CHAPTER 2

Image Pre-Processing

"I entered, and found Captain Nemo deep in algebraical calculations of x and other quantities."

—*Jules Verne,* 20,000 Leagues Under The Sea

This chapter describes the methods used to prepare images for further analysis, including interest point and feature extraction. Some of these methods are also useful for global and local feature description, particularly the metrics derived from transforms and basis spaces. The focus is on image pre-processing for computer vision, so we do not cover the entire range of image processing topics applied to areas such as computational photography and photo enhancements, so we refer the interested reader to various other standard resources in Digital Image Processing and Signal Processing as we go along [4,9,325,326], and we also point out interesting research papers that will enhance understanding of the topics.

Note Readers with a strong background in image processing may benefit from a light reading of this chapter.

Perspectives on Image Processing

Image processing is a vast field that cannot be covered in a single chapter. So why do we discuss image pre-processing in a book about computer vision? The reason is to advance the science of local and global feature description, as image pre-processing is typically ignored in discussions of feature description. Some general image processing topics are covered here in light of feature description, intended to illustrate rather than to proscribe, as applications and image data will guide the image pre-processing stage.

Some will argue that image pre-processing is not a good idea, since it distorts or changes the true nature of the raw data. However, intelligent use of image pre-processing can provide benefits and solve problems that ultimately lead to better local and global feature detection. We survey common methods for image enhancements and corrections that will affect feature analysis downstream in the vision pipeline in both favorable and unfavorable ways, depending on how the methods are employed.

Image pre-processing may have dramatic positive effects on the quality of feature extraction and the results of image analysis. Image pre-processing is analogous to the mathematical normalization of a data set, which is a common step in many feature descriptor methods. Or to make a musical analogy, think of image pre-processing as a sound system with a range of controls, such as raw sound with no volume controls; volume control with a simple tone knob; volume control plus treble, bass, and mid; or volume control plus a full graphics equalizer, effects processing, and great speakers in an acoustically superior room. In that way, this chapter promotes image pre-processing by describing a combination of corrections and enhancements that are an essential part of a computer vision pipeline.

Problems to Solve During Image Pre-Processing

In this section we suggest opportunities for image pre-processing that are guided according to the feature descriptor method you've chosen. Raw image data direct from a camera may have a variety of problems, as discussed in Chapter 1, and therefore it is not likely to produce the best computer vision results. This is why careful consideration of image pre-processing is fundamental. For example, a local binary descriptor using gray scale data will require different pre-processing than will a color SIFT algorithm; additionally, some exploratory work is required to fine-tune the image pre-processing stage for best results. We explore image pre-processing by following the vision pipelines of four fundamental families of feature description methods, with some examples, as follows:

1. **Local Binary Descriptors** (LBP, ORB, FREAK, others)
2. **Spectra Descriptors** (SIFT, SURF, others)
3. **Basis Space Descriptors** (FFT, wavelets, others)
4. **Polygon Shape Descriptors** (blob object area, perimeter, centroid)

These families of feature description metrics are developed into a taxonomy in Chapter 5. Before that, though, Chapter 4 discusses the feature descriptor building concepts, while Chapter 3 covers global feature description and then Chapter 6 surveys local feature description. The image pre-processing methods and applications introduced here are samples, but a more developed set of examples, following various vision pipelines, is developed in Chapter 8, including application-specific discussions of the pre-processing stage.

Vision Pipelines and Image Pre-Processing

Table 2-1 lists common image pre-processing operations, with examples from each of the four descriptor families, illustrating both differences and commonality among these image pre-processing steps, which can be applied prior to feature description. Our intent here is to illustrate rather than proscribe or limit the methods chosen.

Table 2-1. *Possible Image Pre-Processing Enhancements and Corrections as Applied to Different Vision Pipelines*

Image Pre-Processing	Local Binary (LBP, ORB)	Spectra (SIFT, SURF)	Basis Space (FFT, Code books)	Polygon Shape (Blob Metrics)
Illumination corrections	x	x	x	x
Blur and focus corrections	x	x	x	x
Filtering and noise removal	x	x	x	x
Thresholding				x
Edge enhancements		x		x
Morphology				x
Segmentation				x
Region processing and filters		x	x	x
Point processing		x		x
Math and statistical processing		x		x
Color space conversions		x	x	x

Local binary features deal with the pixel intensity comparisons of *point-pairs*. This makes the comparisons relatively insensitive to illumination, brightness, and contrast, so there may not be much need for image pre-processing to achieve good results. Current local binary pattern methods as described in the literature do not typically call for much image pre-processing; they rely on a simple comparison threshold that can be adjusted to account for illumination or contrast.

Spectra descriptors, such as SIFT (which acts on local region gradients) and SURF (which uses HAAR-like features with integrated pixel values over local regions), offer diverse pre-processing opportunities. Methods that use image pyramids often perform some image pre-processing on the image pyramid to create a scale space representation of the data using Gaussian filtering to smooth the higher levels of the pyramid. Basic illumination corrections and filtering may be useful to enhance the image prior to computing gradients—for example, to enhance the contrast within a band of intensities that likely contain gradient-edge information for the features. But in general, the literature does not report good or bad results for any specific methods used to pre-process the image data prior to feature extraction, and therein resides the opportunity.

Basis space features are usually global or regional, spanning a regular shaped polygon—for example, a Fourier transform computed over the entire image or block. However, basis space features may be part of the local features, such as the Fourier spectrum of the LBP histogram, which can be computed over histogram bin values of a local descriptor to provide rotational invariance. Another example is the Fourier descriptor used to compute polygon factors for radial line segment lengths showing the roundness of a feature to provide rotational invariance. See Chapter 3, especially Figure 3-19.

The most complex descriptor family is the polygon shape based descriptors, which potentially require several image pre-processing steps to isolate the polygon structure and shapes in the image for measurement. Polygon shape description pipelines may involve everything from image enhancements to structural morphology and segmentation techniques. Setting up the pre-processing for polygon feature shape extraction typically involves more work than any other method, since thresholds and segmentation require fine-tuning to achieve good results. Also note that polygon shape descriptors are not local patterns but, rather, larger regional structures with features spanning many tens and even hundreds of pixels, so the processing can be more intensive as well.

In some cases, image pre-processing is required to correct problems that would otherwise adversely affect feature description; we look at this next.

Corrections

During image pre-processing, there may be artifacts in the images that should be corrected prior to feature measurement and analysis. Here are various candidates for correction.

- **Sensor corrections.** Discussed in Chapter 1, these include dead pixel correction, geometric lens distortion, and vignetting.
- **Lighting corrections.** Lighting can introduce deep shadows that obscure local texture and structure; also, uneven lighting across the scene might skew results. Candidate correction methods include rank filtering, histogram equalization, and LUT remap.
- **Noise.** This comes in many forms, and may need special image pre-processing. There are many methods to choose from, some of which are surveyed in this chapter.
- **Geometric corrections.** If the entire scene is rotated or taken from the wrong perspective, it may be valuable to correct the geometry prior to feature description. Some features are more robust to geometric variation than others, as discussed in Chapters 4, 5, and 6.
- **Color corrections.** It can be helpful to redistribute color saturation or correct for illumination artifacts in the intensity channel. Typically color hue is one of the more difficult attributes to correct, and it may not be possible to correct using simple gamma curves and the sRGB color space. We cover more accurate colorimetry methods later in this chapter.

Enhancements

Enhancements are used to optimize for specific feature measurement methods, rather than fix problems. Familiar image processing enhancements include sharpening and color balancing. Here are some general examples of image enhancement with their potential benefits to feature description.

- **Scale-space pyramids.** When a pyramid is constructed using an octave scale and pixel decimation to sub-sample images to create the pyramid, sub-sampling artifacts and jagged pixel transitions are introduced. Part of the scale-space pyramid building process involves applying a Gaussian blur filter to the sub-sampled images, which removes the jagged artifacts.
- **Illumination.** In general, illumination can always be enhanced. Global illumination can be enhanced using simple LUT remapping and pixel point operations and histogram equalizations, and pixel remapping. Local illumination can be enhanced using gradient filters, local histogram equalization, and rank filters.
- **Blur and focus enhancements.** Many well-known filtering methods for sharpening and blurring may be employed at the pre-processing stage. For example, to compensate for pixel aliasing artifacts introduced by rotation that may manifest as blurred pixels which obscure fine detail, sharpen filters can be used to enhance the edge features prior to gradient computations. Or, conversely, the rotation artifacts may be too strong and can be removed by blurring.

In any case, the pre-processing enhancements or corrections are dependent on the descriptor using the images, and the application.

Preparing Images for Feature Extraction

Each family of feature description methods has different goals for the pre-processing stage of the pipeline. Let's look at a few examples from each family here, and examine possible image pre-processing methods for each.

Local Binary Family Pre-Processing

The local binary descriptor family is primarily concerned with point-pair intensity value comparisons, and several point-pair patterns are illustrated in Chapter 4 for common methods such as FREAK, BRISK, BRIEF, and ORB. As illustrated in Figure 2-4, the

comparative difference (<, >, =) between points is all that matters, so hardly any image pre-processing seems needed. Based on this discussion, here are two approaches for image pre-processing:

1. **Preserve pixels as is.** Do nothing except use a pixel value-difference compare threshold, such as done in the Census transform and other methods, since the threshold takes care of filtering noise and other artifacts.

   ```
   if(|point1-point2|>threshold)
   ```

2. **Use filtering.** In addition to using the compare threshold, apply a suitable filter to remove local noise, such as a smoothing or rank filter. Or, take the opposite approach and use a sharpen filter to amplify small differences, perhaps followed by a smoothing filter. Either method may prove to work, depending on the data and application.

Figure 2-1 uses center-neighbor point-pair comparisons in a 3x3 local region to illustrate the difference between local threshold and a pre-processing operation for the local binary pattern LBP, as follows:

- Left image: Original unprocessed local 3x3 region data; compare threshold = 5, dark pixels > 5 from center pixel.
- Left center image: Compare threshold = 10; note pattern shape is different simply by changing the threshold.
- Right center image: After a Laplacian sharpening filter is applied to 3x3 region, note that the center pixel value is changed from 52 to 49, so with the compare threshold set to 5 the pattern is now different from original on the left.
- Right image: Threshold on Laplacian filtered data set to 10; note different resulting binary pattern.

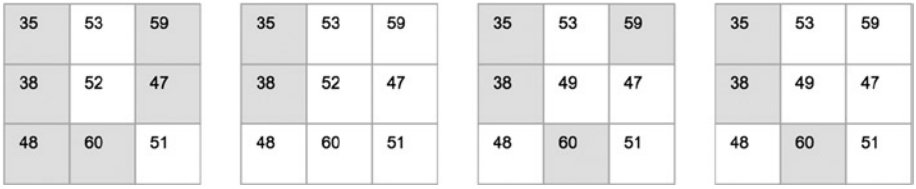


Figure 2-1. *How the LBP can be affected by pre-processing, showing the compare threshold value effects. (Left) Compare threshold = 5. (Center left) Compare threshold = 10. (Center right) Original data after Laplacian fitler applied. (Right) Compare threshold = 5 on Laplacian filtered data*

Spectra Family Pre-Processing

Due to the wide range of methods in the spectra category, it is difficult to generalize the potential pre-processing methods that may be useful. For example, SIFT is concerned with gradient information computed at each pixel. SURF is concerned with combinations of HAAR wavelets or local rectangular regions of integrated pixel values, which reduces the significance of individual pixel values.

For the integral image-based methods using HAAR-like features such as SURF and Viola Jones, here are a few hypothetical pre-processing options.

1. **Do nothing.** HAAR features are computed from integral images simply by summing local region pixel values; no fine structure in the local pixels is preserved in the sum, so one option is to do nothing for image pre-processing.

2. **Noise removal.** This does not seem to be needed in the HAAR pre-processing stage, since the integral image summing in local regions has a tendency to filter out noise.

3. **Illumination problems.** This may require pre-processing;for example, contrast enhancement may be a good idea if the illumination of the training data is different from the current frame. One pre-processing approach in this situation is to compute a global contrast metric for the images in the training set, and then compute the same global contrast metric in each frame and adjust the image contrast if the contrast diverges beyond a threshold to get closer to the desired global contrast metric. Methods for contrast enhancement include LUT remapping, global histogram equalization, and local adaptive histogram equalization.

4. **Blur.** If blur is a problem in the current frame, it may manifest similar to a local contrast problem, so local contrast enhancement may be needed, such as a sharpen filter. Computing a global statistical metric such as an SDM as part of the ground truth data to measure local or global contrast may be useful; if the current image diverges too much in contrast, a suitable contrast enhancement may be applied as a pre-processing step.

Note in Figure 2-2 that increasing the local-region contrast results in larger gradients and more apparent edges. A feature descriptor that relies on local gradient information is affected by the local contrast.

Figure 2-2. *The effects of local contrast on gradients and edge detection: (Left) Original image and Sobel edges. (Right) Contrasted adjusted image to amplify local region details and resulting Sobel edges*

For the SIFT-type descriptors that use local area gradients, pre-processing may be helpful to enhance the local gradients prior to computation, so as to affect certain features:

1. **Blur.** This will inhibit gradient magnitude computation and may make it difficult to determine gradient direction, so perhaps a local rank filter, high-pass filter, or sharpen filter should be employed.
2. **Noise.** This will exacerbate local gradient computations and make them unreliable, so perhaps applying one of several existing noise-removal algorithms can help.
3. **Contrast.** If local contrast is not high enough, gradient computations are difficult and unreliable. Perhaps a local histogram equalization, LUT remap, rank filter, or even a sharpen filter can be applied to improve results.

Basis Space Family Pre-Processing

It is not possible to generalize image pre-processing for basis space methods, since they are quite diverse, according to the taxonomy we are following in this work. As discussed in Chapters 4, 5, and 6, basis space methods include Fourier, wavelets, visual vocabularies, KTL, and others. However, here we provide a few general observations on pre-processing.

1. **Fourier Methods, wavelets, Slant transform, Walsh Hadamard, KLT.** These methods transform the data into another domain for analysis, and it is hard to suggest any pre-processing without knowing the intended application. For example, computing the Fourier spectrum produces magnitude and phase, and phase is shown to be useful in feature description to provide invariance to blur, as reported in the LPQ linear phase quantization method described in Chapter 6, so a blurry image may not be a problem in this case.

2. **Sparse coding and visual vocabularies.** These methods rely on local feature descriptors, which could be SURF, SIFT, LBP, or any other desired feature, derived from pixels in the spatial domain. Therefore, the method for feature description will determine the best approach for pre-processing. For example, methods that use correlation and raw pixel patches as sparse codes may not require any pre-processing. Or perhaps some minimal pre-processing can be used, such as illumination normalization to balance contrast, local histogram equalization or a LUT contrast remap.

In Figure 2-3, the contrast adjustment does not have much affect on Fourier methods, since there is no dominant structure in the image. Fourier spectrums typically reveal that the dominant structure and power is limited to lower frequencies that are in the center of the quadrant-shifted 2D plot. For images with dominant structures, such as lines and other shapes, the Fourier power spectrum will show the structure and perhaps pre-processing may be more valuable. Also, the Fourier power spectrum display is scaled to a logarithmic value and does not show all the details linearly, so a linear spectrum rendering might show the lower frequencies scaled and magnified better for erase of viewing.

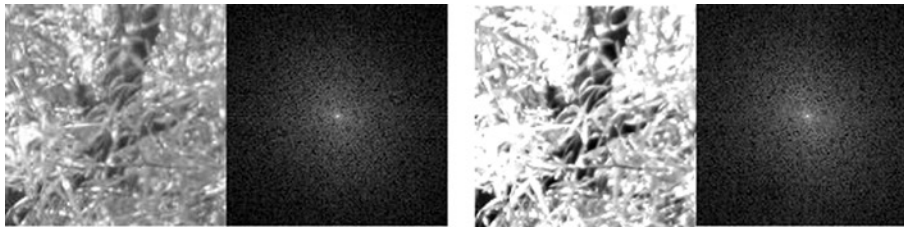


Figure 2-3. *In this example, no benefit is gained from pre-processing as shown in the Fourier spectrum; (Left) Before. (Right) After contrast adjusting the input image*

Polygon Shape Family Pre-Processing

Polygon shapes are potentially the most demanding features when considering image pre-processing steps, since as shown in Table 2-1, the range of potential pre-processing methods is quite large and the choice of methods to employ is very data-dependent. Possibly because of the challenges and intended use-cases for polygon shape measurements, they are used only in various niche applications, such as cell biology.

One of the most common methods employed for image preparation prior to polygon shape measurements is to physically correct the lighting and select the subject background. For example, in automated microscopy applications, slides containing cells are prepared with florescent dye to highlight features in the cells, then the illumination angle and position are carefully adjusted under magnification to provide a uniform background under each cell feature to be measured; the resulting images are then much easier to segment.

As illustrated in Figures 2-4 and 2-5, if the pre-processing is wrong, the resulting shape feature descriptors are not very useful. Here are some of the more salient options for pre-processing prior to shape based feature extraction, then we'll survey a range of other methods later in this chapter.

Figure 2-4. *Use of thresholding to solve problems during image pre-processing to prepare images for polygon shape measurement: (Left) Original image. (Center) Thresholded red channel image. (Right) Perimeter tracing above a threshold*

Figure 2-5. *Another sequence of morphological pre-processing steps preceding polygon shape measurement: (Left) Original image. (Center) Range thresholded and dilated red color channel. (Right) Morphological perimeter shapes taken above a threshold*

1. **Illumination corrections.** Typically critical for defining the shape and outline of binary features. For example, if perimeter tracking or boundary segmentation is based on edges or thresholds, uneven illumination will cause problems, since the boundary definition becomes indistinct. If the illumination cannot be corrected, then other segmentation methods not based on thresholds are available, such as texture-based segmentation.

2. **Blur and focus corrections.** Perhaps not as critical as illumination for polygon shape detection, since the segmentation of object boundary and shape is less sensitive to blur.

3. **Filtering and noise removal.** Shape detection is somewhat tolerant of noise, depending on the type of noise. Shot noise or spot noise may not present a problem, and is easily removed using various noise-cleaning methods.

4. **Thresholding.** This is critical for polygon shape detection methods. Many thresholding methods are employed, ranging from the simple binary thresholding to local adaptive thresholding methods discussed later in this chapter. Thresholding is a problematic operation and requires algorithm parameter fine-tuning in addition to careful control of the light source position and direction to deal with shadows.

5. **Edge enhancements.** May be useful for perimeter contour definition.

6. **Morphology.** One of the most common methods employed to prepare polygon shapes for measurement, covered later in this chapter in some detail. Morphology is used to alter the shapes, presumably for the better, mostly by combinations or pipelines of erosion and dilation operations, as shown in Figure 2-5. Morphological examples include object area boundary cleanup, spur removal, and general line and perimeter cleanup and smoothing.

7. **Segmentation.** These methods use structure or texture in the image, rather than threshold, as a basis for dividing an image into connected regions or polygons. A few common segmentation methods are surveyed later in this chapter.

8. **Area/Region processing.** Convolution filter masks such as sharpen or blur, as well as statistical filters such as rank filters or media filters, are potentially useful prior to segmentation.

9. **Point processing.** Arithmetic scaling of image data point by point, such as multiplying each pixel by a given value followed by a clipping operation, as well as LUT processing, often is useful prior to segmentation.

10. **Color space conversions.** Critical for dealing accurately with color features, covered later in this chapter.

As shown In Figure 2-4, a range thresholding method uses the red color channel, since the table background has a lot of red color and can be thresholded easily in red to remove the table top. The image is thresholded by clipping values outside an intensity band; note that the bottom right USB stick is gone after thresholding, since it is red and below the threshold. Also note that the bottom center white USB stick is also mostly gone, since it is white (max RGB values) and above the threshold. The right image shows an attempt to trace a perimeter above a threshold; it's still not very good, as more pre-processing steps are needed.

The Taxonomy of Image Processing Methods

Before we survey image pre-processing methods, it is useful to have a simple taxonomy to frame the discussion. The taxonomy suggested is a set of operations, including point, line, area, algorithmic, and data conversions, as illustrated in Figure 2-6. The basic categories of image pre-processing operations introduced in Figure 2-1 fit into this simple taxonomy. Note that each stage of the vision pipeline, depending on intended use, may have predominant tasks and corresponding pre-processing operations.

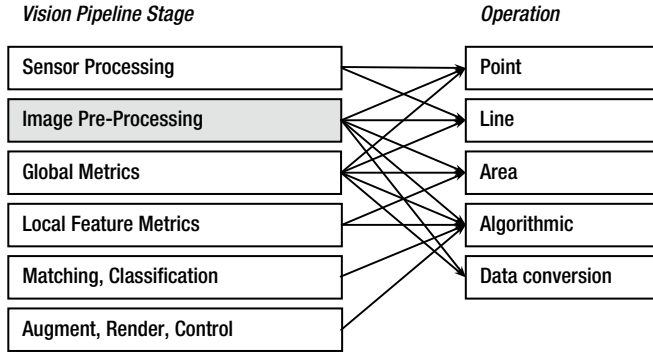


Figure 2-6. *Simplified, typical image processing taxonomy, as applied across the vision pipeline*

We provide a brief introduction to the taxonomy here, followed by a more detailed discussion in Chapter 5. Note that the taxonomy follows memory layout and memory access patterns for the image data. Memory layout particularly affects performance and power.

Point

Point operations deal with one pixel at a time, with no consideration of neighboring pixels. For example, point processing operations can be divided into math, Boolean, and pixel value compare substitution sections, as shown in Table 2-2 in the section later on "Point Filtering." Other point processing examples include color conversions and numeric data conversions.

Line

Line operations deal with discrete lines of pixels or data, with no regard to prior or subsequent lines. Examples include the FFT, which is a separable transform, where pixel lines and columns can be independently processed in parallel as 1D FFT line operations. If an algorithm requires lines of data, then optimizations for image pre-processing memory layout, pipelined read/write, and parallel processing can be made. Optimizations are covered in Chapter 8.

Area

Area operations typically require local blocks of pixels—for example, spatial filtering via kernel masks, convolution, morphology, and many other operations. Area operations generate specific types of memory traffic, and can be parallelized using fine-grained methods such as common shaders in graphics processors and coarse-grained thread methods.

Algorithmic

Some image pre-processing methods are purely serial or algorithmic code. It is difficult or even impossible to parallelize these blocks of code. In some cases, algorithmic blocks can be split into a few separate threads for coarse-grained parallelism or else pipelined, as discussed in Chapter 8.

Data Conversions

While the tasks are mundane and obvious, significant time can be spent doing simple data conversions. For example, integer sensor data may be converted to floating point for geometric computations or color space conversions. Data conversions are a significant part of image pre-processing in many cases. Example conversions include:

- Integer bit-depth conversions (8/16/32/64)
- Floating point conversions (single precision to double precision)
- Fixed point to integer or float
- Any combination of float to integer and vice versa
- Color conversions to and from various color spaces
- Conversion for basis space compute, such as integer to and from float for FFT

Design attention to data conversions and performance are in order and can provide a good return on investment, as discussed in Chapter 8.

Colorimetry

In this section, we provide a brief overview of color science to guide feature description, with attention to color accuracy, color spaces, and color conversions. If a feature descriptor is using color, then the color representation and processing should be carefully designed, accurate, and suited to the application. For example, in some applications it is possible to recognize an object using color alone, perhaps recognizing an automobile using its paint color, assuming that the vendor has chosen a unique paint color each year for each model. By combining color with another simple feature, such as shape, an effective descriptor can be devised.

Color Science is a well-understood field defined by international standards and amply described in the literature [249,250,251]. We list only a few resources here.

- The Rochester Institute of Technology's Munsel Color Science Laboratory is among the leading research institutions in the area or color science and imaging. It provides a wide range of resources and has strong ties to industry imaging giants such as Kodak, Xerox, and others.
- The International Commission on Illumination (CIE) provides standard illuminant data for a range of light sources as it pertains to color science, as well as standards for the well-known color spaces CIE XYZ, CIE Lab, and CIE Luv.
- The ICC International Color Consortium provides the ICC standard color profiles for imaging devices, as well as many other industry standards, including the sRGB color space for color displays.
- Proprietary color management systems, developed by industry leaders, include the Adobe CMM and Adobe RGB, Apple ColorSync, and HP ColorSmart; perhaps the most advanced is Microsoft's Windows Color System, which is based on Canon's earlier Kyuanos system using on CIECAM02.

Overview of Color Management Systems

A full-blown color management system may not be needed for a computer vision application, but the methods of color management are critical to understand when you are dealing with color. As illustrated in Figure 2-7, a color management system converts colors between the device color spaces, such as RGB or sRGB, to and from a *colorimetric color space,* such as CIE Luv, Lab, Jch, or Jab, so as to perform *color gamut mapping*. Since each device can reproduce color only within a specific gamut or color range, gamut mapping is required to convert the colors to the closest possible match, using the mathematical models of each color device.

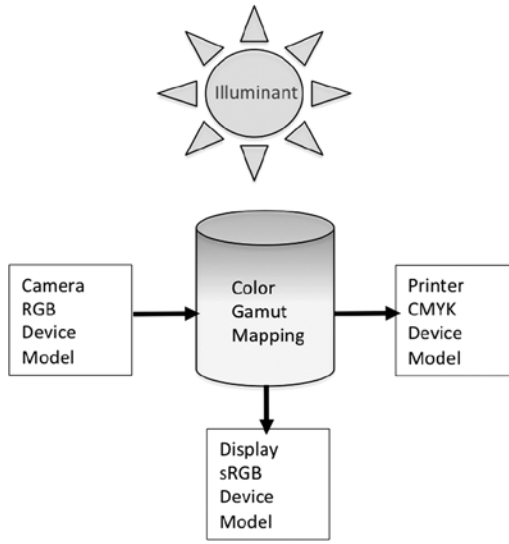


Figure 2-7. *Color management system with an RGB camera device model, sRGB display device model, CMYK printer device model, gamut mapping module, and an illuminant model*

Illuminants, White Point, Black Point, and Neutral Axis

An *illuminant* is a light source such as natural light or a fluorescent light, defined as the *white point* color by its spectral components and spectral power or color temperature. The white point color value in real systems is never perfectly white and is a measured quantity. The white point value and the oppositinal *black point* value together define the endpoints of the *neutral axis* (gray scale intensity) of the color space, which is not a perfectly straight color vector.

Color management relies on accurate information and measurements of the light source, or the illuminant. Color cannot be represented without accurate information about the light source under which the color is measured, since color appears different under florescent light versus natural light, and so on. The CIE standards define several values for standard illuminants, such as D65, shown in Figure 2-8.

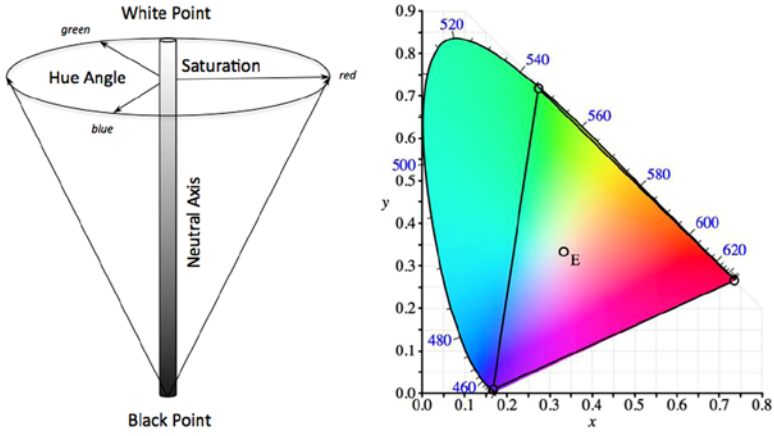


Figure 2-8. *(Left) Representation of a color space in three dimensions, neutral axis for the amount of white, hue angle for the primary color, and saturation for amount of color present. (Right) CIE XYZ chromaticity diagram showing values of the standard illuminant D65 OE as the white point, and the color primaries for R,G and B*

Device Color Models

Real devices like printers, displays, and cameras conventionally reproduce colors as compared against standard color patches that have been measured using calibrated light sources and spectrographic equipment—for example, the widely used Munsel color patches that define color in terms hue, value, and chroma (HVC) against standard illuminants. In order to effectively manage colors for a given device, a mathematical model or device color model must be created for each device, defining the anomalies in the device color gamut and its color gamut range.

For the color management system to be accurate, each real device must be spectrally characterized and modeled in a laboratory to create a mathematical device model, mapping the color gamut of each device against standard illumination models. The device model is used in the gamut transforms between color spaces.

Devices typically represent color using the primary and secondary colors RGB and CYMK. RGB is a primary, additive color space; starting with black, the RGB color primaries red, green, and blue are added to create colors. CYMK is a secondary color space, since the color components cyan, yellow, and magenta, are secondary combinations of the RGB primary colors; cyan = green plus blue, magenta = red plus blue, and yellow = red plus green. CYMK is also a subtractive color space, since the colors are subtracted from a white background to create specific colors.

Color Spaces and Color Perception

Colorimetric spaces represent color in abstract terms such as lightness, hue or color, and color saturation. Each color space is designed for a different reason, and each color space is useful for different types of analysis and processing. Example simple color spaces include HSV (hue, saturation, value) and HVC (hue, value, chroma). In the case of the CIE color spaces, the RGB color components are replaced by the standardized value CIE XYZ components as a basis for defining the CIE Luv and CIE Lab color spaces.

At the very high end of color science, we have the more recent CIECAM02 color models and color spaces such as Jch and Jab. CIECAM02 goes beyond just the colorimetry of the light source and the color patch itself to offer advanced color appearance modeling considerations that include the surroundings under which colors are measured [254,249].

While CIECAM02 may be overkill for most applications, it is worth some study. Color perception varies widely based on the surrounding against which the colors are viewed, the spectrum and angles of combined direct and ambient lighting, and the human visual system itself, since people do not all perceive color in the same way.

Gamut Mapping and Rendering Intent

Gamut mapping is the art and science of converting color between two color spaces and getting the best fit. Since the color gamuts of each device are different, gamut mapping is a challenge, and there are many different algorithms in use, with no clear winner. Depending on the intent of the rendering, different methods are useful—for example, gamut mapping from camera color space to a printer color space is different from mapping to an LCD display for viewing.

The CAM02 system provides a detailed model for guidance. For example, a color imaging device may capture the color blue very weakly, while a display may be able to display blue very well. Should the color gamut fitting method use color clipping or stretching? How should the difference between color gamuts be computed? Which color space? For an excellent survey of over 90 gamut mapping methods, see the work of Morovic [252].

In Figure 2-9 (left image), the sRGB color space is shown as fitting inside the Adobe RGB color space, illustrating that sRGB does not cover a gamut as wide as Adobe RGB. Each color gamut reproduces color differently, and each color space may be linear or warped internally. The right image in Figure 2-9 illustrates one gamut mapping method to determine the nearest color common to both color gamuts, using Euclidean distance and clipping; however, there are many other gamut mapping distance methods as well. Depending on the surrounding light and environment, color perception changes further complicating gamut mapping.

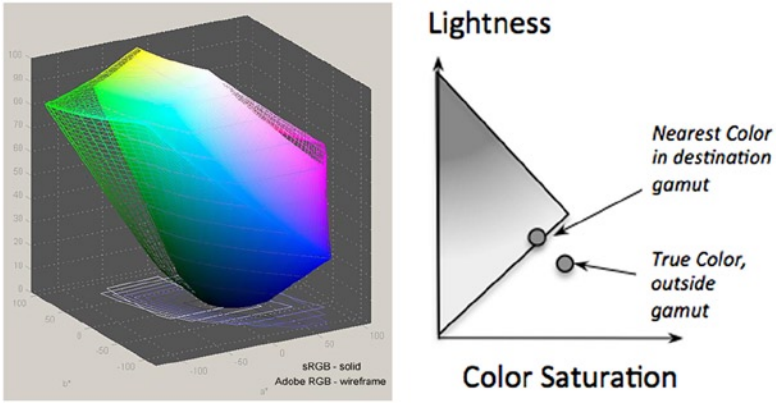


Figure 2-9. *The central problem of gamut mapping: (Left) Color sRGB and Adobe RGB color gamuts created using Gamutvision software. (Right) Gamut mapping details*

In gamut mapping there is a source gamut and a destination gamut. For example, the source could be a camera and the destination could be an LCD display. Depending on the rendering intent of the gamut conversion, different algorithms have been developed to convert color from source to destination gamuts. Using the *perceptual intent*, color saturation is mapped and kept within the boundaries of the destination gamut in an effort to preserve relative color strength; and out-of-gamut colors from the source are compressed into the destination gamut, which allows for a more reversible gamut map translation. Using the *colorimetric intent,* colors may be mapped straight across from source to destination gamut, and colors outside the destination gamut are simply clipped.

A common method of color correction is to rely on a simple gamma curve applied to the intensity channel to help the human eye better visualize the data, since the gamma curve brightens up the dark regions and compresses the light regions of the image, similar to the way the human visual system deals with light and dark regions. However, gamut correction bears no relationship to the true sensor data, so a calibrated, colorimetrically sound approach is recommended instead.

Practical Considerations for Color Enhancements

For image pre-processing, the color intensity is usually the only color information that should be enhanced, since the color intensity alone carries a lot of information and is commonly used. In addition, color processing cannot be easily done in RGB space while preserving relative color. For example, enhancing the RGB channels independently with a sharpen filter will lead to Moiré fringe artifacts when the RGB channels are recombined into a single rendering. So to sharpen the image, first *forward-convert* RGB to a color

space such as HSV or YIQ, then sharpen the V or Y component, and then *inverse-convert* back to RGB. For example, to correct illumination in color, standard image processing methods such as LUT remap or histogram equalization will work, provided they are performed in the intensity space.

As a practical matter, for quick color conversions to gray scale from RGB, here are a few methods. (1) The G color channel is a good proxy for gray scale information, since as shown in the sensor discussion in Chapter 1, the RB wavelengths in the spectrum overlap heavily into the G wavelengths. (2) Simple conversion from RGB into gray scale intensity I can be done by taking I = R+G+B / 3. (3) The YIQ color space, used in the NTSC television broadcast standards, provides a simple forward/backward method of color conversion between RGB and a gray scale component Y, as follows:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.9663 & 0.6210 \\ 1 & -0.2721 & -0.6474 \\ 1 & -1.1070 & 1.7046 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595716 & -0.274453 & -0.321263 \\ 0.211456 & -0.522591 & 0.311135 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Color Accuracy and Precision

If color accuracy is important, 8 bits per RGB color channel may not be enough. It is necessary to study the image sensor vendor's data sheets to understand how good the sensor really is. At the time of this writing, common image sensors are producing 10 to 14 bits of color information per RGB channel. Each color channel may have a different spectral response, as discussed in Chapter 1.

Typically, green is a good and fairly accurate color channel on most devices; red is usually good as well and may also have near infrared sensitivity if the IR filter is removed from the sensor; and blue is always a challenge since the blue wavelength can be hardest to capture in smaller silicon wells, which are close to the size of the blue wavelength, so the sensor vendor needs to pay special attention to blue ssnsing details.

Spatial Filtering

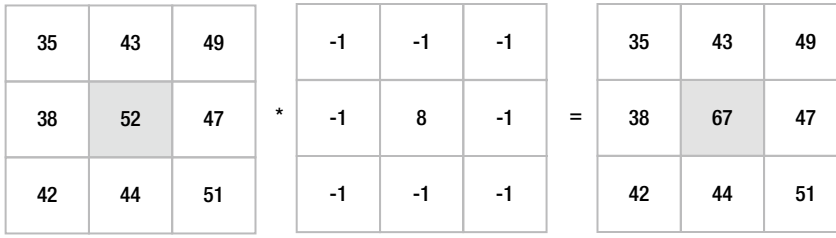
Filtering on discrete pixel arrays is considered *spatial filtering,* or time domain filtering, in contrast to filtering in the frequency domain using Fourier methods. Spatial filters are alternatives to frequency domain methods, and versatile processing methods are possible in the spatial domain.

Convolutional Filtering and Detection

Convolution is a fundamental signal processing operation easily computed as a discrete spatial processing operation, which is practical for 1D, 2D, and 3D processing. The basic idea is to combine, or convolve, two signals together, changing the source signal to be more like the filter signal. The source signal is the array of pixels in the image; the filter signal is a weighted kernel mask, such as a gradient peak shape and oriented edge shape or an otherwise weighted shape. For several examples of filter kernel mask shapes, see the section later in the chapter that discusses Sobel, Scharr, Prewitt, Roberts, Kirsch, Robinson, and Frei-Chen filter masks.

Convolution is typically used for filtering operations such as low-pass, band pass, and high-pass filters, but many filter shapes are possible to detect *f*eatures, such as edge detection kernels tuned sensitive to edge orientation, or even point, corner, and contour detectors. Convolution is used as a detector in the method of convolution networks [85], as discussed in Chapter 4.

The sharpen kernel mask in Figure 2-10 (center image) is intended to amplify the center pixel in relation to the neighboring pixels. Each pixel is multiplied by its kernel position, and the result (right image) shows the center pixel as the sum of the convolution, which has been increased or amplified in relation to the neighboring pixels.



-(35 + 43 + 49 + 47 + 51 + 44 + 42 + 38) + (52*8) = 67

Figure 2-10. *Convolution, in this case a sharpen filter: (Left to right) Image data, sharpen filter, and resulting image data*

A convolution operation is typically followed up with a set of postprocessing point operations to clean up the data. Following are some useful postprocessing steps; many more are suggested in the "Point Filters" section that follows later in the chapter.

```
switch (post_processor)
{
case RESULT_ASIS:
        break;
case RESULT_PLUS_VALUE:
        sum += value;
        break;
```

```
case RESULT_MINUS_VALUE:
        sum -= value;
        break;
case RESULT_PLUS_ORIGINAL_TIMES_VALUE:
        sum = sum + (result * value);
        break;
case RESULT_MINUS_ORIGINAL_TIMES_VALUE:
        sum = sum - (result * value);
        break;
case ORIGINAL_PLUS_RESULT_TIMES_VALUE:
        sum = result + (sum * value);
        break;
case ORIGINAL_MINUS_RESULT_TIMES_VALUE:
        sum = result - (sum * value);
        break;
case ORIGINAL_LOW_CLIP:
        sum = (result < value ? value : result);
        break;
case ORIGINAL_HIGH_CLIP:
        sum = (result > value ? value : result);
        break;
}

switch (post_processing_sign)
{
case ABSOLUTE_VALUE:
        if (sum < 0) sum = -sum;
        if (sum > limit) sum = limit;
        break;
case POSITIVE_ONLY:
        if (sum < 0) sum = 0;
        if (sum > limit) sum = limit;
        break;
case NEGATIVE_ONLY:
        if (sum > 0) sum = 0;
        if (-sum > limit) sum = -limit;
        break;
case SIGNED:
        if (sum > limit) sum = limit;
        if (-sum > limit) sum = -limit;
        break;
}
```

Convolution is used to implement a variety of common filters including:

- **Gradient or sharpen filters,** which amplify and detect maxima and minima pixels. Examples include Laplacian.
- **Edge or line detectors,** where lines are connected gradients that reveal line segments or contours. Edge or line detectors can be steerable to a specific orientation, like vertical, diagonal, horizontal, or omni-directional; steerable filters as basis sets are discussed in Chapter 3.
- **Smoothing and blur filters,** which take neighborhood pixels.

Kernel Filtering and Shape Selection

Besides convolutional methods, kernels can be devised to capture regions of pixels generically for statistical filtering operations, where the pixels in the region are sorted into a list from low to high value. For example, assuming a 3x3 kernel region, we can devise the following statistical filters:

```
sort(&kernel, &image, &coordinates, &sorted_list);

switch (filter_type)
case RANK_FILTER:
        // Pick highest pixel in the list, rank = 8 for a 3x3 kernel 0..8
        // Could also pick the lowest, middle, or other rank
        image[center_pixel] = sorted_list[rank];
        break;
case MEDIAN_FILTER:
        // Median value is kernel size / 2, (3x3=9)/2=4 in this case
        image[center_pixel] = sorted_list[median];
        break;
case MAJORITY_FILTER:
        // Find the pixel value that occurs most often, count sorted pixel values
        count(&sorted_list, &counted_list);
        image[center_pixel] = counted_list[0];
        break;
}
```

The rank filter is a simple and powerful method that sorts each pixel in the region and substitutes a pixel of desired rank for the center pixel, such as substitution of the highest pixel in the region for the center pixel, or the median value or the majority value.

Shape Selection or Forming Kernels

Any regional operation can benefit from shape selection kernels to select pixels from the region and exclude others. Shape selection, or forming, can be applied as a pre-processing step to any image pre-processing algorithm or to any feature extraction method. Shape selection kernels can be binary truth kernels to select which pixels from the source image are used as a group, or to mark pixels that should receive individual processing. Shape selection kernels, as shown in Figure 2-11, can be applied to local feature descriptors and detectors also; similar but sometimes more complex local region pixel selection methods are often used with local binary descriptor methods, as discussed in Chapter 4.

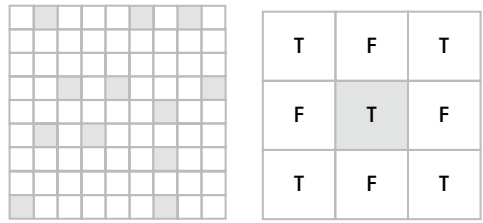


Figure 2-11. *Truth and shape kernels: (Left) A shape kernel gray kernel position indicating a pixel to process or use—for example, a pixel to convolve prior to a local binary pattern point-pair comparison detector.(Right) A truth shape kernel specifying pixels to use for region average, favoring diagonals—T means use this pixel, F means do not use*

Point Filtering

Individual pixel processing is typically overlooked when experimenting with image pre-processing. Point processing is amenable to many optimization methods, as will be discussed in Chapter 8. Convolution, as discussed above, is typically followed by point postprocessing steps. Table 2-2 illustrates several common pixel point processing methods in the areas of math operations, Boolean operations, and compare and substitution operations, which seem obvious but can be quite valuable for exploring image enhancement methods to enhance feature extraction.

Table 2-2. *Possible Point Operations*

// Math ops	// Compare & Substitution ops
NAMES math_ops[] = {	NAMES change_ops[] = {
"src + value -> dst",	"if (src = thresh) value -> dst",
"src - value -> dst",	"if (src = dst) value -> dst",
"src * value -> dst",	"if (src != thresh) value -> dst",
"src / value -> dst",	"if (src != thresh) src -> dst",
"(src + dst) * value -> dst",	"if (src != dst) value -> dst",
"(src - dst) * value -> dst",	"if (src != dst) src -> dst",
"(src * dst) * value -> dst",	"if (src >=thresh) value -> dst",
"(src / dst) * value -> dst",	"if (src >=thresh) src -> dst",
"sqroot(src) + value -> dst",	"if (src >=dst) value -> dst",
"src * src + value -> dst",	"if (src >=dst) src -> dst",
"exp(src) + value -> dst",	"if (src <= thresh) value -> dst",
"log(src) + value -> dst",	"if (src <= thresh) src -> dst",
"log10(src) + value -> dst",	"if (src <= dst) value -> dst",
"pow(src ^ value) -> dst",	"if (src <= dst) src -> dst",
"sin(src) + value -> dst",	"if (lo <= src <= hi) value -> dst",
"cos(src) + value -> dst",	"if (lo <= src <= hi) src -> dst",
"tan(src) + value -> dst",	};
"(value / max(all_src)) * src -> dst",	
"src - mean(all_src) -> dst",	
"absval(src) + value -> dst",	
};	
// **Boolean ops**	
NAMES bool_ops[] = {	
"src AND value -> dst",	
"src OR value -> dst",	
"src XOR value -> dst",	
"src AND dst -> dst",	
"src OR dst -> dst",	
"src XOR dst -> dst",	
"NOT(src) -> dst",	
"LO_CLIP(src, value) -> dst",	
"LO_CLIP(src, dst) -> dst",	
"HI_CLIP(src, value) -> dst",	
"HI_CLIP(src, dst) -> dst",	
};	

Noise and Artifact Filtering

Noise is usually an artifact of the image sensor, but not always. There are several additional artifacts that may be present in an image as well. The goal of noise removal is to remove the noise without distorting the underlying image, and the goal of removing artifacts is similar. Depending on the type of noise or artifact, different methods may be employed for pre-processing. The first step is to classify the noise or artifact, and then to devise the right image pre-processing strategy.

- **Speckle, random noise.** This type of noise is apparently random, and can be removed using a rank filter or median filter.
- **Transient frequency spike.** This can be determined using a Fourier spectrum and can be removed using a notch filter over the spike; the frequency spike will likely be in an outlier region of the spectrum, and may manifest as a bright spot in the image.
- **Jitter and judder line noise.** This is an artifact particular to video streams, usually due to telecine artifacts, motion of the camera or the image scene, and is complex to correct. It is primarily line oriented rather than just single-pixel oriented.
- **Motion blur.** This can be caused by uniform or nonuniform motion and is a complex problem; several methods exist for removal; see reference[305].

Standard approaches to noise removal are discussed by Gonzalez[4]. The most basic approach is to remove outliers, and various approaches are taken, including thresholding and local region based statistical filters such as the rank filter and median filter. Weighted image averaging is also sometime used for removing noise from video streams; assuming the camera and subjects are not moving, it can work well. Although deblurring or Gaussian smoothing convolution kernels are sometimes used to remove noise, such methods may cause smearing and may not be the best approach.

A survey of noise-removal methods and a performance comparison model are provided by Buades et al.[511]. This source includes a description of the author's NL-means method, which uses nonlocal pixel value statistics in addition to Euclidean distance metrics between similar weighted pixel values over larger image regions to identify and remove noise.

Integral Images and Box Filters

Integral images are used to quickly find the average value of a rectangular group of pixels. An integral image is also known as a *summed area table*, where each pixel in the integral image is the integral sum of all pixels to the left and above the current pixel. The integral image can be calculated quickly in a single pass over the image. Each value in the summed area table is calculated using the current pixel value from the image *i(n,m)* combined with previous entries *s(n,m)* made into the summed area table, as follows:

$$s(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$

As shown in Figure 2-12, to find a HAAR rectangle feature value from the integral image, only four points in the integral image table A,B,C,D are used, rather than tens or hundreds of points from the image. The integral image sum of a rectangle region can then be divided by the size of the rectangle region to yield the average value, which is also known as a *box filter*.

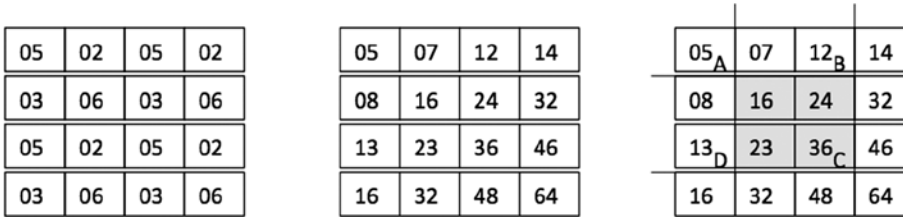


Figure 2-12. *(Left) Pixels in an image. (Center) Integral image. (Right) Region where a box filter value is computed from four points in the integral image: sum = s(A) + s(D) – s(B) – s(C)*

Integral images and box filters are used in many computer vision methods, such as HAAR filters and feature descriptors. Integral images are also used as a fast alternative to a Gaussian filter of a small region, as a way to lower compute costs. In fact, descriptors with a lot of overlapping region processing, such as BRISK [131], make effective use of integral images for descriptor building and use integral images as a proxy for a fast Gaussian blur or convolution.

Edge Detectors

The goal of an edge detector is to enhance the connected gradients in an image, which may take the form of an edge, contour, line, or some connected set of edges. Many edge detectors are simply implemented as kernel operations, or convolutions, and we survey the common methods here.

Kernel Sets: Sobel, Scharr, Prewitt, Roberts, Kirsch, Robinson, and Frei-Chen

The Sobel operator detects gradient magnitude and direction for edge detection. The basic method is shown here.

1. Perform two directional Sobel filters (x and y axis) using basic derivative kernel approximations such as 3x3 kernels, using values as follows:

2. $$S_y = . \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$S_x = . \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

3. Calculate the total gradient as $G_v = |S_x| + |S_y|$
4. Calculate the gradient direction as $theta = ATAN(S_y / S_x)$
5. Calculate gradient magnitude $G_m = \sqrt{\mathrm{Sy}^2 + \mathrm{Sx}^2}$

Variations exist in the area size and shape of the kernels used for Sobel edge detection. In addition to the Sobel kernels shown above, other similar kernel sets are used in practice, so long as the kernel values cancel and add up to zero, such as those kernels proposed by Scharr, Prewitt, Roberts, Robinson, and Frei-Chen, as well as Laplacian approximation kernels. The Frei-Chen kernels are designed to be used together at a set, so the edge is the weighted sum of all the kernels. See reference[4] for more information on edge detection masks. Some kernels have compass orientations, such as those developed by Kirsch, Robinson, and others. See Figure 2-13.

$$\begin{pmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{pmatrix} \begin{pmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{pmatrix} \text{Scharr}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \text{Roberts}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \text{Prewitt}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} .5 & 1 & .5 \\ 1 & -6 & 1 \\ .5 & 1 & .5 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{pmatrix} \text{Laplacians}$$

$$\begin{pmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix} \begin{pmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{pmatrix} \begin{pmatrix} 5 & -3 & 5 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{pmatrix} \begin{pmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{pmatrix} \text{Kirsch Compass}$$

$$\begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{pmatrix} \begin{pmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{pmatrix} \begin{pmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{pmatrix} \begin{pmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{pmatrix} \text{Kirsch Compass}$$

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{pmatrix} \text{Robinson Compass}$$

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -2 \end{pmatrix} \begin{pmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix} \text{Robinson Compass}$$

$$\frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{pmatrix}, \frac{1}{2\sqrt{2}} \begin{pmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{pmatrix}, \frac{1}{2\sqrt{2}} \begin{pmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{pmatrix} \text{Fre - Chen}$$

$$\frac{1}{2\sqrt{2}} \begin{pmatrix} \sqrt{2} & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & -\sqrt{2} \end{pmatrix}, \frac{1}{2} \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}, \frac{1}{2} \begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix} \text{Frei - Chen}$$

$$\frac{1}{6} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}, \frac{1}{6} \begin{pmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{pmatrix}, \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \text{Frei - CHen}$$

Figure 2-13. *Several edge detection kernel masks*

Canny Detector

The Canny method [154] is similar to the Sobel-style gradient magnitude and direction method, but it adds postprocessing to clean up the edges.

1. Perform a Gaussian blur over the image using a selected convolution kernel (7x7, 5,5, etc.), depending on the level of low-pass filtering desired.
2. Perform two directional Sobel filters (*x* & *y* axis).

3. Perform nonmaximal value suppression in the direction of the gradient to set to zero (0) pixels not on an edge (minima values).

4. Perform hysteresis thresholding within a band (high,low) of values along the gradient direction to eliminate edge aliasing and outlier artifacts and to create better connected edges.

Transform Filtering, Fourier, and Others

This section deals with basis spaces and image transforms in the context of image filtering, the most common and widely used being the Fourier transform. A more comprehensive treatment of basis spaces and transforms in the context of feature description is provided in Chapter 3. A good reference for transform filtering in the context of image processing is provided by Pratt [9].

Why use transforms to switch domains? To make image pre-processing easier or more effective, or to perform feature description and matching more efficiently. In some cases, there is no better way to enhance an image or describe a feature than by transforming it to another domain—for example, for removing noise and other structural artifacts as outlier frequency components of a Fourier spectrum, or to compact describe and encode image features using HAAR basis features.

Fourier Transform Family

The Fourier transform is very well known and covered in the standard reference by Bracewell [227], and it forms the basis for a family of related transforms. Several methods for performing fast Fourier transform (FFT) are common in image and signal processing libraries. Fourier analysis has touched nearly every area of world affairs, through science, finance, medicine, and industry, and has been hailed as "the most important numerical algorithm of our lifetime" [290]. Here, we discuss the fundamentals of Fourier analysis, and a few branches of the Fourier transform family with image pre-processing applications.

The Fourier transform can be computed using optics, at the speed of light [516]. However, we are interested in methods applicable to digital computers.

Fundamentals

The basic idea of Fourier analysis [227,4,9] is concerned with decomposing periodic functions into a series of sine and cosine waves (Figure 2-14). The Fourier transform is bi-directional, between a periodic wave and a corresponding series of harmonic basis functions in the frequency domain, where each basis function is a sine or cosine function, spaced at whole harmonic multiples from the base frequency. The result of the forward FFT is a complex number composed of magnitude and phase data for each sine and cosine component in the series, also referred to as *real data* and *imaginary data*.

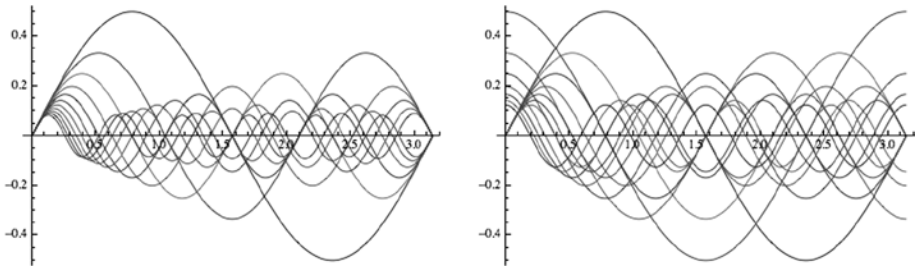


Figure 2-14. *(Left) Harmonic series of sine waves. (Right) Fourier harmonic series of sine and cosine waves*

Arbitrary periodic functions can be synthesized by summing the desired set of Fourier basis functions, and periodic functions can be decomposed using the Fourier transform into the basic functions as a Fourier series. The Fourier transform is invertible between the time domain of discrete pixels and the frequency domain, where both magnitude and phase of each basis function are available for filtering and analysis, magnitude being the most commonly used component.

How is the FFT implemented for 2D images or 3D volumes? The Fourier transform is a *separable transform* and so can be implemented as a set of parallel 1D FFT line transforms (Figure 2-15). So, for 2D images and 3D volumes, each dimension, such as the *x, y, z* dimension, can be computed in place, in parallel as independent *x* lines, then the next dimension or *y* columns can be computed in place as parallel lines, then the *z* dimension can be computed as parallel lines in place, and the final results are scaled according to the transform. Any good 1D FFT algorithm can be set up to process 2D images or 3D volumes using parallelization.

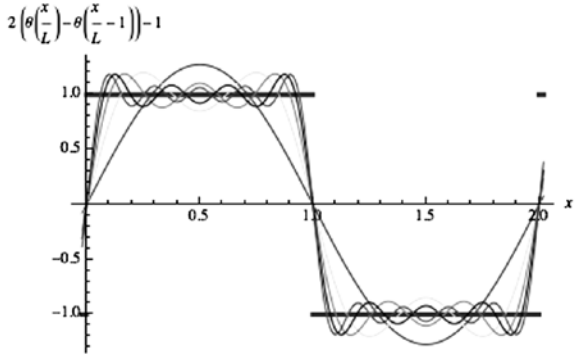


Figure 2-15. *Fourier series and Fourier transform concepts showing a square wave approximated from a series of Fourier harmonics*

For accuracy of the inverse transform to go from frequency space back to pixels, the FFT computations will require two double precision 64-bit floating point buffers to hold the magnitude and phase data, since transcendental functions such as sine and cosine require high floating point precision for accuracy; using 64-bit double precision floating point numbers for the image data allows a forward transform of an image to be computed, followed by an inverse transform, with no loss of precision compared to the original image—of course, very large images will need more than double precision.

Since 64-bit floating point is typically slower and of higher power, owing to the increased compute requirements and silicon real estate in the ALU, as well as the heavier memory bandwidth load, methods for FFT optimization have been developed using integer transforms, and in some cases fixed point, and these are good choices for many applications.

Note in Figure 2-16 that the low-pass filter (center right) is applied to preserve primarily low-frequency information toward the center of the plot and it reduces high-frequency components toward the edges, resulting in the filtered image at the far right.

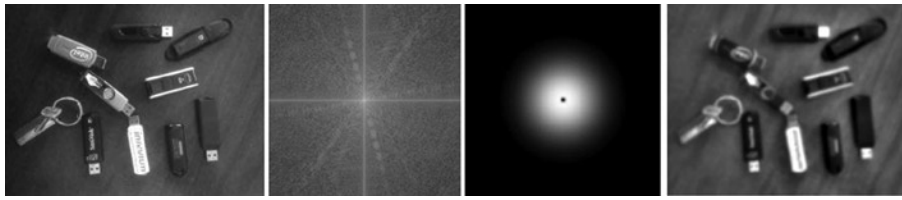


Figure 2-16. *Basic Fourier filtering: (Left) Original. (Center left) Fourier spectrum. (Center right) Low-pass filter shape used to multiply against Fourier magnitude. (Right) Inverse transformed image with low-pass filter*

A key Fourier application is filtering, where the original image is forward-transformed into magnitude and phase; the magnitude component is shown as a Fourier power spectrum of the magnitude data, which reveals structure in the image as straight lines and blocks, or outlier structures or spots that are typically noise. The magnitude can be filtered by various filter shapes, such as high-pass, low-pass, band pass, and spot filters to remove spot noise, to affect any part of the spectrum.

In Figure 2-16, a circular symmetric low-pass filter shape is shown with a smooth distribution of filter coefficients from 1 to 0, with high multiplicands in the center at the low frequencies, ramping down to zero toward the high frequencies at the edge. The filter shape is multiplied in the frequency domain against the magnitude data to filter out the higher frequency components, which are toward the outside of the spectrum plot, followed by an inverse FFT to provide the filtered image. The low-frequency components are toward the center; typically these are most interesting and so most of the image power is contained in the low-frequency components. Any other filter shape can be used, such as a spot filter, to remove noise or any of the structure at a specific location of the spectrum.

Fourier Family of Transforms

The Fourier transform is the basis for a family of transforms [4], some of which are:

1. **DFT, FFT.** The discrete version of the Fourier transform, often implemented as a fast version, or FFT, commonly used for image processing. There are many methods of implementing the FFT [227].
2. **Sine transform.** Fourier formulation composed of only sine terms.
3. **Cosine transform.** Fourier formulation composed of only cosine terms.
4. **DCT**, **DST, MDCT.** The discrete Fourier transform is implemented in several formulations: discrete sine transform (DST), discrete cosine transform (DCT), and the modified discrete cosine transform (MDCT). These related methods operate on a macroblock, such as 16x16 or 8x8 pixel region, and can therefore be highly optimized for compute use with integers rather than floating point. Typically the DCT is implemented in hardware for video encode and decode applications for motion estimation of the macro blocks from frame to frame. The MDCT operates on overlapping macroblock regions for compute efficiency.
5. **Fast Hartley transform, DHT.** This was developed as an alternative formulation of the Fourier transform for telephone transmission analysis about 1925, forgotten for many years, then rediscovered and promoted again by Bracewell[227] as an alternative to the Fourier transform. The Hartley transform is a symmetrical formulation of the Fourier transform, decomposing a signal into two sets of sinusoidal functions taken together as a c*osine-*a*nd-*s*ine* or *cas( )* function, where $cas(vx) \equiv cos(vx) + sin(vx)$. This includes positive and negative frequency components and operates entirely on real numbers for input and output. The Hartley formulation avoids complex numbers as used in the Fourier complex exponential $exp(jwx)$. The Hartley tansform has been developed into optimized versions called the DHT, shown to be about equal in speed to an optimized FFT.

Other Transforms

Several other transforms may be used for image filtering, including wavelets, steerable filter banks, and others that will be described in Chapter 3, in the context of feature description. Note that transforms often have many common uses and applications that overlap, such as image description, image coding, image compression, and feature description.

Morphology and Segmentation

For simplicity, we define the goal of morphology as shape and boundary definition, and the goal of segmentation is to define regions with internal similarity, such as textural or statistical similarity. *Morphology* is used to identify features as polygon shaped regions that can be described with shape metrics, as will be discussed in Chapters 3 and 6, distinct from local interest point and feature descriptors using other methods. An image is *segmented into regions* to allow independent processing and analysis of each region according to some policy or processing goal. Regions cover an area smaller than the global image but usually larger than local interest point features, so an application might make use of global, regional, and small local interest point metrics together as an *object signature*.

An excellent review of several segmentation methods can be found in work by Haralick and Shapiro[321]. In practice, segmentation and morphology are not easy: results are often less useful than expected, trial and error is required, too many methods are available to provide any strict guidance, and each image is different. So here we only survey the various methods to introduce the topic and illustrate the complexity. An overview of region segmentation methods is shown in Table 2-3.

Table 2-3. *Segmentation Methods*

Method	Description
Morphological segmentation	The region is defined based on thresholding and morphology operators.
Texture-based segmentation	The texture of a region is used to group like textures into connected regions.
Transform-based segmentation	Basis space features are used to segment the image.
Edge boundary segmentation	Gradients or edges alone are used to define the boundaries of the region with edge linking in some cases to form boundaries.
Color segmentation	Color information is used to define regions.
Super-Pixel Segmentation	Kernels and distance transforms are used to group pixels and change their values to a common value.
Gray scale / luminance segmentation	Gray scale thresholds or bands are used to define the regions.
Depth segmentation	Depth maps and distance from viewer is used to segment the image into foreground, background, or other gradations of inter-scene features.

Binary Morphology

Binary morphology operates on binary images, which are created from other scalar intensity channel images. Morphology [9] is used to *morph* a feature shape into a new shape for analysis by removing shape noise or outliers, and by strengthening predominant feature characteristics. For example, isolated pixels may be removed using morphology, thin features can be fattened, and the predominant shape is still preserved. Note that morphology all by itself is quite a large field of study, with applications to general object recognition, cell biology, medicine, particle analysis, and automated microscopy. We introduce the fundamental concepts of morphology here for binary images, and then follow this section with applications to gray scale and color data.

Binary morphology starts with binarizing images, so typically thresholding is first done to create images with binary-valued pixels composed of 8-bit black and white values, 0-value = black and 255-value = white. Thresholding methods are surveyed later in this chapter, and thresholding is critical prior to morphology.

Binary morphology is a neighborhood operation, and can use a forming kernel with truth values, as shown in Figure 2-17. The forming kernel guides the morphology process by defining which surrounding pixels contribute to the morphology. Figure 2-17 shows two forming kernels: kernel a, where all pixels touching the current pixel are considered, and kernel b, where only orthogonally adjacent pixels are considered.

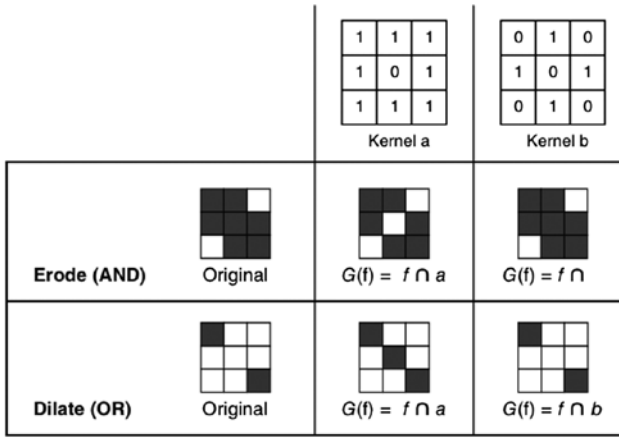


Figure 2-17. *3x3 forming kernels and binary erosion and dilation using the kernels; other kernel sizes and data values may be useful in a given application. (Image used by permission, © Intel Press, from Building Intelligent Systems)*

The basic operations of morphology include Boolean AND, OR, NOT. The notation used for the fundamental morphological operations is for *dilation* and for *erosion*. In binary morphology, dilation is a Boolean OR operator, while erosion is a Boolean AND operator. In the example provided in Figure 2-17, only kernel elements with a "1" are used in the morphology calculation, allowing for neighborhood contribution variations. For erosion, the pixels under all true forming kernel elements are AND'd together; the result is 1 if all are true and the pixel feature remains, otherwise the pixel feature is eroded or set to 0.

All pixels under the forming true kernel must be true for erosion of the center pixel. Erosion attempts to reduce sparse features until only strong features are left. Dilation attempts to inflate sparse features to make them fatter, only 1 pixel under the forming kernel elements must be true for dilation of the center pixel, corresponding to Boolean OR.

Based on simple erosion and dilation, a range of morphological operations are derived as shown here, where ⊕ = dilation and ⊖ = erosion.

Erode	$G(f) = f \ominus b$
Dilate	$G(f) = f \oplus b$
Opening	$G(f) = (f \oplus b) \ominus b$
Closing	$G(f) = (f \ominus b) \oplus b$
Morphological Gradient	$G(f) = f \ominus b$ or $G(f) = f \oplus b - f \ominus b$
Morphological Internal gradient	$G\,i(f) = f - f \ominus b$
Morphological External gradient	$G\,e(f) = f \oplus b - f$

Gray Scale and Color Morphology

Gray scale morphology is useful to synthesize and combine pixels into homogeneous intensity bands or regions with similar intensity values. Gray scale morphology can be used on individual color components to provide color morphology affecting hue, saturation, and color intensity in various color spaces.

For gray scale morphology or color morphology, the basic operations are MIN, MAX, and MINMAX, where pixels above the MIN are changed to the same value and pixels below the MAX are changed to the same value, while pixels within the MINMAX range are changed to the same value. MIN and MAX are a form of thresholding, while MINMAX allows bands of pixel values to be coalesced into equal values forming a homogenous region.

Morphology Optimizations and Refinements

Besides simple morphology [9], there are other methods of morphological segmentation using adaptive methods [254,255,256]. The simple morphology methods rely on using a fixed kernel across the entire image at each pixel and assume the threshold is already applied to the image; while the adaptive methods combine the morphology operations with variable kernels and variable thresholds based on the local pixel intensity statistics. This allows the morphology to adapt to the local region intensity and, in some cases, produce better results. Auto-thresholding and adaptive thresholding methods are discussed later in this chapter and are illustrated in Figures 2-24 and 2-26.

Euclidean Distance Maps

The distance map, or Euclidean distance map (EDM), converts each pixel in a binary image into the distance from each pixel to the nearest background pixel, so the EDM requires a binary image for input. The EDM is useful for segmentation, as shown in Figure 2-18, where the EDM image is thresholded based on the EDM values—in this case, similar to the ERODE operator.

Figure 2-18. *Pre-processing sequence: (Left) Image after thresholding and erosion. (Center) EDM showing gray levels corresponding to distance of pixel to black background. (Right) Simple binary thresholded EDM image*

Super-Pixel Segmentation

A super-pixel segmentation method [257,258,259,260,261] attempts to collapse similar pixels in a local region into a larger super-pixel region of equal pixel value, so similar values are subsumed into the larger super-pixel. Super-pixel methods are commonly used for digital photography applications to create a scaled or watercolor special effect. Super-pixel methods treat each pixel as a node in a graph, and edges between regions are determined based on the similarity of neighboring pixels and graph distance. See Figure 2-19.

Figure 2-19. *Comparison of various super-pixel segmentation methods (Image © Dr. Radhakrishna Achanta, used by permission)*

Feature descriptors may be devised based on super-pixels, including super-pixel value histograms, shape factors of each polygon shaped super-pixel, and spatial relationships of neighboring super-pixel values. Apparently little work has been done on super-pixel based descriptors; however, the potential for several degrees of robustness and invariance seems good. We survey a range of super-pixel segmentation methods next.

Graph-based Super-Pixel Methods

Graph-based methods structure pixels into trees based on the distance of the pixel from a centroid feature or edge feature for a region of like-valued pixels. The compute complexity varies depending on the method.

- **SLIC Method** [258] Simple Linear Iterative Clusting (SLIC) creates super-pixels based on a 5D space, including the CIE Lab color primaries and the XY pixel coordinates. The SLIC algorithm takes as input the desired number of super-pixels to generate and adapt well to both gray scale and RGB color images. The clustering distance function is related to the size of the desired number of super-pixels and uses a Euclidean distance function for grouping pixels into super-pixels.
- **Normalized Cuts** [262,263] Uses a recursive region partitioning method based on local texture and region contours to create super-pixel regions.
- **GS-FH Method** [264] The graph-based Felzenszwalb and Huttenlocher method attempts to segment image regions using edges based on perceptual or psychological cues. This method uses the minimum length between pixels in the graph tree structure to create the super-pixel regions. The computational complexity is *O(n Log n),* which is relatively fast.
- **SL Method** [265] The Super-Pixel Lattice (SL) method finds region boundaries within tiled image regions or strips of pixels using the graph cut method.

Gradient-Ascent-Based Super-Pixel Methods

Gradient ascent methods iteratively refine the super-pixel clusters to optimize the segmentation until convergence criteria are reached. These methods use a tree graph structure to associate pixels together according to some criteria, which in this case may be the RGB values o*r* Cartesian coordinates of the pixels, and then a distance function or other function is applied to create regions. Since these are iterative methods, the performance can be slow.

- **Mean-Shift** [266] Works by registering off of the region centroid based on a kernel-based mean smoothing approach to create regions of like pixels.
- **Quick-Shift** [267] Similar to the mean-shift method but does not use a mean blur kernel and instead uses a distance function calculated from the graph structure based on RGB values and XY pixel coordinates.

- **Watershed** [268] Starts from local region pixel value minima points to find pixel value-based contour lines defining watersheds, or basin contours inside which similar pixel values can be substituted to create a homogeneous pixel value region.
- **Turbopixels** [269] Uses small circular seed points placed in a uniform grid across the image around which super-pixels are collected into assigned regions, and then the super-pixel boundaries are gradually expanded into the unassigned region, using a geometric flow method to expand the boundaries using controlled boundary value expansion criteria, so as to gather more pixels together into regions with fairly smooth and uniform geometric shape and size.

Depth Segmentation

Depth information, such as a depth map as shown in Figure 2-20, is ideal for segmenting objects based on distance. Depth maps can be computed from a wide variety of depth sensors and methods, including a single camera, as discussed in Chapter 1. Depth cameras, such as the Microsoft Kinect camera, are becoming more common. A depth map is a 2D image or array, where each pixel value is the distance or *Z* value.

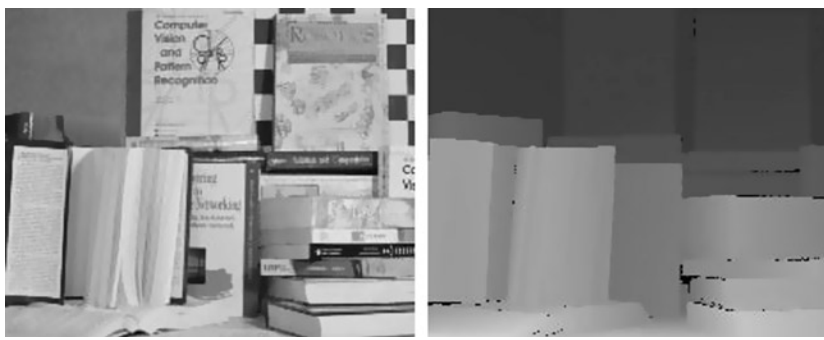


Figure 2-20. *Depth images from Middlebury Data set: (Left) Original image. (Right) Corresponding depth image. Data courtesy of Daniel Scharstein and used by permission*

Many uncertainties in computer vision arise out of the problems in locating three-dimensional objects in a two-dimensional image array, so adding a depth map to the vision pipeline is a great asset. Using depth maps, images can be easily segmented into the foreground and background, as well as be able to segment specific features or objects—for example, segmenting by simple depth thresholding.

Depth maps are often very fuzzy and noisy, depending on the depth sensing method, so image pre-processing may be required. However, there is no perfect filtering method for depth map cleanup. Many practitioners prefer the bi-lateral filter [302] and variants, since it preserves local structure and does a better job of handling the edge transitions.

Color Segmentation

Sometime color alone can be used to segment and threshold. Using the right color component can easily filter out features from an image. For example, in Figure 2-6, we started from a red channel image from an RGB set, and the goal was to segment out the USB sticks from the table background. Since the table is brown and contains a lot of red, the red channel provides useful contrast with the USB sticks allowing segmentation via red. It may be necessary to color-correct the image to get the best results, such as gamut corrections or boosting the hue or saturation of each color to accentuate difference.

Thresholding

The goal of thresholding is to segment the image at certain intensity levels to reveal features such as foreground, background, and specific objects. A variety of methods exist for thresholding, ranging from global to locally adaptive. In practice, thresholding is very difficult and often not satisfactory by itself, and must be tuned for the dataset and combined with other pre-processing methods in the vision pipeline.

One of the key problems in thresholding is nonuniform illumination, so applications that require thresholding, like cell biology and microscopy, pay special attention to cell preparation, specimen spacing, and light placement. Since many images do not respond well to global thresholding involving simple methods, local methods are often required, which use the local pixel structure and statistical relationships to create effective thresholds. Both global and local adaptive methods for thresholding are discussed here. A threshold can take several forms:

- **Floor** Lowest pixel intensity allowed
- **Ceiling** Highest pixel intensity allowed
- **Ramp** Shape of the pixel ramp between floor and ceiling, such as linear or log
- **Point** May be a binary threshold point with no floor, ceiling, or ramp

Global Thresholding

Thresholding entire images at a globally determined thresholding level is sometimes a good place to start to explore the image data, but typically local features will suffer and be unintelligible as a result. Thresholding can be improved using statistical methods to determine the best threshold levels. Lookup tables (LUT) can be constructed, guided by statistical moments to create the floor, ceiling, and ramps and the functions to perform rapid LUT processing on images, or false-color the images for visualization.

Histogram Peaks and Valleys, and Hysteresis Thresholds

Again we turn to the old stand-by, the image histogram. Peaks and valleys in the histogram may indicate thresholds useful for segmentation and thresholding [319]. A hysteresis region marks pixels with similar values, and is easy to spot in the histogram, as shown in Figure 2-21. Also, many image processing programs have interactive sliders to allow the threshold point and even regions to be set with the pointer device.[1] Take some time and get to know the image data via the histogram and become familiar with using interactive thresholding methods.

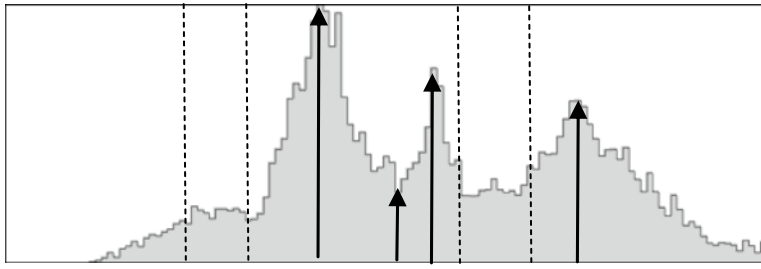


Figure 2-21. *Histogram annotated with arrows showing peaks and valleys, and dotted lines showing regions of similar intensities defined using hysteresis thresholds*

If there are no clear valleys between the histogram peaks, then establishing two thresholds, one on each side of the valley, is a way to define a region of hysteresis. Pixel values within the hysteresis region are considered inside the object. Further, the pixels can be classified together as a region using the hysteresis range and morphology to ensure region connectivity.

LUT Transforms, Contrast Remapping

Simple lookup tables (LUTs) are very effective for contrast remapping and global thresholding, and interactive tools can be used to create the LUTs. Once the interactive experimentation has been used to find the best floor, ceiling, and ramp function, the LUTs can be generated into table data structures and used to set the thresholds in fast code. False-coloring the image using pseudo-color LUTs is common and quite valuable for understanding the thresholds in the data. Various LUT shapes and ramps can be devised. See Figure 2-22 for an example using a linear ramp function.

[1]See the open-source package ImageJ2, and menu item Image ➤ Adjust-Brightness/Contrast for interactive thresholding.

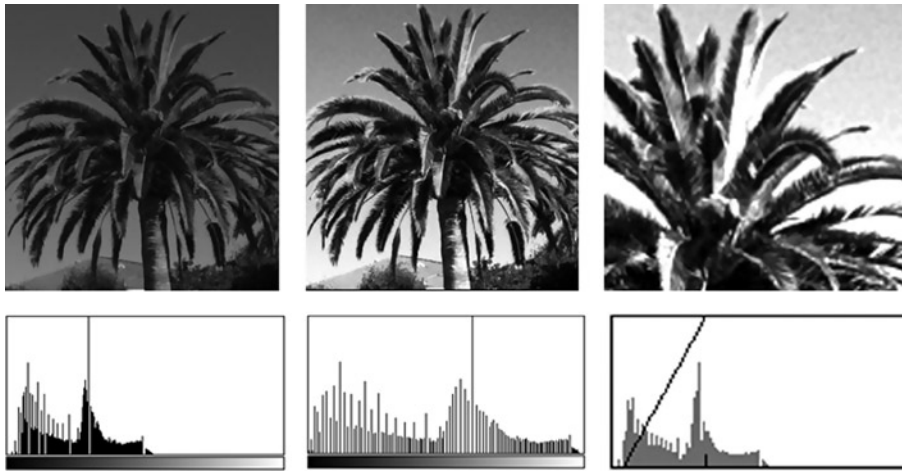


Figure 2-22. *Contrast corrections: (Left) Original image shows palm frond detail compressed into a narrow intensity range obscuring details. (Center) Global histogram equalization restores some detail. (Right) LUT remap function spreads the intensity values to a narrower range to reveal details of the palm fronds. The section of the histogram under the diagonal line is stretched to cover the full intensity range in the right image; other intensity regions are clipped. The contrast corrected image will yield more gradient information when processed with a gradient operator such as Sobel*

Histogram Equalization and Specification

Histogram equalization spreads pixel values between a floor and ceiling using a contrast remapping function, with the goal of creating a histogram with approximately equal bin counts approaching a straight-line distribution. See Figure 2-23. While this method works well for gray scale images, color images should be equalized in the intensity channel of a chosen color space, such as HSV V. Equalizing each RGB component separately and rerendering will produce color moiré artifacts. Histogram equalization uses a fixed region and a fixed remapping for all pixels in the region; however, adaptive local histogram equalization methods are available [314].

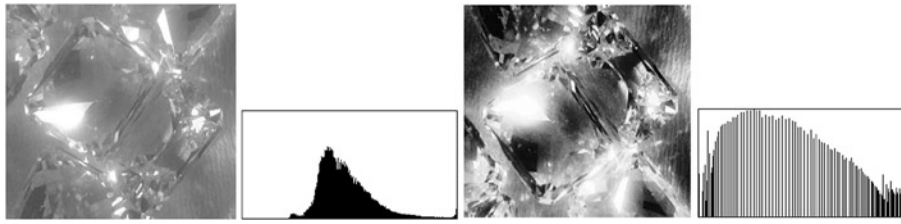


Figure 2-23. *(Left) Original image and histogram. (Right) Histogram equalized image and histogram*

It is possible to create a desired histogram shape or value distribution, referred to as *histogram specification*, and then remap all pixel values from the source image to conform to the specified histogram shape. The shape may be created directly, or else the histogram shape from a second image may be used to remap the source image to match the second image. With some image processing packages, the histogram specification may be interactive, and points on a curve may be placed and adjusted to create the desired histogram shape.

Global Auto Thresholding

Various methods have been devised to automatically find global thresholds based on statistical properties of the image histogram [320,513,514,515] and in most cases the results are not very good unless some image pre-processing precedes the auto thresholding. Table 2-4 provides a brief survey of auto thresholding methods, while Figure 2-24 displays renderings of each method.

Table 2-4. *Selected Few Global Auto-Thresholding Methods Derived from Basic Histogram Features [303]*

Method	Description
Default	A variation of the IsoData method, also knowm as iterative intermeans.
Huang	Huang's method of using fuzzy thresholding.
Intermodes	Iterative histogram smoothing.
IsoData	Iterative pixel averaging of values above and below a threshold to derive a new threshold above the composite average.
Li	Iterative cross-entropy thresholding.
MaxEntropy	Kapur-Sahoo-Wong (Maximum Entropy) algorithm.
Mean	Uses mean gray level as the threshold.
MinError	Iterative method from Kittler and Illingworth to converge on a minimum error threshold.
Minimum	Iterative histogram smoothing, assuming a bimodal histogram.
Moments	Tsai's thresholding algorithm intending to threshold and preserve the original image moments.
Otsu	Otsu clustering algorithms to set local thresholds by minimizing variance.
Percentile	Adapts the threshold based on pre-set allocations for foreground and background pixels.
RenyiEntropy	Another entropy-based method.
Shanbhag	Uses fuzzy set metrics to set the threshold.
Triangle	Uses image histogram peak, assumes peak is not centered, sets threshold in largest region on either side of peak.

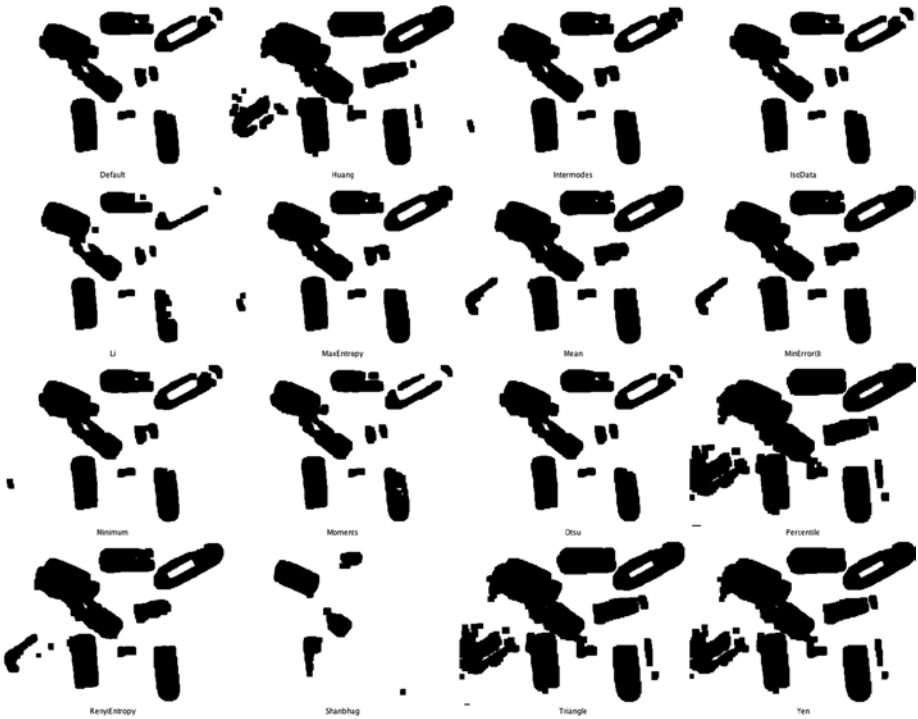


Figure 2-24. *Renderings of selected auto-thresholding methods (Images generated using ImageJ auto threshold plug-ins [303])*

Local Thresholding

Local thresholding methods take input from the local pixel region and threshold each pixel separately. Here are some common and useful methods.

Local Histogram Equalization

Local histogram equalization divides the image into small blocks, such as 32x32 pixels, and computes a histogram for each block, then rerenders each block using histogram equalization. However, the contrast results may contain block artifacts corresponding to the chosen histogram block size. There are several variations for local histogram equalization, including Contrast Limited Adaptive Local Histogram Equalization (CLAHE) [304].

Integral Image Contrast Filters

A histogram-related method uses integral images to compute local region statistics without the need to compute a histogram, then pixels are remapped accordingly, which is faster and achieves a similar effect (Figure 2-25).

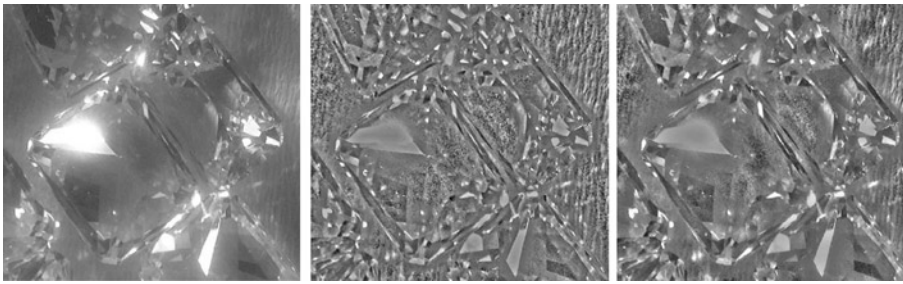


Figure 2-25. *Integral image filter from ImageJ to remap contrast in local regions, similar to histogram equalization: (Left) Original. (Center) 20x20 regions. (Right) 40x40 regions*

Local Auto Threshold Methods

Local thresholding adapts the threshold based on the immediate area surrounding each target pixel in the image, so local thresholding is more like a standard area operation or filter [513,514,515]. Local auto thresholding methods are available in standard software packages.[2] Figure 2-26 provides some example adaptive local thresholding methods, summarized in Table 2-5.

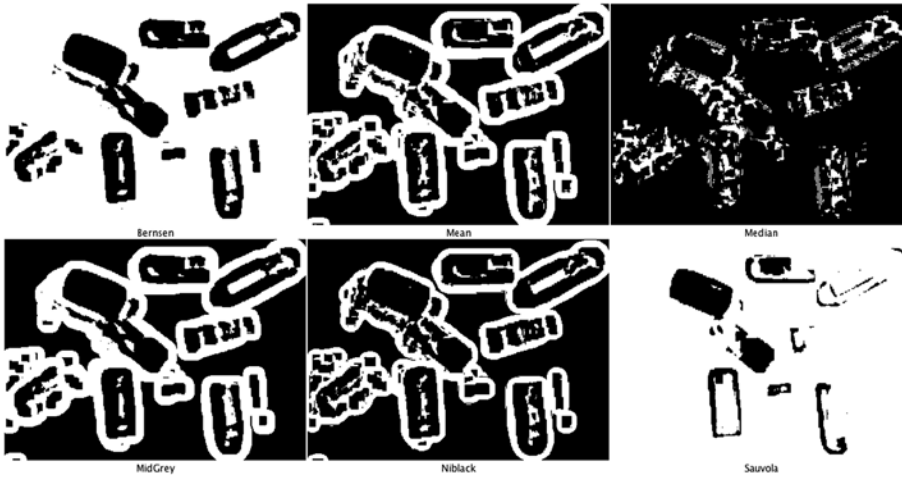


Figure 2-26. *Renderings of a selected few local auto and local thresholding methods using ImageJ plug-ins [303]*

[2]See the open-source package Imagej2, menu item Image ➤ Adjust ➤ Auto Local Threshold | Auto Threshold.

Table 2-5. *Selected Few Local Auto-thresholding Methods [303]*

Method	Description
Bernsen	Bernsen's algorithm using circular windows instead of rectangles and local midgray values
Mean	Uses the local gray level mean as the threshold
Median	Uses the local gray level mean as the threshold
MidGrey	Uses the local area gray level mean - C (where C is a constant)
Niblack	Niblack's algorithm is: p = (p > mean + k * standard_deviation - c) ? object : background
Sauvola	Sauvola's variation of Niblack: p = (p > mean * (1 + k *(standard_deviation / r - 1))) ? object : background

Summary

In this chapter, we surveyed image processing as a pre-processing step that can improve image analysis and feature extraction. We developed a taxonomy of image processing methods to frame the discussion, and applied the taxonomy to examples in the four fundamental vision pipelines, as will be developed in the taxonomy of Chapter 5, including (1) local binary descriptors such as LBP, ORB, FREAK; (2) spectra descriptors such as SIFT, SURF; (3) basis space descriptors such as FFT, wavelets; and (4) polygon shape descriptors such as blob object area, perimeter, and centroid. Common problems and opportunities for image pre-processing were discussed. Starting with illumination, noise, and artifact removal, we covered a range of topics including segmentation variations such as depth segmentation and super-pixel methods, binary, gray scale and color morphology, spatial filtering for convolutions and statistical area filters, and basis space filtering.