Assignment Report

Data Visualization

TOPIC:-

VISUALIZATION TYPES, MODELS and APPLICATIONS

Team members :

Bindu K 20201ISE0059

DeepaRani B A 20201ISE0060

Chandana J 20201ISE0090

K Hruthika Ruth 20201ISE0077

Nishchay B N 20201ISE0086

Sridar S 20201ISE0093

Manoj B S 20201ISE0084

Vikas H 20201ISE0056

Gagan Urs 20201ISE0064

INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **SL.NO** | **TOPICS** | **PAGE NO** | **DONE BY** |
| 01. | Scalar Visualization | 3-14 | Gagan and Manoj |
| 02. | Vector Visualization | 15-25 | Bindu and Chandana |
| 03. | Tensor Visualization | 26-47 | Sridar , Vikas and Hruthika |
| 04. | Domain Modelling Visualization | 48-62 | Deeparani and Nishchay |
| 05. | Summary | 63-65 |  |
| 06. | Exercise | 66-67 |  |
| 07. | Reference | 68-69 |  |

# INTRODUCTION

Introduction Data visualization is the graphical representation of information and data. By using visual elements such as charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data[1]. In the contemporary digital age, where vast amounts of data are generated every second, the ability to interpret and communicate this data effectively has become crucial [2]. Data visualization not only aids in making sense of complex data sets but also enhances the decision-making process by presenting data in a clear and actionable format.[3] The roots of data visualization can be traced back to the 17th century, with early examples like John Snow’s cholera map in 1854 and Florence Nightingale’s coxcomb charts in the 1850s [4]. These historical examples demonstrate the long-standing importance of visualizing data to understand and solve problems. In the modern era, advancements in technology have transformed data visualization from static images into dynamic and interactive experiences. Modern tools and software, such as Tableau, Power BI, and D3.js, empower users to create sophisticated visualizations that can be easily shared and interpreted across various platforms. Data visualization serves multiple purposes. It helps in simplifying complex data sets, making large amounts of information comprehensible at a glance.

This simplification is crucial in various fields, from business analytics to scientific research, where understanding data quickly and accurately can lead to better insights and decisions. Furthermore, data visualization is essential in storytelling, allowing data to tell a story that highlights key messages and insights that might be overlooked in text-based data presentations. One of the most compelling aspects of data visualization is its ability to reveal insights that are not immediately obvious. Patterns, correlations, and trends can emerge through visual representation, enabling analysts and decision-makers to identify opportunities 1 and risks that might not be evident through raw data alone. Effective visualizations leverage principles of design and cognitive science to present data in ways that enhance comprehension, retention, and decision-making.

They use color, size, shape, and motion to highlight relationships and trends, making complex information more accessible and understandable. The role of data visualization has become even more critical in an era dominated by big data and analytics. Organizations across various sectors, including business, healthcare, science, and public policy, rely heavily on data-driven insights to inform their strategies and operations. In business, data visualization helps in identifying market trends, customer behaviors, and operational efficiencies. In healthcare, it aids in tracking disease outbreaks, patient outcomes, and treatment efficacy. In science, visualizations are used to explore research data, uncover new findings, and communicate results to the broader community. In public policy, they help in understanding and addressing societal issues, from economic trends to environmental impacts. As the volume of data continues to grow exponentially, so does the need for effective data visualization techniques.

The increasing complexity of data requires innovative approaches to visualization, combining the latest technological advancements with robust analytical methods. This report will explore the fundamental concepts of data visualization, examine the tools and techniques used by professionals, and highlight best practices for creating impactful visualizations. By understanding these key elements, we can better harness the power of data to drive innovation and inform decisions. In summary, data visualization is more than just a tool for making data look attractive; it is a critical process for interpreting and communicating information in a way that is both insightful and actionable.

As we delve deeper into the world of data visualization, we will uncover the principles that make visualizations effective, the technologies that enable their creation, and the practical applications that demonstrate their value across various domains

**SCALAR VISUALIZATION**

# Introduction

Visualizing scalar data is a fundamental task in various fields such as science, engineering, medicine, and even in daily life applications. Scalar datasets, also known as scalar fields, represent functions 𝑓:𝐷→𝑅*f*:*D*→R, where 𝐷*D* is typically a subset of 𝑅2R2 or 𝑅3R3. These datasets describe how a scalar quantity varies over space, and effectively visualizing this variation is crucial for analysis and decision-making. Scalar data visualization transforms numerical data into graphical representations, making complex information easier to understand and interpret.

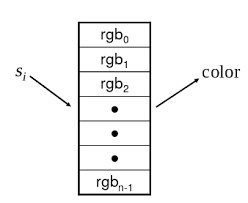
Scalar data visualization is crucial for several reasons. It can reveal patterns, trends, and anomalies that are not immediately apparent from raw data alone. For example, in medicine, visualizing scalar data from medical imaging can help in diagnosing diseases by highlighting abnormalities in tissues. In meteorology, visualizing temperature or pressure fields aids in weather forecasting. In engineering, stress and strain fields in materials can be visualized to identify potential points of failure.

Several techniques have been developed for visualizing scalar fields, each suited to different types of data and specific purposes. The choice of technique often depends on the nature of the dataset and the specific insights required. Among the most popular techniques are color mapping, contouring, and height plots. Each of these methods offers unique advantages and can be used to highlight different aspects of the data. the visualization of scalar data using techniques such as color mapping, contouring, and height plots allows for the effective interpretation and analysis of complex datasets. Each method has its strengths and is chosen based on the nature of the data and the specific insights required. As we delve deeper into these techniques, we will explore their applications, advantages, and the contexts in which they are most useful. Understanding these visualization techniques and their proper application is essential for extracting meaningful insights from scalar data, thereby aiding in research, decision-making, and problem-solving across various domains. As technology advances, the development of new visualization tools and techniques continues to enhance our ability to understand and manipulate scalar data, pushing the boundaries of what is possible in data analysis and interpretation.

# Color Mapping

Color mapping is a widely used scalar visualization technique that assigns colors to scalar data, displaying these colors on a computer system. This process involves using scalar values as indices to access a color lookup table. The lookup table, containing an array of colors, is associated with a defined minimum and maximum scalar range. Scalar values within this range are mapped to corresponding colors in the table, while values exceeding the maximum are mapped to the maximum color, and values below the minimum are mapped to the minimum color.

Then, for each scalar value si , the index i into the color able with n entries is given as:



## Transfer Functions

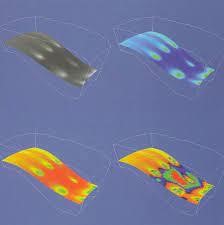
A more general form of the lookup table is called transfer function. A transfer function is any expression that maps scalar values into a color specification. For example, a function can be used to map scalar values into separate intensity values for the red, green, and blue components. Transfer functions can also be used to map scalar data to other attributes, such as local transparency. This concept will be explored further in the context of volume rendering.

Essentially, a lookup table represents a discrete sampling of a transfer function. By sampling a transfer function at specific discrete points, we can construct a lookup table from it.

Color mapping is a one-dimensional visualization technique. It maps one piece of information (i.e. a scalar value) into a color specification. However, the display of color information is not limited to one-dimensional displays. Often we use color information mapped onto 1-D, 2-D, or 3-D objects. This is a simple way to increase the information content of our visualization. In 3-D, cutting planes can be used to visualize the data inside.

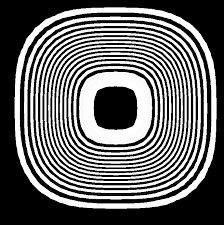
The effectiveness of color mapping for visualization hinges on the careful selection of lookup table entries. Crafting these tables is both an art and a science. Practically speaking, lookup tables should highlight significant features and downplay less important or irrelevant details. It is also beneficial to use color palettes that naturally convey scaling information. For instance, a rainbow color scale ranging from blue to red is commonly used to depict temperature, as people typically associate blue with cold and red with heat.

Examples:



In certain applications, emphasizing data variations rather than absolute values is important. This approach is useful for identifying areas in the dataset where the data changes rapidly or remains constant. To achieve this, a color map with two or more distinctly different alternating colors can be used. As the data values change, the colors shift abruptly, creating noticeable band-like patterns in the visualization that are easy to detect.

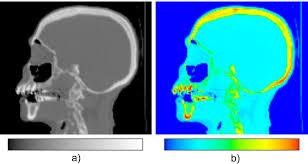
Example:



Zebra Color Map

There are numerous possible designs for color maps. For example, in geographical applications, landscape elevation is often depicted using a color map with colors that represent typical terrain features at different heights: blue for sea level, green for fields, beige for hills, brown for mountains, and white for mountain peaks. In medical imaging, a simple luminance color map is often more effective. Using a rainbow color map in this context can distort linearity, as the mapping of color values to hues can cause some users to perceive colors changing "faster" per spatial unit in the higher yellow-to-red range compared to the lower blueto-cyan range.

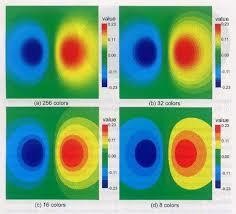
Example:



Illuminance color Map Rainbow color Map

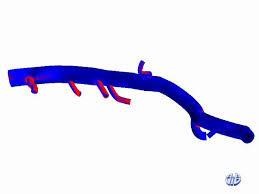
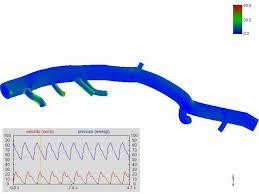
Another critical factor in color map design is selecting the appropriate number of colors, 𝑁*N*. Opting for a small 𝑁*N* often results in color banding, a common issue experienced by computer users who have reduced the number of colors in an image using processing software. Mathematically, color banding creates artifacts similar to those produced by undersampling the scalar signal range. For instance, if we use only 32 colors to visualize a dataset, the outcome is nearly the same as first undersampling the scalar signal to 5 bits and then visualizing it with a high number of colors. This highlights the importance of choosing a sufficient number of colors to avoid such artifacts and ensure a smooth representation of the data.

Example:



Additionally, the preceding images underscore the necessity of including a legend with color maps, elucidating how the colors correspond to the original values. Without this contextual information, the color representation may be virtually meaningless, lacking a frame of reference. By furnishing this context, users can better interpret the image and understand the visualization of the data.

Visualization without color map: Visualization with color Map:

## Contouring

Contouring is a logical progression from color mapping. When we observe a surface colored with data values, our eyes naturally distinguish similarly colored areas as separate regions. By contouring data, we essentially delineate the boundaries between these regions. These boundaries manifest as contour lines in two dimensions or contour surfaces in three dimensions, each representing a constant scalar value.

For instance, in 2D contour displays, we commonly encounter weather maps adorned with lines indicating constant temperature (isotherms), or topographical maps illustrating lines of constant elevation. These contour lines provide valuable insights into the spatial distribution and variation of scalar quantities, facilitating easier interpretation of complex datasets.

Isosurfaces, also known as three-dimensional contours, can be represented by various polygonal primitives. These isosurfaces depict regions where a scalar quantity remains constant. For instance, in medical imaging, isosurfaces correspond to constant image intensity levels representing various body tissues such as skin, bone, or organs. It's worth noting that the specific isovalue corresponding to the same tissue may vary among different scans. Additionally, abstract isosurfaces, such as those indicating constant pressure or temperature in fluid flow, can also be generated to visualize physical phenomena beyond medical contexts.



To initiate the generation of a 2-D isocontour, we start with a regular grid where scalar values are assigned to the grid nodes. The first step in contouring is to choose a scalar value, known as the isovalue or contour value, which determines the contour lines or surface to be generated. Using linear interpolation on the regular grid, we can identify locations along the edges of the grid where the data assumes the isovalue.

For instance, if we have an edge with scalar values 10 and 0 at its two endpoints, and we aim to generate a contour line with a value of 5, then the contour will pass through the midpoint of that edge. This process is repeated for all edges in the grid, resulting in the construction of the desired isocontour.

Once the points on all edges are generated, we can connect these points into contours using a few different approaches. One approach detects an edge intersection, i.e. the contour passes through an edge, and then tracks this contour as it moves across cell boundaries. We know that if a contour edge enters a cell, it must exit a cell as well. The contour is tracked until it closes back on itself, or exits a data set boundary.

## Marching Squares

An alternative method employs a divide and conquer strategy, treating cells independently. Known as the marching squares algorithm, this approach assumes that a contour can traverse a cell in only a finite number of ways due to the linear interpolation utilized. A case table is then constructed, cataloging all conceivable topological configurations of a cell, based on combinations of scalar values at the cell's points. This systematic approach enables efficient determination of contour paths through each cell, facilitating the generation of isocontours across the entire dataset.

While trying this algorithm on different configurations we realize that some cases may be ambiguous. That is the situation for the squares 5 and 10.

As depicted in the preceding image, making interpretations in such situations can be challenging. Nevertheless, it's essential to note that these exceptions don't indicate any actual errors, as the edges remain closed.

## Marching Cubes

In 1987, Lorensen and Cline introduced the Marching Cubes algorithm, detailed in their paper titled "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," published in the ACM Computer Graphics journal, Volume 21, No. 4 (SIGGRAPH 1987 Proceedings).

Marching Cubes (MC) offers an efficient approach for extracting isosurfaces from scalar datasets defined on regular grids. Much like the marching squares algorithm in two dimensions, Marching Cubes computes surface segments for each cell of the grid, approximating the isosurface in three dimensions. This method has become a cornerstone in three-dimensional surface reconstruction and visualization, enabling the representation of complex scalar data in a comprehensible manner.

## Computation of normal vectors

The quality of the resulting representation of the extracted isosurface can be enhanced by computing the normal vectors of all vertices. This improvement leverages the fact that the gradient of the scalar function is always orthogonal to the isosurface. Marching Cubes approximates the gradient at the vertices of the grid as and interpolates linearly to determine the gradient at the intersection.

## Problems with Marching Cubes

After the article about Marching Cubes was published it turned out that the isosurfaces extracted using Marching Cubes can contain holes under certain circumstances due to ambiguities in the case table. Several follow-up papers exist to fix several issues with

Marching Cubes.

**Variants**

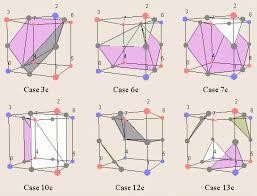
There are variants of Marching Cubes for triangles and tetrahedra as well.

## Coping with the Ambiguities

The original set of cases for creating triangles within the cells to generate an isosurface can create holes in some cases. This is basically due to ambiguities, i.e. there are more than one way to generate triangles for some cases. Hence, by introducing additional cases to our case table we can cope with the ambiguities. Obviously, in order to decide which configuration to use, we also have to look at the neighboring cells.

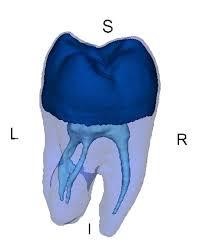
To address topology errors, such as holes in the 3D model, six additional cases have been incorporated into the marching cubes algorithm. These cases are intended to complement the standard cases and must be used accordingly. For example, in the scenario depicted in the previous image, one would utilize case 6c instead of the standard complementary case 6. The list of these new cases is provided on the following slide.

Additional Cases:



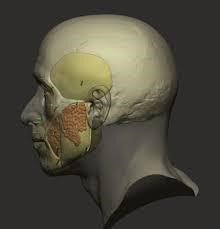
In visualization, it's possible to depict multiple isosurfaces from the same dataset simultaneously. This approach is often coupled with transparency, enabling the visualization of several isosurfaces concurrently. Rendering multiple nested semitransparent isosurfaces corresponding to a discrete sequence of isovalues can be extended to the continuous case, as we will explore when discussing volume rendering techniques. This enables the creation of complex visualizations that provide insights into the underlying scalar data across a range of values.

Consider an example where multiple isosurfaces are visualized simultaneously. The blue opaque isosurface represents a high isovalue, indicating a hard material akin to enamel present on the tooth's upper surface. Meanwhile, the beige isosurface corresponds to a lower isovalue, indicative of dentine material within the tooth. Within the tooth, one can observe the nerve chamber. This visualization effectively captures the internal structure of the tooth, highlighting different materials and their spatial relationships within.



It's important to approach the interpretation of marching cubes results with caution. For instance, small holes in the output, smaller than the dataset resolution, may arise due to inherent ambiguities in the data. Moreover, the appearance of waviness in an isosurface doesn't always indicate a genuine feature of the data; rather, it can result from subsampling the original signal on a low-resolution uniform grid. Being aware of these factors is crucial when analyzing and making interpretations based on marching cubes-generated isosurface visualizations.

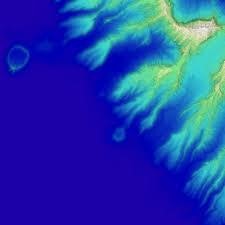
Example



## Scalar Generation

Scalar generation involves transforming raw data into a format suitable for visualization using techniques such as color mapping and contouring. These methods are simple and effective for displaying scalar information, making them a natural choice for initial visualization attempts. However, data may not always be in a convenient form for these techniques. It might not be single-valued, resembling a scalar, or it could involve complex mathematical relationships. This presents a creative challenge in visualization: we must harness our creativity to convert data into a format conducive to visualization. This process of transforming data for visualization is both engaging and rewarding, requiring innovative approaches to effectively convey information visually.

Consider a terrain data set. We assume that the data is given as x-y-z coordinates, where x and y represents the coordinates in the plane, and z represents the elevation above sea level. Our desired visualization is to color the terrain according to elevation. This requires creating a colormap – possibly using white for high altitude, blue for sea level and below, and various shades of green and brown corresponding to elevation between sea level and high altitude. We also need scalars to index into the colormap. The obvious choice here is to extract the zcoordinate. That is, scalars are simply the z-coordinate. The resulting visualization may look like this:



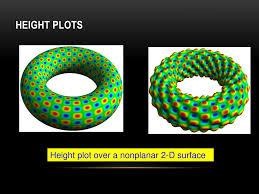
## Height Plots

Height plots, also known as elevation or carpet plots, were introduced earlier in the introduction. Given a two-dimensional surface 𝐷𝑠⊂𝐷*Ds*⊂*D*, which is part of a scalar dataset 𝐷*D*, height plots can be described by the mapping operation:

𝑚:𝐷𝑠→𝐷,𝑚(𝑥)=𝑥+𝑠(𝑥)𝑛(𝑥),∀𝑥∈𝐷𝑠*m*:*Ds*→*D*,*m*(*x*)=*x*+*s*(*x*)*n*(*x*),∀*x*∈*Ds*

Here, 𝑠(𝑥)*s*(*x*) represents the scalar value of 𝐷*D* at the point 𝑥*x*, and 𝑛(𝑥)*n*(*x*) is the normal to the surface 𝐷𝑠*Ds* at 𝑥*x*. In simpler terms, the height plot mapping operation "warps" a given surface 𝐷𝑠*Ds* included in the dataset along the surface normal with a factor proportional to the scalar values. This operation effectively alters the surface geometry based on the scalar values, providing a visual representation of the dataset that emphasizes variations in elevation.

Example: a torus surface (a) and its “warped” variant with the height corresponding to the scalar value.



## Conclusion

Scalar visualization techniques play a crucial role in understanding and interpreting data across various fields, including science, engineering, medicine, and beyond. Throughout this exploration, we've delved into several key methods for visualizing scalar data, including color mapping, contouring, and height plots, each offering unique advantages and insights into the underlying datasets.

Color mapping provides a visually intuitive representation by assigning colors to scalar values, enabling the quick identification of patterns and variations in the data. Contouring, on the other hand, delineates regions of constant scalar value, aiding in the visualization of gradients and distribution across 2D and 3D datasets. Height plots, also known as elevation or carpet plots, offer a way to visually emphasize elevation variations by warping surfaces along surface normals proportional to scalar values.

Additionally, we've explored advanced techniques such as Marching Cubes for generating isosurfaces from scalar data, enhancing our ability to visualize complex three-dimensional datasets. While these methods offer powerful tools for data visualization, it's essential to approach their interpretation with care, considering factors such as dataset resolution, subsampling, and potential topology errors.

In conclusion, scalar visualization techniques empower researchers, engineers, and practitioners to gain valuable insights from complex datasets, aiding in decision-making, problem-solving, and discovery across diverse domains. As technology continues to advance, the development of innovative visualization techniques promises to further expand our understanding and utilization of scalar data in the future.

In summary, scalar visualization serves as a bridge between raw data and meaningful insights, allowing us to perceive and comprehend complex phenomena in a visual manner. By leveraging techniques such as color mapping, contouring, and height plots, we can extract valuable information from scalar datasets, uncovering patterns, trends, and relationships that may otherwise remain hidden. As we continue to refine and innovate in the field of data visualization, scalar visualization techniques will undoubtedly remain indispensable tools for understanding the world around us.

VECTOR VISUALIZATION

# Introduction

Vector data is a fundamental and significant type of data, just as scalar data is. In technical terms, a vector can be defined as an ordered set of n scalar components v = (v1,...,vn), where each component vi belongs to the set of real numbers (R). Vectors with n dimensions are used to represent various physical properties such as position, direction, velocity, or force in Rn. Although vectors can have any number of dimensions, many visualization applications focus on visualizing phenomena in two- or three-dimensional space. As a result, most visualization software simplifies vectors by defining them to have only three components. In the world of visualization software, 2D vectors are commonly represented as 3D vectors with a null third component. While it is possible to create specific implementation methods for 2D vectors, doing so would greatly complicate the overall structure of the software and result in duplicated code, ultimately decreasing performance. As discussed in Section , vector fields are defined as functions f : D → R3, where D is typically a subset of either R2 or R3. Vector datasets are essentially samples of vector fields taken at discrete spatial domains. In this chapter, we shall discuss several popular visualization methods for vector datasets: vector glyphs, vector color coding, displacement plots, stream objects, texture-based vector visualization, and the simplified representation of vector fields.

One highly significant field where vector visualization plays a crucial role is computational fluid dynamics (CFD). CFD simulations can forecast the dynamic behavior of 3D fluid flows that may consist of multiple interacting substances with varying densities and pressures, in complex spatial geometries. The solution to a CFD simulation comprises several datasets, each corresponding to a different time step. For every time step, various attributes such as velocity, pressure, density, flow divergence, and vorticity are calculated and recorded in the solution dataset.

As divergence and vorticity are key concepts in comprehending vector fields and are utilized by many visualization methods for vector data, we will outline these concepts first. We will begin by explaining some fundamental mathematical operators commonly employed for analyzing vector fields. Then we will introduce vector glyphs, one of the most straightforward and widely used techniques in such fields. This article discusses the application of scalar visualization techniques in depicting vector fields. The paper then introduces the displacement plot technique, which utilizes integral methods to create paths in vector fields for visual representation. In Section , textures are explored as another approach for visualizing vector fields. Additionally, various strategies for simplifying vector datasets are also discussed. The article presents a range of illustrative visualization techniques specifically tailored for vector fields, offering an alternative means of simplified representation compared to the techniques.

# Divergence and Vorticity

Divergence and vorticity are important quantities for vector field visualization, but also for the visualization and processing of other types of datasets, such as meshes, images, and scalar and tensor fields.

Divergence. Given a vector field v : R3 → R3, the divergence of v = (vx, vy, vz)1 is the scalar quantity

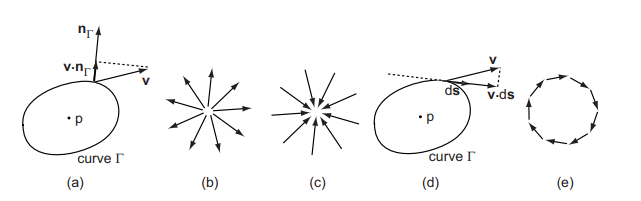
div v = ∂vx /∂x + ∂vy /∂y + ∂vz/ ∂z .

Intuitively, if v is a flow field that transports mass, div v characterizes the increase or loss of mass at a given point p in the vector field in unit time.

• A positive divergence at p denotes that mass would spread from p outward. Positive divergence points are called sources.

• A negative divergence at p denotes that mass gets sucked into p. Negative divergence points are called sinks.

• A zero divergence at p denotes that mass is transported without getting spread or sucked, i.e., without compression or expansion.



# Divergence and curl in 2D.

(a)Divergence construction.(b) Source point. (c) Sink point. (d) Rotor construction. (e) High-vorticity field.

An equivalent definition of the divergence of v at a point p is



Here, Γ is a closed hypersurface (a curve for 2D vector fields and a surface for 3D vector fields) around the current point p, |Γ| is the area (2D) or volume (3D) of the space enclosed by Γ, and nΓ is the outward normal of Γ. The integral in Equation computes the flux that the vector field transports through the imaginary boundary Γ. The limit Γ → p describes a curve that shrinks around the current point p until it becomes infinitesimally short.

shows the divergence of a 2D flow field using a blue-to-red colormap. The vector field is visualized with arrow glyphs for illustration purposes. Red areas indicate high positive divergence or sources. Two such sources are visible. Blue areas indicate high negative divergence or sinks. Within the dark blue area, two pronounced sinks are visible. If we correlate the divergence and vector glyph visualizations, we get the image of a flow field that emerges from the sources and ends up in the sinks.

Vorticity. Given a vector field v: R3 → R3, the vorticity of v, also called the curl or rotor of v2, is the vector quantity.



The vorticity rot v of v is a vector field that is locally perpendicular to the plane of rotation of v and whose magnitude expresses the speed of angular rotation of v around rot v. Hence, the vorticity vector characterizes the speed and direction of rotation of a given vector field at every point. In some textbooks, the rotor is also denoted as curl v.

An equivalent definition of the vorticity of v at a point p is given using line integrals. If we take any plane Π passing through p, and n is normal to Π, and consider a closed curve Γ ⊂ Π around p, then the projection of rot v on n is given by

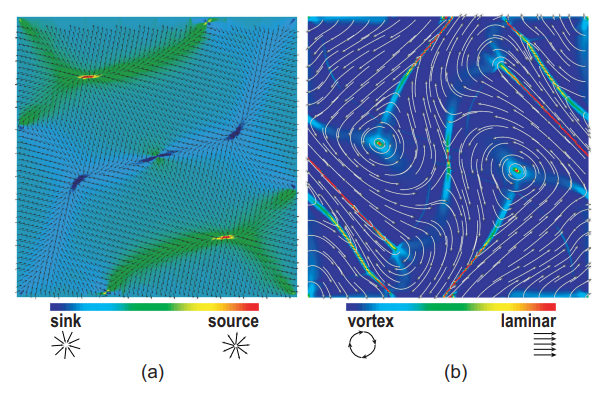


where ds is the infinitesimal oriented displacement on Γ.

Vorticity is a measure that indicates the presence of vortices within a vector field. A vortex can be informally defined as a region in which the vector field exhibits circular motion around a central point known as the vortex center. High levels of vorticity indicate the presence of vortices within the vector field. It is worth noting that high levels of vorticity and high levels of divergence are often inverse to each other. This means that a high-vorticity field will have zero divergence, while a high-divergence field will have zero vorticity. However, this is not always true, as there are instances where both divergence and vorticity can be high in vector fields that spiral around a central point and either converge or diverge away from it. Furthermore, when using both divergence and curl metrics to create visualizations, it is important to remember that div rot v = 0 for any vector field v.

This visualization displays the absolute value of vorticity in a velocity field from a magnetohydrodynamic (MHD) simulation, as outlined in [Brandenburg 03]. The colormap used ranges from blue to red, indicating low and high vorticity respectively. Stream tubes with arrow caps are employed to visualize the field, using a technique explained in detail. Blue regions represent laminar areas with low vorticity while red areas indicate regions with high vorticity. It is evident that there are two types of high-vorticity regions: small circular red spots which denote localized vortices, and elongated thin red strips which represent separation lines where the vector field rapidly changes direction. To identify these high-vorticity areas, one can simply contour the vorticity field for high iso values and select data points within this contour. This method provides reasonably accurate results and can be utilized for locating such features of interest.

This visualization displays the absolute value of vorticity in a velocity field from a magnetohydrodynamic (MHD) simulation, as outlined in [Brandenburg 03]. The colormap used ranges from blue to red, indicating low and high vorticity respectively. Stream tubes with arrow caps are employed to visualize the field, using a technique explained in detail. Blue regions represent laminar areas with low vorticity while red areas indicate regions with high vorticity. It is evident that there are two types of high-vorticity regions: small circular red spots which denote localized vortices, and elongated thin red strips which represent separation lines where the vector field rapidly changes direction.



(a)Divergence of a 2D vector field. (b) The absolute value of vorticity of a 2D vector field.

To identify these high-vorticity areas, one can simply contour the vorticity field for high iso values and select data points within this contour. This method provides reasonably accurate results and can be utilized for locating such features of interest.

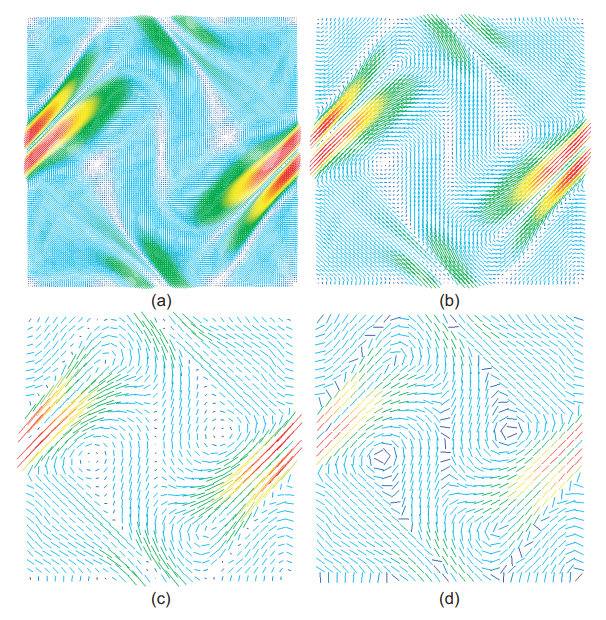
# Vector Glyphs

Vector glyphs are probably the simplest, fastest, and most popular technique for visualizing vector fields. The vector glyph mapping technique essentially associates a vector glyph, or vector icon, with every sample point of the vector dataset. Various properties of the icon, such as location, direction, orientation, size, and color, are adjusted to reflect the value of the vector attribute it represents. The name glyph, meaning “sign” in Greek, demonstrates this principle of associating discrete visual signs with individual vector attributes. Every glyph is a sign that conveys, by its appearance, properties of the represented vector, such as direction, orientation, and magnitude.

# Line glyphs

There are many variations of this framework for vector glyphs. Essentially, they propose various trade-offs between sampling density (how many glyphs we can display on a given screen area) and the number of encoded attributes (how many attributes we can display per glyph). We shall present a number of vector glyphs, starting with the simplest one: the line. Lines essentially show the position, direction, and magnitude of a set of vectors, given a vector dataset. Vector Glyphs 189 defined on a sampled domain D, we associate a line l = (x, x+kv(x)) with every sample point x ∈ D that has a vector attribute v(x). The parameter k represents the scaling factor used to map the vector magnitudes to the geometric domain. Oriented line glyphs are sometimes also called hedgehogs, due to the particularly spiky appearance of the visualization.

Figure shows a line glyph, or hedgehog, visualization of a 2D vector field defined on a square domain. The vector field is the MHD dataset earlier used in



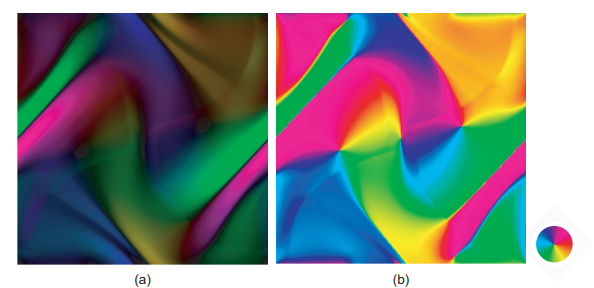
Hedgehog visualization of a 2D magnetohydrodynamic velocity field. (Data courtesy of Prof. Martin Rumpf, University of Bonn, Germany.

The original uniform dataset has a resolution of 256× 256 sample points. The images show the hedgehog visualization of the vector field uniformly subsampled in both x and y dimensions at a rate of 2 and a rate of 8. In all these images, the line glyphs are scaled proportionally to the vector field magnitude, the scaling factor k being proportional to the subsampling rate. In Figure, the vector field is uniformly subsampled at a rate of 8, but the line glyphs are all scaled to the same length. The glyphs are colored by color mapping the vector field magnitude scalar field to blue-to-red colormap. In this way, color cues strengthen (or replace) length cues to convey information about the vector magnitude. In many applications, color is used to show other scalar fields related to the vector field, such as pressure, temperature, or density.

Looking at the Figure, we can make several important observations. First, high-resolution vector datasets must be subsampled to be visualized with hedgehogs. Comparing Figures (a), (b), and (c), we can argue that it is easier to comprehend the vector field in the last image than in the first two, as the line glyphs are longer, hence their direction and orientation are easier to discern. The direction is even easier to follow in the last image (Figures (d)), where all glyphs have the same, relatively large, size. Hence, the clarity of hedgehog visualizations depends strongly on the glyph scaling factor. Ideally, a glyph should be as large as possible, since larger glyphs have an easier perceivable direction, but are not too large, so it would not intersect neighboring glyphs. If we scale all glyphs to the same size, as in Figure (d), this constraint is easy to obey by scaling each glyph to the average cell size at its origin. This removes clutter but eliminates the use of the glyph size (length) as a visual cue for the vector field magnitude. If we scale the glyphs to reflect the vector field magnitude, such as in Figures (a)–(c), eliminating clutter is more problematic. We could still use a unique glyph scaling factor k so that all glyphs are locally smaller than the cell size. Another option is to use a nonlinear term kv, which, e.g., has constrained minimal and maximal values or has a logarithmic, instead of linear, variation with |v|. This will prevent clutter and guarantee glyph visibility but will drop the one-to-one relationship between vector magnitude and glyph length.

# Vector Color Coding

As we have seen in the previous section, dense visualizations, such as color-mapped surfaces, have several advantages compared to sparse visualizations, such as glyphs. The natural question that arises is whether we can develop dense visualizations for vector fields, similar to the color-mapped surfaces used for scalar fields. One of the simplest techniques to produce such visualizations is vector color coding. Similar to scalar color mapping, vector color coding as



Vector color coding. (a) Orientation and magnitude. (b) Orientation only.

sociates a color with every point of a given surface on which we have defined a vector dataset. The color is used to encode the vector orientation and direction attributes. Vector color coding can be easiest understood if we represent colors in the hue-saturation-value (HSV) system. Colors in the HSV system can be visualized using a so-called color wheel, such as the one shown at the right in Figure 6.10. Every distinct hue corresponds to a different angle of the color wheel: red is 0◦, magenta is 60◦, blue is 120◦, cyan is 180◦, green is 240◦, and yellow is 300◦. Saturation is represented as the distance from the wheel center to a given color point. Value is usually represented as a separate one-dimensional “luminance” parameter since the color wheel can encode only two distinct parameters.

Color coding on 2D surfaces. Vector color coding for 2D vector fields proceeds as follows. Assume we have a color wheel of unit radius and all vectors in the 2D dataset are scaled so that the longest one has unit length. Under these conditions, every vector is represented by the color it points to if we place it at the center of the color wheel. The vector orientation is encoded in the hue and the vector length is in the value. The saturation parameter is set to one, i.e., we use only fully saturated colors. The color-coding process is applied for every point of the dataset, similarly to scalar color coding, either via texture or polygon color interpolation.

Figure (a) shows the vector color coding for the same 2D vector dataset as was used in the previous section. This image does not suffer from the sampling problems discussed for glyph visualizations, which is a positive element. Low-vector-magnitude regions can be easily detected as dark (low-value) areas, whereas high-vector-magnitude regions show up as brightly colored areas. However, in contrast to the intuitive arrow plots, this visualization is highly abstract. The inverse mapping from hue to vector orientation takes quite some time to be learned, so users have to be trained extensively to interpret such images.

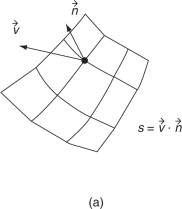
Several variations of the basic idea exist. If we are interested only in the vector orientation and not the magnitude, we can set the value component to one, and we obtain the visualization shown in Figure (b). Here, the orientation patterns of the vector field are easier to distinguish than in Figure (a), since the image is brighter. Besides the standard color wheel containing all rainbow hues, other color wheels can be used to emphasize certain orientations, similar to the various colormap manipulations for scalar fields.

Besides the directional color coding, we can also directly encode the vector components vx, vy, vz into colors. In this setting, a 3D vector field is visualized by three separate scalar color-mapped fields. Although this method is probably the simplest way, from a technical perspective, to produce a visualization of a vector field, it has limited effectiveness. The user must visually correlate the same locations in three color images to get insight into the vector data at that location. Even if the user were able to accurately identify the location of the same spatial point in three different images, mentally performing three separate color-to-scalar mappings independently is a very hard task. Furthermore, it is difficult to imagine the direction of a vector just by looking at three scalar fields representing its components. For a more involved discussion on the reasons to avoid this type of color coding, see Colin Ware’s book on information visualization [Ware 04]. All in all, this method is seldom used, except in cases when users are very familiar with the vector field structure and domain shape, e.g., due to low variability, and want to look for specific outlier-like details.

# Displacement Plot :

Vector displacement on the surface of an object can be visualized with displacement plots. A displacement plot shows the motion of an object in the [direction perpendicular](https://www.sciencedirect.com/topics/engineering/direction-perpendicular) to its surface. The object motion is caused by an applied vector field. In a typical application, the vector field is a displacement or strain field.

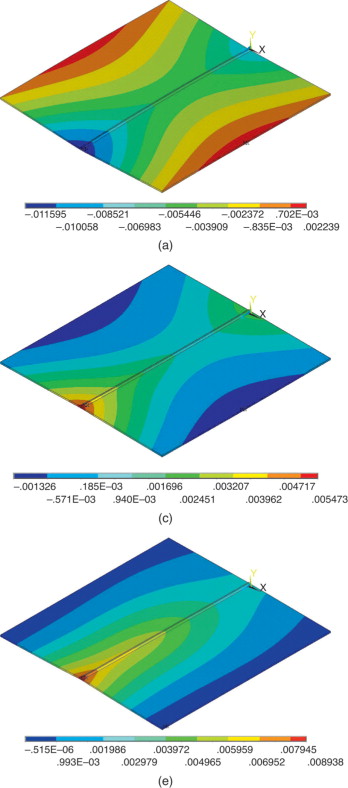
Vector displacement plots draw on the ideas in Section. Vectors are converted to scalars by computation of the dot product between the surface normal and vector at each point. If positive values result, the motion at the point is in the direction of the surface normal. [Negative values](https://www.sciencedirect.com/topics/computer-science/negative-value) indicate that the motion is opposite the surface normal



The displacement plot in Figure 4-3-1 shows zero displacement at the fixed end of the beam and a maximum displacement of 2.766e-3 m at the free end of the beam. This is within 1% of the results from exercise 4-1 (2.758e-3 m) and exercise 4-2 (2.739e-3 m), and is very similar to the theoretical value (2.736e-3 m). The plot of the [longitudinal stress](https://www.sciencedirect.com/topics/engineering/longitudinal-stress) in Figure 4-3-2 shows that the maximum stress occurs at the fixed end of the beam and has a value of 30.1 MPa. This is within 1% of the values obtained in exercise 4-1 (30 MPa), exercise 4-2 (30.3 MPa), and from beam theory (30 MPa). Finally, Figure 4-3-3 shows that the maximum equivalent stress is also 30.1 MPa. This is also within 1% of the values obtained in exercise 4-1 (30 MPa), exercise 4-2 (29.8 MPa), and from beam theory (30 MPa).



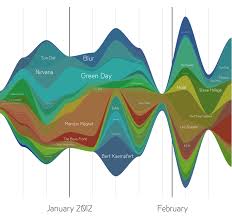
Figure 12.23 shows out-of-plane displacement plots for specimen A and 420 mm longitudinal heater spacing. Weld completion without TTT generates angular V-shaped distortion (greater at the finishing end) and a ‘hogging’ longitudinal bow. The lowest level of heater application (0.9 W/m2) flattens the plate considerably but reverses the [angular distortion](https://www.sciencedirect.com/topics/engineering/angular-distortion) to an inverted V and the longitudinal distortion to a ‘sagging’ mode (Fig. 12.23 (b)).

Increasing the heater input to 1.5 W/m2 (Fig. 12.23(c)) flattens the plate to a slightly greater extent in the same pattern, but further increase of heat input produces poorer results. Although the bowing distortion decreases, the angular distortion increases

# StreamObjects :

Visualization of data streams in the era of big data has an important role. The streams are currently being created everywhere - from personal logs, where people track their travels over the world or training routines, to large network and sensor infrastructures, from financial transactions to social media text streams and this will be an ongoing trend in the years to come. Efforts in databases and data management communities on how to efficiently transfer and store all that data have been joined in the last years by the efforts in the data mining community on how to deal with the automated analysis algorithms on such a large scale. What we are seeing now are the challenges that arise in the domain of data science and visual analytics - how to visualize, explore, and make sense of all these vast amounts of data.

Therefore, visualizations that can help humans to efficiently analyze the part of the current and the past data are of great importance. Visualization of streaming data is strongly related to its temporal context and very often methods that map time to the horizontal axis are used to visualize the data stream. How do we define which part of past data is relevant for the current data and the current point in time? Although the data being generated and delivered in the streams has a strong temporal component, in many cases it is not only the temporal component that the analysts are interested in. Other important data dimensions are equally important and time might be just an additional aspect that they care about. In those cases, we might want to rely on other visualization methods that can show other attributes better than temporal visualizations



# Texture-based vector visualization :

ext visualization technologies, as forms of computer-supported knowledge discovery, aim to understand and utilize the wealth of text-based information available to us.

 Text visualization is mainly achieved through the use of graph, chart, word cloud, map, network, timeline, etc. It is these visualized results that make it possible for humans to read the most important aspects of a huge amount of information.

In the coming of the digital age, study of data becomes more and more important when dealing with a huge amount of information. For example, there are more than 155.3 million items in the Library of Congress [1] which would take about 500 hundred years for one to read 10 items each day. Human brains are not good at studying such a huge amount of information while machines with great computational power are good at handling big data by doing repetitive works fast. According to Moore’s Law, “the number of transistors in integrated circuits doubles approximately every two years” (See Fig. 1.), which means that the computational power of machines grows faster and faster. It is a trend to study a huge amount of information by converting it to data and then analyzing it by use of the computational power of machines

## Example World Cloud:

Word Cloud may be the most beautiful, creative and powerful tool for text visualization. By making words stand out either by means of font size or color according to their usage frequency, Word Cloud has its significance in both text analysis and digital humanities scholarship. Text analysis result in the form of a word cloud can show the theme of texts obviously if the presumption that more important words appear more often is taken to be true. In order to show the power of word cloud, I made a word cloud of all my weekly blogs by using Wordle [5] as shown in Fig. 5. It is easy for the reader to know what I have written this semester without reading my blogs. The several most important words are “literature”, “project”, “media”, “texts” and “data”. It seems that I did not digress from the subject this semester. Besides being a tool for text visualization, a word cloud can also be used as an art piece due to its beautiful form. Tagxedo [5] is a tool to do word cloud artistically. In this tool, the user can design the shape of word cloud, the front of words and the contrast of words. I do a word cloud of my weekly blogs using Tagxedo and get a beautiful word cloud as shown in Fig. 6. This is an art piece with both aesthetic character and literary elements.



# Simplified representation of vectors :

Feature detection methods reduce the vector field to a set of features of interest – Feature type, position, extent, and strength

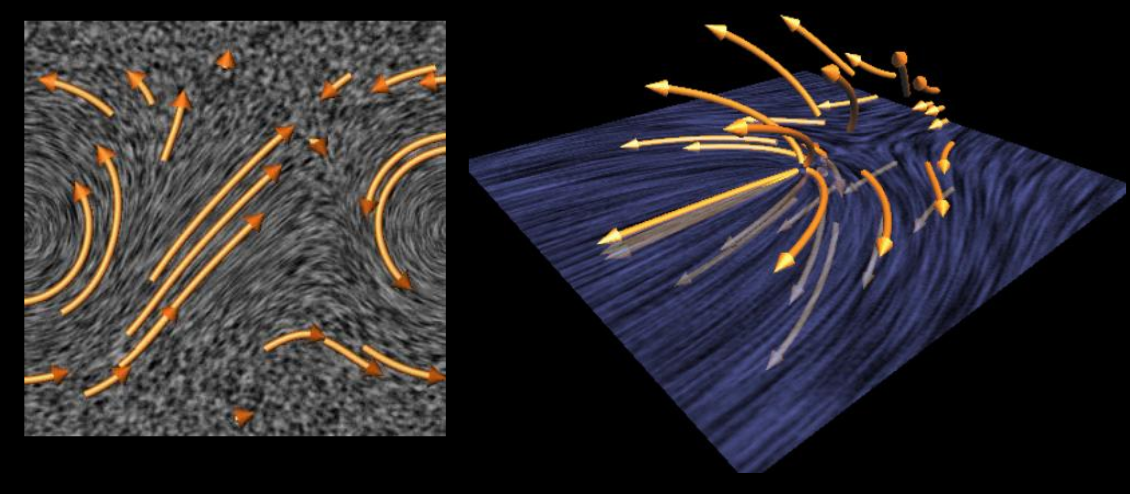
• Feature defined – Analytically by a feature energy-like function – A set of examples or patterns

• Some problems – Hard to define precise numerical criteria to detect such features – Appear at different spatial scales in vector fields – No clear spatial separation between a feature and a nonfeature area Field Decomposition Methods

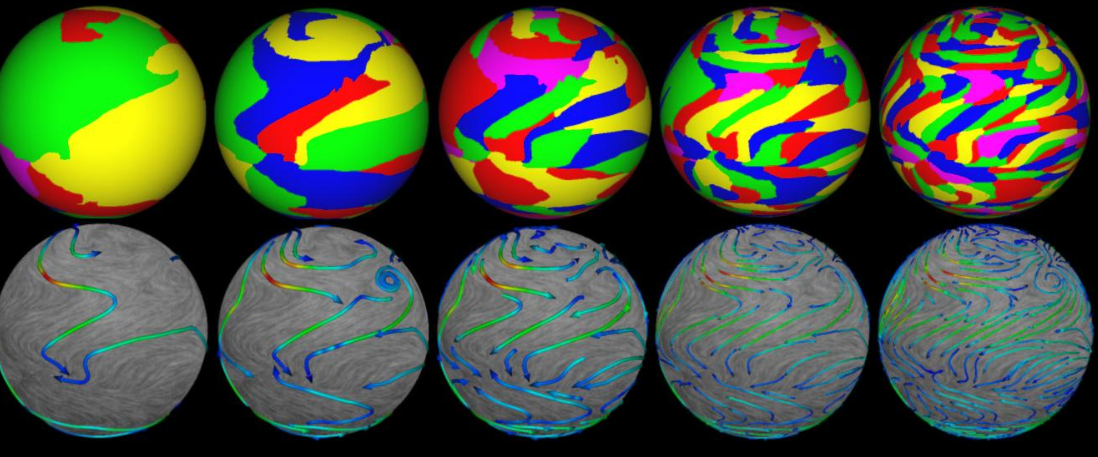
• Partition the vector dataset into – regions of strong intra-region (internal) – weak inter-region similarity

• Core: similarity metric f that defines how similar two regions are – Different metric -> different decomposition

• A frequently used: compares the direction and magnitude of the vector data • Usually perform a top-down partitioning or bottomup agglomerative clustering of dataset



AMG: the Algebraic MultiGrid — The idea of producing a hierarchy of bases that approximates a given vector field at several levels of detail



Climate dataset decomposition, five coarsest levels Domains, and flow texture overlaid with curved arrow icons

# Conclusion :

A number of visualization methods for vector fields – From simple visual representations, straightforward implementation: vector glyph – To multiscale textures animated in real-time, complex implementations: LIC: Line Integrated Convolution

• Another classification method: based on the dimensionality of the data domain – 2D surface: planar or curved ones – 3D volumetric vector fields, more challenging

• Inherent occlusion of the visualization primitives

• The challenge of creating insightful visualization of 3D time-dependent vector fields describing complex phenomena is still an active area of research

**Tensor Visualization**

# Introduction

Tensor data encode spatial properties that vary based on position and direction, such as the curvature of a three-dimensional surface at a specific point and direction. Most visualization applications handle rank 2 symmetric tensors, so this chapter focuses on visualizing such tensors. These tensors are represented as 3 × 3 matrices in a global coordinate system, with each point in a tensor dataset carrying one of these matrices as its attribute data.

Tensor datasets are prevalent in various application domains. For instance, properties of 3D surfaces, such as curvature, are described by curvature tensors. In mechanical engineering, material properties like stress and strain in 3D volumes are represented by stress tensors.

Water diffusion in tissues can occur isotropically, with equal speed in all directions, or anisotropically, with varying speeds in different directions. In human brain tissue, for example, diffusion is more pronounced along neural fibers and less so across them. These fibers, consisting of axon bundles, are known as white matter due to the myelin layer's characteristic color. At any given point in the tissue volume, diffusion is described by a 3 × 3 diffusion tensor matrix. Since diffusion is stronger along the fibers, measuring the diffusion tensor and visualizing the direction of strongest diffusion can provide insights into the complex structure of neural fibers in the human brain.

The measurement of water diffusion in living tissues is performed using techniques collectively known as diffusion tensor magnetic resonance imaging (DT-MRI). The overall process of creating visualizations of anatomical structures from the measured diffusion data is called diffusion tensor imaging (DTI). This is a prominent research area in scientific visualization and medical imaging. DTI techniques are employed in diagnosing and analyzing various brain diseases and studying the relationship between brain structure and function (functional brain anatomy).

As we discussed in the previous chapter, visualizing 3D vector fields is challenging because we need to map three independent values to a graphical representation for each data point. Visualizing a tensor field is even more complex, as we must represent a complete 3 × 3 matrix for every data point. One approach might be to visualize the matrix entries as separate scalar fields using methods like isosurfaces or color-coded slice planes. However, this method does not effectively convey how the tensor data varies with direction. Fortunately, tensor data has an intrinsic structure that can be leveraged to create more insightful visualizations. This structure is determined using a technique called principal component analysis, which we will explain in the next section.

The structure of this chapter is as follows. We begin with a brief overview of principal component analysis in Section. This technique processes tensor matrices to extract information that is crucial for visualization and is fundamental to many tensor data processing and visualization algorithms. We then demonstrate how the results of principal component analysis can be visualized using simple color-mapping techniques. Next, in Section , we show how the same data can be visualized using tensor glyphs, building on the vector visualization techniques introduced earlier. Sections and elaborate on the similarities between tensors and vectors, introducing streamline-like visualization techniques for tensor fields. Finally, Section concludes this chapter.

# Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique used to simplify the complexity in high-dimensional data by transforming it into a lower-dimensional form. PCA identifies the directions (principal components) in which the data varies the most. This method is essential in tensor visualization as it helps reduce the dimensionality of the data while preserving as much variance as possible.

Let us consider again, for illustration purposes, the curvature tensor of a 3D surface, which we introduced in Section. Consider a local coordinate system xyz centered at a point x0 on our surface, where the z-axis coincides with n, the surface normal at x0, andthex-andy-axes are tangent to the surface at x0 . Close to x0, our surface can be locally described as some function z = f(x,y), with f(x0) = 0. We have shown that we can compute the normal curvature at some point x0 in some direction s in the tangent plane as the second derivative ∂2f/∂s2 of f using the two-by-two Hessian matrix of partial derivatives of f . For smooth surfaces, the normal curvature varies smoothly as the direction vector s rotates around the current

point in the tangent plane. In many applications, we are actually interested only in the extremal (minimal and maximal) values of the curvature as a function of the direction s. Since these directions depend only on the surface shape at a given point, they are invariant to the choice of the local coordinate system. How can we compute these directions? Since s is a unit direction vector, we can write it as sT =(cosα,sinα), where α is the angle between s and the x axis of our local coordinate system, and the superscript T denotes a transposed column vector.

Example: Curvature Tensor on a 3D Surface

Figure illustrates the principal directions of the curvature tensor for a 3D surface:

* The surface is shown in green.
* The direction of minimal curvature is 𝑠1s1 (shown in yellow).
* The direction of maximal curvature is 𝑠2s2 (shown in red).
* Other directions in the tangent plane, orthogonal to the surface normal 𝑛n at the considered point, are shown in black. The curvature along these directions varies between the minimal and maximal values.

Principal Directions and Eigenvalues

The solutions 𝑠𝑖si of Equation are the principal directions or eigenvectors of the tensor 𝐻H. The values 𝜆𝑖λi, called eigenvalues, are the extremal values of the quantity encoded in the tensor (such as curvature in this example) in the corresponding principal directions.

Principal Component Analysis

Computing the eigenvalues and eigenvectors of a tensor using this technique is known as principal component analysis (PCA). For an 𝑛×𝑛n×n symmetric matrix, the principal directions (eigenvectors) are perpendicular to each other. These directions form a local coordinate system in which the tensor-encoded quantity reaches its extremal values.

Detailed Steps

1. Determinant Calculation:

* Given a matrix 𝐻H, subtract 𝜆𝐼λI from 𝐻H, resulting in 𝐻−𝜆𝐼H−λI.
* Compute the determinant of 𝐻−𝜆𝐼H−λI:

det(𝐻−𝜆𝐼)=(ℎ11−𝜆)(ℎ22−𝜆)−ℎ12ℎ21=0.det(H−λI)=(h11−λ)(h22−λ)−h12h21=0.

* This is a quadratic equation in 𝜆λ.

2. Solving for Eigenvalues:

• Solve the quadratic equation to find the eigenvalues 𝜆1λ1 and 𝜆2λ2.

3. Finding Eigenvectors:

• Substitute each eigenvalue 𝜆𝑖λi back into the equation (𝐻−𝜆𝐼)𝑠=0(H−λI)s=0 to find the corresponding eigenvector 𝑠𝑖si.

4. Principal Directions and Extremal Values:

• The eigenvectors 𝑠𝑖si are the principal directions, and the eigenvalues 𝜆𝑖λi represent the extremal values of the tensor-encoded quantity in these directions.

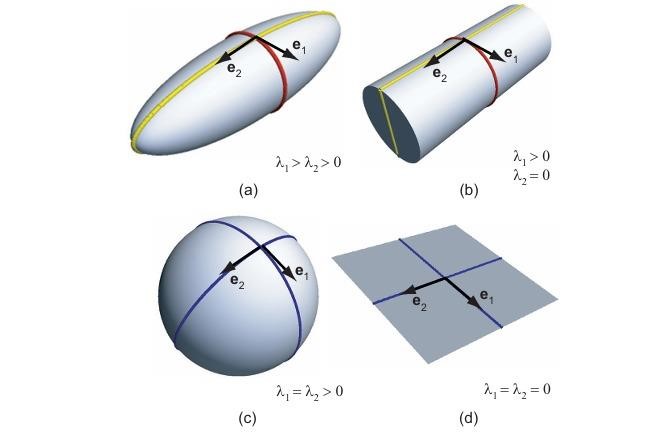
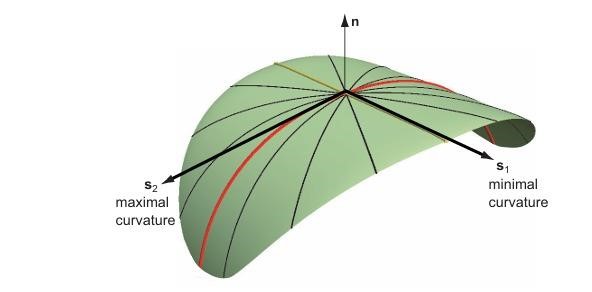
5. Orthogonality and Local Coordinate System:

* For a symmetric matrix, the principal directions are orthogonal, forming a local coordinate system. This system aligns with the directions in which the tensor quantity (e.g., curvature) reaches its extremal values.

Visualization Example

* Surface Visualization: The green surface represents the 3D object being analyzed.
* Principal Directions:
* 𝑠1s1 (yellow) indicates the direction of minimal curvature.
* 𝑠2s2 (red) indicates the direction of maximal curvature.
* Tangent Plane Directions: Black vectors in the tangent plane show directions with curvatures between the minimum and maximum values.
* Eigenvalues and Eigenvectors: The eigenvalues provide the magnitude of curvature in the principal directions, and the eigenvectors provide the orientation of these directions.

By using PCA to compute the eigenvalues and eigenvectors, we can analyze the principal directions and magnitudes of variation in tensor-encoded data, such as curvature on a 3D surface. This method reveals the fundamental geometric properties of the data.



# Visualizing Components

Visualizing the principal components involves plotting these components to understand the underlying structure of the data. This can be done in various ways, such as scatter plots or bar charts, which depict the variance explained by each principal component. Visualizing components aids in interpreting the significance of each component in the data set.

The simplest way to visualize a tensor dataset is by treating it as a collection of scalar datasets. For example, consider a 3 × 3 tensor matrix 𝐻. Each element ℎ𝑖𝑗 of this matrix can be viewed as an individual scalar field.

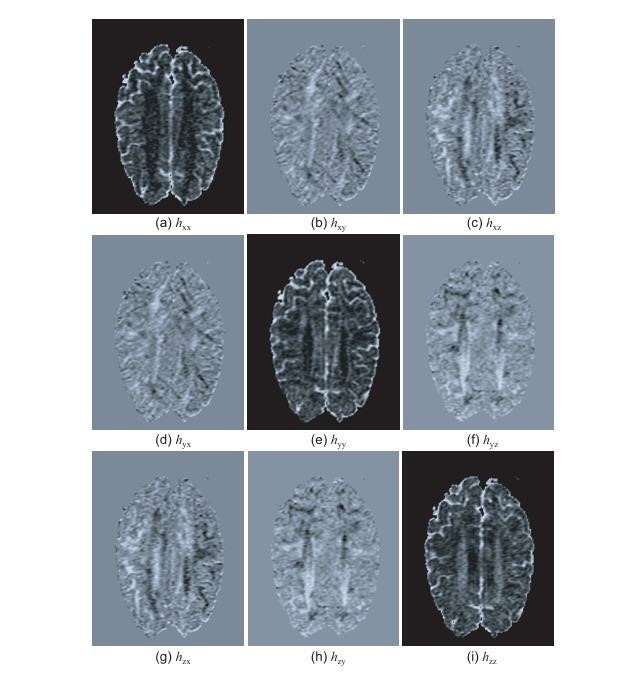
In Figure, we see these components for a single 2D slice from a brain diffusion tensor volumetric dataset. Each component of the tensor matrix is visualized using a grayscale colormap that maps scalar values to luminance. For clarity and ease of interpretation, the areas outside the actual tissue, which correspond to air and contain noisy tensor values, are set to a neutral uniform gray background.

Additionally, due to the symmetry of the tensor matrix, we only need to visualize six unique components instead of nine. This is because the tensor matrix is symmetric, meaning that ℎ𝑖𝑗=ℎ𝑗 . As a result, the following pairs are equivalent and need not be displayed twice: ℎ12 and ℎ21, ℎ13nd ℎ31, and ℎ32.Thus, the six unique images provide a complete representation of the tensor matrix's scalar fields.

# Visualizing Scalar PCA Information

When visualizing scalar PCA information, each data point is reduced to a single value representing its projection onto a principal component. This visualization can be represented through histograms, box plots, or color-coded scalar fields in images or volumes. This method helps in identifying clusters and trends within the data.

A more intuitive method for visualizing tensor data involves focusing on derived data that holds more physical significance. For instance, the mean diffusivity (MD) of brain diffusion tensor data can be visualized across sagittal, axial, and coronal slices.



Brain Dataset and Visualization

The brain dataset used here is courtesy of Gordon Kindlmann from the Scientific Computing and Imaging Institute at the University of Utah, and Andrew Alexander from the W. M. Keck Laboratory for Functional Brain Imaging and Behavior at the University of WisconsinMadison.

Mean Diffusivity Visualization

Mean diffusivity represents the average diffusion rate of water molecules across all directions at a given point. Figure shows this scalar quantity using a grayscale colormap over three orthogonal slice planes. The small image in the lower-right corner displays the brain surface along with the three slices for orientation.

Using Orthogonal Slices

Utilizing sagittal, axial, and coronal slices is a common technique in medical imaging to quickly gain insight into a volumetric dataset. These slices correspond to the different anatomical planes:

* Sagittal Plane: Divides the body into left and right parts.
* Axial Plane: Divides the body into top and bottom parts.
* Coronal Plane: Divides the body into front and back parts.

Visualizing Anisotropy with PCA

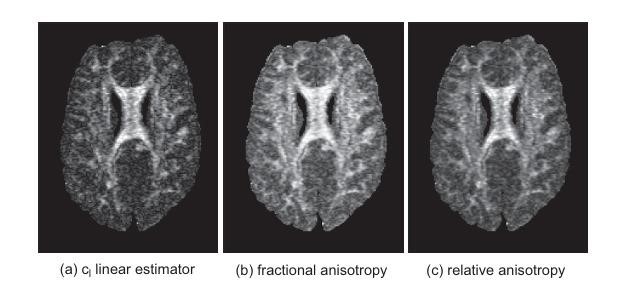
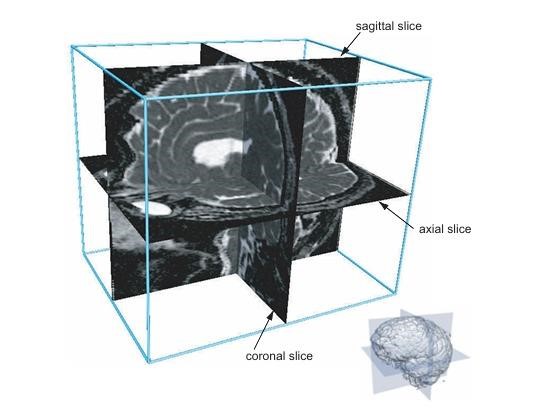
Further insights can be obtained by visualizing the results of principal component analysis (PCA). The eigenvectors of a tensor indicate the directions of maximum and minimum variation in the data at a given point, while the corresponding eigenvalues quantify these variations. In diffusion tensor imaging (DTI), eigenvalues describe the degree of anisotropy in the tissue.

* Isotropy: In isotropic tissues, diffusion is uniform in all directions.
* Anisotropy: In anisotropic tissues, diffusion varies in different directions. This property is useful for distinguishing neural fibers, which are highly anisotropic, from other tissues.

Anisotropy Measures

Several techniques exist to estimate anisotropy from diffusion tensor data using PCA. One well-known set of metrics was proposed by Westin et al. (1997), which calculates the certainties 𝑐𝑙, cp, and 𝑐𝑠cs for a tensor having linear, planar, or spherical shapes, respectively.

These metrics provide a quantitative way to describe the shape of the diffusion tensor and hence the anisotropy of the tissue at each point in the dataset



# Visualizing Vector PCA Information

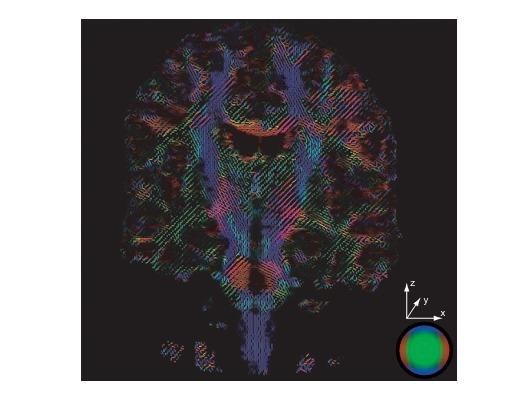
Vector PCA visualization involves representing each data point as a vector in the reduceddimensional space defined by the principal components. Techniques like quiver plots or arrow plots can be used to visualize these vectors. This approach is beneficial for understanding the directional trends and relationships within the data.

Visualizing Anisotropy Directions

Previously, we discussed visualizing various anisotropy metrics from PCA analysis of a tensor field using standard scalar visualization methods, such as color mapping. However, often we are also interested in the directions of anisotropy, not just its magnitude.

Major Eigenvector Visualization

To visualize the direction of maximal variation in the tensor-encoded data, we can use vector visualization methods. Specifically, we can visualize the major eigenvector field, which represents the direction of the highest variance.



Example: Hedgehog Plot

Figure demonstrates this concept with a hedgehog plot of the major eigenvector over a coronal slice of the same DT-MRI dataset used in Figure . In this plot:

* Vectors are uniformly placed at points where diffusion measurements are above a certain accuracy threshold.
* The hue of each vector indicates its direction.

This method allows us to see both the direction and magnitude of anisotropy across the dataset, providing a comprehensive view of the tensor field's directional properties.

In the hedgehog plot depicted in Figure , eigenvectors are color-coded based on their alignment with the coordinate axes of the dataset. The color mapping is as follows:

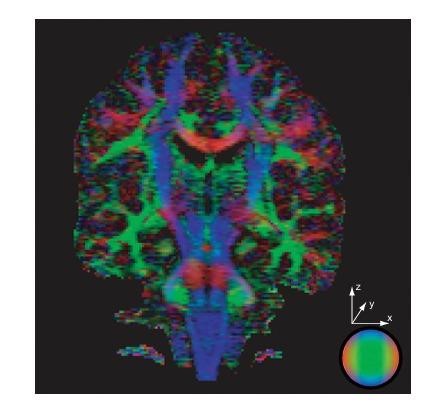
Red: Vectors aligned with the x-axis.

Green: Vectors aligned with the y-axis.

Blue: Vectors aligned with the z-axis.

Color Mapping Icon

The icon in the bottom-right corner of Figure illustrates this color mapping. This icon is a shaded sphere where the color at each point represents the direction of the radial vector at that point. Essentially, it serves as a legend for interpreting the direction of the eigenvectors based on their color.



Luminance and Measurement Confidence

The brightness (luminance) of the vectors indicates the confidence level of the measurements:

Bright Vectors: Represent areas with high confidence in the diffusion measurements.

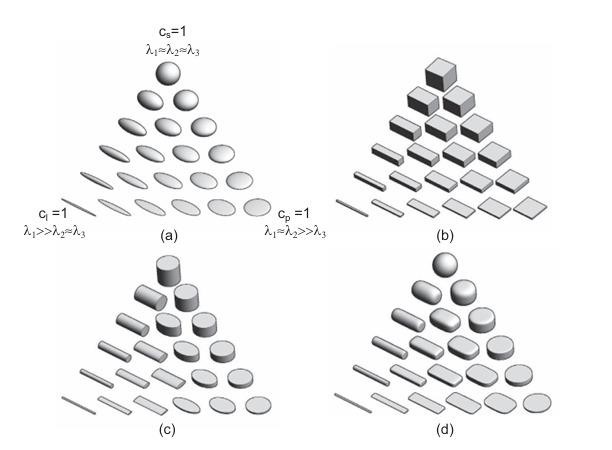
Dark Vectors: Represent areas with low confidence or noisy measurements.

By using this color and luminance scheme, the hedgehog plot provides a comprehensive visualization of both the direction and reliability of the tensor data, making it easier to interpret the anisotropic properties of the tissue.

# Tensor glyphs

Tensor glyphs are graphical representations of tensors that provide a visual summary of tensor properties such as orientation and magnitude. Common tensor glyphs include ellipsoids, boxes, or super quadric shapes, which can be used to represent the tensor’s eigenvalues and eigenvectors. Tensor glyphs are particularly useful in fields like medical imaging, where they can represent diffusion tensor imaging (DTI) data.

Tensor glyphs provide a generalized method for visualizing tensor data by extending the concept of vector glyphs. These glyphs visually encode the eigenvalues and eigenvectors of tensors at representative sample points within the dataset.



Construction of Tensor Glyphs

1. × 2 Tensor Data: For each sample point in a 2 × 2 tensor dataset, we encode two eigenvalues and two eigenvectors by constructing a 2D ellipse. The ellipse's half-axes are oriented in the directions of the eigenvectors and scaled by the absolute values of the eigenvalues.
2. × 3 Tensor Data: For a 3 × 3 tensor dataset, we create a 3D ellipsoid. This ellipsoid encodes three eigenvalues and three eigenvectors in a similar manner to the 2D case.

Visualization Algorithm

Principal Component Analysis (PCA): Perform PCA at each sample point to obtain the eigenvalues and eigenvectors.

Ellipsoid Construction: Scale the ellipsoid glyph according to the eigenvalues.

Rotation: Rotate the ellipsoid using a matrix with the eigenvectors as columns.

Translation: Position the ellipsoid at the corresponding sample point.

Repetition: Repeat the process for all sample points where tensor ellipsoids need to be drawn. Example: Glyph Shapes

Figure (a) illustrates the possible shapes of ellipsoid glyphs:

* Triangle Corners: Show extreme cases where one of the linear (𝑐𝑙), planar (𝑐𝑝), or spherical (𝑐𝑠) certainties is at its maximum value of one, while the other two are zero. These correspond to line, disc, and sphere glyph shapes, respectively.
* In-between Shapes: Represent different combinations of certainty values. The

triangular "glyph space" can be parameterized by 𝑐𝑙, 𝑐𝑝, and 𝑐𝑠cs, with the constraint that cl+cp+cs=1. The different shapes reflect the varying certainty values and corresponding eigenvalue ratios.

Alternative Glyph Shapes

While ellipsoids are commonly used, other shapes can also encode tensor information:

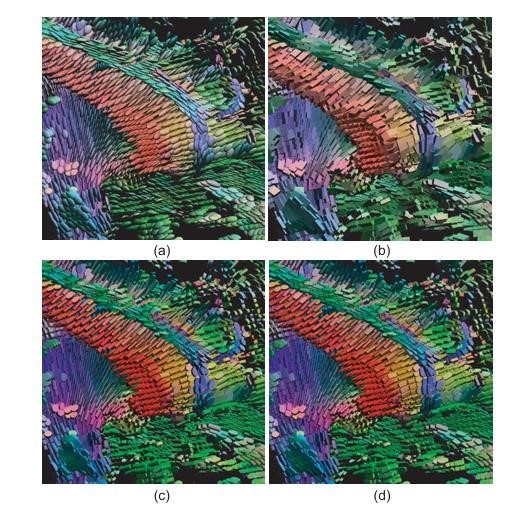
* Parallelepipeds (Cuboids): These can be used to represent tensors, offering a different visual clarity and information expression.
* Cylinders: Another alternative shape.

Figures (b) and (c) show the forms these alternative glyphs can take, depending on different certainty values or eigenvalue ratios.

Detailed Explanation

* Sampling the Dataset: The dataset domain is sampled at various points to represent the tensor field effectively.
* Eigenvalues and Eigenvectors: At each sample point, the eigenvalues indicate the magnitude of variations, and the eigenvectors show the directions of these variations.
* Glyph Construction:
* For a 2D tensor, construct an ellipse where the axes align with the eigenvectors, scaled by the eigenvalues.
* For a 3D tensor, construct an ellipsoid similarly.
* Shape Interpretation:
* Line glyphs represent linear certainty (𝑐𝑙).
* Disc glyphs represent planar certainty (𝑐𝑝).
* Sphere glyphs represent spherical certainty (𝑐𝑠).
* Glyph Space: The triangular parameter space, constrained by cl+cp+cs=1, helps visualize different shapes reflecting various certainty values.

By using tensor glyphs, we can effectively visualize complex tensor fields, capturing both the magnitudes and directions of variations within the data.



Various Tensor Glyph Shapes

1. Ellipsoids:

* Ellipsoids represent the tensor by scaling and orienting according to its eigenvalues and eigenvectors.
* They provide a smooth and less distracting visualization.
* However, their 2D projection can sometimes obscure the 3D orientation.

2. Cuboids:

* Cuboids clearly indicate the directions of the eigenvectors with their edges and faces.
* They can fail to convey directional ambiguities when eigenvalues are equal, making it hard to interpret some aspects of the data.

3. Cylinders:

* Cylinders highlight the major eigenvector direction through their axis.
* They can rotate abruptly by 90 degrees with small changes in eigenvalues, leading to confusing discontinuities in the visualization.

4. Superquadrics:

* Superquadrics offer a solution to the ambiguity problems of other glyphs.
* They are parameterized using the planar and linear certainty metrics 𝑐𝑙cl and 𝑐𝑝cp.

Visualization of DT-MRI Diffusion Tensor Dataset

Figure showcases the use of these glyph shapes in a DT-MRI diffusion tensor dataset. The focus is on the corpus callosum structure, visualized with red glyphs. The glyphs are colored by direction, similar to the hedgehog plot from Figure , and their color saturation varies with the fractional anisotropy (FA). Saturated glyphs indicate regions of high anisotropy, while gray glyphs show low-anisotropy areas.

In contrast to hedgehog plots where vectors are seeded over a 2D slice, tensor glyphs are seeded over a 3D region, providing a more comprehensive visualization. Smooth glyph shapes like ellipsoids present a cleaner picture compared to cuboids and cylinders, which have sharp edges.

Advantages and Disadvantages of Different Glyph Shapes

* Cuboids:
* Good at showing eigenvector directions with their faces and edges.
* Less effective in displaying directional ambiguities when eigenvalues are equal.
* Cylinders:
* Clearly depict the major eigenvector direction.
* Can cause confusing visual discontinuities due to abrupt rotations when eigenvalues change slightly.
* Ellipsoids:
* Avoid the issues of cuboids and cylinders.
* Can be ambiguous in 2D projections regarding their 3D orientation.

Super quadric Glyphs

To overcome the limitations of ellipsoids, super quadric glyphs were introduced by Kindlmann. These glyphs are defined by super quadric shapes parameterized by the planar and linear certainty metrics 𝑐𝑙and 𝑐𝑝.

Super quadric shapes can be represented by the implicit function 𝑞(𝑥,𝑦,𝑧)=0q(x,y,z)=0:

𝑞(𝑥,𝑦,𝑧)={(𝑦2𝛼+𝑧2𝛼)𝛼/𝛽+𝑥2𝛽−1=0,if𝑐𝑙≥𝑐𝑝,where

𝛼=(1−𝑐𝑝)𝛾,𝛽=(1−𝑐𝑙)𝛾,(𝑥2𝛼+𝑦2𝛼)𝛼/𝛽+𝑧2𝛽−1=0,if𝑐𝑙<𝑐𝑝,where

𝛼=(1−𝑐𝑙)𝛾,𝛽=(1−𝑐𝑝)𝛾.q(x,y,z)=⎩⎨⎧(αy2+αz2)α/β+βx2−1=0,(αx2+αy2)α/β+βz2−1=0,if cl≥cp, where α=(1−cp)γ,β=(1−cl)γ,if cl<cp, where α=(1−cl)γ,β=(1−cp)γ.

This parameterization helps in accurately visualizing the anisotropy of the tensor data.

Tensor glyphs are powerful tools for visualizing the directional properties of tensor data. Each glyph shape—ellipsoids, cuboids, cylinders, and super quadrics—has its own strengths and weaknesses. Super quadric glyphs, in particular, offer a robust solution to common visualization challenges by effectively encoding the tensor's eigenvalues and eigenvectors.

# Fiber Tracking

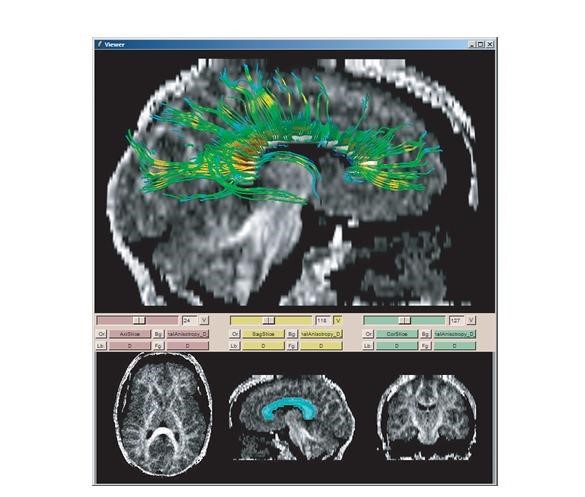
Fiber tracking, or tractography, is a technique used to visualize the pathways of fiber bundles within a tensor field. This method is widely used in medical imaging to map neural tracts in the brain. By tracking the direction of the principal eigenvector of the tensor, fiber tracking algorithms can reconstruct the pathways of these fibers, providing insights into the structural connectivity of the brain.

Using tensor glyphs for visualizing tensor fields is similar to using vector glyphs for visualizing vector fields. Therefore, it is natural to consider whether other vector field visualization techniques can be adapted for tensor data

Streamlines are among the most effective and popular techniques for visualizing vector fields. The question arises whether (and how) we can use streamlines to gain insights into a tensor field. Consider, for example, a DT-MRI tensor dataset. As mentioned earlier, regions of high anisotropy, especially those with high values of the linear certainty metric cl, correspond to neural fibers aligned with the major eigenvector el . To visualize the location and direction of these fibers, it makes sense to track the direction of this eigenvector in regions of high anisotropy. We can accomplish this using the streamline technique introduced for vector fields.

A typical fiber tracking method proceeds as follows. First, a seed region is identified, which is an area where the fibers intersect. This can be detected by thresholding one of the anisotropy metrics. Next, streamlines are densely seeded in this region and traced (integrated) both forward and backward in the major eigenvector field el until a desired stopping criterion is reached. The stopping criterion is usually a combination of various conditions that describe the desired features of the resulting visualization. These conditions may include a minimum value of the anisotropy metric (beyond which the fiber structure becomes less apparent), a maximum fiber length (similar to vector streamlines), exiting or entering a predefined region of interest specified by the user (which may describe a previously segmented anatomical structure), and a maximum distance from other tracked fibers (beyond which the current fiber "strays" from a potential bundle structure targeted for visualization).

After the fibers are tracked, they can be visualized using the stream tubes technique to further highlight their geometry. Similar to vector visualization, these constructed tubes can be colored to represent the value of a relevant scalar field, such as the major eigenvalue, anisotropy metric, or another quantity measured along with the tensor data.

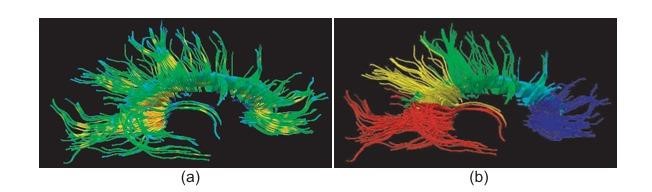


The process of tracking fibers in DT-MRI datasets is quite delicate and often requires significant user intervention, particularly during the step of defining the seed region. To assist users in this process, various integrated tools have been developed. These tools enable interactive visualization of scalar quantities on slices in the tensor dataset, computation of anisotropy metrics, and definition of regions of interest for seeding the fiber tracking process. Figure shows a snapshot from one such tool, called Slicer. In the lower part of the snapshot, three axial, sagittal, and coronal slices display the fractional anisotropy metric, similar to Figure. The middle slice shows a region of interest selected by the user based on high anisotropy values, highlighted in light blue. This region corresponds to a sagittal cross-section.

The top image in Figure shows the sagittal slice along with fibers tracked from seed points densely distributed in the region of interest. These fibers are colored using a blue-to-red rainbow colormap based on the fractional anisotropy metric and visualized with stream tubes. The fibers terminate when the linear certainty metric 𝑐𝑙 falls below a value of 0.15.

By removing the slice plane from the fiber visualization, we can analyze the resulting fiber structure in more detail. This reveals the symmetric fanning out of fibers that emerge from the corpus callosum and "radiate" into the two hemispheres of the brain. Additionally, there are several fibers oriented nearly horizontally, corresponding to the structure of the corpus callosum itself.

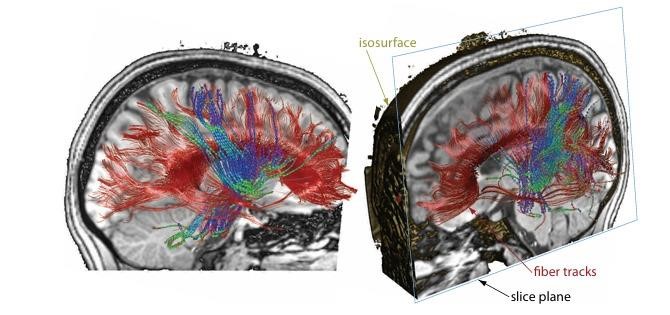
Focus and Context: Fiber tracks are most useful when shown in the context of the brain's anatomy. Figure illustrates this concept. The two images display different views of a set of fiber tracks from a DT-MRI scan with dimensions of 81 × 106 × 76 voxels. The fibers are densely seeded in the corpus callosum region. They are colored red and shaded using the Phong lighting model to enhance the understanding of their curved positions in space. To better comprehend the spatial embedding of the fibers, three elements are added: (a) a slice plane showing tissue density with grayscale color mapping, (b) an iso surface of the same scalar value colored yellow, and (c) ellipsoid tensor glyphs at all fiber points where the anisotropy exceeds a userdefined minimum.



Threshold Visualization and Directional Colormap

In our visualization approach, we use a directional colormap to color glyphs based on the direction of the major eigenvector. This is similar to the methods shown in Figures. To minimize occlusion and keep context, the isosurface is displayed only for the dataset area behind the slice plane. This technique keeps the focus on tensor glyphs, which represent the highest anisotropy fiber points, within the context of shaded fiber tracks and brain anatomy.

More details on this focus-and-context technique can be found in [Peeters et al. 09].



Fiber Clustering

While useful, the initial visualization shows many disjoint fibers. In reality, fibers are grouped into bundles with quasi-parallel structures. To mimic this anatomical feature, we can construct a visualization that groups fibers into bundles, as described in [O’Donnel and Westin 05].

First, we define the directional similarity between two fibers. Given two fibers, 𝑎a and 𝑏b, represented as 3D parametric curves 𝑎=𝑎(𝑡)a=a(t) and 𝑏=𝑏(𝑡)b=b(t) with 𝑡∈[0,1]t∈[0,1], we

calculate the distance 𝑑(𝑎,𝑏)d(a,b) as:

𝑑(𝑎,𝑏)=12𝑁∑𝑖=1𝑁(𝑑(𝑎(𝑖/𝑁),𝑏)+𝑑(𝑏(𝑖/𝑁),𝑎))d(a,b)=2N1∑i=1N(d(a(i/N),b)+d(b(i/N),a)) Here, 𝑑(𝑝(𝑡),𝑞)d(p(t),q) denotes the smallest distance between a point 𝑝(𝑡)p(t) on fiber 𝑎a and all points on fiber 𝑏b: 𝑑(𝑝(𝑡),𝑞)=min⁡𝜏∈[0,1]∥𝑝(𝑡)−𝑞(𝜏)∥d(p(t),q)=minτ∈[0,1]∥p(t)−q(τ)∥ Fibers that are parallel and close to each other will have a low distance value. The similarity between fibers is defined as the inverse of this distance.

Using this distance measure, fibers are clustered in order of increasing distance, from the most to the least similar, until the desired number of clusters is reached. This can be done using the hierarchical agglomerative clustering technique introduced in Section for vector fields. Figure (b) shows this clustering technique applied to the same set of tracked fibers as in Figure (a), with five user-selected clusters shown in different colors. This visualization reveals several structures corresponding to fibers emerging from distinct regions of the corpus callosum.

Tracking Challenges

Fiber tracking in DT-MRI datasets poses specific challenges. Current DT-MRI scanning technology often introduces noise and misses fine-scale details due to its sampling frequency. Unlike vector fields from numerical simulations, DT-MRI data is subject to acquisition noise. Moreover, tensor data is derived through several preprocessing steps, including PCA, each of which can introduce inaccuracies.

For instance, PCA estimation of eigenvectors may fail if tensor matrices are not nearly symmetric. Even when PCA is successful, robust fiber tracking requires a significant distinction between the largest eigenvalue and the other two. This distinction is often missing in regions where fiber bundles intersect.

Advances and Tools in Fiber Tracking

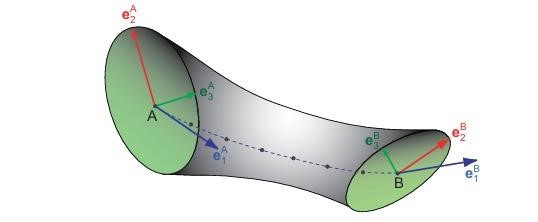
Fiber tracking in DT-MRI datasets is an active research area. Innovations include better definition of seed regions, robust criteria for stopping streamlines, and advanced rendering techniques like volume rendering with data-driven opacity transfer functions. These techniques enhance the visualization of complex structures resulting from the tracking process.

In terms of toolkits, the Diffusion Toolkit [Wang et al. 13] provides extensive utilities for fiber tracing and visualization in DT-MRI datasets. Unlike Slicer [Slicer 13], which is a general framework for 3D slice-based data volumes, the Diffusion Toolkit is specifically designed for DT-MRI datasets, offering more comprehensive and user-friendly options for fiber tracking.

# Hyper streamlines

Hyper streamlines extend the concept of streamlines used in vector field visualization to tensor fields. A hyper streamline follows the path of a principal eigenvector through a tensor field, with the shape and size of the streamline cross-section representing the tensor's other eigenvalues and eigenvectors. Hyper streamlines provide a comprehensive way to visualize the local diffusion characteristics in tensor data.

In the previous section, we explored how fiber tracking can be used to visualize tensor data. Fiber tracking involves integrating streamlines along the major eigenvector component of the tensor field, using various stop criteria determined by other derived quantities from the tensor data, such as the anisotropy measure in DT-MRI datasets. However, fibers do not capture directional information from the tensor field beyond the major eigenvector. This additional directional information is important as it provides insights into how tensor anisotropy varies in space. While tensor glyphs can visualize this information, they lack the spatial continuity of streamlines. This raises the question of whether we can enhance the streamline metaphor to incorporate this additional information, thus combining the advantages of streamlines and tensor glyphs.

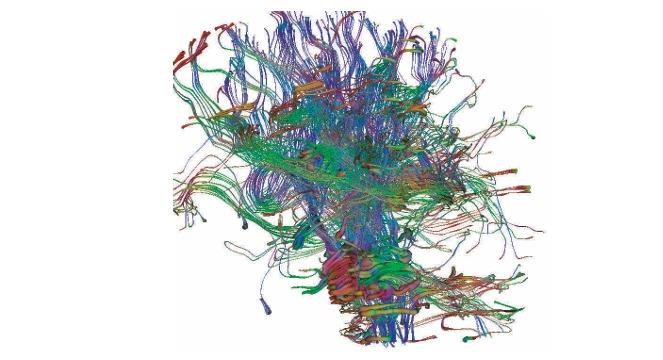


Hyper streamlines provide a solution to this problem. The principle behind hyper streamlines is straightforward. First, we perform principal component analysis as explained in Section 7.1 to decompose the tensor field into three eigenvector fields (ei ) and three corresponding scalar eigenvalue fields (𝜆1≥𝜆2≥𝜆3λ ). Next, we construct stream tubes in the major eigenvector field (𝑒1 ), similar to the fiber tracking method described earlier. At each point along a stream tube, we then visualize the medium and minor eigenvectors (𝑒2 and 𝑒3 ). Instead of using a circular cross-section of constant size and shape as in fiber tracking, we use an elliptic cross-section. The axes of this ellipse are oriented along the directions of the medium and minor eigenvectors (e 2and e 3), and are scaled by λ 2and 𝜆3, respectively.

Hyper streamlines convey the absolute values of the tensor eigenvalues, while the ellipse shape indicates their relative values and the orientation of the eigenvector frame along a streamline. Circular cross-sections signify that the medium and minor eigenvalues are equal. To show the value of the major eigenvalue, it can be encoded as color.

Figure illustrates the use of hyper streamlines in a diffusion tensor imaging (DT-MRI) brain dataset. Several hyper streamlines are seeded at various locations in the dataset, following the techniques described for fiber tracking. However, instead of tracing stream tubes with circular cross-sections, we use hyper streamlines. Color is used to indicate the local direction of the

Hyer streamlines, following the method discussed. Tracing of the hyper streamlines stops when the local anisotropy falls below a certain threshold. This is evident in the funnel-like endings of some hyper streamlines. At these points, the eigenvalues corresponding to the medium and minor eigenvectors are relatively large, indicating low anisotropy and a less pronounced fiber structure.



Several variations of this construction are possible. Any of the three eigenvectors can be used to determine the hyper streamline direction. Additionally, other shapes can be used for the cross

In the previous section, we explored how fiber tracking can be used to visualize tensor data. Fiber tracking involves integrating streamlines along the major eigenvector component of the tensor field, using various stop criteria determined by other derived quantities from the tensor data, such as the anisotropy measure in DT-MRI datasets. However, fibers do not capture directional information from the tensor field beyond the major eigenvector. This additional directional information is important as it provides insights into how tensor anisotropy varies in space. While tensor glyphs can visualize this information, they lack the spatial continuity of streamlines. This raises the question of whether we can enhance the streamline metaphor to incorporate this additional information, thus combining the advantages of streamlines and tensor glyphs. ection; for example, a cross shape can be employed instead of an ellipse.

# Conclusions

Tensor visualization techniques, from PCA to hyper streamlines, offer powerful tools for interpreting complex, multi-dimensional data. These methods enhance our ability to analyze and understand data structures in various scientific and engineering fields, making them invaluable in data handling and visualization tasks. By reducing dimensionality, visualizing components, and using graphical representations like tensor glyphs and hyper streamlines, we can gain deeper insights into the underlying patterns and relationships within tensor data.

Tensor visualization techniques, from PCA to hyper streamlines, offer powerful tools for interpreting complex, multi-dimensional data. These methods enhance our ability to analyze and understand data structures in various scientific and engineering fields, making them invaluable in data handling and visualization tasks. By reducing dimensionality, visualizing components, and using graphical representations like tensor glyphs and hyper streamlines, we can gain deeper insights into the underlying patterns and relationships within tensor data.

Summary of Methods for Visualizing Tensor Data

In this chapter, we explored various techniques for visualizing tensor data. These datasets, which can be either 2D or 3D, consist of 2×22×2 or 3×33×3 tensor matrices at each sample point. These matrices typically contain second-order partial derivatives of some quantity. We employed principal component analysis (PCA) to derive the eigenvectors and eigenvalues of the tensor data. These eigenvectors and eigenvalues indicate the directions and magnitudes of the extremal variations in the quantity encoded by the tensor. Notably, these directions are invariant under coordinate transformations, making them highly relevant for visualization purposes in various applications.

Scalar and Vector Field Visualization

Tensor data can be simplified to scalar or vector fields, which are then visualized using appropriate scalar or vector visualization techniques. The scalar or vector fields may directly result from the PCA analysis (i.e., the eigenvalues and eigenvectors) or may be derived metrics, such as anisotropy measurements. This approach allows for a straightforward interpretation of the tensor data by focusing on a single aspect at a time.

Combined Visualization Techniques

Alternatively, tensor data can be visualized by integrating multiple PCA results into a single view. Techniques such as tensor glyphs and hyper streamlines combine several aspects of the PCA results, offering a more comprehensive view of the data. These methods can convey more complex information about the tensor data by showing the relationships between different tensor components simultaneously.

Ongoing Research and Application-Specific Methods

Tensor visualization is a rapidly evolving field, with new methods continually being developed to address specific application areas, such as clinical investigations using DT-MRI datasets. These specialized visualization methods often integrate multiple datasets into a single view and offer advanced user interaction capabilities. Users can explore the datasets by selecting regions of interest, adjusting color transfer functions, and controlling various visualization parameters.

These features enhance the ability to extract meaningful insights from complex tensor data.

Advanced Tools and Documentation

For those seeking more detailed information on tensor visualization techniques and tools, several resources provide comprehensive documentation. These include works by Kindlmann (2006), Slicer (2013), Schroeder et al. (2006), and the National Library of Medicine (2014). These resources offer in-depth explanations of the visualization methods and their applications, as well as guidance on using specific tools for tensor data visualization.

In summary, the methods and techniques presented in this chapter provide powerful tools for visualizing and interpreting tensor data. Whether reducing tensor data to scalar or vector fields or combining multiple aspects into a single view, these visualization methods are essential for understanding the complex structures and variations encoded in tensor datasets.

**Domain-Modeling Techniques**

# Introduction

Domain-modeling techniques form the last class of visualization techniques. By domain-modeling techniques, we mean those operations on datasets that modify the sampling domain representation but not the sampled data. As we shall see, domain-modeling techniques can modify the actual values of the data attributes stored on a given grid, for example in the case of resampling the data on a different grid. However, this modification does not change the reconstructed function, so the meaning of the data attributes stays the same, even though their internal representation may change. In this , we shall present a number of different modeling techniques: cutting, selection , constructing grids from scattered points, and grid-processing techniques.

# Cutting

Cutting methods are domain-modeling techniques that map the data from a given source domain to a target subdomain. Consider some function *f* defined on a domain D. Given a domain Dʹ∈ **D**, how can we compute the restriction of *f* to Dʹ? Let us now consider that *f* on D is represented by a sampled “source” dataset *D*ₛ=({*pᵢ*},{*cᵢ*},{*fᵢ*},{*Φ*ᵢ}). Cutting the domain D with the domain Dʹ means, essentially, resampling *f* from D to Dʹ. This implies creating a new target dataset *D*ʹₛ=({*p*ʹ*ᵢ*},{*c*ʹ*ᵢ*},{*f* ʹ*ᵢ*},{*Φʹᵢ*}). The grid points {*p*ʹ*ᵢ*}, cells {*c*ʹ*ᵢ*}, and interpolation functions {*Φ*ʹᵢ} of the target dataset are all user specified, since it is up to the user to say where to resample the source dataset. The attribute values {*f* ʹ*ᵢ*} are computed by sampling the reconstructed function  of the source dataset at the locations *p*ʹ*ᵢ* of the target dataset. The cutting operation has several properties. First, the target domain is assumed to be a subset of the source domain. More exactly, we assume the points {*p*ʹ*ᵢ*} of the target dataset to be contained in the cells {*cᵢ*} of the source dataset. Since we use convex cells in our datasets, this means that all cells {*c*ʹ*ᵢ*} in the target dataset are also contained in the cells {*cᵢ*} of the source dataset. We never attempt to evaluate the source dataset outside its sampling domain, hence the name cutting. A second property of cutting is that the dimensionality of the source and target datasets, and hence the interpolation functions *Φ*ᵢ and *Φ*ʹᵢ of the two, need not be the same. The only restriction is that the target dataset is of equal or lower dimensionality than the source dataset.

We will now present some of the most widely used variants of the cutting operation: extracting a brick, slicing, cutting with an implicit function, and generalized cutting.

1. Extracting a Brick

Extracting a brick, also called bricking or extracting a volume of interest (VOI), is a cutting operation that produces a target dataset with the same dimensionality as the source dataset. Moreover, the target grid points are a subset of the source grid points, {*p*ʹ*ᵢ*}∈{*pᵢ*}. Bricking takes advantage of the regular structure of sample points in uniform, structured, and rectilinear grids to efficiently implement the cutting operation. Uniform, structured, and rectilinear grids arrange their sample points in a regular axis-aligned lattice. For a d-dimensional dataset, we can identify every sample point by *d* integers *n₁,...,nd*, called structured coordinates. Hence, we can easily specify the target domain as an axis-aligned brick contained in the source dataset, defined by its minimum and maximum integer coordinates *(m₁,M₁),...,(md,Md)*,where *1 <mᵢ <Mᵢ <nᵢ* for all *i* ∈[*1,d*]. This set of structured coordinates is called the brick extent. When we use this definition of bricking, the extracted object extent is parallel with the dataset extent and not the coordinate axes, and the extracted brick contains only whole cells.

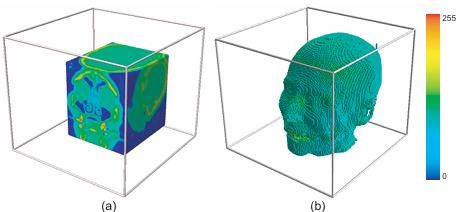


Figure 1.(a) shows a brick extracted from a volumetric magnetic resonance imaging (MRI) scan of a human head. The dataset extent is shown by the wireframe and the brick surface is color-mapped with the scalar values at the respective points of the extracted brick.(b)Selection of cells with scalar value above 50.

1. Slicing in Structured Datasets

Slicing is a cutting operation that is very similar to bricking. Given a uniform, rectilinear, or structured grid, we define a slice as all grid points that have one of the structured integer coordinates *n₁,...,nd*  equal. Extracting a slice can be seen as a bricking operation where the brick extent *(m₁,M₁),...,(md,Md)* is equal to the grid extent for *d*−1 of the dimensions, except for the slicing axis *s*, where *mₛ = Mₛ.* Slicing a *d*-dimensional dataset generates a *d*−1 dimensional dataset. Where we introduced the geometrical and topological dimensions of a dataset, this means that slicing creates cells of a lower dimension, but whose vertices are points in the same 3-dimensional space as the source dataset.

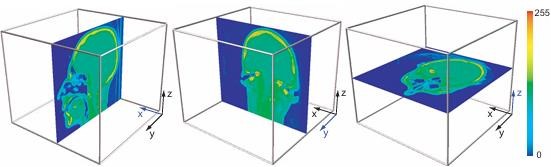


Figure 2.Slicing with planes perpendicular to the x-axis (left), y-axis (middle), and z-axis (right).

The most common type of slicing is extracting a set of planar cells, or a slice, from a volumetric dataset, hence the name slicing. Just as bricking, slicing is simple to implement: We iterate over all the sample points in the slice, in order of the structured coordinates *nᵢ, i ≠ s* that span the slice. Since our source dataset has a uniform, rectilinear, or structured grid, these integer coordinates directly correspond to d − 1-dimensional cells on the slice itself. In the target dataset, we save the sample points and the d − 1-dimensional cells on the slice plane, as well as their data attributes. We can do this easily by using the method of converting from point to cell attributes. The most common use of slicing is to extract 2D datasets from 3D volumes, and then visualize the extracted slices by one of the 2D visualization methods, such as color mapping or isolines for scalar data or streamlines for vector data. Figure 2 shows this technique applied to three slices perpendicular to the x-, y-, and z-axes of the same MRI scan uniform dataset as the one used in Figure 1. All three slices are taken at the middle of the respective axes of the dataset. In medical imaging, slices with these orientations are also called sagittal, axial, and coronal slices. If desired, slicing can reduce the dimensionality of the dataset by more than one dimension at a time. Slicing works also in higher dimensions.

1. Implicit Function Cutting

Slicing is a powerful and widely used visualization technique, especially when one wants to quickly browse through a large, high-dimensional dataset, without having to resort to slower, more complex visualization methods. Slicing is limited to structured topology datasets. Moreover, even for such datasets, slicing limits the extraction to subsets spanned by the dataset’s integer coordinate axes.

We can generalize the slicing concept that reduces the topological dimensionality of a dataset to different subspaces than those spanned by integer coordinates in structured datasets. One way to do this is to cut an arbitrary dataset with a given lower-dimensional domain. A simple way to specify the cutting domain is to use implicit functions. Given some function 𝜙 :D→ **R**, where D is the domain of the source dataset, we define the target or cutting, domain as all points p ∈ D for which 𝜙(p) = 0. To cut the source dataset, first,we compute a scalar dataset *D*cut that has the same grid as the source dataset and that evaluates 𝜙. Second, we compute a contour of *D*cut for the value zero.This yields an unstructured grid. This grid is our sampled representation of the cutting domain. Finally, we resample the source dataset attributes on this unstructured grid, using one of the available forms of interpolation, and we obtain the desired result.



1. Generalized Cutting

We can generalize the cutting technique presented in the previous sections by allowing a target dataset of arbitrary definition. Instead of using a target dataset defined by structured coordinates or an implicit function, we can use an arbitrary grid, as long as its cells are contained in the source dataset. In this case, the cutting operation is identical to the last part of the implicit function cutting procedure described previously. The source attributes are interpolated at the locations of the target grid vertices or target grid cell centers .

## Selection

In contrast to cutting, which projects the values of a source dataset to a target domain, selection methods extract the data from a source dataset based on data properties. Cutting enforces various geometrical and/or topological properties on the target domain, since the target grid is specified by the user, but cannot explicitly enforce any properties on the data values, as these are fully specified by the source dataset. In contrast, selection explicitly specifies which data values we are interested in, but cannot enforce, in general, a certain topology and/or geometry of the shape or connectivity of the extracted dataset domain.

Selecting cells: If we wish to extract cells, there are several ways to apply the selection criterion on a cell. A cell can meet the selection criterion if one of its vertices, all vertices, or its center point meet the selection criterion as defined for a point. The one-vertex criterion produces more cells, essentially selecting cells that are neighbors of the ones produced by the all-vertex selection criterion. The center point criterion is equivalent to applying the onepoint selection criterion on a slightly different sampling grid. If cells are selected in the output dataset, we assume these to have the same interpolation functions as in the input dataset, since we just copied them from the input dataset. If only points are selected in the output dataset, we actually create a scattered dataset. Finally, the output dataset is assumed to have the same interpolation functions as the input dataset.

Thresholding, segmentation, and contouring: Selection based on the scalar value matching a given target value s₀ produces results related to the contouring operation. Selection based on the scalar value being larger or equal than a given threshold s₀ produces one or more, compact subsets of the input dataset, also called threshold sets. Such an operation is known as thresholding or segmentation. A variant of segmentation tests the scalar value against a given value range [sₘᵢₙ,sₘₐₓ]. In addition to using the scalar values themselves, one can use their derivatives, too. In the case of scalar values that represent the luminance, or intensity, of an image, selecting data points based on the derivative values is related to edge-detection methods. Selection can also involve other properties than the data attributes of the current point. Selection is advantageous, as it lets us implement methods simply by designing different types of local selection functions *s* :D→ 𝔹. Moreover, selection methods can be easily parallelized, as they treat all data points independently. However, in some cases, we are interested in selection criteria that have a quasiglobal or global nature. This means, on one hand, that the selection criterion needs to check more points together to determine whether they pass or fail the test. On the other hand, the selection will output all these points as a set instead of separately. Figure 1(b) shows a selection of all cells from our sample MRI dataset whose scalar values are greater than or equal to 50.

# Grid Construction from Scattered Points

Gridless methods are attractive when one has to manipulate datasets that contain very large numbers of unstructured point samples, which have a rather high point density. One of the uses of gridless interpolation is to render surfaces represented as 3D dense point clouds. However attractive, gridless methods have also several drawbacks: They trade the grid storage and management for storing and managing some type of spatial search structure for neighboring sample points, and they use radial basis functions that are computationally more expensive compared to piecewise linear basis functions. Most visualization software packages would require the data to be in a grid-based representation of one of the standard dataset types before it can be processed by the available algorithms. Direct support for processing and visualizing data in gridless representations is not frequent. In such cases, constructing a grid from the scattered point set is a better alternative. There are several methods that construct grids from scattered points. We will now present several such methods.

1. Triangulation Methods

Triangulation methods are probably the most-used class of methods for constructing grids from scattered points. Given a set of points *p*ᵢ, a triangulation method produces a grid (*p*ᵢ,*c*ᵢ) by generating a set of cells *c*ᵢ that have the sample points *p*ᵢ as vertices. The cells *c*ᵢ form a tiling of the convex hull of the point set {*p*ᵢ}.

Delaunay triangulations: This method generates triangular cells *c*ᵢ for a set of 2D points *p*ᵢ ∈ **R**2 and tetrahedra for a set of 3D points *p*i ∈ **R**3. A Delaunay triangulation of a point set consists of a set of triangles that covers the convex hull of the point set. An important property of a Delaunay triangulation is that no point from the input point set {*p*i} lies in the circumscribed circle of any triangle in the triangulation. Triangulations that obey this property are called conforming Delaunay triangulations. Given a set of scattered points with data values recorded at the point locations, using the Delaunay triangulation is the most natural way to create a *C*1, piecewise linear, interpolation of the data values over the convex hull of the points. To do this, we define piecewise linear basis functions over the triangles contained in the unstructured grid generated by the Delaunay triangulation, and use these functions to interpolate the vertex data values, as explained in Figure 3(a) shows a Delaunay triangulation of a random point cloud containing 600 points.



Voronoi diagrams: For every Delaunay triangulation, there exists an associated geometric structure called a Voronoi diagram. A Voronoi diagram consists of a set of convex polygonal cells in 2D and polyhedral cells in 3D. The vertices of the Voronoi cells are the centers of the circumscribed circles of the triangles present in the associated Delaunay triangulation. The edges of the Voronoi cells are line segments contained in the lines perpendicular to, and passing through, the midpoints of the edges of the triangles present in the associated Delaunay triangulation. The centers of the Voronoi cells are the vertices of the Delaunay triangulation, i.e., the given scattered points. Figure 3(b) shows the Voronoi diagram of the same point set whose Delaunay triangulation is given in Figure 3(a).

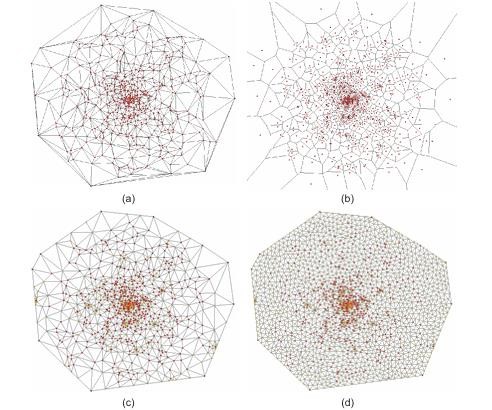


Figure 3. (a) Delaunay triangulation and (b) Voronoi diagram of a random point cloud. (c) Angle-constrained and (d) area-constrained Delaunay triangulations.

2. Surface Reconstruction and Rendering

A particular use of scattered-point interpolation is to render a 3D surface that is sampled by a point cloud. This task consists of two steps. First, we must specify which is the actual surface that the given point cloud approximates. Second, we must render this surface. For every point in the point cloud, we assume we have three pieces of information: the point’s location *p*i, the surface normal *n*i at that location, and the average distance *R*i to the neighboring points on the surface at that location. In the following sections, we describe several approaches for this.

Using radial basis functions: We approach this goal by first using 3D radial basis functions (RBFs) to construct a function



where Φ is the reference RBF in 3D and Ti−1 is the world-to-reference system coordinate transform for sample point *p*i.

If our basis functions 𝜙i =Φ(Ti−1 (x)) satisfy the partition of unity property over the entire domain defined by *S*, but not outside this domain, then the actual surface *S* is the isosurface ˜f(x) = 1 of the function ˜f. We can reconstruct the surface *˜S* by first computing a 3D volumetric dataset that samples the function ˜f and then extracting and rendering the isosurface ˜f =1, using the marching cubes algorithm. This has several disadvantages. First, we must explicitly compute, and possibly store, the 3D dataset that samples ˜f. Second, we must extract the isosurface, using, e.g., the marching cubes algorithm, and store it as a separate 3D unstructured mesh. This is expensive from both computational and memory viewpoints. Third, the basis functions must satisfy the partition of unity property.The last problem can be alleviated by using different ways to define the function ˜f whose isosurface we want

Using signed distance functions: A refinement of the previous method is to use a signed distance function. For every point *p*i in the point cloud, a tangent plane *T*i that approximates our surface *S* in the neighborhood of *p*i. The plane *T*i is defined by its center *ci* and normal *n*i. For every point *p*i in the point set, we determine a neighbor set *Ni* = *{pj|kRi≥|| pj−pi||}* that contains all neighbors of *p*i closer than a fraction *k* of the support radius *R*i of *p*i. Next, we compute Ti as the plane that minimizes the sum of the squared distances ∑*p*∈*Ni* d(*p,T*i)2 to the points in *N*i. It can be shown that center *c*i is the centroid of the points in *Ni:*



Figure 4. Scattered point cloud (left) and surface reconstruction with isosurface (right).

where |*N*i| denotes the number of points in the neighbor set *N*i and the normal ni of *T*i is the

eigenvector corresponding to the smallest eigenvalue of the 3 × 3 covariance matrix of the points *p* ∈ *N*i:

𝑎11 𝑎12 𝑎13

A=(ajk)= (𝑎21 𝑎22 𝑎23) with the elements

𝑎31 𝑎32 𝑎33

Here, *p*j denotes the *j*th component, or coordinate, of point *p*. Once we have the tangent planes *T*i, we define a function ˜f : **R**3 → **R** + such that ˜f(x) is the signed distance between a given point x ∈ **R** and the tangent plane *T*i at the point *p*i closest to x: ˜f(x)=(x*−c*i)·ni

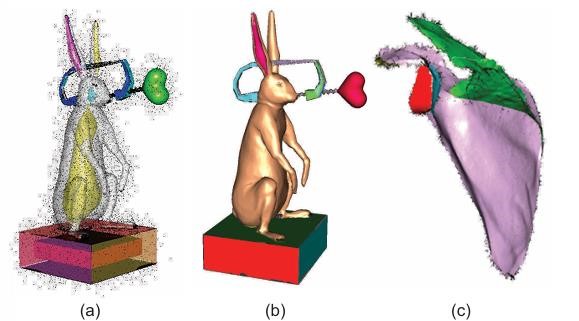


3



Local triangulations: Another class of methods constructs an unstructured triangle mesh from a scattered point set by performing local 2D Delaunay triangulations. First, we compute a tangent plane *T*i for every point *p*i, using its neighbor set *N*i. Next, we project the points in *N*i on *T*i and compute the 2D Delaunay triangulation Tri of these projections. Next, we add to our mesh those triangles that have *p*i as a vertex. Although this method is not guaranteed to produce a consistent triangle mesh, since it treats every point *p*i separately, it usually produces meshes with no defects such as holes or intersecting triangles. Also, this method is more memory efficient and computationally faster than the isosurface-based method.

Multiple local triangulations: The local triangulation method presented above assumes that the point cloud accurately approximates a single surface without self-intersections. Depending on their creation process, point clouds may contain outlier noise, or points which do not actually belong to the sampling of an actual surface. Where present, outliers will corrupt the estimation of the tangent plane *T*i may be inaccurate. Additionally point clouds may sample a set of intersecting surfaces. In intersection regions, we need to construct several, rather than a single, tangent plane *T*i. Both above problems can be handled if we allow our algorithm to accommodate several surfaces that pass through a point, and in the same time select the most likely such surfaces. In contrast to the earlier triangulation method, this is not guaranteed to produce triangles that lie in the same plane.



Black dots in Figure 5(a) show the input point cloud which contains many noisy outliers. The several surfaces reconstructed by the triangulation method are shown half-transparent. This method can capture surfaces with complex topologies, such as the outer and inner surfaces shown by model. Figure 5(b) shows the reconstructed outer surface for the same input cloud. Each color indicates a different surface component found by the method. As visible, noise points are excluded from the reconstruction. Figure 5(c) shows the reconstruction of a medial point cloud, or surface skeleton, of a shoulder-blade bone, also called a scapula.

Alpha shapes: Another surface reconstruction method for unoriented point sets is provided by alpha shapes. Assume that the 2D or 3D space containing our point set *P* is filled up with ice cream, and we have a spherical spoon of radius α, we first carve out all ice cream we can without removing, or touching, any point. The resulting shape will have a boundary composed of spherical pieces (in 3D) or circle arcs (in 2D). The alpha shape corresponding to *P* is obtained by straightening out these curved parts into line segments (in 2D), respectively triangles (in 3D).

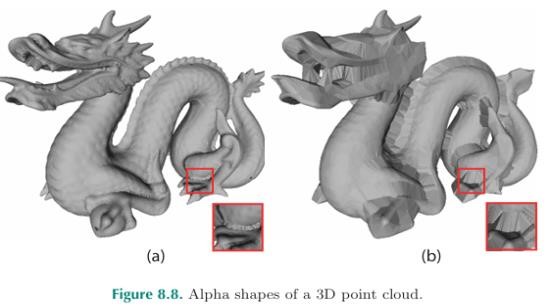
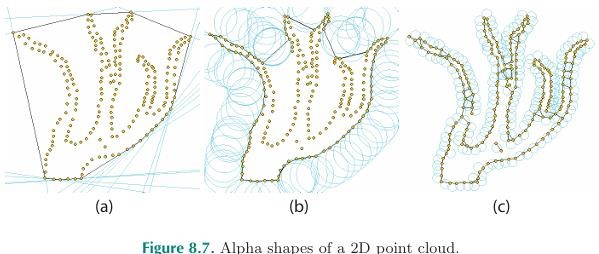


Figure 6 illustrates this for a 2D point. Here, points are rendered in yellow, and the circles corresponding to the closest locations of our carving spoon to the point set *P* are rendered in blue. For a value of α = ∞, we obtain the result in Figure 6(a). The spoon is now a half-space, thus we cannot carve inside any concavity of *P* .Consequently, the obtained alpha shape equals the convex hull of *P*. As we reduce the circle radius α, increasingly less shallow concavities are captured by the alpha shape (b). As visible, we cannot carve between the closely-located branches in the top of the shape. Reducing α even further yields the alpha shape in Figure 6 (c). The shape now captures most of the concavities visible in *P*. However, points on the perceived boundary of the shape implied by *P* which lie farther apart than α will also get disconnected, as the circle can pass through them to the interior. When α = 0, all space between the points can be carved out, so the corresponding alpha shape is the point set *P* itself.

Figure 7 shows 3D alpha shapes, the left image, obtained for a low α value, gives a good surface reconstruction of the point set. However, as visible in the zoom-in detail, stitches appear close to concave edges, where the α sphere cannot penetrate. The right image, obtained for a higher α value, shows clearly how these stitches fill up a larger part of the point set’s concavities.

Ball pivoting: A different technique to reconstruct 3D surfaces from oriented point clouds is ball pivoting. The principle of the method is simple: Given an oriented cloud with points *X* = {xi} and normals *N* = {ni}, and a fixed radius value ρ, we first find a seed triangle *T*seed having points in X such that a 3D ball *B*ρ of radius ρ touching these points contains no other data points. The edges of *T*seed are added to a so-called expanding front *F* and *T*seed is added to the reconstructed mesh *M*. Next, we iteratively pivot a ball *B*ρ around each edge *e =(p1,p2)*  *F* so that *B*ρ touches *p1* and *p2*. If the pivoting ball hits another point q  X, we distinguish three cases, as follows.

1. If *q* ∉ *M*, we grow *F* and *M* to include *Tʹ* =(*p1,p2,q*), i.e., remove e from *F* and add (*p1,q*) and (*p2,q*) to *F* and *Tʹ* to *M*, respectively.
2. *q* ∈ *M \ F*, i.e., *q* is an interior vertex of *M*, skip adding *Tʹ* to *M* and *e* becomes boundary edge for *M*.
3. If *q* ∈ *F*, we perform the operations from case (1), and also check *F* to remove identical edges with opposite orientation.

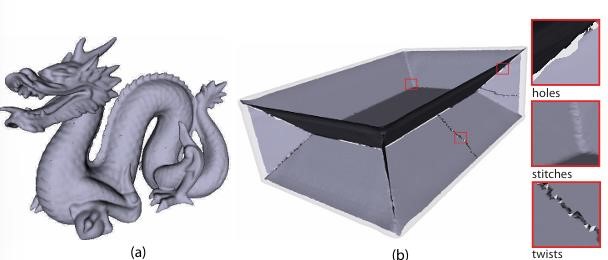
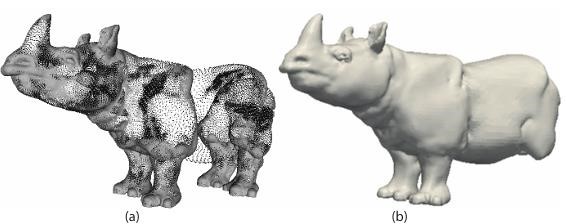
Figure 7. Ball pivoting reconstruction of a manifold (a) and nonmanifold (b) point cloud.

Figure 7(a) shows the surface reconstruction for the point cloud, using ball pivoting. Since the input cloud has a rather uniform point density, ball pivoting produces a high-quality surface with no holes or artificial stitches, and which captures the details present in the input cloud. Figure 7(b) Here, the point cloud contains the sampling of several intersecting manifolds—specifically, the cloud samples a surface skeleton of a 3D box, which is shown half-transparent in the figure. Here, ball pivoting creates several stitches and small gaps between the mesh reconstructions of these manifolds. Also, ball pivoting can have problems with the consistent orientation of some faces, in areas that contain noisy point normals. It is still one of the simplest, fastest, and thus best-known surface reconstruction algorithms in data visualization.

Poisson reconstruction: We can extend the basic idea of using distance functions to reconstruct surfaces from point clouds presented earlier in this section in several ways.

 Figure 8.Poisson reconstruction of a point cloud

The left image shows the actual point cloud. The right image shows the reconstruction result. As visible in the reconstruction image, the Poisson method can faithfully handle highly nonuniform point samples, such as the areas around the model’s eye and ear. However, regions where the cloud contains points sampling intersecting manifolds, such as the hind legs, can be altogether skipped in the reconstruction. Problems arise also in areas where the point cloud contains large holes, such as in the case of the sampling of a surface with boundaries. In such regions, the Poisson method will artificially close the surface with a smooth component. This cannot be avoided, since the reconstructed surfaces are computed via isosurfacing.

Surface splatting: The surface reconstruction methods presented so far produce a surface represented as a triangular mesh, whether via the marching cubes algorithm or directly by triangulating the point set. However, sometimes we need a simple-to-implement and fast, albeit possibly less accurate, method to directly render the surface S from the scattered points, without having to perform any ex-plicit surface reconstruction.

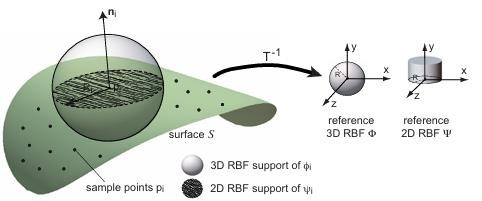


Figure 9. Radial basis functions for surface reconstruction

To display the surface *S* described by our point set, we can render the 2D radial basis functions ψi. Just as for the 3D RBFs, the 2D RBFs ψi are transformed versions of a reference 2D radial basis function that we shall call Ψ. If Φ is a 3D Gaussian then Ψ is a 2D Gaussian. If Φ is a 3D constant RBF, whose support would be a sphere of radius *R*, then Ψ is a 2D constant RBF, whose support is a disc of radius *R*. To draw our surface *S*, we have to draw the radial domains of ψi, which are nothing but discs of radius *R*i, centered at every sample point *p*i and oriented in the local tangent plane to *S*, which is perpendicular to the surface normal ni at *p*i. We can do this efficiently by taking advantage of the rendering primitives offered by modern graphics hard ware. First, we regularly sample the 2D radial basis function Φ on a pixel grid and store it as a 2D transparency texture *T*, where the value Φ = 1 maps to a totally opaque pixel *T* = 1 and a value Φ = 0 maps to a totally transparent pixel *T* = 0. The size of the texture *T* is taken so that *T* encloses the compact support of radius *R* of Φ.

Next, we implement the two-dimensional equivalent by drawing the texture T mapped onto square polygonal supports centered at the sample points *p*i, rotated to be orthogonal to the surface normals ni and scaled to the radius *R*i. For constant RBFs, this actually means drawing an opaque disc of radius *R*i at every point *p*i. We compute the surface lighting at the sample point locations *p*i only, using the available surface normals ni at those locations. The texture *T* that encodes a 2D RBF is sometimes called a splat or surfel. Hence, the previous surface reconstruction is also called splatting. Splats can encode other surface-information data necessary for the rendering besides the normal, color, and radius.

Sphere splatting: Surface splatting requires careful tuning of the shapes of the 2D basis functions φi to ensure that they satisfy the partition of unity on the surface *S*. In areas where this condition does not hold, gaps will become visible in *S*. An alternative reconstruction method that avoids such problems is sphere splatting. The key idea is to compute a function ˜f, with 3D constant RBFs Φ whose graphs are spheres of radii *R*, i.e., are equal to 1 for ||x|| <*R* and otherwise 0.

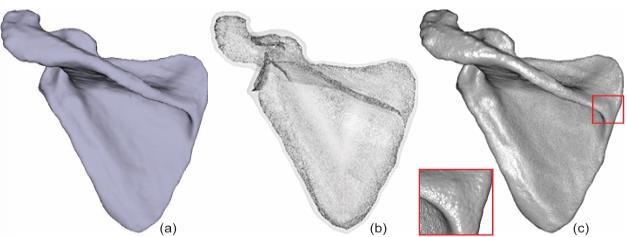
Figure 10. Sphere splatting for surface reconstruction from oriented point clouds.

Figure 10.(a) shows a polygonal rendering of the scapula bone. For illustration, we consider the oriented point cloud {*p*i,ni} obtained from this polygonal mesh by removing all triangles, keeping their vertices *p*i, and computing point normals i by averaging face normals. Figure 10(b) shows the medial point cloud {*q*i} computed from this oriented point cloud. The original polygonal mesh is shown half-transparent around the cloud for illustration purposes. Figure 10(c) shows the surface reconstruction using sphere splatting. The result is very similar to the original polygonal mesh. Zooming in, however, we can see that the reconstructed surface is nothing but a union of rendered spheres.

# Grid-Processing Techniques

Grid-processing techniques are methods that change both the grid geometry and its topology. By gridprocessing techniques, we mean those techniques that manipulate the grid itself, and have no knowledge about data attributes sampled on that grid. Such grid-processing techniques are used in many application domains besides data visualization, such as numerical methods and simulations or computer-graphics applications. There exist a wealth of grid-processing methods. Studying all these methods in depth would be a standalone subject matter by itself. However, since grid processing is important for visualization applications, we shall present a selection of some of the most-used grid-processing methods in data visualization.

1. Geometric Transformations

Geometric transformations are domain-modeling techniques that change the position of the sample points, or grid points, but do not modify the basis functions, cells, or data attributes. Transformations in this class include affine operations such as translation, rotation, and scaling, but also nonaffine operations, such as tapering, twisting, and bending. These transformations are relatively straight forward. A second type of geometric transformation changes the position of the sample points based on the data attributes. We have such techniques in the form of warping, height plots, and displacement plots, for the visualization of scalars and vectors. A third type of geometric transformation changes the position of the sample points based on the characteristics of the grid itself.

1. Grid Simplification

Many visualization applications produce large datasets that take considerable time to manipulate and store. One wants to reduce the size of these datasets, yet keep the data features that are important for the task at hand. Grid simplification, we mean situations involving grids that describe two-dimensional surfaces embedded in 3D, such as isosurfaces or polygonal models. The simplification criteria we shall look at here are mainly geometric, i.e., based on the shape of the grid. However, such criteria can be adapted to include information about the data attributes stored on the grid itself. An example of this technique is the progressive meshes method.

Many grid-simplification algorithms exist. The field is huge, given the applicability in computer graphics, data compression, mesh storage and transmission, shape matching, terrain visualization and rendering, and data visualization. It is, hence, impractical to aim for a comprehensive overview of grid-simplification methods in the current context. We next briefly describe a number of the main techniques used in this field: triangle mesh decimation, vertex clustering, and progressive meshes.

Triangle mesh decimation: Decimation algorithms were originally designed to reduce the huge number of triangles produced by the marching-cubes method. Given a triangle mesh, the algorithm does multiple passes over the mesh vertices, checking each vertex for removal. A vertex are removed if the removal does not change the mesh topology and if the resulting surface lies within a user-specified distance from the unsimplified surface. The hole left in the mesh is then retriangulated. Decimation continues until a user-specified reduction factor and/or some maximal error criterion are met. The decimated model contains a subset of the original mesh vertices, which is convenient for reusing the vertex information. However, this constraint can limit the decimation accuracy. A variant of the decimation algorithm can also handle topological changes.

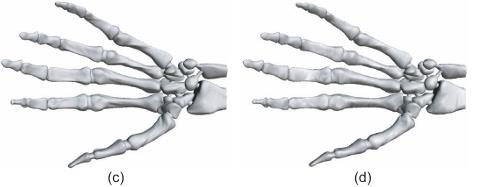
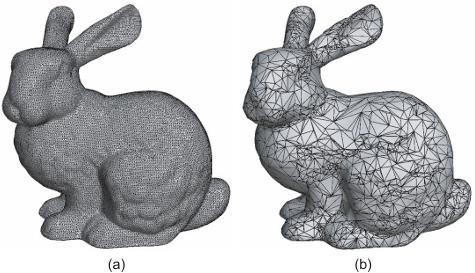


Figure 11. Decimation of a surface grid. (a) Original grid with 36,000 points and (b) decimated grid with 3510 points. (c) Original isosurface with 373,000 points and (d) decimated version with 6536 points.

Figure 11 shows two grid decimation examples produced by the latter algorithm. The bunny geometric model figure 11(a) is simplified to under 10% of the initial 36,000 grid points, yielding the result in Figure 11(b). As visible from the grid rendering, the simplification works adaptively. More triangles are kept in the high-curvature area around the bunny’s ear, marked red in Figure 11(b) in order to preserve the surface shape. In low-curvature areas, such as the bunny’s back, the simplification reaches its maximum. Figure 11(c) shows an isosurface of the skeleton of a human hand from a CT scan. The input data is a 3D uniform grid, so the marching-cubes algorithm produces an unstructured triangle mesh with almost constant point density, since the isosurface points are always located on the edges of the input grid. After the simplification, surface vertex normals are computed by cell normal averaging, which lets us create a rendering of the isosurface Figure 11(d) of comparable quality to the original.

Vertex clustering: Several algorithms simplify meshes by clustering vertices. Every vertex gets an importance value. Vertices attached to large polygons and vertices of high curvature are more important than vertices attached to small polygons and vertices of low curvature. A 3D grid is overlaid onto the mesh to be simplified. All vertices within a grid cell are collapsed to a new aggregated vertex position within that cell. The polygons whose vertices are collapsed together become degenerate and are removed. Vertices are sorted in importance order. All vertices within the cell are collapsed to the most-important vertex, which becomes the center of the next cell, and the process repeats. Different error metrics can be used to reposition this most-important vertex in order to ensure a simplified mesh close to the original.

Progressive meshes: A progressive mesh consists of a base mesh, created by a sequence of edge collapse operations on a polygonal mesh, and a sequence of vertex split operations. A split is the dual of a collapse, and it replaces a vertex with two edge-connected vertices, creating an extra vertex and two extra triangles. The base mesh can be exactly transformed into the original model via splits, and the model is transformed into the base mesh via collapses. Intermediate versions correspond to progressive simplifications. The collapses are driven by an energy function. Different types of energies can model simplifications driven by mesh geometry, normals, and color, but also additional data attributes. Attributes are classified as discrete, e.g., material and texture IDs, and scalar, e.g., color, normal, and texture coordinates. All edges are put into a priority queue in decreasing order of effect on the energy. The mesh is simplified by collapsing edges in this order, until topological constraints prevent further simplification. The remaining edges and triangles form the base mesh, and the sequence of collapse operations become the split operations

3. Grid Refinement

Grid refinement is the opposite of grid simplification. Given a coarse grid G that approximates some dataset D, refinement produces a grid G that also approximates D but has more sample points than the original grid G. Interms of sampling, the refined grid G can be seen as a supersampling of D as compared to the original grid G. Grid refinement has several uses. First, rendering a refined grid can produce a higher-quality image than rendering a coarse grid. A second use of refinement is as a preprocessing step before applying other grid manipulation operations. Also, deformations such as strong stretching change an uniformly sampled grid into a nonuniformly sampled one that exhibits coarsely sampled areas.

Loop subdivision: Refining a grid by adding extra sample points to the existing ones is a simple but effective strategy. A simple but effective method in this class is the Loop subdivision scheme. For surfaces S approximated by unstructured triangular grids G, this procedure works as follows. Each triangle T ∈ G is split into four other triangles, by adding extra vertices at the midpoints of the edges of T. This delivers the topology of the new grid Gʹ. Further on, the coordinates of the initial grid’s vertices, as well as the newly added vertices, are recomputed in order to create a grid Gʹ that best keeps the features of the original surface S.

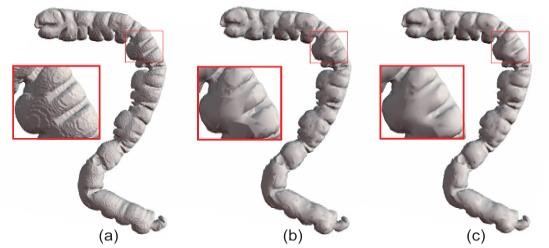
Figure 12. Refining an isosurface. (a) Original grid. (b) Simplified grid. (c) Refined grid. The zoomed-in insets show the grid quality.

Figure 12 shows a grid-refinement scenario using Loop subdivision. We start with an isosurface of a human colon Figure 12(a) containing 315,600 points. The isosurface exhibits strong staircasing artifacts due to

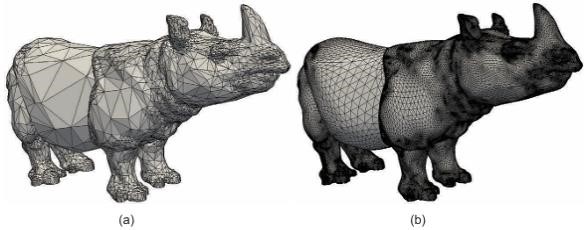
some type of nearest-neighbor approximation. We can render this isosurface using fewer data points and yet achieve a better visual quality. First, we simplify the surface using the decimation method. The simplification yields about 13,000 points but also decreases the visual quality of the surface can see Figure 12(b). If we refine the simplified surface by Loop subdivision, we obtain a surface with around 52,200 points and also a visibly better quality can see Figure 12(c). The final result has a sixth of the points of the initial isosurface and also arguably a better quality and smoother appearance.

Figure 13. (a) Nonuniform surface mesh and (b) result of two Loop subdivision steps.

However, Loop subdivision can produce suboptimal results. Consider the tri angular grid in Figure 13(a), which has 16,031 triangles. This mesh is highly nonuniform, contains large triangles and small triangles. The grid in Figure 13(b) shows the result of applying two steps of Loop subdivision. This mesh contains 16,031∗4∗4 = 256,496 triangles. Subdivision nicely refines the coarse areas from the initial mesh, such as the rhinoceros’ flanks. However, since every triangle is split, this creates many very small-size triangles in detail areas. These do not bring added value, but make the dataset larger, thus more costly to store and render. This problem did not occur for the colon dataset,because all its triangles were of relatively similar size.

Advanced subdivision tools: Just like grid simplification, refinement can also proceed adaptively by inserting more points where the surface varies more rapidly. As a measure of surface variation, we can use curvature or any scalar signal defined on the surface. Finally, if data attributes are defined on the original surface, these can be transferred to the refined surface by interpolation. A very good tool for simplification and refinement of triangular and quad grids is the Yams library. Yams can both simplify and refine surface meshes constrained by either a userdefined target size of the resulting cells, or adapting the size to a user-defined scalar signal defined on the surface. Yams can optimize a given mesh, by moving its vertices so that the result stays close to the input mesh but the resulting triangles are close to equilateral triangles. An alternative is the MeshLab, provides a wide range of surface processing techniques, including but not limited to grid simplification, refinement, remeshing, surface reconstruction from point clouds, and mesh cleaning, e.g., removing duplicate vertices, orienting normals consistently, non-manifold face removal.

4. Grid Smoothing

Grid-smoothing methods are a separate class of geometric transformations. Grid smoothing methods are most often used for grids that represent 2D curved sur faces embedded in **R**3, although they can be used also for 3D volumetric grids or 1D curve grids. Recall that, for such grids, we can reconstruct a piecewise continuous surface ˜*f* from the grid sample points xi by using the reconstruction formula. The aim of a grid-smoothing algorithm is to modify the positions of the grid sample points such that the surface ˜*f* reconstructed by the grid becomes smoother. Intuitively, a smooth surface is a surface that contains mostly blunt creases instead of sharp ones. Mathematically, this can be described by a surface with a low curvature. Grid-smoothing methods are useful in case the available grids contain geometric noise. This can be due either to the way the surface was acquired or to some other gridprocessing operations that have been previously applied on the sampled surface. The simplest way to think about diffusion is as smoothing.

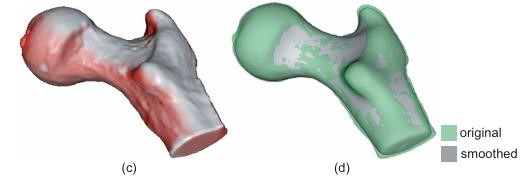
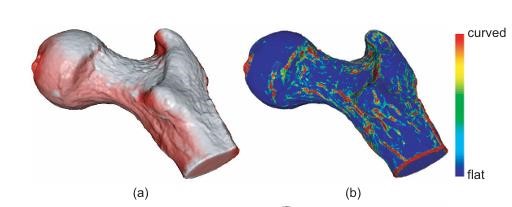


Figure 14. Laplacian smoothing of an isosurface. (a) Original surface. (b) Surface curvature. (c) Smoothed surface. (d) Comparison of original and smoothed surfaces.

Figure 14 shows the Laplacian smoothing in action on an isosurface showing a femur bone. The number of iterations combined with the diffusivity k can be used to control the scale of details one wants to remove from a given dataset. In Figure 14(c), small-scale irregularities of the original surface (Figure 14(a) are smoothed out in approximately 100 iterations. If more iterations are applied, larger-scale details would disappear, which is undesirable in this case. Figure 14(d) compares the initial object (green, semitransparent) with the smoothed one (gray, opaque) after 10,000 iterations, by rendering both objects overlapped in the same image. The smoothed object is visibly smaller than the original in the green regions and larger in the gray ones. Laplacian smoothing removes both noise and small-scale surface details. An undesired effect is that sharp surface creases, such as the edge of the femur cross section in Figure 14, also get smoothed out.

Grid-smoothing and grid-refinement methods can be effectively combined to produce high-quality grids. A simple way to do this is to first refine the grid, in order to create enough sample points so that the smoothing can act on a small spatial scale, and next perform a number of smoothing steps until the desired surface quality is reached. More efficient implementations can alternate smoothing and refinement steps and also introduce grid-decimation steps, all in order to redistribute the points to the optimal locations on the surface in order to achieve the desired quality. Grid processing is an extensive topic of research. Polygon mesh processing, a key ingredient to all domain processing algorithms.

## Conclusion

Grids are a fundamental element of visualization datasets. They provide a discrete representation of a compact spatial domain on which the signals to be visualized are sampled. Grids can vary in several respects: the type of cells, the regularity of the discretization, and the types of basis functions used to perform interpolation on the grid. In this chapter, we have presented a number of fundamental methods for grid manipulation and processing. Since grids are used in visualization as a representation of the underlying domain of a signal, these methods are referred to as domain-modeling techniques. Cutting techniques extract a lower-dimensional domain from a higherdimensional one, such as when slicing a volume with a surface, or a subdomain with the same dimensionality, such as in the case of bricking. Selection techniques extract a set of cells or points from a dataset, based on data properties. These techniques are useful when we are interested in extracting a specific subset of interest from a larger dataset. Grids can be constructed from scattered points using triangulation techniques. An alternative is to use gridless techniques, e.g., using radial basis functions. Triangulation has the advantage of producing a cellbased grid, which can be further used by all grid-based visualization methods. Gridless methods do not require the usually complex triangulation step, require less memory because they do not explicitly store cells, offer fast surface interpolationusing splatting, but bear additional computational costs. Either of these techniques can be used when we need a continuous domain representation, such as a surface, and all we have is a set of scattered points. Grids can be processed by a variety of operations, such as simplification, refinement, and smoothing. These operations are especially useful when the grid itself is the visualization target, such as when it represents a surface of interest, e.g., produced by contouring techniques. All in all, domain modeling techniques are an indispensable element of the visualization process, as grids are an indispensable ingredient of datasets.

# **SUMMARY**

## SCALAR VISUALIZATION

Scalar visualization techniques play a crucial role in understanding and interpreting data across various fields, including science, engineering, medicine, and beyond

In summary, scalar visualization serves as a bridge between raw data and meaningful insights, allowing us to perceive and comprehend complex phenomena in a visual manner. By leveraging techniques such as color mapping, contouring, and height plots, we can extract valuable information from scalar datasets, uncovering patterns, trends, and relationships that may otherwise remain hidden. As we continue to refine and innovate in the field of data visualization, scalar visualization techniques will undoubtedly remain indispensable tools for understanding the world around us.

## VECTOR VISUALIZATION

A number of visualization methods for vector fields – From simple visual representations, straightforward implementation: vector glyph – To multiscale textures animated in real-time, complex implementations: LIC: Line Integrated Convolution

• Another classification method: based on the dimensionality of the data domain – 2D surface: planar or curved ones – 3D volumetric vector fields, more challenging

• Inherent occlusion of the visualization primitives

• The challenge of creating insightful visualization of 3D time-dependent vector fields describing complex phenomena is still an active area of research

## TENSOR VISUALIZATION

In this chapter, we explored various techniques for visualizing tensor data. These datasets, which can be either 2D or 3D, consist of 2×22×2 or 3×33×3 tensor matrices at each sample point. These matrices typically contain second-order partial derivatives of some quantity. We employed principal component analysis (PCA) to derive the eigenvectors and eigenvalues of the tensor data. These eigenvectors and eigenvalues indicate the directions and magnitudes of the extremal variations in the quantity encoded by the tensor. Notably, these directions are invariant under coordinate transformations, making them highly relevant for visualization purposes in various applications.

## DOMAIN MODELLING TECHNIQUES

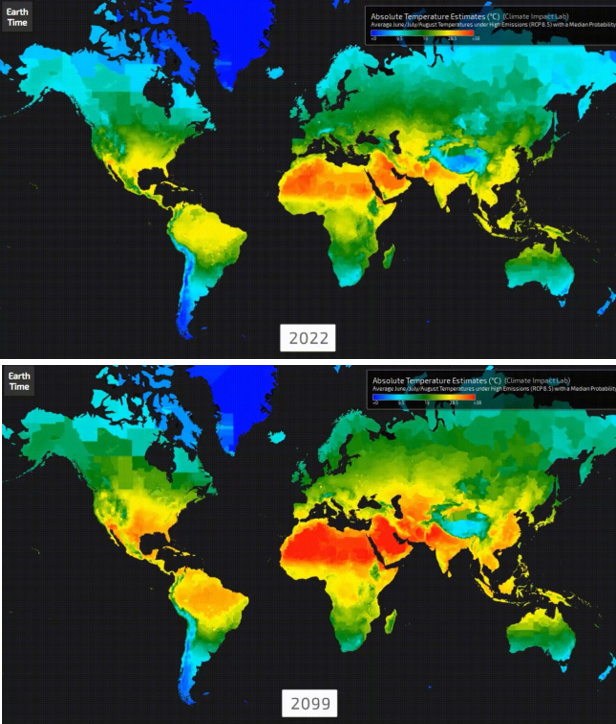
Grids can be constructed from scattered points using triangulation techniques. An alternative is to use gridless techniques, e.g., using radial basis functions. Triangulation has the advantage of producing a cellbased grid, which can be further used by all grid-based visualization methods. Gridless methods do not require the usually complex triangulation step, require less memory because they do not explicitly store cells, offer fast surface interpolationusing splatting, but bear additional computational costs. Either of these techniques can be used when we need a continuous domain representation, such as a surface, and all we have is a set of scattered points. Grids can be processed by a variety of operations, such as simplification, refinement, and smoothing. These operations are especially useful when the grid itself is the visualization target, such as when it represents a surface of interest, e.g., produced by contouring techniques. All in all, domain modeling techniques are an indispensable element of the visualization process, as grids are an indispensable ingredient of datasets

# EXERCISES

EXERCISE 01

Heat map cretation:

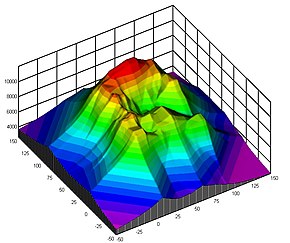
Create a heat map to visualize temperature variations across a geographic area . Gather temperature data for different locations in a city or region . Divide the area into a grid with equal-sized cells . Assign temperature values to each cell based on the data. Use a color scale to reprent different temperature ranges. Plot the heat map using software like Excel, MATLAB, or Pyhton



EXERCISE 02 :

Contour Plot Analysis

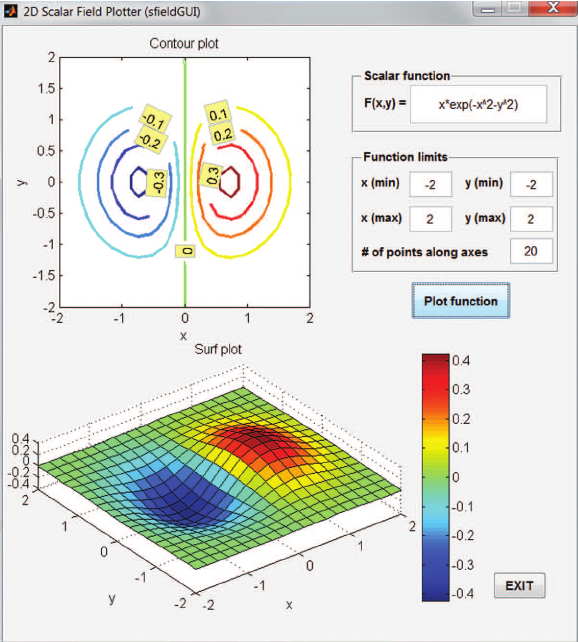
Generate and analyze a contour plot of elevation data . Obtain elevation data for a hilly or mountain region . Create a 2D grid over the region. Use software to generate contour lines at different elevation levels . Plot the contour map using GIS software or python



EXERCISE 03 :

Scalar filed simulation

Simulate and visualize a scalar field such as pressure distribution in a fuild. Create a simulation environment and define boundary conditions. Use computational fluid dynamics software or custom code. Python with libraries like /numPy and SciPy to simulate the pressure field . Visualize the pressure distribution using a color map or surface plot .



EXERCISE 04 :

3D Scalar field visualization

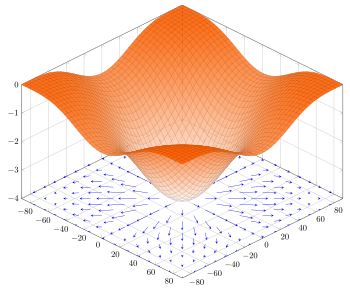
Visualize a 3D scalar field such as temperature in room. Obation temperature data at various points within a 3D space. Define the 3D grid for the space. Interpolate the temperature values across the grid . Use 3D plotting tools to create a volumetric representation.



EXERCISE 05 :

Gradient field interpretation

Use any of the scalar fields from previous exercises. Calculate the gradient vector field from the scalar field finite difference methods or analytical techniques. Plot the gradient vectors overload on the scalar field using arrows or streamlines



References :

[1]<https://lshoshia.science.tsu.ge/modeling/cignebi/Alexandru%20C.%20Telea-Data%20Visualization_%20Principles%20and%20Practice-A%20K%20Peters_CRC%20Press%20(2014).pdf>

[2]<https://www.cs.sjtu.edu.cn/~shengbin/course/datavis/LEC7.Domain.Modeling.pdf>

[3]<https://lshoshia.science.tsu.ge/modeling/cignebi/Alexandru%20C.%20Telea-Data%20Visualization_%20Principles%20and%20Practice-A%20K%20Peters_CRC%20Press%20(2014).pdf>

[4]<https://www.cs.sjtu.edu.cn/~shengbin/course/datavis/LEC7.Domain.Modeling.pdf>

[5]S. Bachthaler and D. Weiskopf. Continuous scatterplots. IEEE Transactions on Visualization and Computer Graphics, 14(6):1428–1435, 2008.

[6] S. Bachthaler and D. Weiskopf. Efficient and adaptive rendering of 2D continuous scatterplots. Computer Graphics Forum, 28(3):743–750, 2009.

[7] U. Bordoloi and H.-W. Shen. View selection for volume rendering. In Vis ’05: Proceedings of the IEEE Visualization 2005, pages 487–494, 2005.

[8] A. H. Charles and T. A. Porsching. Numerical Analysis of Partial Differential Equations. Prentice Hall, Engelwood, 1990.

[9] Y. Chen, J. Cohen, and J. Krolik. Similarity-guided streamline placement with error evaluation. IEEE Transactions on Visualization and Computer Graphics, 13(6):1448–1455, 2007.

[10] T. M. Cover and J. A. Thomas. Elements of Information Theory. WileyInterscience, 99th edition, August 1991.

[11] M. Feixas, E. del Acebo, P. Bekaert, and M. Sbert. An information theory framework for the analysis of scene complexity. Computer Graphics Forum, 18(3):95–106, 1999.

[12] Z. Liu, R. Moorhead, and J. Groner. An advanced evenly-spaced streamline placement algorithm. IEEE Transactions on Visualization and Computer Graphics, 12(5):965–972, 2006.

[13] A. Mebarki, P. Alliez, and O. Devillers. Farthest point seeding for efficient placement of streamlines. In Vis ’05: Proceedings of the IEEE Visualization 2005, pages 479–486, 2005.

[14] A. Orzan, A. Bousseau, H. Winnemoller, P. Barla, J. Thollot, and ¨ D. Salesin. Diffusion curves: a vector representation for smooth-shaded images. ACM Transactions on Graphics, 27(3):1–8, 2008.

[15] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever. Mutual information based registration of medical images: A survey. IEEE Transcations on Medical Imaging, 22(8):986–1004, 2003.

[20] J. Rigau, M. Feixas, and M. Sbert. Informational aesthetics measures. IEEE Computer Graphics and Applications, 28(2):24–34, 2008.

[21] R. Y. Rubinstein. Simulation and the Monte Carlo Method. John Wiley & Sons, 2008.

[22] M. Schlemmer, M. Heringer, F. Morr, I. Hotz, M. Hering-Bertram, C. Garth, W. Kollmann, B. Hamann, and H. Hagen. Moment invariants for the analysis of 2D flow fields. IEEE Transactions on Visualization and Computer Graphics, 13(6):1743–1750, 2007.

[23] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging, 13(1):146–168, 2004.

[24] B. Spencer, R. S. Laramee, G. Chen, and E. Zhang. Evenly spaced streamlines for surfaces: An image-based approach. Computer Graphics Forum, 28(6):1618–1631, 2009

[25] P.-P. Vazquez, M. Feixas, M. Sbert, and W. Heidrich. Automatic view s ´ election using viewpoint entropy and its applications to image-based modelling. Computer Graphics Forum, 22(4):689–700, 2003