Chapter 6
Transformations, Mappings, and Data Summaries

Ross Whitaker and Ingrid Hotz

Abstract Fundamentally, data visualization is the process of placing dabs of ink or color on a 2D plane. However, the complexity of data is increasing so that we see large numbers of instances, dimensions, parameters, etc. Such data surpasses what can readily be shown on a 2D or 3D display. One solution to this challenge is the development of better or more complex interfaces, that include, for instance, linked views, large displays, dynamic visualizations, and sophisticated user interactions. The alternative and complementary approach is to develop sets of mathematical and statistical tools to transform, map, or summarize data and thereby reduce its complexity so that visualization and understanding of large and complex becomes more feasible. The role of visualization research, in this case, is to identify common use cases and develop methods and tools that can readily be adapted to particular applications. To address the challenges of complexity in the data, previous works have proposed reducing items and attributes and associated visualization conventions and practices. Here we take deeper (and complementary) look at the analytical frameworks and approaches for transforming data into forms that are appropriate for display devices, considered generally. The approach in this chapter is to begin by characterizing different types of data in a way that is well suited for this discussion. We will then focus on a few particular classes of data and different ways of summarizing and transforming data of those types. Finally, we will broaden the discussion to other types of data and how they map into the various methodologies.

R. Whitaker (✉)
University of Utah, Salt Lake City, USA
e-mail: Whitaker@CS.Utah.edu

I. Hotz
Linköping University, Norrköping, Sweden
e-mail: Ingrid.Hotz@LiU.se

M. Chen et al. (eds.), *Foundations of Data Visualization*,
https://doi.org/10.1007/978-3-030-34444-3_6

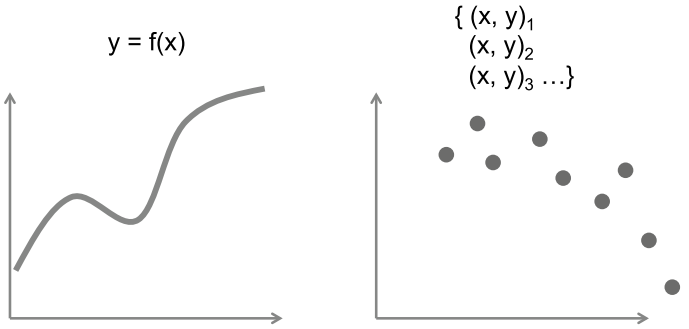


Fig. 6.1 In visualization, the placement of color or ink in 2D depicts instances that are generated from a process (scatterplot-right) or satisfy some constraint (graph of function-left)

6.1 Types of Data

Here we break different types of data down into a roughly hierarchical taxonomy and introduce some terminology and data properties that will facilitate the subsequent discussion. As we do so, we should bear in mind that this partitioning of data types and visualization goals is not unique and can be applied in different ways under different circumstances. As we shall see, even the same data set can be viewed as one type or another, depending on one's perspective or the goals of the analysis or visualization.

The first distinction to make is that of *instances* versus *mappings*. When visualizing *instances*, we are typically considering independent examples of data that share some common characteristics or sample space. Points in the x-y plane, shown as a scatterplot (each dot is an instance) in Fig. 6.1, are an example of a collection of instances. Alternatively, we are sometimes interested in visualizing a *mapping* that shows a relationship between two sets. A function $y = f(x)$, where $f : \Re \mapsto \Re$, is a special case of a mapping. Figure 6.1 shows the graph of a particular function $f(x)$. A graph of a function shares some properties with the scatterplot, because it shows, via ink on the page, all of the instances that satisfy the relation $y = f(x)$.

As we consider the distinction between instances and mappings, we should note that for a particular data set the difference may be in how we think about the data or the goals of the visualization. A discrete sampling of n points from of a function $y = f(x)$ could also be considered a set of instances, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, but in general we are interested in different questions about these two cases. In the case of instances, there is typically an assumption (explicit or not) that these instances are generated from some kind of stochastic process or probability distribution, and one would like to understand, in geometric terms, the relative densities of data and the relationships between points. With functions, the structure of the independent axis (here the x-axis) is given, and we typically want to interrogate the geometric structure of $y = f(x)$, rather than densities and distances.

x	y	z
5.2mm	3.1mm	7.5mm
1.4mm	3.2mm	6.3mm
2.6mm	3.4mm	5.1mm
⋮		

(a)

Name	Age	City	Status
Kim	32	St. Louis	member
Samir	27	San Francisco	nonmember
Hari	45	Salt Lake City	senior member
⋮			

(b)

Fig. 6.2 Data sets often consist of lists or arrays of *instances*, where the data for each instance may exist in a physical space with commensurate quantities (**a**), or may consist of heterogeneous quantities (**b**)

Another consideration for types of data is whether or not the data has an inherent *structure*. This distinction is most important in the case of instances. Some data sets consist of instances (or *points*) that have a consistent structure, e.g, each instance consists of values derived from a common set of *fields*. Each field might consist of a numerical value from a discrete or continuous space, a categorical value, or an ordinal value (ordered, but not quantitative). Often, structured data is defined in terms of a *data model*, where the model describes the structure of the fields, the possible values they can take, and their physical or semantic meaning. Figure 6.2 shows two examples of structured data sets. The first is a small set of records that contain points in three dimensions, and thus the attributes are all similar (e.g., same units, meters in this case) and where the space (three dimensions, $\Re^3$) has a special structure. The second example is a *heterogeneous* mixture of attributes—but where each record still contains the same attributes.

Alternatively, some data sets consist of *unstructured* data. In the case of unstructured data, instances each contain some set of data, but the data is not consistently organized into distinct fields with well-defined values, as it is in Fig. 6.2. Unstructured data is commonly text-heavy, but it often also contains other nontext data such as dates, numbers, and categorical attributes. A typical case of unstructured data comes in the form of free-form text, which one might see in online/electronic reviews, notes taken by a doctor/clinician in a medical exam, or other electronic communication, such text messages or email.

In the context of visualization, the type of data becomes important, because ultimately, visualization deals with the problem of how to assign colors to pixels in a 2D (or 3D) display. The choices of colors and where to put them are quantitative decisions; pixels are associated with 2D coordinates and colors are chosen from a multi-dimensional color palate. Similar decisions of placement, size, and color are important even when one is dealing with conventional visualization techniques such as graphs, glyphs, and various kinds of charts. *Virtually all data visualization strategies require one to represent instances or functions with relatively few quantities.*

In many or most cases, the data does not come in a form that maps directly onto these quantities, and typically a transformation from the original data into the desired visualization scheme is required. Even when the data lends itself to direct mapping

into a 2D domain, as in the case of 2D, scalar fields (or images), one is often interested in some particular property of the data, rather than the entire data set, and this often entails some kind of transformation of the data to produce a set of relevant features.

6.2 Functions

Here we begin with a very brief overview of the transformations that are relevant for 2D, 3D, and high-dimensional functions. Please note that in the context of scientific visualization, such functions are also often called fields (Chap. 5). We do not discuss particular algorithms for fast or efficient rendering of such functions, but focus on the mathematics of the transformations. For functional data, we are considering mappings from $\Re^m$ to $\Re^k$, and we assume that k is relatively small. We also treat these objects, unless otherwise stated, as continuous mappings (e.g., the domain is continuous) and assume that discrete representations are suitably interpolated, as in Chap. 5, such that they are defined over continuous domains.

There are some trivial examples, such as $m = k = 1$, where one can simply graph the function to see its structure. Also, for $m = 2$ and $k = 1$, one can use the pixels on the 2D viewing plane to assign color values to points, thus treating the function as an *image*, and we can use the notation $f(x, y)$ to denote the values at each 2D point. While there are many interesting and important questions about displaying such scalar data using various *color maps*, it is a topic that is studied extensively in the literature [55]. It is also possible to graph such data as height-fields in 3D space and project the resulting surfaces on a 2D screen.

The topic of *transforming* 2D, scalar data for better visualization or interpretation is (or *was*) covered extensively in the field of image processing [13]. Here we only mention a few basic ideas. One of the main strategies is to transform the range with a function $g : \Re \mapsto \Re$, so that we obtain a new image, $f'(x, y) = g(f(x, y))$. Of course, $g(\cdot)$ could also be $g : \Re \mapsto \Re^3$ and thereby represent the operation of color mapping scalar values in a function.

Understanding such transformations entails studying the structure of $g(\cdot)$. Typical mappings will lighten or darken images. Another common operation is to increase or decrease the overall range of a function. For enhancing the contrast in images, often it is advisable to consider the histogram of values of the image (histogram of values in the range). There are a variety of methods for flattening histograms (e.g. histogram equalization), or targeting or matching certain histograms [13].

For visualization, a more challenging example is $k = 1$, and $m = 3$, where we have scalar values given in a 3D volume $f(x, y, z)$. This kind of data arises, for instance, in medical imaging in the case of MRI or CT or in physical simulations, e.g., of temperature fields. The challenge with volume data is that the dimensionality does not lend itself to direct display of the raw data. Graphs of such functions would require 4D displays and a direct display of values as colors would require a 3D display. Thus, mappings onto 2D grids or displays are important. One approach is to provide some *slicing* capability, often arranged along the grid axes by fixing one coordinate, for

instance, to the kth slice. The resulting function $f'(u, v) = f(x = u, y = v, z = k)$ is defined over a 2D domain. More generally, arbitrary, 2D surfaces can be sampled from the 3D domain and then be displayed as (flat) images or rendered as texture-mapped surfaces, illustrated in Fig. 6.4a.

More commonly, 3D functions are rendered after some kind of *projection* onto a 2D viewing plane. Typically, the projection is a line integral following a ray from each point in the view plane into the 3D volumes, as illustrated in Fig. 6.3. The simplest case is:

$$f'(u, v) = \int f(u, v, \alpha) d\alpha, \tag{6.23}$$

which is a projection along the z axis to form a 2D function, which is then mapped onto pixel/display intensities. The specific bounds for the integration are a visualization decision, where the bounds should include some finite viewing frustrum. Other views can be obtained by applying a coordinate transformation, $\phi : \Re^3 \mapsto \Re^3$ (which should probably smooth and invertible),

$$f'(u, v) = \int f(\phi(u, v, \alpha)) d\alpha. \tag{6.24}$$

The transformation ϕ could include rotations and translations, but also could encode a perspective projection, or even nonlinear *curves* through the volume, effectively warping the 3D data. In the discussion that follows, we will leave off this coordinate transformation for simplicity (and without a loss of generality).

Another simple projection of 3D functions that is useful is, for instance, the maximum intensity projection, which takes the maximum value of $f(\cdot)$ along the rays associated with each pixel (along the z direction in this case),

$$f'(u, v) = \sup_{\alpha} f(u, v, \alpha). \tag{6.25}$$

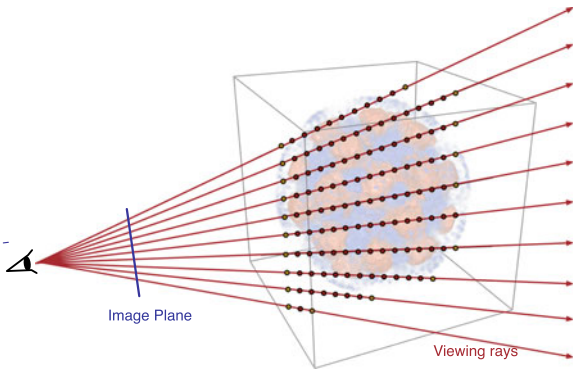


Fig. 6.3 3D functions or volumes are often transformed into 2D functions by accumulating data along rays that intersect the volume and a viewing plane (in blue)

The field of *volume rendering* [10] addresses various levels of complexity for these kinds of projections. A typical formulation of the volume rendering equation is (Fig. 6.4):

$$f'(u, v) = \int G\,(u, v, \alpha,\, f(\cdot), Df(\cdot), \ldots, O(\cdot))\, d\alpha, \tag{6.26}$$

where the "$\cdot$" notation indicates volume coordinates (u, v, α). The *occlusion function* $O(u, v, \alpha)$ quantifies how much a point contributes to the rendering. It also is a line integral (from the viewing plane to the 3D point) depending on the opacity of the volume that lies between the point and the viewing plane. The range of $G(\cdot)$ is mostly a 3D color space. The positional information, u, v, α, can also be used for view-dependent lighting effects. The first derivatives of f, denoted Df, indicates the local gradient vector, which provides normals, for lighting/shading, or edge enhancement in volumes, which are characterized by high gradients. Many other parameters have been considered in the function $G(\cdot)$, indicated by the ellipses $(\ldots)$, for instance higher order derivatives of $f(\cdot)$ [25]. The mapping of values of $f(\cdot)$ and its derivatives into colors and opacities is called a *transfer function*. The transfer function defines which parts of the data will be visible in the final rendering and essentially contributes to a good visualization result [32].

The integral in Eq. 6.26 describes many of the most basic options for high-quality volume rendering. Research beyond this basic formulation has focused on fast methods for volume rendering, e.g., on specialized hardware [49] and more *realistic* models and volume illumination. Early work focuses on methods for efficiently approximating light transport by restricting the type and number of light sources, e.g., the seminal method by Kniss et al. [26]. Deep shadow maps by Hadwiger et al. [16] enable complex lighting models in interactive direct volume rendering

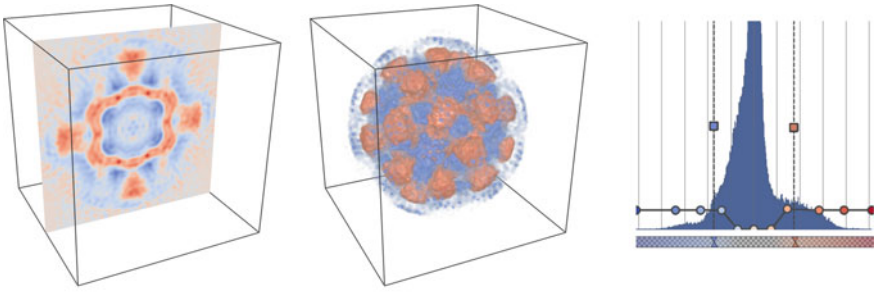


Fig. 6.4 Volume visualization of an electron microscopy data set of a feline calicivirus. **a** A slice through the data sets shows the entire data range in the respective slice. **b** Volume rendering using ray casting highlights selected scalar values in the data set. **c** Transfer function used for the volume rendering, overlaid with the data-histogram. The scalar values are mapped to the x-axis, and the y-axis shows the transparency assigned to the scalar values. Visualization: Martin Falk, https://inviwo.org

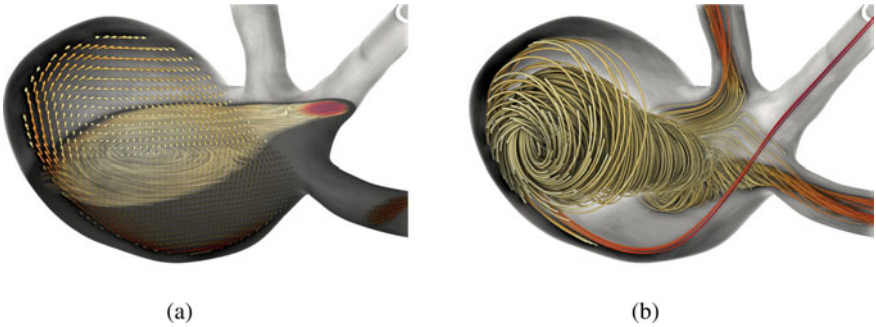


Fig. 6.5 Visualization of the blood flow in an aneurysm. **a** Vector-like glyphs represent the flow on a vertical slice through the aneurysm. A texture shows the flow an a horizontal slice through the aneurysm. **b** A selected set of streamlines illustrates the overall flow behavior. Visualization: Wito Engelke

(DVR). Early approaches aiming at full global illumination include the work by Hernell et al. [18]. More recently, volumetric illumination with multiple scattering based on photon mapping and Monte Carlo ray tracing has been introduced [20]. For a fuller account of the development of illumination in DVR, see the survey by Jönsson et al. [21].

For domains of higher-dimension, e.g., $m = 4$, the situation becomes even more challenging. If the dimensions are space and time, as is often the case, then there is a natural mapping into a dynamic visualization (e.g. a cine of a 3D rendering). For other situations, visualizations often depend on application-dependent choices of 2–3 coordinates to render, with some interaction or dynamics to convey the behavior across other coordinates.

For functions with higher-dimensional range $k > 1$, there are several approaches, with some depending on the specific application. An example from imaging is color images for which extensions for volume rendering exist [11]. Another special case is that of *vector fields*, where commonly $m = k \in \{2, 3\}$, and the domain and range are the same space. Direct visualization using vector-like glyphs is often feasible. However, such representations easily suffer from clutter or miss important details of the data. An alternative strategy is to map the vector field onto a scalar quantity, such as the magnitude (or length) of the vector or its orientation. Most commonly used methods are integration-based. They represent a set of lines (e.g., streamlines) following the vector field through the domain or generate textures conveying the directional properties of the field [37]. Figure 6.5 shows some examples of basic vector field visualizations. More advanced vector field visualization methods have been motivated through the task of *flow analysis* with very domain-specific demands. The analysis includes questions related to material transport and characteristic flow structures, as vortices, which are often expressed by derived scalar fields which will be discussed in the next section that discusses the *features*.

Another example is that of *tensor fields*, which often arise from physical processes, such as diffusion [2] or mechanical deformations and stresses [29]. The most direct visualization of tensors is displaying glyphs (e.g., [23, 28]) in selected

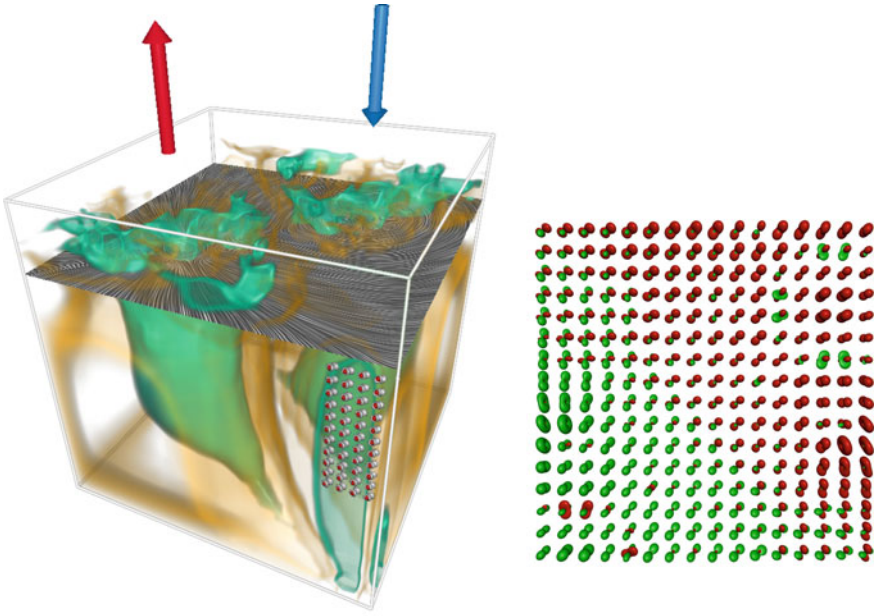


Fig. 6.6 Basic visualization of a stress tensor field of a solid block with one pushing and one pulling force. **a** Hybrid visualization: volume rendering of a derived scalar field, here an anisotropy measure, a slice with a texture highlighting the principal stress directions and glyphs in a selected region. **b** A slice showing glyphs (Reynolds glyphs) displaying the entire tensor information in selected locations. Visualization: Jochen Jankoway, https://inviwo.org

positions. Glyphs represent the entire tensor information but are limited to low resolution. Continuous visualization methods entail the extraction of scalar values from the tensors, such as tensor magnitudes, eigenvalues, anisotropy, or orientations of eigenvectors. Tensor lines following the main eigenvector direction or textures are used [19] to emphasize the directional character of the tensors. Most commonly used visualizations are hybrid methods combining glyphs with textures and volume rendering of scalar fields [27]. Figure 6.6 shows an example of some basic tensor field visualizations.

More advanced methods consider physically derived fields of tensors or vectors, which often resemble derivatives in their mathematical structure and as such are invariants (e.g., to coordinate transformations), of these objects are particularly interesting, as described in the next section.

There is some work on more general, higher-dimensional transfer functions. These would typically be defined with user input and require effective controls and interfaces, e.g., [35]. The other option is to perform dimensionality reduction on this data, treating the collection of pixel positions in the range as *instance data*, and using some of the techniques in Sect. 6.4 to find lower-dimensional proxies for the data in the

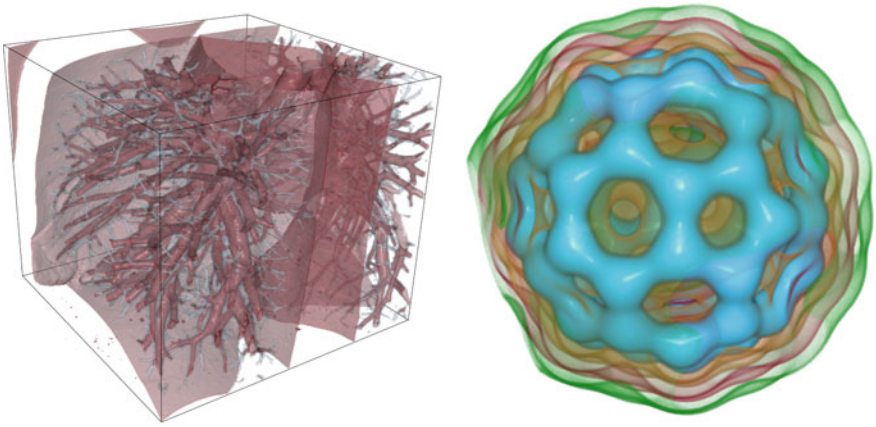


Fig. 6.7 Isosurface rendering. **a** CT imaging of a human lung, isosurfaces for two different densities emphasizing the vessel structure in the lung; **b** nested isosurface for a Fullerene molecule. Visualization: Martin Falk, https://inviwo.org

range. One can also deal with such data by visualizing a lower-dimensional *feature* extracted from the function, rather than the function directly; this is the topic of the next section.

6.3 Extracting Features from Functions

Often, functions are best understood in terms of specific structural attributes, rather than a description or depiction of the complete function. These special, or meaningful, attributes of a function often consist of subsets of the domain and are referred to as *features*. They can come in the form of points, curves, surfaces, or regions in the image domain. They sometimes include attributes associated with the original function data.

Perhaps the most common or prevalent derived feature associated with the visualization of functional data is *isocontours* (*isosurfaces* for 3D domains), also called *level sets*. In 2D, these contours can help show qualitative features such as high and low points (e.g., their locations and shapes), as well as ridges and valleys. In 3D, these features form surfaces, which allow the use of 3D rendering tools associated with graphics conventions and protocols to facilitate their display. Figure 6.7 shows two examples of level set visualization.

Mathematically, the specification of level sets of functions is stated as a subset of the domain that satisfies a constraint. In 3D, for the kth level set (or isosurface), we have

$$\mathscr{S} = \{(x, y, z) \in \mathscr{D} | f(x, y, z) = k\}, \tag{6.27}$$

which means that the isosurface is the set of points in the domain of f such that the function evaluates to k at those points. Often, when discussing level sets of functions, we consider for simplicity only the zero sets of a function, with the understanding that the kth level set of $f(\cdot)$ is the zero level set of $f'(\cdot) = f(\cdot) - k$. Often, we only consider functions that are considered *generic*, which means that the functions have nonzero derivatives almost everywhere and that the level sets follow certain structures. Level sets in any dimension have several important properties:

- Level sets are closed, except at the boundaries of the domain.
- If f is smooth and generic, level sets are smooth almost everywhere.
- Level sets of different values of k cannot cross and they are nested (enclose each other) according to the values of k, Fig. 6.7b.

Figure 6.9 depicts the general structure of the level sets and the particular examples of singularities for 2D domains.

This focuses on mathematical transformations, rather than specific numerical algorithms, but here we mention that extraction and representation of level sets from functions is itself an important consideration. The most common way to represent level sets is to construct a mesh of simplicies, which are edges in 2D and triangles in 3D. These discrete geometric objects are typically computed from a continuous representation of $f(\cdot)$. There are several strategies. The most common approach is to cover the domain of $f(\cdot)$ with a regular background grid (for instance, squares or cubes) and to identify the cells where the level intersects the boundaries of those cells. From those intersections, the algorithm typically infers some connectivity to insert simplices within the cell that appropriately intersect the cell boundaries. For instance, in 2D, the 2D grid lines intersect the 2D level sets at points, and line segments are used to connect those points within the cell according to a case table (as in Fig. 6.8). For 3D domains, the conversion of cubic or hexahedral intersections with isosurfaces into small patches of a triangular mesh forms the underlying machinery of the marching cubes algorithm [33].

Some other methods for identifying and representing level sets are: placing mesh vertices in cells/cubes that are adjacent to level sets and deforming the resulting mesh onto the level set [56]; placing systems of particles or points near level sets and attracting them to the level set [38]; and finding level-set points and growing surface representations outward from such points [30].

When one varies the value of k, the result is a family of level sets of f, parameterized by k. One can study or visualize the behavior of these sets as f continuously varies. The sets can split, merge, disappear, or appear with different values of k. These behaviors are well defined and these events, combined with the nesting structure of the levels sets, have led to a family of visualization algorithms that represent functions as the family of nested level sets and the *special* events that occur when these sets exhibit isolated, not smooth, behaviors such as merging or splitting, as illustrated in Fig. 6.9.

Beyond level sets of $f(\cdot)$, it is also helpful to consider the derivatives of $f(\cdot)$. Here we use the notation D to represent the set of partial derivatives in vector/tensor format, so that in 3D we have:

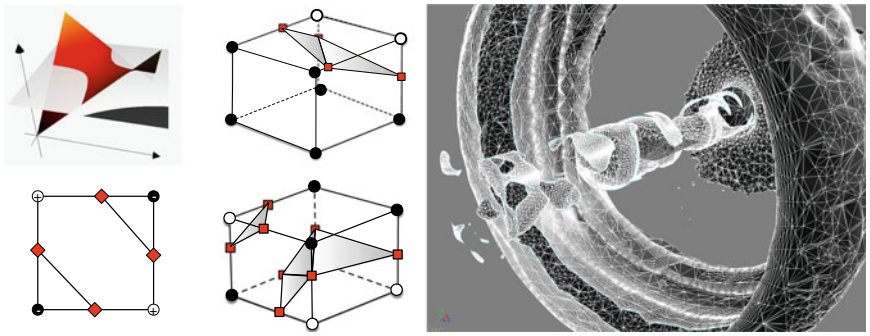


Fig. 6.8 Marching cubes algorithm for contour and isosurface computation. **a** Height field of 2D scalar field over one cell and its discrete approximation using lines. **b** Examples of two case for the approximation of isosurface using triangular simplicies. **c** Example of a mesh resulting from a marching cubes computation

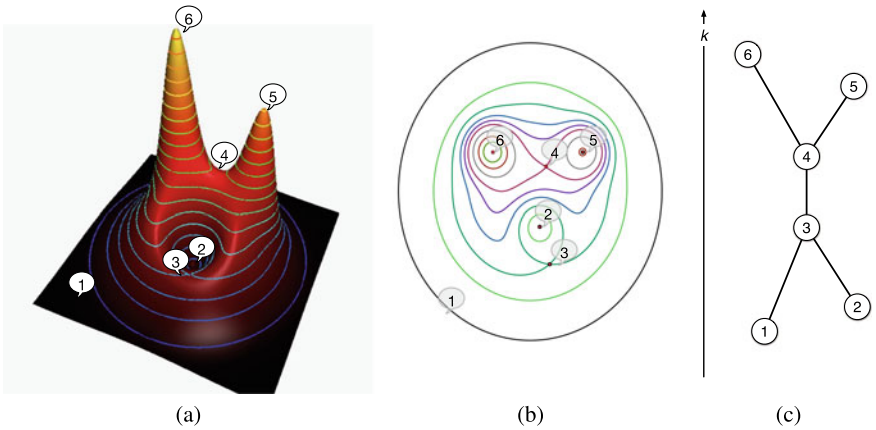


Fig. 6.9 Level sets or isocontours of a 2D analytical data set. **a** Displayed as a height field over the domain; **b** nested contours are shown in the domain; the red dots show the points where the gradient is zero. In maxima and minima, the contours degenerate to points. In saddle points, contours merge or split. **c** The contour tree tracks the changes of the contours when changing the isovalue k

$$Df = \begin{pmatrix} c\frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} \text{ and } \mathrm{D^2f} = \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial^2 y} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial x \partial z} & \frac{\partial^2 f}{\partial y \partial z} & \frac{\partial^2 f}{\partial^2 z} \end{pmatrix} \tag{6.28}$$

The magnitude or norm of Df, denoted $||Df||$, is a conventional measure of the contrast a point in the image domain. It is sometimes used to filter level sets of f, so that we can restrict the set to include only those locations in a region that have sufficiently high contrast:

$$\mathscr{S}_{\mathscr{T}} = \{(x, y, z) \in \mathscr{D} | f(x, y, z) = k \text{ and } ||\text{Df}|| \geq \text{T}\}. \tag{6.29}$$

The derivatives of f also help to define the extrema or singularities of f, which are the set of points:

$$\mathscr{E} = \{(x, y, z) | Df(x, y, z) = 0\}, \tag{6.30}$$

where the comparison with zero indicates that all partial derivatives are zero. This operation can be used to produce a set of singularities that can be viewed as an intersection of level sets. That is, the zero-crossing of each partial derivative produces a level set (on a derivative), or isosurface, and the intersections of those isosurfaces (there are 2 in 2D and 3 in 3D) consist of points that represent the singularities.

These extremal points come in several different forms, depending on the dimension of the domain. In 2D, there are minima, maxima, and saddle points. One can categorize these by examining the eigenvalues of the matrix $D^2 f = DD^T f$ (at every point), which is also called the *Hessian* of f, see also Chap. 5.

When considering or computing features on functions using derivatives, one must keep in mind how these features transform under some basic operations. For instance, the choice of axes for independent coordinates is often arbitrary (e.g., spatial coordinates), and therefore one would expect the features not to depend on that choice. For this reason, we often consider *differential invariants*, that is, features that commute with the rigid transformations (rotation, translation). If we denote the transformation as $T : \Re^3 \mapsto \Re^3$, the invariant feature operator G would behave as follows (in 3D) :

$$g(x, y, z) = G \circ f(x, y, z) \;\leftrightarrow\; g(T(x, y, z)) = G \circ f(T(x, y, z)). \tag{6.31}$$

One can easily confirm, for instance, that the length of the gradient vector, $||Df||$, does not change with a change in coordinates. Likewise, singular points are also invariant to rotations/translations of the domain.

Several other aspects of differential operators and invariants are important for visualization. First, many features are developed to characterize *local contrast or variation in function values*. This is true, for instance, of the gradient magnitude, $||Df||$, which is typically considered a place of high contrast in f, also called *edge*. Second, zero crossings of differential operators are often used to find points or features in the domain that are extremal in some property of f.

For instance, the famous *Canny* edge [3] is defined, mostly simply, as the zero crossings of the directional derivatives of $||Df||^2$, in the particular direction of Df. This gives the following condition for these extremal points (in 3D):

$$\mathscr{C} = \{(x, y, z) | g(x, y, z) = 0\} \text{ where } g = (Df)^T D^2 f (Df), \tag{6.32}$$

and thus we see that *edges* are zero level sets of a differential invariant. For robust edges, one typically imposes addition criteria, such as a minimal value (threshold) of $||Df||$ and that the directional derivative of g be negative (for a maximum), although this is rarely necessary when using a threshold.

There are many such invariants and features that can be derived by either thresholding them or finding zero crossings of their derivatives. Other examples include:

- Edges of various types by considering zero sets of second derivatives, as in the Canny edge above, and alternatives such as $D^T Df = 0$, as proposed by Marr and Hildredth [36].
- Extremal points of level sets (local max/min of curvature) by considering level sets of second derivatives of the gradient (third derivatives of f) along the direction(s) perpendicular to the level set. Special care must be taken in 3D, where there is a tangent plane to the level set.
- Ridges on f by considering extremal points of the eigenvalues of $D^2 f$ in various directions. There are various choices here, described extensively by Eberly [7].

Kindlmann [24] gives a compelling overview of this strategy along with various practical considerations. In particular, derivatives are prone to high-frequency artifacts (amplify the magnitudes of small features) and can increase the effects of noise and errors associated with approximations from a discrete grid. The solution to this is usually some combination of smoothing and approximating functions with higher order smoothness guarantees.

In considering these differential invariants, transformations and features on vector fields are also important. For this discussion, we consider vector fields of the form $\mathbf{v} : \Re^m \mapsto \Re^m$, and where the domain and the range are the same space (e.g., the vector is expressed in the same coordinates as the domain). Often, in 2D this would entail $\mathbf{v}(x, y) = v_x(x, y), v_y(x, y)$, and where the subscripts represent components of $\mathbf{v}$ associated with those coordinate directions. This representation is important because it means that the domain and range of $\mathbf{v}$ transform with the same operations (e.g., rotations affect both the domain and range).

The magnitude of such a vector field, $||\mathbf{v}||$, is an invariant. So too are singularities, where $\mathbf{v} = 0$ (once again, the crossing of level set curves/surfaces). Often, such vector fields are the output of a physical simulation, and they represent a physical quantity such as a fluid flow or a mechanical deformation. In these cases, the second derivatives are also important, and they are characterized by the Jacobian matrix:

$$J = D\mathbf{v}^T = \begin{bmatrix} \frac{\partial \mathbf{v}_x}{\partial x} & \frac{\partial \mathbf{v}_y}{\partial x} \\ \frac{\partial \mathbf{v}_x}{\partial y} & \frac{\partial \mathbf{v}_y}{\partial y}. \end{bmatrix} \tag{6.33}$$

The Jacobian of a vector field bears a resemblance to the Hessian of a scalar function—indeed, the Hessian is a special case. It is the Jacobian of the *gradient field* of a function. While the Hessian is symmetric, Jacobians in general need not be.

A typical strategy in computing features from the Jacobian is to compute invariants of this matrix. For instance, the eigenvalues of the Jacobian, which might be complex-valued, are invariant (to coordinate transformations) and of interest because the real parts describe how the field is pushed in or away from a point, where the imaginary part describes the rotation of the field (around a point). There has been

a considerable amount of visualization work that has sought to identify singularities in flow fields (where the flow is zero) and to characterize the rotations and compressions/expansions around those points.

Generally, there is a mathematical system for computing invariants of the Jacobian. Two invariants of particular interest are the trace of J, which is $\mathrm{Tr}(J) = J_{11} + J_{22}$ and is also the sum of the eigenvalues. The norm, which is $\mathrm{Tr}(J\,J^T) = \sum_{k,l} J_{kl}^2$, is the sum of the squared magnitudes of the eigenvalues. The determinant of J, also the product of eigenvalues, is also relevant to understanding the structure of the field.

In the context of displacement fields, we are often interested in the total amount of deformation pointwise, which is captured in the symmetrized Jacobian

$$\varepsilon = \frac{1}{2}\left(J + J^T\right), \tag{6.34}$$

and the norms of ε produce scalars that summarize this deformation.

In the context flows, the vorticity of the vector field gives the rate rotation at each point—i.e, how would in infinitesimal circle/sphere rotation if its surface followed the flow. In two-dimensions, the vorticity is

$$\omega_z = \frac{\partial v_x}{\partial y} - \frac{\partial v_y}{\partial x}, \tag{6.35}$$

and it has the convention of being a vector perpendicular to the plane (either inward- or outward-facing, depending on the sign). In 3D, the vorticity is written as the *curl* of the velocity

$$\boldsymbol{\omega} = \nabla \times \mathbf{v}, \tag{6.36}$$

and the vector is along the axis of rotation (with direction defined according to the right-hand rule). In both the 2D and 3D cases, vorticity computation results in another scalar or vector field, respectively. Scalar invariants such as magnitude or the acceleration magnitude combined with extremal analysis or level sets produce subsets that allow for the visualization of vortex structures in flows [22]. It is worth noting that in many fluid applications the velocity fields are dynamic, and are functions of space and time, e.g., $\mathbf{v}(x, y, t)$. In such cases, the analysis of vorticity and other flow properties over time becomes important but is beyond the scope of this discussion [15].

6.3.1 Integral Curves of Functional Data

In addition to transformations that rely on derivatives of functions, many transformations done for visualization rely on *integrals* of vector fields. Here we consider $\mathbf{v} : \Re^3 \mapsto \Re^3$, as above.

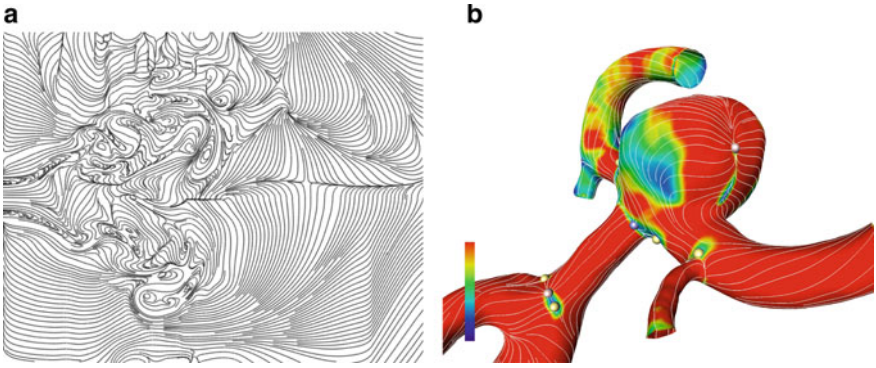


Fig. 6.10 Topology guided uniform streamline placement. **a** 2D jet flow. **b** Surface blood flow of an aneurysm. The background color represents wall shear stress. Visualization: Olufemi Rosanwo, Amira

In analyzing or visualizing such flow fields, the integral lines of $\mathbf{v}$. Thus, one can define a streamline $\mathbf{u}$, parameterized with s as:

$$\frac{\partial \mathbf{u}}{\partial s} \times \mathbf{v}\left(\mathbf{u}(s)\right) = 0, \tag{6.37}$$

which says that the tangents to the curve $u(s)$ are parallel to the vector field. In practice, these curves are usually computed via integration:

$$\mathbf{u}(s) = \int_0^s \mathbf{v}\left(\mathbf{u}(\alpha)\right) d\alpha, \tag{6.38}$$

where $\mathbf{u}(0) = \mathbf{u}_0$ is the starting point of the streamline. A typical streamline visualization of a vector field consists of a rendering (in 2D or 3D) of a collection of polylines or tubes that give the overall structure of the field, see Fig. 6.5b. The placement of the initial points for these lines requires some care, so that streamlines are too sparse or too cluttered, and this is an area of significant attention in the visualization community [46], see Fig. 6.10.

When considering dynamic vector fields (vector fields that are functions of time), several more options for integrating curves arise. One option is to let s in the integral above be t, the dynamic parameter of $\mathbf{v}(x, y, t)$, and let the vector field change with t. This is called a *pathline*, and it is the equivalent of letting a particle loose in the flow and rendering the path it travels. A second option is to simulate the path of a continuous stream of particles placed into the flow at a point. This dynamic curve is called a *streakline*.

A special kind of vector field is a *gradient field*, which arises when $\mathbf{v}$ is the gradient of f, i.e., $\mathbf{g} = Df$, where we use $\mathbf{g}$ to denote this special case. Under these circumstances, the field is *curl free* by construction (and the vorticity of such a field would always be zero). Because of this property, the integral curves of such a gra-

dient field always connect singularities, where $Df = 0$ These singularities consist of different types: minima, maxima, and saddle points. The status of a singularity is determined by evaluating the eigenvalues of the Hessian. If we consider the eigenvalues in descending order, $k_1, \ldots, k_m$ for an m-dimensional domain, we have the following:

$$\begin{array}{rl} & k_1 \ldots, k_m > 0 \text{ minimum} \\ k_1 \ldots, k_j > 0, \; k_{j+1}, \ldots, k_m < 0 & \text{saddle} \\ & k_1 \ldots, k_m < 0 \text{ maximum} \end{array} \tag{6.39}$$

Notice that we do not normally consider cases where the eigenvalues are zero, because these are not considered *generic* or regular points, and they show up with very low probability (in theory). In practice, special care must be taken to avoid the numerical problems associated with data sets that do not meet these criteria.

Virtually every point in the domain of a (generic, regular) function has a gradient. The integral curve of the gradient field from that point terminates at a maximum. A relatively few points will terminate at a saddle. From saddle points, one can trace curves (in the directions of the eigenvectors of the Hessian) toward sets of maxima (e.g., pairs in 2D). This same analysis extends to toward minima/saddle points if one integrates the negative of the gradient field, $-\mathbf{g}$. Note that a very similar concept can be applied to general vector fields. Here, limit sets play the role of critical points. As for the gradient field, there are locally defined limit sets. These are sources, sinks, and saddle points. In addition, there are, however, also nonlocal limit sets. In 2D, these are periodic orbits; in 3D, more complex configurations are possible [17].

Using these ideas, we can partition the domain of the image into regions that each share the same minimum. The set of points in the domain whose integral curves of $-\mathbf{g}$ lead to the same minimum is often called a *watershed*, because if the function were treated as a topographical surface and water were to flow toward minima (due to gravity), the water falling (e.g., in a rainstorm) on that region of the domain would all flow toward the same location. This kind of *watershed segmentation* has shown up extensively in the image processing literature (and software) for partitioning images around edge-like features, such as the gradient magnitude, as in Fig. 6.11. In performing this kind of analysis, one must recognize that the number of minima and/or maxima in a field of data (function) can be arbitrarily large, especially in regions of the image where the gradients are small (nearly flat regions). To address this, we typically filter this partitioning of the domain, and combine regions based on the *depth* of the watershed, as in Fig. 6.11. Each watershed has along its boundaries a sequence of maxima and saddle points. The difference between the function value at the minimum and the saddle point of least value is the watershed depth. More recent work has referred to this depth as the *persistence* of a watershed region, and which shows stability under certain conditions [8]. Watersheds that are not deep (shallow) are often combined with adjacent watersheds to form larger, deeper regions. This can be done interactively by users, with an appropriate interface [5].

Several other aspects of this kind of *topological analysis* are important for visualization. First, if one considers the ascending and descending integral curves (integrating negative and positive gradient fields), they (almost always) terminate at maxima

and minima points, respectively. The sets of points that share maxima and minima (terminations of descending and ascending gradient flows) also form a partitioning of the domain, which is sometimes called the *Morse–Smale complex* and the individual elements (region and min/max pair) are *crystals*. The boundaries of these regions consist of ascending/descending integral curves (surfaces, or families of curves, in 3D) that pass through saddle points.

This strategy, of reducing a function to it singularities (minima, maxima, and saddles) and connecting those singularities by either the Morse–Smale crystals or the curves that connect saddles along the boundaries, has been proposed as a way of visualizing the structure of complex or high-dimensional functions [31]. Much like level sets and streamlines, this kind of analysis produces a discrete set of geometric

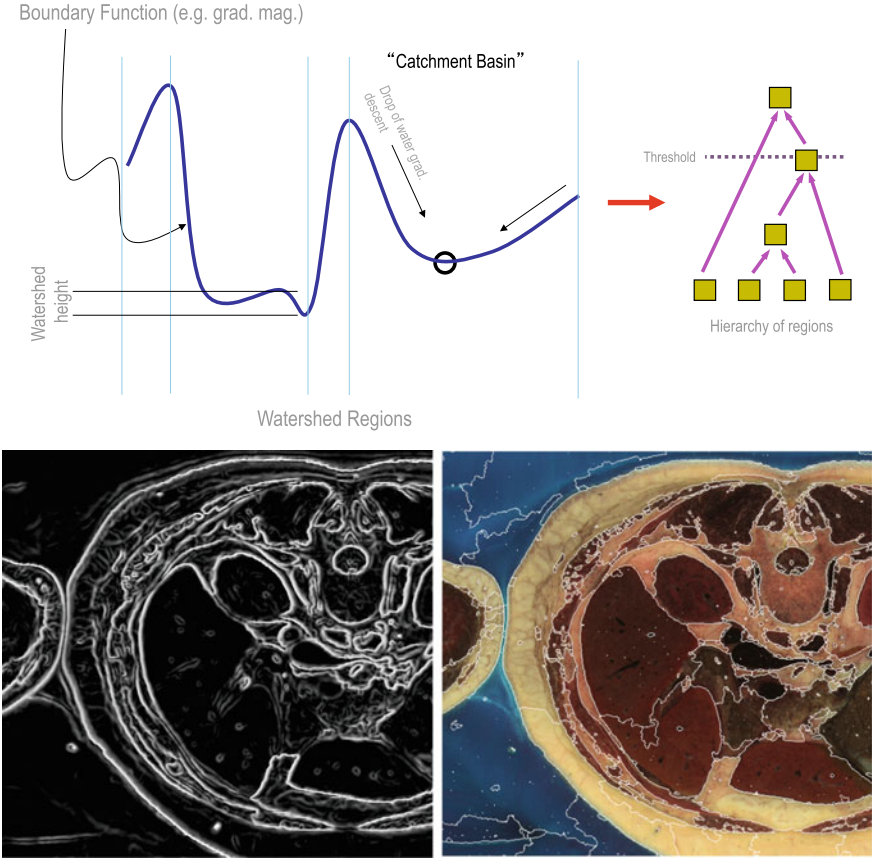


Fig. 6.11 Top: a watershed decomposition of a function tracks regions for which the integrals of the gradient fields terminate in a common minimum (or maximum). Bottom-left: the gradient magnitude of an anatomical image indicates boundaries of regions. Bottom-right: a partitioning of the function (overlaid with white lines on the original color image) shows watershed regions

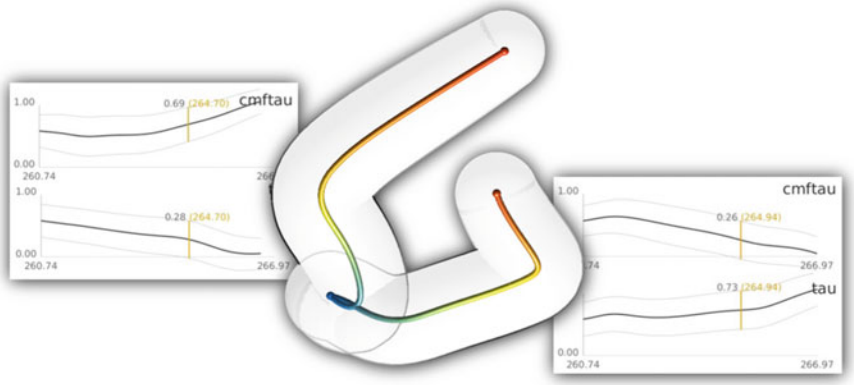


Fig. 6.12 Figures from Gerber et al. [12] show a rendering of a Morse–Smale (M–S) decomposition of the high-dimensional parameter space associated with climate simulations, including selected parameter values from the two M–S crystals

objects that are more easily rendered than the original function. Virtually any visualization method that relies on this kind of topological analysis must include some manner, as described above, of removing/combining shallow, small, or otherwise insignificant regions.

An example of this kind of topological analysis is the work of Gerber et al. [12], where they visualize high-dimensional scalar functions by rendering the function as a graph, with extremal points as vertices, connected by edges, rendered as curves/tubes, that represent the structure of the Morse–Smale crystals that connect those extrema. See Fig. 6.12. The method relies on embedding discrete sets of singularities into lower-dimensional spaces (2D or 3D) as described later in this chapter.

Also important to these topological analysis methods are the methods that combine the analysis of singularities with levels sets (or *contours*). If one considers the level sets of a function at some value k, then the family of level-set curves or surfaces forms patterns that adhere to certain rules. For k increasing and considering a contour to be a curve with an interior defined with $f(\cdot) < k$, we can track the behavior of contours:

A. Topologically separate contours form/begin at minima (of value k), as points and then isolated, closed contours (curves or surfaces).
B. Contours join/merge at saddle points, and the new structures can achieve alternative/complex topologies (e.g., holes) as they merge.
C. Isolated holes in contours contract to points (and annihilate) at maxima.

This kind of analysis [9] forms a graph (sometimes referred to as the *Reeb graph*) that facilitates the visualization of a function in terms of its associated Reeb graph [34], as in Fig. 6.13.

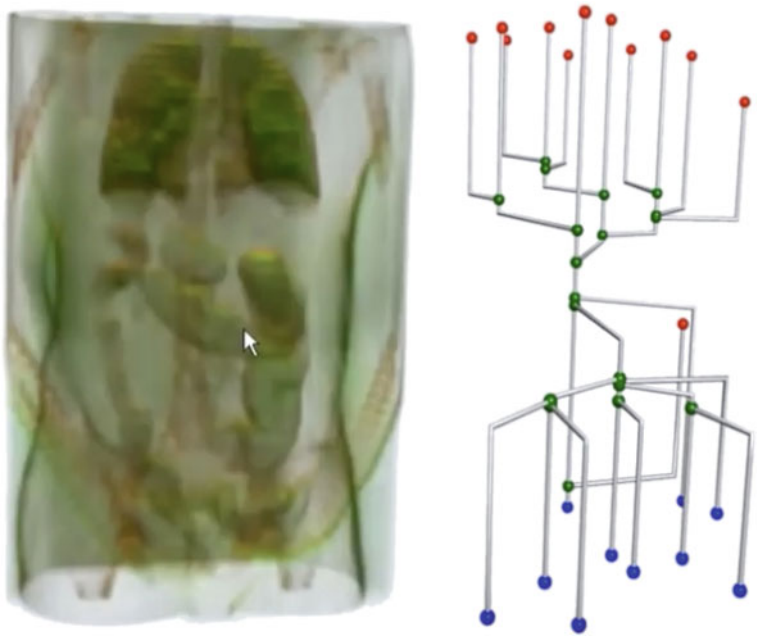


Fig. 6.13 The structure of a 3D function is characterized by a volume rendering and a rendering of the associated Reeb graph. Image courtesy of Vijay Natarajan and Harish Doraiswamy

6.4 Visualizing Instances by Dimensionality Reduction

A typical visualization problem is as follows. A data set consists of a number of instances of structured data. In the following discussion, we also refer to an instance as a *data point*. One would like to visualize these points to understand the following:

- Do the points group together or form *clusters*? If so, how distinct are these clusters, how many are there, etc.?
- Are there trends or relationships among points and variables that could give qualitative or quantitative insights into the collection of data?
- Does the data conform to expectations of samples from known probability distributions, such as normal distributions?
- Does the set contain instances that are unusual or very different from the other instances? How different and how many are there?

If the data points are samples in a 2D (or even 3D) space, one can typically rely on direct visualization via a *scatterplot*, where individual points are represented via the positions of symbols or glyphs (e.g., dots, squares) on a 2D graph (or a 3D cloud within a 3D or interactive display). Figure 6.14 shows examples of 2D scatterplots that demonstrate some of the properties above.

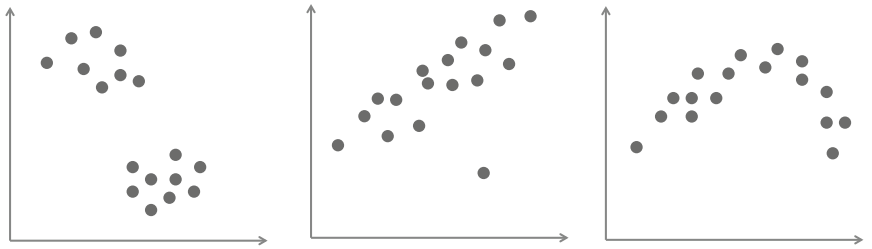


Fig. 6.14 Scatterplots show clustering, correlation, and nonlinear structure

Of course, as we consider the analysis of data points, we must be aware of the opportunity and/or need for quantitative analyses. For instance, often when looking for relationships among variables, one considers the correlations among variables or the best-fitting linear model (e.g., fitting a line in the 2D case). Anscombe [1] describes a quartet of examples where best-fitting lines for 2D data points can be misleading, as a motivation for direct data visualization. This danger, of being potentially misled (or at least underinformed) by a simple model, is a very general threat to people using and analyzing data; it goes beyond linear models. For instance, people will often consider the mean and (co)variance of a distribution, which often misses important aspects of a data set (such as outliers, skew), and the *whisker plot* (or *box plot*) is a common visualization tool for 1D points, using rank statistics, that helps evaluate properties beyond mean and variance. In general, virtually any parameterization or low-dimensional model of a set of instances risks missing some important aspects of the data. Yet, for high-dimensional data, direct visualization is often impossible. Thus, a complementary approach that combines visualization and analysis is often required.

One of the most common methods for visualizing point sets (instances) of more than 2 dimensions (and assuming a metric space) is to *project* the data onto a 2D subspace. The most widely used method for this is *principal component analysis* (PCA), which is equivalent to finding the k-dimensional, linear subspace that minimizes the projection distance onto subspace. The procedure, mathematically, is as follows: A point set X is represented as a matrix

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & & \vdots & \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}, \tag{6.40}$$

where x_{ij} is the ith coordinate of the jth data point. The data is first *centered*, so that the mean is zero. Thus, we have $\hat{X} = X - \bar{x}$, where $\bar{x} \in \Re^m$ is the mean across the data. That is

$$\hat{x}_{ij} = x_{ij} - \frac{1}{n}\sum_{j} x_{ij}. \tag{6.41}$$

From the centered data, next compute the inner product, or correlation matrix

$$C = \hat{X}\hat{X}^T. \tag{6.42}$$

The k-dimensional *basis* for the projection consists of the first k eigenvectors (ordered by decreasing eigenvalue) of C, which we denote, $E = e_1, \ldots, e_k$. The lower-dimensional coordinates for centered data points are the *loadings* of the data onto this new basis:

$$Y = E^T \hat{X}. \tag{6.43}$$

These coordinates can then be used for visualization, e.g., when $k = 2$.

The new coordinates, Y are in terms of the basis vectors, V, which form a k-dimensional, hyperplane in $\Re^m$. The hyperplane coordinates for the data are computed as

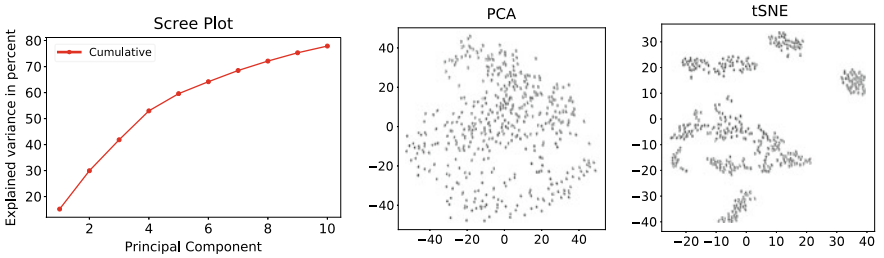
$$X_p = EE^T \hat{X} + \bar{x}, \tag{6.44}$$

where X_p is the projection of the data onto the best-fitting, k-dimensional hyperplane.

This kind of transformation, of finding a lower-dimensional space (and a smaller set of coordinates) to represent a set of points etc. is sometimes called an *embedding* of the data, because it assumes that the original nD data is positioned on a kD manifold (in this case, a hyperplane) that is embedded in the higher-dimensional space. As we consider this process, it is important to keep several things in mind. First is the accuracy of the representation. For PCA, the *projection error* of the points onto the kD hyperplane is given by the root of the sum of squares of the eigenvalues associated with the $n - k$ smaller eigenvectors. To visualize the effects of projection, we often use a *scree plot*, which shows the percentage of the total variance captured in the first k eigenvalues, as shown in Fig. 6.15. Also note that PCA is the optimal choice of kD hyperplanes to model data since it minimizes this projection error, or residual. Also worth noting is that in some cases the number of samples is smaller than the dimensionality of the ambient space (i.e., $m < n$). In this case, the better computational strategy is to work on the dual of the original problem, which operates on the linear subspace defined by the data points. For this, we conduct the eigenanalysis on the matrix $C' == \hat{X}\hat{X}^T$, which has the same eigenstructure, E', as C, defined in (Eq. 6.42). The basis is obtained by multiplication with the data itself:

$$E = X\Lambda^{-\frac{1}{2}}E', \tag{6.45}$$

where *Lamda* is the diagonal matrix of eigenvalues. In cases where m and n are both very large, the construction of the associated (large) covariance matrix can be prohibitive. In these cases, the largest eigenvectors/values can be found through



(a) (b) (c)

Fig. 6.15 **a** A scree plot of MNIST handwritten digit data depicting percentage of variance explained by PCA modes. **b** A 2D embedding/layout using PCA loadings (units are arbitrary). **c** A 2D embedding using t-SNE

iterative methods, such as *power methods*, that do not require explicit construction or decomposition of the matrix (e.g., the power method).

Notice that the dual formulation of the PCA problem relies on the analysis of the $n \times n$, inner product matrix (also called a *Gramm matrix*). This opens up the possibility of embedding data points that may not be given in a conventional, metric space, but for which there exists only an inner product (or similarity) operator. This situation arises, for instance, when using the *kernel method* for analysis of data.

An alternative method for formulating the embedding of data points into low-dimensional spaces is via the matrix of distances between all pairs of data points, which we denote as D, with elements d_{ij}. The goal is to find parameters $Y \in \Re^k$ so that the distances between points in $\Re^k$ match, as closely as possible, those given in D. This problem is sometimes called *multi-dimensional scaling* (MDS), and MDS is often used to refer to the family of methods that try to find such coordinates, Y. The *classic* approach to MDS (indeed, called *cMDS*) is an algorithm that centers the matrix D and then computes the eigenvectors, as in PCA. The algorithm is:

A. Construct the squared distance matrix, $D^{(2)}$, where $d_{ij}^{(2)} = d_{ij}^2$.
B. Center and negate the squared distance matrix:

$$B = -\frac{1}{2} J D^{(2)} J \quad \text{where} \quad J = I - \frac{1}{n} 1 1^T \tag{6.46}$$

and 1 is a vector with values of 1 and length n.
C. The coordinates are $Y = E_k \Lambda_k^{frac12}$.

Note that cMDS minimizes the normalized strain of the embedding in the case where the original distances are from a Euclidean space (e.g. $X \in \Re^m$). However, this same algorithm is used in many other settings for visualizing data, e.g., $k = 2, 3$, with useful results.

There are many other approaches to MDS, but one that it also very widely used is to directly minimize the stress associated with the embedding. The stress is typically

$$\text{Stress}_D(Y) = \left(\sum_{ij} (d_{ij} - ||y_i - y_j||)^2 \right)^{\frac{1}{2}}, \tag{6.47}$$

where y_i, the ith column of Y, are the new, embedded coordinates for the ith data point. This function is bounded by an approximation that is quadratic in the unknowns, and minimization is solved efficiently using the iterative *SMACOF* algorithm.

There is a deep relationship between (squared) distance matrices and the matrix of inner products, also called the *Gramian matrix*, which is used in the dual formulation of PCA. Under certain conditions (e.g., distances/products in Euclidean space) one can be derived from the other. For embedding points for the purposes of visualization, these two types of matrices are used in a similar manner; their eigenvectors are used to construct coordinates in a new, lower-dimensional space. Thus, for many visualization applications, practitioners will use either distance or inner product matrices, depending on what is available and appropriate for the original data.

The ability to compute low-dimensional coordinates using only distances (or similarities, inner products) gives rise to some important technologies for visualizing collections of points, even if these data points do not have well-defined coordinates in some metric space. Here we give several examples of how this is useful.

When modeling high-dimensional data, it is sometimes useful to treat the data as existing on a lower-dimensional, curved (or nonlinear) surface, or manifold. The so-called *manifold learning* problem has received a great deal of attention in machine learning and statistics, but has become less important in recent years because of advances in machine learning technologies that can learn directly on complex, high-dimensional data sets. However, for visualization, manifold learning is still a useful dimensionality reduction method. A relatively easy to use for discovering manifold coordinates is the method of *isomap* [51]. The isomap algorithm is designed to construct an approximate distance matrix that captures distances *within the manifold*, which is embedded in a high-dimensional, mD, space. The isomap algorithm works as follows:

A. Compute the distance matrix D for the original data.
B. Determine the K nearest neighbors (kNN) for each point (K is a free parameter), and construct the KNN graph with edge lengths being distance.
C. Compute the distance between each pair of points on the KNN graph (e.g., using breadth-first search), and construct a new distance matrix D'.
D. Determine new coordinates from MDS (any method) on D'.

This algorithm has been shown, in some cases, to learn the manifold structure from very curved or convoluted manifolds in high-dimensions. The main challenge with the isomap algorithm is the selection of the number of nearest neighbors K, because this determines which jumps on the NN graph will be considered within the manifold. If K is too small, the graph becomes disconnected (and the eigenstructure of D' shows this), and if K is too large (in the limit), the method produces results that resemble MDS on the original distances.

An alternative to preserving distances between points is to pose the embedding problem as preserving *data density*, which is the strategy of the *t-sne* embedding method [54]. Local probability densities are computed with nonparametric density estimation, and the target coordinates Y are constructed so that every point has a similar nearby density of points. The method is widely used and generally effective, but has the effect of preserving or enhancing clusters in the data (which have higher data density).

Another case in which MDS-like methods are useful is when the data points have intrinsically no coordinates, but where distances or similarities are readily available. This comes often in the context of graph layout. The edges of a graph often have weights that are associated with either dissimilarity (approximate distance) or similarity (inner products). This happens, for instance, when vertices have associated signals, such as voting patterns for politicians, weather patterns for cities or stations, or, in biology, interactions between genes, molecules, or organisms. In these cases, it is sometimes helpful to embed the graph vertices in 2D as part of the visualization. As above, the affinities/similarities or distances are part of the computation of 2D coordinates for the vertices. Stress minimization, as above, is often the method of choice, in part because it can be combined with other criteria. The problem of graph layout is widely studied, and effective solutions often address concerns in addition to distance, such as edge crossings, and edge/vertex density.

6.5 Data Summaries

In many cases, we are presented with a collection of *instances*, where each individual instance may be a data point or some more complex object, such as function, unstructured document/record, or a graph. Organizing these data as points or icons in a 2D display, as in the previous section, can often be helpful, it is sometimes effective to summarize the overall structure and relationships of these instances. This section describes some methods of summarizing collections of instances. Here we assume that the data is homogeneous and structured, and we leave the discussion of more complex data types for the next section. Of course, the choices we make in summarizing data depend on the kinds of questions we are trying to answer and the applications we have in mind. Here we give some examples of the most widely used strategies.

A typical problem in visualization of instances or points sets is to understand the relationships among samples and if the data naturally form groups or clusters (and the nature of those clusters). The notion of *clustering* is a longstanding problem in pattern recognition and data analysis, and there are a wide variety of approaches. Typically, data is said to consist of clusters if there are two or more subsets of the data for which the point-to-point distances are smaller than the distances to nearby groups.

A typical formulation of the clustering problem is as follows. Inter- and intragroup distances are quantified (e.g., as sums over distances between point pairs), giving

rise to an objective (or energy) function that can be optimized. Because the energy involves assignments to groups, it is combinatoric and nonconvex, and thus it is often solved iteratively. A widely used and generally effective clustering algorithm for points in a metric space (distances can be computed) is *K-means*, which is the following algorithm, for input data $X = \{x_1, \ldots, x_K\} \in \mathscr{D}$, where $\mathscr{D}$ is the domain of the data points:

A. The user decides on the number of clusters K, and the cluster centers $C_1, \ldots, C_K \in \mathscr{D}$ are initialized (usually at random).
B. Each data point is assigned to the nearest cluster center.
C. The cluster centers are updated and assigned to the average of the data points to which they are assigned.
D. If the update is sufficiently small, terminate, otherwise, go to step B.

The output is the positions of the cluster centers and their assigned data points.

This kind of clustering or grouping of data points can impact visualization in several ways. In some cases, it is useful to visualize the clusters themselves. Thus, the clusters would be embedded (e.g., using MDS, as above) in 2D, and each cluster would be represented with a mark or glyph, which might also encode information about the cluster, such as its number of elements, extent/variability, or center element. Another use for clusters is to modify the glyphs in a visualization of a 2D embedding/projection, which can help identify differences in data along dimensions that are not well captured in the embedding. Finally, some embedding algorithms are designed to preserve information about clusters in the data, where the separation of clusters (detected as a preprocessing step) is a criterion that is built into the objective function that is optimized for the embedding. The general strategy of clustering data is useful in other contexts, where collapsing or summarizing groups of instances aids in interpretation. For instance, clustering of edges in large graphs has been used for *edge bundling*, to reduce complexity in graph visualization [6].

The problem of clustering data points has also been examined from a very different point of view—using hierarchies of graphs. We consider the ε-graph of a data set as consisting of a vertex for each data point and edges connecting every vertex pair x_i, x_j if and only if $d_{ij} \leq \varepsilon$. Then we can consider the connected components of that graph to be individual clusters. This is sometimes called *single-linkage clustering* and it is known to be unstable when one makes small perturbations to the data points. However, if one considers the hierarchy of clusters as a function of ε, as in the *dendritic tree*, this can provide information about the texture/structure of the point set. Also, clusters that are stable or persistent through a wider range of ε values might be considered more important (e.g., more robust to perturbations in data), sharing the same mathematical underpinnings as the watershed depth, described in the previous section.

This kind of basic, distance-based, cluster analysis is the simplest example of a very rich set of methods in computational topology called *persistent homology* [4, 8]. Here we give only a high-level view of the methodology. First, one extends the notion of the ε-graph, as above, to include a filtration (nested sequence) of *simplices*, which are not only vertices and edges, but also triangles, tetrahedra, etc. There are several

approaches for constructing those filtrations of simplices. One can then compute, in a very precisely defined manner, topological summaries, including not only the number of connected components, but also tunnels (holes in one-dimension, loops in higher-dimensions), and cavities (2D holes, hollow regions enclosed by a surface). We can also track changes in these summaries (or the corresponding feature, such as a hole) as ε increases. These summaries are sometimes visualized as a collection of stacked horizontal line segments (or bars), where the ends of each bar correspond to the appearance (or birth) and disappearance (or death) of the associated feature, as a function of ε.

6.5.1 Statistical Summaries

In visualizing sets of instances of data, it is sometimes difficult to make sense of the raw data, especially if the individual data points have an inherently complex structure or if there are especially many of them. In many cases, one would like to get a high-level or *big picture* view of the data. This kind of visual analysis is often for quality control or to inform some other type of quantitative analysis. For this reason, it is often useful to construct statistical summaries of data and to visualize those summaries either instead of or in addition to the raw data.

A typical summarization strategy is to compute the mean an variance of a data set. For instance, if we consider functions $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})$, with 1-2-3D domains, there is a sample mean $\bar{f}(\mathbf{x}) = (1/n) \sum_i f_i(\mathbf{x})$ and an associated covariance. Here we avoid a discussion of the technical issues associated with variance in functions spaces, and instead assume that f_i has a finite-dimensional representation (e.g., values evaluated on a regular grid, as with an image or volume), and each instance is represented as a vector of length n. The covariance structure can be very high-dimensional ($n \times n$ matrix), and difficult to visualize, so simplifications or approximations are common. The most common simplification is to compute the variance of f pointwise over the ensemble for every $\mathbf{x}$ in the domain. This is equivalent to considering only the diagonal of the covariance matrix, and it ignores the correlations between points. Figure 6.16 shows this mean and pointwise covariance for a data set from a fluid simulation. Alternatively, one can visualize the eigenvectors of the full covariance matrix (typically, one would use the dual method described in section about dimension reduction) and visualize the eigenvectors of the covariance.

With functions, one is often interested in features, and how those features behave within a set or ensembles of functions. As in Sect. 5.4, many interesting features can be represented as zero-crossing of the function itself or fields of data derived from that function and its derivatives. Several researchers have proposed to extend the computation of level sets to the probabilistic setting. The problem can be stated as computing the probability of a level set passing through or between a set of grid points (or pixels), given a stochastic model of nearby function values. This is the strategy behind probabilistic marching cubes and several variants [42].

The strategy of computing the mean and (co)variance to summarize data has limitations, in that it reduces the ensemble of (possibly) complex data sets to a relatively

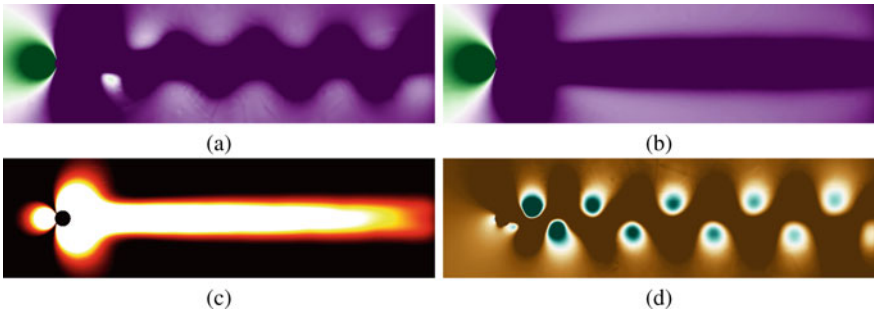


Fig. 6.16 Analysis of a set of pressure data from a fluid simulation, with flow left to right across a circular obstacle. **a** One example of a pressure field from this simulation (purple-low, green-high). **b** The mean pressure field from an ensemble of 300 samples. **c** The pointwise variance (heat map). **d** The first eigenvector of the covariance matrix

small number of values, and these summaries are sensitive to data outliers, and, as in the case of point-wise variances, ignore global relationships (e.g., correlations) in the high-dimensional data. Indeed, these are often the very things that we are attempting to detect or understand as we visualize such ensembles. Thus, nonparametric is *descriptive* approaches to summarizing data are also important.

The descriptive approach to summarizing data is well motivated by one of the most widely used of all visualization tools—the box or whisker plot, proposed by Tukey [53] and shown in Fig. 6.17. The whisker plot typically shows a summary of rank statistics of a set of 1D data points, with bars or icons to indicate the median (and often the mean as well), various percentile ranks (e.g., 25 and 75%), and outliers. These rank statistics are computed from an *ordering* of the data along a single axis. The extension of this kind of visualization to more complex data requires two developments. The first development is the generalization of rank statistics to multidimensional and nonmetric data. For this, several researchers have proposed the use of *data depth*, which is a tool from descriptive statistics that constructs a *center outward* ordering of a collection of data points. In such a scheme, the median of a data sent would typically be the *deepest* among the given ensemble. There are several methods for computing the depth of a data point within an ensemble, but a useful strategy is the method of *band depth*, where the depth of a data point is computed as the probability that it lies between a small, random selection of the data. The notion of *between* must be defined for each data type, depending on the application, and the probability of lying within a band formed by a random set of samples is computed with a Monte Carlo approach, which is a sample average is computing by choosing small subsets from the given data.

The data depth approach is developed for functions, where the band is formed by the min/max values at each point in the domain for a small set of j functions (chosen at random) [41]. Given a set of functions, one can compute the median and the min/max extent of the functions within a certain rank of the data (e.g., 50%). Sun

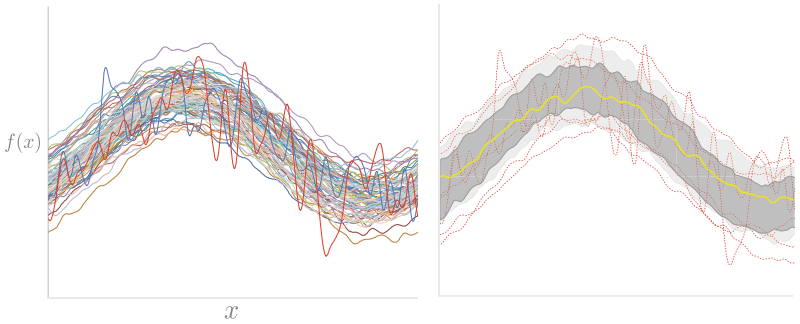


Fig. 6.17 Left: an ensemble of functions have some common structure and show significant variability. Right: a functional box plot, as proposed by [50]

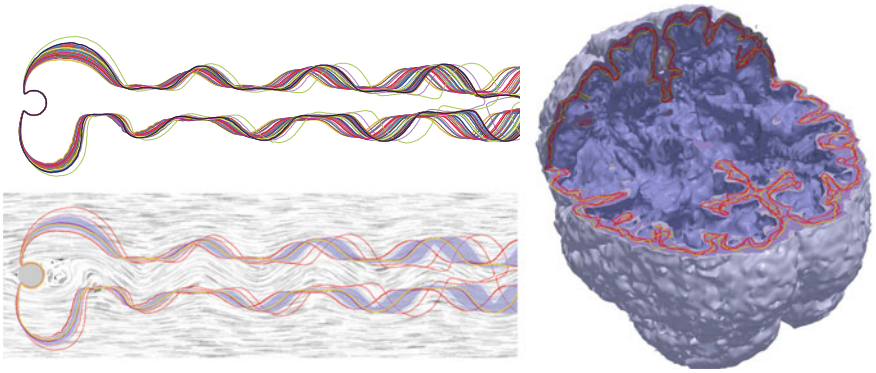


Fig. 6.18 Left: contour box plots of isocontours of pressure in an ensemble of fluid simulations. Right: 3D contour box plots from an ensemble of registered brain images

and Genton [50] use function band depth to construct function box plots, which are the natural extension of whisker plots to functions, as in Fig. 6.17.

For points in $\Re^n$, band depth is the probability that a given point lies in the simplex or convex hull of $K > n$ randomly chosen points. For large n, the availability of sufficiently many K-sized subset is often prohibitive, and alternatives, such as half-space depth [52] and spatial depth [48], become desirable. For points in $\Re^2$, a depiction of the 50% band and an inflated version of that, with outliers marked, is called *bagplot*.

Several researchers have proposed extensions of data depth and associated visualizations, extensions to box plots, to more complex data types. The method applies to 2D, scalar functions [50], as well as curves in 2D and 3D. Whitaker et al. [57] have extended data depth to sets and show box plots for level sets (contours) in 2D and 3D [44]. Figure 6.18 shows some examples of 2D and 3D box plots for different objects. Raj and Whitaker have extended data depth to vertices and paths on graphs [45].

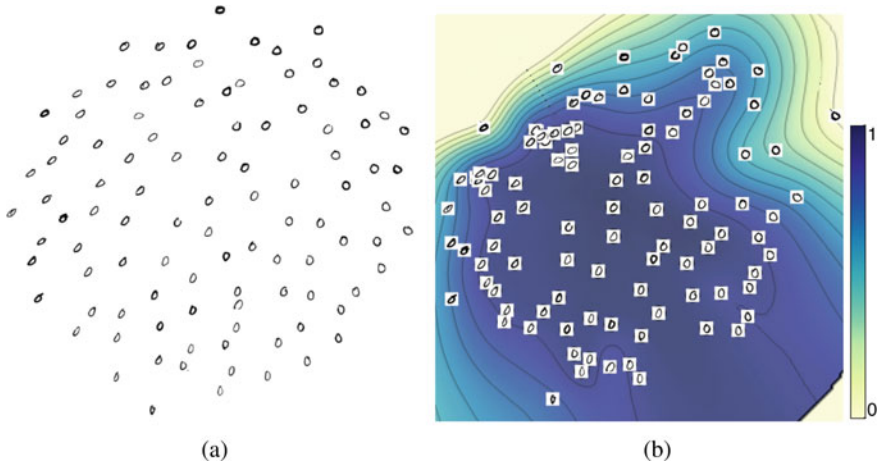


Fig. 6.19 2D layouts of the MNIST "0" digit data. **a** Layout using an MSD embedding. **b** Layout with depth-aware embedding that organizes by depth with contours/colors that show relative depth of samples

In dimensionality reduction, the projection of data into lower-dimensions often obscures the relative depth of a data point, and thus outliers can be misrepresented as being central to the dataset. Raj and Whitaker [43] have proposed dimensionality reduction techniques that preserve data depth, in addition to distance (or density), as in Fig. 6.19.

6.6 Transformations on Unstructured and Discrete Data

Virtually all of the methods in the previous sections of this chapter rely on quantitative relationships between samples or points in the range or domain of a function. However, many data sets come in forms that are not well suited to the quantification of distances, similarities, or coordinates. A very common example is a *corpus* of text documents, as one may have from a collection of news articles or emails. Each instance, in this example, is a document consisting of words, spaces, and punctuation (we will ignore emojis for this discussion :-)). One might like to visualize the corpus of documents and understand how they relate to each other. Typically one might like to know if there trends over time, if they form clusters, if there are outliers, etc. These kinds of questions might benefit from a scatterplot visualization, a clustering, or topological analysis. However, these methods will require distances or coordinates for each sample (each document in this case). This chapter discusses some of the methods by which unstructured data, such as a text document, are encoded for subsequent analysis or visualization.

6.6.1 *Organizing Data in* Bags

A text document consists of a collection of words in a particular order. The actual words used and the order in which they appear give the document its meaning. However, practitioners of natural language processing have noticed that some information about a document can be discerned from the types of words that are used, and their frequency, while ignoring the ordering of the words, the sentences, grammar, etc., in the document. This leads to a way giving coordinates to a document—we simply count the number of times each word occurs in a document and the resulting histogram becomes a quantitative descriptor of the document, which induces a distance and/or inner product computation with other documents. Of course, there are a great many words in any particular language, and typically the word count strategy ignores very common words that are present in large numbers in all documents, such as articles, conjunctions, and prepositions. Through histograms of meaningful words, documents can be clustered or embedded in kD spaces for analysis and visualization. Because of the large number of words (bins in the histogram), this analysis is often combined with PCA to produce a smaller set of descriptors, which make subsequent computation more tractable. Figure 6.20 shows an example of an embedding of a corpus of news articles by word counts. Notice that this kind of analysis can generally give information about the general *topic* of a document, but it loses the meaning of the document, because to discern meaning one must typically examine the semantics of individual sentences.

This general strategy for documents is referred to as a *bag of words* because it treats each document as a container (bag) for words that ignores the ordering of words, as well as the construction of phrases and sentences. This *bag* strategy has been used in a variety of contexts to deal with large sets of unstructured instances. For instance, in images, local features (measured through some time of detector and descriptor, such as corners or textures) are counted, and their location ignored, to determine the environment of an image or to quantify the similarity/difference between images in a large collection. Graphs are often unstructured and difficult to compare, but one can construct a *fingerprint* of a graph by quantifying different types of local neighborhood structures around vertices. Researchers have compared histograms of valences of vertices, numbers of cliques of different sizes, or, if the vertices have labels, categories of vertices based on their neighborhood structure [40].

If we consider a text document as a string of tokens (words), then the bag approach has been modified for many variations. For instance, besides documents, one might also need to quantify or give coordinates to words (this also helps in documents). In deciding if two words are similar, we can quantify how often they occur with or are near other words. In this way, cooccurrence (defined at some scale—phrase, sentence, document) becomes a signature for comparing words. This strategy has been incorporated into various neural net approaches, as described in subsequent sections.

Fig. 6.20 A visualization of a corpus of articles from [14] are organized in 2D based on associations with (probabilities) topics, which are derived from vectors of word counts, i.e., *bag of words* analysis

6.6.2 Edit Distance

Another common way of quantifying distances/similarities between unstructured or complex data types is to consider the cost of converting one instance to another. This is often done by describing a set of atomic *editing* operations on the data object and assigning a cost to each type of edit. For instance, in comparing the lexicographic words (ignoring their semantics), we could assign a cost to changing a letter in a word, as well as adding or deleting letters. Thus, to convert the word "Sunday" to "Saturday", we notice that first characters are the same, as are the last three. To convert the middle "un" to "atur", we might replace "n" with "r", and then insert a "t" and an "a". The precise edit sequence would depend on the cost of each operation, but the *edit distance* is typically the cost of the *least expensive edit* that converts one object to another. This edit distance is used extensively in genetics to compare genetic sequences (which are, essentially, strings). If the cost structure is properly constructed, this edit distance is computed efficiently using Dykstra's algorithm.

Another example of edit distance is in the analysis of graphs. Unaligned graphs are graphs where the vertices are not uniquely identified from one graph to another. Graphs with different types of nodes can be compared by removing, introducing, or changing the labels on nodes, and by allowing similar edits on edges. This kind

of edit distance can then be used to cluster ensembles of graphs or embed them in lower-dimensional spaces, or visualize their evolution in time. Graph edit distance is computationally challenging; it is NP-hard in general. However, special cases of graphs (e.g., acyclic) are compared more tractably, and approximate solutions are often quite effective.

6.6.3 Kernel Methods

A very useful tool in data analysis is to construct a similarity measure between pairs of instances for a particular data type and then rely on the *kernel method* or *kernel trick* to conduct analysis in the space induced by this similarity measure. Mercer's theorem states that for any *kernel*, $k(x_1, x_2)$, operating on pairs of data points/instances that it is guaranteed to produce a positive-definite inner product matrix, there is a corresponding Euclidean space for which this kernel is the Euclidean inner product (dot product). This technique allows one to define inner products to create high-dimensional spaces for which there might not be an explicit representation.

If the data points in the analysis have an associated metric space, then monotonically decreasing functions of point-to-point distance from a Mercer kernel. Indeed, a widely used kernel is the Gaussian function of distance:

$$k(x_1, x_2) = \exp\left(-\frac{|x_1 - x_2|^2}{2\sigma^2}\right), \tag{6.48}$$

where σ is a free parameter that must be tuned to a specific application. The Euclidean space associated with this kernel is not finite-dimensional, and has no explicit set of coordinates. All operations in the kernel space are represented in terms of inner products with the given data ensemble. However, this lifting of the data into the kernel space provides opportunities for

A variety of methods have been adapted for kernel spaces, including PCA (called *kernel PCA* [47]), clustering (also *spectral clustering*), regression, and classification (e.g., *support vector machines*). Kernels or inner products can be defined using bag-of-words strategies (kernel is typically a function of the product of the histograms), or some other distance measures on structured or unstructured data. The lifting of data into the kernel space has the advantages of (i) analysis operations without an explicit distance measure and (ii) moving the data into spaces where simpler models (e.g., linear) for regression, clustering, and classification are often more effective. Likewise, this kind of separation of data in the kernel space can aid in visualizing trends or clusters.

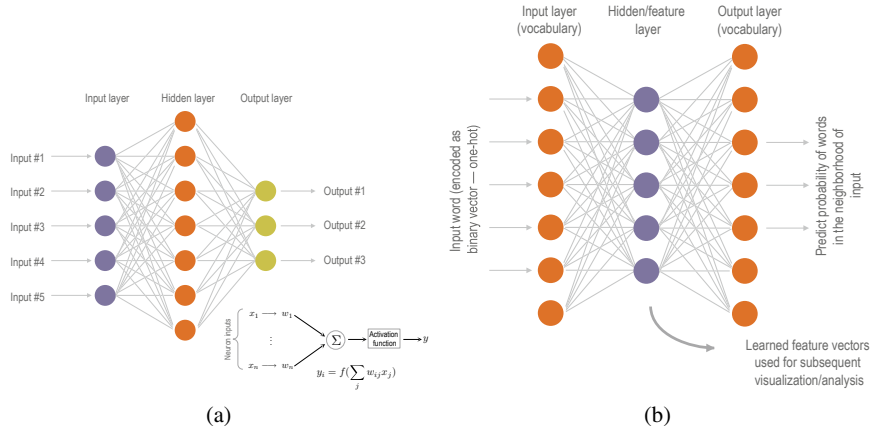


Fig. 6.21 **a** A neural network is a sequence of layers consisting of individual elements (neurons) that linearly combine outputs of the previous layer and perform a nonlinear activation (e.g., smooth threshold). **b** The skip-gram architecture for assigning words to vectors develops a feature vector that is effective at predicting the *context* of a given word—i.e., the probability of nearby words

6.6.4 Neural Networks

Recently, technology in the training and application of artificial neural networks has provided new opportunities for transforming data and embedding data sets into spaces that are well suited for analysis and visualization. There are a great many introductions or tutorials on neural networks, and here we assume that the reader is familiar with the basic technology, which we review briefly.

A neural network (NN) is a set of processing units, each of which performs a linear combination (weighted sum) of inputs and produces an output, which is a nonlinear function of that weighted sum:

$$y = \phi\left(\sum_j w_j x_j\right), \tag{6.49}$$

where x_j is a vector of inputs and w_j are the weights associated with the inputs to this particular neuron. The nonlinear *activation function*, $\phi : \Re \mapsto \Re$, is typically some kind of soft threshold. These individual elements are arranged in layers (where the elements of a layer share the same inputs, x_j), and the network transforms data by passing it through a sequence of layers, as in Fig. 6.21.

Most NNs are trained in a *supervised* manner, and the weights are modified incrementally so that the outputs of the network approximate the training data for any particular input. The training of NNs has an associated, and very extensive, set of methods, research, and theory.

The conventional wisdom is that the layers of a neural network perform a sequence of transformations on the input data, making the data progressively better suited to the task of the final layer, which must produce the desired output from a linear combination of input data. Thus, the network in its intermediate layers is successively transforming the data into spaces that are well suited to the task, and thus the intermediate layers represent transformations of the data that can aid in analysis and visualization. Besides the architecture and training of the network, the important issues for embedding are how to encode the inputs and outputs and how to define the supervised problem. Here we give an example that demonstrates the principles.

A classic problem in the analysis of text and documents is the vectorization of words, which is the assignment of each word to a coordinate in a space where distances reflect the similarities between the meaning and usage of words. Recent work in neural networks has addressed this problem by constructing and training NNs that learn associations between words in sentences. There are several versions of this architecture and here we give the *skip-gram* version of the method. The strategy, depicted in Fig. 6.21, is to train the network to predict nearby words in a sentence from a single word input. Words are typically coded as binary or *hot* vectors ("1" indicates the presence of the word), and thus the size of the network is proportional to the size of the vocabulary. The output is a vector (again the size of the vocabulary), where the signal indicates the probability that a given word appears within a window of nearby words (5–10 nearby words, typically). The hidden layer is constructed to be 100–1000 units and the output of the hidden layer (a vector) is used as the embedding for subsequent processing.

This example demonstrates the general strategy for using NNs to construct transformations. First, one must encode the input in a general manner that has the appropriate symmetries. Categorical data, for instance, is often best encoded with binary vectors, which also allows for sets or bags of examples. Second, the network must be constructed with a hidden layer appropriate for output into some other part of the visualization or analysis pipeline. This is sometimes called a *latent representation*. Third, the NN must be trained with a task that is appropriate for the transformations. In the word2vec method, the task is the prediction of context words (nearby words in sentences), which captures the meaning and usage of words (indeed, the ability of one word to replace another in a context), which has been shown to capture similarities between words.

References

1. Anscombe, F.J.: Graphs in statistical analysis. Am. Stat. **27**(1), 17–21 (1973). http://www.jstor.org/stable/2682899
2. Bihan, D.L., Mangin, J., Poupon, C., Clark, C.A., Pappata, S., Molko, N., Chabriat, H.: Diffusion tensor imaging: concepts and applications. J. Magn. Reson. Imaging **13**, 534–546 (2001)
3. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
4. Carlsson, G.: Topology and data. AMS Bull. **46**(2), 255–308 (2009)

5. Cates, J.E., Whitaker, R.T., Jones, G.M.: Case study: an evaluation of user-assisted hierarchical watershed segmentation. Med. Image Anal. **9**(6), 566–578 (2005)
6. Cui, W., Zhou, H., Qu, H., Wong, P.C., Li, X.: Geometry-based edge clustering for graph visualization. IEEE Trans. Vis. Comput. Graph. **14**(6), 1277–1284 (2008). https://doi.org/10.1109/TVCG.2008.135
7. Eberly, D.: Ridges in Image and Data Analysis. Kluwer Academic Publishers, Boston (1996)
8. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. Discret. Comput. Geom. **28**, 511–533 (2002)
9. Edelsbrunner, H., Harer, J., Natarajan, V., Pascucci, V.: Morse-Smale complexes for piecewise linear 3-manifolds. In: SCG'03: Proceedings of the 19th Annual Symposium on Computational Geometry, pp. 361–370. ACM, New York, NY, USA (2003)
10. Engel, K., Hadwiger, M., Kniss, J.M., Rezk-Salama, C., Weiskopf, D.: Real-Time Volume Graphics. A. K. Peters Ltd., Natick (2006)
11. Falk, M., Hotz, I., Ljung, P., Treanor, D., Ynnerman, A., Lundström, C.: Transfer function design toolbox for full-color volume datasets. In: Proceedings of the IEEE PacificVis'17 (2017)
12. Gerber, S., Bremer, P.T., Pascucci, V., Whitaker, R.: Visual exploration of high dimensional scalar functions. IEEE Trans. Vis. Comput. Graph. **16**(6), 1271–1280 (2010)
13. Gonzales, R.C., Woods, R.E.: Digital Image Processing, 2nd edn. Prentice Hall, Englewood Cliffs (2013)
14. Gretarsson, B., O'Donovan, J., Bostandjiev, S., Höllerer, T., Asuncion, A.U., Newman, D., Smyth, P.: Topicnets: visual analysis of large text corpora with topic modeling. ACM TIST **3**, 23:1–23:26 (2012)
15. Günther, T., Theisel, H.: The state of the art in vortex extraction. Comput. Graph. Forum **37**(6), 1–24 (2018)
16. Hadwiger, M., Kratz, A., Sigg, C., Bühler, K.: GPU-accelerated deep shadow maps for direct volume rendering. In: Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, GH'06, pp. 49–52. ACM, New York, NY, USA (2006)
17. Helman, J.L., Hesselink, L.: Visualizing vector field topology in fluid flows. IEEE Comput. Graph. Appl. **11**, 36–46 (1991)
18. Hernell, F., Ljung, P., Ynnerman, A.: Interactive global light propagation in direct volume rendering using local piecewise integration. In: Eurographics/IEEE VGTC Symposium on Volume and Point-Based Graphics, pp. 105–112. Eurographics Association (2008)
19. Hotz, I., Feng, L., Hagen, H., Hamann, B., Jeremic, B., Joy, K.I.: Physically based methods for tensor field visualization. In: VIS'04: Proceedings of IEEE Visualization 2004, pp. 123–130. IEEE Computer Society Press (2004)
20. Jönsson, D., Kronander, J., Ropinski, T., Ynnerman, A.: Historygrams: enabling interactive global illumination in direct volume rendering using photon mapping. IEEE Trans. Vis. Comput. Graph. **18**(12), 2364–2371 (2012)
21. Jönsson, D., Sundén, E., Ynnerman, A., Ropinski, T.: A survey of volumetric illumination techniques for interactive volume rendering. Comput. Graph. Forum **33**(1), 27–51 (2014)
22. Kasten, J., Reininghaus, J., Hotz, I., Hege, H.C., Noack, B.R., Daviller, G., Morzynski, M.: Acceleration feature points of unsteady shear flows. Arch. Mech. **68**(1), 55–80 (2016)
23. Kindlmann, G.: Superquadric tensor glyphs. In: Proceedings of the Joint Eurographics - IEEE TCVG Symposium on Visualization, pp. 147–154 (2004)
24. Kindlmann, G., San Jose Estepar, R., Smith, S.M., Westin, C.F.: Sampling and visualizing creases with scale-space particles. IEEE Trans. Vis. Comput. Graph. **15**(6), 1415–1424 (2009)
25. Kniss, J., Kindlmann, G., Hansen, C.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In: VIS'01: Proceedings of the Conference on Visualization'01, pp. 255–262. IEEE Computer Society (2001)
26. Kniss, J., Kindlmann, G., Hansen, C.: Multidimensional transfer functions for interactive volume rendering. IEEE Trans. Vis. Comput. Graph. **8**(3), 270–285 (2002)
27. Kratz, A., Auer, C., Stommel, M., Hotz, I.: Visualization and analysis of second-order tensors: moving beyond the symmetric positive-definite case. Comput. Graph. Forum - State Art Rep. **32**(1), 49–74 (2013)

28. Kratz, A., Auer, C., Hotz, I.: Tensor invariants and glyph design. In: Westin, C., Burgeth, B., Vilanova, A. (eds.) Visualization and Processing of Tensors and Higher Order Descriptors for Multi-valued Data (Dagstuhl'11), Mathematics and Visualization, pp. 17–33. Springer (2014)
29. Kratz, A., Schöneich, M., Zobel, V., Burgeth, B., Scheuermann, G., Hotz, I., Stommel, M.: Tensor visualization driven mechanical component design. In: Proceedings of Pacific Vis Conference (2014)
30. van Kreveld, M., van Oostrum, R., Bajaj, C., Pascucci, V., Schikore, D.: Contour trees and small seed sets for isosurface traversal. In: SCG'97 Proceedings of the 13th Annual Symposium on Computational Geometry (1997)
31. Liu, S., Maljovec, D., Wang, B., Bremer, P.T., Pascucci, V.: Visualizing high-dimensional data advances in the past decade. In: Borgo, R., Ganovelli, F., Viola, I. (eds.) Eurographics Conference on Visualization (EuroVis) - STARs, pp. 127–147. The Eurographics Association (2015)
32. Ljung, P., Krüger, J., Gröller, E., Hadwiger, M., Hansen, C.D., Ynnerman, A.: State of the art in transfer functions for direct volume rendering. In: Maclejewsk, R., Ropinski, T., Vilanova, A. (eds.) Computer Graphics Forum - STAR, vol. 35, p. 23 (2016)
33. Lorensen, W.E., Cline, H.E.: Marching cubes: a high resolution 3D surface construction algorithm. In: ACM SIGGRAPH Computer Graphics and Interactive Techniques, pp. 163–169 (1987)
34. Maadasamy, S., Doraiswamy, H., Natarajan, V.: A hybrid parallel algorithm for computing and tracking level set topology. In: Proceedings of the 19th International Conference on High Performance Computing, pp. 1–10 (2012). https://doi.org/10.1109/HiPC.2012.6507496
35. Maciejewski, R., Chen, W., Woo, I., Ebert, D.: Structuring feature space - a non-parametric method for volumetric transfer function generation. IEEE Trans. Vis. Comput. Graph. **15**(6), 1473–1480 (2009)
36. Marr, D., Hildreth, E.: Theory of edge detection. Proc. R. Soc. Lond. Ser. B Biol. Sci. **207**(1167), 187–217 (1980)
37. McLoughlin, T., Laramee, R.S., Peikert, R., Post, F.H., Chen, M.: Over two decades of integration-based, geometric flow visualization. Comput. Graph. Forum - State Art Rep. **29**(6), 1807–1829 (2010)
38. Meyer, M., Kirby, R.M., Whitaker, R.: Topology, accuracy, and quality of isosurface meshes using dynamic particles. IEEE Trans. Vis. Comput. Graph. **12** (2007)
39. Munzner, T.: Visualization Analysis and Design. CRC Press, Taylor & Francis Group, Boca Raton (2014)
40. Neumann, M., Garnett, R., Bauckhage, C., Kersting, K.: Propagation kernels: efficient graph kernels from propagated information. Mach. Learn. **102**(2), 209–245 (2016). https://doi.org/10.1007/s10994-015-5517-9
41. Pintado, S., Jörnsten, R.: Functional analysis via extensions of the band depth. IMS Lect. Notes-Monogr. Ser. **54** (2007). https://doi.org/10.1214/074921707000000085
42. Pöthkow, K., Weber, B., Hege, H.C.: Probabilistic marching cubes. Comput. Graph. Forum **30**(3) (2011)
43. Raj, M., Whitaker, R.T.: Visualizing multidimensional data with order statistics. Comput. Graph. Forum **37**, 277–287 (2018). https://doi.org/10.1111/cgf.13419
44. Raj, M., Mirzargar, M., Kirby, R.M., Whitaker, R.T., Preston, J.S.: Evaluating alignment of shapes by ensemble visualization. IEEE Comput. Graph. Appl. **36** (2015). https://doi.org/10.1109/MCG.2015.70
45. Raj, M., Mirzargar, M., Ricci, R., Kirby, R., Whitaker, R.T.: Path boxplots: a method for characterizing uncertainty in path ensembles on a graph. J. Comput. Graph. Stat. **26** (2016). https://doi.org/10.1080/10618600.2016.1209115
46. Rosanwo, O., Petz, C., Hotz, I., Prohaska, S., Hege, H.C.: Dual streamline seeding. In: IEEE Pacific Visualization Symposium'09, pp. 9–16 (2009)
47. Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)

48. Serfling, R.: A depth function and a scale curve based on spatial quantiles. In: Dodge, Y. (ed.) Statistical Data Analysis Based on the L1-Norm and Related Methods, pp. 25–38. Birkhäuser, Basel (2002)
49. Stegmaier, S., Strengert, M., Klein, T., Ertl, T.: A simple and flexible volume rendering framework for graphics-hardware–based raycasting. In: Proceedings of the International Workshop on Volume Graphics'05, pp. 187–195 (2005)
50. Sun, Y., Genton, M.G.: Functional boxplots. J. Comput. Graph. Stat. **20**(2), 316–334 (2011). https://doi.org/10.1198/jcgs.2011.09224
51. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290**(5500), 2319–2323 (2000). https://doi.org/10.1126/science.290.5500.2319
52. Tukey, J.W.: Mathematics and the picturing of data. In: James, R. (ed.) Proceedings of the International Congress of Mathematicians, pp. 523–531 (1975)
53. Tukey, J.W.: Exploratory Data Analysis. Addison-Wesley, Reading (1977)
54. van der Maaten, L., Hinton, G.: Visualizing high-dimensional data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
55. Ware, C.: Information Visualization, 3rd edn. Morgan Kaufmann, San Francisco (2013)
56. Whitaker, R.T.: Reducing aliasing artifacts in iso-surfaces of binary volumes. In: IEEE Symposium on Volume Visualization (2000)
57. Whitaker, R.T., Mirzargar, M., Kirby, R.: Contour boxplots: a method for characterizing uncertainty in feature sets from simulation ensembles. IEEE Trans. Vis. Comput. Graph. **19**, 2713–2722 (2013). https://doi.org/10.1109/TVCG.2013.143