CCS349 IMAGE AND VIDEO ANALYTICS

COURSE OBJECTIVES:

- To understand the basics of image processing techniques for computer vision.
- To learn the techniques used for image pre-processing.
- To discuss the various object detection techniques.
- To understand the various Object recognition mechanisms.
- To elaborate on the video analytics techniques.

UNIT I INTRODUCTION 6

Computer Vision – Image representation and image analysis tasks - Image representations – digitization – properties – color images – Data structures for Image Analysis - Levels of image data representation - Traditional and Hierarchical image data structures.

UNIT II IMAGE PRE-PROCESSING 6

Local pre-processing - Image smoothing - Edge detectors - Zero-crossings of the second derivative - Scale in image processing - Canny edge detection - Parametric edge models - Edges in multispectral images - Local pre-processing in the frequency domain - Line detection by local preprocessing operators - Image restoration.

UNIT III OBJECT DETECTION USING MACHINE LEARNING 6

Object detection– Object detection methods – Deep Learning framework for Object detection– bounding box approach-Intersection over Union (IoU) –Deep Learning Architectures-R-CNN-Faster R-CNN-You Only Look Once(YOLO)-Salient features-Loss Functions-YOLO architectures

UNIT IV FACE RECOGNITION AND GESTURE RECOGNITION 6

Face Recognition-Introduction-Applications of Face Recognition-Process of Face Recognition- DeepFace solution by Facebook-FaceNet for Face Recognition- Implementation using FaceNetGesture Recognition.

UNIT V VIDEO ANALYTICS 6

Video Processing – use cases of video analytics-Vanishing Gradient and exploding gradient problem - RestNet architecture-RestNet and skip connections-Inception Network-GoogleNet architecture Improvement in Inception v2-Video analytics-RestNet and Inception v3.

30 PERIOD

LIST OF EXERCISES

1. Write a program that computes the T-pyramid of an image.
2. Write a program that derives the quad tree representation of an image using the homogeneity criterion of equal intensity
3. Develop programs for the following geometric transforms: (a) Rotation (b) Change of scale (c) Skewing (d) Affine transform calculated from three pairs of corresponding points (e) Bilinear transform calculated from four pairs of corresponding points.
4. Develop a program to implement Object Detection and Recognition
5. Develop a program for motion analysis using moving edges, and apply it to your image sequences.
6. Develop a program for Facial Detection and Recognition
7. Write a program for event detection in video surveillance system

TOTAL: 60 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO1: Understand the basics of image processing techniques for computer vision and video analysis.

CO2: Explain the techniques used for image pre-processing.

CO3: Develop various object detection techniques.

CO4: Understand the various face recognition mechanisms.

CO5: Elaborate on deep learning-based video analytics.

TEXT BOOK:

1. Milan Sonka, Vaclav Hlavac, Roger Boyle, "Image Processing, Analysis, and Machine Vision", 4nd edition, Thomson Learning, 2013.

2. Vaibhav Verdhan, 2021, Computer Vision Using Deep Learning Neural Network Architectures with Python and Keras, Apress 2021(UNIT-III,IV and V)

REFERENCES

1. Richard Szeliski, "Computer Vision: Algorithms and Applications", Springer Verlag London Limited,2011.
2. Caifeng Shan, FatihPorikli, Tao Xiang, Shaogang Gong, "Video Analytics for Business Intelligence", Springer, 2012.
3. D. A. Forsyth, J. Ponce, "Computer Vision: A Modern Approach", Pearson Education, 2003.
4. E. R. Davies, (2012), "Computer & Machine Vision", Fourth Edition, Academic Press.

CCS349 IMAGE AND VIDEO ANALYTICS

UNIT I : INTRODUCTION

Computer Vision – Image representation and image analysis tasks - Image representations – digitization – properties – color images – Data structures for Image Analysis - Levels of image data representation - Traditional and Hierarchical image data structures.

1.1. INTRODUCTION:

Vision allows humans to perceive and understand the world surrounding them, while computer vision aims to duplicate the effect of human vision by electronically perceiving and understanding an image.

Giving computers the ability to see is not an easy task—we live in a three-dimensional (3D) world, and when computers try to analyze objects in 3D space, the visual sensors available (e.g., TV cameras) usually give two-dimensional (2D) images, and this projection to a lower number of dimensions incurs an enormous loss of information. Sometimes, equipment will deliver images that are 3D but this may be of questionable value: analyzing such datasets is clearly more complicated than 2D, and sometimes the
'three-dimensionality' is less than intuitive to us . . . terahertz scans are an example of this. Dynamic scenes such as those to which we are accustomed, with moving objects or a moving camera, are increasingly common and represent another way of making computer vision more complicated.

Figure 1.1 could be witnessed in thousands of farmyards in many countries, and serves to illustrate just some of the problems that we will face.

Figure1.1 : A frame from a video of a typical farm- yard scene: the cow is one of a number walking naturally from right to left.

There are many reasons why we might wish to study scenes such as this, which are attractively simple *to us*. The beast is moving slowly, it is clearly black and white, its movement is rhythmic, etc.; however, automated analysis is very troubled—in fact, the animal's boundary is often very difficult to distinguish clearly

from the background, the motion of the legs is self-occluding and (subtly) the concept of ‘cow-shaped’ is not something easily encoded. The application from which this picture was taken made use of many of the algorithms: starting at a low level, moving features were identified and grouped. A ‘training phase’ taught the system what a cow might look like in various poses (see Figure 1.2), from which a model of a ‘moving’ cow could be derived (see Figure 1.3).

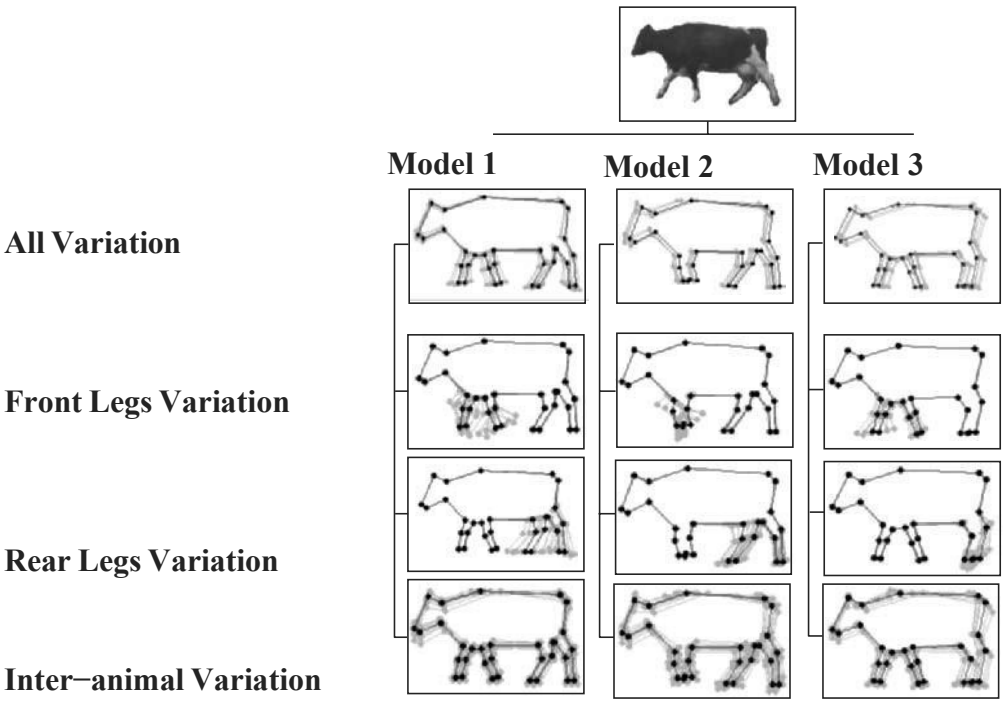


Figure 1.2: Various models for a cow Shadow: a straight-line boundary approximation has been learned from training data and is able to adapt to different animals and different forms of occlusion.

These models could then be fitted to new (‘unseen’) video sequences. Crudely, at this stage anomalous behavior such as lameness could be detected by the model failing to fit properly, or well.

Thus we see a sequence of operations—image capture, early processing, segmentation, model fitting, motion prediction, qualitative/quantitative conclusion—that is characteristic of image understanding and computer vision problems. Each of these phases (which may not occur sequentially!) may be addressed by a number of algorithms which we

shall cover in due course.

The application was serious; there is a growing need in modern agriculture for automatic monitoring of animal health, for example to spot lameness. A limping cow is trivial for a human to identify, but it is very challenging to do this automatically.

Figure 1.3: Three frames from a cow sequence: notice the model can cope with partial occlusion as the animal enters the scene, and the different poses exhibited.

This example is relatively simple to explain, but serves to illustrate that many computer vision techniques use the results and methods of mathematics, pattern recognition, artificial intelligence (AI), psycho-physiology, computer science, electronics, and other scientific disciplines.

1.2. COMPUTER VISION:

Computer vision is a **field of artificial intelligence (AI) enabling computers to derive information** from images, videos and other inputs.

Why is computer vision difficult?

Six complex landscape of computer vision are.

1) **Loss of information in 3D→ ❼ 2D** is a phenomenon which occurs in typical image capture devices such as a camera or an eye. Their geometric properties have been approximated by a pinhole model for centuries (a box with a small hole in it—a

'camera obscura' in Latin). This physical model corresponds to a mathematical model of perspective projection; Figure 1.4 summarizes the principle. The projective transformation maps points along rays but does not preserve angles and collinearity.

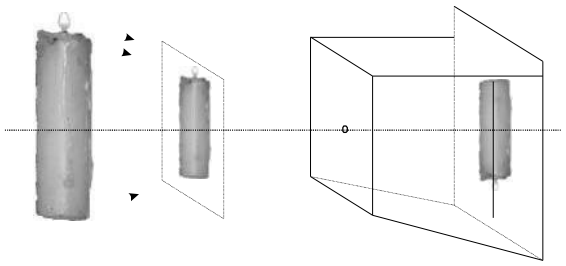


Figure 1.4: The pinhole model of imaging geometry does not distinguish size of objects.

The main trouble with the pinhole model and a single available view is that the projective transformation sees a small object close to the camera in the same way as a big object remote from the camera. In this case, a human needs a 'yardstick' to guess the actual size of the object which the computer does not have. 2)

The interpretation of images, which is unconsciously solved by humans, serves as the fundamental basis for computer vision. When a human tries to understand an image then previous knowledge and experience is brought to the current observation. Human ability to reason allows representation of long-gathered knowledge, and its use to solve new problems. Artificial intelligence has worked for decades to endow computers with the capability to understand observations; while progress has been tremendous, the practical ability of a machine to understand observations remains very limited.

From the mathematical logic and/or linguistics point of view, image interpretation can be seen as a mapping

$$\textit{interpretation: image data} \longrightarrow \textit{model}\,.$$

The (logical) model means some specific world in which the observed objects make sense. Examples might be nuclei of cells in a biological sample, rivers in a satellite image, or parts in an industrial process being checked for quality. There may be several interpretations of the same image(s). Introducing interpretation to computer vision allows us to use concepts from mathematical logic, linguistics as syntax (rules describing correctly formed expressions), and semantics (study of meaning). Considering observations (images) as an instance of formal expressions, semantics studies relations between expressions and their meanings. The interpretation of image(s) in computer vision can be understood as an instance of semantics.

Practically, if the image understanding algorithms know into which particular domain the observed world is constrained, then automatic analysis can be used for complicated problems.

3) **Noise** is inherently present in each measurement in the real world. Its existence calls for mathematical tools which are able to cope with uncertainty; an example is probability theory. Of course, more complex tools make the image analysis much more complicated compared to standard (deterministic) methods.
4) **Too much data.** Images are big, and video—increasingly the subject of vision applications–correspondingly bigger. Technical advances make processor and memory requirements much less of a problem than they once were, and much can be achieved with consumer level products. Nevertheless, efficiency in problem solutions is still important and many applications remain short of real-time performance.
5) **Brightness measured** in images is given by complicated image formation physics.

   The radiance (brightness, image intensity) depends on the irradiance≈ **(light source type, intensity and position), the observer's position, the surface local geometry, and the surface reflectance properties.** The inverse tasks are ill-posed—for example, to reconstruct local surface orientation from intensity variations. For this reason, imagecapture physics is usually avoided in practical attempts at image understanding. Instead, a direct link between the appearance of objects in scenes and their interpretation is sought.
6) **Local window vs. need for global view.** Commonly, image analysis algorithms analyze a particular storage bin in an operational memory (e.g., a pixel in the image) and its local neighborhood; the computer sees the image through a keyhole; this makes it very difficult to understand more global context. This problem has a long tradition in artificial intelligence: in the 1980s McCarthy argued that formalizing context was a crucial step toward the solution of the problem of generality. It is often very difficult to interpret an image if it is seen only locally or if only a few local keyholes are available. Figure 1.5 illustrates this pictorially. How context is taken into account is an important facet of image analysis.

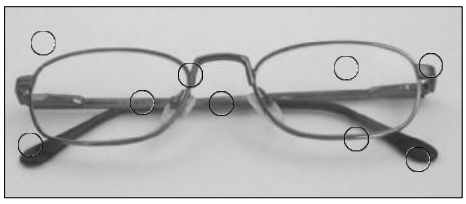


Figure 1.5: Illustration of the world seen through several keyholes providing only a local context.

It is very difficult to guess what object is depicted; the complete image is shown in Figure 1.6.

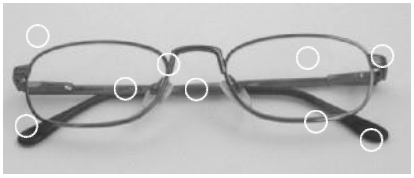


Figure 1.6: It is easy for humans to interpret an image if it is seen globally: compare to Figure 1.5.

1.3 IMAGE REPRESENTATION AND IMAGE ANALYSIS TASKS:

Image understanding by a machine can be seen as an attempt to find a relation between input image(s) and previously established models of the observed world. Transition from the input image(s) to the model reduces the information contained in the image to relevant information for the application domain. This process is usually divided into several steps and several levels representing the image are used.

The bottom layer contains raw image data and the higher levels interpret the data. Computer vision designs these intermediate representations and algorithms serving to establish and maintain relations between entities within and between layers.

Image representation can be roughly divided according to data organization into four levels.

The boundaries between individual levels are inexact, and more detailed divisions are also proposed in the literature. Figure 1.7 suggests a bottom up approach, from signals with almost no abstraction, to the highly abstract description needed for image understanding. The flow of information does not need to be unidirectional; often feedback loops are introduced which allow the modification of algorithms according to intermediate results.

This hierarchy of image representation and related algorithms is frequently categorized in an even simpler way—*low-level* image processing and *high-level* image understanding.

Low-level processing methods usually use very little knowledge about the content of images. In the case of the computer knowing image content, it is usually provided by high-level algorithms or directly by a human who understands the problem domain.

Low- level methods may include image compression, pre-processing methods for noise filtering, edge extraction, and image sharpening. Low- level image processing uses data which resemble the input image; for example, an input image captured by a TV camera is 2D in nature, being described by an image function $f(x, y)$ whose value is usually brightness depending on the co-ordinates x, y of the location in the image.

If the image is to be processed using a computer it will be digitized first, after which it may be represented by a rectangular matrix with elements corresponding to the brightness at appropriate image locations.

More probably, it will be presented in color, usually three channels: red, green and blue. Very often, such a data set will be part of a video stream with an associated frame rate. Nevertheless, the raw material will be a set or sequence of matrices which represent the inputs and outputs of low-level image processing.

High-level processing is based on knowledge, goals, and plans of how to achieve those goals, and artificial intelligence methods are widely applicable. High-level computer vision tries to imitate human cognition and the ability to make decisions according to the information contained in the image.

In the example described, high-level knowledge would be related to the 'shape' of a cow and the subtle interrelationships between the different parts of that shape, and their (inter-) dynamics.

High-level vision begins with some form of formal model of the world, and then the 'reality' perceived in the form of digitized images is compared to the model. A match is attempted, and when differences emerge, partial matches (or subgoals) are sought that overcome them; the computer switches to low-level image processing to find information needed to update the model. This process is then repeated iteratively, and 'understanding' an image thereby becomes a co-operation between top-down and bottom-up processes. A feedback loop is introduced in which high-level partial results create tasks for low-level image processing, and the iterative image understanding process should eventually converge to the global goal.

Computer vision is expected to solve very complex tasks, the goal being to obtain similar results to those provided by biological systems. To illustrate the complexity of these tasks, consider Figure 1.8 in which a particular image representation is presented— the value on the vertical axis gives the brightness of its corresponding location in the [gray-scale] image. Consider what this image might be before looking at Figure 1.9, which is a rather more common representation of the same image.

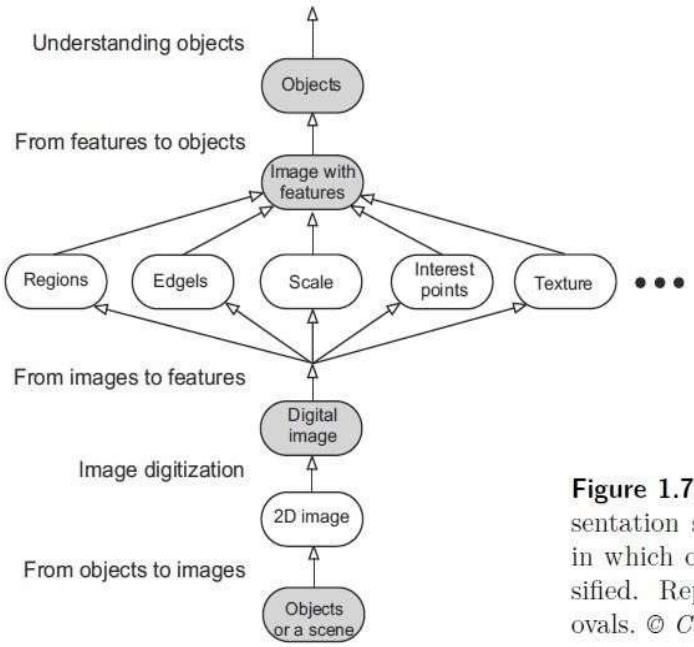


Figure 1.7: Four possible levels of image representation suitable for image analysis problems in which objects have to be detected and classified. Representations are depicted as shaded ovals. © *Cengage Learning 2015.*

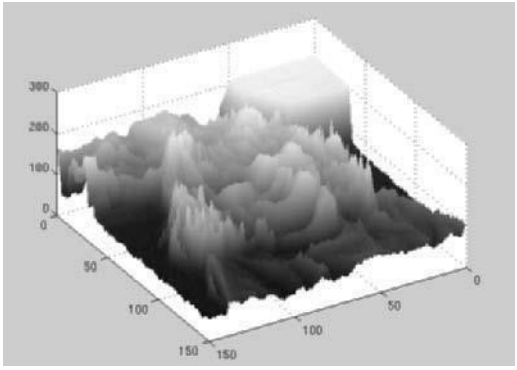


Figure 1.8: An unusual image representation.

Figure 1.9: Another representation of Figure 1.8.

Both representations contain exactly the same information, but for a human observer it is very difficult to find a correspondence between them, and without the second, it is unlikely that one would recognize the face of a child. The point is that a lot of a priori knowledge is used by humans to interpret the images; the machine only begins with an array of numbers and so will be attempting to make identifications and draw conclusions from data that to us are more like Figure 1.8 than Figure 1.9. Increasingly, data capture equipment is providing very large data sets that do *not* lend themselves to straightforward interpretation by humans—we have already mentioned terahertz imaging as an example.

Internal image representations are not directly understandable—while the computer is able to process local parts of the image, it is difficult for it to locate global knowledge. General knowledge, domain-specific knowledge, and information extracted from the image will be essential in attempting to 'understand' these arrays of numbers.

Low-level computer vision techniques overlap almost completely with digital image processing, which has been practiced for decades. The following sequence of processing steps is commonly seen:

An image is captured by a sensor (such as a camera) and digitized; then the computer suppresses noise (image pre-processing) and may be enhances some object features which are relevant to understanding the image. Edge extraction is an example of processing carried out at this stage.

Image segmentation is the next step, in which the computer tries to separate objects from the image background and from each other. Total and partial segmentation may be distinguished; total segmentation is possible only for very simple tasks, an example being the recognition of dark non-touching objects from a light background. For example, in analyzing images of printed text (an early step in optical character recognition, OCR) even this superficially simple problem is very hard to solve without error. In more complicated problems (the general case), low-level image processing techniques handle the partial segmentation tasks, in which only the cues which will aid further high-level

processing are extracted. Often, finding parts of object boundaries is an example of lowlevel partial segmentation.

Object description and classification in a totally segmented image are also understood as

part of low-level image processing. Other low-level operations are image compression, and techniques to extract information from (but not *understand*) moving scenes.

Low-level image processing and high-level computer vision differ in the data used.

Low-level data are comprised of original images represented by matrices composed of brightness (or similar) values, while high-level data originate in images as well, but only those data which are relevant to high-level goals are extracted, reducing the data quantity considerably.

High-level data represent knowledge about the image content—for example, object size, shape, and mutual relations between objects in the image. High-level data are usually expressed in symbolic form.

Many low-level image processing methods were proposed in the 1970s or earlier: research is trying to find more efficient and more general algorithms and is implementing them on more technologically sophisticated equipment, in particular, parallel machines (including GPU's) are being used to ease the computational load. The requirement for better and faster algorithms is fuelled by technology delivering larger images (better spatial or temporal resolution), and color.

A complicated and so far unsolved problem is how to order low-level steps to solve a specific task, and the aim of automating this problem has not yet been achieved. It is usually still a human operator who finds a sequence of relevant operations, and domain-specific knowledge and uncertainty cause much to depend on this operator's intuition and previous experience.

High-level vision tries to extract and order image processing steps using all available knowledge—image understanding is the heart of the method, in which feedback from high-level to low-level is used. Unsurprisingly this task is very complicated and computationally intensive.

Consider *3D vision problems* for a moment. We adopt the user's view, i.e., what tasks performed routinely by humans would be good to accomplish by machines. What is the relation of these 3D vision tasks to low-level (image processing) and high-level (image analysis) algorithmic methods? There is no widely accepted view in the academic community. Links between (algorithmic) components and representation levels are tailored to the specific application solved, e.g., navigation of an autonomous vehicle. These applications have to employ specific knowledge about the problem solved to be

competitive with tasks which humans solve. Many researchers in different fields work on related problems and research in 'cognitive systems' could be the key which may disentangle the complicated world of perception which includes also computer vision.

Figure 1.10 depicts several 3D vision tasks and algorithmic components expressed on different abstraction levels. In most cases, the bottom-up and top-down approach is adopted to fulfill the task.

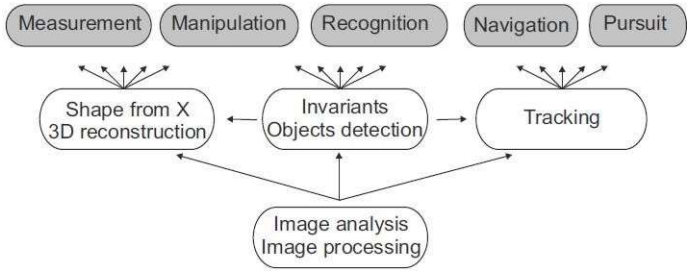


Figure 1.10: Several 3D computer vision tasks from the user's point of view are on the upper line (filled). Algorithmic components on different hierarchical levels support it in a bottom-up fashion.

1.4. IMAGE, ITS REPRESENTATIONS AND PROPERTIES

Image representations:

Mathematical models are often used to describe images.

A monochrome or monochromatic image, object or palette is composed of one color (or values of one color). Images using only shades of grey are called grayscale (typically digital) or black-and-white (typically analog).

A scalar function might be sufficient to describe a monochromatic image, while vector functions may be used to represent color images consisting of three component colors.

Functions are categorized as

- **Continuous**
- **Discrete**
- D**ig ital**.

A continuous function has continuous domain and range; if the domain set is discrete, then we have a discrete function; if the range set is also discrete, then we have a digital function. Many of these functions will be linear, and correspondingly simple to deal with.

Image can be modeled by a continuous function of two variables $f(x, y)$ where (x, y) are co-ordinates in a plane, or perhaps three variables $f(x, y, t)$, where t is time. This model is reasonable in the great majority of applications. Nevertheless, it is worth realizing that an 'image' may be acquired in many ways. We shall note often that color is the norm, even when algorithms are presented for monochromatic images, but we do not need to constrain ourselves to the visible spectrum. Infra-red cameras are now very common (for example, for night-time surveillance). Other parts of the electro-magnetic [EM] spectrum may also be used; microwave imaging, for example, is becoming widely available. Further, image acquisition outside the EM spectrum is also common: in the medical domain, datasets are generated via magnetic resonance (MR), X-ray computed tomography (CT), ultrasound etc. All of these approaches generate large arrays of data requiring analysis and understanding and with increasing frequency these arrays are of 3 or more dimensions.

The continuous image function

The (gray-scale) image function values correspond to brightness at image points. The function value can express other physical quantities as well (temperature, pressure distribution, distance from the observer, etc.). **Brightness** integrates different optical quantities—using brightness as a basic quantity allows us to avoid the complicated process of image formation.

The image on the retina or on a camera sensor is intrinsically two-dimensional (2D). We shall call such an image bearing information about brightness points an **intensity image**. The 2D image on the imaging sensor is commonly the result of projection of a three-dimensional (3D) scene. The simplest mathematical model for this is a pin-hole camera.

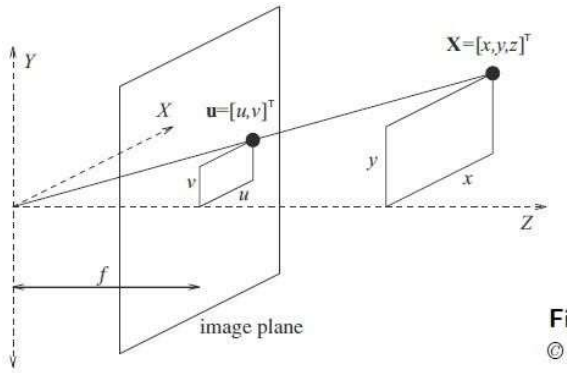


Figure 2.1: Perspective projection geometry.

The image plane is the plane that contains the object's projected image.
The 2D intensity image is the result of a **perspective projection** of the 3D scene, which is modeled by the image captured by a pin-hole camera illustrated in Figure 2.1. In this figure, the image plane has been reflected with respect to the *XY* plane in order not to get a mirrored image with negative co-ordinates. The quantities x, y, and z are coordinates of the point **X** in a 3D scene, and ***f* is the distance from the pinhole to the image plane. *f* is commonly called the focal length** because in lenses it has a similar meaning. The projected point **u** has co-ordinates (u, v) in the 2D image plane, which can easily be derived from similar triangles.

$$u = \frac{x f}{z}, \qquad v = \frac{y f}{z}. \tag{2.1}$$

A non-linear perspective projection is often approximated by a linear **parallel** (or **orthographic) projection**, where f. Implicitly, x says that the orthographic projection is a limiting case$\to \infty$ of the perspective$\to \infty$ projection for faraway objects.

When 3D objects are mapped into the camera plane by perspective projection, a lot of information disappears because such a transform is not one-to-one.

Recovering information lost by perspective projection is only one, problem of computer vision—another is understanding image brightness.

The only information available in an intensity image is the brightness of the appropriate pixel, which is dependent on a number of independent factors such as object surface reflectance properties (given by the surface material, microstructure, and marking), illumination properties, and object surface orientation with respect to viewer and light source.

It is a non-trivial and again ill-posed problem to separate these components when trying to recover the 3D geometry of an object from the intensity image.

Some applications work with 2D images directly—for example, an image of a flat specimen viewed by a microscope with transparent illumination, a character drawn on a sheet of paper, the image of a fingerprint, etc. Many basic and useful methods used in digital image analysis do not therefore depend on whether the object was originally 2D or 3D.

Image processing often deals with **static** images, in which time is constant. A monochromatic static image is represented by a continuous image function $f(x, y)$ whose arguments are coordinates in the plane.

Computerized image processing uses digital image functions which are usually rep resented by matrices, so co-ordinates are natural numbers. The domain of the image function is a region R in the plane

$$R = (x, y),\ 1 \leq x \leq x_m,\ 1 \leq y \leq y_n \ , \qquad (2.2)$$

where $x_m,\ y_n$ represent the maximal co-ordinates. The function has a limited domain—infinite summation or integration limits can be used, as the image function value is zero outside the domain R. The customary orientation of co-ordinates in an image is in the normal Cartesian fashion (horizontal x-axis, vertical y-axis, origin bottom-left), although the (*row, column*, origin top-left) orientation used in matrices is also often used.

The range of image function values is also limited; by convention, in monochromatic images the lowest value corresponds to black and the highest to white. Brightness values bounded by these limits are **gray-levels.** The quality of a digital image grows in proportion to the spatial, spectral, radiometric, and time resolutions. The **spatial resolution** is given by the proximity of image samples in the image plane; **spectral resolution** is given by the bandwidth of the light frequencies captured by the sensor; **radiometric resolution** corresponds to the number of distinguishable gray-levels; and **time resolution** is given by the interval between time samples at which images are captured. The question of time resolution is important in dynamic image analysis, where time sequences of images are processed.

Images $f(x, y)$ can be treated as deterministic functions or as realizations of stochastic processes. Mathematical tools used in image description have roots in linear system theory, integral transforms, discrete mathematics, and the theory of stochastic processes.

Mathematical transforms usually assume that the image function $f(x, y)$ is 'well- behaved', meaning that the function is integrable, has an invertible Fourier transform, etc.

Image digitization

An image to be processed by computer must be represented using an appropriate discrete data structure, for example, a matrix.

An image captured by a sensor is expressed as a continuous function $f(x, y)$ of two coordinates in the plane.

Image digitization means that the function $f(x, y)$ is **sampled** into a matrix with M rows and N columns. Image **quantization** assigns to each continuous sample an integer value—the continuous range of the image function $f(x, y)$ is split into K intervals. The finer the sampling (i.e., the larger M and N) and quantization (the larger K), the better the approximation of the continuous image function $f(x, y)$ achieved.

Image function sampling poses two questions. First, the sampling period should be determined—this is the distance between two neighboring sampling points in the image. Second, the geometric arrangement of sampling points (sampling grid) should be set.

Sampling

A continuous image is digitized at **sampling points**. These sampling points are ordered in the plane, and their geometric relation is called the **grid**. The digital image is then a data structure, usually a matrix. Grids used in practice are usually square (Figure 2.2a) or hexagonal (Figure 2.2b). It is important to distinguish the grid from the raster; the **raster** is the grid on which a neighborhood relation between points is defined.

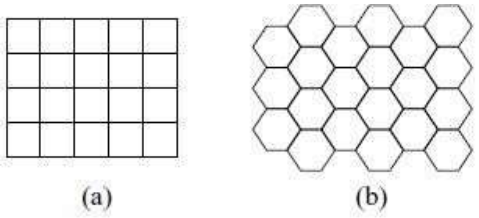


Figure 2.2: (a) Square grid. (b) Hexagonal grid.

One infinitely small sampling point in the grid corresponds to one picture element also called a **pixel** or **image element** in the digital image; in a three-dimensional image, an image element is called a **voxel** (volume element). The set of pixels together covers the entire image; however, the pixel captured by a real digitization device has finite size (since the sampling function is not a collection of ideal Dirac impulses but a collection of limited impulses). The pixel is a unit which is not further divisible from the image analysis point of view. We shall often refer to a pixel as a 'point'.

Quantization

A value of the sampled image $f_s(j\ \Delta x,\ k\ \Delta y)$ is expressed as a digital value in image processing.

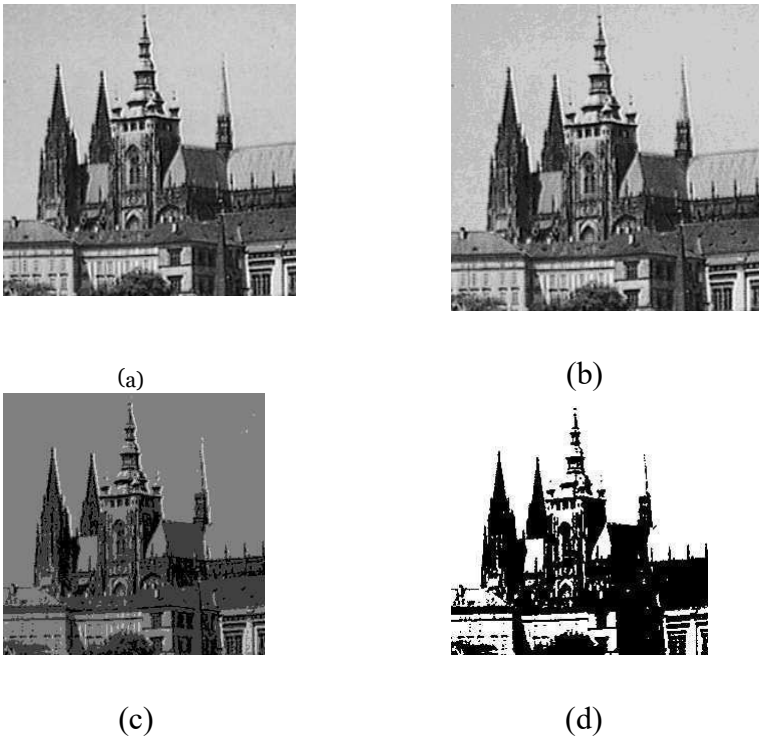
The transition between continuous values of the image function (brightness) and its digital equivalent is called **quantization**.

The number of quantization levels should be high enough to permit human perception of fine shading details in the image.

Most digital image processing devices use quantization into k equal intervals. If b bits are used to express the values of the pixel brightness then the number of brightness levels is $k = 2b$.

Eight bits per pixel per channel (one each for red, green, blue) are commonly used although systems using other numbers (e.g., 16) can be found. An efficient computer representation of brightness values in digital images requires that eight bits, four bits, or one bit are used per pixel, meaning that one, two, or eight pixel brightness can be stored in one byte.

The main problem in images quantized with insufficient brightness levels is the occurrence of false contours which effect arises when the number of brightness levels is lower than that which humans can easily distinguish. This number is dependent on many factors—for example, the average local brightness—but displays which avoid this effect will normally provide a range of at least 100 intensity levels. This problem can be reduced when quantization into intervals of unequal length is used; the size of intervals corresponding to less probable brightness in the image is enlarged.



(a) (b)

(c) (d)

Figure 2.3: Brightness levels. (a) 64. (b) 16. (c) 4. (d) 2. © *Cengage Learning 2015.*

Figures 3.11a and 2.3a-d demonstrate the effect of reducing the number of brightness levels in an image. An original image with 256 brightness levels has its number of brightness

---

levels reduced to 64 (Figure 2.3a), and no degradation is perceived. Figure 2.3b uses 16 brightness levels and false contours begin to emerge, and this becomes clearer in Figure 2.3c with four brightnesses and in Figure 2.3d with only two.

1.5. DIGITAL IMAGE PROPERTIES

A digital image has several properties, both metric and topological, which are some- what different from those of continuous two-dimensional functions. Human perception of digital images is a frequent aspect, since judgment of image quality is also important.

Metric and topological properties of digital images

A digital image consists of picture elements with finite size—these pixels carry informa tion about the brightness of a particular location in the image. Usually (and we assume this hereafter) pixels are arranged into a rectangular sampling grid. Such a digital image is represented by a two-dimensional matrix whose elements are natural numbers corresponding to the quantization levels in the brightness scale.

Some intuitively clear properties of continuous images have no straightforward anal- ogy in the domain of digital images. **Distance** is an important example. Any function D holding the following three condition is a 'distance' (or a *metric*)

$D(\mathbf{p}, \mathbf{q}) \geq 0$; $D(\mathbf{p}, \mathbf{q}) = 0$ if and only if $\mathbf{p} = \mathbf{q}$, *identity,*

$D(\mathbf{p}, \mathbf{q}) = D(\mathbf{q}, \mathbf{p})$, *symmetry,*

$D(\mathbf{p}, \mathbf{r}) \leq D(\mathbf{p}, \mathbf{q}) + D(\mathbf{q}, \mathbf{r})$, *triangular inequality.*

The distance between points with co-ordinates (i, j) and (h, k) may be defined in several different ways.

The **Euclidean distance** D_E known from classical geometry and everyday experi ence is defined by

$$D_E\big((i,j),(h,k)\big) = \sqrt{(i-h)^2 + (j-k)^2}\,. \quad (2.3)$$

The advantage of Euclidean distance is that it is intuitively obvious. The disadvantages are costly calculation due to the square root, and its non-integral value.

The distance between two points can also be expressed as the minimum number of elementary steps in the digital grid which are needed to move from the starting point to the end point. If only horizontal and vertical moves are allowed, the **'city block' distance** distance D_4 is obtained (also called the L_1 metric or Manhattan distance, because of the analogy with the distance between two locations in a city with a rectangular grid of streets):

$D_4 (i, j) , (h, k) = | i - h | + | j - k |$. (2.4)

If moves in diagonal directions are allowed in the digitization grid, we obtain the distance D_8, or **'chessboard' distance**. D_8 is equal to the minimal number of king- moves on the chessboard from one part to another:

$$D_8\big((i,j),(h,k)\big) = \max\{\,|\,i-h\,|,\,|\,j-k\,|\,\} \quad (2.5)$$

These distance definitions are illustrated in Figure 2.4.

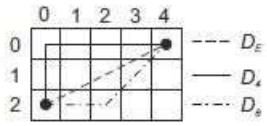


Figure 2.4: Distance metrics *De*, *D*4, and *D*8.

Pixel **adjacency** is another important concept in digital images. Two pixels (**p**, **q**) are called **4-neighbors** if they have distance D_4(**p**, **q**) = 1. Analogously, **8-neighbors** have D_8(**p**, **q**) = 1—see Figure 2.5.

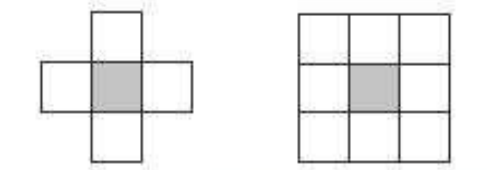


Figure 2.5: Neighborhood of the representative pixel (gray filled pixel in the middle). © *Cengage Learning 2015.*

It will become necessary to consider important sets consisting of several adjacent pixels—**regions** (in set theory, a region is a connected set). More descriptively, we can define a **path** from pixel *P* to pixel *Q* as a sequence of points $A_1, A_2, \ldots, A_n$, where $A_1 = P$, $A_n = Q$, and A_{i+1} is a neighbor of A_i, $i = 1, \ldots, n$ 1; then a region is a set of pixels in which there is a path between any pair of its pixels, all of whose pixels also belong to the set.

If there is a path between two pixels in the set of pixels in the image, these pixels are called **contiguous**

The relation 'to be contiguous' is reflexive, symmetric, and transitive and therefore defines a decomposition of the set (the image in our case) into equivalence classes (regions). Figure 2.6 illustrates a binary image decomposed by the relation 'contiguous' into three regions.

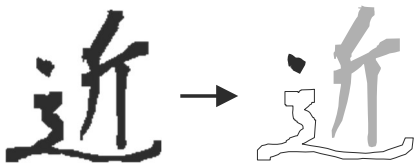


Figure 2.6: The relation ‘to be contiguous’ decomposes an image into individual regions. The Japanese Kanji character meaning ‘near from here’ decomposes into 3 regions.

Assume that R_i are disjoint regions in the image which were created by the relation ‘to be contiguous’, and further assume (to avoid special cases) that these regions do not touch the image bounds. Let R be the union of all regions R_i; R^C be the set complement of R with respect to the image. The subset of R^C which is contiguous with the image bounds is called the **background**, and the remainder of the complement R^C is called **holes**. A region is called **simple contiguous** if it has no holes. Equivalently, the complement of a simply contiguous region is contiguous. A region with holes is called **multiple contiguous**.

Note that the concept of region uses only the property ‘to be contiguous’. Secondary properties can be attached to regions which originate in image data interpretation. It is common to call some regions in the image **objects**; a process which determines which regions in an image correspond to objects in the world is a part of image **segmentation** and is discussed in Chapters 6 and 7.

The brightness of a pixel is a very simple property which can be used to find objects in some images; if, for example, a pixel is darker than some predefined value (threshold), then it belongs to the object. All such points which are also contiguous constitute one object. A hole consists of points which do not belong to the object and are surrounded by the object, and all other points constitute the background. An example is the black printed text on this white sheet of paper, in which individual letters are objects. White areas surrounded by the letter are holes, for example, the area inside the letter ‘o’. Other white parts of the paper are the background.

These neighborhood and contiguity definitions on the square grid create interesting paradoxes. Figure 2.7a shows two digital line segments with 45◦ slope. If 4-connectivity is used, the lines are not contiguous at each of their points. An even worse conflict with intuitive understanding of line properties is also illustrated; two perpendicular lines do intersect in one case (upper right intersection) and do not intersect in another case (lower left), as they do not have any common point (i.e., their set intersection is empty).

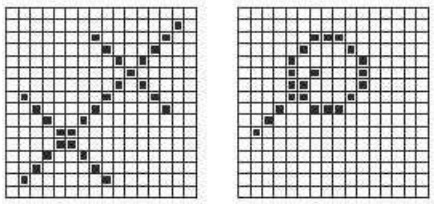


Figure 2.7: Paradoxes of crossing lines.

It is known from Euclidean geometry that each closed curve (e.g., a circle) divides the plane into two non-contiguous regions. If images are digitized in a square grid using 8connectivity, we can draw a line from the inner part of a closed curve into the outer part which does not intersect the curve (Figure 2.7b). This implies that the inner and outer parts of the curve constitute only one region because all pixels of the line belong to only one region, giving another paradox. One possible solution to contiguity paradoxes is to treat objects using 4-neighborhoods and background using 8-neighborhoods (or vice versa).

These problems are typical on square grids—a hexagonal grid (see Figure 2.2), however, solves many of them. Any point in the hexagonal raster has the same distance to all its six neighbors. There are some problems peculiar to the hexagonal raster as well (for example, it is difficult to express a Fourier transform on it). For reasons of simplicity and ease of processing, most digitizing devices use a square grid despite the known drawbacks.

An alternative approach to the connectivity problems is to use discrete topology based on cellular complexes. This approach develops a complete strand of image encoding and segmentation that deals with many issues we shall come to later, such as the representation of boundaries and regions. The idea, first proposed by the German mathematician Riemann in the nineteenth century, considers families of sets of different dimensions; points, which are 0-dimensional sets, may then be assigned to sets containing higher-dimensional structures (such as pixel arrays). This approach permits the removal of the paradoxes we have seen.

The **distance transform**—also called the **distance function** or **chamfering algorithm** or simply **chamfering**—is a simple application of the concept of distance. The idea is important as it provides the basis of several fast algorithms that will be seen multiple times in this book. The distance transform provides the distance of pixels from some image subset (perhaps describing objects or some features). The resulting ‘image’ has pixel values of 0 for elements of the relevant subset, low values for close pixels, and then high values for pixels remote from it—the appearance of this array gives the name to the technique.

For illustration, consider a binary image, in which ones represent the objects and zeros the background. The input image is shown in Figure 2.8 and the result of the D_4 distance transform is illustrated in Figure 2.9.

To calculate the transform, a two-pass algorithm has been suggested for distances D_4 and D_8. The idea is to traverse the image by a small local mask. The first pass starts from the top-left of the image and moves horizontally

0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	1	1	0	0	0	1	0
0	1	0	0	0	0	0	1
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0

Figure 2.8: Input image: gray pixels correspond to objects and white to background.

5	4	4	3	2	1	0	1
4	3	3	2	1	0	1	2
3	2	2	2	1	0	1	2
2	1	1	2	1	0	1	2
1	0	0	1	2	1	0	1
1	0	1	2	3	2	1	0
1	0	1	2	3	3	2	1
1	0	1	2	3	4	3	2

Figure 2.9: Result of the $[D_4]$ distance transform. © *Cengage Learning 2015.*

left to right until it reaches the bounds of the image and then returns to the beginning of the next row. The second pass goes from the bottom-right corner in the opposite bottom-up, right to left direction using a different local mask. The effectiveness of the algorithm comes from propagating the values of the previous image investigation in a ‘wave-like’ manner. The masks used in calculations are shown in Figure 2.10.

Algorithm 2.1: Distance transform

1. Choose a distance N_{max} that exceeds the image dimension with respect to the chosen distance metric (D_4 or D_8). Initialize an image F of the same dimension as the input: set pixels corresponding to the subset(s) to be chamfered to 0, and all others to N_{max}.
2. Pass through image pixels from top to bottom and left to right. For a given pixel, consider neighbors above and to the left and set

$$F(\mathbf{p}) = \min_{\mathbf{q} \in AL} F(\mathbf{p}), D(\mathbf{p}, \mathbf{q}) + F(\mathbf{q}) \quad .$$

3. Pass through image pixels from bottom to top and rightto left. For a given pixel, consider neighbors above and to the left and set

$$F(\mathbf{p}) = \min_{\mathbf{q} \in BR} F(\mathbf{p}), D(\mathbf{p}, \mathbf{q}) + F(\mathbf{q}) \quad .$$

4. If any pixels remain still set at N_{max}, go to (2).
5. The array F now holds a chamfer of the chosen subset(s).

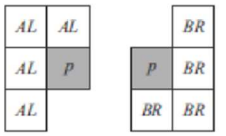
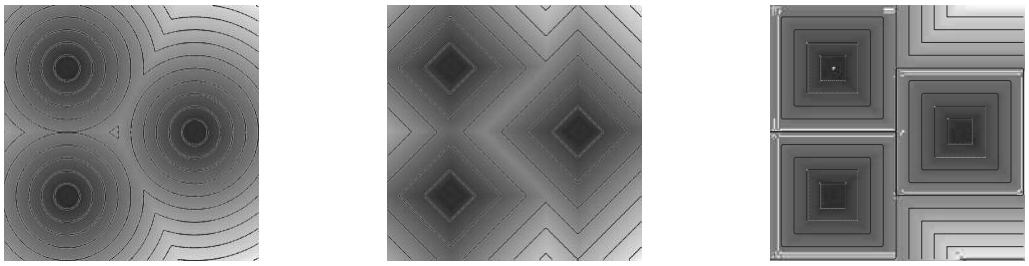


Figure 2.10: Pixel neighborhoods used in distance transform calculations—**p** is the central one. The neighborhood on the left is used in the first pass (top-down, left to right); that on the right is used in the second (bottom-up, right to left). © *Cengage Learning 2015.*



(a) (b) (c)

Figure 2.11: Three distances used often in distance transform calculations—the input consists of three isolated 'ones'. Output distance is visualized as intensity; lighter values denote higher distances. Contour plots are superimposed for better visualization. (a) Euclidean distance *DE* . (b) City block distance *D*4. (c) Chessboard distance *D*8.

This algorithm needs obvious adjustments at image boundaries, where the sets AL and BR are truncated. It is open to various improvements by using different distance calculations. Distance transform performance for D_E, D_4, D_8 on an input consisting only of three distinct 'ones' is shown in Figure 2.11.

The distance transform has many applications, e.g., in discrete geometry, path planning and obstacle avoidance in mobile robotics, finding the closest feature in the image, and skeletonization.

An **edge** is a further important concept used in image analysis. This is a local property of a pixel and its immediate neighborhood—it is a vector given by a magnitude and direction which tells us how fast the image intensity varies in a small neighborhood of a pixel. Images with many brightness levels are used for edge computation, and the gradient of the image function is used to compute edges. The edge direction is perpendicular to the gradient direction which points in the direction of the fastest image function growth.

The related concept of the **crack edge** creates a structure between pixels in a similar manner to that of cellular complexes. However, it is more pragmatic and less mathematically rigorous. Four crack edges are attached to each pixel, which are defined by its relation to its 4-neighbors. The direction of the crack edge is that of increasing bright- ness, and is a multiple of 90◦, while its magnitude is the absolute difference between the brightness of the relevant pair of pixels. Crack edges are illustrated in Figure 2.12 and may be be used in considering image segmentation.

The **border** (boundary) of a region is another important concept in image analysis. The border of a region R is the set of pixels within the region that have one or more neighbors outside R. The definition corresponds to an intuitive understanding of the border as a set of points at the bound of the region. This definition is sometimes referred to as the **inner border** to distinguish it from the **outer border**, that is, the border of the background (i.e., its complement) of the region. Inner and outer borders are illustrated in Figure 2.13. Due to the discrete nature of the image, some inner border elements which would be distinct in the continuous case coincide in the discrete case, as can be seen with the one-pixel-wide line at the right of Figure 2.13.

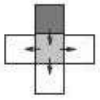


Figure 2.12: Crack edges. © *Cengage Learning 2015.*

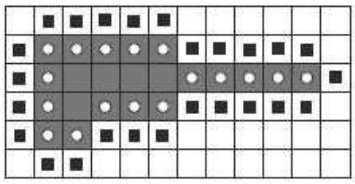


Figure 2.13: Region inner borders shown as white circles and outer borders shown as black squares (using 4-neighborhoods). © *Cengage Learning 2015.*

While edges and borders are related, they are not the same thing. 'Border' is a global concept related to a region, while 'edge' expresses local properties of an image function. One possibility for finding boundaries is chaining the significant edges (points with high gradient of the image function). Methods of this kind are described in Section 6.2.

A region is described as **convex** if any two points within it are connected by a straight line segment, and the whole line lies within the region—see Figure 2.14. The property of convexity decomposes all regions into two equivalence classes: convex and non-convex.

Figure 2.14: A convex region (left) and non-convex region (right). © *Cengage Learning 2015.*

A **convex hull** of a region is the smallest convex region containing the input (possibly non-convex) region. Consider an object whose shape resembles the letter 'R' (see Figure 2.15). Imagine a thin rubber band pulled around the object; the shape of the rubber band provides the convex hull of the object.

Figure 2.15: Description using topological components: An 'R' object, its convex hull, and the associated lakes and bays. © *Cengage Learning 2015.*

Topological properties are not based on the distance concept. Instead, they are invariant to *homeomorphic* transforms which can be illustrated for images as **rubber sheet transforms**. Imagine a small rubber balloon with an object painted on it; topo- logical properties of the object are those which are invariant to arbitrary stretching of the rubber sheet. Stretching does not change contiguity of the object parts and does not change the number of holes in regions. We use the term 'topological properties' of the region to describe its qualitative properties invariant to small changes (e.g., the property of being convex), even though an arbitrary homeomorphic transformation can change a convex region to a nonconvex one and vice versa. Considering the rubber sheet analogy, we mean that the stretching of the sheet is only gentle.

An object with non-regular shape can be represented by a collection of its topological components, Figure 2.15. The set inside the convex hull which does not belong to an object is called the **deficit of convexity**. This can be split into two subsets: **lakes** (dark gray) are

fully surrounded by the object; and **bays** (light gray) are contiguous with the border of the convex hull of the object.

The convex hull, lakes, and bays are sometimes used for object description; these features are used in Chapter 8 (object description) and in Chapter 13 (mathematical morphology).

Histograms

The **brightness histogram** $h_f(x)$ of an image provides the frequency of the brightness value x in the image—the histogram of an image with L gray-levels is represented by a onedimensional array with L elements.

Algorithm 2.2: Computing the brightness histogram

1. Assign zero values to all elements of the array h_f.
2. For all pixels (x, y) of the image f, increment $h_f\ f(x, y)$ by 1.

The histogram provides a natural bridge between images and a probabilistic description. We might want to find a first-order probability function $p_1(x; x, y)$ to indicate the probability that pixel (x, y) has brightness x. Dependence on the position of the pixel is not of interest in the histogram; the function $p_1(x)$ is of interest and the brightness histogram is its estimate. The histogram is often displayed as a bar graph, see Figure 2.16.

The histogram is usually the only global information about the image which is available. It is used when finding optimal illumination conditions for capturing an image, gray-scale transformations, and image segmentation to objects and background. Note

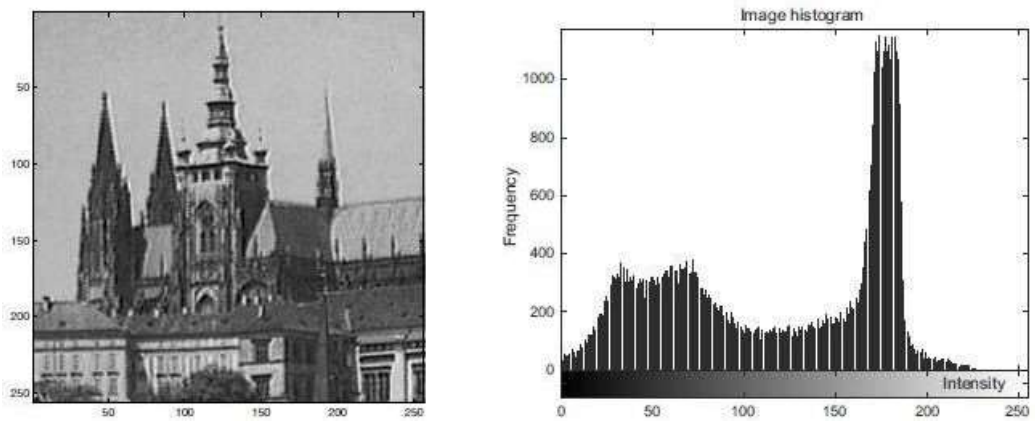


Figure 2.16: Original image (a) and its brightness histogram (b).

that one histogram may correspond to several images; for instance, a change of the object position on a constant background does not affect it.

The histogram of a digital image typically has many local minima and maxima, which may complicate its further processing. This problem can be avoided by local smoothing of the histogram; this may be done, for example, using local averaging of neighboring histogram elements as the base, so that a new histogram $h'f(x)$ is calculated according to

$$h'_f(z) = \frac{1}{2K+1} \sum_{j=-K}^{K} h_f(z+j)\,, \qquad (2.6)$$

where K is a constant representing the size of the neighborhood used for smoothing. This algorithm would need some boundary adjustment, and carries no guarantee of removing all local minima.

Entropy

If a probability density p is known then image information content can be estimated regardless of its interpretation using **entropy** H. The concept of entropy has roots in thermodynamics and statistical mechanics but it took many years before it was related to information. The information-theoretic formulation comes from Shannon and is often called **information entropy**.

An intuitive understanding of information entropy relates to the amount of uncertainty about an event associated with a given probability distribution. The entropy can serve as an measure of 'disorder'. As the level of disorder rises, entropy increases and events are less predictable.

Entropy is formally defined assuming a discrete random variable X with possible outcomes (called also states) $x_1, \ldots, x_n$. Let $p(x_k)$ be the probability of the outcome x_k, $k = 1, \ldots n$. Then the entropy is defined as

$$H(X) \equiv \sum_{k=1}^{n} p(x_k) \log_2 \left(\frac{1}{p(x_k)} \right) = -\sum_{k=1}^{n} p(x_k) \log_2 p(x_k)\,. \qquad (2.7)$$

The entropy of the random variable X is the sum, over all possible outcomes k of X, of the product of the probability of outcome x_k with the logarithm of the inverse of the probability of x_k.

The base of the logarithm in this formula determines the unit in which entropy is measured. If this base is two then the entropy is given in bits. Recall that the probability density $p(x_k)$ needed to calculate the entropy is often estimated using a gray- level histogram in image analysis.

Entropy measures the uncertainty about the realization of a random variable. For Shannon, it served as a proxy capturing the concept of information contained in a message as opposed to the portion of the message that is strictly determined and predictable by inherent structures. For example, we shall explore entropy to assess redundancy in an image for image compression.

Visual perception of the image

Anyone who creates or uses algorithms or devices for digital image processing should take into account the principles of human image perception. If an image is to be analyzed by a human the information should be expressed using variables which are easy to perceive; these are psycho-physical parameters such as contrast, border, shape, texture, color, etc. Humans will find objects in images only if they may be distinguished effortlessly from the background. A detailed description of the principles of human visual perception can be found in. Human perception of images provokes many illusions, the understanding of which provides valuable clues about visual mecha nisms.

The situation would be relatively easy if the human visual system had a linear response to composite input stimuli—i.e., a simple sum of individual stimuli. A decrease of some stimulus, e.g., area of the object in the image, could be compensated by its intensity, contrast, or duration. In fact, the sensitivity of human senses is roughly logarithmically proportional to the intensity of an input signal. In this case, after an initial logarithmic transformation, response to composite stimuli can be treated as linear.

Contrast

Contrast is the local change in brightness and is defined as the ratio between average brightness of an object and the background. Strictly speaking, we should talk about luminance4 instead of brightness if our aim is to be physically precise. The human eye is logarithmically sensitive to brightness, implying that for the same perception, higher brightness requires higher contrast.

Apparent brightness depends very much on the brightness of the local surroundings; this effect is called conditional contrast. Figure 2.17 illustrates this with five circles of the same size surrounded by squares of different brightness. Humans perceive the brightness of the small circles as different.

Figure 2.17: Conditional contrast effect. Circles inside squares have the same brightness and are perceived as having different brightness values. © *Cengage Learning 2015.*

Acuity

Acuity is the ability to detect details in an image. The human eye is less sensitive to slow and fast changes in brightness in the image plane but is more sensitive to intermediate changes. Acuity also decreases with increasing distance from the optical axis.

Resolution in an image is firmly bounded by the resolution ability of the human eye; there is no sense in representing visual information with higher resolution than that of

---

the viewer. Resolution in optics is defined as the inverse value of a maximum viewing angle between the viewer and two proximate points which humans cannot distinguish, and so fuse together.

Human vision has the best resolution for objects which are at a distance of about 250 mm from an eye under illumination of about 500 lux; this illumination is provided by a 60 W bulb from a distance of 400 mm. Under these conditions the distance between two distinguishable points is approximately 0.16 mm.

Some visual illusions

Human perception of images is prone to many illusions.

Object borders carry a lot of information for humans. Boundaries of objects and simple patterns such as blobs or lines enable adaptation effects similar to conditional contrast, mentioned above. The Ebbinghaus illusion is a well-known example—two circles of the same diameter in the center of images appear to have different diameters (Figure 2.18).

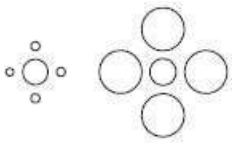


Figure 2.18: The Ebbinghaus illusion. © *Cengage Learning 2015.*

Perception of one dominant shape can be fooled by nearby shapes. Figure 2.19 shows parallel diagonal line segments which are not perceived as parallel. Figure 2.20 contains rows of black and white squares which are all parallel. However, the vertical zigzag squares disrupt our horizontal perception.

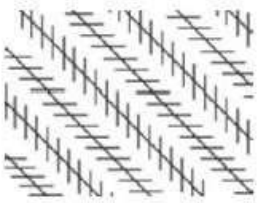


Figure 2.19: Disrupted parallel diagonal lines. © *Cengage Learning 2015.*

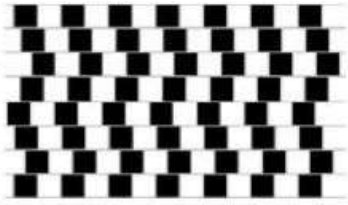


Figure 2.20: Horizontal lines are parallel, although not perceived as such. © *Cengage Learning 2015.*

Perceptual grouping

Perceptual grouping [Palmer, 1999] is a principle used in computer vision to aggregate elements provided by low-level operations such as edges, which are small blobs to bigger chunks having some meaning. Its roots are in Gestalt psychology first postulated by Wertheimer in 1912 [Brett King and Wertheimer, 2005]. Gestalt psychology proposes that the operational principle of the mind and brain is holistic, parallel, and with self- organizing tendencies.

Gestalt theory was meant to have general applicability; its main tenets, however, were induced almost exclusively from observations on visual perception. The overriding theme of the theory is that stimulation is perceived in organized or configuration terms. Gestalt in German means configuration, structure or pattern of physical, biological, or psychological phenomena so integrated to constitute a functional unit with properties not derivable by summation of its parts. Patterns take precedence over elements and have properties that are not inherent in the elements themselves.

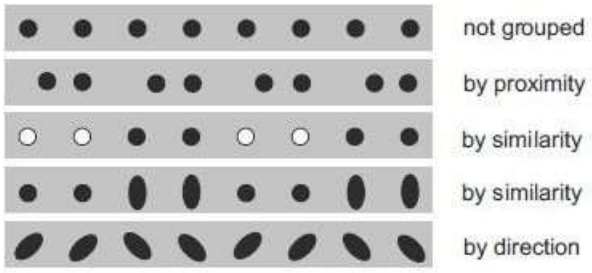


Figure 2.21: Grouping according to properties of elements.

The human ability to group items according to various properties is illustrated in Figure 2.21. Perceived properties help people to connect elements together based on strongly perceived properties as parallelism, symmetry, continuity and closure taken in a loose sense as illustrated in Figure 2.22.

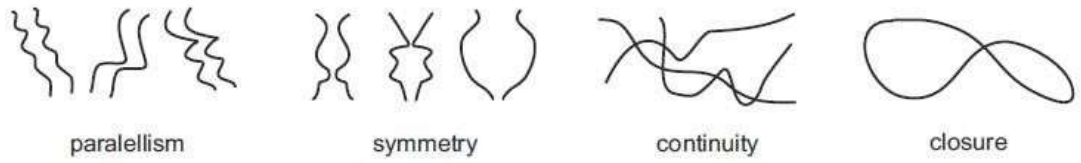


Figure 2.22: Illustration of properties perceived in images which allow humans to group together elements in cluttered scenes.

It has been demonstrated that mimicking perceptual grouping in machine vision system is a plausible technique. It permits the creation of more meaningful chunks of information from meaningless outcomes of low-level operations such as edge detection. Such grouping is useful in image understanding. This principle will be used in this book mainly for image segmentation.

Image quality

An image might be degraded during capture, transmission, or processing, and measures of image quality can be used to assess the degree of degradation. The quality required naturally depends on the purpose for which an image is used.

Methods for assessing **image quality** can be divided into two categories: subjective and objective. Subjective methods are often used in television technology, where the ultimate criterion is the perception of a selected group of professional and lay viewers. They appraise

an image according to a list of criteria and give appropriate marks. Details about subjective methods may be found in [Pratt, 1978].

Objective quantitative methods measuring image quality are more interesting for our purposes. Ideally such a method also provides a good subjective test, and is easy to apply; we might then use it as a criterion in parameter optimization. The quality of an image $f(x, y)$ is usually estimated by comparison with a known reference image $g(x, y)$ [Rosenfeld and Kak, 1982], and a synthesized image is often used for this purpose. One class of methods uses simple measures such as the mean quadratic difference $\Sigma (g - f)^2$, but this does not distinguish a few big differences from many small differences. Instead of the mean quadratic difference, the mean absolute difference or simply maximal absolute difference may be used. Correlation between images f and g is another alternative.

Another class measures the resolution of small or proximate objects in the image. An image consisting of parallel black and white stripes is used for this purpose; then the number of black and white pairs per millimeter gives the resolution.

2.21.1 Noise in images

Real images are often degraded by some random errors—this degradation is usually called **noise**. Noise can occur during image capture, transmission, or processing, and may be dependent on, or independent of, the image content.

1) white noise

white noise is **a random signal having equal intensity at different frequencies, giving it a constant** power spectral density.

2) Gaussian noise

Gaussian noise, named after Carl Friedrich Gauss, is a kind of signal noise that has a probability density function equal to that of the normal distribution (which is also known as the Gaussian distribution)

3) Additive noise

Additive noise is the undesired abrupt signal that gets added into some genuine signal.

4) Multiplicative noise

Multiplicative noise is the undesired abrupt signal that gets multiplied into some genuine signal.

5) Quantization noise

Quantization noise results when a continuous random variable is converted to a discrete one or when a discrete random variable is converted to one with fewer levels. In images, quantization noise often occurs in the acquisition process.

6) Impulse noise

Impulse noise is a category of noise that includes unwanted, almost instantaneous sharp sounds —typically caused by electromagnetic interference, scratches on disks, gunfire, explosions, and synchronization issues in digital audio.

7) Salt and pepper noise

salt--and--pepper noise describes a situation where random pixels get replaced by extremely dark or bright values.

1.6 COLOR IMAGES

Human color perception adds a subjective layer on top of underlying objective physical properties—the wavelength of electromagnetic radiation.
color may be considered a psycho-physical phenomenon.

Color has long been used in painting, photography and films to display the surrounding world to humans in a similar way to that in which it is perceived in reality. There is considerable literature on the variants in the naming of colors across languages, which is a very subtle affair. The human visual system is not very precise in perceiving color in absolute terms; if we wish to express our notion of color precisely we would describe it relative to some widely used color which is used as a standard: e.g.,
the red of a British pillar box. There are whole industries which present images to humans—the press, films, displays, and hence a desire for color constancy. In computer vision, we have the advantage of using a camera as a measuring device, which yields measurements in absolute quantities.

Newton reported in the 17th century that white light from the sun is a spectral mixture, and used the optical prism to perform decomposition. This was a radical idea to propose at

time; over 100 years later influential scientists and philosophers such as Goethe refused to believe it.

2.3.1 Physics of color

The electromagnetic spectrum is illustrated in Figure 2.23.

Only a narrow section of the electromagnetic spectrum is visible to a human, with wavelength from approximately 380 nm to 740 nm. Visible colors with the wavelengths shown in Figure 2.24 are called **spectral colors** and are those which humans see when white light is decomposed using a Newtonian prism, or which are observed in a rainbow on the sky. Colors can be represented as combinations of the **primary colors**, e.g., red,

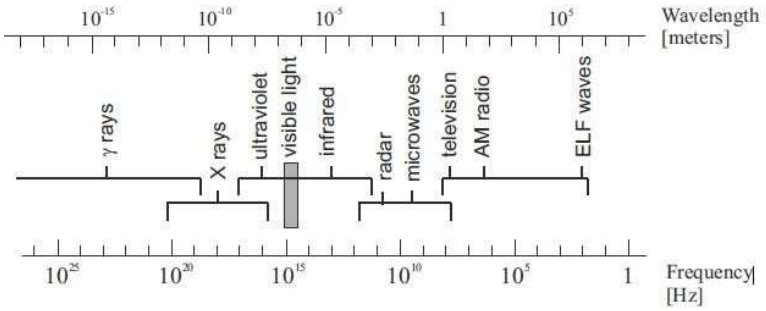


Figure 2.23: Division of the electromagnetic spectrum (ELF is Extremely Low Frequencies).

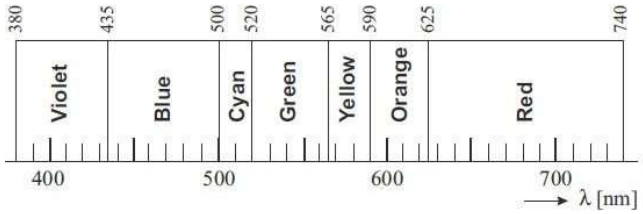


Figure 2.24: Wavelength λ of the spectrum visible to humans.

green, and blue, which for the purposes of standardization have been defined as 700 nm, 546.1 nm, and 435.8 nm, respectively [Pratt, 1978], although this standardization does not imply that all colors can be synthesized as combinations of these three.

The intensity of irradiation for different wavelengths λ usually changes. This varia tion is expressed by a **power spectrum** (called also power spectrum distribution) $S(\lambda)$. Why do we see the world in color? There are two predominant physical mechanisms describing what happens when a surface is irradiated. First, the **surface reflection** rebounds incoming energy in a similar way to a mirror. The spectrum of the reflected light remains the same as that of the illuminant and it is independent of the surface— recall that shiny metals 'do not have a color'. Second, the energy diffuses into the material and reflects randomly from the internal pigment in the matter. This mechanism is called **body reflection** and is predominant in dielectrics such as plastic or paints. Figure 2.25 illustrates both surface reflection (mirroring along surface normal **n**) and body reflection.

Colors are caused by the properties of pigment particles which absorb certain wavelengths from the incoming illuminant wavelength spectrum.

Most sensors used for color capture, e.g., in cameras, do not have direct access to color; the exception is a **spectrophotometer** which in principle resembles Newton's prism. Incoming irradiation is decomposed into spectral colors and intensity along the spectrum with changing wavelength λ is measured in a narrow wavelength band, for in- stance, by a mechanically moved point sensor. Actual spectrophotometers use diffraction gratings instead of a glass prism.

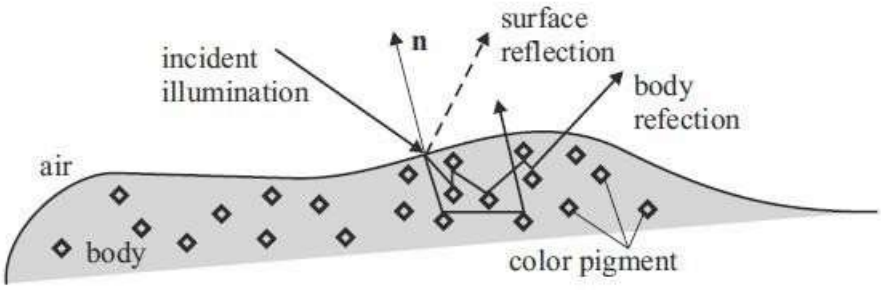


Figure 2.25: Observed color of objects is caused by certain wavelength absorptions by pigment particles in dielectrics.

Sometimes, intensities measured in several narrow bands of wavelengths are collected in a vector describing each pixel. Each spectral band is digitized independently and is represented by an individual digital image function as if it were a monochromatic image. In this way, **multispectral images** are created. Multispectral images are commonly used in remote sensing from satellites, airborne sensors and in industry. Wavelength usually span from ultraviolet through the visible section to infrared. For instance, the LAND- SAT 7 satellite transmits digitized images in eight spectral bands from near-ultraviolet to infrared.

Color perceived by humans

Evolution has developed a mechanism of indirect color sensing in humans and some animals. Three types of sensors receptive to the wavelength of incoming irradiation have been established in humans, thus the term **trichromacy**. Color sensitive receptors on the human retina are the **cones**. The other light sensitive receptors on the retina are the **rods** which are dedicated to sensing monochromatically in low ambient light conditions. Cones are categorized into three types based on the sensed wavelength range: S (short) $\approx$ with maximum sensitivity$\approx$ at 430 nm, M (medium) at 560 nm, and L (long) at 610 nm. Cones S, M, L are occasionally called cones B, G and R, respectively, but that is slightly misleading. We do not see red solely because an L cone is activated. Light with equally distributed wavelength spectrum looks white to a human, and an unbalanced spectrum appears as some shade of color.

The reaction of a photoreceptor or output from a sensor in a camera can be modeled mathematically. Let i be the specific type of sensor, $i = 1, 2, 3,$ (the retinal cone type S, M, L in the human case). Let $R_i(\lambda)$ be the spectral sensitivity of the sensor, $I(\lambda)$ be the spectral density of the illumination, and $S(\lambda)$ describe how the surface patch reflects each wavelength of the illuminating light. The spectral response q_i of the i-th sensor, can be modeled by integration over a certain range of wavelengths

$$q_i = \int_{\lambda_1}^{\lambda_2} I(\lambda)\, R_i(\lambda)\, S(\lambda)\, \mathrm{d}\lambda\,. \qquad (2.16)$$

Consider the cone types S, M, L. How does the vector (q_S, q_M, q_L) represent the color of the surface patch? It does not according to equation (2.16) since the output from the photosensors depends on the three factors $I(\lambda)$, $S(\lambda)$ and $R(\lambda)$. Only the factor $S(\lambda)$ is related to the surface patch. Only in the ideal case, when the illumination is perfectly white, i.e., $I(\lambda) = 1$, can we consider (q_S, q_M, q_L) as an estimate of the color of the surface.

Figure 2.26 illustrates qualitatively the relative sensitivities of S, M, L cones. Measurements were taken with the white light source at the cornea so that absorption of wavelength in cornea, lens and inner pigments of the eye is taken into account [Wandell, 1995].
A phenomenon called **color metamer** is relevant. A metamer, in general, means two things that are physically different but perceived as the same. Red and green adding to produce yellow is a color metamer, because yellow could have also been produced by a spectral color. The human visual system is fooled into perceiving that red and green is the same as yellow.

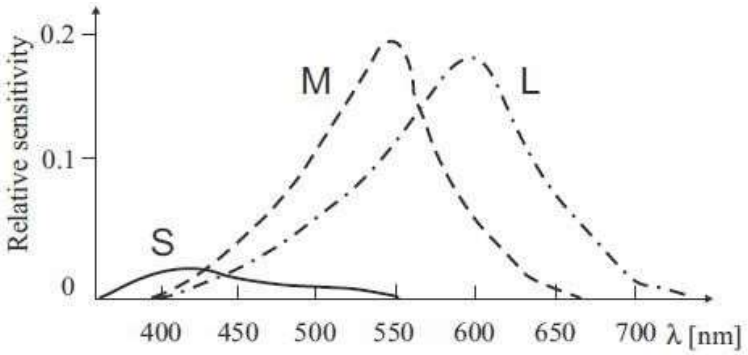


Figure 2.26: Relative sensitivity of S, M, L cones of the human eye to wave- length.

Consider a color matching experiment in which someone is shown a pattern consisting of two adjacent color patches. The first patch displays a test light—a spectral color of certain wavelength. The second patch is created as an additive combination of three selected primary lights, e.g., colors red=645.2 nm, green=525.3 nm and blue=444.4 nm. The observer is asked to control the red, green and blue intensities until both patches look identical. This color matching experiment is possible because of the color metamer. The result of measurements (redrawn from [Wandell, 1995]) is in Figure 2.27. Negative lobes can be seen on the curves for red and green in this figure. This would seem to be impossible. For wavelengths exhibiting negative values the three additive lights do not perceptually match the spectral color because it is darker. If the perceptual match has to be obtained then the observer has to add the intensity to the patch corresponding to the spectral color. This increase of this intensity is depicted as a decrease in the color matching function. Hence the negative values.

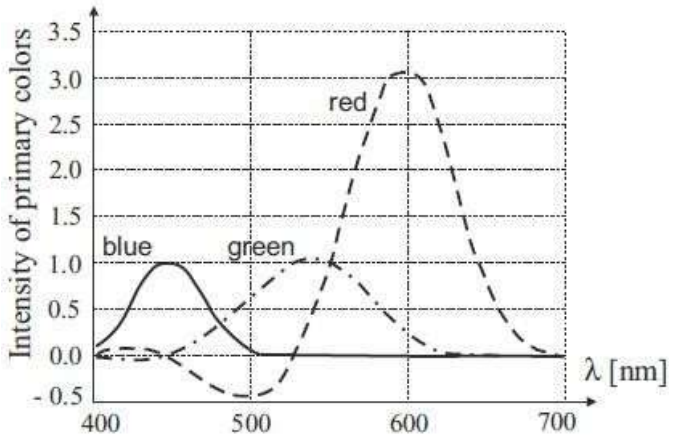


Figure 2.27: Color matching functions ob- tained in the color matching experiment. Intensities of the selected primary colors which perceptually match spectral color of given wavelength λ.

Human vision is prone to various illusions. Perceived color is influenced, besides the spectrum of the illuminant, by the colors and scene interpretation surrounding the observed color. In addition, eye adaptation to changing light conditions is not very fast and perception is influenced by adaptation. Nevertheless, we assume for simplicity that the spectrum of light coming to a point on the retina fully determines the color.

Since color can be defined by almost any set of primaries, the world community agreed on primaries and color matching functions which are widely used. The **color model** was introduced as a mathematical abstraction allowing us to express colors as tuples of numbers, typically as three or four values of color components. Being motivated by the press and the development of color film, in 1931, CIE (International Commission on Illumination, still acting in Lausanne, Switzerland) issued a technical standard called **XYZ color space**.

The standard is given by the three imaginary lights X=700.0 nm, Y =546.1 nm, Z=435.8 nm and by the color matching functions $X(\lambda)$, $Y(\lambda)$ and $Z(\lambda)$ corresponding to the perceptual ability of an average human viewing a screen through an aperture providing a 2◦ field of view. The standard is artificial because there is no set of physically realizable primary lights that would yield the color matching functions in the experiment. Nevertheless, if we wanted to characterize the imaginary lights then, roughly speaking, X red, Y green and Z blue. The CIE standard is an example of an absolute standard, i.e., defining unambiguous representation of color which does not depend on other external factors. There are more recent and more precise absolute standards: CIELAB 1976 (ISO 13665) and Hunter Lab

(http://www.hunterlab.com). Later, we will also dis- cuss relative color standards such as RGB color space: there are several RGB color spaces used—two computer devices may display the same RGB image differently.

The XYZ color standard fulfills three requirements:

- Unlike the color matching experiment yielding negative lobes of color matching functions, the matching functions of XYZ space are required to be non-negative.
- The value of $Y(\lambda)$ should coincide with the brightness (luminance).
- Normalization is performed to assure that the power corresponding to the three color matching functions is equal (i.e., the area under all three curves is equal). The resulting color matching functions are shown in Figure 2.28. The actual color is a mixture (more precisely a convex combination) of

$$c_X X + c_Y Y + c_Z Z, \qquad (2.17)$$

where $0 \; c_X, \; c_Y, \; c_Z \; 1$ are weights (intensities) in the mixture. The subspace of colors perceivable by humans$^{\leq}$ is called$^{\leq}$ the color gamut and is demonstrated in Figure 2.29.

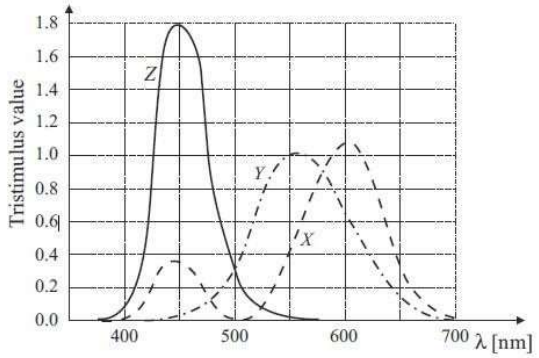


Figure 2.28: Color matching functions for the CIE standard from 1931. $X(\lambda)$, $Y(\lambda)$, $Z(\lambda)$ are color matching functions. *Based on [Wandell, 1995].*

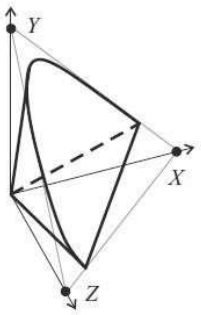


Figure 2.29: Color gamut - a subspace of the X, Y, Z color space showing all colors perceivable by humans. © *Cen-*

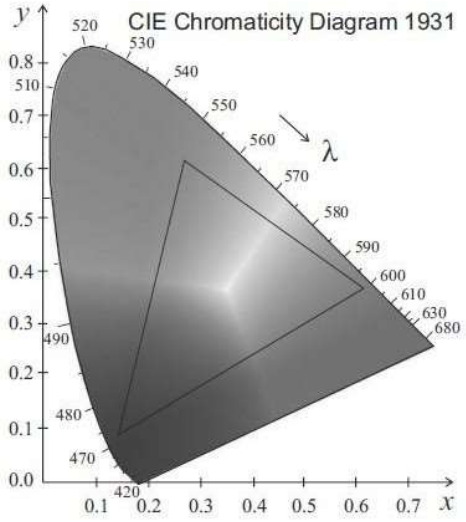


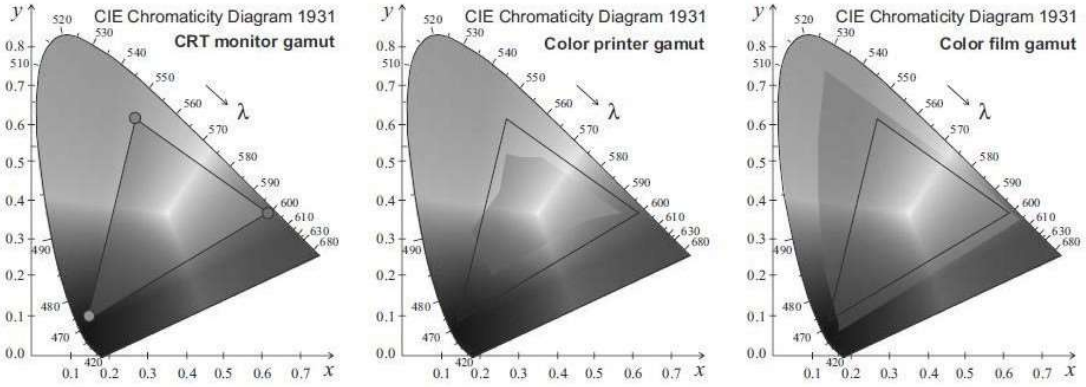
Figure 2.30: CIE chromaticity diagram is a projection of XYZ color space into a plane. The triangle depicts a subset of colors spanned by red, green, and blue. These are TV colors, i.e., all possible colors, which can be seen on a CRT display. © *Cengage Learning 2015. A color version of this figure may be seen in the color inset—Plate 1.*

3D figures are difficult to represent, and so a planar view of a 3D color space is used. The projection plane is given by the plane passing through extremal points on all three axes, i.e., points *X, Y, Z*. The new 2D coordinates *x, y* are obtained as

$$x = \frac{X}{X+Y+Z}, \qquad y = \frac{Y}{X+Y+Z}, \qquad z = 1 - x - y\,.$$

The result of this plane projection is the CIE chromaticity diagram, see Figure 2.30. The horseshoe like subspace contains colors which people are able to see. All mono- chromatic spectra visible to humans map into the curved part of the horseshoe—their wavelengths are shown in Figure 2.30.

Display and printing devices use three selected real primary colors (as opposed to three synthetic primary colors of XYZ color space). All possible mixtures of these primary colors fail to cover the whole interior of the horseshoe in CIE chromaticity diagram. This situation is demonstrated qualitatively for three particular devices in Figure 2.31.



(a) CRT monitor (b) printer (c) film

Figure 2.31: Gamuts which can be displayed using three typical display devices. © *Cengage Learning 2015. A color version of this figure may be seen in the color inset—Plate 2.*

Color spaces

Several different primary colors and corresponding color spaces are used in practice, and these spaces can be transformed into each other. If the absolute color space is used then the transformation is the one-to-one mapping and does not lose information (except for rounding errors). Because color spaces have their own gamuts, information is lost if the transformed value appears out of the gamut. See [Burger and Burge, 2008] for a full explanation and for algorithms; here, we list several frequently used color spaces.

The **RGB** color space has its origin in color television where Cathode Ray Tubes (CRT) were used. RGB color space is an example of a relative color standard (as opposed to the absolute one, e.g., CIE 1931). The primary colors (R–red, G–green and B–blue) mimicked phosphor in CRT luminophore. The model uses additive color mixing to inform what kind of light needs to be emitted to produce a given color. The value of a particular color is expressed as a vector of three elements—intensities of three primary colors,× and a transformation to a different color space is expressed by a 3 3 matrix. Assume that values for each primary are quantized to $m^{-} = 2n$ values; let the highest - intensity value be $k = m$ 1; then (0, 0, 0) is black,

(*k, k, k*) is (television) white, (*k,* 0*,* 0) is 'pure' red, and so on. The value $k = 255 = 28 \quad 1$ is common, i.e., 8 bits per color channel. There are $2563 = 224 = 16,777,216$ possible colors in such a discretized space.

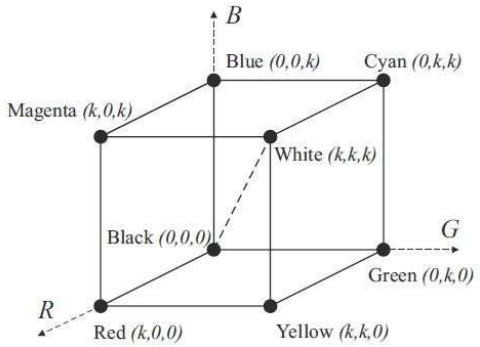


Figure 2.32: RGB color space with primary colors red, green, blue and secondary colors yellow, cyan, magenta. Gray-scale images with all intensities lie along the dashed line connecting black and white in RGB color space.

The RGB model may be thought of as a 3D co-ordinatization of color space (see Figure 2.32); note the secondary colors which are combinations of two pure primaries. There are specific instances of the RGB color model such as sRGB, Adobe RGB and Adobe Wide Gamut RGB, which differ slightly in transformation matrices and the gamut. One of the transformations between RGB and XYZ color spaces is

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.24 & -1.54 & -0.50 \\ -0.98 & 1.88 & 0.04 \\ 0.06 & -0.20 & 1.06 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix},$$
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41 & 0.36 & 0.18 \\ 0.21 & 0.72 & 0.07 \\ 0.02 & 0.12 & 0.95 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad (2.18)$$

The US and Japanese color television formerly used **YIQ** color space. The Y component describes intensity and I, Q represent color. YIQ is another example of additive

color mixing. This system stores a luminance value with two chrominance values, corresponding approximately to the amounts of blue and red in the color. This color space corresponds closely to the YUV color model in the PAL television norm (Australia, Eu-

rope, except France, which uses SECAM). YIQ color space is rotated 33◦ with respect to the YUV color space. The YIQ color model is useful since the Y component provides all that is necessary for a monochrome display; further, it exploits advantageous properties of the human visual system, in particular our sensitivity to **luminance**, the perceived energy of a light source.

The **CMY**—for Cyan, Magenta, Yellow—color model uses subtractive color mixing which is used in printing processes. It describes what kind of inks need to be applied so the light reflected from the white substrate (paper, painter's canvas) and passing through the inks produces a given color. CMYK stores ink values for black in addition. Black can be generated from C, M and Y components but as it is abundant in printed documents, it is of advantage to have a special black ink. Many CMYK colors spaces are used for different sets of inks, substrates, and press characteristics (which change the color transfer function for each ink and thus change the appearance).

HSV – Hue, Saturation, and Value (also known as HSB, hue, saturation, brightness) is often used by painters because it is closer to their thinking and technique. Artists commonly use three to four dozen colors (characterized by the hue; technically, the dominant wavelength). If another color is to be obtained then it is mixed from the given ones, for example, 'purple' or 'orange'. The painter also wants colors of different saturation, e.g., to change 'fire brigade red' to pink. She will mix the 'fire brigade red' with white (and/or black) to obtain the desired lower saturation. The HSV color model is illustrated in Figure 2.33.

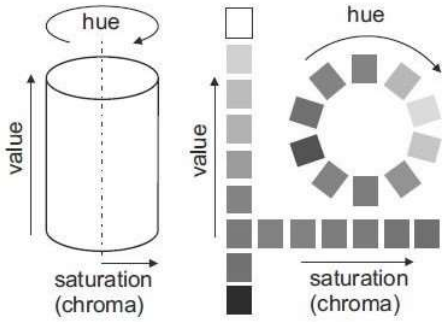


Figure 2.33: HSV color model illustrated as a cylinder and unfolded cylinder. © *Cengage Learning 2015. A color version of this figure may be seen in the color inset—Plate 3.*

HSV decouples intensity information from color, while hue and saturation corre- spond to human perception, thus making this representation very useful for developing image processing algorithms. This will become clearer as we proceed to describe image enhancement algorithms (for example, equalization Algorithm 5.1), which if applied to

each component of an RGB model would corrupt the human sense of color, but which would work more or less as expected if applied to the intensity component of HSV (leaving the color information unaffected). HSL (hue, saturation, lightness/luminance), also known as HLS or HSI (hue, saturation, intensity) is similar to HSV. 'Lightness' replaces 'brightness'. The difference is that the brightness of a pure color is equal to the brightness of white, while the lightness of a pure color is equal to the lightness of a medium gray.

Color Models

	spaces	
Colorimetric	XYZ	Colorimetric calculations
Device oriented, nonuniform spaces	RGB, UIQ	Storage, processing, coding, color TV
Device oriented, Uniform spaces	LAB, LUV	Color difference, analysis
User oriented	HSL, HIS	Color perception, computer graphics

Applications

Palette images:

Palette images (also called **indexed images**) provide a simple way to reduce the amount of data needed to represent an image. The pixel values constitute a link to a **lookup table** (also called a color table, color map, **palette**). The table contains as many entries as the range of possible values in the pixel, which is typically 8 bits 256 values. Each entry of the table ≡ maps the pixel value to the color, so there are three values, one for each of three color components. In the typical case of the RGB color model, values for red, green and blue are provided. It is easy to see that this approach would reduce data consumption to one-third if each of the RGB channels had originally been using 8 bits (plus size of the look up table).

Many widely used image formats for raster images such as TIFF, PNG and GIF can store palette images.

If the number of colors in the input image is less than or equal to the number of entries in the lookup table then all colors can be selected and no loss of information occurs. Such images may be cartoon movies, or program outputs. In the more common case, the number of colors in the image exceeds the number of entries in the lookup table, a subset of colors has to be chosen, and a loss of information occurs.

This color selection may be done many ways. The simplest is to quantize color space regularly into cubes of the same size. In the 8 bit example, there would be $8 \times 8 \times 8 = 512$ such cubes. If there is, e.g., a green frog in green grass in the picture then there will not be enough shades of green available in the lookup table to display the image well. In such a case, it is better to check which colors appear in the image by creating histograms for all three color components and quantize them to provide more shades for colors which occur in the image frequently. If an image is converted to a palette representation then the nearest color (in some metric sense) in the lookup table is used to represent the original color. This is an instance of **vector quantization** which is widely used in analyzing large multi-dimensional datasets. It is also possible to view the occupation by the pixels of RGB space as a **cluster analysis** problem, susceptible to algorithms such as k-means.

The term **pseudo-color** is usually used when an original image is gray-level and is displayed in color; this is often done to exploit the color discriminatory power of human vision. The same palette machinery as described above is used for this purpose; a palette is loaded into the lookup table which visualizes the particular gray-scale image the best. It could either enhance local changes, or might provide various views of the image. Which palette to choose depends on the semantics of the image and cannot be derived from image statistics alone. This selection is an interactive process.

Almost all computer graphics cards work with palette images directly in hardware. The content of the lookup table will be filled by the programmer.

Color constancy

Consider the situation in which the same surface is seen under different illumination, e.g., for a Rubik's cube in Figure 2.34. The same surface colors are shown fully illuminated and in shadow. The human vision system is able to abstract to a certain degree from the illumination changes and perceive several instances of a particular color as the same. This phenomenon is called color constancy. Of course, it would be desirable to equip artificial perception systems based on photosensors with this ability too, but this is very challenging.

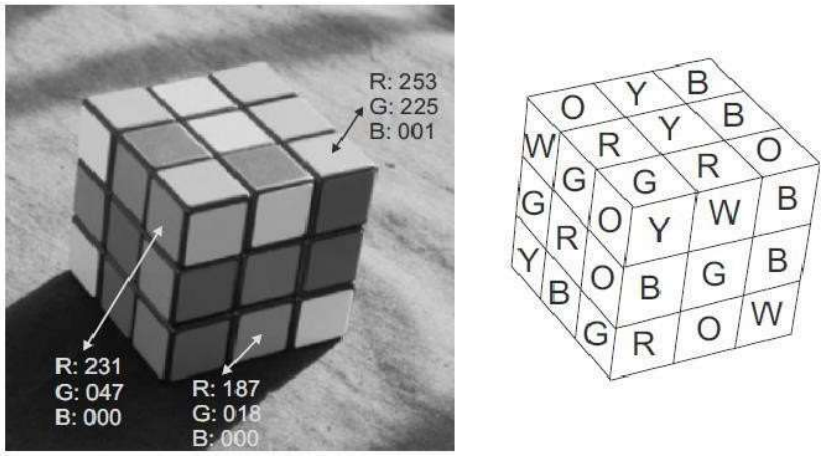


Figure 2.34: Color constancy: The Rubik cube is captured in sunlight, and two of three visible sides of the cube are in shadow. The white balance was set in the shadow area. There are six colors on the cube: R-red, G-green, B-blue, O-orange, W-white, and Y-yellow. The assignment of the six available colors to 3 9 visible color patches is shown on the right. Notice how different the same color patch can be: see *RGB* values for the three instances of orange. *© Cengage Learning 2015.* × *A color version of this figure may be seen in the color inset—Plate 4.*

Recall equation (2.16) which models the spectral response q_i of the i-th sensor by integration over a range of wavelengths as a multiplication of three factors: spectral sensitivity $R_i(\lambda)$ of the sensor $i = 1, 2, 3$, spectral density of the illumination $I(\lambda)$, and surface reflectance $S(\lambda)$. A color vision system has to calculate the vector q_i for each pixel as if $I(\lambda) = 1$. Unfortunately, the spectrum of the illuminant $I(\lambda)$ is usually unknown.

Assume for a while the ideal case in which the spectrum $I(\lambda)$ of the illuminant is known. Color constancy could be obtained by dividing the output of each sensor with its sensitivity to the illumination. Let qi' be the spectral response after compensation for the illuminant (called von Kries coefficients), $qi' = \rho_i \, q_i$, where

$$\rho_i = 1 \Big/ \int_{\lambda_1}^{\lambda_2} I(\lambda)\, R_i(\lambda)\, \mathrm{d}\lambda \,. \qquad (2.19)$$

Partial color constancy can be obtained by multiplying color responses of the three photosensors with von Kries coefficients ρ_i.

In practice, there are several obstacles that make this procedure intractable. First, the illuminant spectrum $I(\lambda)$ is not known; it can only be guessed indirectly from reflec tions in surfaces. Second, only the approximate spectrum is expressed by the spectral response q_i of the i-th sensor. Clearly the color constancy problem is ill-posed and cannot be solved without making additional assumptions about the scene.

Several such assumptions have been suggested in the literature. It can be assumed that the average color of the image is gray. In such a case, it is possible to scale the sensitivity of each sensor type until the assumption becomes true. This will result in an insensitivity to the color of the illumination. This type of color compensation is often used in automatic white balancing in video cameras. Another common assumption is that the brightest point in the image has the color of the illumination. This is true when the scene contains specular reflections which have the property that the illuminant is reflected without being transformed by the surface patch.

The problem of color constancy is further complicated by the perceptual abilities of the human visual system. Humans have quite poor quantitative color memory, and also perform color adaptation. The same color is sensed differently in different local contexts.

1.7. DATA STRUCTURES FOR IMAGE ANALYSIS

Data and an algorithm are the two essentials of any program. Data organization often considerably affects the simplicity of the selection and the implementation of an algorithm, and the choice of data structures is therefore a fundamental question when writing a program.

1.8 : LEVELS OF IMAGE DATA REPRESENTATION

The aim of computer visual perception is to find a relation between an input image and models of the real world. During the transition from the raw input image to the model, image information becomes denser and semantic knowledge about the interpretation of image data is used more. Several levels of visual information representation are defined on the way between the input image and the model; computer vision then comprises a design of the:

- **Intermediate representations** (data structures).
- **Algorithms** used for the creation of representations and introduction of relations between them.

The representations can be stratified in four levels [Ballard and Brown, 1982]—however, there are no strict borders between them and a more detailed classification of the representational levels is used in some applications. These four representational levels are ordered from signals at a low level of abstraction to the description that a human can perceive. The information flow between the levels may be bi-directional, and for some specific uses, some representations can be omitted.

1) The lowest representational level—**iconic images**—consists of images containing original data: integer matrices with data about pixel brightness. Images of this kind are also outputs of pre-processing operations used for highlighting some aspects of the image important for further treatment.
2) The second level is **segmented images**. Parts of the image are joined into groups that probably belong to the same objects. For instance, the output of the segmentation of a scene with polyhedra is either line segments coinciding with borders or two-dimensional regions corresponding to faces of bodies. It

is useful to know something about the application domain while doing image segmentation; it is then easier to deal with noise and other problems associated with erroneous image data.

3) The third level is **geometric representations** holding knowledge about 2D and 3D shapes. Quantification of a shape is very difficult but also very important. Geometric representations are useful while doing general and complex simulations of the influence of illumination and motion in real objects. We also need them for the transition between natural raster images acquired by a camera) and data used in computer graphics (CAD— computer-aided design, DTP— desktop publishing).
4) The fourth representational level is **relational models**. They give us the ability to treat data more efficiently and at a higher level of abstraction. A priori knowledge about the case being solved is usually used in processing of this kind. Artificial intelligence (AI) techniques are often explored; the information gained from the image may be represented by semantic nets or frames.

An example will illustrate a priori knowledge. Imagine a satellite image of a piece of land, and the task of counting planes standing at an airport; the a priori knowledge is the position of the airport, which can be deduced, for instance, from a map. Relations to other objects in the image may help as well, e.g., to roads, lakes, or urban areas. Additional a priori knowledge is given by geometric models of planes for which we are searching. Segmentation will attempt to identify meaningful regions such as runways, planes and other vehicles, while third-level reasoning will try to make these identifications more definite. Fourth-level reasoning may, for example, determine whether the plane is arriving, departing or undergoing maintenance, etc.

1.9. TRADITIONAL IMAGE DATA STRUCTURES

Traditional image data structures such as matrices, chains, graphs, lists of object properties, and relational databases are important not only for the direct representation of image information, but also as a basis for more complex hierarchical methods of image representation.

1.9.1. Matrices

A **matrix** is the most common data structure for low-level representation of an image. Elements of the matrix are integer numbers corresponding to brightness, or to another property of the corresponding pixel of the sampling grid. Image data of this kind are usually the direct output of the image-capturing device. Pixels of both rectangular and hexagonal sampling grids can be represented by a matrix. The correspondence between data and matrix elements is obvious for a rectangular grid; with a hexagonal grid every even row in the image is shifted half a pixel to the right.

Image information in the matrix is accessible through the co-ordinates of a pixel that correspond with row and column indices. The matrix is a full representation of the image, independent of the contents of image data—it implicitly contains **spatial relations** among semantically important parts of the image. The space is two-dimensional in the case of an image. One very natural spatial relation is the **neighborhood relation**. Some special images that are represented by matrices are:

- A **binary image** (an image with two brightness levels only) is represented by a matrix containing only zeros and ones.

- Several matrices can contain information about one **multispectral image**. Each single matrix contains one image corresponding to one spectral band.

- Matrices of different resolution are used to obtain **hierarchical image data structures**. Such hierarchical representations can be very convenient for parallel computers with the 'processor array' architecture.

Most programming languages use a standard array data structure to represent a matrix. Historically, memory limitations were a significant obstacle to image applications, but this is no longer the case.

There is much image data in the matrix. Algorithms can be sped up if global information is derived from the original image matrix first—global information is more concise and occupies less memory. We have already mentioned the most popular example of global information—the histogram. Looking at the image from a probabilistic point of view, the normalized histogram is an estimate of the probability density of a phenomenon: that an image pixel has a certain brightness.

Another example of global information is the **co-occurrence matrix** [Pavlidis, 1982], which represents an estimate of the probability of two pixels appearing in a spatial relationship in which a pixel (i_1, j_1) has intensity x and a pixel (i_2, j_2) has intensity y. Suppose that the probability depends only on a certain spatial relation r between a pixel of brightness x and a pixel of brightness y; then information about the relation r is recorded in the square co-occurrence matrix C_r, whose dimensions correspond to the number of brightness levels of the image. To reduce the number of matrices C_r, introduce some simplifying assumptions; first consider only direct neighbors, and then treat relations as symmetrical (without orientation). The following algorithm calculates the co-occurrence matrix C_r from the image $f(i, j)$.

Algorithm 4.1: Co-occurrence matrix $C_r(x, y)$ for the relation r

1. Set $C_r(x, y) = 0$ for all $x, y \in [0, L]$, where L is the maximum brightness.
2. For all pixels (i_1, j_1) in the image, determine all (i_2, j_2) which have the relation r with the pixel (i_1, j_1), and perform

$$C_r f(i_1, j_1), f(i_2, j_2) = C_r f(i_1, j_1), f(i_2, j_2) + 1 .$$

If the relation r is *to be a southern or eastern 4-neighbor of the pixel* (i_1, j_1)*, or identity*, elements of the co-occurrence matrix have some interesting properties. Diagonal elements of the matrix $C_r(k, k)$ are equal to the area of the regions in the image with brightness k, and so correspond to the histogram. Off-diagonal elements $C_r(k, j)$ are equal to the length of the border dividing regions with brightnesses k and j, $k\,j$. For instance, in an image with low contrast, the elements of the co-occurrence matrix that are far from the diagonal are equal to zero or are very small. For high-contrast images the opposite is true.

The main reason for considering co-occurrence matrices is their ability to describe texture: this approach is introduced in Chapter 15.

The **integral image** is another matrix representation that holds global image information [Viola and Jones, 2001]. It is constructed so that its values $ii(i, j)$ in the location (i, j) represent the sums of all the original image pixel-values left of and above (i, j):

$$ii(i,j) = \sum_{k \le i, l \le j} f(k,l)\,, \qquad (4.1)$$

where f is the original image. The integral image can be efficiently computed in a single image pass using recurrences:

Algorithm 4.2: Integral image construction

1. Let $s(i, j)$ denote a cumulative row sum, and set $s(i, -1) = 0$.
2. Let $ii(i, j)$ be an integral image, and set $ii(-1, j) = 0$.
3. Make a single row-by-row pass through the image. For each pixel (i, j) calculate the cumulative row sums $s(i, j)$ and the integral image value $ii(i, j)$ using

$$s(i, j) = s(i, j - 1) + f(i, j)\,, \quad (4.2)$$

$$ii(i, j) = ii(i - 1, j) + s(i, j)\,. \quad (4.3)$$

4. After completing a single pass through the image, the integral image ii is constructed.

The main use of integral image data structures is in rapid calculation of simple rectangle image features at multiple scales. This kind of features is used for rapid object identification (Section 10.7) and for object tracking (Section 16.5).

Figure 4.1 illustrates that any rectangular sum can be computed using four array references, and so a feature reflecting a difference between two rectangles requires eight references. Considering the rectangle features shown in Figure 4.2a,b, the two-rectangle features require only six array references since the rectangles are adjacent. Similarly, the three- and four-rectangle features of Figure 4.2c,d can be calculated using eight and nine references to the integral image values, respectively. Such features can be computed extremely efficiently and in constant time once the integral image is formed.

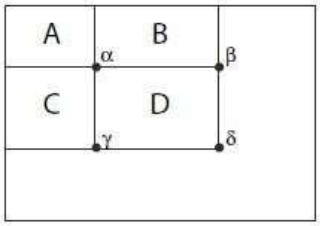


Figure 4.1: Calculation of rectangle features from an integral image. The sum of pixels within rectangle *D* can be obtained using four array references. $Dsum^{-} = ii(\delta) + ii(\alpha)$ $(ii(\beta) + ii(\gamma))$, where $ii(\alpha)$ is the value of the integral image at point α (and similarly for β, γ, δ). .

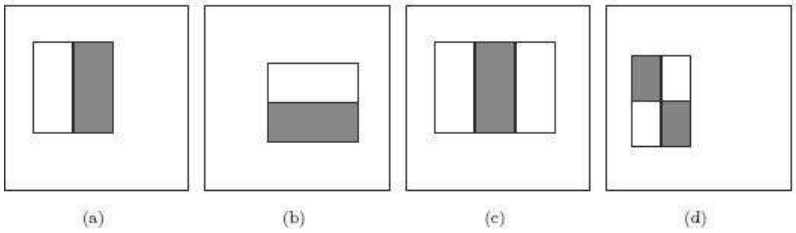


Figure 4.2: Rectangle-based features may be calculated from an integral image by subtraction of the sum of the shaded rectangle(s) from the non-shaded rectangle(s). The figure shows (a,b) two- rectangle, (c) three-rectangle, and (d) four-rectangle features. Sizes of the individual rectangles can be varied to yield different features as well as features at different scales. Contributions from the regions may be normalized to account for possibly unequal region sizes.

Chains

Chains are used for the description of object borders in computer vision. One element of the chain is a basic symbol; this approach permits the application of formal language theory for computer vision tasks. Chains are appropriate for data that can be arranged as a sequence of symbols, and the neighboring symbols in a chain usually correspond to the neighborhood of primitives in the image. The primitive is the basic descriptive element that is used in syntactic pattern recognition.

This rule of proximity (neighborhood) of symbols and primitives has exceptions—for example, the first and the last symbol of the chain describing a closed border are not neighbors, but the corresponding primitives in the image are. Similar inconsistencies are typical of image description languages, too. Chains are linear structures, which is why they cannot describe spatial relations in the image on the basis of neighborhood or proximity.

Chain codes are often used for the description of object borders, or other one-pixel-wide lines in images. The border is defined by the co-ordinates of its reference pixel and the sequence of symbols corresponding to the line of the unit length in several pre-defined orientations. Notice that a chain code is of a relative nature; data are expressed with respect to some reference point. Figure 4.3 shows an example of a chain code in which where 8neighborhoods are used—4-neighborhoods can be used as well.

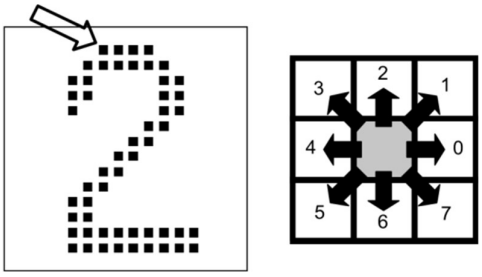


Figure 4.3: An example chain code; the reference pixel starting the chain is marked by an arrow: 00077665555556600000006444444442221111112234445652211.

If local information is needed from the chain code, then it is necessary to search through the whole chain systematically. For instance, if we want to know whether the border turns somewhere to the left by 90◦, we must just find a sample pair of symbols in the chain—it is simple. On the other hand, a question about the shape of the border near the pixel (i_0, j_0) is not trivial. It is necessary to investigate all chain elements until the pixel (i_0, j_0) is found and only then we can start to analyze a short part of the border that is close to the pixel (i_0, j_0).

The description of an image by chains is appropriate for syntactic pattern recognition based on formal language theory methods. When working with real images, the problem of how to deal with uncertainty caused by noise arises, which is why several syntactic analysis techniques with deformation correction have arisen [Lu and Fu, 1978]. Another way to deal with noise is to smooth the border or to approximate it by another curve. This new border curve is then described by chain codes [Pavlidis, 1977].

Run length coding has been used for some time to represent strings of symbols in an image matrix. For simplicity, consider a binary image first. Run length coding records only areas that belong to objects in the image; the area is then represented as a list of lists. Various schemes exist which differ in detail—a representative one describes each row of the image by a sublist, the first element of which is the row number. Subsequent terms are coordinate pairs; the first element of a pair is the beginning of a run and the second is the end (the beginning and the end are described by column coordinates). There can be several such sequences in the row. Run length coding is illustrated in Figure 4.4. The main advantage of run length coding is the existence of simple algorithms for intersections and unions of regions in the image.

Run length coding can be used for an image with multiple brightness levels as well—in this case sequences of neighboring pixels in a row that has constant brightness are considered. In the sublist we must record not only the beginning and the end of the sequence, but its brightness, too.

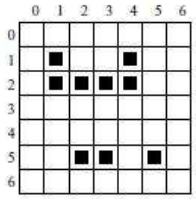


Figure 4.4: Run length coding; the code is ((1114)(214)(52355)).

Topological data structures

Topological data structures describe the image as a set of elements and their relations; these relations are often represented using graphs. A **graph** $G = (V, E)$ is an algebraic structure which consists of a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ and a set of arcs $E = \{ e_1, e_2,$

. . . , e_m } . Each arc e_k is incident to an unordered (or ordered) pair of nodes {v_i, v_j } which are not necessarily distinct. The *degree* of a node is equal to the number of incident arcs of the node.

A **weighted graph** is a graph in which values are assigned to arcs, to nodes, or to both—these values may, for example, represent weights, or costs.

The **region adjacency graph** is typical of this class of data structures, in which nodes correspond to regions and neighboring regions are connected by an arc. The segmented image consists of regions with similar properties (brightness, texture, color, . . .) that correspond to some entities in the scene, and the neighborhood relation is fulfilled when the regions have some common border. An example of an image with areas labeled by numbers and the corresponding region adjacency graph is shown in Figure 4.5; the label 0 denotes pixels out of the image. This label is used to indicate regions that touch borders of the image in the region adjacency graph.

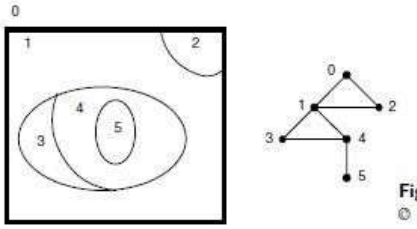


Figure 4.5: An example region adjacency graph.

The region adjacency graph has several attractive features. If a region encloses other regions, then the part of the graph corresponding with the areas inside can be separated by a cut in the graph. Nodes of degree 1 represent simple holes.

Arcs of the graph can include a description of relations between neighboring regions— the relations *to be to the left* or *to be inside* are common. It can be used for matching with a stored pattern for recognition purposes.

The region adjacency graph is usually created from the **region map**, which is a matrix of the same dimensions as the original image matrix whose elements are identification labels of the regions. To create the region adjacency graph, borders of all regions in the image are traced, and labels of all neighboring regions are stored. The region adjacency graph can also easily be created from an image represented by a quadtree (Section 4.3.2).

The region adjacency graph stores information about the neighbors of all regions in the image explicitly. The region map contains this information as well, but it is much more difficult to recall from there. If we want to relate the region adjacency graph to the region map quickly, it is sufficient for a node in the region adjacency graph to be marked by the identification label of the region and some representative pixel (e.g., the top left pixel of the region).

Construction of the boundary data structures that represent regions is not trivial, and is considered in Section 6.2.3. Region adjacency graphs can be used to approach region merging (where, for instance, neighboring regions thought to have the same image interpretation are merged into one region)—this topic is considered in Section 10.10. In particular, note that merging representations of regions that may border each other more than once can be intricate, for example, with the creation of 'holes' not present before the merge—see Figure 4.6.

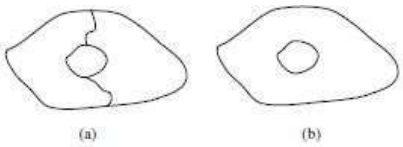


Figure 4.6: Region merging may create holes: (a) Before a merge. (b) After.

Relational structures

Relational databases [Kunii et al., 1974] can also be used for representation of information from an image; all the information is then concentrated in relations between semantically important parts of the image—objects—that are the result of segmentation. Relations are recorded in the form of tables. An example of such a representation is shown in Figure 4.7 and Table 4.1, where individual objects are associated with their names and other features, e.g., the top-left pixel of the corresponding region in the image. Relations between objects are expressed in the relational table. Here, such a relation is *to be inside*; for example, the object 7 (pond) is situated inside the object 6 (hill).

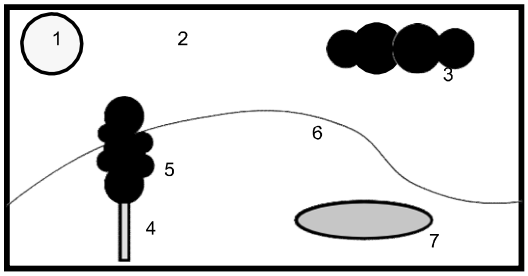


Figure 4.7: Description of objects using relational structure.

No.	Object name	Color	Min. row	Min. col.	Inside
1	sun	white	5	40	2
2	sky	blue	0	0	–
3	cloud	gray	20	180	2
4	tree trunk	brown	95	75	6
5	tree crown	green	53	63	–
6	hill	light green	97	0	–
7	pond	blue	100	160	6

Table 4.1: Relational table. © *Cengage Learning 2015.*

Description by means of relational structures is appropriate for higher levels of image understanding. In this case searches using keys, similar to database searches, can be used to speed up the whole process.

1.10 HIERARCHICAL DATA STRUCTURES

Computer vision is by its nature very computationally expensive, if for no other reason than the large amount of data to be processed. Usually a very quick response is expected because video real-time or interactive systems are desirable. One of the solutions is to use parallel computers (in other words brute force). Unfortunately there are many computer vision problems that are very difficult to divide among processors, or decompose in any way. Hierarchical data structures make it possible to use algorithms which decide a strategy for processing on the basis of relatively small quantities of data. They work at the finest resolution only with those parts of the image for which it is essential, using knowledge instead of brute force to ease and speed up the processing. We are going to introduce two typical hierarchical structures, pyramids and quadtrees.

Pyramids

Pyramids are among the simplest hierarchical data structures. We distinguish between **M-pyramids** (matrix-pyramids) and **T-pyramids** (tree-pyramids).

A **Matrix-pyramid** (M-pyramid) is a sequence $\{ M_L, M_{L-1}, \ldots, M_0 \}$ of images, where M_L has the same dimensions and elements as the original image, and M_{i-1} is derived from the M_i by reducing the resolution by one-half. When creating pyramids, it is customary to work with square matrices having dimensions equal to powers of 2—then M_0 corresponds to one pixel only.

M-pyramids are used when it is necessary to work with an image at different resolutions simultaneously. An image having one degree smaller resolution in a pyramid contains four times less data, so it is processed approximately four times as quickly.

Often it is advantageous to use several resolutions simultaneously rather than choose just one image from the M-pyramid. For such algorithms we prefer to use **tree-pyramids**, a tree structure.

Let $2L$ be the size of an original image (the highest resolution). A tree- pyramid (T-pyramid) is defined by:

1. A set of nodes $P = \{ p = (k, i, j)$ such that level $k \in [0, L]$; $i, j \in [0, 2k - 1] \}$
2. A mapping F between subsequent nodes P_{k-1}, P_k of the pyramid
$$F(k, i, j) = (k - 1, \text{floor}(i/2), \text{floor}(j/2)) .$$
3. A function V that maps a node of the pyramid P to Z, where Z is the subset of the whole numbers corresponding to the number of brightness levels, for example, $Z = \{0, 1, 2, \ldots, 255\}$.

Nodes of a T-pyramid correspond for a given k with image points of an M-pyramid; elements of the set of nodes $P = \{ (k, i, j) \}$ correspond with individual matrices in the Mpyramid—k is called the level of the pyramid. An image $P = \{ (k, i, j) \}$ for a specific k constitutes an image at the k^{th} level of the pyramid. F is the so-called parent mapping, which is defined for all nodes P_k of the T-pyramid except its root (0, 0, 0). Every node of

the T-pyramid has four child nodes except leaf nodes, which are nodes of level L that correspond to the individual pixels in the image.

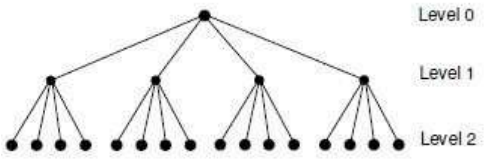


Figure 4.8: T-pyramid.

Values of individual nodes of the T-pyramid are defined by the function V. Values of leaf nodes are the same as values of the image function (brightness) in the original image at the finest resolution; the image size is $2L$. Values of nodes

in other levels of the tree are either an arithmetic mean of four child nodes or they are defined by coarser sampling, meaning that the value of one child (e.g., top left) is used. Figure 4.8 shows the structure of a simple T-pyramid.

The number of image pixels used by an M-pyramid for storing all matrices is given by

$$N^2\left(1+\frac{1}{4}+\frac{1}{16}+\ldots\right)\approx 1.33\,N^2, \qquad (4.4)$$

where N is the dimension of the original matrix (the image of finest resolution)—usually a power of two, $2L$.

The T-pyramid is represented in memory similarly. Arcs of the tree need not be recorded because addresses of the both child and parent nodes are easy to compute due to the regularity of the structure. An algorithm for the effective creation and storing of a Tpyramid is given in [Pavlidis, 1982].

Quadtrees

Quadtrees are modifications of T-pyramids. Every node of the tree except the leaves has four children (NW, north-western; NE, north-eastern; SW, south-western; SE, south-eastern). Similarly to T-pyramids, the image is divided into four quadrants at each hierarchical level; however, it is not necessary to keep nodes at all levels. If a parent node has four children of the same value (e.g., brightness), it is not necessary to record

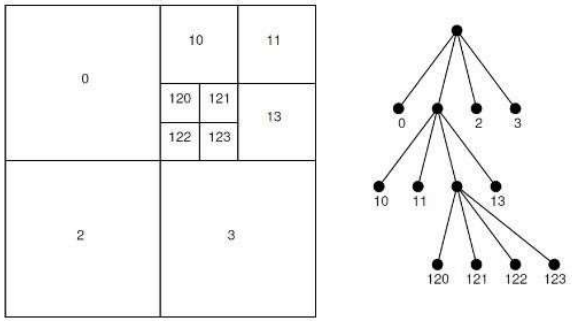


Figure 4.9: Quadtree..

them. This representation is less expensive for an image with large homogeneous regions; Figure 4.9 is an example of a simple quadtree.

An advantage of image representation by means of quadtrees is the existence of simple algorithms for addition of images, computing object areas, and statistical moments. The main disadvantage of quadtrees and pyramid hierarchical representations is their dependence on the position, orientation, and relative size of objects. Two similar images with just very small differences can have very different pyramid or quadtree representations. Even two images depicting the same, slightly shifted scene, can have entirely different representations.

These disadvantages can be overcome using a normalized shape of quadtree in which we do not create the quadtree for the whole image, but for its individual objects. Geometric features of objects such as the center of gravity and principal axis are used; the center of gravity and principal axis of every object are derived first and then the smallest enclosing square centered at the center of gravity having sides parallel with the principal axes is located. The square is then represented by a quadtree. An object described by a normalized quadtree and several additional items of data (co- ordinates of the center of gravity, angle of main axes) is invariant to shifting, rotation, and scale.

Quadtrees are usually represented by recording the whole tree as a list of its individual nodes, every node being a record with several items characterizing it. An example is given in Figure 4.10. In the item *Node type* there is information about whether the node is a leaf or inside the tree. Other data can be the level of the node in the tree, position in the picture, code of the node, etc. This kind of representation is expensive in memory. Its advantage is easy access to any node because of pointers between parents and children.

Node type
Pointer to the NW son
Pointer to the NE son
Pointer to the SW son
Pointer to the SE son
Pointer to the father
Other data

Figure 4.10: Record describing a quadtree node.

.

It is possible to represent a quadtree with less demand on memory by means of a **leaf code**. Any point of the picture is coded by a sequence of digits reflecting successive divisions of the quadtree; zero means the NW (north-west) quadrant, and likewise for other quadrants: 1-NE, 2-SW, 3-SE. The most important digit of the code (on the left) corresponds to the division at the highest level, the least important one (on the right) with the last division. The number of digits in the code is the same as the number of levels of

the quadtree. The whole tree is then described by a sequence of pairs—the leaf code and the brightness of the region. Programs creating quadtrees can use recursive procedures to advantage.

T-pyramids are very similar to quadtrees, but differ in two basic respects. A T- pyramid is a balanced structure, meaning that the corresponding tree divides the image regardless of the contents, which is why it is regular and symmetric. A quadtree is not balanced. The other difference is in the interpretation of values of the individual nodes. Quadtrees have seen widespread application, particularly in the area of Geographic Information Systems (GIS) where, along with their three-dimensional generalization *octrees*, they have proved very useful in hierarchical representation of layered data.

Other pyramidal structures

The pyramidal structure is widely used, and has seen several extensions and modi- fications. Recalling that a (simple) M-pyramid was defined as a sequence of images $\{ M_L, M_{L\,1}, \ldots, M_0 \}$ in which M_i is a 2 x 2 reduction of M_{i+1}, we can define the notion of a **reduction window**; for every cell c of M_i, the reduction window is its set of children in M_{i+1}, $w(c)$. Here, a *cell* is any single element of the image M_i at the corresponding level of pyramidal resolution. If the images are constructed such that all interior cells have the same number of neighbors (e.g., a square grid, as is customary), and they all have the same number of children, the pyramid is called **regular**.

A taxonomy of regular pyramids may be constructed by considering the reduction window together with the **reduction factor** λ, which defines the rate at which the image area decreases between levels;

$$\lambda \leq \frac{|M_{i+1}|}{|M_i|}, i = 0, 1, \ldots, L-1 \, .$$

In the simple case, in which reduction windows do not overlap and are $\times$ 2 2, we have λ = 4; if we choose to let the reduction windows overlap, the factor will reduce. The notation used to describe this characterization of regular pyramids is (*reduction win- dow*)/(*reduction factor*). Figure 4.11 illustrates some simple examples.

The reduction window of a given cell at level i may be propagated down to higher resolution than level $i + 1$. For a cell c_i at level i, we can write $w^0(c_i) = w(c_i)$, and then recursively define

$$w^{k+1}(c_i) = \bigcup_{q \in w(c_i)} w^k(q) \, ,$$

(4.5)

$^k(c_i)$ is the **equivalent window** that covers all cells at level i+k+1 that link to the cell c_i. Note that the shape of this window is going to depend on the type of pyramid—for example, an $n \times n/2$ pyramid will generate octagonal equivalent windows, while for an

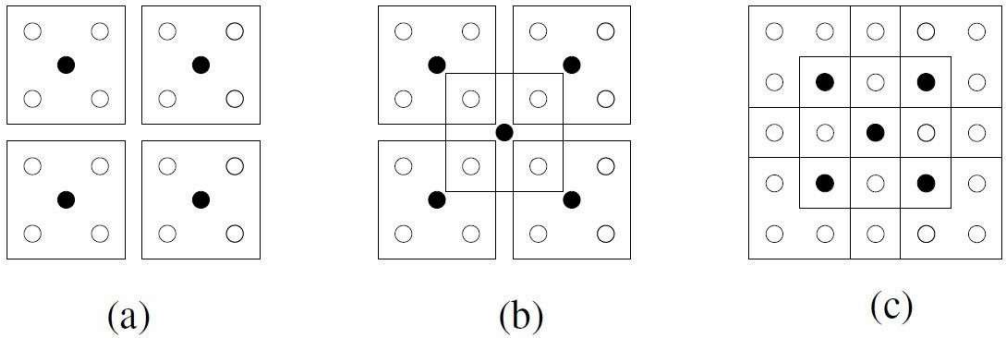


Figure 4.11: Several regular pyramid definitions.

(a) 2 x 2/4. (b) 2 x 2/2. (c) 3 x 3/2. (Solid dots are at the higher level, i.e., the lowerresolution level.)

nx n/4 pyramid they will be square. Use of non-square windows prevents domination of square features, as is the case for simple 2x 2/4 pyramids.

The 2 2/4 pyramid is widely used and is what is usually called an 'image pyramid';× the 2 2/2 structure is often referred to as an 'overlap pyramid'. 5 × 5/2 pyramids have× been used in compact image coding, where the image pyramid is augmented by a **Laplacian** pyramid of differences. Here, the Laplacian at a given level is computed as the per-pixel difference between the image at that level, and the image derived by 'expanding' the image at the next lower resolution. The Laplacian may be expected to have zero (or close) values in areas of low contrast, and therefore be amenable to compression.

Irregular pyramids are derived from contractions of graphical representations of images (for example, region adjacency graphs). Here, a graph may be reduced to a smaller one by selective removal of arcs and nodes. Depending on how these selections are made, important structures in the parent graph may be retained while reducing its overall complexity. The pyramid approach is quite general and lends itself to many developments—for example, the reduction algorithms need not be deterministic.