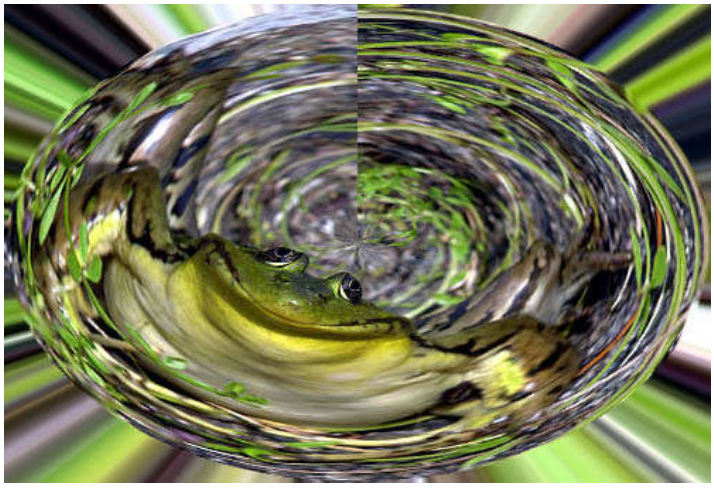
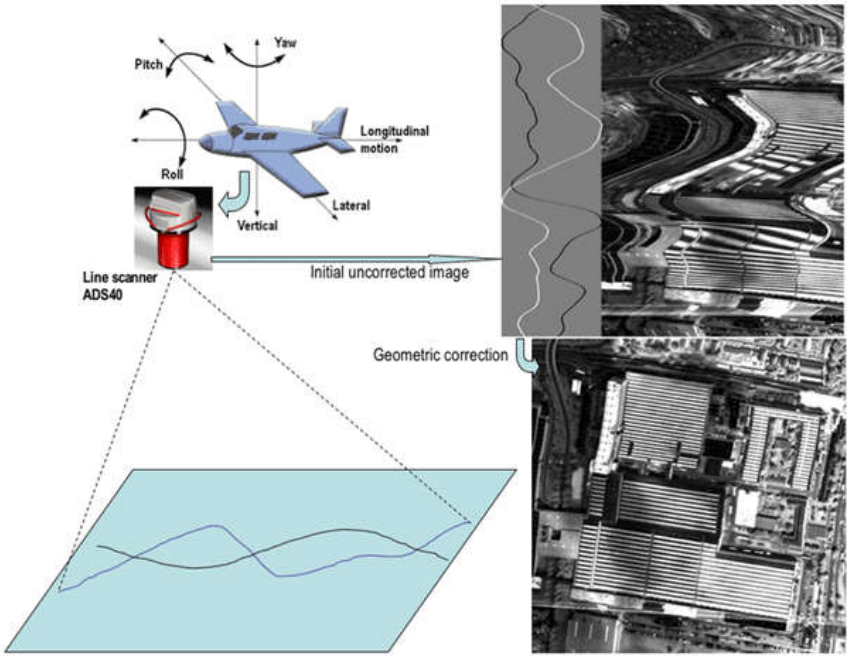
Geometric Operations

Geometric distortions

Image geometry appears in the form of spatial relationships between the pixels or groups of pixels. Geometric operations change these relationships by moving pixels to new locations while preserving to some extent pixel neighborhoods:

Geometric transformations are necessary if the imaging process suffers from some inherent **geometric distortions**. For instance, a high-resolution airborne line scanner, which sweeps each sensor across the terrain below (so called "pushbroom imaging") produces extremely distorted images due to changes in velocity, altitude, and attitude, i.e. yaw, pitch, and swing angles of the aircraft during the image acquisition:



Even if an image has no geometric distortion, its geometry may need some modification, e.g. to adjust an image of the Earth's surface to a certain map projection or to *register* two or more images of the same scene or object, acquired with different imaging devices or obtained from different viewpoints. **Image registration** pursues the goal of bringing common features in two or more images into coincidence.

Simple techniques for manipulating image geometry such as replication of each pixel to an $n \times n$ block of pixels to enlarge an image or subsampling (taking one pixel from each $n \times n$ block) to shrink an image have serious limitations due to information losses and Moire effects of subsampling as well as "blocky" appearance of images enlarged by pixel replication. The limitations become obvious when a subsampled image is enlarged to its previous size by simply replicating the sampled pixels - see, e.g. below the original and subsampled/enlarged images taken from `http://www.robots.ox.ac.uk/~improofs/super-resolution/super-res1.html`, or the aliasing due to the inadequate sampling resolution, or pixel density, which appears as a Moire pattern (the latter image pair is borrowed from from Wikipedia `http://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem`):


Information losses due to subsampling and subsequent block enenlargement


Aliasing due to the inadequate sampling resolution

Affine transformations

An arbitrary geometric transformation moves a pixel at coordinates (x,y) to a new position, (x',y'). The movement is specified by a pair of **transformation equations**:

$$x' = T_x(x,y); \quad y' = T_y(x,y)$$

being typically expressed as polynomials in x and y. Linear transformation equations specify an **affine transformation**:

$$x' = a_0x + a_1y + a_2; \quad y' = b_0x + b_1y + b_2$$

Images below from the HIPR (Hypermedia Image Processing Resource) web page of the Department of Artificial Intelligence, University of Edinburgh, UK, show results of an affine transformation.


Initial image

Affinely transformed image

The affine transformation has an important property that two successive affine transformations combine also into an affine transformation. If 2D points are represented in vectorial form, then the affine transformation consists of multiplication by a 2×2 matrix followed by addition of a vector; more convenient **homogeneous coordinates** representing 2D points as 3D vectors with the third dimension equal to 1 make it possible to represent the entire transformation as a 3×3 matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ b_0 & b_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_2 \\ b_2 \end{bmatrix} \quad \Leftrightarrow \quad \underbrace{\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ x \\ 1 \end{bmatrix}}_{\text{affine transformation in homogeneous coordinates}}$$

This representation proves the aforementioned property: if two affine transformations are applied one after another, their matrices are multiplied resulting in a combined matrix. Special cases of affine transformation are **translation**, **scaling**, **rotation**, and **shearing**:

Transformation	a_0	a_1	a_2	b_0	b_1	b_2
Translation by $(\Delta x,\Delta y)$	1	0	Δx	0	1	Δy
Uniform scaling by a factor s	s	0	0	0	s	0
Non-uniform scaling by factors s_x, s_y	s_x	0	0	0	s_y	0
Clockwise rotation through angle θ	cos θ	sin θ	0	−sin θ	cos θ	0
Horizontal shear by a factor h	1	h	0	0	1	0

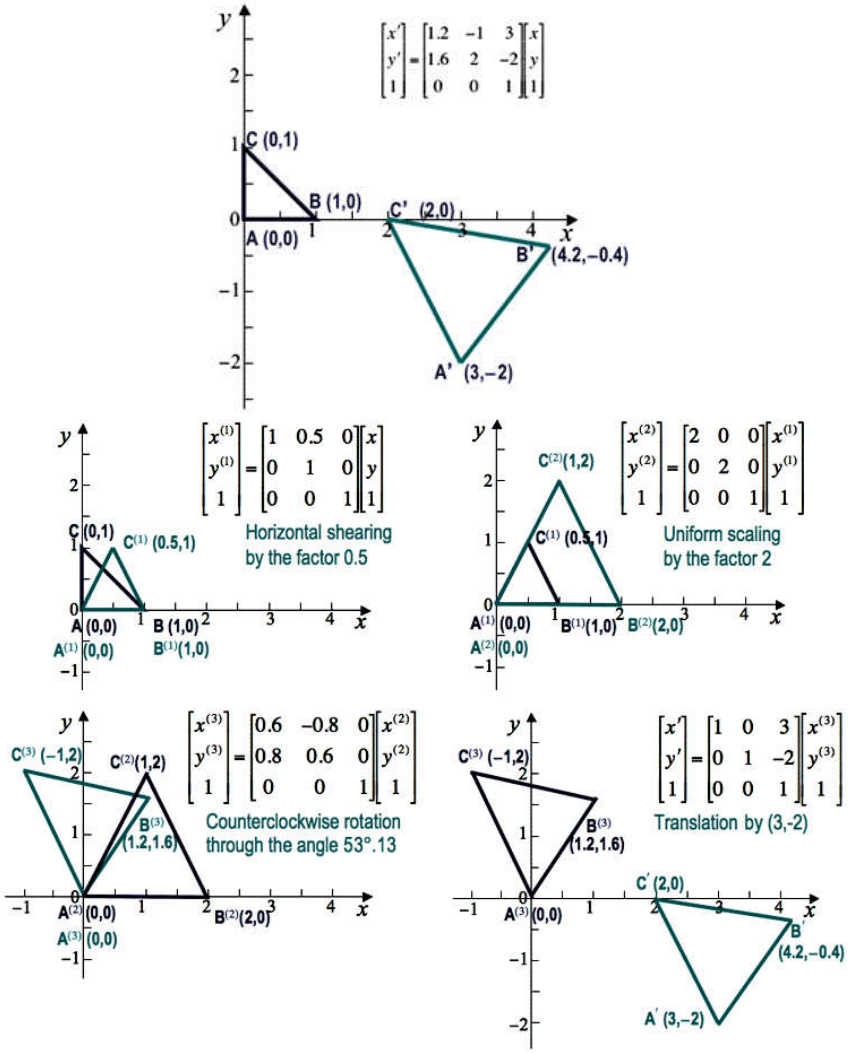
An *identity* transformation with $\Delta x=\Delta y=h=0$; $s=s_x=s_y=1$, $\theta=0$, i.e. $a_0=b_1=1$ and zero-valued other coefficients, does not change pixel positions. Given an arbitrary transformation T, an **inverse transformation** T^{-1} returns every transformed point to its initial position.

Combinations of translations and rotations only form so-called *Euclidean transformations* that preserve angles between lines and distances between points.

Because any combination of these special cases is also an affine transformation, any arbitrary affine transformation is usually expressed as some combination of these simpler processing steps, performed in sequence. Generally, such a sequence is more meaningful than direct specification of the transformation matrix: e.g. the four sequential steps of (*i*) horizontal shear by the factor 0.5, (*ii*) clockwise rotation through the angle $\theta = -53^o.13$ (so that $\cos\theta = 0.6$ and $\sin\theta = -0.8$), i.e. counterclockwise rotation through the angle $53^o.13$, (*iii*) uniform scaling by the factor 2, and (*iv*) translation by (3,−2) result in the following affine transformation matrix in homogeneous coordinates:

$$\underbrace{\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Translation by } (3,-2)} \underbrace{\begin{bmatrix} 0.6 & -0.8 & 0 \\ 0.8 & 0.6 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Rotation through } 53^\circ.13} \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Scaling by } 2} \underbrace{\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Shearing by } 0.5} = \begin{bmatrix} 1.2 & -1 & 3 \\ 1.6 & 2 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

This transformation maps a triangle ABC (A=(0,0), B=(1,0), C=(0,1)) onto the triangle A′B′C′ (A′=(3,−2), B′=(4.2,−0.4), C′=(2,0)):



The affine transformation is usually viewed as the mapping of one triangle onto another because the six transformation coefficients can be found by solving six simultaneous equations in x and y, given the coordinates of three points before and after transformation:

$$\underbrace{\begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Unknown affine coefficients}} \underbrace{\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Known 3 points before affine transformation}} = \underbrace{\begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Known 3 points after affine transformation}} \Rightarrow \underbrace{\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ x'_3 & y'_3 \end{bmatrix}}_{\text{System of six linear equations for the affine coefficients}}$$

An example of computing the coefficients for the above triangle-to-triangle mapping is as follows:

$$\underbrace{\begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Unknown affine coefficients}} \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Known 3 points before affine transformation}} = \underbrace{\begin{bmatrix} 3 & 4.2 & 2 \\ -2 & -0.4 & 0 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Known 3 points after affine transformation}} \Rightarrow$$

$$\underbrace{\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 4.2 & -0.4 \\ 2 & 0 \end{bmatrix}}_{\text{System of six linear equations for the affine coefficients}} \Rightarrow \underbrace{\begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 3 & -2 \\ 4.2 & -0.4 \\ 2 & 0 \end{bmatrix}}_{\text{Affine coefficients mapping one triangle onto another}} \Rightarrow$$

$$\underbrace{\begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 3 & -2 \\ 4.2 & -0.4 \\ 2 & 0 \end{bmatrix}}_{\text{Inversion of the asymmetric matrix (left inverted matrix)}} = \underbrace{\begin{bmatrix} 1.2 & 1.6 \\ -1 & 2 \\ 3 & -2 \end{bmatrix}}_{\text{Affine coefficients}} \Rightarrow \underbrace{\begin{bmatrix} 1.2 & -1 & 3 \\ 1.6 & 2 & -2 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Affine transformation matrix}}$$

This example uses a left inverse 3×3 matrix, $\mathbf{M}^{-1}$, for the asymmetric 3×3 matrix of the pixel coordinates, $\mathbf{M}$, such that $\mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$ where $\mathbf{I}$ is the 3×3 identity matrix. Although this formalism is convenient to specify the desired solution of the system of six equations, actually the system consists of the two linear subsystems, having the three unknowns coefficients each, that can be solved easily without such an inversion.

Forward / backward mapping

Any geometric transformation, including an affine transformation, can be implemented as forward or backward mapping. The **forward mapping** iterates over each pixel of the input image, computes new coordinates for it, and copies its value to the new location. But the new coordinates may not lie within the bounds of the output image and may not be integers. The former problem is easily solved by checking the computed coordinates before copying pixel values. The second problem is solved by assigning the nearest integers to x' and y' and using these as the output coordinates of the transformed pixel. The

problem is that each output pixel may be addressed several times or not at all (the latter case leads to "holes" where no value is assigned to a pixel in the output image).

The **backward mapping** iterates over each pixel of the output image and uses the *inverse* transformation to determine the position in the input image from which a value must be sampled. In this case the determined positions also may not lie within the bounds of the input image and may not be integers. But the output image has no holes.

Image interpolation

Any interpolation scheme convolves image data with an interpolation function giving grey levels for real-valued positions in an image.

Zero-order interpolation, or *nearest-neighbour interpolation* rounds real-valued coordinates calculated by a geometric transformation to their nearest integers. Let $x = T_x(x',y')$ and $y = T_y(x',y')$ be real coordinates of a point in the input image calculated with an inverse transformation T for backward mapping of the integer output position (x',y'). Then the nearest-neighbour interpolation copied to the latter position the value in the integer position ($x_{ZO} = \lfloor x-0.5 \rfloor$, $y_{ZO} = \lfloor y-0.5 \rfloor$) where $\lfloor z \rfloor$ denotes the integer upper bound of z, i.e. the minimum integer greater than or equal to z. The corresponding 1D interpolation function is: $h(z) = 1$ if $-0.5 \le z \le 0.5$ and 0 otherwise, and the 2D interpolation finction $h(x,y)$ is specified a separable product $h(x,y) = h(x)h(y)$.

Zero-order interpolation is simple computationally but input linear elements may become degraded in a transformed image, and the latter is too "blocky" after being scaled up in size by a large factor due to a discontinuous zero-order interpolation function.

First-order interpolation, or *bilinear interpolation* produces better visual appearance of the transformed image. An output pixel grey level is computed as a hyperbolic distance-weighted function of the four pixels in integer positions (x_0,y_0), (x_1,y_0), (x_0,y_1), and (x_1,y_1), surrounding the calculated real-valued position (x,y). Here, $x_1 = \lfloor x \rfloor$; $y_1 = \lfloor y \rfloor$; $x_0 = x_1 - 1$, and $y_0 = y_1 - 1$. Let $f_{\alpha\beta} = f(x_\alpha,y_\beta)$; $\alpha,\beta \in [0,1]$, be grey levels in the surrounding pixels. Let ($\Delta x = x - x_0$, $\Delta y = y - y_0$) denote the real-valued translation of the transformed position with respect to its integer rounding. Then first-order interpolation function is as follows:

$$\begin{aligned} f(x,y) &= f_{00}(1-\Delta x)(1-\Delta y) + f_{10}(1-\Delta x)\Delta y + f_{01}\Delta x(1-\Delta y) + f_{11}\Delta x\Delta y \\ &= f_{00} + [f_{10} - f_{00}]\Delta x + [f_{01} - f_{00}]\Delta y + [f_{11} - f_{10} - f_{01} + f_{00}]\Delta x\Delta y \end{aligned}$$

being the product of the two 1D linear interpolation functions $h(z) = 1 - |\Delta z|$ if $-1 \le z \le 1$ and 0 otherwise. It is continuous but has a discontinuous first derivative.

More visually appealing but far more complex computationally **third-order interpolation**, or *bicubic interpolation* convolves a 16×16 pixel neighbourhood with a continuous cubic function having a continuous derivative.

Warping and morphing

The transformation equations mapping (x,y) to (x',y') are generally expressed as polynomials in x and y, e.g. a **quadratic warp** with 12 coefficients

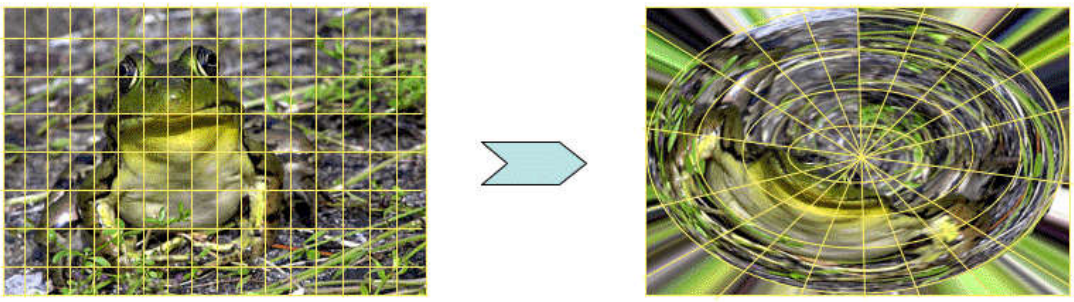
$$\underbrace{\begin{aligned} x' &= a_0x^2 + a_1y^2 + a_2xy + a_3x + a_4y + a_5 \\ y' &= b_0x^2 + b_1y^2 + b_2xy + b_3x + b_4y + b_5 \end{aligned}}_{\text{Forward mapping}} \Leftrightarrow \underbrace{\begin{aligned} x &= \tilde{a}_0(x')^2 + \tilde{a}_1(y')^2 + \tilde{a}_2x'y' + \tilde{a}_3x' + \tilde{a}_4y' + \tilde{a}_5 \\ y &= \tilde{b}_0(x')^2 + \tilde{b}_1(y')^2 + \tilde{b}_2x'y' + \tilde{b}_3x' + \tilde{b}_4y' + \tilde{b}_5 \end{aligned}}_{\text{Backward mapping}}$$

to introduce more complex geometric distortions than an affine transformation, or a cubic warp with 20 coefficients, etc. Polynomial warps are specified in practice by taking account of the effect they have on a set of **control points**. The quadratic warp can be determined by relating six control points before and after the transformation (this gives a system of 12 linear equations for the unknown 12 coefficients). Usually, more points than the minimum required for the determination of the warp are specified by detecting key image features and choosing their locations in the output image. For example, **image registration** may involve the search for the locations of characteristic features both in the reference image and an input image to be registered to the reference image by warping. In such a case, the system of equations for the warping coefficients is usually overdetermined and the coefficients of a warp that best fits the specified relationships (displacements between the input and output feature locations) are computed with a least-square technique.

In a **piecewise warping**, a control grid on the input image guides the warping. The grid, being a mesh of horizontal and vertical lines, covers the entire image with a set of quadrilaterals (rectangles). The intersections of the grid lines are control points being moved to new positions in the output (transformed) image. After the piecewise warping, every quadrilateral in the initial grid maps onto a quadrilateral in the warped grid. To uniquely determine the mapping, an 8-parameter **bilinear transformation** similar to the affine transformation but with an extra term in xy is used:

$$\underbrace{\begin{aligned} x' &= a_0xy + a_1x + a_2y + a_3 \\ y' &= b_0xy + b_1x + b_2y + b_3 \end{aligned}}_{\text{Forward mapping}} \Leftrightarrow \underbrace{\begin{aligned} x &= \tilde{a}_0x'y' + \tilde{a}_1x' + \tilde{a}_2y' + \tilde{a}_3 \\ y &= \tilde{b}_0x'y' + \tilde{b}_1x' + \tilde{b}_2y' + \tilde{b}_3 \end{aligned}}_{\text{Backward mapping}}$$

The four corner points of the corresponding quadrilaterals are sufficient to find the desired eight coefficients for each corresponding pair of the input and outpit quadrilaterals. Generally, the lines need not be only straight, e.g. the warping from a rectangular to a polar grid:



Morphing is an incremental transformation of one image into another. It combines the ideas of piecewise warping / registration and is used mostly for special TV, movie, or presentation effects rather than in image processing. At each step, a certain mesh is mapped into a transformed one, using an affine transformation to relate one triangle of the initial triangular mesh to the corresponding triangle of the goal mesh, or a bilinear transformation for the quadrilateral meshes. The basic difference from the piecewise warping is that the morphing computes the overall warp incrementally, as a sequence of smaller warps.

References

These lecture notes follow Chapter 9 "Geometric Operations" of the textbook

- Nick Efford. *Digital Image Processing: A Practical Introduction Using Java™*. Pearson Education, 2000.

with extra examples and teaching materials taken mostly, with corresponding references, from the Web.

Return to the local table of contents

Return to the general table of contents