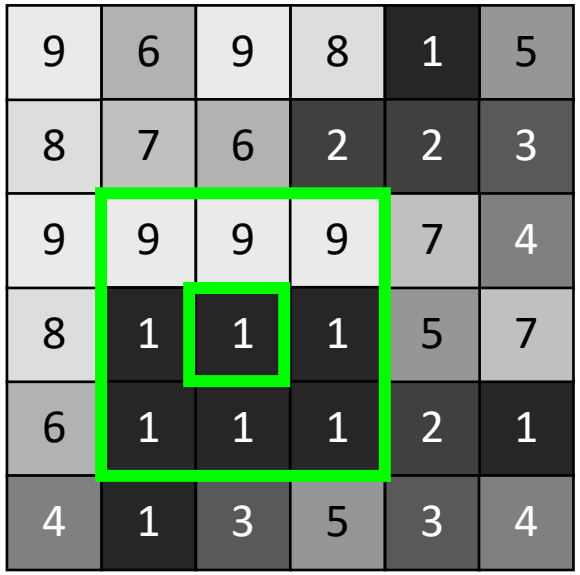
Edge vertical

$$\begin{matrix} -\frac{1}{2} & -1 & -\frac{1}{2} \\ -0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} \end{matrix}$$

Edge (Sobel Operator)

Linear Filters and Image Convolutions

- Edge filter (Sobel Operator):



$$H = \begin{bmatrix} -\frac{1}{2} & -1 & -\frac{1}{2} \\ 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} \end{bmatrix}$$

$$[F * K]_{2,3} = (-0.5 \times 9 + -1 \times 9 + -0.5 \times 9 + \ldots)$$

$$= -16$$

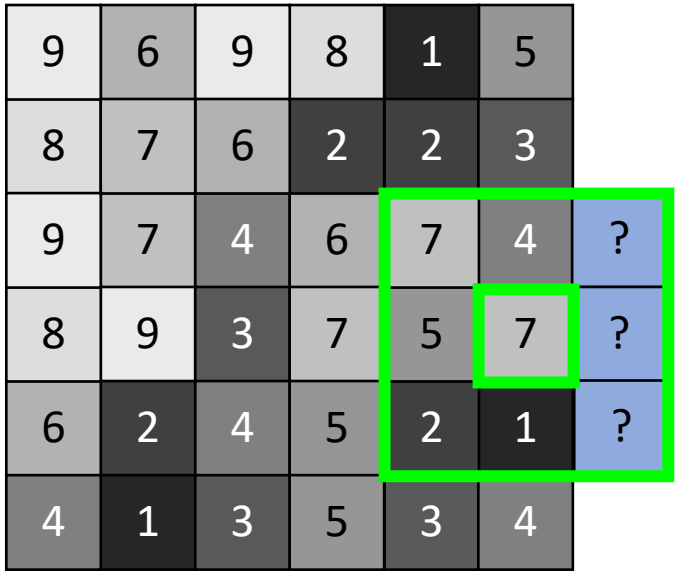
Linear Filters and Image Convolutions

- Implementation:

```
// Image is width x height, filtered into Result
// Kernel has ‘radius’ r, so is (2r+1) square
for (y = 0; y < height; ++y) {
    for (x = 0; x < width; ++x) {
        Result[x][y] = 0;
        for (dy = -r; dy <= r; ++dy) {
            for (dx = -r; dx <= r; ++dx) {
                Result[x][y] += Kernel[r+dx][r+dy] * Image[x+dx][y+dy];
            }
        }
    }
}
```

Linear Filters and Image Convolutions

- Problematic cases for Image Convolutions
- Edges of images:
  - Out of bounds errors
  - Can stay away from edges
  - Or use nearest valid pixel
- Results out of range
  - What if the sum is less than 0 or greater than 255?
  - Some filters give results in different ranges (e.g. edge detection)



Linear Filters and Image Convolutions

- Convolution Rules for SILS:
  - **Superimposition**: the sum of two filtered images is equivalent to the filtered image with the sum of the two kernels (component-wise addition)

$$H_1 \otimes R + H_2 \otimes R = (H_1 + H_2) \otimes R$$

  - **Scaling:** scaling of a filtered image is equivalent to filtering the image with a scaled kernel

$$(kH) \otimes R = k\,(H \otimes R)$$

  - **Symmetric:** the convolution of two filter to an image is symmetric

$$H_1 \otimes (H_2 \otimes R) = H_2 \otimes (H_1 \otimes R)$$

  - **Associative:** convolution is associative, which means that we can find a single kernel that behaves like the composition of two kernels

$$H_1 \otimes (H_2 \otimes R) = (H_1 \otimes H_2) \otimes R$$

Linear Filters and Image Convolutions

- Something to discuss:
  - Many convolutions have kernels where the values add up to 1. Why is this the case?
  - What are the largest and smallest values you might get from the different filters?
  - How can you apply these filters to colour images?

F

R

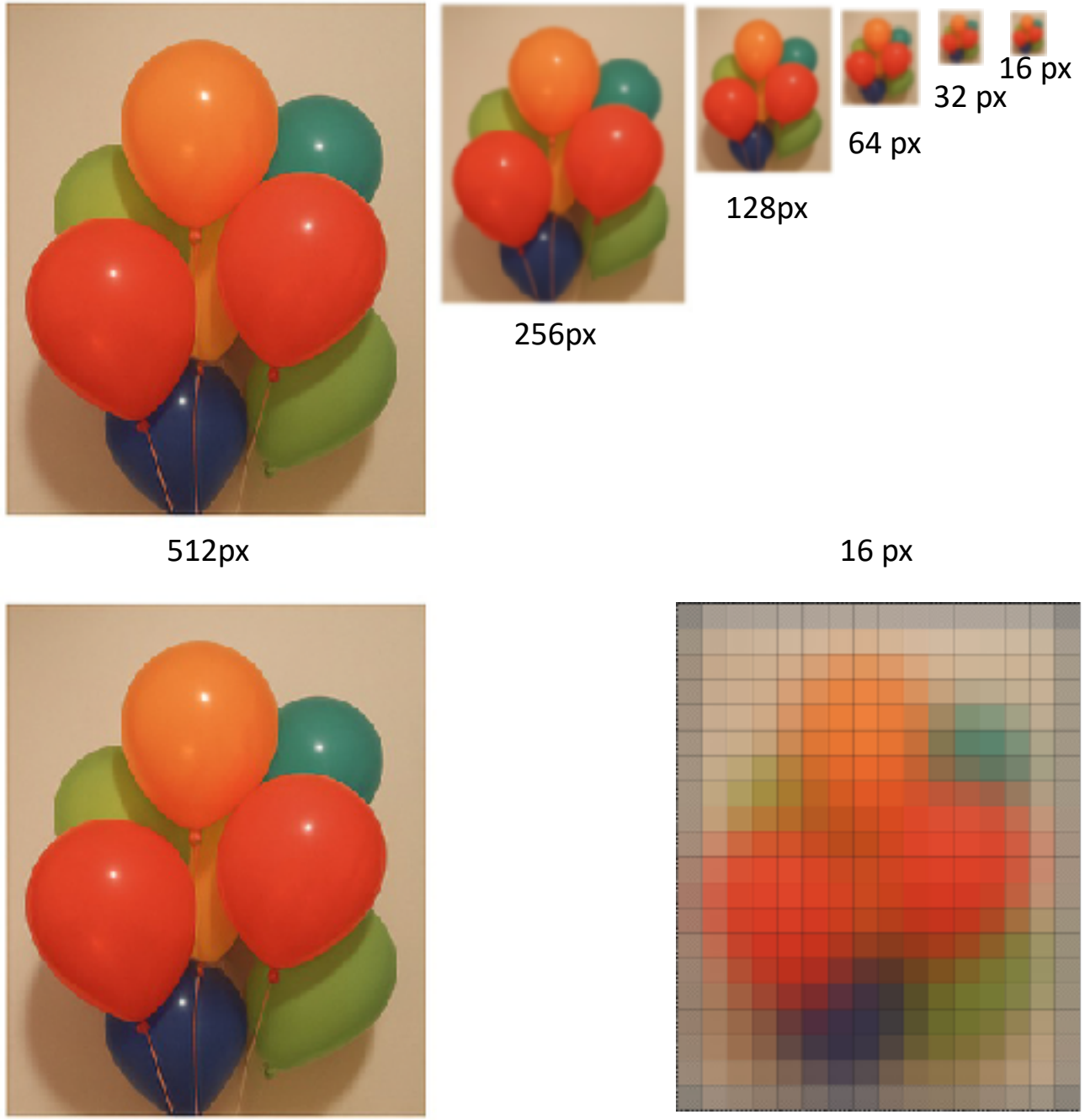
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Kernel H

Image Pyramids

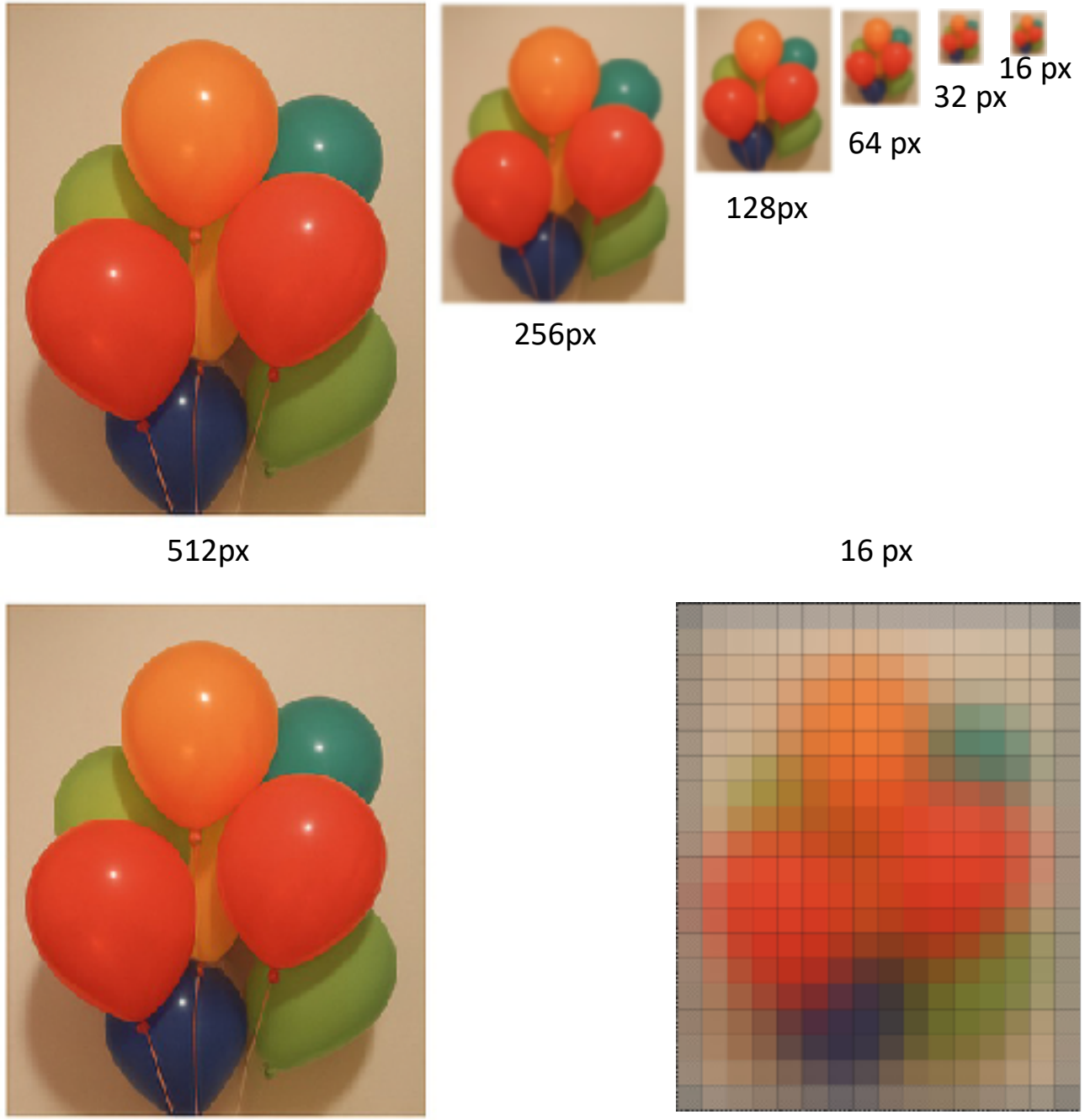
Convolutions

- Image Pyramids (here **Gaussian Pyramid**):
  - An image pyramid is a collection of representations of the same image at different scales
  - Typically, each layer is half the width and half the height of the previous layer
  - At each layer, a Gaussian pyramid stores a smoothened (Gaussian kernel) and downsampled version of the previous layer



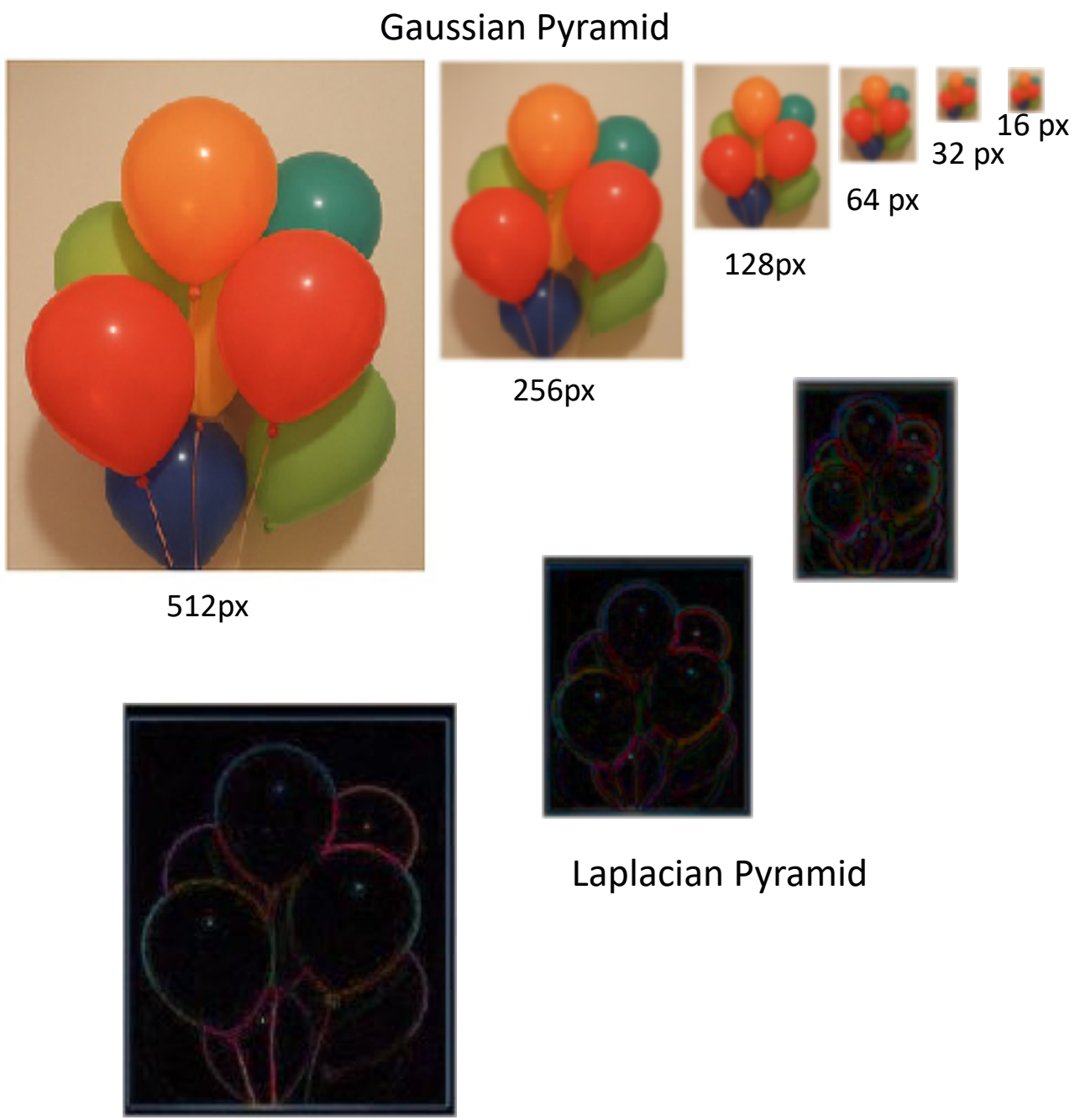
Convolutions

- What are possible applications for Gaussian pyramids?
  - **Search over scale:** scale invariant pattern matching, search with same pattern over multiple scales
  - **Spatial search:** search matches in coarse layers of two and track downwards, thereby search regions are decreased
  - **Feature tracking:** accept features found in fine-scale images only if they can be tracked across different coarser scales to suppress noise
    - More about that next time!



Convolutions

- A Gauss filter is a low-pass filter
  - Gaussian pyramid successively removes high frequencies
  - What happens if two consecutive layers are subtracted?
    - The resulting image contains the removed (via Gauss filter) spatial frequencies
  - What happens if this is done for each layer of the Gaussian pyramid?
    - The result is another image pyramid, called **Laplacian** pyramid, that can be thought of as the response of a band-pass filter



2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg

- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median

- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg
- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median
- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

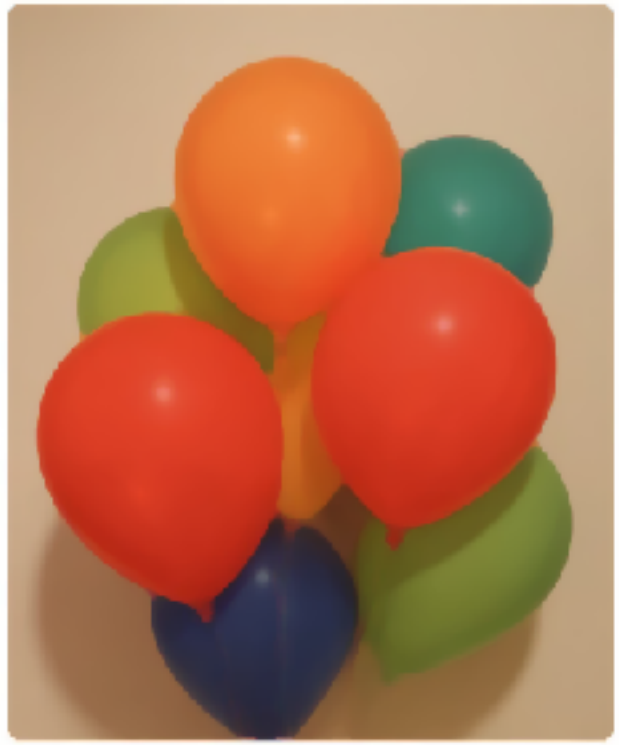
Image Filters - Non-Linear Filters

Non-Linear Filters

- Not all filters are implemented as convolution operations
- E.g. Median filter
  - Take a local neighbourhood (e.g. a 3×3 or 5×5 window)
  - Sort the pixel values
  - Use the middle value as the result
  - Has a smoothing effect, but preserves sharp edges

Original

Median



2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg

- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median

- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

2D Image Processing

- Luminance
  - Brightness
  - Contrast
  - Histogram equalization
  - Gamma
- Color
  - Grayscale
  - Saturation
- Dithering
  - Quantization
  - Random dither
  - Floyd-Steinberg

- Linear filtering
  - Convolution
  - Blur & Sharpen
  - Edge detection
  - Image Pyramids
- Non-linear filtering
  - Median

- Image Feature Detection
  - Edges (e.g Canny Edge)
  - Corners (e.g. Shi-Tomasi, Harris Corner)
  - Blobs (e.g. Difference of Gaussians)

Image Features & Feature Detection

Image Features & Feature Detection

- Often required:
    - Features in an image that can be reliably detected in different images of the same scene
- A ‘feature’ or ‘key point’ is
    - A 2D location in an image
    - Accurately/precisely located
    - Can be found in different views

What are good image features?

Image Features & Feature Detection

- Often required:
    - Features in an image that can be reliably detected in different images of the same scene
- A ‘feature’ or ‘key point’ is
    - A 2D location in an image
    - Accurately/precisely located
    - Can be found in different views
- Feature descriptors
    - Used to tell features apart
    - Ideally robust to changes in lighting, viewpoint, contrast, etc.
- Three common types of features
    - Edges – gradient-based
    - Corners – gradient-based
    - Blobs – bright/dark centre/surround

What are good image features?

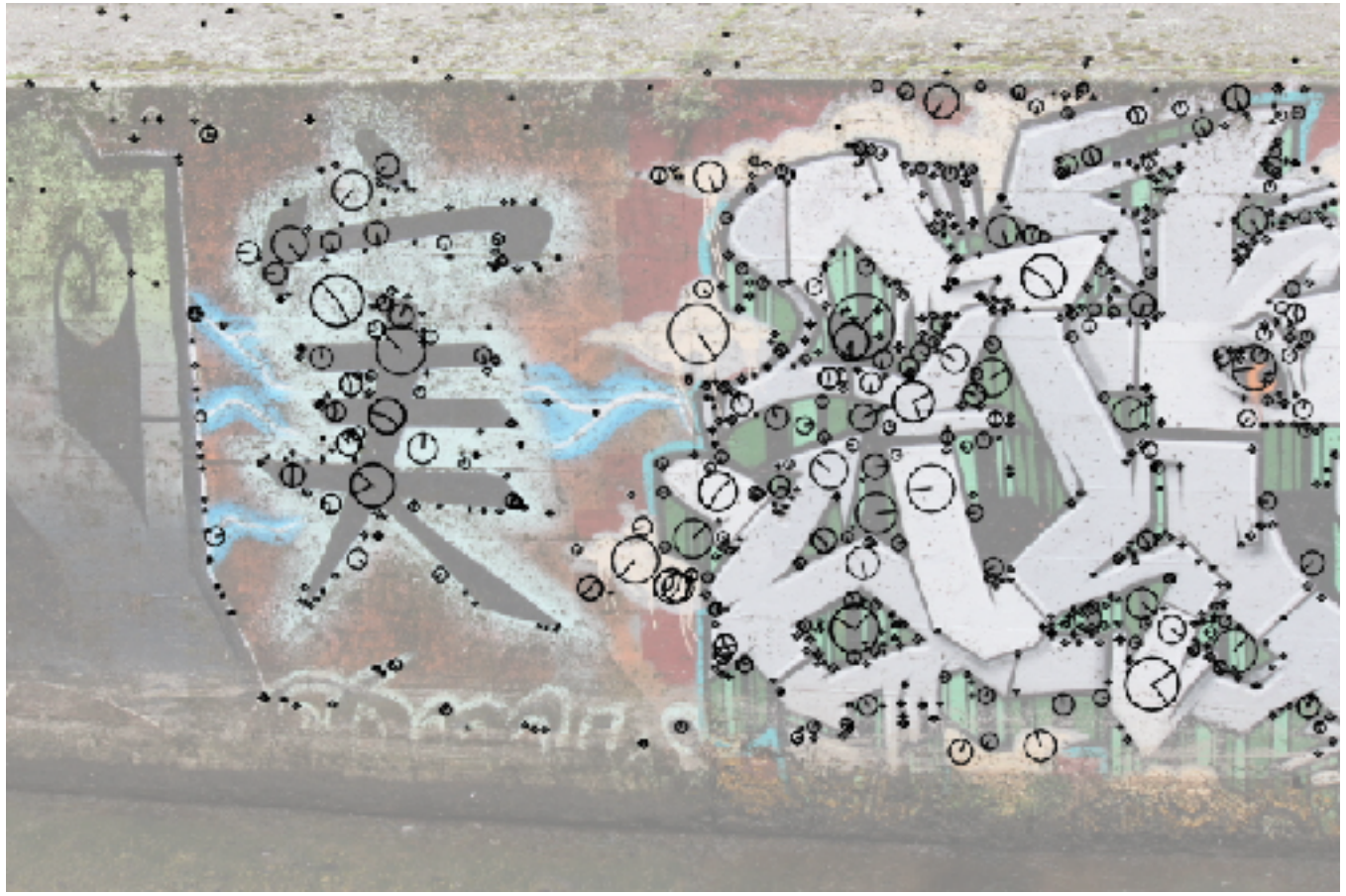
Image Features & Feature Detection

- Often required:
    - Features in an image that can be reliably detected in different images of the same scene
- A ‘feature’ or ‘key point’ is
    - A 2D location in an image
    - Accurately/precisely located
    - Can be found in different views
- Feature descriptors
    - Used to tell features apart
    - Ideally robust to changes in lighting, viewpoint, contrast, etc.
- Three common types of features
    - Edges – gradient-based
    - Corners – gradient-based
    - Blobs – bright/dark centre/surround

Example: Corners

Image Features & Feature Detection

- Often required:
    - Features in an image that can be reliably detected in different images of the same scene
- A ‘feature’ or ‘key point’ is
    - A 2D location in an image
    - Accurately/precisely located
    - Can be found in different views
- Feature descriptors
    - Used to tell features apart
    - Ideally robust to changes in lighting, viewpoint, contrast, etc.
- Three common types of features
    - Edges – gradient-based
    - Corners – gradient-based
    - Blobs – bright/dark centre/surround



Example: Blobs

Edge Detection - Canny Edge Detector

- Canny Edge Detector
  - Step 1: Reduce noise (> Gaussian filter G_σ)

Original

Edge Detection - Canny Edge Detector

- Canny Edge Detector
  - Step 1: Reduce noise (> Gaussian filter G_σ)
  - Step 2: Compute gradients in x and y direction (> Sobel operators H_u, H_v), gradient is a vector $\mathbf{g} = \begin{bmatrix} H_u & H_v \end{bmatrix}^T$

Original

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Edge Detection - Canny Edge Detector

- Canny Edge Detector
  - Step 1: Reduce noise (> Gaussian filter G_σ)
  - Step 2: Compute gradients in x and y direction (> Sobel operators H_u, H_v), gradient is a vector $\mathbf{g} = \begin{bmatrix} H_u & H_v \end{bmatrix}^T$
  - Step 3: compute gradient magnitudes (edge strength) and gradient directions (edge directions)

Original

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Magnitude

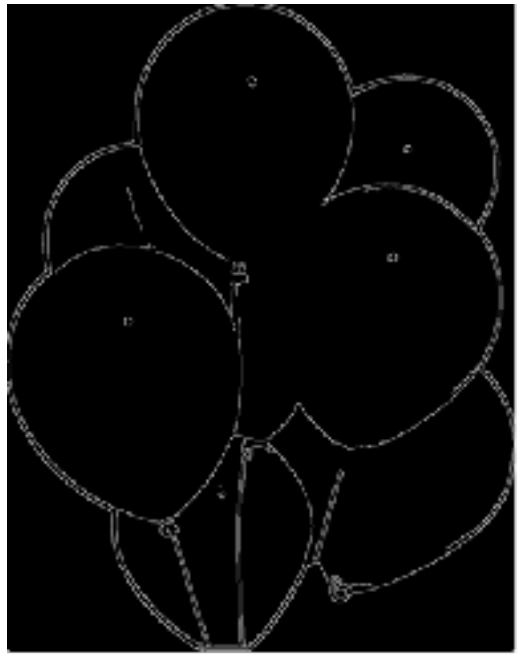
Edge Detection - Canny Edge Detector

- Canny Edge Detector
  - Step 1: Reduce noise (> Gaussian filter G_σ)
  - Step 2: Compute gradients in x and y direction (> Sobel operators H_u, H_v), gradient is a vector $\mathbf{g} = \begin{bmatrix} H_u & H_v \end{bmatrix}^T$
  - Step 3: compute gradient magnitudes (edge strength) and gradient directions (edge directions)
  - Step 4: select points with maximum magnitudes (above threshold T1)
  - Step 5: non-maximum suppression with hysteresis:
    - For all points with magnitudes above high-threshold T1
    - Trace along the edge (in edge direction) suppress all pixels that are not on traced line (pixel by pixel) and mark as visited
    - Stop tracing when point's magnitude is below low-threshold T2 or already visited

Original

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Edges using Canny Edge Detector



Magnitude

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

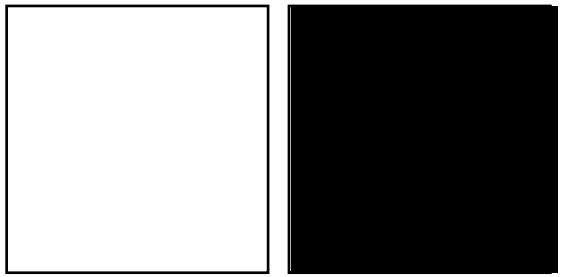
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



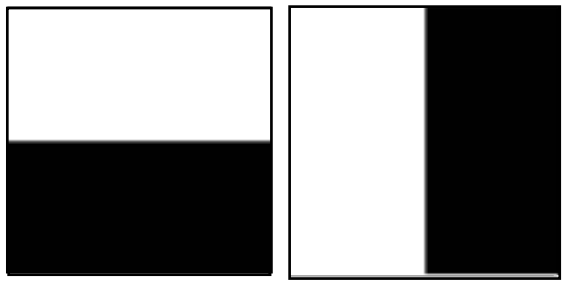
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

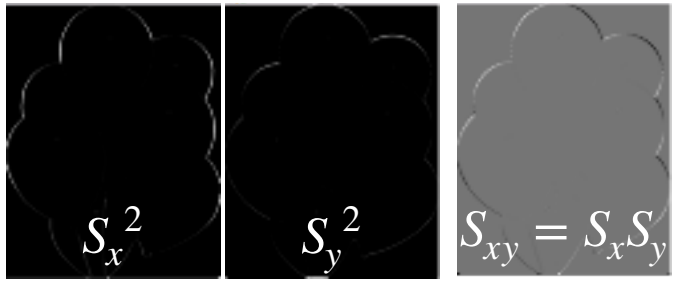
$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$





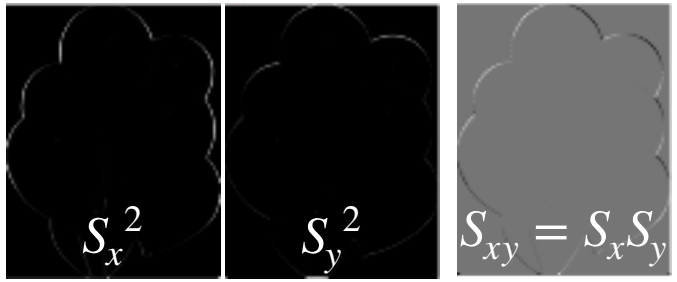
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low
      - Diag. edge: A,B and C are high (removed later)

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

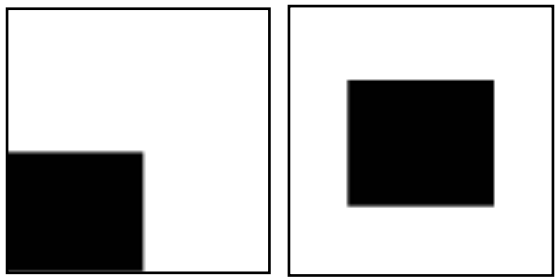


$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low
      - Diag. edge: A,B and C are high (removed later)
      - Corner: A,B and C are high



$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

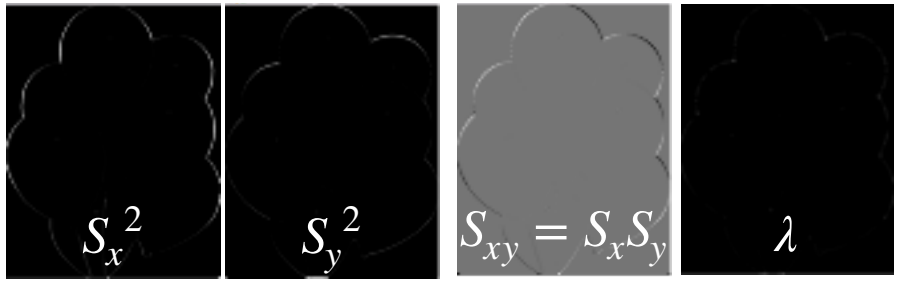
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low
      - Diag. edge: A,B and C are high (removed later)
      - Corner: A,B and C are high
  - Step 3 (Evaluation of M): Computing of Eigenvalues (λ) for M (Shi-Tomasi) or approximation using determinant and trace of M (Harris), can be used to correctly remove diagonal edge

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



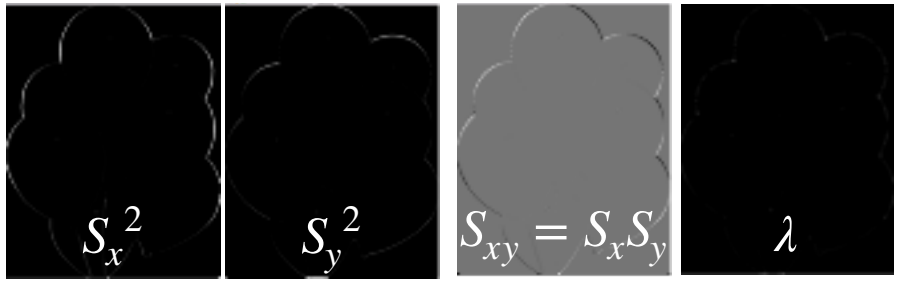
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low
      - Diag. edge: A,B and C are high (removed later)
      - Corner: A,B and C are high
  - Step 3 (Evaluation of M): Computing of Eigenvalues (λ) for M (Shi-Tomasi) or approximation using determinant and trace of M (Harris), can be used to correctly remove diagonal edge
  - Step 4 (Corner Detection): A point is identified if the eigenvalues or their approximation are large (meaning the local image patch changes significantly in all directions)
  - 6. Post-processing:
    Non-maximal suppression to keep only the strongest corners

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



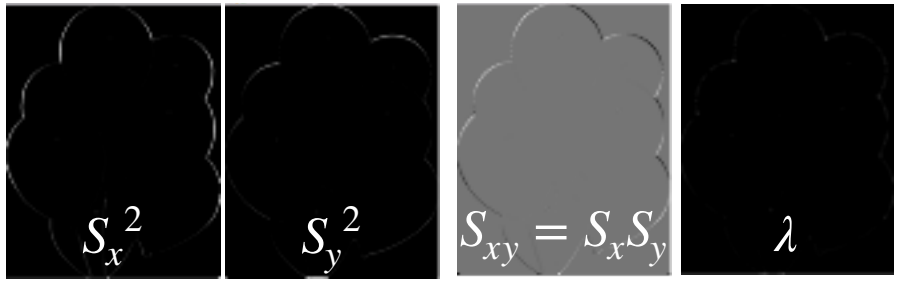
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low
      - Diag. edge: A,B and C are high (removed later)
      - Corner: A,B and C are high
  - Step 3 (Evaluation of M): Computing of Eigenvalues (λ) for M (Shi-Tomasi) or approximation using determinant and trace of M (Harris), can be used to correctly remove diagonal edge
  - Step 4 (Corner Detection): A point is identified if the eigenvalues or their approximation are large (meaning the local image patch changes significantly in all directions)
  - 6. Post-processing:
    Non-maximal suppression to keep only the strongest corners

$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$



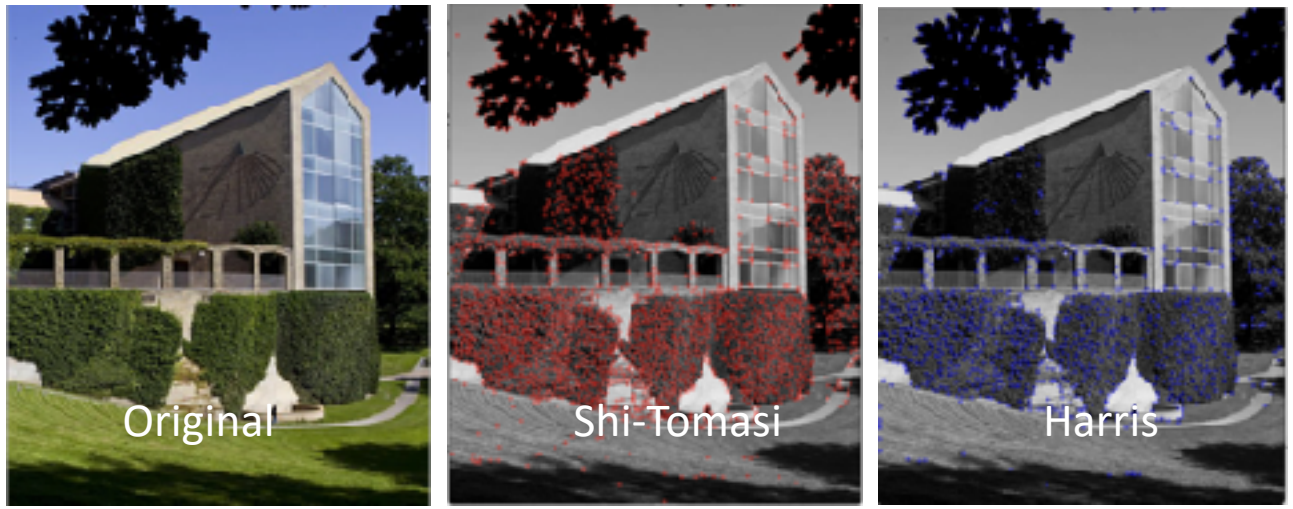
$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

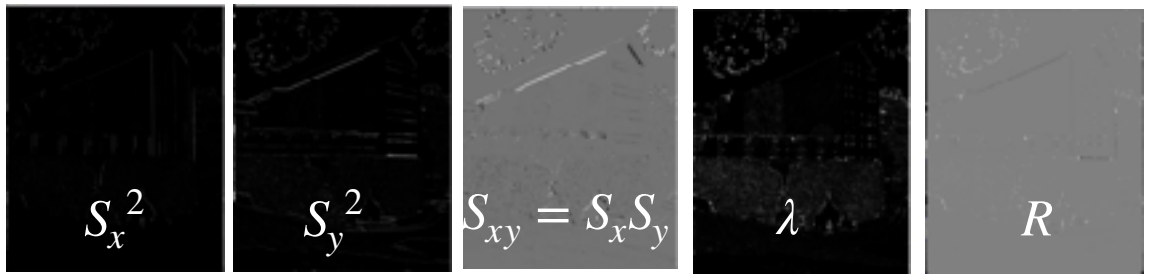
Corner Detection - Shi-Tomasi / Harris detector

Corner Detection - Shi-Tomasi / Harris detector

- Corner Detection
  - Step 1 (Gradients): For each pixel, the algorithm calculates the image's x and y derivatives S_x, S_y (gradients) using filters like Sobel.
  - Step 2 (Second Moment Matrix): The derivatives are used to compute the second moment matrix (M) for a small window around the pixel:
    - Properties of M:
      - Constant windows: A and B are low, C is low
      - Hori./vert. edge: A or B is high, C is low
      - Diag. edge: A,B and C are high (removed later)
      - Corner: A,B and C are high
  - Step 3 (Evaluation of M): Computing of Eigenvalues (λ) for M (Shi-Tomasi) or approximation using determinant and trace of M (Harris), can be used to correctly remove diagonal edge
  - Step 4 (Corner Detection): A point is identified if the eigenvalues or their approximation are large (meaning the local image patch changes significantly in all directions)
  - 6. Post-processing:
    Non-maximal suppression to keep only the strongest corners



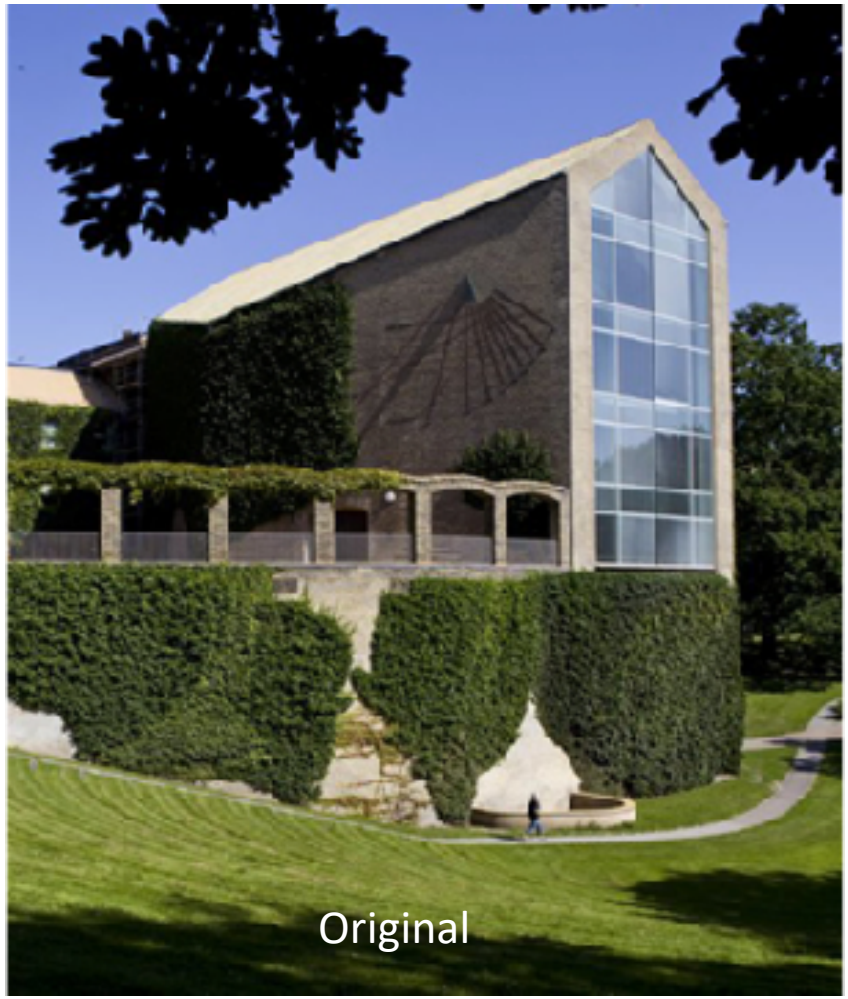
$$H_u = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

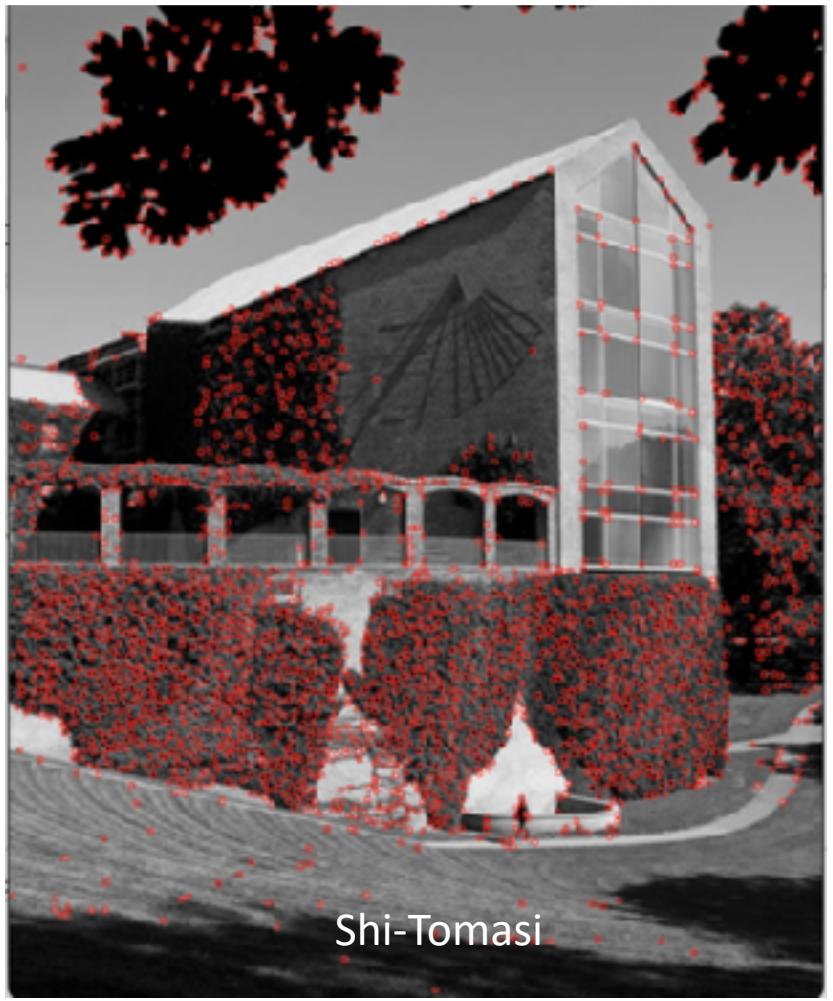


$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} (S_x)^2 & S_x S_y \\ S_x S_y & (S_y)^2 \end{bmatrix}$$

$$A = S_x^2, B = S_y^2, C = S_x S_y$$

Corner Detection - Shi-Tomasi / Harris detector





Tell me more about blobs!
And what can I do with all that stuff?

Next time :)

The end!