4 Point Operations

Point operations perform a modification of the pixel values without changing the size, geometry, or local structure of the image. Each new pixel value $a' = I'(u, v)$ depends exclusively on the previous value $a = I(u, v)$ at the *same* position and is thus independent from any other pixel value, in particular from any of its neighboring pixels.[1] The original pixel values are mapped to the new values by a function $f(a)$,

$$\begin{aligned} a' &\leftarrow f(a) \qquad \text{or} \\ I'(u, v) &\leftarrow f\big(I(u, v)\big), \end{aligned} \tag{4.1}$$

for each image position (u, v). If the function $f()$ is independent of the image coordinates (i. e., the same throughout the image), the operation is called "global" or "homogeneous". Typical examples of homogeneous point operations include, among others,

- modifying image brightness or contrast,
- applying arbitrary intensity transformations ("curves"),
- quantizing (or "posterizing") images,
- global thresholding,
- gamma correction,
- color transformations.

[1] If the result depends on more than one pixel value, the operation is called a "filter", as described in Ch. 5.

W. Burger, M.J. Burge, *Principles of Digital Image Processing*, Undergraduate Topics in Computer Science, DOI 10.1007/978-1-84800-191-6_4, © Springer-Verlag London Limited, 2009

We will look at some of these techniques in more detail in the following.

In contrast, the mapping function $g()$ for a *nonhomogeneous* point operation would also take into account the current image coordinate (u, v); i. e.,

$$\begin{aligned} a' &\leftarrow g(a, u, v) \qquad \text{or} \\ I'(u, v) &\leftarrow g\,(I(u, v), u, v)\,. \end{aligned} \tag{4.2}$$

A typical nonhomogeneous operation is the local adjustment of contrast or brightness used for example to compensate for uneven lighting during image acquisition.

4.1 Modifying Image Intensity

4.1.1 Contrast and Brightness

Let us start with a simple example. Increasing the image's contrast by 50% (i. e., by the factor 1.5) or raising the brightness by 10 units can be expressed by the mapping functions

$$f_{\text{contr}}(a) = a \cdot 1.5 \qquad \text{and} \qquad f_{\text{bright}}(a) = a + 10, \tag{4.3}$$

respectively. The first operation is implemented as an ImageJ plugin by the code shown in Prog. 4.1, which can easily be adapted to perform any other type of point operation. Rounding to the nearest integer values is accomplished by simply adding 0.5 before the truncation effected by the `(int)` typecast in line 7 (this only works for positive values). Also note the use of the more efficient image processor methods `get()` and `set()` (instead of `getPixel()` and `putPixel()`) in this example.

4.1.2 Limiting the Results by Clamping

When implementing arithmetic operations on pixels, we must keep in mind that the computed results may exceed the maximum range of pixel values for a given image type ($[0 \ldots 255]$ in the case of 8-bit grayscale images). To avoid this, we have included the "clamping" statement

```
if (a > 255)  a = 255;
```

in line 9 of Prog. 4.1, which limits any result to the maximum value 255. Similarly one should, in general, also limit the results to the minimum value (0) to avoid negative pixel values (which cannot be represented by this type of 8-bit image), for example by the statement

```
if (a < 0)  a = 0;
```

This second measure is not necessary in Prog. 4.1 because the intermediate results can never be negative in this particular operation.

```
1  public void run(ImageProcessor ip) {
2    int w = ip.getWidth();
3    int h = ip.getHeight();
4
5    for (int v = 0; v < h; v++) {
6      for (int u = 0; u < w; u++) {
7        int a = (int) (ip.get(u, v) * 1.5 + 0.5);
8        if (a > 255)
9          a = 255;   // clamp to maximum value
10       ip.set(u, v, a);
11     }
12   }
13 }
```

Program 4.1 Point operation to increase the contrast by 50% (ImageJ plugin). Note that in line 7 the result of the multiplication of the integer pixel value by the constant 1.5 (implicitly of type `double`) is of type `double`. Thus an explicit type cast `(int)` is required to assign the value to the `int` variable `a`. 0.5 is added in line 7 to round to the nearest integer values.

4.1.3 Inverting Images

Inverting an intensity image is a simple point operation that reverses the ordering of pixel values (by multiplying with -1) and adds a constant value to map the result to the admissible range again. Thus, for a pixel value $a = I(u, v)$ in the range $[0, a_{\max}]$, the corresponding point operation is

$$f_{\text{invert}}(a) = -a + a_{\max} = a_{\max} - a. \tag{4.4}$$

The inversion of an 8-bit grayscale image with $a_{\max} = 255$ was the task of our first plugin example in Sec. 2.2.4 (Prog. 2.1). Note that in this case no clamping is required at all because the function always maps to the original range of values. In ImageJ, this operation is performed by the method `invert()` (for objects of type `ImageProcessor`) and is also available through the Edit→Invert menu. Obviously, inverting an image mirrors its histogram, as shown in Fig. 4.5 (c).

4.1.4 Threshold Operation

Thresholding an image is a special type of quantization that separates the pixel values in two classes, depending upon a given threshold value a_{th} that is usually constant. The threshold function $f_{\text{threshold}}(a)$ maps all pixels to one of two fixed intensity values a_0 or a_1; i. e.,

$$f_{\text{threshold}}(a) = \begin{cases} a_0 & \text{for } a < a_{\text{th}} \\ a_1 & \text{for } a \geq a_{\text{th}} \end{cases} \tag{4.5}$$

with $0 < a_{\text{th}} \leq a_{\max}$. A common application is *binarizing* an intensity image with the values $a_0 = 0$ and $a_1 = 1$.

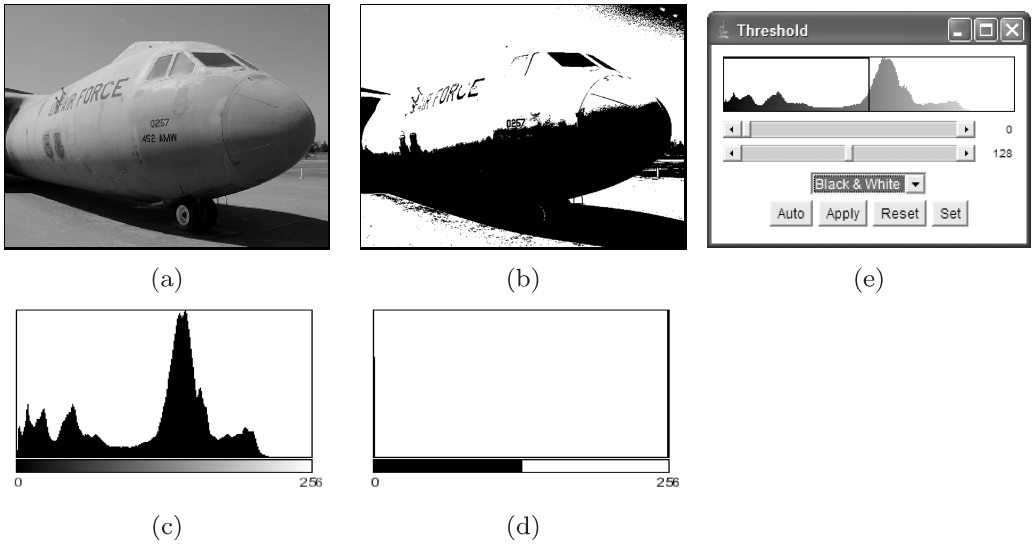


Figure 4.1 Threshold operation: original image (a) and corresponding histogram (c); result after thresholding with $a_{\text{th}} = 128$, $a_0 = 0$, $a_1 = 255$ (b) and corresponding histogram (d); ImageJ's interactive Threshold menu (e).

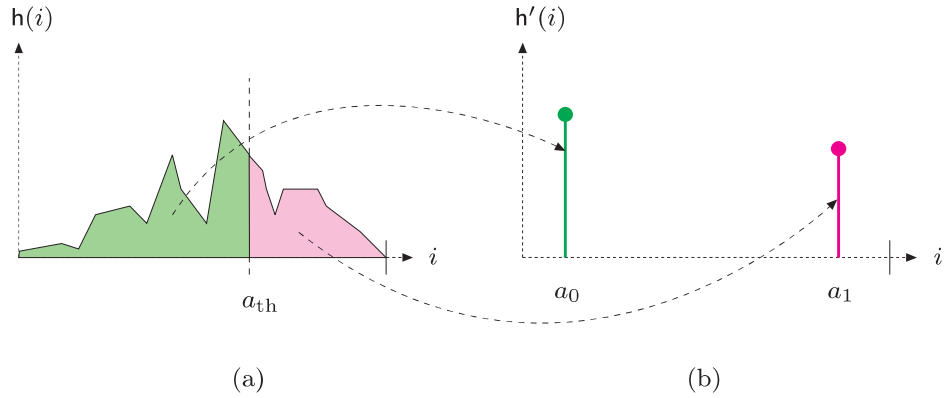


Figure 4.2 Effects of thresholding upon the histogram. The threshold value is a_{th}. The original distribution (a) is split and merged into two isolated entries at a_0 and a_1 in the resulting histogram (b).

ImageJ does provide a special image type (`BinaryProcessor`) for binary images, but these are actually implemented as 8-bit intensity images (just like ordinary intensity images) using the values 0 and 255. ImageJ also provides the `ImageProcessor` method `threshold(int level)`, with `level` $\equiv a_{\text{th}}$, to perform this operation, which can also be invoked through the Image→Adjust→Threshold menu (see Fig. 4.1 for an example). Thresholding affects the histogram by splitting and merging the distribution into two entries at positions a_0 and a_1, as illustrated in Fig. 4.2.

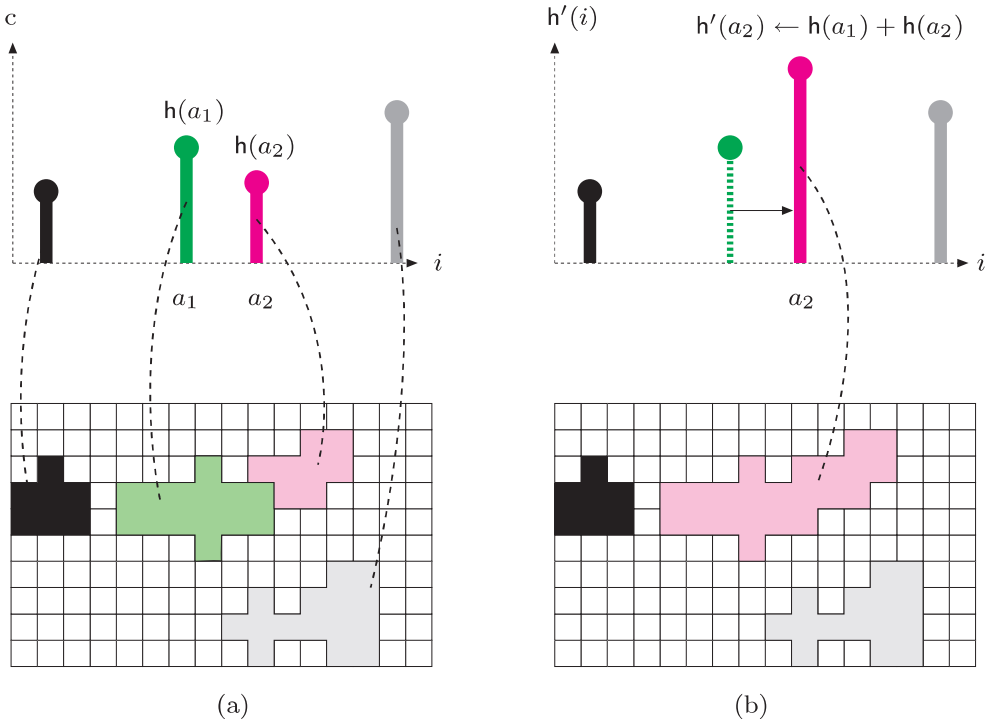


Figure 4.3 Histogram entries map to *sets* of pixels of the same value. If a histogram line is moved as a result of some point operations, then all pixels in the corresponding set are equally modified (a). If, due to this operation, two histogram lines $\mathsf{h}(a_1)$, $\mathsf{h}(a_2)$ coincide on the same index, the two corresponding pixel sets join and the contained pixels become undiscernable (b).

4.2 Point Operations and Histograms

We have already seen that the effects of a point operation on the image's histogram are quite easy to predict in some cases. For example, increasing the brightness of an image by a constant value shifts the entire histogram to the right, raising the contrast widens it, and inverting the image flips the histogram. Although this appears rather simple, it may be useful to look a bit more closely at the relationship between point operations and the resulting changes in the histogram.

As the illustration in Fig. 4.3 shows, every entry (bar) at some position i in the histogram maps to a *set* (of size $\mathsf{h}(i)$) containing all image pixels whose values are exactly i.[2] If a particular histogram line is *shifted* as a result of some point operation, then of course all pixels in the corresponding set are equally modified and vice versa. So what happens when a point operation (e. g.,

[2] Of course this is only true for ordinary histograms with an entry for every single intensity value. If *binning* is used (see Sec. 3.4.1), each histogram entry maps to pixels within a certain *range* of values.

reducing image contrast) causes two previously separated histogram lines to fall together at the same position i? The answer is that the corresponding pixel sets are *merged* and the new common histogram entry is the sum of the two (or more) contributing entries (i. e., the size of the combined set). At this point, the elements in the merged set are no longer distinguishable (or separable), so this operation may have (perhaps unintentionally) caused an irreversible reduction of dynamic range and thus a permanent loss of information in that image.

4.3 Automatic Contrast Adjustment

Automatic contrast adjustment ("auto-contrast") is a point operation whose task is to modify the pixels such that the available range of values is fully covered. This is done by mapping the current darkest and brightest pixels to the lowest and highest available intensity values, respectively, and linearly distributing the intermediate values.

Let us assume that a_{low} and a_{high} are the lowest and highest pixel values found in the current image, whose full intensity range is $[a_{\min}, a_{\max}]$. To stretch the image to the full intensity range (see Fig. 4.4), we first map the smallest pixel value a_{low} to zero, subsequently increase the contrast by the factor $(a_{\max} - a_{\min})/(a_{\text{high}} - a_{\text{low}})$, and finally shift to the target range by adding $a_{\min}$. The mapping function for the auto-contrast operation is thus defined as

$$f_{\text{ac}}(a) = a_{\min} + \big(a - a_{\text{low}}\big) \cdot \frac{a_{\max} - a_{\min}}{a_{\text{high}} - a_{\text{low}}}, \tag{4.6}$$

provided that $a_{\text{high}} \neq a_{\text{low}}$; i. e., the image contains at least *two* different pixel values. For an 8-bit image with $a_{\min} = 0$ and $a_{\max} = 255$, the function in Eqn. (4.6) simplifies to

$$f_{\text{ac}}(a) = (a - a_{\text{low}}) \cdot \frac{255}{a_{\text{high}} - a_{\text{low}}}. \tag{4.7}$$

The target range $[a_{\min}, a_{\max}]$ need not be the maximum available range of values but can be any interval to which the image should be mapped. Of course the method can also be used to reduce the image contrast to a smaller range. Figure 4.5 (b) shows the effects of an auto-contrast operation on the corresponding histogram, where the linear stretching of the intensity range results in regularly spaced gaps in the new distribution.

4.4 Modified Auto-Contrast

In practice, the mapping function in Eqn. (4.6) could be strongly influenced by only a few extreme (low or high) pixel values, which may not be representative of the main image content. This can be avoided to a large extent by "saturating"

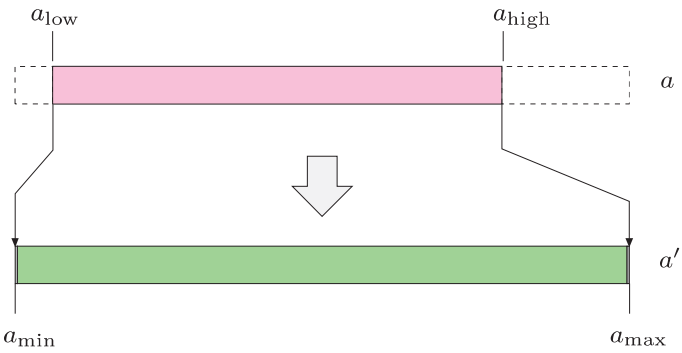
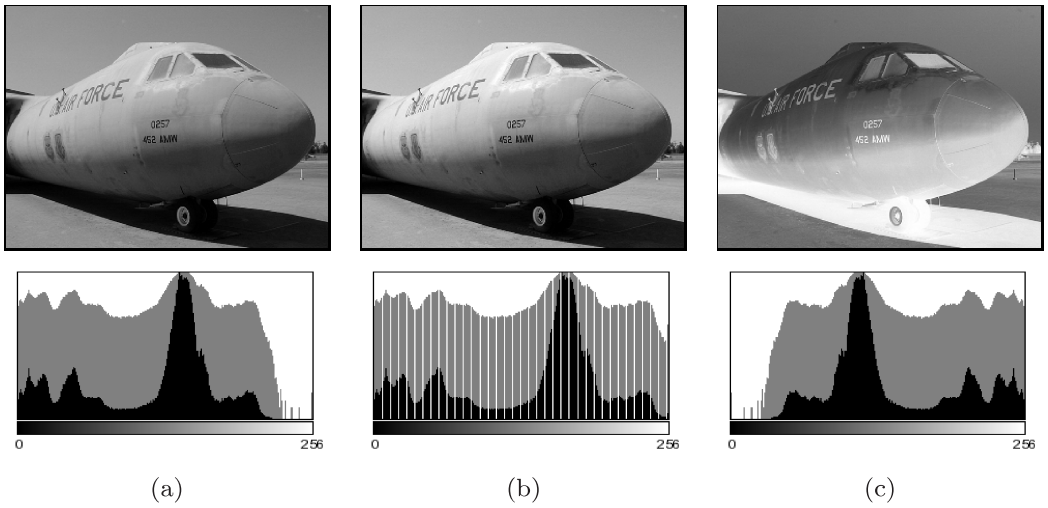


Figure 4.4 Auto-contrast operation according to Eqn. (4.6). Original pixel values a in the range $[a_{\text{low}}, a_{\text{high}}]$ are mapped linearly to the target range $[a_{\text{min}}, a_{\text{max}}]$.



(a) (b) (c)

Figure 4.5 Effects of auto-contrast and inversion operations on the resulting histograms. Original image (a), result of auto-contrast operation (b), and inversion (c). The histogram entries are shown both linearly (black bars) and logarithmically (gray bars).

a fixed percentage (s_{low}, s_{high}) of pixels at the upper and lower ends of the target intensity range. To accomplish this, we determine two limiting values a'_{low}, a'_{high} such that a predefined quantile q_{low} of all pixel values in the image I are smaller than a'_{low} and another quantile q_{high} of the values are greater than a'_{high} (Fig. 4.6). The values a'_{low}, a'_{high} depend on the image content and can be easily obtained from the image's cumulative histogram[3] $\mathsf{H}(i)$:

$$a'_{\text{low}} = \min\{\, i \mid \mathsf{H}(i) \geq M \cdot N \cdot q_{\text{low}} \,\}, \tag{4.8}$$

$$a'_{\text{high}} = \max\{\, i \mid \mathsf{H}(i) \leq M \cdot N \cdot (1 - q_{\text{high}}) \,\}, \tag{4.9}$$

where $0 \leq q_{\text{low}}, q_{\text{high}} \leq 1$, $q_{\text{low}} + q_{\text{high}} \leq 1$, and $M \cdot N$ is the number of pixels in the image. All pixel values *outside* (and including) a'_{low} and a'_{high} are mapped

[3] See Sec. 3.6.

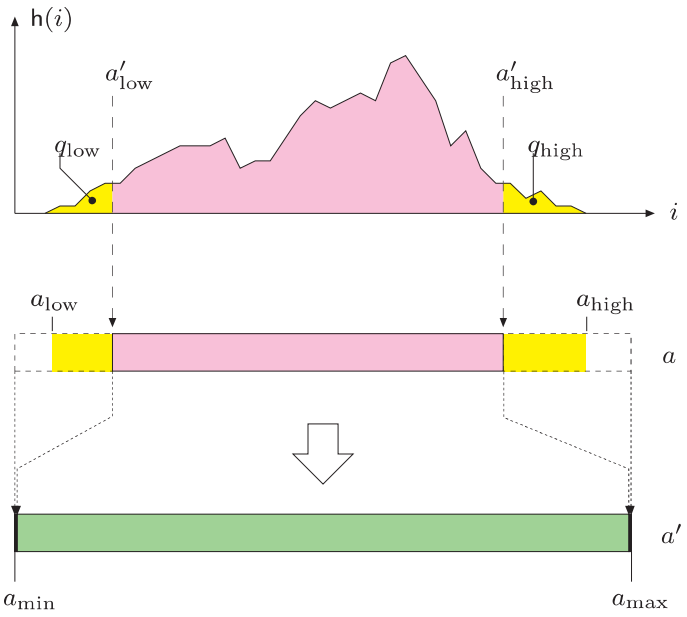


Figure 4.6 Modified auto-contrast operation (Eqn. (4.10)). Predefined quantiles (q_{low}, q_{high}) of image pixels—shown as dark areas at the left and right ends of the histogram $\mathsf{h}(i)$—are "saturated" (i. e., mapped to the extreme values of the target range). The intermediate values ($a = a'_{\text{low}} \ldots a'_{\text{high}}$) are mapped linearly to the interval $[a_{\text{min}}, a_{\text{max}}]$.

to the extreme values a_{min} and a_{max}, respectively, and intermediate values are mapped linearly to the interval $[a_{\text{min}}, a_{\text{max}}]$. The mapping function $f_{\text{mac}}()$ for the modified auto-contrast operation can thus be defined as

$$f_{\text{mac}}(a) = \begin{cases} a_{\text{min}} & \text{for } a \le a'_{\text{low}} \\ a_{\text{min}} + \big(a - a'_{\text{low}}\big) \cdot \dfrac{a_{\text{max}} - a_{\text{min}}}{a'_{\text{high}} - a'_{\text{low}}} & \text{for } a'_{\text{low}} < a < a'_{\text{high}} \\ a_{\text{max}} & \text{for } a \ge a'_{\text{high}}. \end{cases} \tag{4.10}$$

Using this formulation, the mapping to minimum and maximum intensities does not depend on singular extreme pixels only but can be based on a representative set of pixels. Usually the same value is taken for both upper and lower quantiles (i. e., $q_{\text{low}} = q_{\text{high}} = q$), with $q = 0.005 \ldots 0.015$ ($0.5 \ldots 1.5\,\%$) being common values. For example, the auto-contrast operation in Adobe Photoshop saturates 0.5 % ($q = 0.005$) of all pixels at both ends of the intensity range. Auto-contrast is a frequently used point operation and thus available in practically any image-processing software. ImageJ implements the modified auto-contrast operation as part of the Brightness/Contrast and Image→Adjust menus (Auto button), shown in Fig. 4.7.

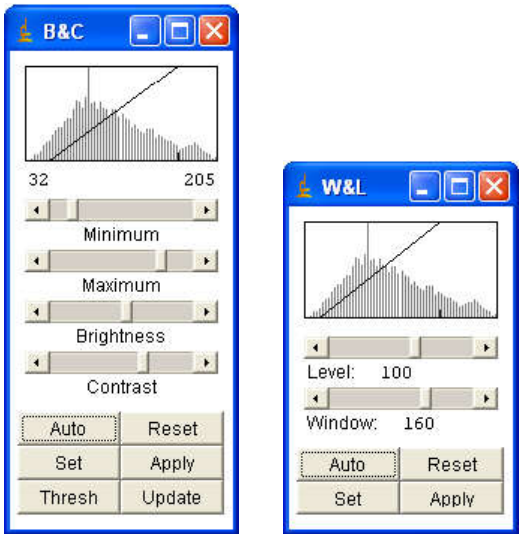


Figure 4.7 ImageJ's Brightness/Contrast tool (left) and Window/Level tool (right) can be invoked through the Image→Adjust menu. The Auto button displays the result of a modified auto-contrast operation. Apply must be hit to actually modify the image.

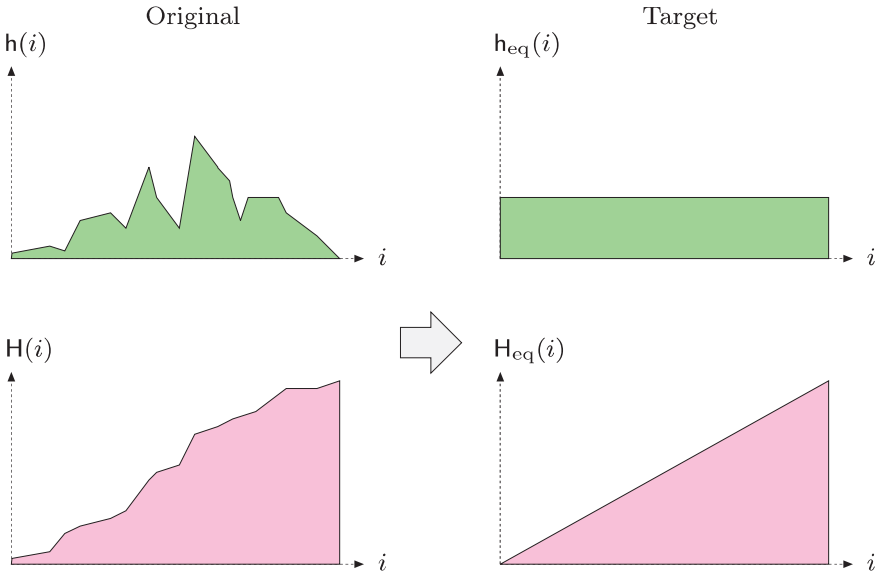


Figure 4.8 Histogram equalization. The idea is to find and apply a point operation to the image (with original histogram h) such that the histogram h_{eq} of the modified image approximates a *uniform* distribution (top). The cumulative target histogram H_{eq} must thus be approximately wedge-shaped (bottom).

4.5 Histogram Equalization

A frequent task is to adjust two different images in such a way that their resulting intensity distributions are similar, for example to use them in a print publication or to make them easier to compare. The goal of histogram equalization is to find and apply a point operation such that the histogram of the modified image approximates a *uniform* distribution (see Fig. 4.8). Since the

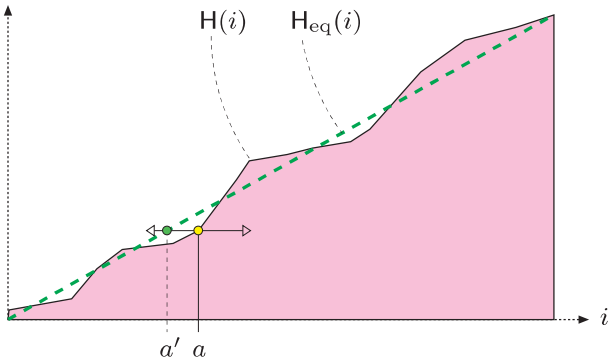


Figure 4.9 Histogram equalization on the cumulative histogram. A suitable point operation $a' \leftarrow f_{\text{eq}}(a)$ shifts each histogram line from its original position a to a' (left or right) such that the resulting cumulative histogram H_{eq} is approximately linear.

histogram is a discrete distribution and homogeneous point operations can only shift and merge (but never split) histogram entries, we can only obtain an approximate solution in general. In particular, there is no way to eliminate or decrease individual peaks in a histogram, and a truly uniform distribution is thus impossible to reach. Based on point operations, we can thus modify the image only to the extent that the resulting histogram is *approximately* uniform. The question is how good this approximation can be and exactly which point operation (which clearly depends on the image content) we must apply to achieve this goal.

We may get a first idea by observing that the *cumulative* histogram (Sec. 3.6) of a uniformly distributed image is a linear ramp (wedge), as shown in Fig. 4.8. So we can reformulate the goal as finding a point operation that shifts the histogram lines such that the resulting cumulative histogram is approximately linear, as illustrated in Fig. 4.9.

The desired point operation $f_{\text{eq}}()$ is simply obtained from the cumulative histogram H of the original image as[4]

$$f_{\text{eq}}(a) = \left\lfloor \mathsf{H}(a) \cdot \frac{K-1}{MN} \right\rfloor, \tag{4.11}$$

for an image of size $M \times N$ with pixel values a in the range $[0, K-1]$. The resulting function $f_{\text{eq}}(a)$ in Eqn. (4.11) is monotonically increasing, because $\mathsf{H}(a)$ is monotonic and K, M, N are all positive constants. In the (unusual) case where an image is already uniformly distributed, linear histogram equalization should not modify that image any further. Also, repeated applications of linear histogram equalization should not make any changes to the image after the first time. Both requirements are fulfilled by the formulation in Eqn. (4.11).

[4] For a derivation, see, e. g., [17, p. 173].

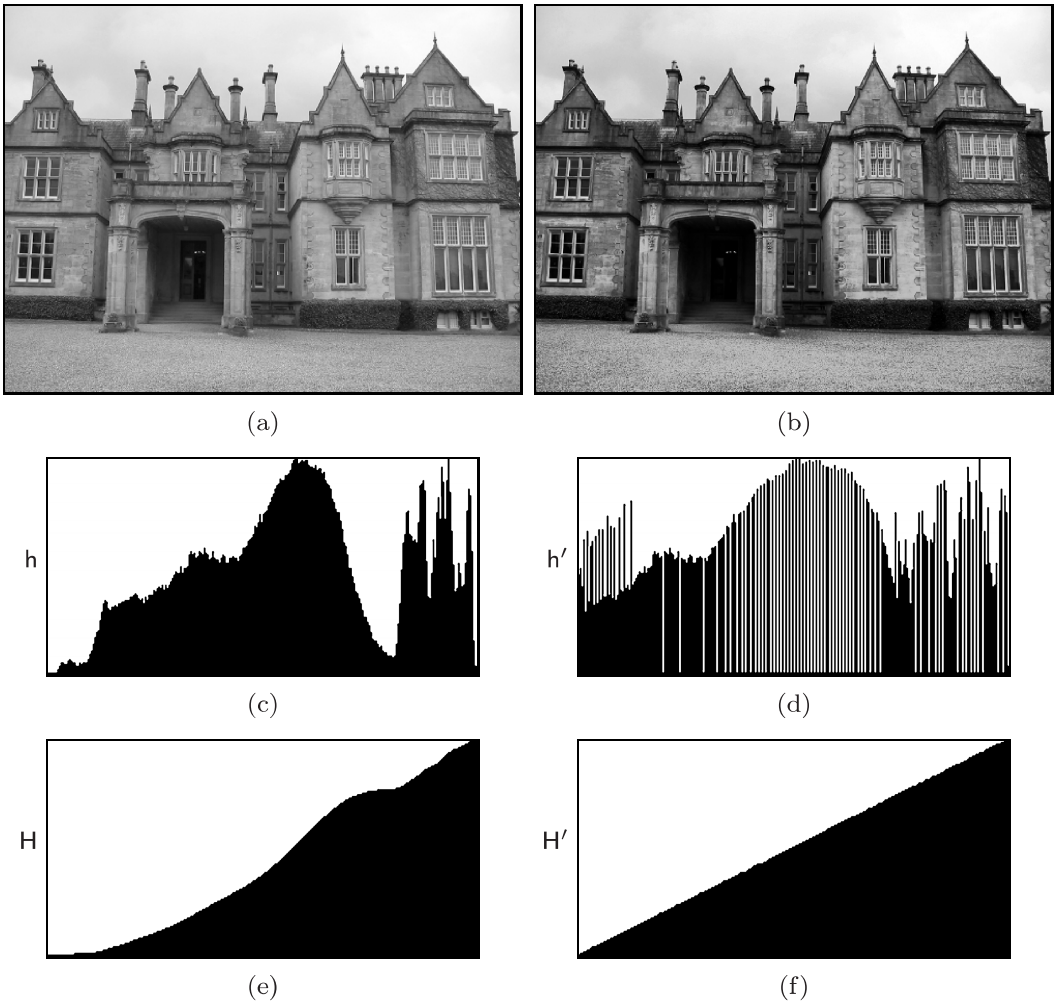


Figure 4.10 Linear histogram equalization (example). Original image I (a) and modified image I' (b), corresponding histograms h, h' (c, d), and cumulative histograms H, H' (e, f). The resulting cumulative histogram H' (f) approximates a uniformly distributed image. Notice that new peaks are created in the resulting histogram h' (d) by merging original histogram cells, particularly in the lower and upper intensity ranges.

Program 4.2 lists the Java code for a sample implementation of linear histogram equalization. An example demonstrating the effects on the image and the histograms is shown in Fig. 4.10.

Notice that for "inactive" pixel values i (i. e., pixel values that do not appear in the image, with $\mathsf{h}(i) = 0$), the corresponding entries in the cumulative histogram $\mathsf{H}(i)$ are either zero or identical to the neighboring entry $\mathsf{H}(i-1)$. Consequently a contiguous range of zero values in the histogram $\mathsf{h}(i)$ corresponds to a constant (i. e., flat) range in the cumulative histogram $\mathsf{H}(i)$, and the function $f_{\text{eq}}(a)$ maps all "inactive" intensity values within such a range to the next lower "active" value. This effect is not relevant, however, since the image contains no such pixels anyway. Nevertheless, a linear histogram equalization

```
1  public void run(ImageProcessor ip) {
2      int w = ip.getWidth();
3      int h = ip.getHeight();
4      int M = w * h;   // total number of image pixels
5      int K = 256;     // number of intensity values
6
7      // compute the cumulative histogram:
8      int[] H = ip.getHistogram();
9      for (int j = 1; j < H.length; j++) {
10         H[j] = H[j-1] + H[j];
11     }
12
13     // equalize the image:
14     for (int v = 0; v < h; v++) {
15         for (int u = 0; u < w; u++) {
16             int a = ip.get(u, v);
17             int b = H[a] * (K-1) / M;
18             ip.set(u, v, b);
19         }
20     }
21 }
```

Program 4.2 Linear histogram equalization (ImageJ plugin). First the histogram of the image `ip` is obtained using the standard ImageJ method `ip.getHistogram()` in line 8. In line 10, the cumulative histogram is computed "in place" based on the recursive definition in Eqn. (3.6). The `int` division in line 17 implicitly performs the required floor ($\lfloor\ \rfloor$) operation by truncation.

may (and typically will) cause histogram lines to merge and consequently lead to a loss of dynamic range (see also Sec. 4.2).

This or a similar form of linear histogram equalization is implemented in almost any image-processing software. In ImageJ it can be invoked interactively through the Process→Enhance Contrast menu (option Equalize). To avoid extreme contrast effects, the histogram equalization in ImageJ by default[5] cumulates the *square root* of the histogram entries using a modified cumulative histogram of the form

$$\tilde{\mathsf{H}}(i) = \sum_{j=0}^{i} \sqrt{\mathsf{h}(j)}\,. \tag{4.12}$$

4.6 Histogram Specification

Although widely implemented, the goal of linear histogram equalization—a uniform distribution of intensity values (as described in the previous section)—appears rather ad hoc, since good images virtually never show such a distri-

[5] The "classic" (linear) approach, as given by Eqn. (3.5), is used when simultaneously keeping the Alt key pressed.

bution. In most real images, the distribution of the pixel values is not even remotely uniform but is usually more similar, if at all, to perhaps a Gaussian distribution. The images produced by linear equalization thus usually appear quite unnatural, which renders the technique practically useless.

Histogram specification is a more general technique that modifies the image to match an arbitrary intensity distribution, including the histogram of a given image. This is particularly useful, for example, for adjusting a set of images taken by different cameras or under varying exposure or lighting conditions to give a similar impression in print production or when displayed. Similar to histogram equalization, this process relies on the alignment of the cumulative histograms by applying a homogeneous point operation. To be independent of the image size (i. e., the number of pixels), we first define *normalized* distributions, which we use in place of the original histograms.

4.6.1 Frequencies and Probabilities

The value in each histogram cell describes the observed frequency of the corresponding intensity value, i. e., the histogram is a discrete *frequency distribution*. For a given image I of size $M \times N$, the sum of all histogram entries $\mathsf{h}(i)$ equals the number of image pixels,

$$\sum_{i=0}^{K-1} \mathsf{h}(i) = M \cdot N. \tag{4.13}$$

The associated *normalized* histogram

$$\mathsf{p}(i) = \frac{\mathsf{h}(i)}{MN}, \tag{4.14}$$

for $0 \le i < K$, is usually interpreted as the *probability distribution* or *probability density function* (pdf) of a random process, where $\mathsf{p}(i)$ is the probability for the occurrence of the pixel value i. The cumulative probability of i being any possible value is 1, and the distribution p must thus satisfy

$$\sum_{i=0}^{K-1} \mathsf{p}(i) = 1. \tag{4.15}$$

The statistical counterpart to the cumulative histogram H (Eqn. (3.5)) is the discrete *distribution function* $\mathsf{P}()$ (also called the *cumulative distribution function* or cdf), with

$$\begin{aligned} \mathsf{P}(i) &= \frac{\mathsf{H}(i)}{\mathsf{H}(K-1)} = \frac{\mathsf{H}(i)}{MN} = \sum_{j=0}^{i} \frac{\mathsf{h}(j)}{MN} \\ &= \sum_{j=0}^{i} \mathsf{p}(j), \qquad \text{for } 0 \le i < K. \end{aligned} \tag{4.16}$$

Algorithm 4.1 Computation of the cumulative distribution function (cdf) $\mathsf{P}()$ from a given histogram h of length K. See Prog. 4.3 (p. 75) for the corresponding Java implementation.

1: CDF(h)
   Returns the cumulative distribution function $\mathsf{P}(i) \in [0, 1]$ for a given histogram $\mathsf{h}(i)$, with $i = 0, \ldots K-1$.
2: Let $K \leftarrow \text{Size}(\mathsf{h})$
3: Let $n \leftarrow \sum_{i=0}^{K-1} \mathsf{h}(i)$
4: Create table P of size K
5: Let $c \leftarrow 0$
6: **for** $i \leftarrow 0 \ldots (K-1)$ **do**
7: $\quad c \leftarrow c + \mathsf{h}(i)$ ▷ cumulate histogram values
8: $\quad \mathsf{P}(i) \leftarrow c/n$
9: **return** P.

The computation of the cdf from a given histogram h is outlined in Alg. 4.1. The resulting function $\mathsf{P}(i)$ is (like the cumulative histogram) monotonically increasing and, in particular,

$$\mathsf{P}(0) = \mathsf{p}(0) \qquad \text{and} \qquad \mathsf{P}(K-1) = \sum_{i=0}^{K-1} \mathsf{p}(i) = 1. \tag{4.17}$$

This statistical formulation implicitly treats the generation of images as a random process whose exact properties are mostly unknown.[6] However, the process is usually assumed to be homogeneous (independent of the image position); i. e., each pixel value is the result of a "random experiment" on a single random variable i. The observed frequency distribution given by the histogram $\mathsf{h}(i)$ serves as a (coarse) estimate of the probability distribution $\mathsf{p}(i)$ of this random variable.

4.6.2 Principle of Histogram Specification

The goal of histogram specification is to modify a given image I_A by some point operation such that its distribution function P_A matches a *reference distribution* P_R as closely as possible. We thus look for a mapping function

$$a' = f_{\text{hs}}(a) \tag{4.18}$$

[6] Statistical modeling of the image generation process has a long tradition (see, e. g., [25, Ch. 2]).

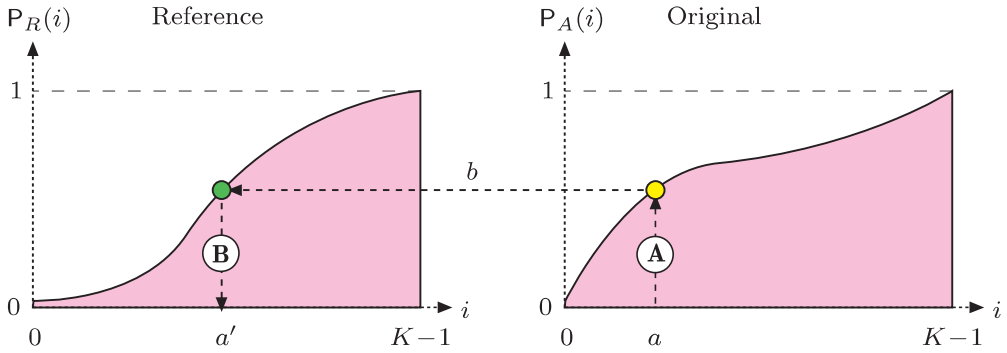


Figure 4.11 Principle of histogram specification. Given is the reference distribution P_R (left) and the distribution function for the original image P_A (right). The result is the mapping function $f_{\text{hs}} : a \rightarrow a'$ for a point operation, which replaces each pixel a in the original image I_A by a modified value a'. The process has two main steps: Ⓐ For each pixel value a, determine $b = \mathsf{P}_A(a)$ from the right distribution function. Ⓑ a' is then found by inverting the left distribution function as $a' = \mathsf{P}_R^{-1}(b)$. In summary, the result is $f_{\text{hs}}(a) = a' = \mathsf{P}_R^{-1}\big(\mathsf{P}_A(a)\big)$.

to convert the original image I_A to a new image $I_{A'}$ by a point operation such that

$$\mathsf{P}_{A'}(i) \approx \mathsf{P}_R(i) \qquad \text{for } \; 0 \le i < K. \tag{4.19}$$

As illustrated in Fig. 4.11, the desired mapping f_{hs} is found by combining the two distribution functions P_R and P_A (see [17, p. 180] for details). For a given pixel value a in the original image, we get the new pixel value a' as

$$a' = \mathsf{P}_R^{-1}\big(\mathsf{P}_A(a)\big), \tag{4.20}$$

and thus the mapping f_{hs} (Eqn. (4.18)) is obtained as

$$f_{\text{hs}}(a) = a' = \mathsf{P}_R^{-1}\big(\mathsf{P}_A(a)\big) \tag{4.21}$$

for $0 \le a < K$. This of course assumes that $\mathsf{P}_R(i)$ is invertible; i. e., that the function $\mathsf{P}_R^{-1}(b)$ exists for $b \in [0, 1]$.

4.6.3 Adjusting to a Piecewise Linear Distribution

If the reference distribution P_R is given as a continuous, invertible function, then the mapping function f_{hs} can be obtained from Eqn. (4.21) without any difficulty. In practice, it is convenient to specify the (synthetic) reference distribution as a *piecewise linear* function $\mathsf{P}_L(i)$; i. e., as a sequence of $N+1$ coordinate pairs

$$\mathcal{L} = [\langle a_0, q_0\rangle, \langle a_1, q_1\rangle, \ldots \langle a_k, q_k\rangle, \ldots \langle a_N, q_N\rangle],$$

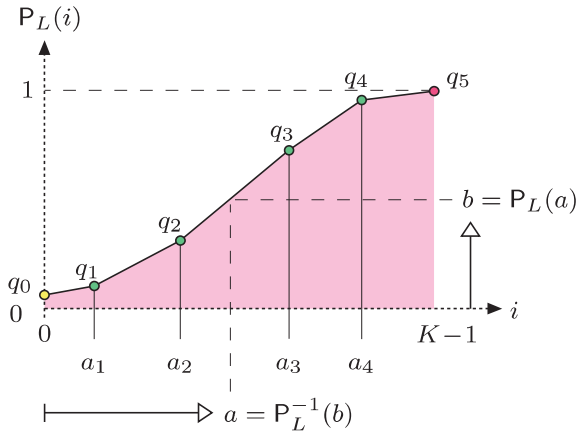


Figure 4.12 Piecewise linear reference distribution. The function $\mathsf{P}_L(i)$ is specified by $N = 5$ control points $\langle 0, q_0\rangle$, $\langle a_1, q_1\rangle$, $\ldots \langle a_4, q_4\rangle$, with $a_k < a_{k+1}$ and $q_k < q_{k+1}$. The final point q_5 is fixed at $\langle K-1, 1\rangle$.

each consisting of an intensity value a_k and the corresponding function value q_k (with $0 \le a_k < K$, $a_k < a_{k+1}$, and $0 \le q_k < 1$). The two endpoints $\langle a_0, q_0\rangle$ and $\langle a_N, q_N\rangle$ are fixed at

$$\langle 0, q_0\rangle \qquad \text{and} \qquad \langle K-1, 1\rangle,$$

respectively. To be invertible, the function must also be strictly monotonic; i. e., $q_k < q_{k+1}$ for $0 \le k < N$. Figure 4.12 shows an example for such a function, which is specified by $N = 5$ variable points $(q_0, \ldots q_4)$ and a fixed end point q_5 and thus consists of $N = 5$ linear segments. The reference distribution can of course be specified at an arbitrary accuracy by inserting additional control points.

The intermediate values of $\mathsf{P}_L(i)$ are obtained by linear interpolation between the control points as

$$\mathsf{P}_L(i) = \begin{cases} q_m + (i - a_m) \cdot \dfrac{(q_{m+1} - q_m)}{(a_{m+1} - a_m)} & \text{for } \ 0 \le i < K-1 \\ 1 & \text{for } \ i = K-1, \end{cases} \tag{4.22}$$

where $m = \max\bigl\{j \in [0, N-1] \mid a_j \le i\bigr\}$ is the index of the line segment $\langle a_m, q_m\rangle \rightarrow \langle a_{m+1}, q_{m+1}\rangle$, which overlaps the position i. For instance, in the example in Fig. 4.12, the point a lies within the segment that starts at point $\langle a_2, q_2\rangle$; i. e., $m = 2$.

For the histogram specification according to Eqn. (4.21), we also need the *inverse* distribution function $\mathsf{P}_L^{-1}(b)$ for $b \in [0, 1]$. As we see from the example in Fig. 4.12, the function $\mathsf{P}_L(i)$ is in general not invertible for values $b < \mathsf{P}_L(0)$. We can fix this problem by mapping all values $b < \mathsf{P}_L(0)$ to zero and thus

obtain a "semi-inverse" of the reference distribution in Eqn. (4.22) as

$$\mathsf{P}_L^{-1}(b) = \begin{cases} 0 & \text{for } 0 \le b < \mathsf{P}_L(0) \\ a_n + (b - q_n) \cdot \dfrac{(a_{n+1} - a_n)}{(q_{n+1} - q_n)} & \text{for } \mathsf{P}_L(0) \le b < 1 \\ K-1 & \text{for } b \ge 1. \end{cases} \tag{4.23}$$

Here $n = \max\bigl\{j \in \{0, \ldots N-1\} \mid q_j \le b\bigr\}$ is the index of the line segment $\langle a_n, q_n\rangle \rightarrow \langle a_{n+1}, q_{n+1}\rangle$, which overlaps the argument value b. The required mapping function f_{hs} for adapting a given image with intensity distribution P_A is finally specified, analogous to Eqn. (4.21), as

$$f_{\text{hs}}(a) = \mathsf{P}_L^{-1}\bigl(\mathsf{P}_A(a)\bigr) \qquad \text{for } 0 \le a < K. \tag{4.24}$$

The whole process of computing the pixel mapping function for a given image (histogram) and a piecewise linear target distribution is summarized in Alg. 4.2. A real example is shown in Fig. 4.14 (Sec. 4.6.5).

4.6.4 Adjusting to a Given Histogram (Histogram Matching)

If we want to adjust one image to the histogram of another image, the reference distribution function $\mathsf{P}_R(i)$ is not continuous and thus, in general, cannot be inverted (as required by Eqn. (4.21)). For example, if the reference distribution contains zero entries (i. e., pixel values k with probability $\mathsf{p}(k) = 0$), the corresponding cumulative distribution function P (just like the cumulative histogram) has intervals of constant value on which no inverse function value can be determined.

In the following, we describe a simple method for histogram matching that works with discrete reference distributions. The principal idea is graphically illustrated in Fig. 4.13. The mapping function f_{hs} is not obtained by inverting but by "filling in" the reference distribution function $\mathsf{P}_R(i)$. For each possible pixel value a, starting with $a = 0$, the corresponding probability $\mathsf{p}_A(a)$ is stacked layer by layer "under" the reference distribution P_R. The thickness of each horizontal bar for a equals the corresponding probability $\mathsf{p}_A(a)$. The bar for a particular intensity value a with thickness $\mathsf{p}_A(a)$ runs from right to left, down to position a', where it hits the reference distribution P_R. This position a' corresponds to the new pixel value to which a should be mapped.

Since the sum of all probabilities p_A and the maximum of the distribution function P_R are both 1 (i. e., $\sum_i \mathsf{p}_A(i) = \max_i \mathsf{P}_R(i) = 1$), all horizontal bars will exactly fit underneath the function P_R. One may also notice in Fig. 4.13 that the distribution value resulting at a' is identical to the cumulated probability $\mathsf{P}_A(a)$. Given some intensity value a, it is therefore sufficient to find the

Algorithm 4.2 Histogram specification using a piecewise linear reference distribution. Given is the histogram h_A of the original image and a piecewise linear reference distribution function, specified as a sequence of N control points $\mathcal{L}_R$. The discrete mapping function f_{hs} for the corresponding point operation is returned.

1: MatchPiecewiseLinearHistogram(h_A, $\mathcal{L}_R$)

h_A: histogram of the original image I_A.
$\mathcal{L}_R$: reference distribution function, given as a sequence of $N+1$ control points $\mathcal{L}_R = [\langle a_0, q_0\rangle, \langle a_1, q_1\rangle, \ldots \langle a_N, q_N\rangle]$, with $0 \le a_k < K$ and $0 \le q_k \le 1$.

Returns a discrete pixel mapping function $f_{\text{hs}}(a)$ for modifying the original image I_A.

2: Let $K \leftarrow \text{Size}(\mathsf{h}_A)$
3: Let $\mathsf{P}_A \leftarrow \text{Cdf}(\mathsf{h}_A)$ ▷ cdf for h_A (Alg. 4.1)
4: Create a table $f_{\text{hs}}[\,]$ of size K ▷ mapping function f_{hs}
5: **for** $a \leftarrow 0 \ldots (K-1)$ **do**
6: $b \leftarrow \mathsf{P}_A(a)$
7: **if** $(b \le q_0)$ **then**
8: $a' \leftarrow 0$
9: **else if** $(b \ge 1)$ **then**
10: $a' \leftarrow K-1$
11: **else**
12: $n \leftarrow N-1$
13: **while** $(n \ge 0) \wedge (q_n > b)$ **do** ▷ find line segment in $\mathcal{L}_R$
14: $n \leftarrow n-1$
15: $a' \leftarrow a_n + (b-q_n)\cdot\dfrac{(a_{n+1}-a_n)}{(q_{n+1}-q_n)}$ ▷ see Eqn. (4.23)
16: $f_{\text{hs}}[a] \leftarrow a'$
17: **return** f_{hs}.

minimum value a', where the reference distribution $\mathsf{P}_R(a')$ is greater than or equal to the cumulative probability $\mathsf{P}_A(a)$; i. e.,

$$f_{\text{hs}}(a) = a' = \min\big\{\, j \mid (0 \le j < K) \wedge \big(\mathsf{P}_A(a) \le \mathsf{P}_R(j)\big)\big\}. \tag{4.25}$$

This results in a very simple method, which is summarized in Alg. 4.3. Due to the use of normalized distribution functions, the *size* of the images involved is not relevant. The corresponding Java implementation in Prog. 4.3, consists of the method `matchHistograms()`, which accepts the original histogram (`Ha`) and the reference histogram (`Hr`) and returns the resulting mapping function (`map`) specifying the required point operation. The following code fragment

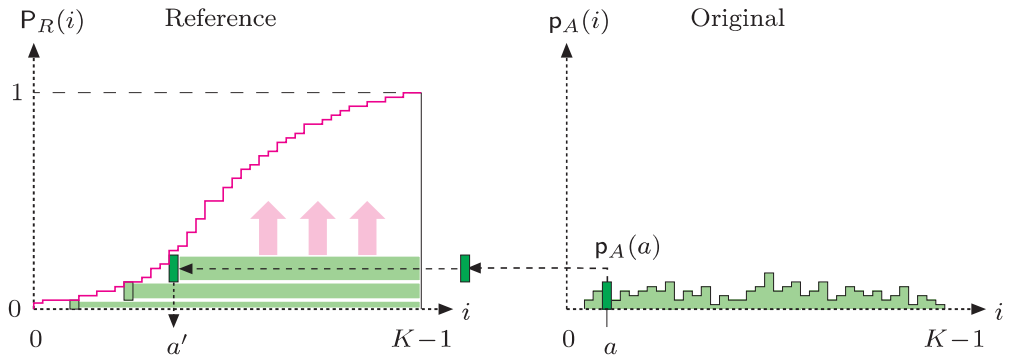


Figure 4.13 Discrete histogram specification. The reference distribution P_R (left) is "filled" layer by layer from bottom to top and from right to left. For every possible intensity value a (starting from $a = 0$), the associated probability $p_A(a)$ is added as a horizontal bar to a stack accumulated 'under" the reference distribution P_R. The bar with thickness $p_A(a)$ is drawn from right to left down to the position a', where the reference distribution P_R is reached. This value a' is the one which a should be mapped to by the function $f_{\text{hs}}(a)$.

demonstrates the use of the method `matchHistograms()` from Prog. 4.3 in an ImageJ program:

```
ImageProcessor ipA = ...    // target image I_A (to be modified)
ImageProcessor ipR = ...    // reference image I_R

int[] hA = ipA.getHistogram(); // get the histogram for I_A
int[] hR = ipR.getHistogram(); // get the histogram for I_R

int[] F = matchHistograms(hA, hR); // mapping function f_hs(a)
ipA.applyTable(F);          // apply f_hs() to the target image I_A
```

The original image `ipA` is modified in the last line by applying the mapping function f_{hs} (`F`) with the method `applyTable()` (see also p. 87).

4.6.5 Examples

Adjusting to a piecewise linear reference distribution

The first example in Fig. 4.14 shows the results of histogram specification for a continuous, piecewise linear reference distribution, as described in Sec. 4.6.3. Analogous to Fig. 4.12, the actual distribution function P_R (Fig. 4.14 (f)) is specified as a polygonal line consisting of five control points $\langle a_k, q_k \rangle$ with coordinates

$k =$	0	1	2	3	4	5
$a_k =$	0	28	75	150	210	255
$q_k =$	0.002	0.050	0.250	0.750	0.950	1.000

The resulting reference histogram (Fig. 4.14 (c)) is a step function with ranges of constant values corresponding to the linear segments of the probability density

Algorithm 4.3 Histogram matching. Given are two histograms: the histogram h_A of the target image I_A and a reference histogram h_R, both of size K. The result is a discrete mapping function $f_{\text{hs}}()$ that, when applied to the target image, produces a new image with a distribution function similar to the reference histogram.

1: MatchHistograms(h_A, h_R)

h_A: histogram of the target image I_A.
h_R: reference histogram (of same size as h_A).

Returns a discrete pixel mapping function $f_{\text{hs}}(a)$ for modifying the original image I_A.

2: Let $K \leftarrow \text{Size}(\mathsf{h}_A)$
3: Let $\mathsf{P}_A \leftarrow \text{Cdf}(\mathsf{h}_A)$ ▷ cdf for h_A (Alg. 4.1)
4: Let $\mathsf{P}_R \leftarrow \text{Cdf}(\mathsf{h}_R)$ ▷ cdf for h_R (Alg. 4.1)
5: Create a table $f_{\text{hs}}[\,]$ of size K ▷ pixel mapping function f_{hs}
6: **for** $a \leftarrow 0 \ldots (K-1)$ **do**
7: $j \leftarrow K-1$
8: **repeat**
9: $f_{\text{hs}}[a] \leftarrow j$
10: $j \leftarrow j-1$
11: **while** $(j \geq 0) \wedge (\mathsf{P}_A(a) \leq \mathsf{P}_R(j))$
12: **return** f_{hs}.

function. As expected, the *cumulative* probability function for the modified image (Fig. 4.14 (h)) is quite close to the reference function in Fig. 4.14 (f), while the resulting *histogram* (Fig. 4.14 (e)) shows little similarity with the reference histogram (Fig. 4.14 (c)). However, as discussed earlier, this is all we can expect from a homogeneous point operation.

Adjusting to an arbitrary reference histogram

In this case, the reference distribution is not given as a continuous function but specified by a discrete histogram. We thus use the method described in Sec. 4.6.4 to compute the required mapping functions. The examples in Fig. 4.15 demonstrate this technique using synthetic reference histograms whose shape is approximately Gaussian.

The target image (Fig. 4.15 (a)) used here was chosen intentionally for its poor quality, manifested by an extremely unbalanced histogram (Fig. 4.15 (f)). The histograms of the modified images thus naturally show little resemblance to a Gaussian. However, the resulting *cumulative* histograms (Fig. 4.15 (j, k)) match nicely with the integral of the corresponding Gaussians (Fig. 4.15 (d, e)),

```
1  int[] matchHistograms (int[] hA, int[] hR) {
2    // hA ... histogram h_A of target image I_A
3    // hR ... reference histogram h_R
4    // returns the mapping function f_hs() to be applied to image I_A
5
6    int K = hA.length;          // hA, hR must be of length K
7
8    double[] PA = Cdf(hA);      // get CDF of histogram h_A
9    double[] PR = Cdf(hR);      // get CDF of histogram h_R
10
11   int[] F = new int[K];       // pixel mapping function f_hs()
12
13   // compute mapping function f_hs()
14   for (int a = 0; a < K; a++) {
15     int j = K-1;
16     do {
17       F[a] = j;
18       j--;
19     } while (j>=0 && PA[a]<=PR[j]);
20   }
21
22   return F;
23 }
```

```
25 double[] Cdf (int[] h) {
26   // returns the cumulative distribution function for histogram h
27   int K = h.length;
28   int n = 0;                  // sum all histogram values
29   for (int i=0; i<K; i++) {
30     n += h[i];
31   }
32
33   double[] P = new double[K]; // create cdf table P
34   int c = 0;                  // cumulate histogram values
35   for (int i=0; i<K; i++) {
36     c += h[i];
37     P[i] = (double) c / n;
38   }
39
40   return P;
41 }
```

Program 4.3 Histogram matching (Java implementation of Alg. 4.3). The method `matchHistograms()` computes the mapping function `F` from the target histogram `Ha` and the reference histogram `Hr` (see Eqn. (4.25)). The method `Cdf()` computes the cumulative distribution function (cdf) for a given histogram (Eqn. (4.16)).

apart from the unavoidable irregularity at the center caused by the dominant peak in the original histogram.

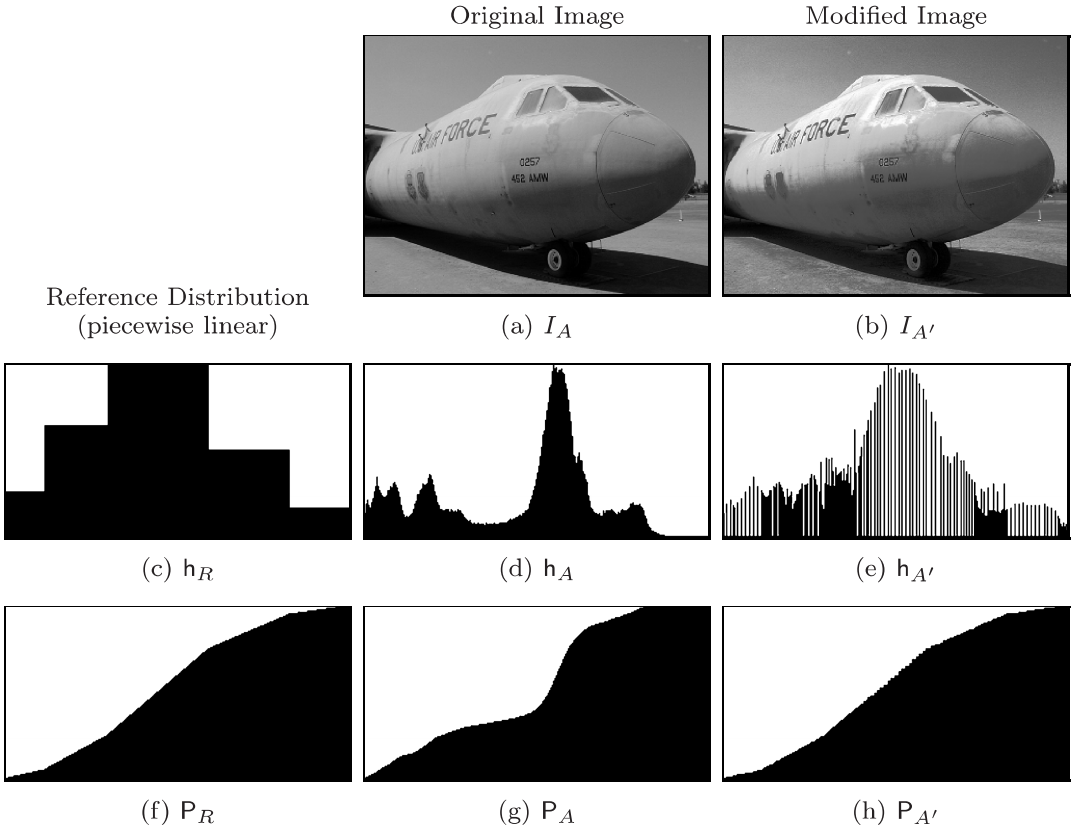


Figure 4.14 Histogram specification with a piecewise linear reference distribution. The target image I_A (a), its histogram (d), and distribution function P_A (g); the reference histogram h_R (c) and the corresponding distribution P_R (f); the modified image $I_{A'}$ (b), its histogram $\mathsf{h}_{A'}$ (e), and the resulting distribution $\mathsf{P}_{A'}$ (h).

Adjusting to another image

The third example in Fig. 4.16 demonstrates the adjustment of two images by matching their intensity histograms. One of the images is selected as the reference image I_R (Fig. 4.16 (b)) and supplies the reference histogram h_R (Fig. 4.16 (e)). The second (target) image I_A (Fig. 4.16 (a)) is modified such that the resulting cumulative histogram matches the cumulative histogram of the reference image I_R. It can be expected that the final image $I_{A'}$ (Fig. 4.16 (c)) and the reference image give a similar visual impression with regard to tonal range and distribution (assuming that both images show similar content).

Of course this method may be used to adjust multiple images to the same reference image (e. g., to prepare a series of similar photographs for a print project). For this purpose, one could either select a single representative image as a common reference or, alternatively, compute an "average" reference histogram from a set of typical images (see also Exercise 4.7).

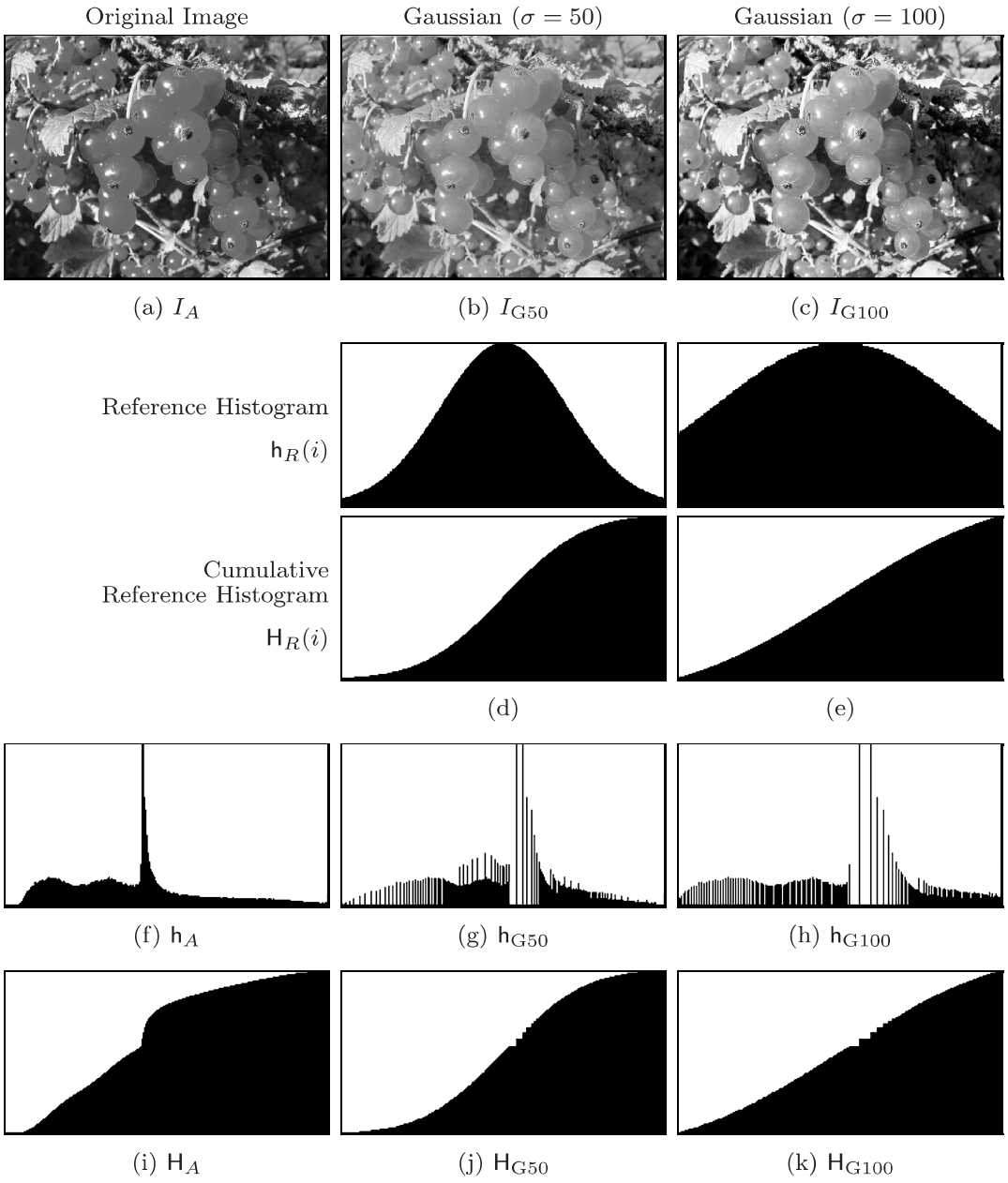


Figure 4.15 Histogram matching: adjusting to a synthetic histogram. Original image I_A (a), corresponding histogram (f), and cumulative histogram (i). Gaussian-shaped reference histograms with center $\mu = 128$ and $\sigma = 50$ (d) and $\sigma = 100$ (e), respectively. Resulting images after histogram matching, I_{G50} (b) and I_{G100} (c) with the corresponding histograms (g, h) and cumulative histograms (j, k).

4.7 Gamma Correction

We have been using the terms "intensity" and "brightness" many times without really bothering with how the numeric pixel values in our images relate to these

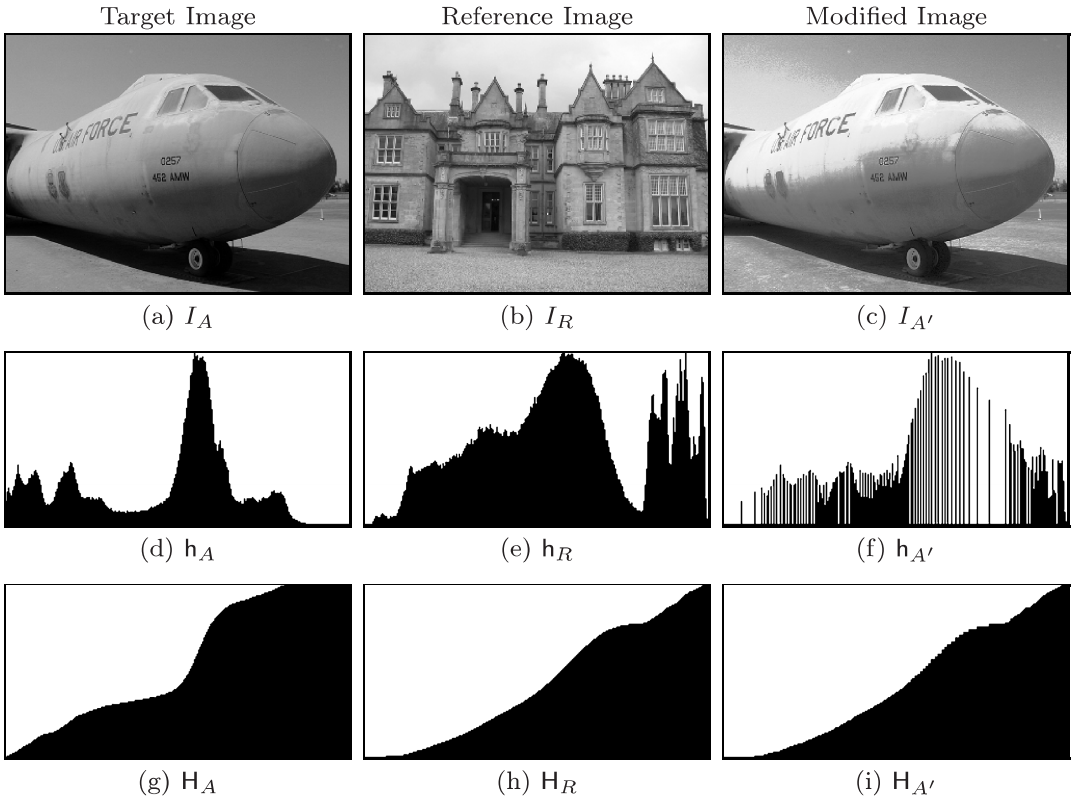


Figure 4.16 Histogram matching: adjusting to a reference image. The target image I_A (a) is modified by matching its histogram to the reference image I_R (b), resulting in the new image $I_{A'}$ (c). The corresponding histograms h_A, h_R, $\mathsf{h}_{A'}$ (d–f) and cumulative histograms $\mathsf{H}_A, \mathsf{H}_R, \mathsf{P}_{A'}$ (g–i) are shown. Notice the good agreement between the cumulative histograms of the reference and adjusted images (h, i).

physical concepts, if at all. A pixel value may represent the amount of light falling onto a sensor element in a camera, the photographic density of film, the amount of light to be emitted by a monitor, the number of toner particles to be deposited by a printer, or any other relevant physical magnitude. In practice, the relationship between a pixel value and the corresponding physical quantity is usually complex and almost always nonlinear. In many imaging applications, it is important to know this relationship, at least approximately, to achieve consistent and reproducible results.

When applied to digital intensity images, the ideal is to have some kind of "calibrated intensity space" that optimally matches the human perception of intensity and requires a minimum number of bits to represent the required intensity range. Gamma correction denotes a simple point operation to compensate for the transfer characteristics of different input and output devices and to map them to a unified intensity space.

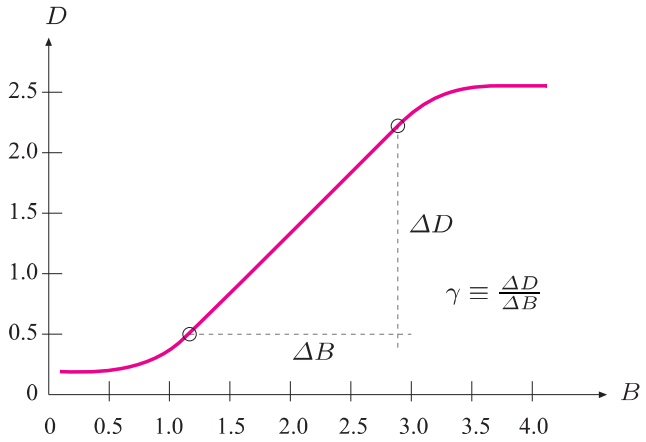


Figure 4.17 Exposure function of photographic film. With respect to the *logarithmic* light intensity B, the resulting film density D is approximately *linear* over a wide intensity range. The slope ($\Delta D/\Delta B$) of this linear section of the function specifies the "gamma" (γ) value for a particular type of photographic material.

4.7.1 Why Gamma?

The term "gamma" originates from analog photography, where the relationship between the light energy and the resulting film density is approximately logarithmic. The "exposure function" (Fig. 4.17), specifying the relationship between the *logarithmic* light intensity and the resulting film density, is therefore approximately *linear* over a wide range of light intensities. The slope of this function within this linear range is traditionally referred to as the "gamma" of the photographic material. The same term was adopted later in television broadcasting to describe the nonlinearities of the cathode ray tubes used in TV receivers, i. e., to model the relationship between the amplitude (voltage) of the video signal and the emitted light intensity. To compensate for the nonlinearities of the receivers, a "gamma correction" was (and is) applied to the TV signal once before broadcasting in order to avoid the need for costly correction measures on the receiver side.

4.7.2 Power Function

Gamma correction is based on the power function

$$f_\gamma(a) = a^\gamma \quad \text{for } a \in \mathbb{R} \text{ and } \gamma > 0, \tag{4.26}$$

where the parameter γ is called the gamma *value*. If a is constrained to the interval $[0, 1]$, then—independent of γ—the value of $f_\gamma(a)$ also stays within $[0, 1]$, and the function always runs through the points $(0, 0)$ and $(1, 1)$. In particular, $f_\gamma(a)$ is the identity function for $\gamma = 1$, as shown in Fig. 4.18. The function runs *above* the diagonal for gamma values $\gamma < 1$, and *below* it for

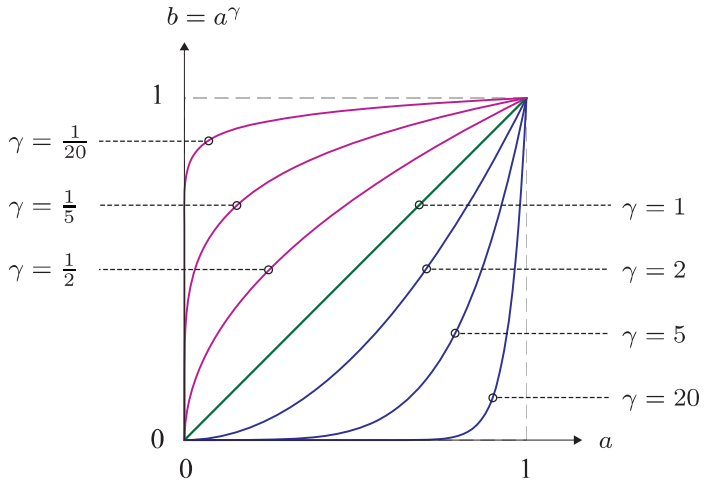


Figure 4.18 Power function $b = f_\gamma(a) = a^\gamma$ for $a \in [0, 1]$ for different gamma values.

$\gamma > 1$. Controlled by a single continuous parameter (γ), the power function can thus "imitate" both logarithmic and exponential types of functions. Within the interval $[0, 1]$, the function is continuous and strictly monotonic, and also very simple to invert as

$$a = f_\gamma^{-1}(b) = b^{1/\gamma}, \tag{4.27}$$

since $b^{1/\gamma} = (a^\gamma)^{1/\gamma} = a^1 = a$. The inverse of the power function $f_\gamma^{-1}(b)$ is thus again a power function,

$$f_\gamma^{-1}(b) = f_{\bar{\gamma}}(b) = f_{1/\gamma}(b), \tag{4.28}$$

with $\bar{\gamma} = 1/\gamma$. Thus the inverse of the power function with parameter γ is another power function with parameter $\bar{\gamma} = 1/\gamma$.

4.7.3 Real Gamma Values

The actual gamma values of individual devices are usually specified by the manufacturers based on real measurements. For example, common gamma values for CRT monitors are in the range 1.8 to 2.8, with 2.4 as a typical value. Most LCD monitors are internally adjusted to similar values. Digital video and still cameras also emulate the transfer characteristics of analog film and photographic cameras by making internal corrections to give the resulting images an accustomed "look".

In TV receivers, gamma values are standardized with 2.2 for analog NTSC and 2.8 for the PAL system (these values are theoretical; results of actual measurements are around 2.35). A gamma value of $1/2.2 \approx 0.45$ is the norm for cameras in NTSC as well as the EBU[7] standards. The current international

[7] European Broadcast Union (EBU).

standard ITU-R BT.709[8] calls for uniform gamma values of 2.5 in receivers and $1/1.956 \approx 0.51$ for cameras [15,20]. The ITU 709 standard is based on a slightly modified version of the gamma correction (see Sec. 4.7.6).

Computers usually allow adjustment of the gamma value applied to the video output signals to adapt to a wide range of different monitors. Note however that the power function $f_\gamma()$ is only a coarse approximation to the actual transfer characteristics of any device, which may also not be the same for different color channels. Thus significant deviations may occur in practice, despite the careful choice of gamma settings. Critical applications, such as prepress or high-end photography, usually require additional calibration efforts based on exactly measured device profiles (see Vol. 2 [6, Sec. 6.6.5]).

4.7.4 Applications of Gamma Correction

Let us first look at the simple example illustrated in Fig. 4.19. Assume that we use a digital camera with a nominal gamma value γ_c, meaning that its output signal s relates to the incident light intensity L as

$$S = L^{\gamma_c}. \tag{4.29}$$

To compensate the transfer characteristic of this camera (i. e., to obtain a measurement S' that is proportional to the original light intensity L), the camera signal S is subject to a gamma correction with the inverse of the camera's gamma value $\bar{\gamma}_c = 1/\gamma_c$, so

$$S' = f_{\bar{\gamma}_c}(S) = S^{1/\gamma_c}. \tag{4.30}$$

The resulting signal $S' = S^{1/\gamma_c} = (L^{\gamma_c})^{1/\gamma_c} = L^{(\gamma_c \cdot \frac{1}{\gamma_c})} = L^1$ is obviously proportional (in theory even identical) to the original light intensity L.

Although the above example is overly simplistic, it still demonstrates the general rule, which holds for output devices as well:

> The transfer characteristic of an input or output device with specified gamma value γ is compensated for by a gamma correction with $\bar{\gamma} = 1/\gamma$.

In the above, we have implicitly assumed that all values are strictly in the range $[0, 1]$, which usually is not the case in practice. When working with digital images, we have to deal with discrete pixel values; e. g., in the range $[0, 255]$ for 8-bit images. In general, performing a gamma correction

$$b = f_{\text{gc}}(a, \gamma),$$

on a pixel value $a \in [0, a_{\max}]$ and a gamma value $\gamma > 0$ requires the following three steps:

[8] International Telecommunications Union (ITU).

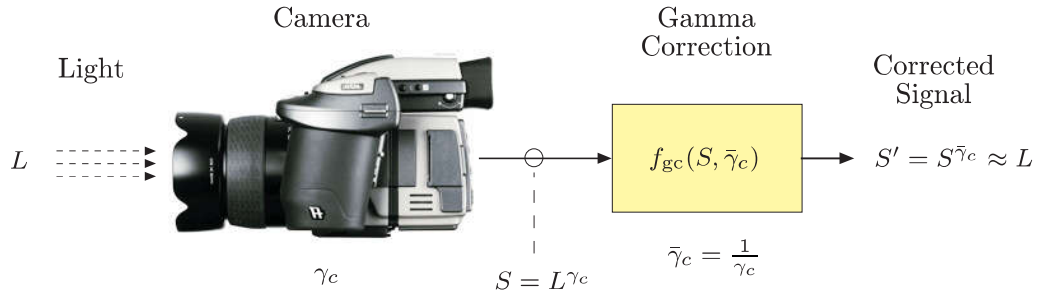


Figure 4.19 Principle of gamma correction. To compensate the output signal S produced by a camera with nominal gamma value γ_c, a gamma correction is applied with $\bar{\gamma}_c = 1/\gamma_c$. The corrected signal S' is proportional to the received light intensity L.

1. Scale a linearly to $a' \in [0, 1]$.
2. Apply the gamma correction function to a': $\quad b' \leftarrow f_\gamma(a') = a'^\gamma$.
3. Scale $b' \in [0, 1]$ linearly back to $b \in [0, a_{\max}]$.

Formulated in a more compact way, the corrected pixel value b is obtained from the original value a as

$$b = f_{\text{gc}}(a, \gamma) = \left(\frac{a}{a_{\max}}\right)^\gamma \cdot a_{\max}. \tag{4.31}$$

Figure 4.20 illustrates the typical role of gamma correction in the digital work flow with two input (camera, scanner) and two output devices (monitor, printer), each with its individual gamma value. The central idea is to correct all images to be processed and stored in a device-independent, standardized intensity space.

4.7.5 Implementation

Program 4.4 shows the implementation of gamma correction as an ImageJ plugin for 8-bit grayscale images. The mapping function $f_{\text{gc}}(a, \gamma)$ is computed as a lookup table (`Fgc`), which is then applied to the image using the method `applyTable()` to perform the actual point operation (see also Sec. 4.8.1).

4.7.6 Modified Gamma Correction

A subtle problem with the simple power function $f_\gamma(a) = a^\gamma$ (Eqn. (4.26)) appears if we take a closer look at the *slope* of this function, expressed by its first derivative,

$$f'_\gamma(a) = \gamma \cdot a^{(\gamma-1)},$$

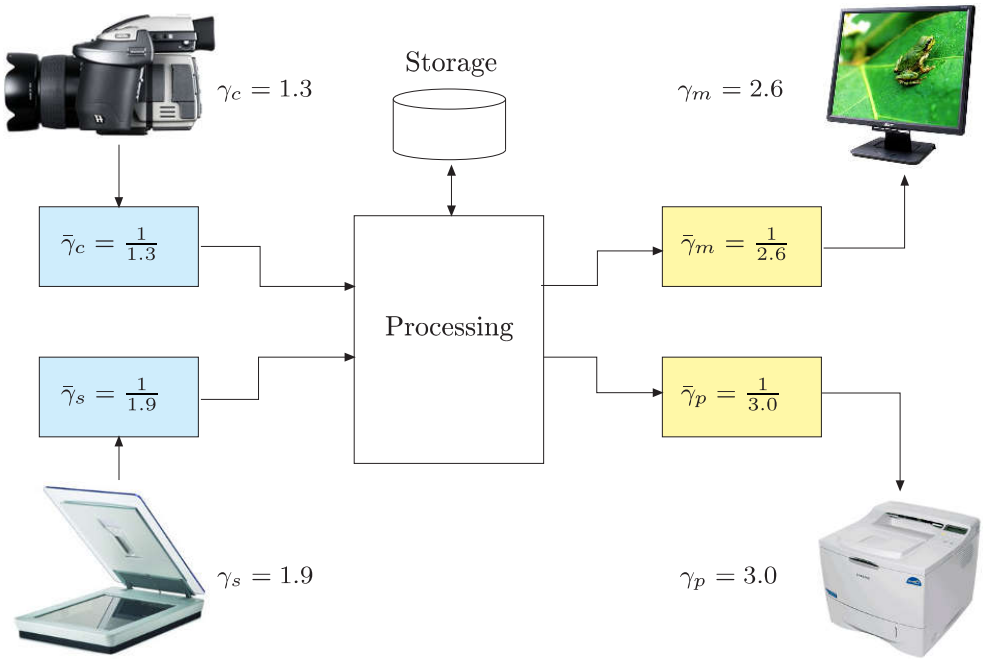


Figure 4.20 Gamma correction in the digital imaging work flow. Images are processed and stored in a "linear" intensity space, where gamma correction is used to compensate for the transfer characteristic of each input and output device. (The gamma values shown are examples only.)

```
1  public void run(ImageProcessor ip) {
2      // works for 8-bit images only
3      int K = 256;
4      int aMax = K - 1;
5      double GAMMA = 2.8;
6
7      // create a lookup table for the mapping function
8      int[] Fgc = new int[K];
9
10     for (int a = 0; a < K; a++) {
11         double aa = (double) a / aMax;  // scale to [0, 1]
12         double bb = Math.pow(aa,GAMMA); // power function
13         // scale back to [0, 255]:
14         int b = (int) Math.round(bb * aMax);
15         Fgc[a] = b;
16     }
17
18     ip.applyTable(Fgc);  // modify the image ip
19 }
```

Program 4.4 Gamma correction (ImageJ plugin). The corrected intensity values `b` are only computed once and stored in the lookup table `Fgc` (line 15). The gamma value `GAMMA` is constant. The actual point operation is performed by calling the ImageJ method `applyTable(Fgc)` on the image object `ip` (line 18).

which for $a = 0$ has the values

$$f'_\gamma(0) = \begin{cases} 0 & \text{for } \gamma > 1 \\ 1 & \text{for } \gamma = 1 \\ \infty & \text{for } \gamma < 1. \end{cases} \tag{4.32}$$

The tangent to the function at the origin is thus either horizontal ($\gamma > 1$), diagonal ($\gamma = 1$), or vertical ($\gamma < 1$), with no intermediate values. For $\gamma < 1$, this causes extremely high amplification of small intensity values and thus increased noise in dark image regions. Theoretically, this also means that the power function is generally not invertible at the origin.

A common solution to this problem is to replace the lower part ($0 \le a \le a_0$) of the power function by a linear segment with constant slope and to continue with the ordinary power function for $a > a_0$. The resulting modified gamma correction $\bar{f}_{\gamma,a_0}(a)$ is defined as

$$\bar{f}_{\gamma,a_0}(a) = \begin{cases} s \cdot a & \text{for } \ 0 \le a \le a_0 \\ (1+d) \cdot a^\gamma - d & \text{for } \ a_0 < a \le 1 \end{cases} \tag{4.33}$$

$$\text{with} \quad s = \frac{\gamma}{a_0(\gamma-1) + a_0^{(1-\gamma)}} \qquad \text{and} \qquad d = \frac{1}{a_0^\gamma(\gamma-1)+1} - 1. \tag{4.34}$$

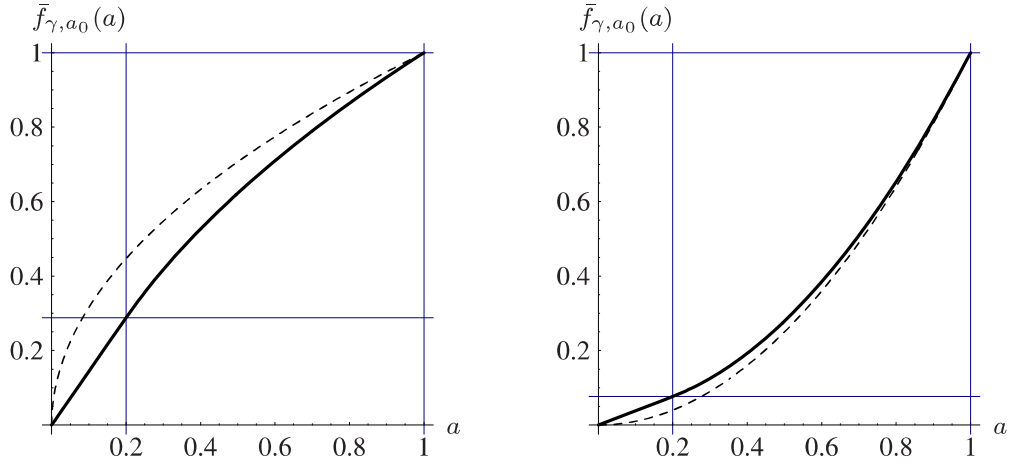
The function thus consists of a *linear* section (for $0 \le a \le a_0$) and a *nonlinear* section (for $a_0 < a \le 1$) that connect smoothly at the transition point $a = a_0$. The linear slope s and the parameter d are determined by the requirement that the two function segments must have identical values as well as identical slopes (first derivatives) at $a = a_0$ to give a (C1) continuous function. The function in Eqn. (4.33) is thus fully specified by the two parameters a_0 and γ.

Figure 4.21 shows two examples of the modified gamma correction $\bar{f}_{\gamma,a_0}()$ with values $\gamma = 0.5$ and $\gamma = 2.0$, respectively. In both cases, the transition point is at $a_0 = 0.2$. For comparison, the figure also shows the ordinary gamma correction $f_\gamma(a)$ for the same gamma values (dashed lines), whose slope at the origin is ∞ (Fig. 4.21 (a)) and zero (Fig. 4.21 (b)), respectively.

Gamma correction in common standards

The modified gamma correction is part of several modern imaging standards. In practice, however, the values of a_0 are considerably smaller than the ones used for the illustrative examples in Fig. 4.21, and γ is chosen to obtain a good overall match to the desired correction function. For example, the ITU-BT.709 specification [20] mentioned in Sec. 4.7.3 specifies the parameters

$$\gamma = \frac{1}{2.222} \approx 0.45 \quad \text{and} \quad a_0 = 0.018,$$



(a) $\gamma = 0.5$, $a_0 = 0.2$ (b) $\gamma = 2.0$, $a_0 = 0.2$

Figure 4.21 The modified gamma correction $\bar{f}_{\gamma,a_0}(a)$ consists of a linear segment with fixed slope s between $a = 0$ and $a = a_0$, followed by a power function with parameter γ (Eqn. (4.33)). The dashed lines show the ordinary power functions for the same gamma values.

Table 4.1 Gamma correction parameters for the ITU and sRGB standards Eqns. (4.33) and (4.34).

Standard	Nominal Gamma Value γ	a_0	s	d	Effective Gamma Value γ_{eff}
ITU BT.709	$1/2.222 \approx 0.450$	0.01800	4.5068	0.09915	$1/1.956 \approx 0.511$
sRGB	$1/2.400 \approx 0.417$	0.00304	12.9231	0.05500	$1/2.200 \approx 0.455$

with the corresponding slope and offset values $s = 4.50681$ and $d = 0.0991499$, respectively (Eqn. (4.34)). The resulting correction function $\bar{f}_{\text{ITU}}(a)$ has a *nominal* gamma value of 0.45, which corresponds to the *effective* gamma value $\gamma_{\text{eff}} = 1/1.956 \approx 0.511$. The gamma correction in the sRGB standard [40] is specified on the same basis (with different parameters; see Vol. 2 [6, Sec. 6.3]).

Figure 4.22 shows the actual correction functions for the ITU and sRGB standards, respectively, each in comparison with the equivalent ordinary gamma correction. The ITU function (Fig. 4.22 (a)) with $\gamma = 0.45$ and $a_0 = 0.018$ corresponds to an ordinary gamma correction with effective gamma value $\gamma_{\text{eff}} = 0.511$ (dashed line). The curves for sRGB (Fig. 4.22 (b)) differ only by the parameters γ and a_0, as summarized in Table 4.1.

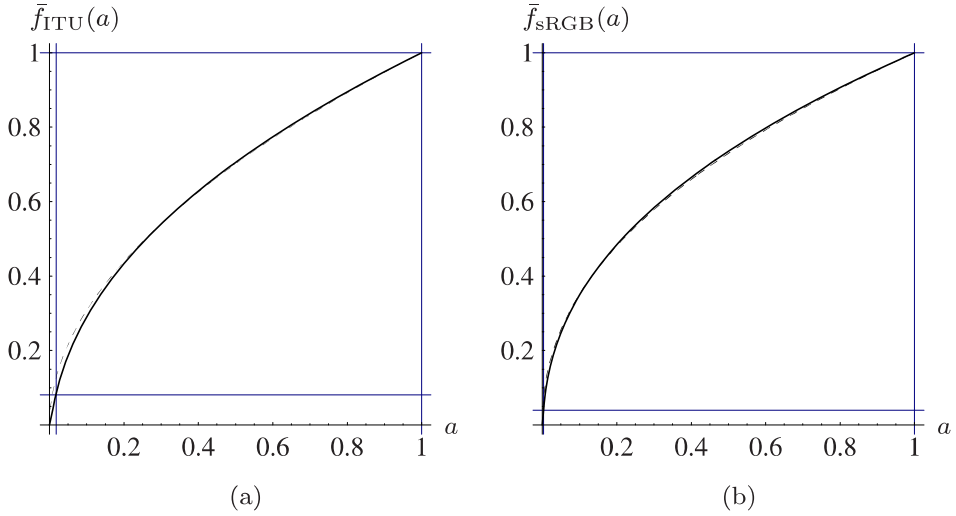


Figure 4.22 Gamma correction functions specified by the ITU-R BT.709 (a) and sRGB (b) standards. The continuous plot shows the modified gamma correction with the nominal gamma value γ and transition point a_0. The dashed lines mark the equivalent ordinary gamma correction with effective gamma γ_{eff}.

Inverting the modified gamma correction

To invert the modified gamma correction of the form $b = \bar{f}_{\gamma,a_0}(a)$ (Eqn. (4.33)), we need the inverse of the function $\bar{f}_{\gamma,a_0}()$, which is again defined in two parts,

$$\bar{f}^{-1}_{\gamma,a_0}(b) = \begin{cases} b/s & \text{for } 0 \le b \le s \cdot a_0 \\ \left(\dfrac{b+d}{1+d}\right)^{1/\gamma} & \text{for } s \cdot a_0 < b \le 1, \end{cases} \tag{4.35}$$

where s and d are the values defined in Eqn. (4.34). The inverse gamma correction function is required in particular for transforming between different color spaces if nonlinear (i. e., gamma-corrected) component values are involved (see also Vol. 2 [6, Sec. 6.2]).

4.8 Point Operations in ImageJ

Several important types of point operations are already implemented in ImageJ, so there is no need to program every operation manually (as shown in Prog. 4.4). In particular, it is possible in ImageJ to apply point operations efficiently by using tabulated functions, to use built-in standard functions for point operations on single images, and to apply arithmetic operations on pairs of images. These issues are described briefly in the remaining parts of this section.

4.8.1 Point Operations with Lookup Tables

Some point operations require complex computations for each pixel, and the processing of large images may be quite time-consuming. If the point operation is *homogeneous* (i. e., independent of the pixel coordinates), the value of the mapping function can be precomputed for every possible pixel value and stored in a lookup table, which may then be applied very efficiently to the image. A lookup table $\mathbf{L}$ represents a discrete mapping (function f) from the original to the new pixel values,

$$\mathbf{L} : [0, K-1] \xrightarrow{f} [0, K-1]. \tag{4.36}$$

For a point operation specified by a particular pixel mapping function $a' = f(a)$, the table $\mathbf{L}$ is initialized with the values

$$\mathbf{L}(a) \leftarrow f(a) \qquad \text{for } 0 \le a < K. \tag{4.37}$$

Thus the K table elements of $\mathbf{L}$ need only be computed once, where typically $K = 256$. Performing the actual point operation only requires a simple (and quick) table lookup in $\mathbf{L}$ at each pixel,

$$I(u, v) \leftarrow \mathbf{L}(I(u, v)), \tag{4.38}$$

which is much more efficient than any individual function call. ImageJ provides the method

```
void applyTable(int[] lut)
```

for objects of type `ImageProcessor`, which requires a lookup table *`lut`* ($\mathbf{L}$) as a one-dimensional `int` array of size K (see Prog. 4.4 on page 83 for an example). The advantage of this approach is obvious: for an 8-bit image, for example, the mapping function is evaluated only 256 times (independent of the image size) and not a million times or more as in the case of a large image. The use of lookup tables for implementing point operations thus always makes sense if the number of image pixels ($M \times N$) is greater than the number of possible pixel values K (which is usually the case).

4.8.2 Arithmetic Operations

ImageJ implements a set of common arithmetic operations as methods for the class `ImageProcessor`, which are summarized in Table 4.2. In the following example, the image is multiplied by a scalar constant (1.5) to increase its contrast:

```
ImageProcessor ip = ... //some image ip
ip.multiply(1.5);
```

Table 4.2 ImageJ methods for arithmetic operations applicable to objects of type `ImageProcessor`.

`void abs()`	$I(u,v) \leftarrow \lvert I(u,v)\rvert$
`void add(int p)`	$I(u,v) \leftarrow I(u,v) + p$
`void gamma(double g)`	$I(u,v) \leftarrow \big(I(u,v)/255\big)^g \cdot 255$
`void invert(int p)`	$I(u,v) \leftarrow 255 - I(u,v)$
`void log()`	$I(u,v) \leftarrow \log_{10}\big(I(u,v)\big)$
`void max(double s)`	$I(u,v) \leftarrow \max\big(I(u,v), s\big)$
`void min(double s)`	$I(u,v) \leftarrow \min\big(I(u,v), s\big)$
`void multiply(double s)`	$I(u,v) \leftarrow \text{round}\big(I(u,v) \cdot s\big)$
`void sqr()`	$I(u,v) \leftarrow I(u,v)^2$
`void sqrt()`	$I(u,v) \leftarrow \sqrt{I(u,v)}$

The image `ip` is destructively modified by all of these methods, with the results being limited (clamped) to the minimum and maximum pixel values, respectively.

4.8.3 Point Operations Involving Multiple Images

Point operations may involve more than one image at once, with arithmetic operations on the pixels of *pairs* of images being a special but important case. For example, we can express the pointwise *addition* of two images I_1 and I_2 (of identical size) to create a new image I' as

$$I'(u,v) \;\leftarrow\; I_1(u,v) + I_2(u,v) \tag{4.39}$$

for all positions (u,v). In general, any function $f(a_1, a_2, \ldots, a_n)$ over n pixel values a_i may be defined to perform pointwise combinations of n images, i. e.,

$$I'(u,v) \;\leftarrow\; f\big(I_1(u,v), I_2(u,v), \ldots I_n(u,v)\big). \tag{4.40}$$

However, most arithmetic operations on multiple images required in practice can be implemented as sequences of successive binary operations on pairs of images.

4.8.4 Methods for Point Operations on Two Images

ImageJ supplies a single method for implementing arithmetic operations on pairs of images,

Table 4.3 Arithmetic operations for pairs of images provided by `ImageProcessor`'s `copyBits()` method. The constants `ADD`, etc., listed in this table are defined by ImageJ's `Blitter` interface.

`ADD`	$I_1(u,v) \leftarrow I_1(u,v) + I_2(u,v)$
`AVERAGE`	$I_1(u,v) \leftarrow \big(I_1(u,v) + I_2(u,v)\big) \,/\, 2$
`DIFFERENCE`	$I_1(u,v) \leftarrow \lvert I_1(u,v) - I_2(u,v)\rvert$
`DIVIDE`	$I_1(u,v) \leftarrow I_1(u,v) \,/\, I_2(u,v)$
`MAX`	$I_1(u,v) \leftarrow \max\big(I_1(u,v), I_2(u,v)\big)$
`MIN`	$I_1(u,v) \leftarrow \min\big(I_1(u,v), I_2(u,v)\big)$
`MULTIPLY`	$I_1(u,v) \leftarrow I_1(u,v) \cdot I_2(u,v)$
`SUBTRACT`	$I_1(u,v) \leftarrow I_1(u,v) - I_2(u,v)$

`void copyBits(ImageProcessor` *`ip2`*`, int` *`u`*`, int` *`v`*`, int` *`mode`*`)`,

which applies the binary operation specified by the transfer mode parameter *`mode`* to all pixel pairs taken from the *source image* *`ip2`* and the *target image* (the image on which this method is invoked) and stores the result in the target image. *`u`*, *`v`* are the coordinates where the source image is inserted into the target image (usually $u = v = 0$). The code fragment in the following example demonstrates the addition of two images:

```
1  // add images ip1 and ip2:
2  ImageProcessor ip1 = ... // target image I1
3  ImageProcessor ip2 = ... // source image I2
4  ...
5  ip1.copyBits(ip2, 0, 0, Blitter.ADD); // I1(u,v) ← I1(u,v) + I2(u,v)
6  // ip1 holds the result, ip2 is unchanged
7  ...
```

By this operation, the target image `ip1` is destructively modified, while the source image `ip2` remains unchanged. The constant `ADD` is one of several arithmetic transfer modes defined by the `Blitter` interface (see Table 4.3). In addition, `Blitter` defines (bitwise) logical operations, such as `OR` and `AND`.[9] For arithmetic operations, the `copyBits()` method limits the results to the admissible range of pixel values (of the target image). Also note that (except for target images of type `FloatProcessor`) the results are *not* rounded but truncated to integer values.

[9] See also Sec. 10 of the ImageJ Short Reference [5].

4.8.5 ImageJ Plugins Involving Multiple Images

ImageJ provides two types of plugin: a generic plugin (`PlugIn`), which can be run without any open image, and plugins of type `PlugInFilter`, which apply to a single image. In the latter case, the currently active image is passed as an object of type `ImageProcessor` to the plugin's `run()` method (see also Sec. 2.2.3).

If two or more images $I_1, I_2 \ldots I_k$ are to be combined by a plugin program, only a single image I_1 can be passed directly to the plugin's `run()` method, but not the additional images $I_2 \ldots I_k$. The usual solution is to make the plugin open a dialog window to let the user select the remaining images interactively. This is demonstrated in the following example plugin for transparently blending two images.

Example: alpha blending

Alpha blending is a simple method for transparently overlaying two images, I_{BG} and I_{FG}. The background image I_{BG} is covered by the foreground image I_{FG}, whose transparency is controlled by the value α in the form

$$I'(u,v) \leftarrow \alpha \cdot I_{\text{BG}}(u,v) + (1-\alpha) \cdot I_{\text{FG}}(u,v) \tag{4.41}$$

with $0 \le \alpha \le 1$. For $\alpha = 0$, the foreground image I_{FG} is nontransparent (opaque) and thus entirely hides the background image I_{BG}. Conversely, the image I_{FG} is fully transparent for $\alpha = 1$ and only I_{BG} is visible. All α values between 0 and 1 result in a weighted sum of the corresponding pixel values taken from I_{BG} and I_{FG} (Eqn. (4.41)).

Figure 4.23 shows the results of alpha blending for different α values. The Java code for the corresponding implementation (as an ImageJ plugin) is listed in Progs. 4.5 and 4.6. The background image (`bgIp`) is passed directly to the plugin's `run()` method. The second (foreground) image and the α value are specified interactively by creating an instance of the ImageJ class `GenericDialog`, which allows the simple implementation of dialog windows with various types of input fields.[10]

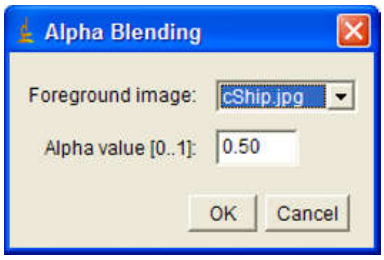
[10] See also Sec. 18.2 of the ImageJ Short Reference [5], where a similar example producing a *stack* of images by stepwise alpha blending can be found in Sec. 15.3.

I_{BG}

$\alpha = 0.25$

I_{FG}

$\alpha = 0.50$



`GenericDialog` window

$\alpha = 0.75$

Figure 4.23 Alpha blending example. Background image (I_{BG}) and foreground image (I_{FG}), `GenericDialog` window (see the implementation in Progs. 4.5 and 4.6), and blended images for transparency values $\alpha = 0.25, 0.50$, and 0.75.

```
1 import ij.IJ;
2 import ij.ImagePlus;
3 import ij.WindowManager;
4 import ij.gui.GenericDialog;
5 import ij.plugin.filter.PlugInFilter;
6 import ij.process.*;
7
8 public class Alpha_Blending implements PlugInFilter {
9
10   static double alpha = 0.5; // transparency of foreground image
11   ImagePlus fgIm = null;     // foreground image
12
13   public int setup(String arg, ImagePlus imp) {
14     return DOES_8G;
15   }
16
17   public void run(ImageProcessor bgIp) { // background image
18     if(runDialog()) {
19       ImageProcessor fgIp
20          = fgIm.getProcessor().convertToByte(false);
21       fgIp = fgIp.duplicate();
22       fgIp.multiply(1-alpha);
23       bgIp.multiply(alpha);
24       bgIp.copyBits(fgIp, 0, 0, Blitter.ADD);
25     }
26   }
27
28   // continued ...
```

Program 4.5 Alpha blending plugin (part 1). A background image is transparently blended with a selected foreground image. The plugin is applied to the (currently active) background image, and the foreground image must also be open when the plugin is started. The background image (`bgIp`), which is passed to the plugin's `run()` method, is multiplied with α (line 23). The foreground image (`fgIP`, selected in part 2) is first duplicated (line 21) and then multiplied with $(1-\alpha)$ (line 22). Thus the original foreground image is not modified. The final result is obtained by adding the two weighted images (line 24).

```
30  // class Alpha_Blending (continued)
31
32  boolean runDialog() {
33    // get list of open images
34    int[] windowList = WindowManager.getIDList();
35    if (windowList == null){
36      IJ.noImage();
37      return false;
38    }
39
40    // get all image titles
41    String[] windowTitles = new String[windowList.length];
42    for (int i = 0; i < windowList.length; i++) {
43      ImagePlus im = WindowManager.getImage(windowList[i]);
44      if (im == null)
45        windowTitles[i] = "untitled";
46      else
47        windowTitles[i] = im.getShortTitle();
48    }
49
50    // create dialog and show
51    GenericDialog gd = new GenericDialog("Alpha Blending");
52    gd.addChoice("Foreground image:",
53           windowTitles, windowTitles[0]);
54    gd.addNumericField("Alpha value [0..1]:", alpha, 2);
55    gd.showDialog();
56    if (gd.wasCanceled())
57      return false;
58    else {
59      int fgIdx = gd.getNextChoiceIndex();
60      fgIm = WindowManager.getImage(windowList[fgIdx]);
61      alpha = gd.getNextNumber();
62      return true;
63    }
64  }
65
66 } // end of class Alpha_Blending
```

Program 4.6 Alpha blending plugin (part 2). To select the foreground image, a list of currently open images and image titles is obtained (lines 34–42). Then a dialog object (`GenericDialog`) is created and opened for specifying the foreground image (`fgIm`) and the α value (`alpha`). `fgIm` and `alpha` are variables in the class `AlphaBlend_` (declared in part 1, Prog. 4.5). The `runDialog()` method returns `true` if successful and `false` if no images are open or the dialog was canceled by the user.

4.9 Exercises

Exercise 4.1
Implement the auto-contrast operation as defined in Eqns. (4.8)–(4.10) as an ImageJ plugin for an 8-bit grayscale image. Set the quantile q of pixels to be saturated at both ends of the intensity range (0 and 255) to $q = q_{\text{low}} = q_{\text{high}} = 1\%$.

Exercise 4.2
Modify the histogram equalization plugin in Prog. 4.2 to use a lookup table (Sec. 4.8.1) for computing the point operation.

Exercise 4.3
Implement the histogram equalization as defined in Eqn. (4.11), but use the *modified* cumulative histogram defined in Eqn. (4.12), cumulating the square root of the histogram entries. Compare the results to the standard (linear) approach by plotting the resulting histograms and cumulative histograms as shown in Fig. 4.10.

Exercise 4.4
Show formally that (a) a linear histogram equalization (Eqn. (4.11)) does not change an image that already has a uniform intensity distribution and (b) that any repeated application of histogram equalization to the same image causes no more changes.

Exercise 4.5
Show that the linear histogram equalization (Sec. 4.5) is only a special case of histogram specification (Sec. 4.6).

Exercise 4.6
Implement (in Java) a histogram specification using a piecewise linear reference distribution function, as described in Sec. 4.6.3. Define a new object class with all necessary instance variables to represent the distribution function and implement the required functions $\mathsf{P}_L(i)$ (Eqn. (4.22)) and $\mathsf{P}_L^{-1}(b)$ (Eqn. (4.23)) as methods of this class.

Exercise 4.7
Using a histogram specification for adjusting *multiple* images (Sec. 4.6.4), one could either use one typical image as the reference or compute an "average" reference histogram from a set of images. Implement the second approach and discuss its possible advantages (or disadvantages).

Exercise 4.8
Implement the modified gamma correction (Eqn. (4.33)) as an ImageJ plugin with variable values for γ and a_0 using a lookup table as shown in Prog. 4.4.

Exercise 4.9

Show that the modified gamma correction function $\bar{f}_{\gamma,a_0}(a)$, with the parameters defined in Eqns. (4.33) and (4.34), is C1-continuous (i. e., both the function itself and its first derivative are continuous).