



Summary of Lecture 1

- “Let there be light” → incident light → reflectivity → reflected light
→ projection → sensitivity → $f_C(x', y')$
- Sampling: $f_C(x', y') \rightarrow f_C(i, j)$.
- Quantization: $f_C(i, j) \rightarrow \hat{f}_C(i, j) \in \{0, \dots, 255\}$.
- As a matrix: $\hat{f}_C(i, j) \rightarrow \hat{f}_{\text{gray}}(i, j) \rightarrow A(i, j)$.



Images as Matrices

- An image matrix ($N \times M$):

$$\mathbf{A} = \left[\begin{array}{cccc} A(0,0) & A(0,1) & A(0,2) & \dots A(0,M-1) \\ A(1,0) & A(1,1) & A(1,2) & \dots A(1,M-1) \\ \vdots & & & \\ A(N-1,0) & A(N-1,1) & A(N-1,2) & \dots A(N-1,M-1) \end{array} \right] \left. \vphantom{\begin{array}{c} \\ \\ \\ \end{array}} \right\} N \text{ rows } \text{☰}$$

- $A(i, j) \in \{0, 1, \dots, 255\}$.
- $A(i, j)$:
 - “Matrix case:” The matrix element (i, j) with value $A(i, j)$.
 - “Image case:” The pixel (i, j) with value $A(i, j)$.
 - Will use both terminologies.



Simple Processing - Transpose

- The transpose image **B** ($M \times N$) of **A** ($N \times M$) can be obtained as

$$B(j, i) = A(i, j)$$

$$(i = 0, \dots, N - 1, j = 0, \dots, M - 1).$$



A



B

```
>> for i = 1 : 512
    for j = 1 : 512
        B(j, i) = A(i, j);
    end
end
```

OR

```
>> B = A';
```



Simple Processing - Flip Vertical

- The vertical flipped image **B** ($N \times M$) of **A** ($N \times M$) can be obtained as $B(i, M - 1 - j) = A(i, j)$ ($i = 0, \dots, N - 1, j = 0, \dots, M - 1$).



A



B

```
>> clear B;  
>> for i = 1 : 512  
    for j = 1 : 512  
        B(i, 512 + 1 - j) = A(i, j);  
    end  
end
```



Simple Processing - Cropping

- The cropped image \mathbf{B} ($N_1 \times N_2$) of \mathbf{A} ($N \times M$), starting from (n_1, n_2) , can be obtained as $B(k, l) = A(n_1 + k, n_2 + l)$ ($k = 0, \dots, N_1 - 1, l = 0, \dots, N_2 - 1$).



A



B

```
>> clear B;  
>> for k = 0 : 64 - 1  
    for l = 0 : 128 - 1  
        B(k + 1, l + 1) = A(255 + k + 1, 255 + l + 1); % n1=n2=255 N1=64,N2=128  
    end  
end
```



Cropping as a Matlab Function

```
function [B] =mycrop(A, n1, n2, N1, N2)
% mycrop.m
% [B] =mycrop(A, n1, n2, N1, N2)
% crops the image A from location n1, n2
% with size N1, N2

for k = 0 : N1 - 1
    for l = 0 : N2 - 1
        B(k + 1, l + 1) = A(n1 + k + 1, n2 + l + 1);
    end
end
```

```
function [B] =mycrop(A, n1, n2, N1, N2)
% mycrop.m
% [B] =mycrop(A, n1, n2, N1, N2)
% crops the image A from location n1, n2
% with size N1, N2
```

```
B(1 : N1, 1 : N2) = A(n1 + 1 : n1 + N1, n2 + 1 : n2 + N2);
```

```
>> help mycrop
```

```
>> B =mycrop(A, 255, 255, 64, 128);
```



Simple Image Statistics - Sample Mean and Sample Variance

- The **sample mean** (m_A) of an image \mathbf{A} ($N \times M$):

$$m_A = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} A(i, j)}{NM} \quad (1)$$

- The **sample variance** (σ_A^2) of \mathbf{A} :

$$\sigma_A^2 = \frac{\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (A(i, j) - m_A)^2}{NM} \quad (2)$$

- The **sample standard deviation**, $\sigma_A = \sqrt{\sigma_A^2}$.



Simple Image Statistics - Histogram

Let S be a set and define $\#S$ to be the cardinality of this set, i.e., $\#S$ is the number of elements of S .

- The **histogram** $h_A(l)$ ($l = 0, \dots, 255$) of the image \mathbf{A} is defined as:

$$h_A(l) = \#\{(i, j) \mid A(i, j) = l, i = 0, \dots, N - 1, j = 0, \dots, M - 1\} \quad (3)$$

- Note that:

$$\sum_{l=0}^{255} h_A(l) = \text{Number of pixels in } \mathbf{A} \quad (4)$$



Calculating the Histogram

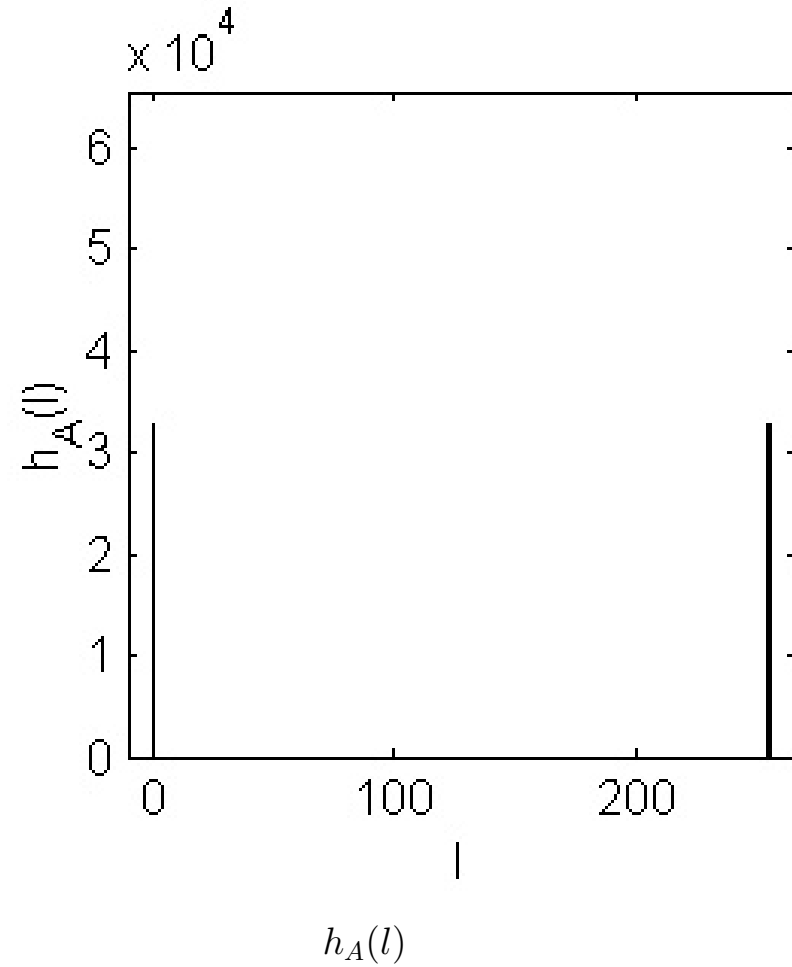
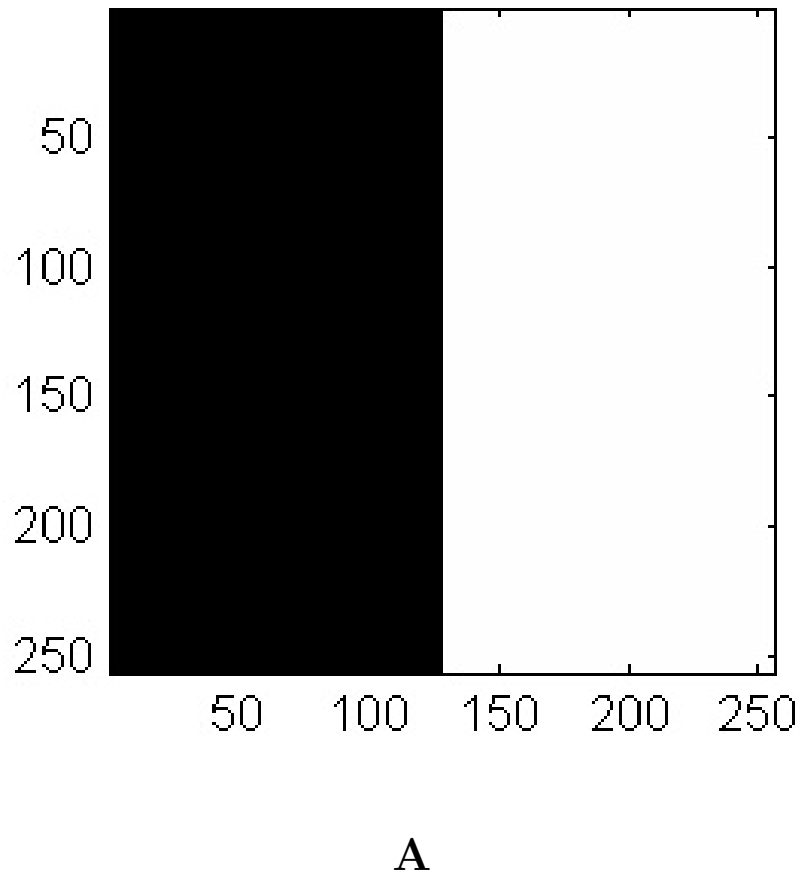
```
>> h=zeros(256,1);  
>> for l = 0 : 255  
    for i = 1 : N  
        for j = 1 : M  
            if (A(i, j) == l)  
                h(l + 1) = h(l + 1) + 1;  
            end  
        end  
    end  
end  
>> bar(0:255,h);
```

```
OR  
>> h=zeros(256,1);  
>> for l = 0 : 255  
    h(l + 1)=sum(sum(A == l));  
end  
>> bar(0:255,h);
```



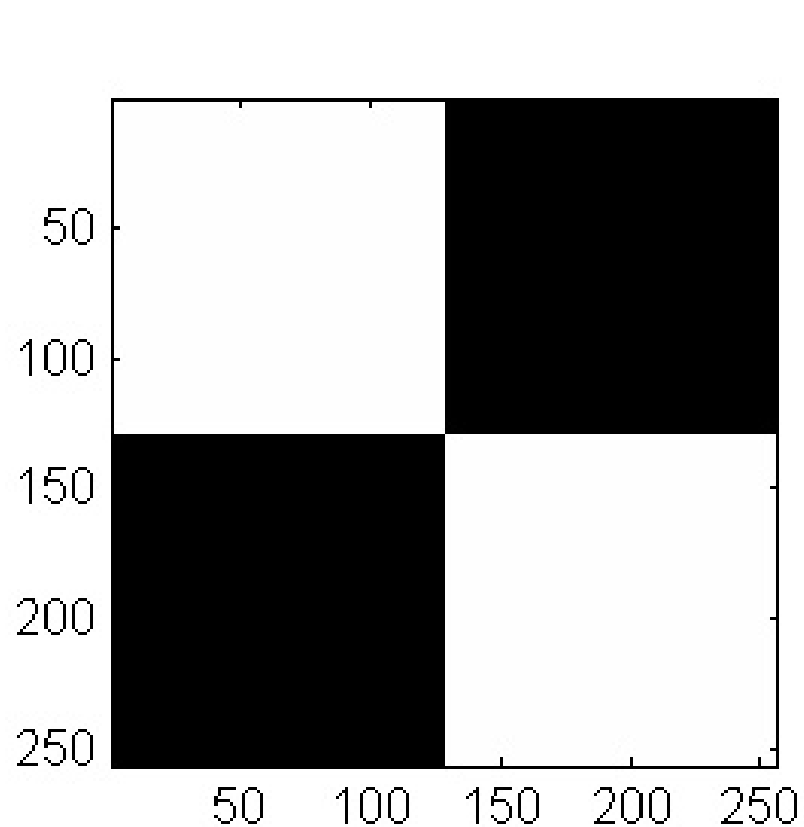


Example I

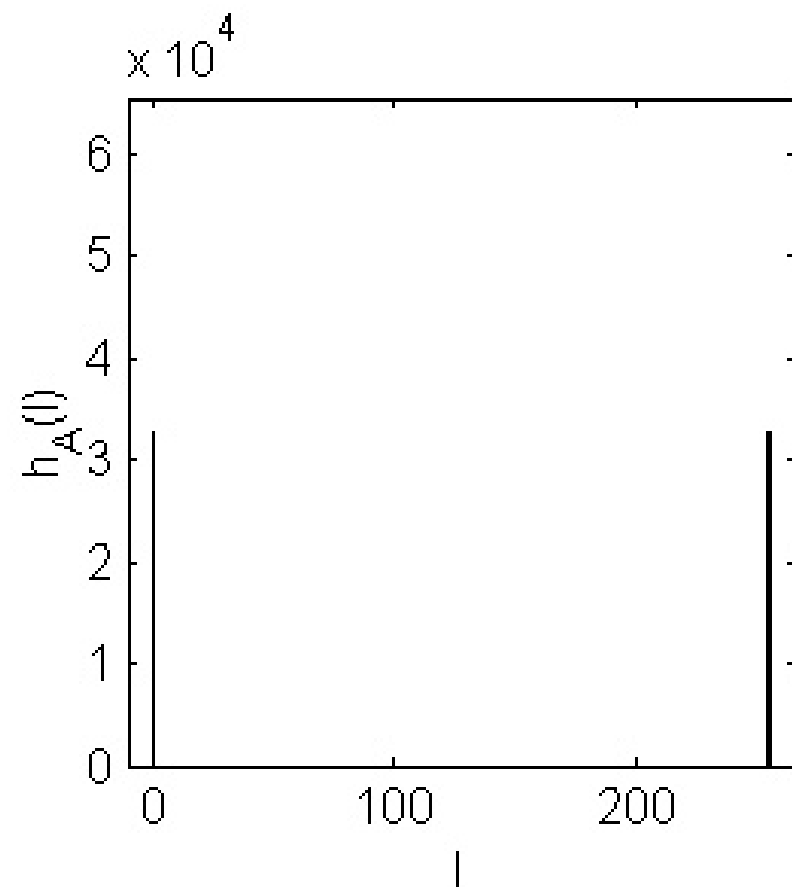




Example II



A



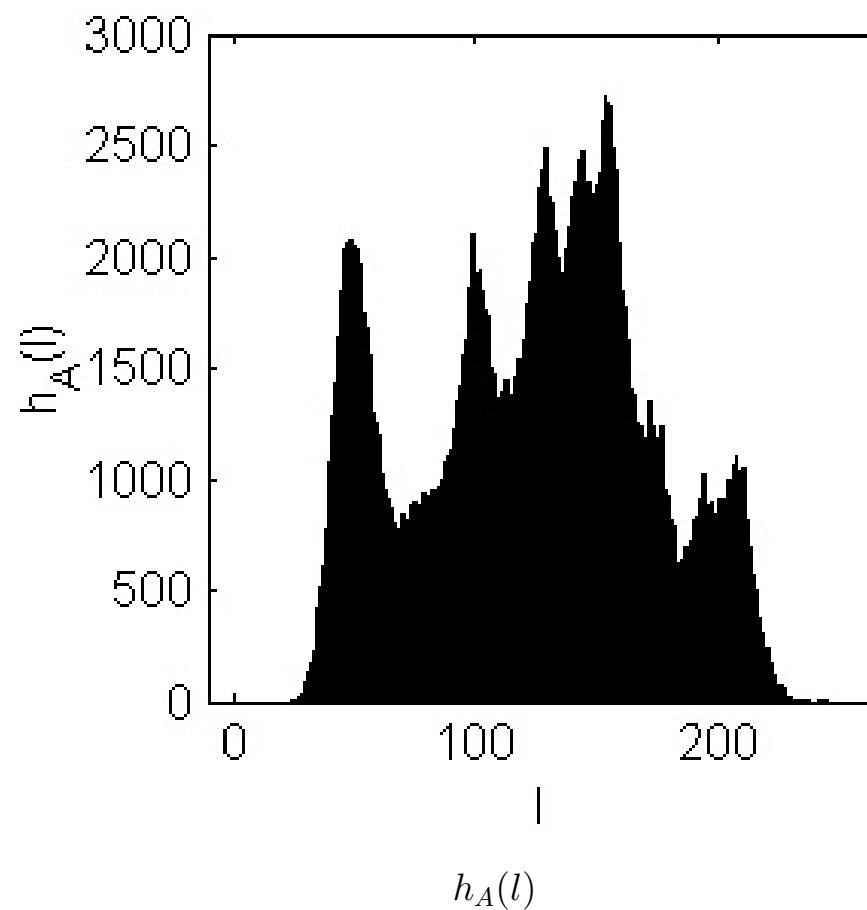
$h_A(l)$



Example III



A





Point Processing

- We will now utilize a “function” $g(l)$ ($l = 0, \dots, 255$) to generate a new image **B** from a given image **A** via:

$$B(i, j) = g(A(i, j)), \quad i = 0, \dots, N - 1, \quad j = 0, \dots, M - 1 \quad (5)$$

- The function $g(l)$ operates on each image pixel or each image **point** independently.
- In general the resulting image $g(A(i, j))$ may not be an image matrix, i.e., it may be the case that $g(A(i, j)) \notin \{0, \dots, 255\}$ for some (i, j) .
- Thus we will have to make sure we obtain an image **B** that is an image matrix.
- The histograms $h_A(l)$ and $h_B(l)$ will play important roles.



Identity point function


- Let $g(l) = l$ ($l = 0, \dots, 255$).

$$\begin{aligned} B(i, j) &= g(A(i, j)), \quad i = 0, \dots, N - 1, \quad j = 0, \dots, M - 1 \\ &= A(i, j) \end{aligned}$$

- In this case $g(A(i, j)) \in \{0, \dots, 255\}$ so no further processing is necessary to ensure \mathbf{B} is an image matrix.
- Note also that $\mathbf{B} = \mathbf{A}$ and hence $h_B(l) = h_A(l)$.



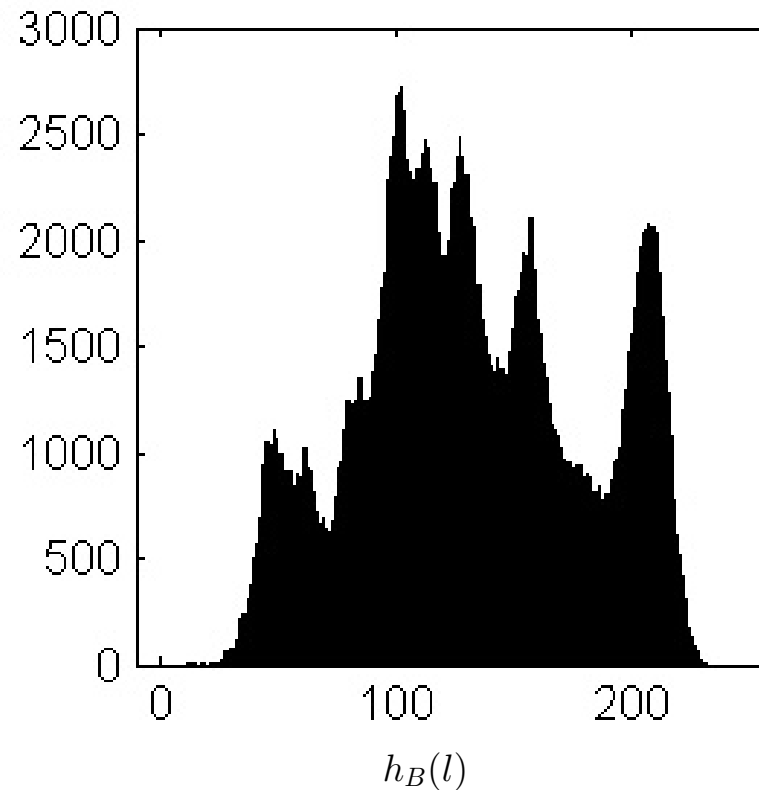
Digital Negative

- Let $g(l) = 255 - l$ ($l = 0, \dots, 255$). 

$$\begin{aligned} B(i, j) &= g(A(i, j)), \quad i = 0, \dots, N - 1, \quad j = 0, \dots, M - 1 \\ &= 255 - A(i, j) \end{aligned}$$

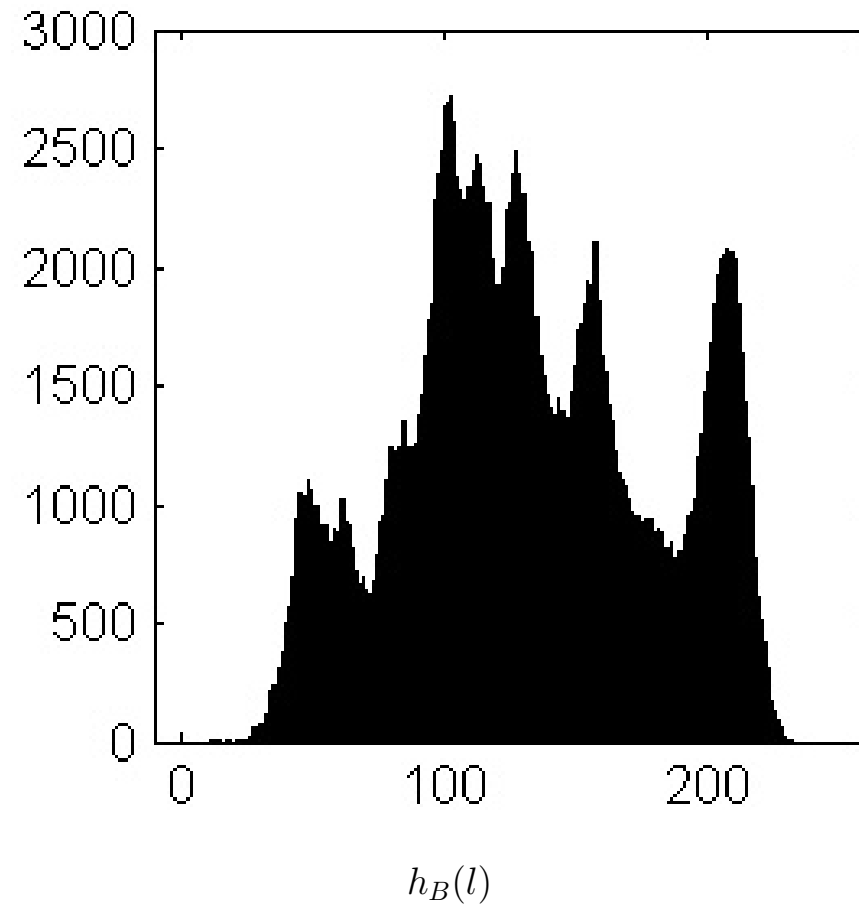
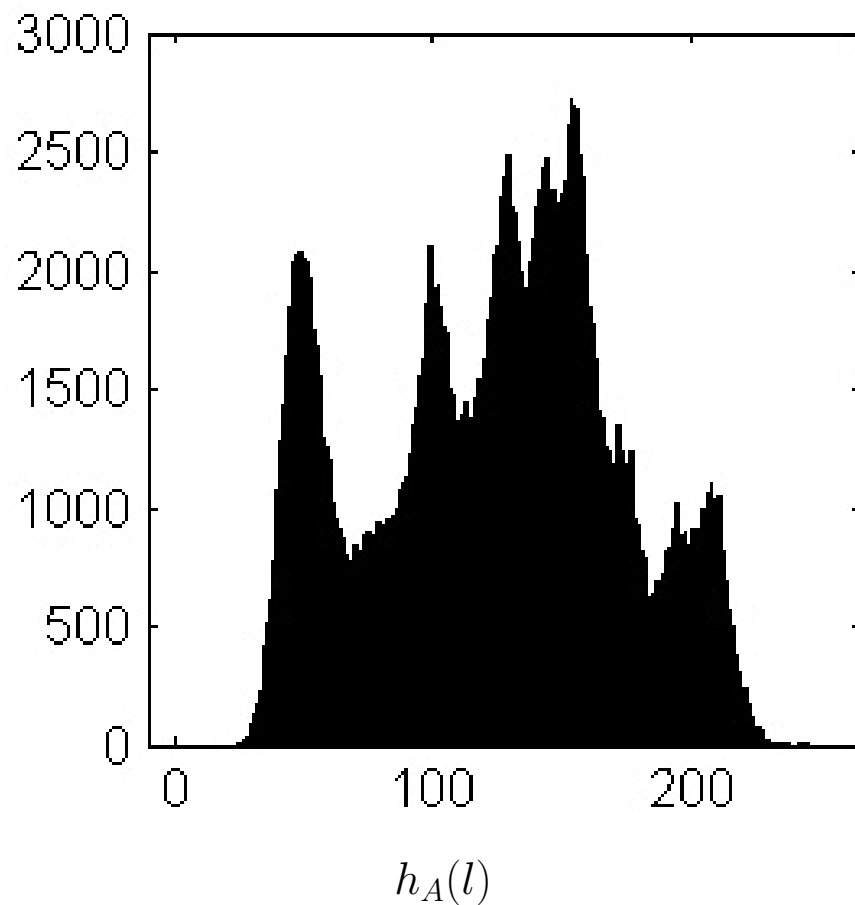


B





Digital Negative Contd.



- In this case it is easy to see that $h_B(255 - l) = h_A(l)$ or $h_B(g(l)) = h_A(l)$.



Calculating the Histogram of Point Processed Images

- For a given point function $g(l)$, “the inverse point function” $g^{-1}(k)$ may not exist.
- **So in general $h_B(g(l)) \neq h_A(l)$.**
- Let $S_{g^{-1}(k)} = \{l \mid g(l) = k, l = 0, \dots, 255\}$ ($k = 0, \dots, 255$).
- Then:

$$h_B(k) = \sum_{l \in S_{g^{-1}(k)}} h_A(l), \quad k = 0, \dots, 255 \quad (6)$$

- You *must* learn how to calculate and sketch $h_B(l)$ given $h_A(l)$ and the point function $g(l)$.



Square-root point function

- Let $g(l) = \sqrt{l}$ ($l = 0, \dots, 255$).

$$\begin{aligned} B(i, j) &= g(A(i, j)), \quad i = 0, \dots, N - 1, \quad j = 0, \dots, M - 1 \\ &= \sqrt{A(i, j)} \end{aligned}$$

- In this case $g(A(i, j)) \notin \{0, \dots, 255\}$ and we must ensure **B** is an image matrix by further processing.
- We can try “rounding” $g(A(i, j))$ to integers by defining a new point function $g_2(l) = \text{round}(g(l))$:

$$\begin{aligned} B(i, j) &= g_2(A(i, j)), \quad i = 0, \dots, N - 1, \quad j = 0, \dots, M - 1 \\ &= \text{round}(\sqrt{A(i, j)}) \quad (>> \quad B = \text{round}(A.^{.5});) \end{aligned}$$

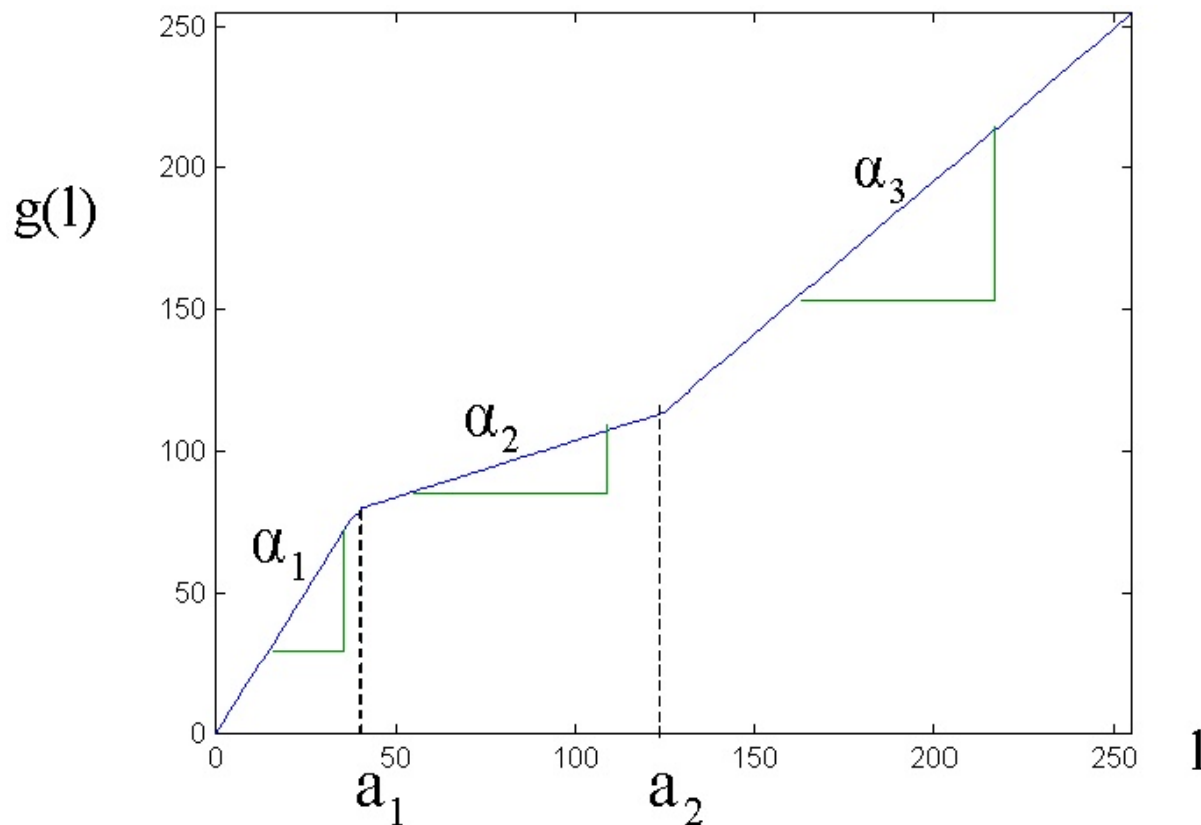
but that will not exactly yield what we want as we shall see.

- We will return to this example after we learn **Contrast Stretching**.



Contrast Stretching

$$g(l) = \begin{cases} \alpha_1 l, & 0 \leq l < a_1 \\ \alpha_2(l - a_1) + \alpha_1 a_1, & a_1 \leq l < a_2 \\ \alpha_3(l - a_2) + (\alpha_2(a_2 - a_1) + \alpha_1 a_1), & a_2 \leq l \leq 255 \end{cases} \quad (7)$$

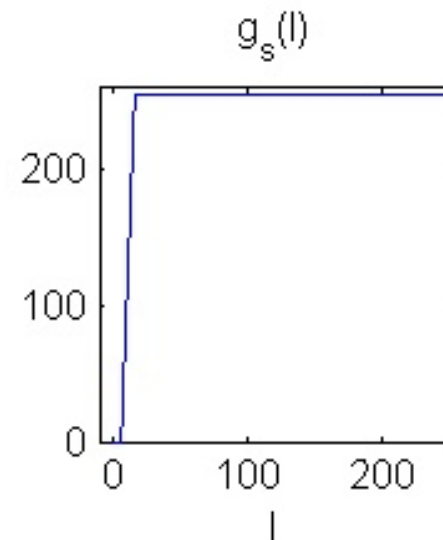
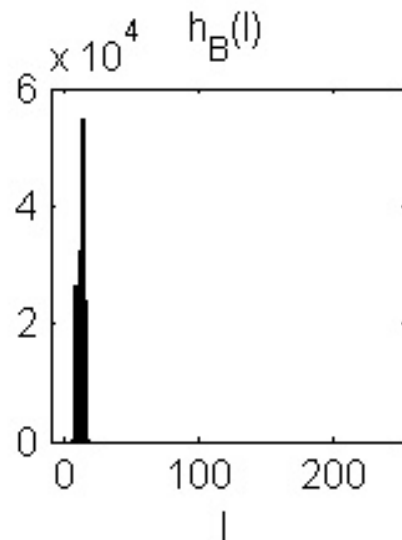
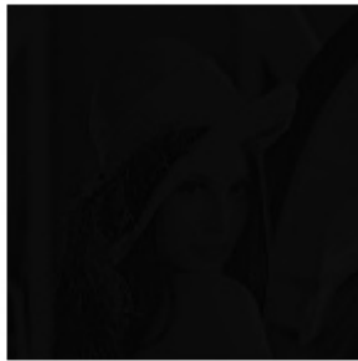


$\alpha_i > 1 \Rightarrow$ Range Stretching
 $\alpha_i < 1 \Rightarrow$ Range Compression

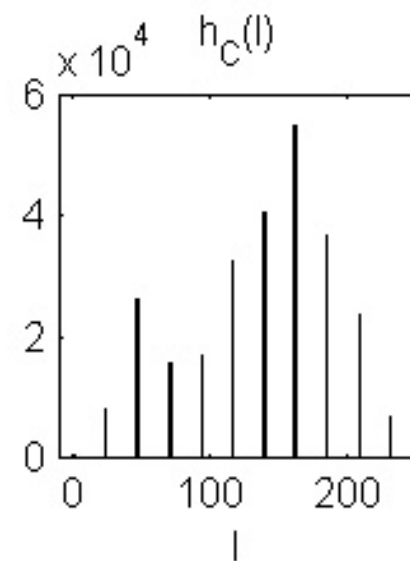


Example - Square-root

$$B(i,j) = \text{round}(A(i,j)^{1/2})$$



$$C(i,j) = g_s(B(i,j))$$





Piecewise Linear, “Continuous” Point Functions

- Let $\alpha_0 = a_0 = 0$.

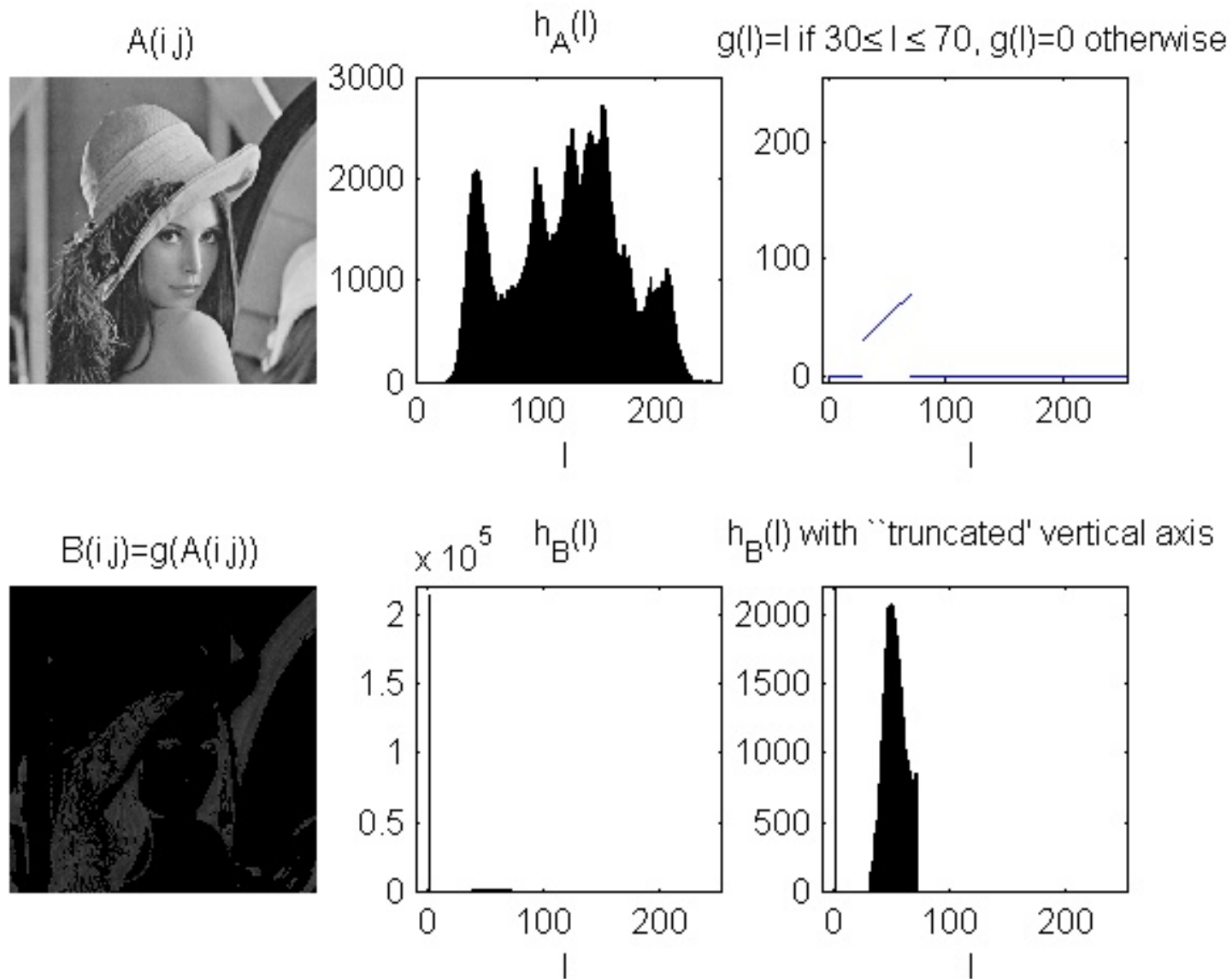
$$g(l) = \begin{cases} \alpha_1 l, & 0 \leq l < a_1 \\ \alpha_2(l - a_1) + \alpha_1 a_1, & a_1 \leq l < a_2 \\ \vdots & \\ \alpha_i(l - a_{i-1}) + (\sum_{j=1}^{i-1} \alpha_j(a_j - a_{j-1})), & a_{i-1} \leq l < a_i \\ \vdots & \end{cases} \quad (8)$$

- $|\alpha_i| < 1 \Rightarrow$ **Range Compression**.
- $|\alpha_i| > 1 \Rightarrow$ **Range Stretching**.
- Assuming $0 \leq g(l) \leq 255$ ($l = 0, \dots, 255$), affect point function by incorporating the $\text{round}(\dots)$ function when necessary, i.e., $g_2(l) = \text{round}(g(l))$:

$$B(i, j) = g_2(A(i, j));$$



A “Discontinuous” Point Function





Normalizing an Image

- Let $mx_A = \max_{(i,j)} A(i, j)$ and $mn_A = \min_{(i,j)} A(i, j)$.

These can be generated in matlab via:

```
>> mx=max(max(A));
```

```
>> mn=min(min(A));
```

- The *special normalizing point function* for **A** is defined as:

$$g_s^A(l) = \text{round}\left(\frac{l - mn_A}{mx_A - mn_A} \times 255\right) \quad (9)$$

- Convention:

A \Rightarrow processing \Rightarrow **B** \Rightarrow “normalize **B**” \Rightarrow $C(i, j) = g_s^B(B(i, j))$. 



More Range Stretching/Compression

- In some cases we wish to view a matrix (image matrix or otherwise) having a wide range of values.

- Let


$$B(i, j) = \begin{cases} A(0, 0) + 10^6 & i = j = 0 \\ A(i, j) & \text{otherwise} \end{cases}$$

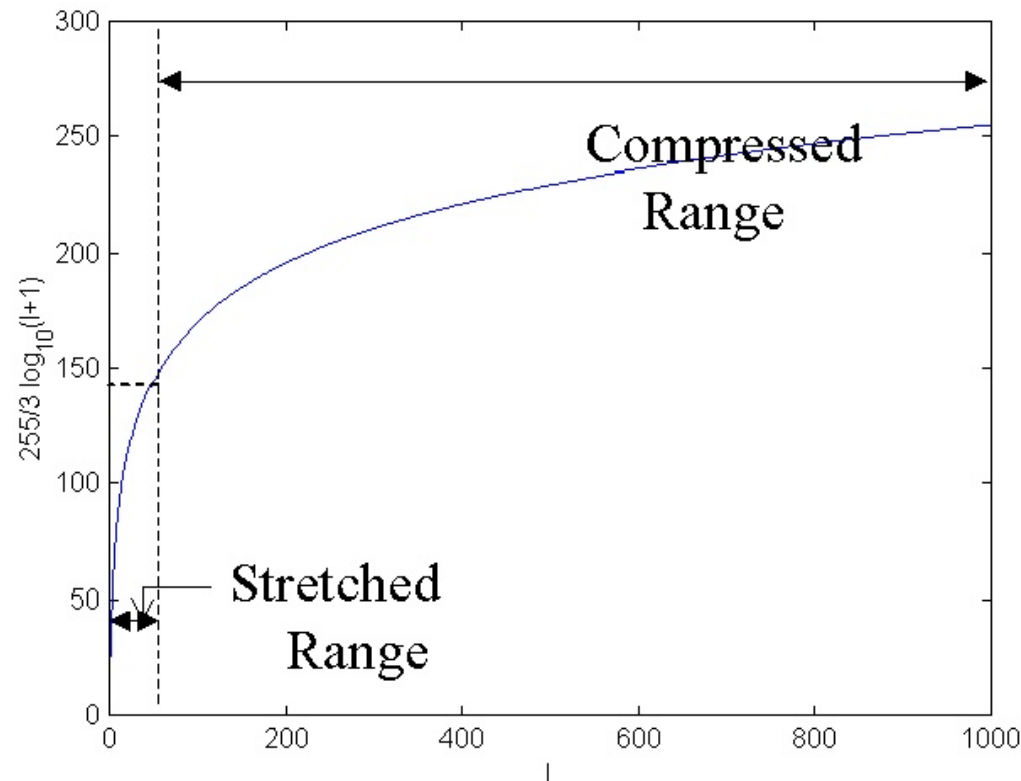
where A is an image matrix ($A(i, j) \in \{0, \dots, 255\}$).

- **Normalizing** B using $C(i, j) = g_s^B(B(i, j))$ will show an image (C) that is completely black except for the pixel $(0, 0)$ which will have the value 255.
- This loses all the information in C about A .
- The next two slides consider an example which addresses the problem by point processing. The resulting point function is sometimes called an emphasis/de-emphasis function.



Example

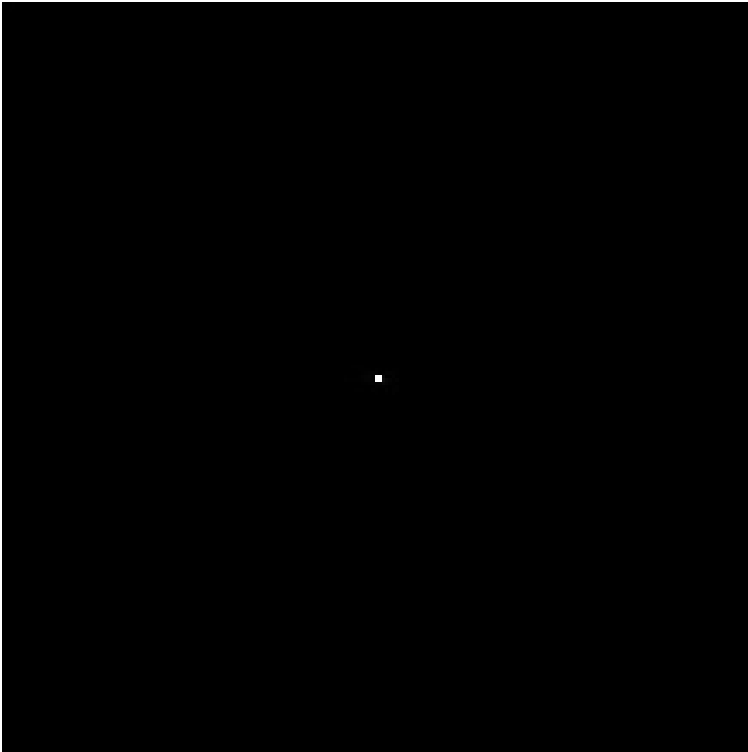
- A good example of the “wide range” problem happens when we want to show the 2-D Discrete Fourier Transform (DFT) of an image. 
- `>> F = round(fftshift(abs(fft2(A))));`
- Let $B(i, j) = \log_{10}(F(i, j) + 1)$ and $C(i, j) = g_s^F(F(i, j))$, $D(i, j) = g_s^B(B(i, j))$.



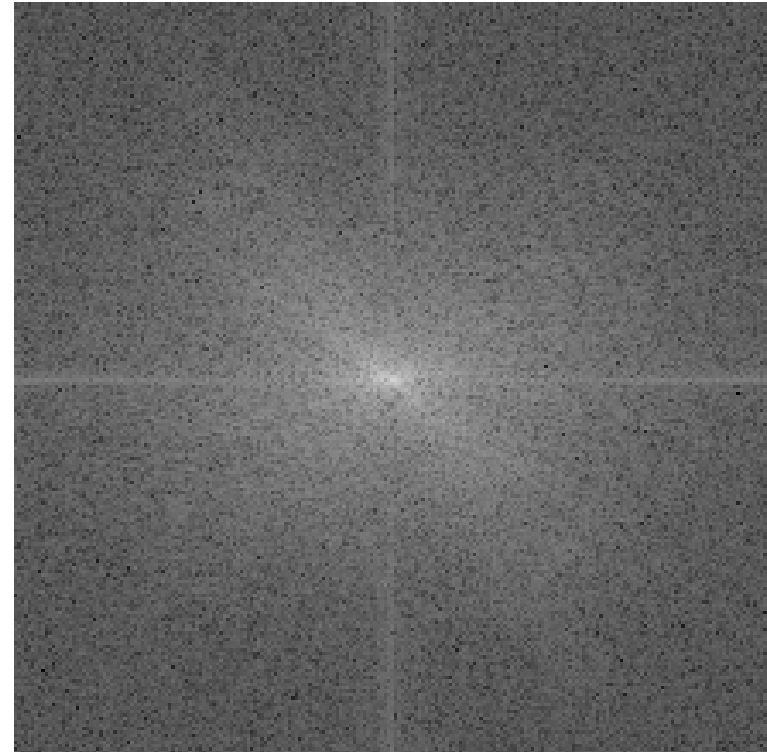


Example - contd.

C



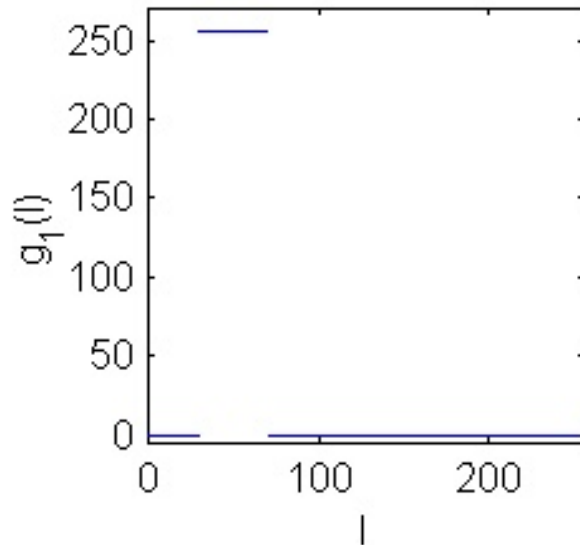
D



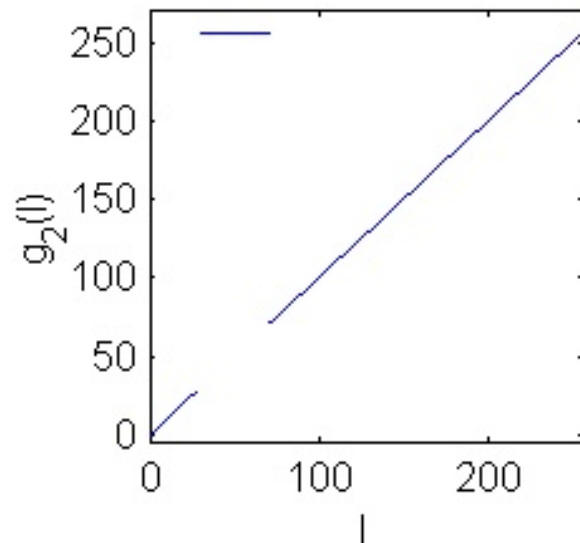
- Specific range stretching/compression might yield even better results. This is an effective tool to visualize things quickly. Different problems can be tackled in the same spirit by utilizing point functions based on $\log_{10}(\dots)$, $\log_{10}(\log_{10}(\dots))$, e^{\dots} and so on.



“Slicing”



without background



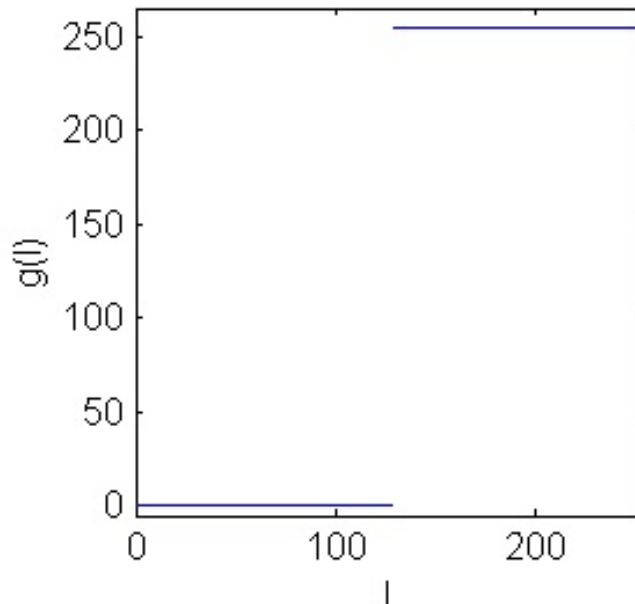
with background



Thresholding

- Binary thresholding ($T = 128$)

$$B(i, j) = \begin{cases} 0 & A(i, j) \leq T \\ 255 & A(i, j) > T \end{cases}$$



- We will see more sophisticated thresholding later. 



Summary

- In this lecture we learnt about some simple structural processing techniques like **transposing**, **flipping** and **cropping**.
- We learnt about simple statistics like **sample mean and sample variance**.
- Most importantly we learnt about **histograms**.
 - Histograms tell us how the values of individual pixels in an image are “distributed”.
 - **An image** may have the same histogram as **another image**.
- We learnt about many **point processing** techniques. These will become very handy as we gain more experience with images. Make sure you read the textbook [1] (Chp. 7, pp. 233-241) for more examples.
- We learnt more about matlab, how to write functions, faster executing scripts, etc.

Homework Rules

- Homeworks will be read as opposed to check-marked.
- Remember that I have been doing this for a while. I usually can tell what kind of processing an image underwent.
- Label all axis for plots and bars. Put a *descriptive* title on plots, bars and images.
- Be neat and always show the original image in a homework for easy comparison to processed images.
- All utilized matlab scripts must be handed in with the homework.
- Try to conserve paper by organizing what you hand in.
- Write general matlab scripts and functions so that you can reuse them in the future. Try to make them “bullet-proof” so that you avoid mistakes.
- Matlab scripts/functions must be commented with descriptive [help information](#).
- When showing images, use the *image*, *colormap(gray(256))*, and *axis('image')* commands. There is a command in matlab (*imagesc*) that does some automatic normalization. **Do not** use the command *imagesc*.
- Remember the best way to view the notes and to work in matlab is after you have set your desktop to 24 or 32 bits color depth. Otherwise you may see artificial artifacts.
- When instructed to give “brief comments”, “brief explanations” or “brief comparisons”, etc. please be brief and to the point.

Homework II

1. Flip your image horizontally and diagonally. Print the results.
2. Crop a portion of your image that mostly shows your head. Print the result and parameters used.
3. Calculate the mean, variance and histogram of your image. Show the histogram as a bar plot next to your image (use the *subplot* command). Indicate the size of your image.
4. Compute the square-root of your image. Round it using the **round** function. Show the result and its histogram. Now normalize the rounded image. Show the result and its histogram. Briefly compare it to your original, unprocessed image.
5. Compute the square-root of your image. Normalize the result and print it together with its histogram. Briefly compare it to the output of (4.) and to your original image. Which pixel value ranges got stretched/compressed?
6. Compute $B(i, j) = \log_{10}(A(i, j) + 1)$ where **A** is your image. **B** is a general matrix and not an image matrix. Generate an image matrix from **B** by normalizing it. Show the normalized image next to the original. Briefly describe the changes. Anything that can be seen better? Which pixel value ranges got stretched/compressed?
7. For an image **A** with histogram $h_A(l)$, sketch the histogram $h_B(l)$ for **(a)** $B(i, j) = g_1(A(i, j))$ and **(b)** $B(i, j) = g_2(A(i, j))$ where $h_A(l)$, g_1 , g_2 are as shown in [Figure 1](#).

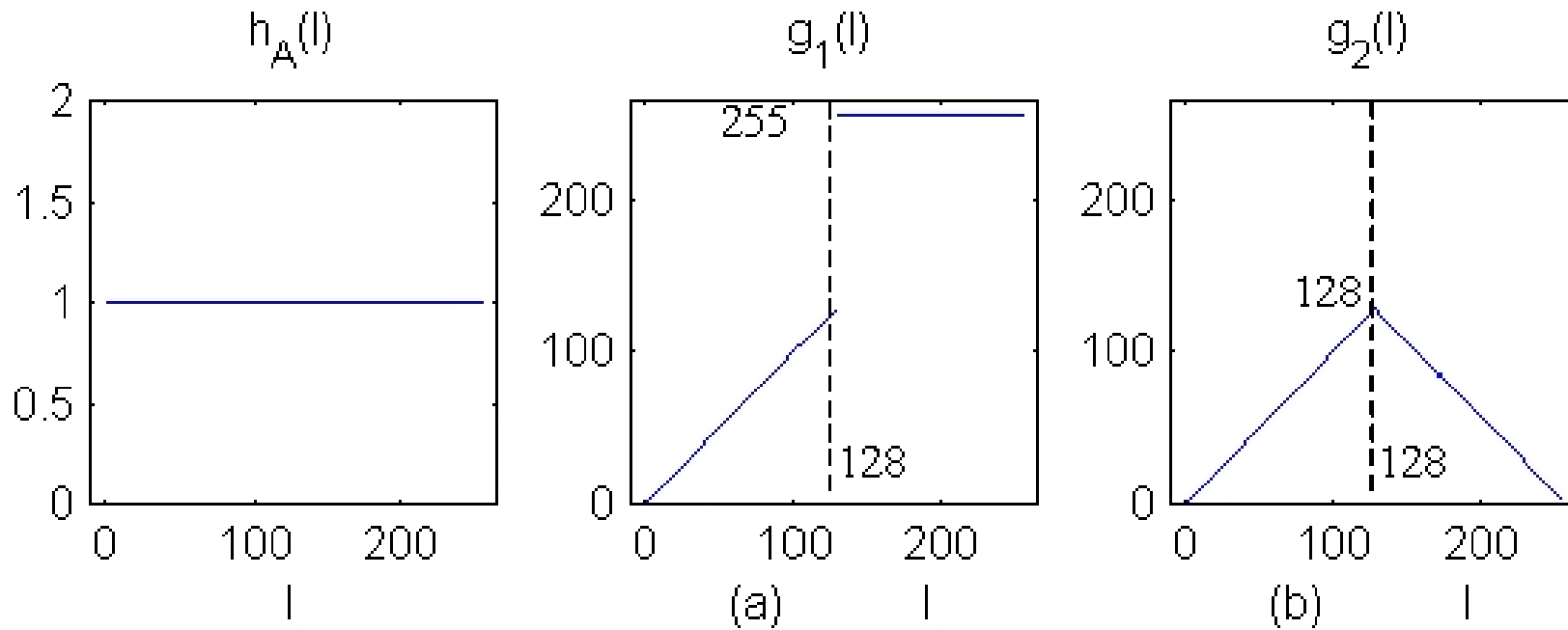


Figure 1: Homework question 7

References

- [1] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.