Introduction to Data Visualization

Dr. Jyotismita Chaki

Overview

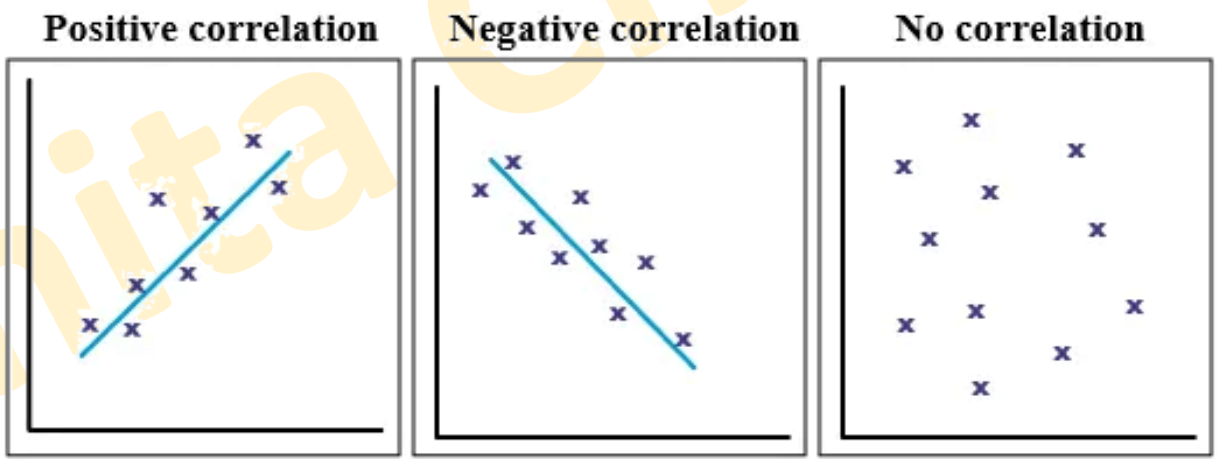
- Visualization, as the name implies, is based on exploiting the human visual system as a means of communication.
- The visual system provides a very high-bandwidth channel to our brains.
- A significant amount of visual information processing occurs in parallel at the preconscious level.
- Vis tools help people in situations where seeing the dataset structure in detail is better than seeing only a brief summary of it.
- One of these situations occurs when exploring the data to find patterns, both to confirm expected ones and find unexpected ones.
- Another occurs when assessing the validity of a statistical model, to judge whether the model in fact fits the data.

Overview

- Statistical characterization of datasets is a very powerful approach, but it has the intrinsic limitation of losing information through summarization.
- Figure 1 shows Anscombe's Quartet, a suite of four small datasets designed by a statistician to illustrate how datasets that have identical descriptive statistics can have very different structures that are immediately obvious when the dataset is shown graphically.
- All four have identical mean, variance, correlation, and linear regression lines.

Overview

- The scatterplot of the first dataset probably isn't surprising, and matches the meaning of mean (the average or the most common value in a collection of numbers.), variance (measures variability from the average or mean) and correlation (measure of the extent to which two variables are related).



Positive correlation	Negative correlation	No correlation
The points lie close to a straight line, which has a positive gradient.	The points lie close to a straight line, which has a negative gradient.	There is no pattern to the points.
This shows that as one variable **increases** the other **increases**.	This shows that as one variable **increases**, the other **decreases**.	This shows that there is **no connection** between the two variables.

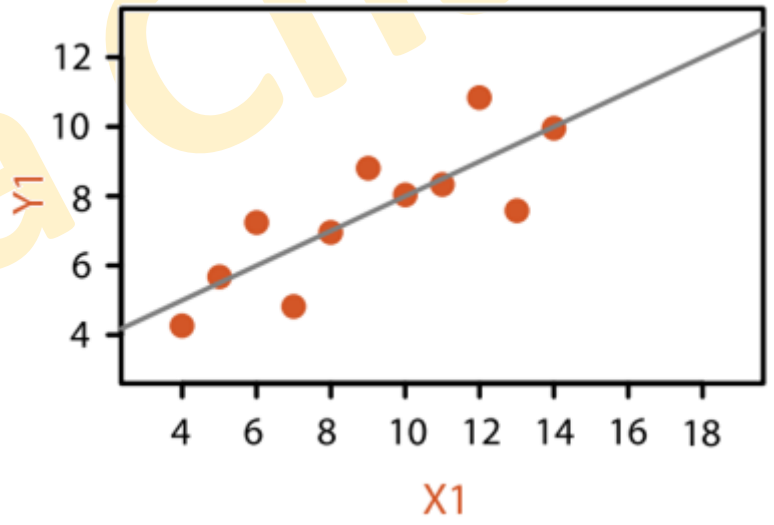
Overview

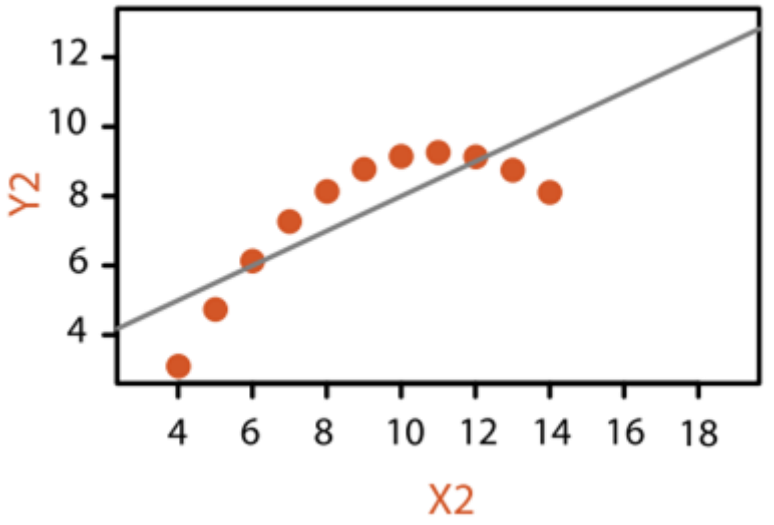
- The second scatterplot shows a clear nonlinear pattern in the data.
- The third dataset shows how a single outlier can lead to a regression line (a line which is used to describe the behavior of a set of data. In other words, it gives the best trend of the given data) that's misleading in a different way because its slope doesn't quite match the line that our eyes pick up clearly from the rest of the data.
- Finally, the fourth dataset shows a truly abmormal case where these measures dramatically mislead, with a regression line that's almost perpendicular to the true pattern we immediately see in the data.
- The basic principle illustrated by Anscombe's Quartet, that a single summary is often an oversimplification that hides the true structure of the dataset, applies even more to large and complex datasets.

Overview

Anscombe's Quartet: Raw Data

	1		2		3		4	
	X	Y	X	Y	X	Y	X	Y
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
Mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
Variance	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
Correlation	0.816		0.816		0.816		0.816	

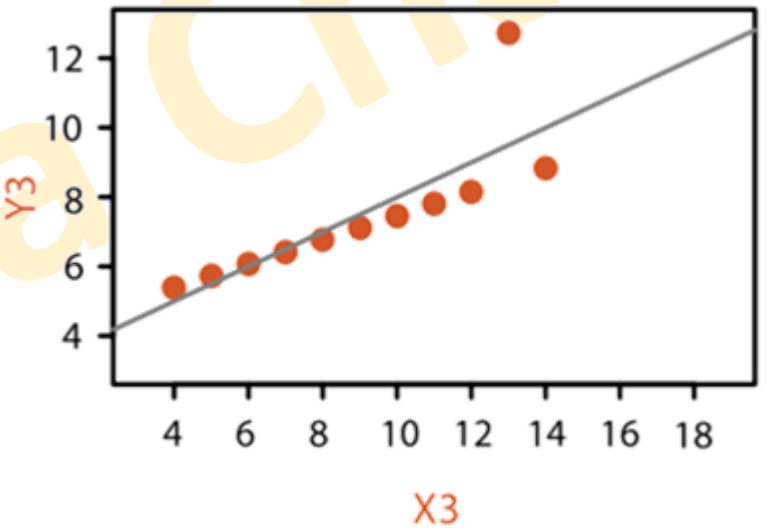


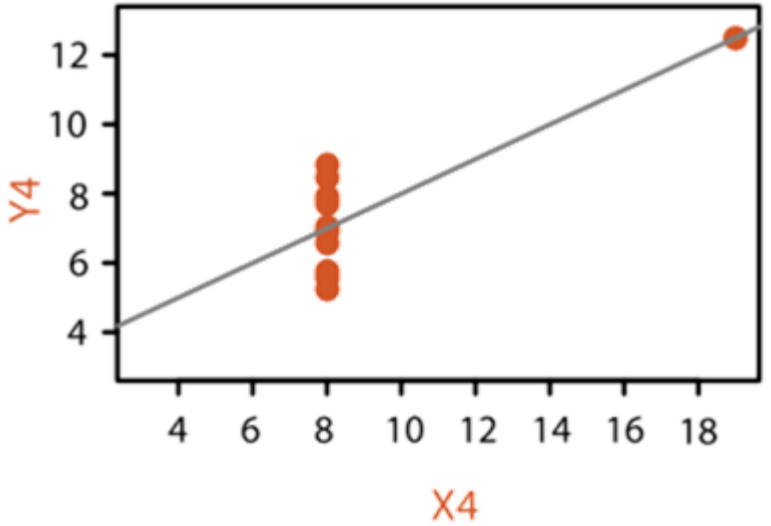


Overview

Anscombe's Quartet: Raw Data

	1		2		3		4	
	X	Y	X	Y	X	Y	X	Y
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
Mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
Variance	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
Correlation	0.816		0.816		0.816		0.816	





Overview: Interactivity

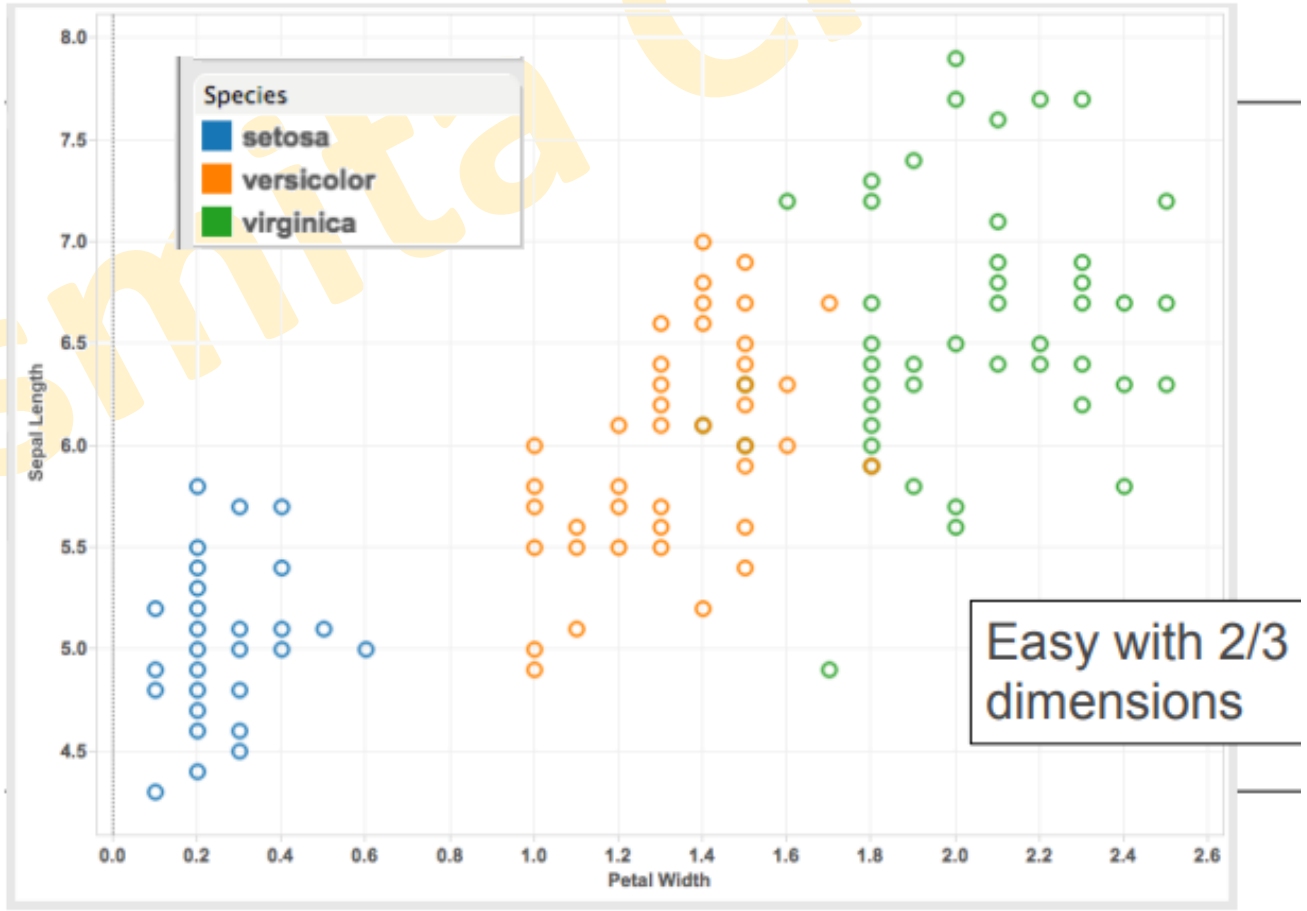
- Interactivity is crucial for building vis tools that handle complexity.
- When datasets are large enough, the limitations of both people and displays preclude just showing everything at once; interaction where user actions cause the view to change is the way forward.
- Moreover, a single static view can show only one aspect of a dataset.
- For some combinations of simple datasets and tasks, the user may only need to see a single visual encoding.
- In contrast, an interactively changing display supports many possible queries.
- In all of these cases, interaction is crucial.

Overview: Visualization Idioms

- A vis idiom is a distinct approach to creating and manipulating visual representations.
- There are many ways to create a visual encoding of data as a single picture.
- The design space of possibilities gets even bigger when you consider how to manipulate one or more of these pictures with interaction.
- Simple static idioms include many chart types that have deep historical roots, such as scatterplots, bar charts, and line charts.
- A more complicated idiom can link together multiple simple charts through interaction.

Overview: Visualization Idioms

ID	Sepal Length	Sepal Width	Petal Length	Petal Width	Species
14	4.3	3	1.1	0.1	setosa
39	4.4	3	1.3	0.2	setosa
43	4.4	3.2	1.3	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
42	4.5	2.3	1.3	0.3	setosa
23	4.6	3.6	1	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
7	4.6	3.4	1.4	0.3	setosa
3	4.7	3.2	1.3	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa
13	4.8	3	1.4	0.1	setosa
12	4.8	3.4	1.6	0.2	setosa
31	4.8	3.1	1.6	0.2	setosa
25	4.8	3.4	1.9	0.2	setosa
46	4.8	3	1.4	0.3	setosa
38	4.9	3.6	1.4	0.1	setosa
10	4.9	3.1	1.5	0.1	setosa
2	4.9	3	1.4	0.2	setosa
35	4.9	3.1	1.5	0.2	setosa
58	4.9	2.4	3.3	1	versicolor
107	4.9	2.5	4.5	1.7	virginica
36	5	3.2	1.2	0.2	setosa
5	5	3.6	1.4	0.2	setosa
50	5	3.3	1.4	0.2	setosa



Overview: Focus on Task

- A tool that serves well for one task can be poorly suited for another, for exactly the same dataset.
- The task of the users is an equally important constraint for a vis designer as the kind of data that the users have.
- Reframing the users' task from domain-specific form into abstract form (Tables: Data item (row) with attributes (columns) : row=key, cells = values; Image: 2d location = key, pixel value expresses etc) allows you to consider the similarities and differences between what people need across many real-world usage contexts.
- For example, a vis tool can support presentation, or discovery, or enjoyment of information; it can also support producing more information for subsequent use.

Data Abstraction: Semantics and Types

- Many aspects of vis design are driven by the kind of data that you have at your disposal.
  - What kind of data are you given?
  - What information can you figure out from the data, versus the meanings that you must be told explicitly?
- Suppose that you see the following data:
  - 14, 2.6, 30, 30, 15, 100001
  - What does this sequence of six numbers mean? (Is it locations for two points far from each other in three-dimensional space / Is it two points closer to each other in two-dimensional space etc)
  - Basil, 7, S, Pear: These numbers and words could have many possible meanings
  - You can't possibly know yet, without more information about how to interpret each number.

Data Abstraction: Semantics and Types

- To move beyond guesses, you need to know two crosscutting pieces of information about these terms: their **semantics** and their **types**.
- The **semantics** of the data is its real-world meaning. For instance,
  - Does a word represent a human first name, or is it the shortened version of a company name where the full name can be looked up in an external list, or is it a city, or is it a fruit?
  - Does a number represent a day of the month, or an age, or a measurement of height, or a unique code for a specific person, or a postal code for a neighborhood, or a position in space?

Data Abstraction: Semantics and Types

- The **type** of the data is its structural or mathematical interpretation.
  - At the **data level**, what kind of thing is it: an item, a link, an attribute?
  - At the **dataset level**, how are these data types combined into a larger structure: a table, a tree, a field of sampled values?
  - At the **attribute level**, what kinds of mathematical operations are meaningful for it?
  - For example,
    - if a number represents a count of boxes of detergent, then its type is a quantity, and adding two such numbers together makes sense.
    - If the number represents a postal code, then its type is a code rather than a quantity—it is simply the name for a category that happens to be a number rather than a textual name. Adding two of these numbers together does not make sense.

Data Abstraction: Semantics and Types

ID	Name	Age	Shirt Size	Favorite Fruit
1	Amy	8	S	Apple
2	Basil	7	S	Pear
3	Clara	9	M	Durian
4	Desmond	13	L	Elderberry
5	Ernest	12	L	Peach
6	Fanny	10	S	Lychee
7	George	9	M	Orange
8	Hector	8	L	Loquat
9	Ida	10	M	Pear
10	Amy	12	M	Orange

A full table with column titles that provide the intended semantics of the attributes

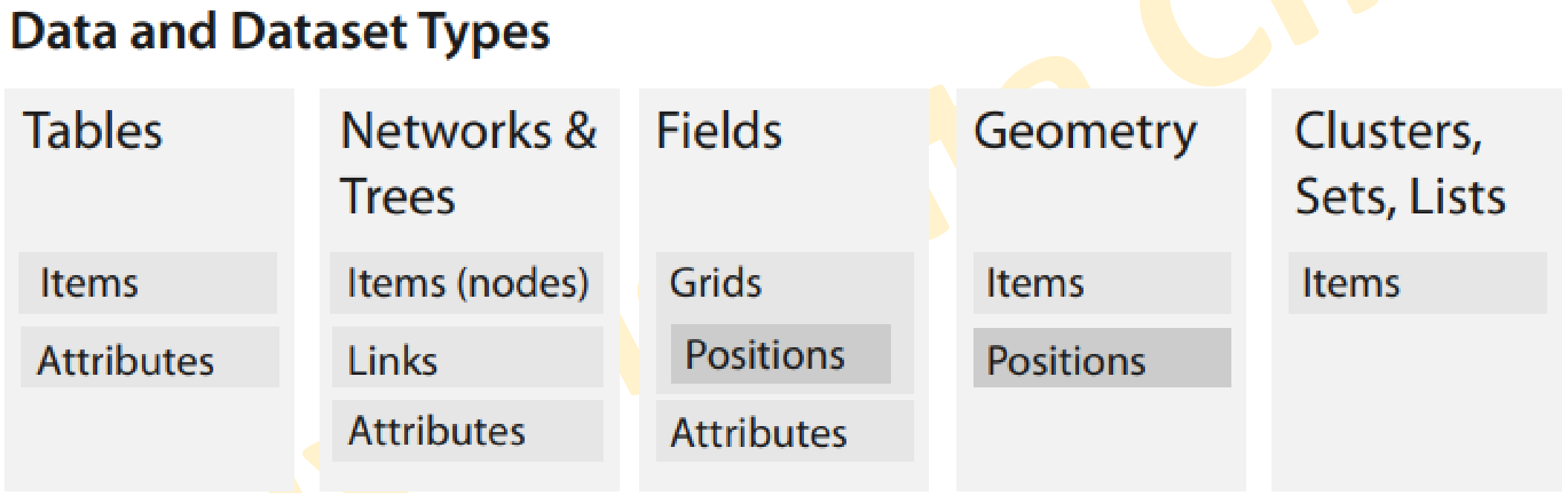
Data Abstraction: Data Types

- Five basic data types:
  - An **attribute** is some specific property that can be measured, observed, or logged. For Example: attributes could be salary, price, number of sales, protein expression levels, or temperature.
  - An **item** is an individual entity that is discrete, such as a row in a simple table or a node in a network. For example, items may be people, stocks, coffee shops, genes, or cities.
  - A **link** is a relationship between items, typically within a network.
  - A **grid** specifies the strategy for sampling continuous data in terms of both geometric and topological relationships between its cells.
  - A **position** is spatial data, providing a location in two-dimensional (2D) or three-dimensional (3D) space. For example, a position might be a latitude–longitude pair describing a location on the Earth's surface or three numbers specifying a location within the region of space measured by a medical scanner.

Data Abstraction: Dataset Types

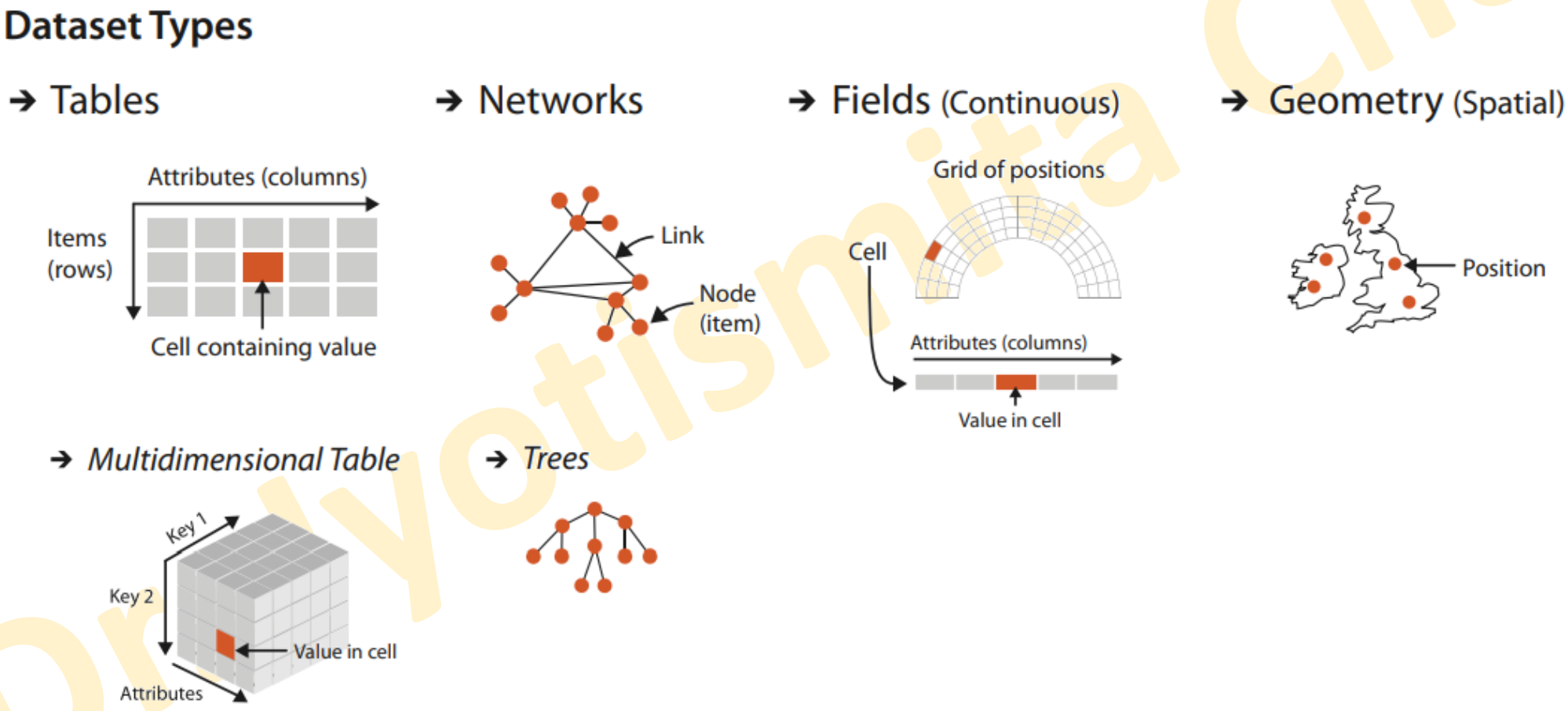
- A dataset is any collection of information that is the target of analysis.
- The four basic dataset types are
  - **Tables** have cells indexed by items and attributes, for either the simple flat case or the more complex multidimensional case.
  - In a **network**, items are usually called nodes, and they are connected with links; a special case of networks is trees.
  - Continuous **fields** have grids based on spatial positions where cells contain attributes.
  - Spatial **geometry** has only position information.
- Other ways to group items together include clusters, sets, and lists.

Data Abstraction: Dataset Types



The four basic dataset types are tables, networks, fields, and geometry; other possible collections of items are clusters, sets, and lists. These datasets are made up of five core data types: items, attributes, links, positions, and grids.

Data Abstraction: Dataset Types



The detailed structure of the four basic dataset types.

Data Abstraction: Dataset Types: Tables

- Tables that are made up of rows and columns.
- Each row represents an item of data, and each column is an attribute of the dataset.
- Each cell in the table is fully specified by the combination of a row and a column—an item and an attribute—and contains a value for that pair.
- A multidimensional table has a more complex structure for indexing into a cell, with multiple keys.

A	B	C	S	T	U
Order ID	Order Date	Order Priority	Product Container	Product Base Margin	Ship Date
3	10/14/06	5-Low	Large Box	0.8	10/21/06
6	2/21/08	4-Not Specified	Small Pack	0.55	2/22/08
32	7/16/07	2-High	Small Pack	0.79	7/17/07
32	7/16/07	2-High	Jumbo Box		7/17/07
32	7/16/07	2-High	Medium Box		7/18/07
32	7/16/07	2-High	Medium Box		7/18/07
35	10/23/07	4-Not Specified	Wrap Bag	0.52	10/24/07
35	10/23/07	4-Not Specified	Small Box	0.58	10/25/07
36	11/3/07	1-Urgent	Small Box	0.55	11/3/07
65	3/18/07	1-Urgent	Small Pack	0.49	3/19/07
66	1/20/05	5-Low	Wrap Bag	0.56	1/20/05
69	5	4-Not Specified	Small Pack	0.44	6/6/05
69	5	4-Not Specified	Wrap Bag	0.6	6/6/05
70	12/18/06	5-Low	Small Box	0.59	12/23/06

attribute

item

cell

Data Abstraction: Dataset Types: Networks and Trees

- **Networks** are well suited for specifying that there is some kind of relationship between two or more items.
- A synonym for networks is graphs.
- An item in a network is often called a node (vertex).
- A link (edge) is a relation between two items.
- For example, in an articulated social network the nodes are people, and links mean friendship.
- Networks with hierarchical structure are more specifically called **trees**.
- In contrast to a general network, trees do not have cycles: each child node has only one parent node pointing to it.
- One example of a tree is the organization chart of a company, showing who reports to whom

Data Abstraction: Dataset Types: Fields

- The field dataset type also contains attribute values associated with cells.
- Each cell in a field contains measurements or calculations from a continuous domain.
- For example,
  - Consider a field dataset representing a medical scan of a human body containing measurements indicating the density of tissue at many sample points, spread regularly throughout a volume of 3D space.
  - A low-resolution scan would have 262,144 cells, providing information about a cubical volume of space with 64 bins in each direction.
  - Each cell is associated with a specific region in 3D space.

Data Abstraction: Dataset Types: Fields: Spatial Fields

- Continuous data is often found in the form of a **spatial field**, where the cell structure of the field is based on sampling at spatial positions.
- For example,
  - With a spatial field dataset that is generated with a medical imaging instrument, the user's task could be to locate suspected tumors that can be recognized through distinctive shapes or densities.
  - An obvious choice for visual encoding would be to show something that spatially looks like an X-ray image of the human body and to use color coding to highlight suspected tumors.

Data Abstraction: Dataset Types: Fields: Grid Types

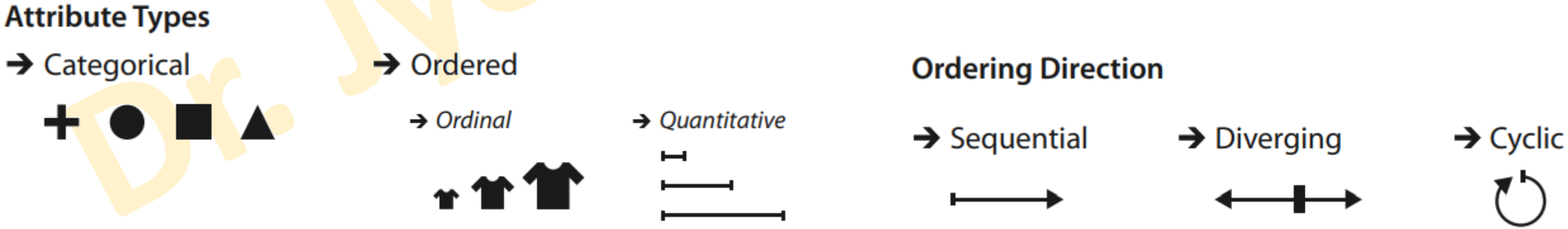
- When a field contains data created by sampling at completely regular intervals, the cells form a **uniform grid**.
- There is no need to explicitly store the **grid geometry** in terms of its location in space, or the **grid topology** in terms of how each cell connects with its neighboring cells.
- More complicated examples require storing different amounts of **geometric** and **topological** information about the underlying grid.
- A **rectilinear grid** supports nonuniform sampling, allowing efficient storage of information that has high complexity in some areas and low complexity in others, at the cost of storing some information about the geometric location of each each row.
- A **structured grid** allows curvilinear shapes, where the geometric location of each cell needs to be specified.
- Finally, **unstructured grids** provide complete flexibility, but the topological information about how the cells connect to each other must be stored explicitly in addition to their spatial positions.

Data Abstraction: Dataset Types: Geometry

- The geometry dataset type specifies information about the shape of items with explicit spatial positions.
- The items could be points, or one-dimensional lines or curves, or 2D surfaces or regions, or 3D volumes.
- Geometry datasets are intrinsically spatial, and like spatial fields they typically occur in the context of tasks that require shape understanding.
- Spatial data often includes hierarchical structure at multiple scales.
- Geometry datasets do not necessarily have attributes, in contrast to the other three basic dataset types.
- One classic example is when contours are derived from a spatial field.
- Another is when shapes are generated at an appropriate level of detail for the task at hand from raw geographic data, such as the boundaries of a forest or a city or a country, or the curve of a road.

Data Abstraction: Attribute Types

- The major distinction is between categorical versus ordered.
- Within the ordered type is a further differentiation between ordinal versus quantitative.
- Ordered data might range sequentially from a minimum to a maximum value, or it might diverge in both directions from a zero point in the middle of a range, or the values may wrap around in a cycle.
- Also, attributes may have hierarchical structure.



Data Abstraction: Attribute Types: Categorical

- The type of categorical data, such as favourite fruit or names, does not have an implicit ordering, but it often has hierarchical structure.
- Categories can only distinguish whether two things are the same (apples) or different (apples versus oranges).
- Fruit could be ordered alphabetically according to its name, or by its price—but only if that auxiliary information were available.
- However, these orderings are not implicit in the attribute itself, the way they are with quantitative or ordered data.
- Other examples of categorical attributes are movie genres, file types, and city names.

Data Abstraction: Attribute Types: Ordered

- All **ordered** data does have an implicit ordering, as opposed to unordered categorical data.
- This type can be further subdivided.
  - With **ordinal** data, such as shirt size, we cannot do full-fledged arithmetic, but there is a well-defined ordering. For ex: top-ten lists of movies, initial lineups for sports tournaments depending on past performance, {Small, medium, large} etc.
  - A subset of ordered data is **quantitative** data, namely, a measurement of magnitude that supports arithmetic comparison. For example, the quantity of 68 inches minus 42 inches is a meaningful concept, and the answer of 26 inches can be calculated. Other examples of quantitative data are height, weight, temperature, stock price, number of calling functions in a program, and number of drinks sold at a coffee shop in a day. Both integers and real numbers are quantitative data.

Data Abstraction: Attribute Types: Sequential versus Diverging

- Ordered data can be either **sequential**, where there is a homogeneous range from a minimum to a maximum value, or **diverging**, which can be deconstructed into two sequences pointing in opposite directions that meet at a common zero point.
- For instance, a **mountain height dataset** is **sequential**, when measured from a minimum point of sea level to a maximum point of Mount Everest.
- A full **elevation dataset** would be **diverging**, where the values go up for mountains on land and down for undersea valleys, with the zero value of sea level being the common point joining the two sequential datasets.

Data Abstraction: Attribute Types: Cyclic

- Ordered data may be cyclic, where the values wrap around back to a starting point rather than continuing to increase indefinitely.
- Many kinds of time measurements are cyclic, including the hour of the day, the day of the week, and the month of the year.

Data Abstraction: Key versus Value Semantics: Tables

- A simple **flat** table has only one key, where each item corresponds to a row in the table, and any number of value attributes.
- In this case, the key might be completely implicit, where it's simply the index of the row. It might be explicit, where it is contained within the table as an attribute.
- In this case, there must not be any duplicate values within that attribute. In tables, keys may be categorical or ordinal attributes, but quantititive attributes are typically unsuitable as keys because there is nothing to prevent them from having the same values for multiple items.

Data Abstraction: Key versus Value Semantics: Tables

- Name is a categorical attribute that might appear to be a reasonable key at first, but the last line shows that two people have the same name, so it is not a good choice.
- Favorite Fruit is clearly not a key, despite being categorical, because Pear appears in two different rows.
- The quantitative attribute of Age has many duplicate values, as does the ordinal attribute of Shirt Size.
- The first attribute in this flat table has an explicit unique identifier that acts as the key.
- This key attribute could either be ordinal, for example if the order that the rows were entered into the table captures interesting temporal information, or categorical, if it's simply treated as a unique code.

ID	Name	Age	Shirt Size	Favorite Fruit
1	Amy	8	S	Apple
2	Basil	7	S	Pear
3	Clara	9	M	Durian
4	Desmond	13	L	Elderberry
5	Ernest	12	L	Peach
6	Fanny	10	S	Lychee
7	George	9	M	Orange
8	Hector	8	L	Loquat
9	Ida	10	M	Pear
10	Amy	12	M	Orange

Data Abstraction: Key versus Value Semantics: Tables

- The more complex case is a **multidimensional** table, where multiple keys are required to look up an item.
- The combination of all keys must be unique for each item, even though an individual key attribute may contain duplicates.
- For example, a common multidimensional table from the biology domain has a gene as one key and time as another key, so that the value in each cell is the activity level of a gene at a particular time.

Data Abstraction: Key versus Value Semantics: Fields

- Fields are structured by sampling in a systematic way so that each grid cell is spanned by a unique range from a continuous domain.
- In spatial fields, spatial position acts as a quantitative key, in contrast to a nonspatial attribute in the case of a table that is categorical or ordinal.
- Fields are typically characterized in terms of the number of keys versus values.
- Their **multivariate** structure depends on the number of value attributes, and their **multidimensional** structure depends on the number of keys.

Data Abstraction: Key versus Value Semantics: Fields

- A field can be both multidimensional and multivariate if it has multiple keys and multiple values.
- The standard classification according to multivariate structure is that a **scalar** field (univariate) has one attribute per cell (temperature in a room at each point in 3D space), a **vector** field (multivariate) has two or more attributes per cell (velocity of air in the room at a specific time point, where there is a direction and speed for each item), and a **tensor** field has many attributes per cell (nine numbers that represent forces acting in three orthogonal directions).

Data Abstraction: Temporal Semantics

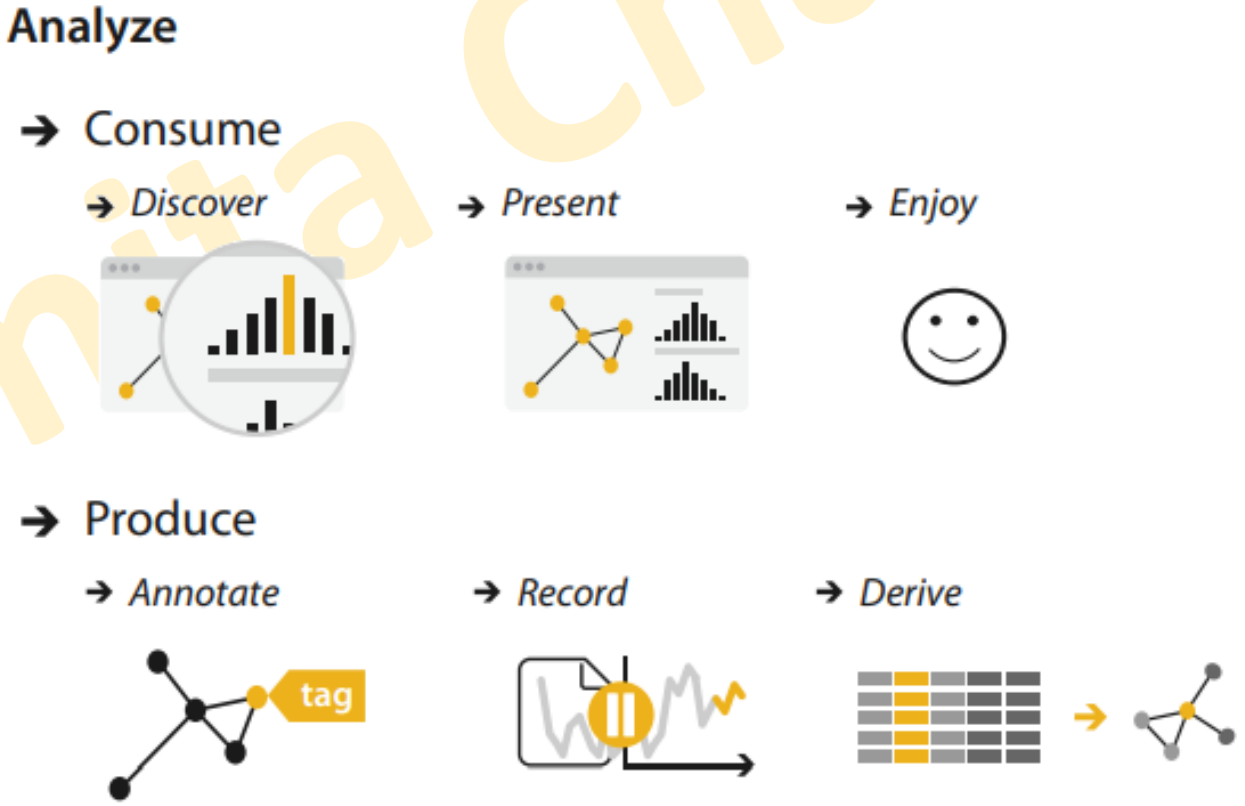
- A temporal attribute is simply any kind of information that relates to time.
- The time hierarchy is deeply multiscale: the scale of interest could range anywhere from nanoseconds to hours to decades to millennia.
- Temporal analysis tasks often involve finding or verifying periodicity either at a predetermined scale or at some scale not known in advance.
- A dataset has time-varying semantics when time is one of the key attributes, as opposed to when the temporal attribute is a value rather than a key.
  - An example of a dataset with time-varying semantics is one created with a sensor network that tracks the location of each animal within a herd by taking new measurements every second.
  - Each animal will have new location data at every time point, so the temporal attribute is an independent key and is likely to be a central aspect of understanding the dataset.

Task Abstraction

- Transforming task descriptions from domain-specific language into abstract form allows you to reason about similarities and differences between them.
- Otherwise, it's hard to make useful comparisons between domain situations, because if you don't do this kind of translation then everything just appears to be different.
- Another important reason to analyze the task is to understand whether and how to transform the user's original data into different forms by deriving new data.
- That is, the task abstraction can and should guide the data abstraction.

Task Abstraction: Actions: Analyze

- Three levels of actions that define user goals.
- The high-level choices describe how the vis is being used to **analyze**, either to consume existing data or to also produce additional data.
- The framework has three further distinctions within that case:
  - whether the goal is to present something that the user already understands to a third party (Present)
  - for the user to discover something new or analyze information that is not already completely understood (Discover)
  - for users to enjoy a vis to indulge their casual interests in a topic (Enjoy)



Task Abstraction: Actions: Analyse: Consume: Discover

- The discover goal refers to using vis to find new knowledge that was not previously known.
- Discovery may arise from the observation of unexpected phenomena, but investigation may be motivated by existing theories, models, hypotheses, or hunches.
- This usage includes the goal of finding completely new things; that is, the outcome is to generate a new hypothesis.
- It also includes the goal of figuring out whether a guess is true or false; that is, to verify—or disconfirm—an existing hypothesis.
- The fundamental motivation of this analysis framework is to help you separate out the questions of **why** the vis is being used from **how** the vis idiom is designed to achieve those goals.

Task Abstraction: Actions: Analyse: Consume: Present

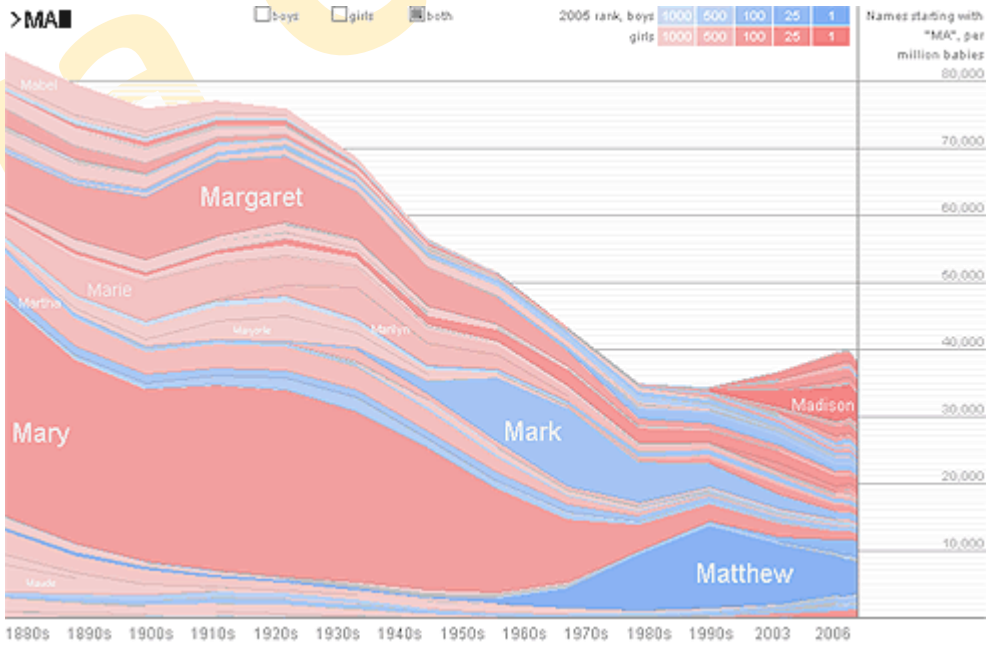
- The present goal refers to the use of vis for the succinct communication of information, for telling a story with data, or guiding an audience through a series of cognitive operations.
- Presentation using vis may take place within the context of decision making, planning, forecasting, and instructional processes.
- The crucial point about the present goal is that vis is being used by somebody to communicate something specific and already understood to an audience.
- Presentation may involve collaborative contexts, and the means by which a presentation is given may vary according to the size of the audience, whether the presentation is live or prerecorded, and whether the audience is in the same place as the presenter.
- A crucial aspect of presentation is that the knowledge communicated is already known to the presenter in advance.
- Sometimes the presenter knows it before using vis at all and uses the vis only for communication.
- In other cases, the knowledge arose from the presenter's previous use of vis with the goal of discovery, and it's useful to think about a chained sequence of tasks where the output of a discover session becomes the input to a present session.

Task Abstraction: Actions: Analyse: Consume: Enjoy

- The enjoy goal refers to casual encounters with vis.
- One aspect of this classification that's tricky is that the goals of the eventual vis user might not be a match with the user goals guessed by the vis designer.
- For example, a vis tool may have been intended by the designer for the goal of discovery with a particular audience, but it might be used for pure enjoyment by a different group of people.

Task Abstraction: Actions: Analyse: Consume: Enjoy

- **Name Voyager**, which was created for expectant parents deciding what to name their new baby.
- When the user types characters of a name, the vis shows data for the popularity of names in the United States since 1900 that start with that sequence of characters.
- The tool uses the visual encoding idiom where each name has a stripe whose height corresponds to popularity at a given time.
- Currently popular names are brighter, and gender is encoded by color.
- The Name Voyager appealed to many people with no interest in having children, who analyzed many different historical trends and posted extensively about their findings in their personal blogs, motivated by their own enjoyment rather than a pressing need



Task Abstraction: Actions: Analyse: Produce

- In contrast to using vis only for consuming existing information, in the produce case the intent of the user is to generate new material.
- Often the goal with produce is to produce output that is used immediately, as input to a next instance.
- Sometimes the user intends to use this new material for some other vis-related task later on, such as discovery or presentation.
- Sometimes the intended use of the new material is for some other purpose that does not require vis.

Task Abstraction: Actions: Analyse: Produce: Annotate

- The annotate goal refers to the addition of graphical or textual annotations associated with one or more preexisting visualization elements, typically as a manual action by the user.
- When an annotation is associated with data items, the annotation could be thought of as a new attribute for them.
- For example, the user could annotate all of the points within a cluster with a text label.

Task Abstraction: Actions: Analyse: Produce: Record

- The record goal saves or captures visualization elements as persistent artifacts.
- These artifacts include screen shots, lists of book marked elements or locations, parameter settings, interaction logs, or annotations.
- The record choice saves a persistent artifact, in contrast to the annotate, which attaches information temporarily to existing elements; an annotation made by a user can subsequently be recorded.
- One interesting example of a record goal is to assemble a graphical history, in which the output of each task includes a static snapshot of the view showing its current state, and these snapshots accumulate in a branching meta-visualization showing what occurred during the user's entire session of using the vis tool.

Task Abstraction: Actions: Analyse: Produce: Derive

- The derive goal is to produce new data elements based on existing data elements.
- New attributes can be derived from information contained within existing ones, or data can be transformed from one type into another.
- Deriving new data is a critical part of the vis design process.
- The common case is that deriving new data is a choice made by vis designers, but this choice could also be driven by a user of a vis tool.
- When you are faced with a dataset, you should always consider whether to simply use it as is, or to transform it to another form: you could create newly derived attributes from the original ones, or even transform the dataset from the original type to another one.
- There is a strong relationship between the form of the data— the attribute and dataset types—and what kinds of vis idioms are effective at displaying it.

Task Abstraction: Actions: Analyse: Produce: Derive

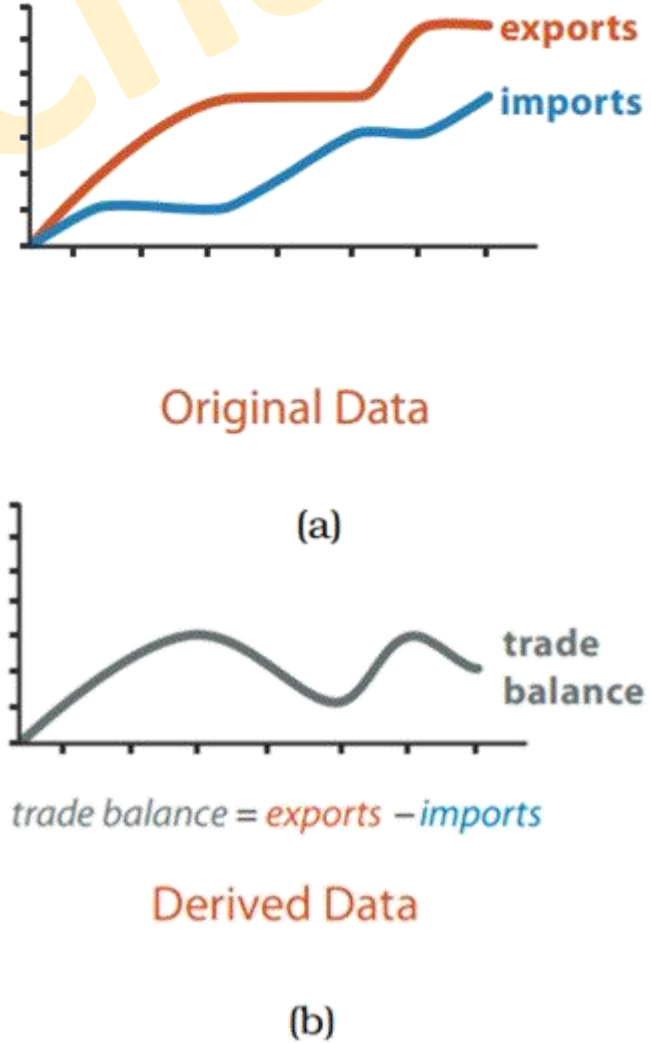
- A dataset often needs to be transformed beyond its original state in order to create a visual encoding that can solve the desired problem. To do so, we can create **derived attributes** that extend the dataset beyond the original set of attributes that it contains.
- In some cases, the derived attribute encodes the same data as the original, but with a change of type.
- For example,
  - a dataset might have an original attribute that is quantitative data: for instance, floating point numbers that represent temperature.
  - For some tasks, like finding anomalies in local weather patterns, that raw data might be used directly.
  - For another task, like deciding whether water is an appropriate temperature for a shower, that quantitative attribute might be transformed into a new derived attribute that is ordered: hot, warm, or cold. In this transformation, most of the detail is aggregated away.

Task Abstraction: Actions: Analyse: Produce: Derive

- In other cases, creating the derived attribute requires access to additional information.
- For a geographic example, a categorical attribute of city name could be transformed into two derived quantitative attributes containing the latitude and longitude of the city.
- This transformation could be accomplished through a lookup to a separate, external database.

Task Abstraction: Actions: Analyse: Produce: Derive

- A new derived attribute may be created using arithmetic, logical, or statistical operations ranging from simple to complex.
- Figure shows an example of encoding two attributes directly, versus encoding the derived variable of the difference between them.
- For tasks that require understanding this difference, Figure (b) is preferable because it encodes the difference directly.
- The user can interpret the information by judging position along a common frame. In contrast, in Figure (a) the user must judge the difference in heights between the two original curves at each step



Task Abstraction: Search

- The mid-level choices cover what kind of **search** is involved, in terms of whether the target and location are known or not.

Search

	Target known	Target unknown
Location known	 *Lookup*	 *Browse*
Location unknown	 *Locate*	 *Explore*

Task Abstraction: Search: Lookup

- If users already know both what they're looking for and where it is, then the search type is simply lookup.
- For example, a user of a tree vis showing the ancestral relationships between mammal species might want to look up humans, and can get to the right spot quickly by remembering how humans are classified: they're in the group that has live young rather having a pouch like kangaroos, and within that group humans fall into the category of primates.

Task Abstraction: Search: Locate and Explore

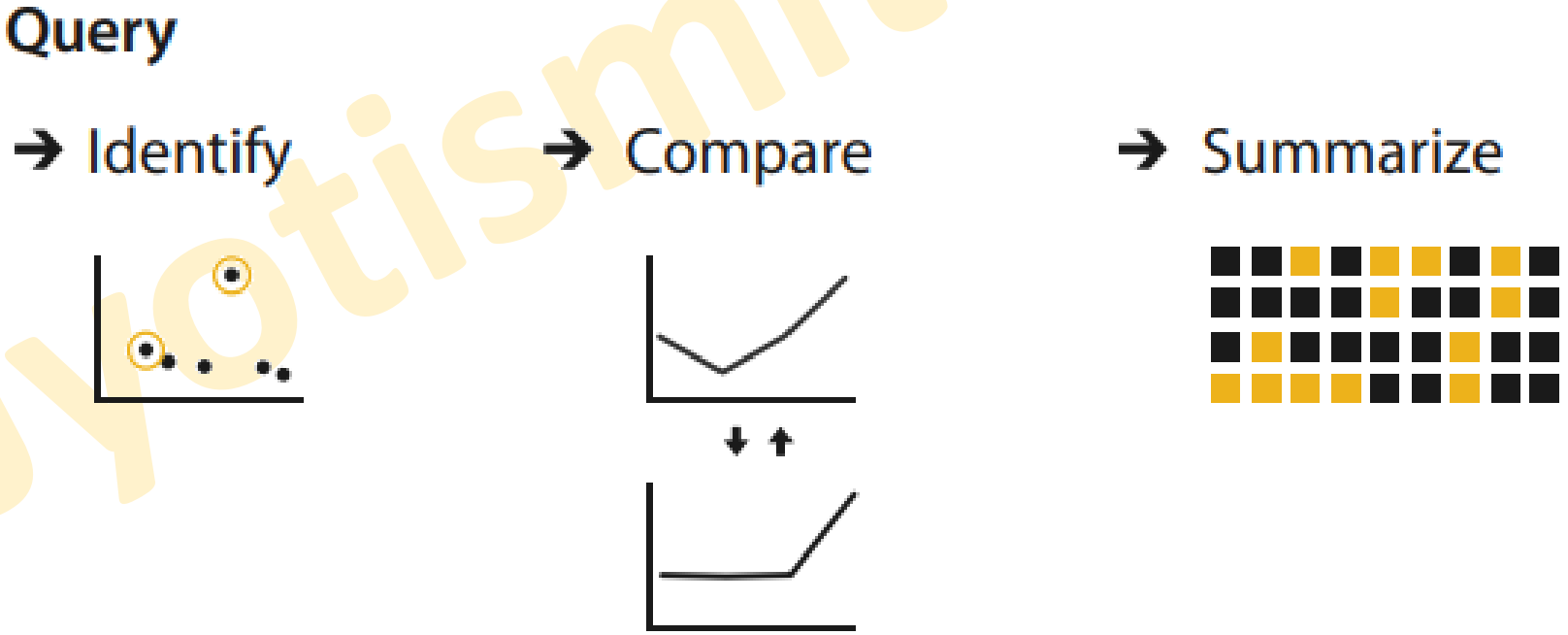
- To find a known target at an unknown location, the search type is **locate**: that is, find out where the specific object is.
- In a similar example, the same user might not know where to find rabbits, and would have to look around in a number of places before locating them as lagomorphs.
- When users are not even sure of the location, the search type is **explore**.
- It entails searching for characteristics without regard to their location, often beginning from an overview of everything.
- Examples include searching for outliers in a scatterplot.

Task Abstraction: Search: Browse

- In contrast, the exact identity of a search target might not be known in advance; rather, it might be specified based on characteristics.
- In this case, users are searching for one or more items that fit some kind of specification, such as matching up with a particular range of attribute values.
- When users don’t know exactly what they’re looking for, but they do have a location in mind of where to look for it, the search type is browse.
- For instance, if a user of a tree vis is searching within a particular subtree for leaf nodes having few siblings, it would be an instance of browse because the location is known in advance, even though the exact identity of the search target isn’t.

Task Abstraction: Query

- The low-level choices pertain to the kind of **query**: does the user need to identify one target, compare some targets, or summarize all of the targets?

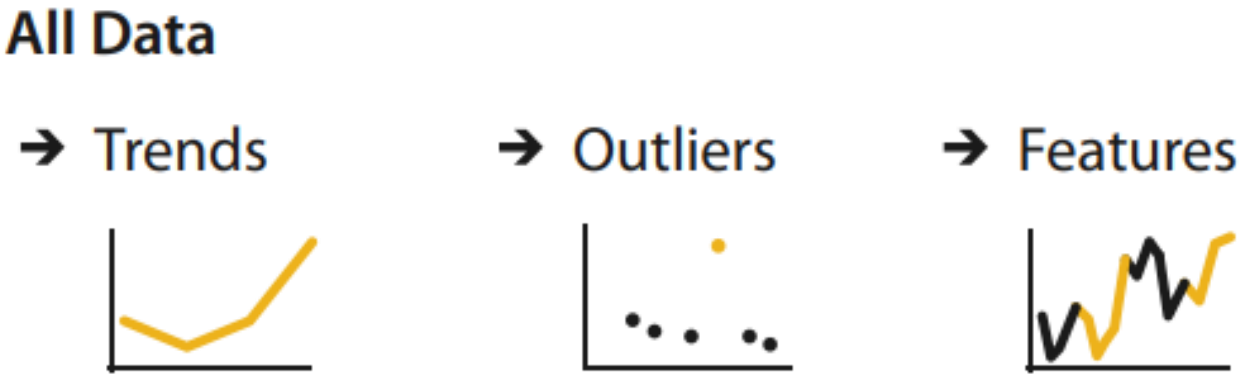


Task Abstraction: Query

- Once a target or set of targets for a search has been found, a lowlevel user goal is to query these targets at one of three scopes: **identify**, **compare**, or **summarize**.
- The progression of these three corresponds to an increase in the amount of search targets under consideration: one, some, or all.
- That is, **identify** refers to a single target, **compare** refers to multiple targets, and **summarize** refers to the full set of possible targets.
- Example: A user can identify the election results for one state, compare the election results of one state to another, or summarize the election results across all states to determine how many favored one candidate or the other or to determine the overall distribution of margin of victory values.

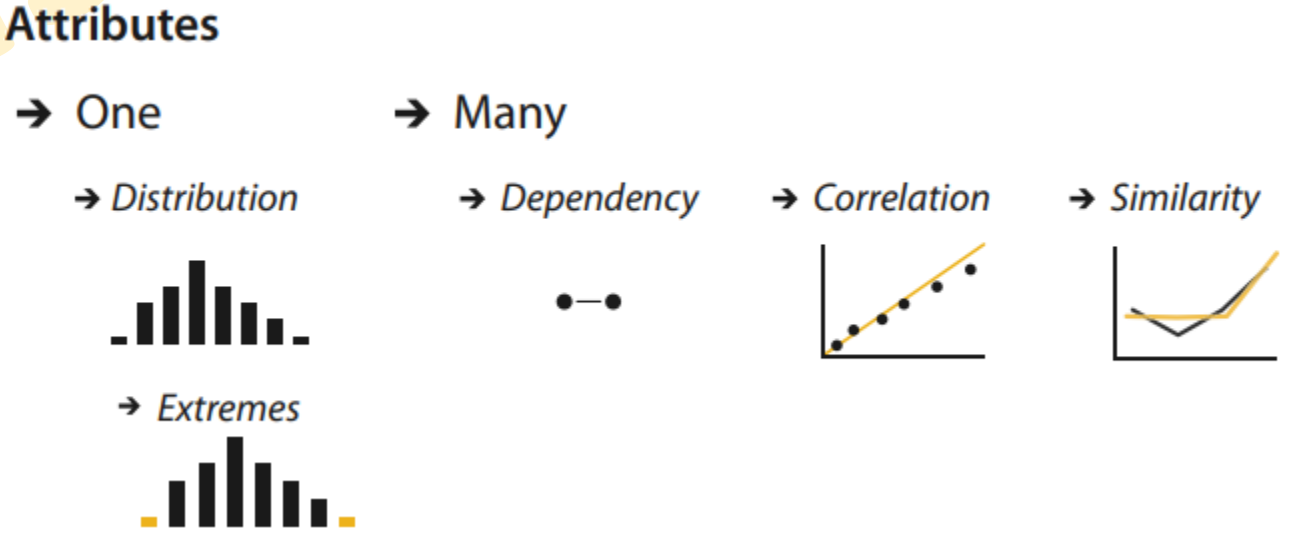
Task Abstraction: Targets

- Target, meaning some aspect of the data that is of interest to the user.
- Three high-level targets are very broadly relevant, for all kinds of data:
  - A **trend** is a high-level characterization of a pattern in the data.
  - Some data doesn't fit well with that backdrop; those elements are the **outliers**.
  - The exact definition of **features** is task dependent, meaning any particular structures of interest.



Task Abstraction: Targets

- Attributes are specific properties that are visually encoded. The lowest-level target for an attribute is to find an individual value.
- Another frequent target of interest is to find the **extremes**: the minimum or maximum value across the range.
- A very common target that has high-level scope is the **distribution** of all values for an attribute.

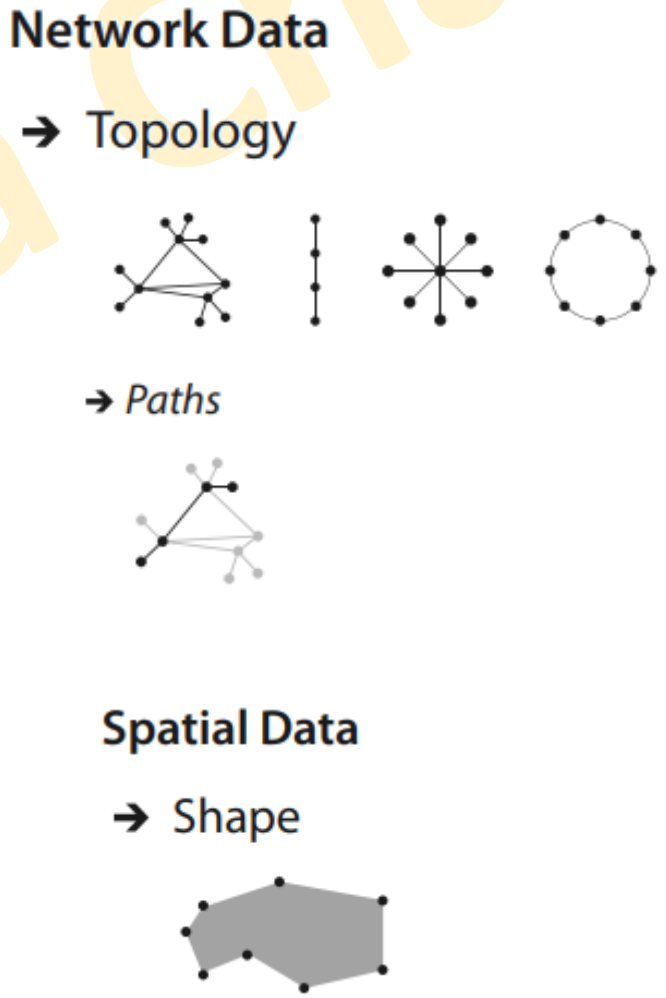


Task Abstraction: Targets

- Some targets encompass the scope of multiple attributes:
  - A first attribute can have a **dependency** on a second, where the values for the first directly depend on those of the second.
  - There is a **correlation** between one attribute and another if there is a tendency for the values of second to be tied to those of the first.
  - The **similarity** between two attributes can be defined as a quantitative measurement calculated on all of their values, allowing attributes to be ranked with respect to how similar, or different, they are from each other.

Task Abstraction: Targets

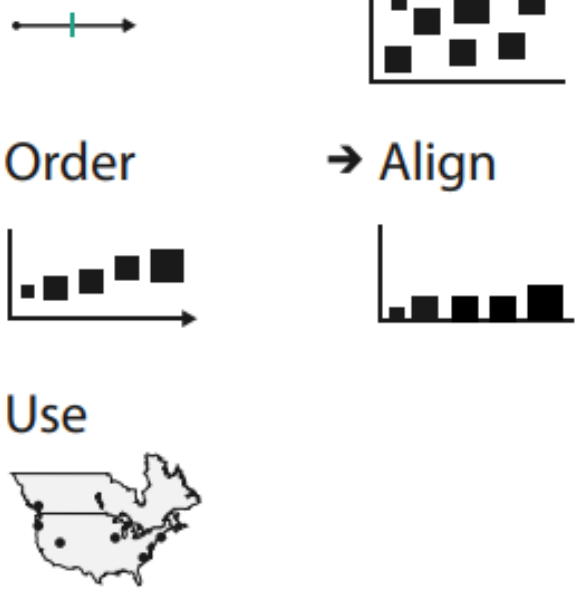
- Network data specifies relationships between nodes as links.
- The fundamental target with network data is to understand the structure of these interconnections; that is, the network's **topology**.
- A more specific topological target is a **path** of one or more links that connects two nodes.
- For **spatial** data, understanding and comparing the geometric shape is the common target of user actions.



How a vis idiom can be constructed

Encode

- **Arrange**
  - Express
  - Separate
  - Order
  - Align
  - Use

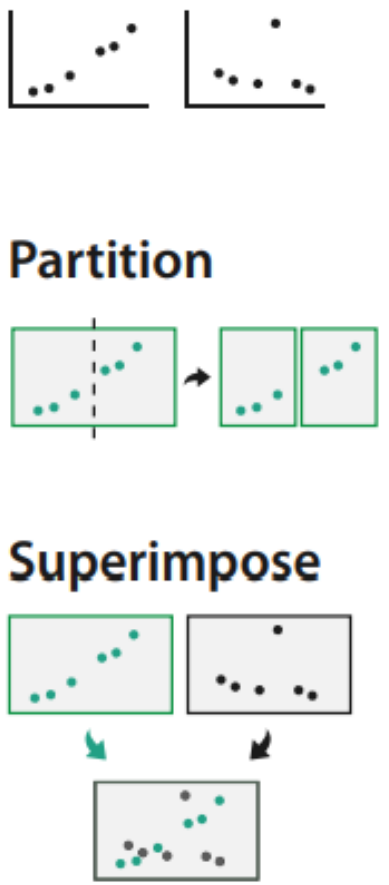


Manipulate

- **Change**
- **Select**
- **Navigate**

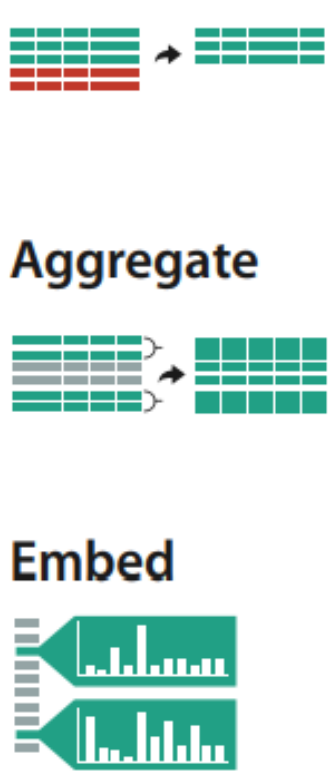
Facet

- **Juxtapose**
- **Partition**
- **Superimpose**



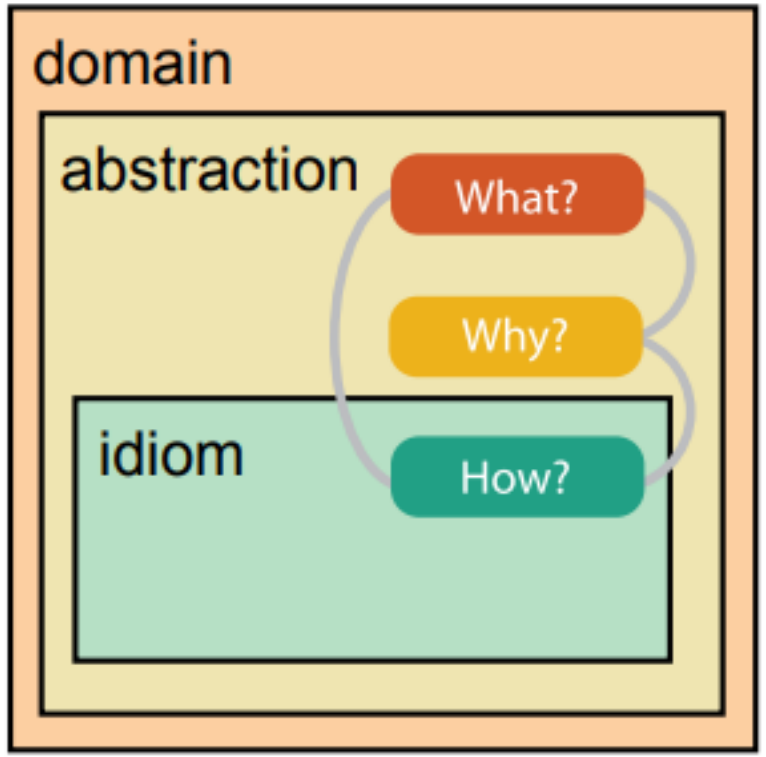
Reduce

- **Filter**
- **Aggregate**
- **Embed**



Analysis Framework: What-Why-How

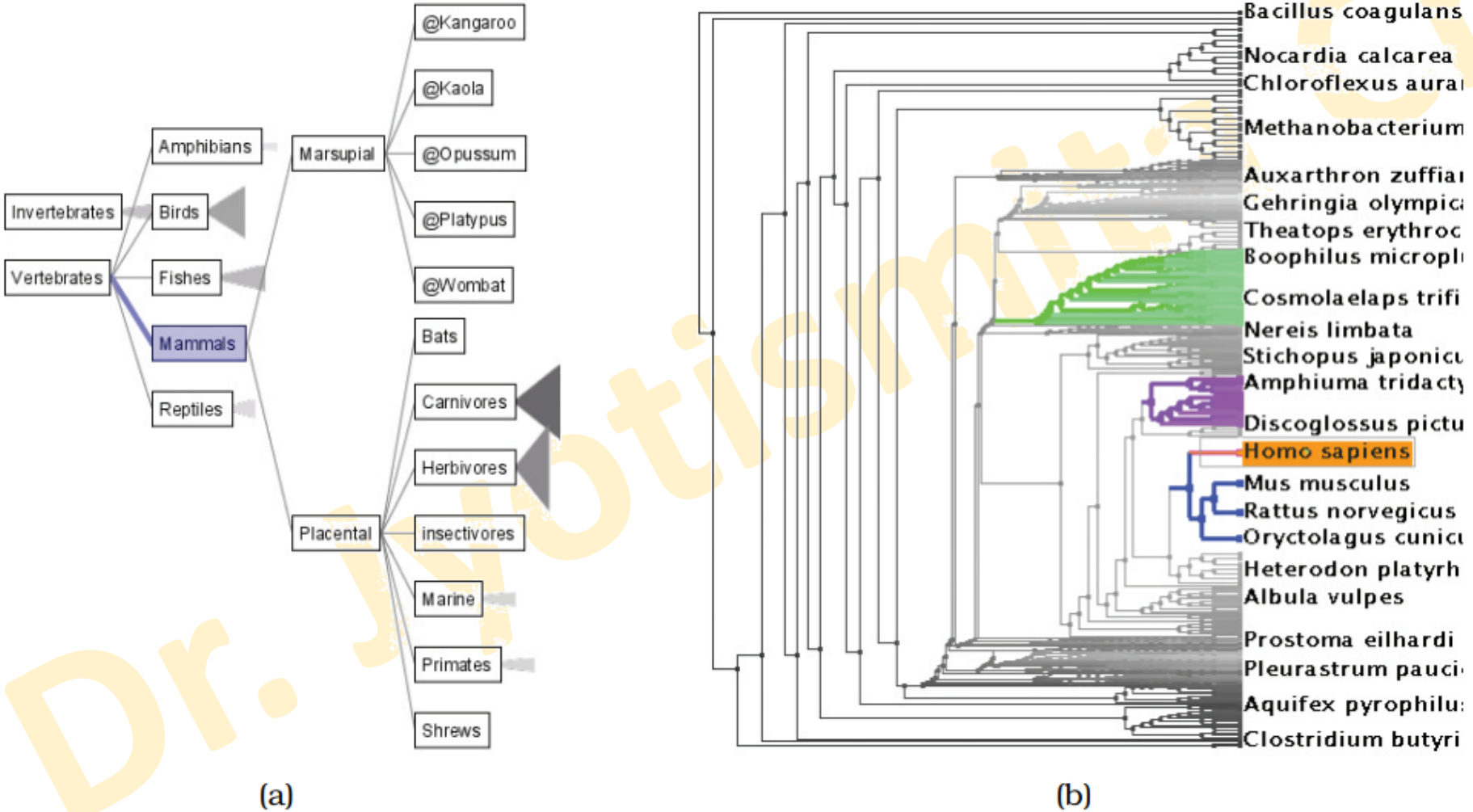
- Domain situation –
  - who are the target users?
- Abstraction – translate from specifics of domain to vocabulary of vis
  - **what** data the user sees? **data abstraction**
  - **why** the user intends to use a vis tool? **task abstraction**
- Idiom – **how** the visual encoding and interaction idioms are constructed in terms of design choices?
  - **visual encoding idiom**: how to draw
  - **interaction idiom**: how to manipulate



Comparing Two Idioms

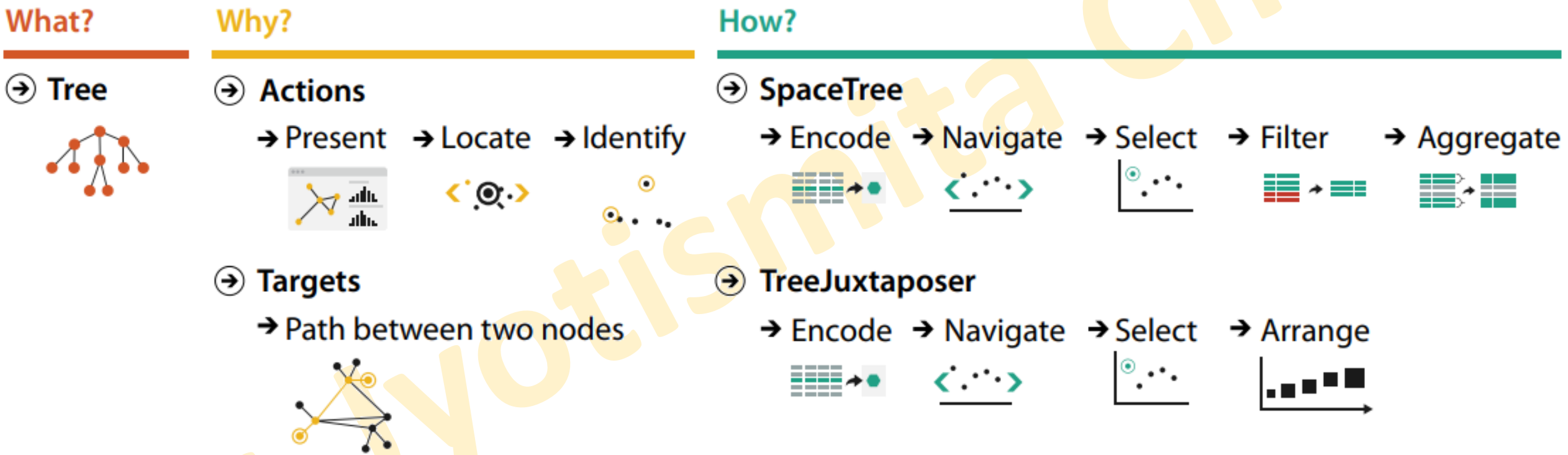
- The what–why–how analysis framework is useful for comparative analysis, for example, to examine two different vis tools that have different answers for the question of how the idiom is designed when used for exactly the same context of why and what at the abstraction level.
- **What** these tools take as input data is the same: a large tree composed of nodes and links.
- **Why** these tools are being used is for the same goal in this scenario: to present a path traced between two nodes of interest to a colleague.
- The systems differ in **how** elements of the visualization are manipulated and arranged.
  - SpaceTree ties the act of selection to a change of what is shown by automatically aggregating and filtering the unselected items.
  - In contrast, TreeJuxtaposer allows the user to arrange areas of the tree to ensure visibility for areas of interest.

Comparing Two Idioms



Comparing two idioms. (a) SpaceTree. (b) TreeJuxtaposer.

Comparing Two Idioms

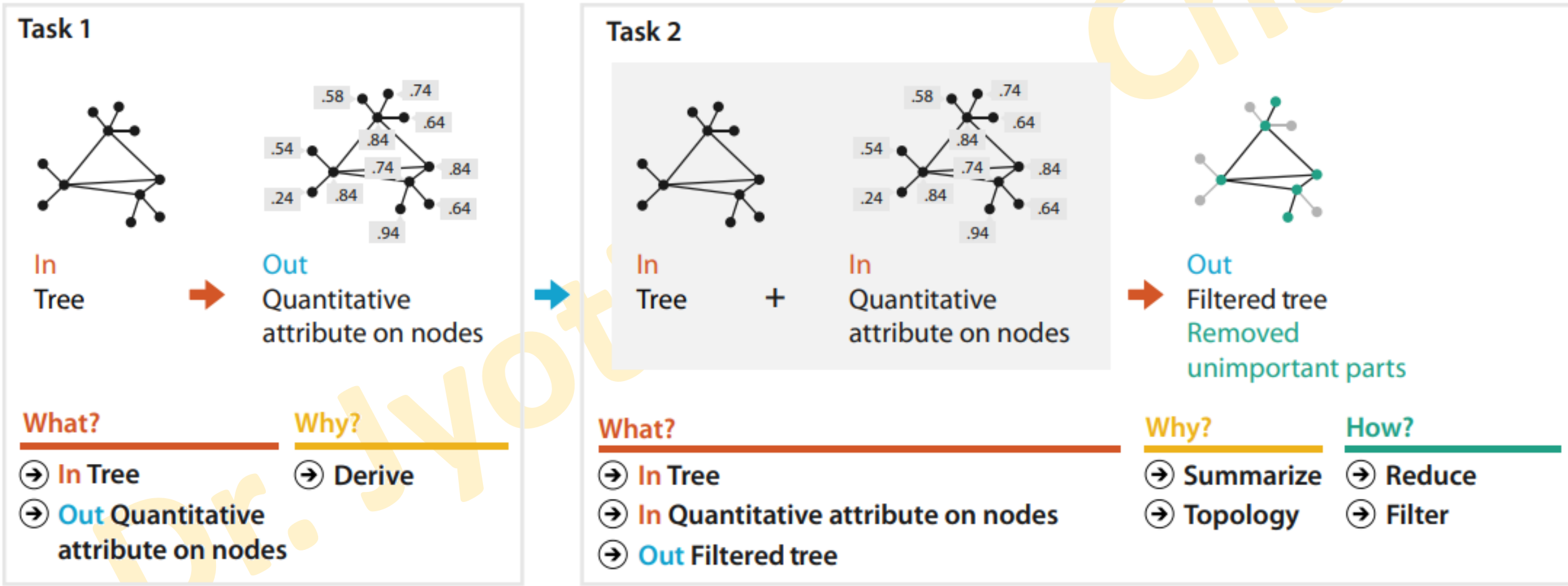


Analyzing what–why–how comparatively for the SpaceTree and TreeJuxtaposer idioms

Deriving One Attribute

- In a vis showing a complex network or tree, it is useful to be able to filter out most of the complexity by drawing a simpler picture that communicates the key aspects of its topological structure.
- One way to support this kind of summarization is to calculate a new derived attribute that measures the importance of each node in the graph and filter based on that attribute.

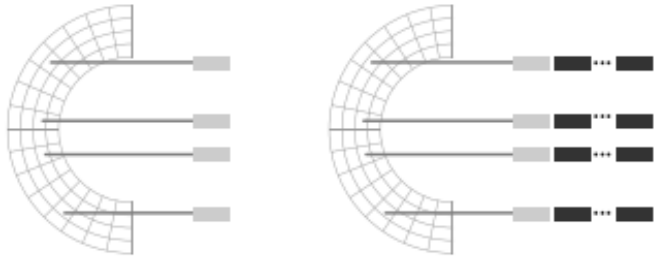
Deriving One Attribute



Analyzing a chained sequence of two instances where an attribute is derived in order to summarize a tree by filtering away the unimportant parts.

Deriving Many New Attributes

Task 1



In
Spatial field

Out
Many quantitative attributes

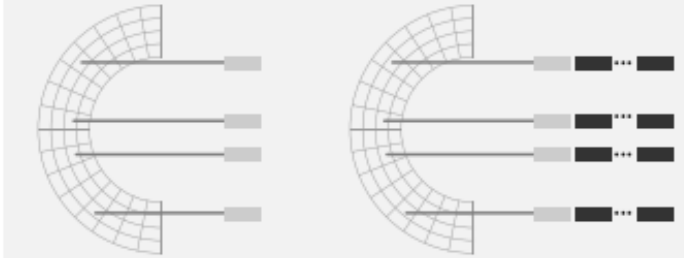
What?

- In Spatial field
- Out Many quantitative attributes

Why?

- Derive

Task 2



In
Spatial field + Many quantitative attributes

Out
Juxtaposed attribute plots with linked coloring

What?

- In Spatial field
- In Many quantitative attributes
- Out Juxtaposed attribute plots with linked coloring

Why?

- Actions
  - Discover
  - Explore
  - Browse
  - Identify
  - Compare
- Targets
  - Features

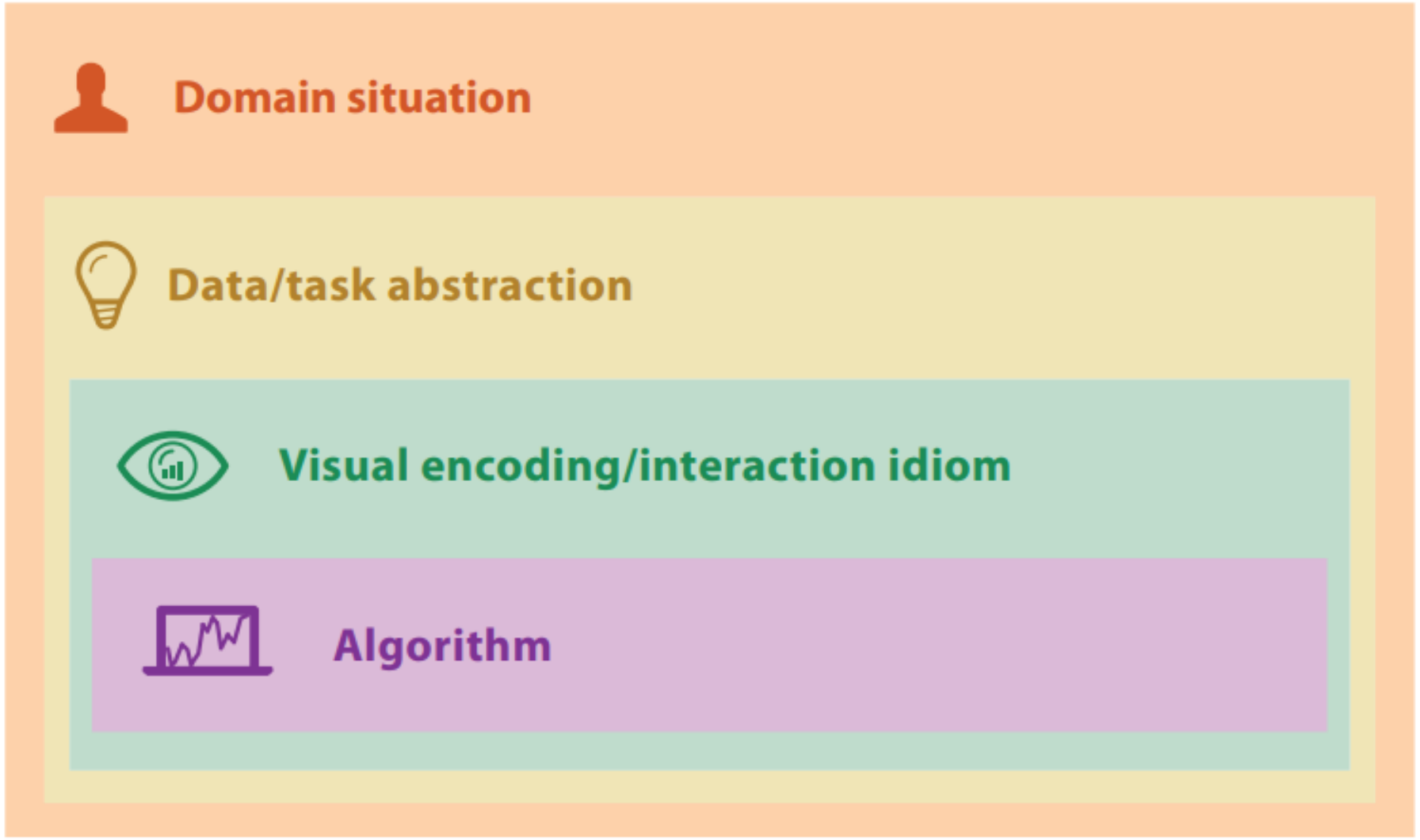
How?

- Map
  - Hue
- Arrange
  - Express
- Facet
  - Juxtapose
  - Partition
- Manipulate
  - Select
  - Navigate

Analysis: Four Levels of Validation: Why validation?

- The problem of validation for a vis design is difficult because there are so many questions that you could ask when considering whether a vis tool has met your design goals. Some of these are:
  - How do you know if it works?
  - How do you argue that one design is better or worse than another for the intended users?
  - For one thing, what does better mean?
  - Do users get something done faster?
  - Do they have more fun doing it?
  - Can they work more effectively?
  - What does effectively mean?
  - What is the design better than?
  - Is it better than another vis system?
  - Is it better than doing the same things manually, without visual support?
  - Is it better than doing the same things completely automatically?
  - How do you decide what sort of benchmark data you should use when testing the system?

Analysis: Four Levels of Validation



The four nested levels of vis design

Analysis: Four Levels of Validation: Domain Situation

- **Domain situation** encompasses a group of target users, their domain of interest, their questions, and their data.
- The term **domain** is frequently used in the vis literature to mean a particular field of interest of the target users of a vis tool.
- Each domain usually has its own vocabulary for describing its data and problems, and there is usually some existing workflow of how the data is used to solve their problems.
- The methods typically used by designers to identify domain situation blocks include interviews, observations, or careful research about target users within a specific domain.

Analysis: Four Levels of Validation: Task and Data Abstraction

- Design at the next level requires abstracting the specific domain questions and data from the domain-specific form.
- Abstracting into the domain-independent vocabulary allows you to realize how domain situation blocks using very different language might have similar reasons why the user needs the vis tool and what data it shows.
- Questions from very different domain situations can map to the same abstract vis tasks.
- The data abstraction level requires you to consider whether and how the same dataset provided by a user should be transformed into another form.

Analysis: Four Levels of Validation: Task and Data Abstraction

- Many vis idioms are specific to a particular data type, such as a table of numbers where the columns contain quantitative, ordered, or categorical data; a node–link graph or tree.
- Your goal is to determine which data type would support a visual representation of it that addresses the user's problem.
- Although sometimes the original form of the dataset is a good match for a visual encoding that solves the problem, often a transformation to another data type provides a better solution.

Analysis: Four Levels of Validation: Visual Encoding and Interaction Idiom

- At the third level, you decide on the specific way to create and manipulate the visual representation of the abstract data block that you chose at the previous level.
- There are two major concerns at play with idiom design.
- One set of design choices covers how to create a single picture of the data: the **visual encoding** idiom controls exactly what users see.
- Another set of questions involves how to manipulate that representation dynamically: the **interaction** idiom controls how users change what they see.

Analysis: Four Levels of Validation: Visual Encoding and Interaction Idiom

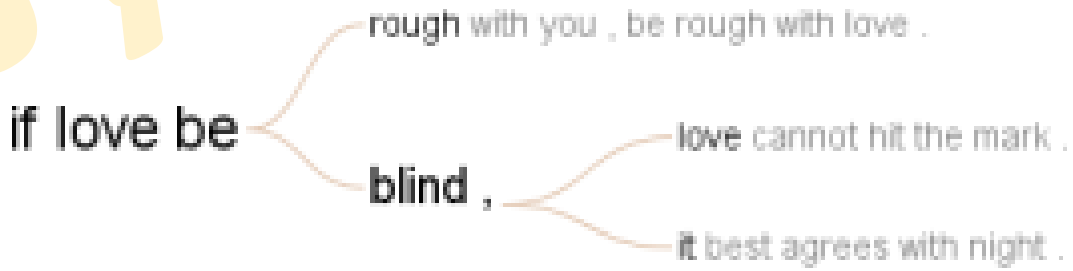
- For Example: Word tree created from Romeo and Juliet
  - All instances of “if love” in Romeo and Juliet

if love be rough with you , be rough with love

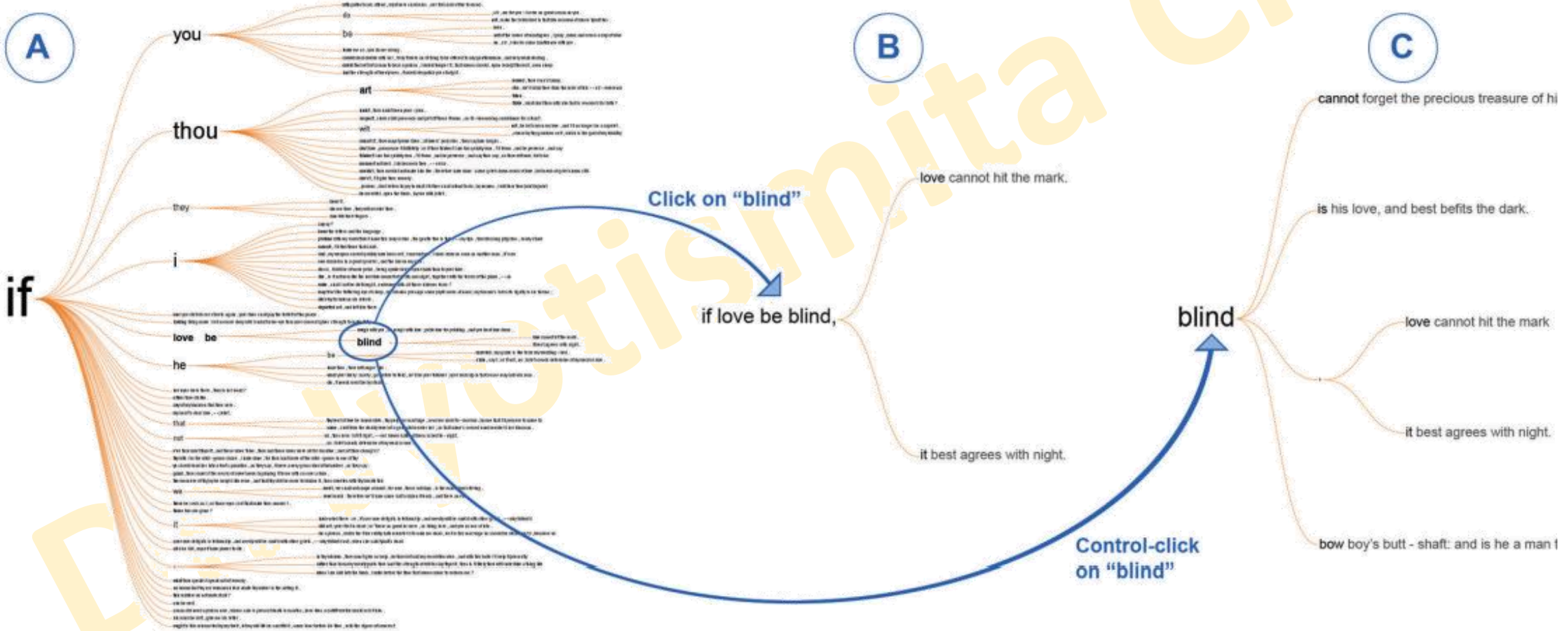
if love be blind , love cannot hit the mark .

if love be blind , it best agrees with night .

  - Word tree showing all instances of “if love” in Romeo and Juliet.



Analysis: Four Levels of Validation: Visual Encoding and Interaction Idiom



Analysis: Four Levels of Validation: Visual Encoding and Interaction Idiom

- Once a word tree is shown, a user can interact with it.
- Moving the mouse over a particular word or phrase brings up additional information, along with a message saying that clicking will explore the tree further.
- Clicking on an individual word will redefine the phrase shown by the tree. This can either narrow or widen the text search.
- For instance, if the current phrase is “if love,” clicking on the initial “if” will re-center the tree on the phrase “if” (see Fig. A).
- On the other hand, if the user clicks on a word in a branch of the tree, such as “blind” in the branch “if love be blind,” then the tree will be re-centered on the longer phrase, “if love be blind” (Fig. B).
- Often when looking at a tree, a user will see an unexpected or interesting word in a branch, and may want to see all uses of that word.
- To support this goal, a second click option is control-click, which recenters the tree on the single word clicked, no matter where it occurs in the tree. Control-clicking on “blind” in the example above will display a tree that is rooted on the word “blind” (Fig. C).

Analysis: Four Levels of Validation: Algorithm

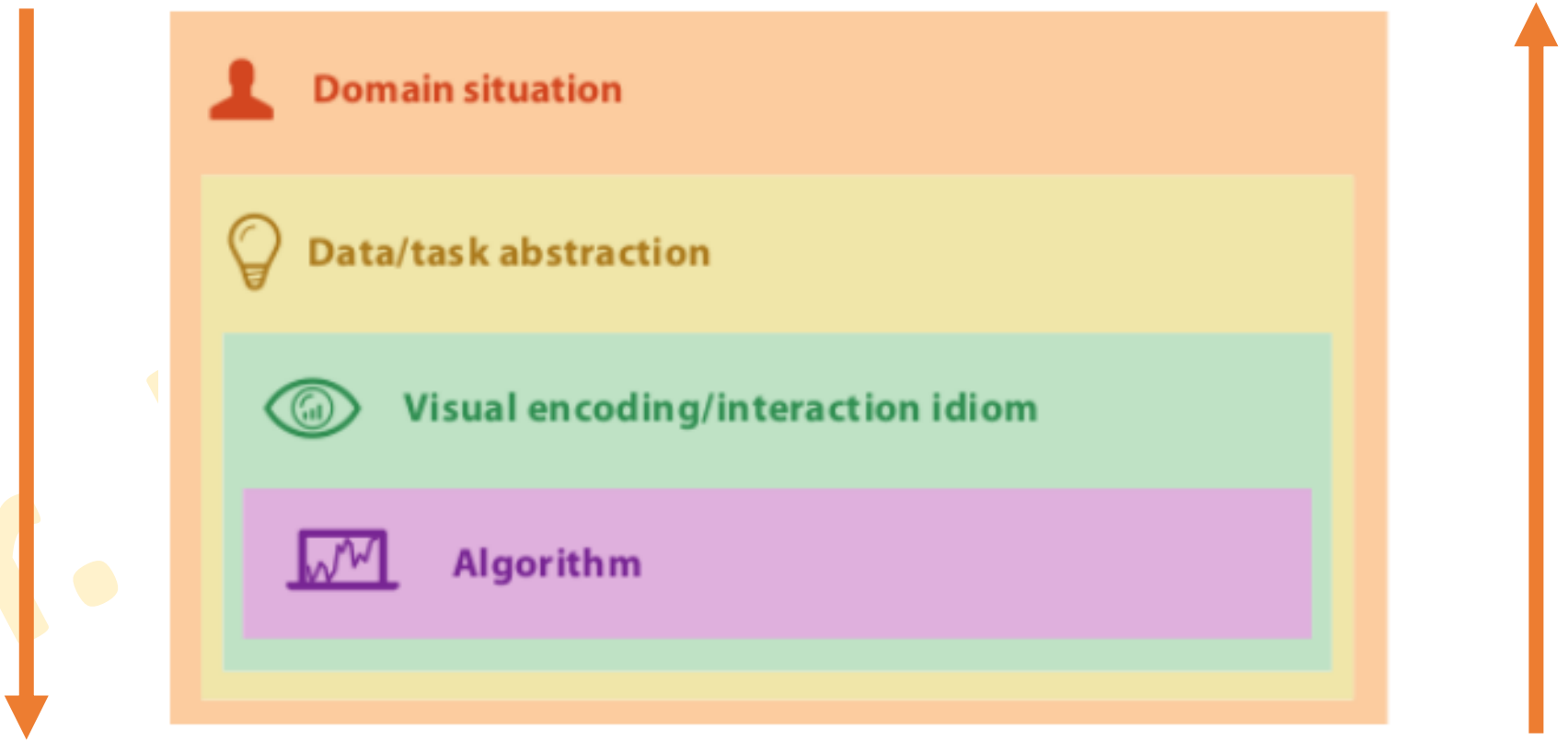
- The innermost level involves all of the design choices involved in creating an algorithm: a detailed procedure that allows a computer to automatically carry out a desired goal.
- In this case, the goal is to efficiently handle the visual encoding and interaction idioms that you chose in the previous level.
- Algorithm blocks are also designed, rather than just identified.
- You could design many different algorithms to instantiate the same idiom.
- For example,
  - One visual encoding idiom for creating images from a three-dimensional field of measurements, such as scans created for medical purposes with magnetic resonance imaging, is direct volume rendering.

Analysis: Four Levels of Validation: Algorithm

- Many different algorithms have been proposed as ways to achieve the requirements of this idiom, including ray casting, splatting, and texture mapping.
- You might determine that some of these are better than others according to measures such as the speed of the computation, how much computer memory is required, and whether the resulting image is an exact match with the specified visual encoding idiom or just an approximation.
- The nested model emphasizes separating algorithm design, where your primary concerns are about computational issues, from idiom design, where your primary concerns are about human perceptual issues.

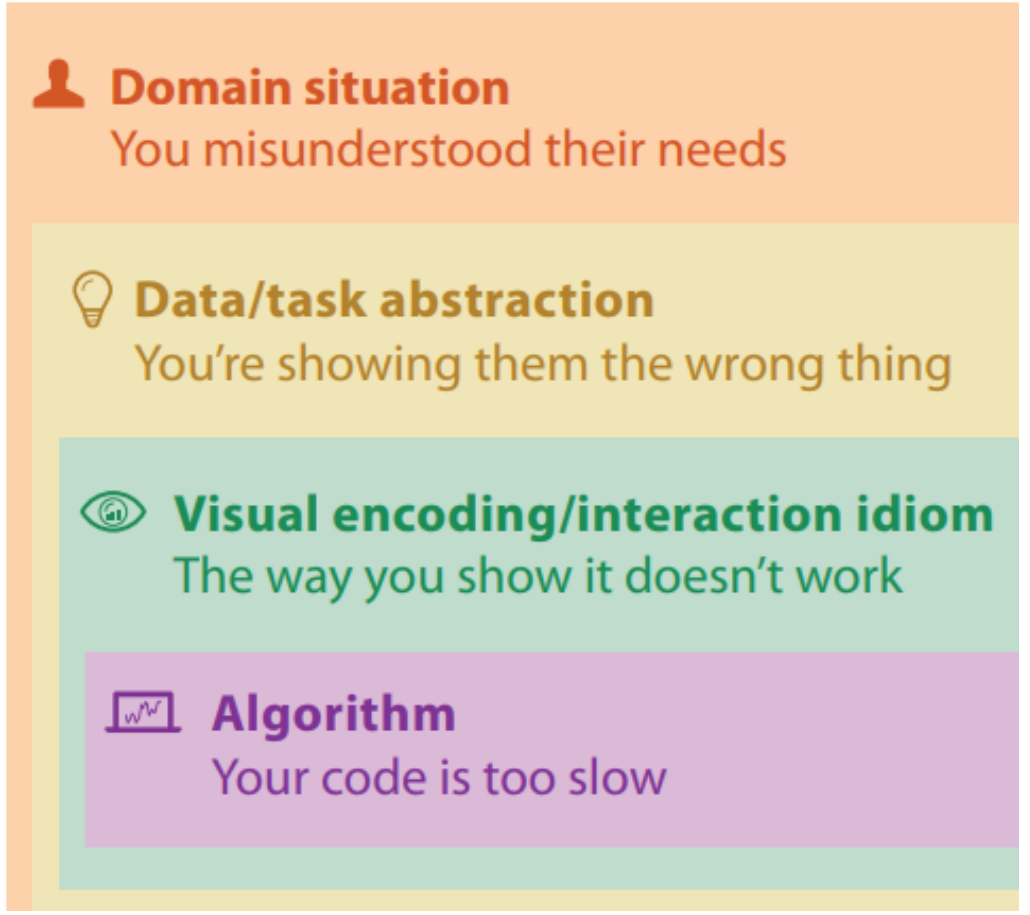
Analysis: Four Levels of Validation: Angles of Attack

- There are two common angles of attack for vis design: **top down** (problem-driven) or **bottom up** (technique-driven).



Analysis: Four Levels of Validation: Threats to Validity

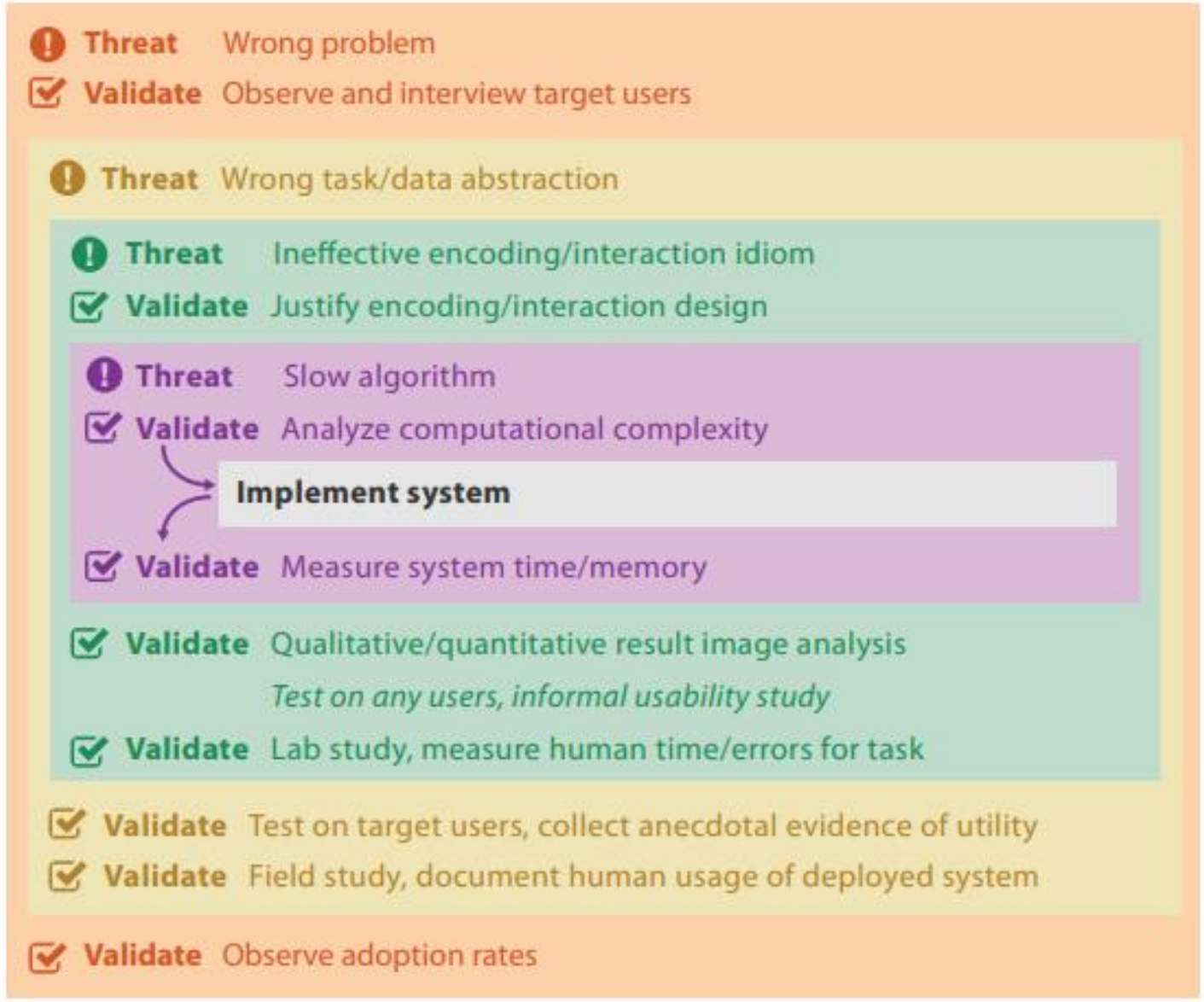
- Each of the four levels has a different set of threats to validity: that is, different fundamental reasons why you might have made the wrong choices.
- In the figure, **they** means the target users and **you** means the vis designer



The four nested levels of vis design have different threats to validity at each level.

Analysis: Four Levels of Validation: Validation Approaches

- Different threats require very different approaches to validation.
- Figure shows a summary of the threats and validation approaches possible at each level.



Analysis: Four Levels of Validation: Validation Approaches: Domain Validation

- The primary threat is that the problem is mischaracterized: the target users do not in fact have these problems.
- An immediate form of validation is to interview and observe the target audience to verify the characterization, as opposed to relying on assumptions or conjectures.
- A common approach for this case is a field study, where the investigator observes how people act in real-world settings, rather than by bringing them into a laboratory setting.
- Field studies for domain situation assessment often involve gathering qualitative data through semi-structured interviews.
- The method of contextual inquiry, where the researcher observes users working in their real-world context and interrupts to ask questions when clarification is needed, is typically better suited for vis designers than silent observation because of the complex cognitive tasks that are targeted.

Analysis: Four Levels of Validation: Validation Approaches: Abstraction Validation

- At the abstraction level, the threat is that the identified task abstraction blocks and designed data abstraction blocks do not solve the characterized problems of the target audience.
- The key aspect of validation against this threat is that the system must be tested by target users doing their own work, rather than doing an abstract task specified by the designers of the vis system.
- A more rigorous validation approach for this level is to conduct a field study to observe and document how the target audience uses the deployed system, again as part of their real-world workflow.
- A common downstream form of validation is to have a member of the target user community try the tool, in hopes of collecting informal evidence that the tool is in fact useful.
- These informal evidences may have the form of insights found or hypotheses confirmed.
- Of course, this observation cannot be made until after all three of the other levels have been fully addressed, after the algorithm designed at the innermost level is implemented.

Analysis: Four Levels of Validation: Validation Approaches: Idiom Validation

- At the visual encoding and interaction idiom level, the threat is that the chosen idioms are not effective at communicating the desired abstraction to the person using the system.
- One immediate validation approach is to carefully justify the design of the idiom with respect to known perceptual and cognitive principles.
- A downstream approach to validate against this threat is to carry out a **lab study**: a controlled experiment in a laboratory setting.
- This method is appropriate for teasing out the impact of specific idiom design choices by measuring human performance on abstract tasks that were chosen by the study designers.
- Another downstream validation approach is the presentation of results in the form of still images or video.
- A third appropriate form of downstream validation is the quantitative measurement of result images created by the implemented system; these are often called quality metrics. For example, many measurable layout metrics such as number of edge crossings and edge bends have been proposed to assess drawings of node–link networks.

Analysis: Four Levels of Validation: Validation Approaches: Algorithm Validation

- At the algorithm level, the primary threat is that the algorithm is suboptimal in terms of time or memory performance, either to a theoretical minimum or in comparison with previously proposed algorithms.
- Obviously, poor time performance is a problem if the user expects the system to respond in milliseconds but instead the operation takes hours or days.
- An immediate form of validation is to analyze the computational complexity of the algorithm, using the standard approaches from the computer science literature.
- While many designers analyze algorithm complexity in terms of the number of items in the dataset, in some cases it will be more appropriate to consider the number of pixels in the display.
- Another threat is incorrectness at the algorithm level, where the implementation does not meet the specification from the idiom level.

Visualization Techniques

Dr. Jyotismita Chaki

Scalar Techniques

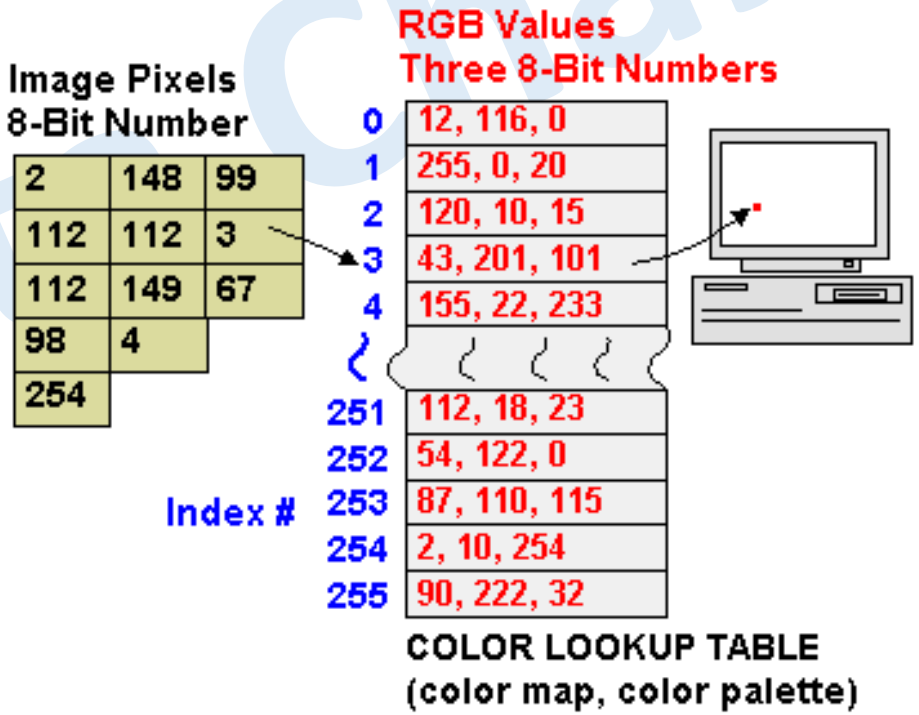
- Visualizing scalar data is frequently encountered in science, engineering, and medicine, but also in daily life.
- There exist many scalar visualization techniques, both for two-dimensional (2D) and three-dimensional (3D) datasets.
- The most popular scalar visualization techniques are: color mapping, contouring, and height plots.

Color Maps

- Color mapping is probably the most widespread visualization method for scalar data.
- Color mapping is a mapping function m : $D \rightarrow D_V$. The geometry of D_V is the same as D, but with a color that depends on the scalar data defined on D.
- Color mapping is not concerned with creating specific shapes to visualize data, but with coloring such shapes on which scalar data is defined to show the data values.
- There are several ways to define such a scalar-to-color function

Color Maps: Color look-up tables

- Color look-up tables are the simplest way to implement color mapping.
- A matrix of color data that is searched in order to change a source set of colors to a destination set.
- Color lookup tables (CLUTs) are found in graphics cards (display adapters) in order to translate the colors in an image to the colors in the hardware.
- They are also found in graphics formats, such as GIFs.



The pixels in the image contain index numbers that point to the RGB value in the color lookup table. The RGB values are the ones used by the display system.

Color Maps: Color look-up tables

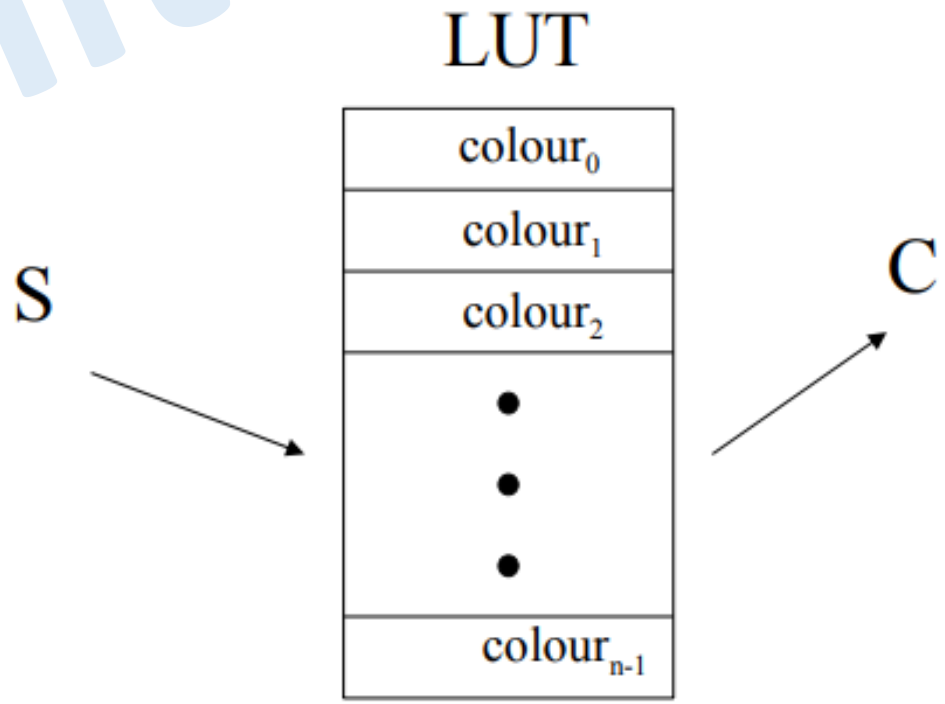
- A color look-up table C, also called a colormap, is a uniform sampling of the color-mapping function c:

$$C = \{c_i\}_{i=1..N}, \qquad \text{where } c_i = c\left(\frac{(N-i)f_{\min} + if_{\max}}{N}\right)$$

- This equation is implemented as a table of N colors $c_1,\ldots,c_N$, which are associated with the scalar dataset values f, assumed to be in the range $[f_{min}, f_{max}]$.
- Knowing the scalar range is important, as it allows us to construct a color mapping with a clear and simple meaning:
  - the colors c_i with low indices i in the colormap represent low scalar values close to f_{min}, whereas colors with indices close to N in the colormap represent high scalar values close to f_{max}.

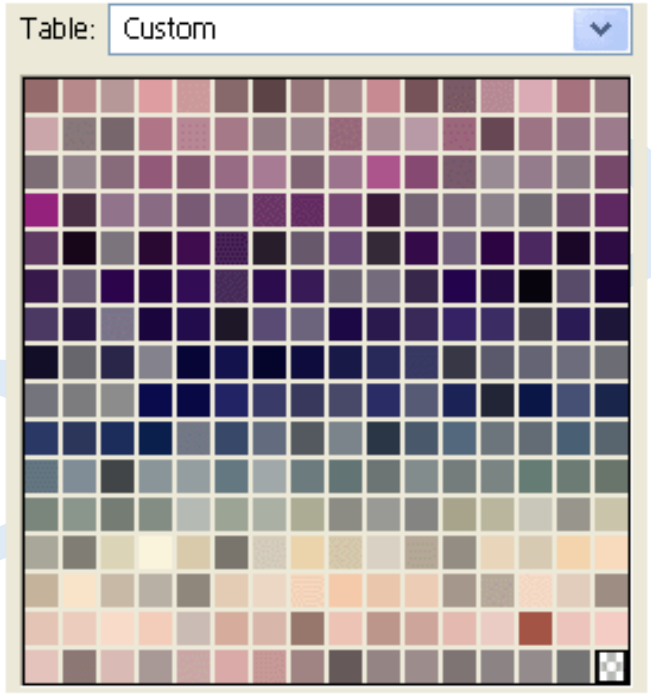
Color Maps: Color look-up tables

- Assume –
  - scalar values S_i in range {min→max}
  - n unique colours, {colour 0 ... colour n-1 } in Look-Up-Table (LUT)
- Define mapped colour C:
  - if S< min then C = $colour_{min}$
  - if S> max then C = $colour_{max}$
  - Else
    - For (j = 0; j < n ; j++)
      - if (Cj min < S < Cj max) C = Cj



Color Maps: Color look-up tables

- **24-Bit vs. 8-Bit:** Indexed color images can look nearly identical to their 24-bit originals, because the 256 most frequently used colors are identified. The 256-color palette (left) was created in Photoshop from the original 24-bit image (middle) to generate the image as 8 bits (right).

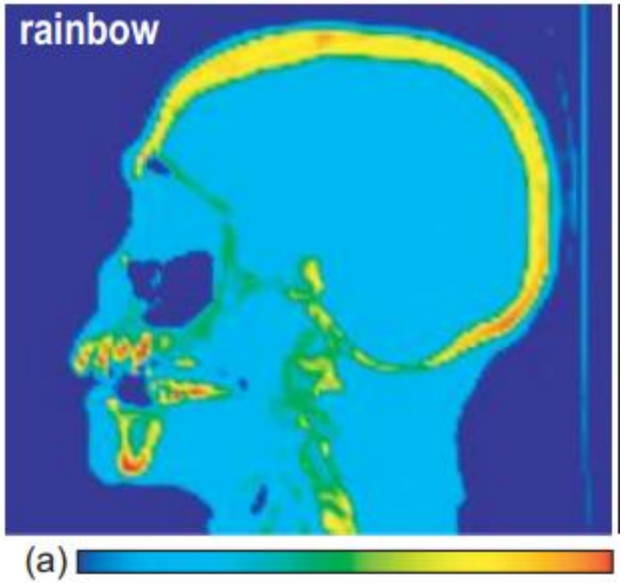


16.8 Million Colors

256 Colors

Color Maps: Rainbow colormap

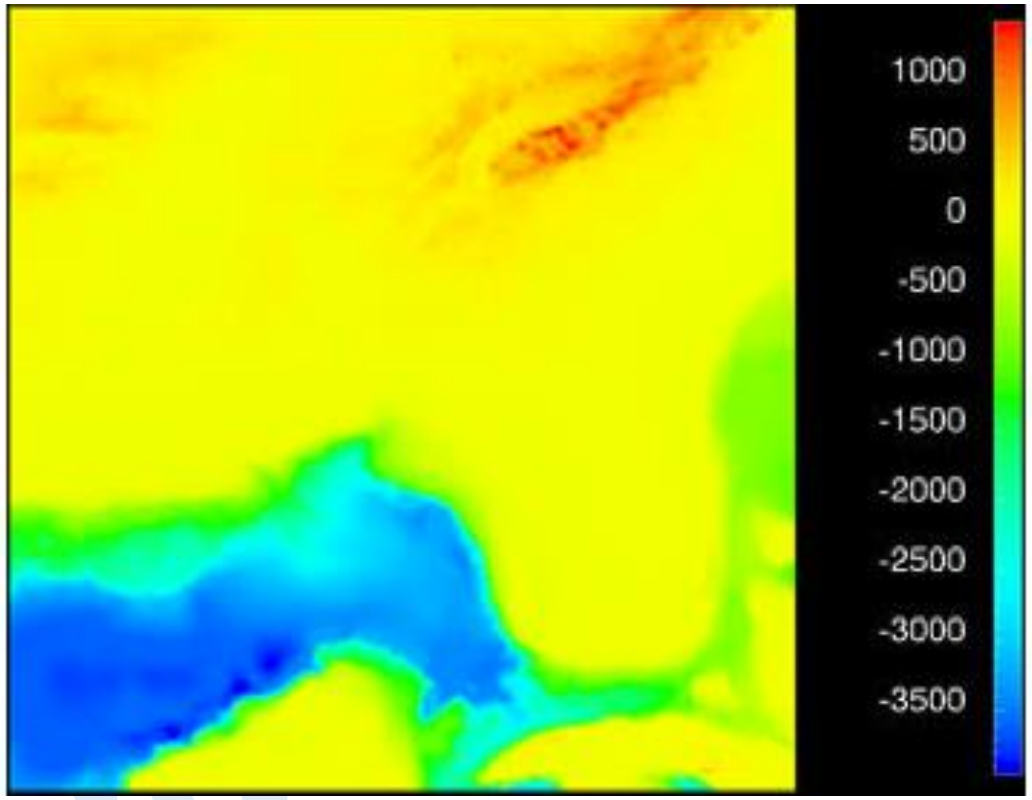
- The rainbow color map is named that way because it goes through all the rainbow's colors.
- The lower values are in the deep blue range and the higher values in the reds. In between it passes trough light blue green, yellow, orange…
- It is used as a default in many visualization systems since it is easy to calculate (it is a linear interpolation between (0,0,255) and (255,0,0) in RGB color space), and because the bright colors are visually appealing.

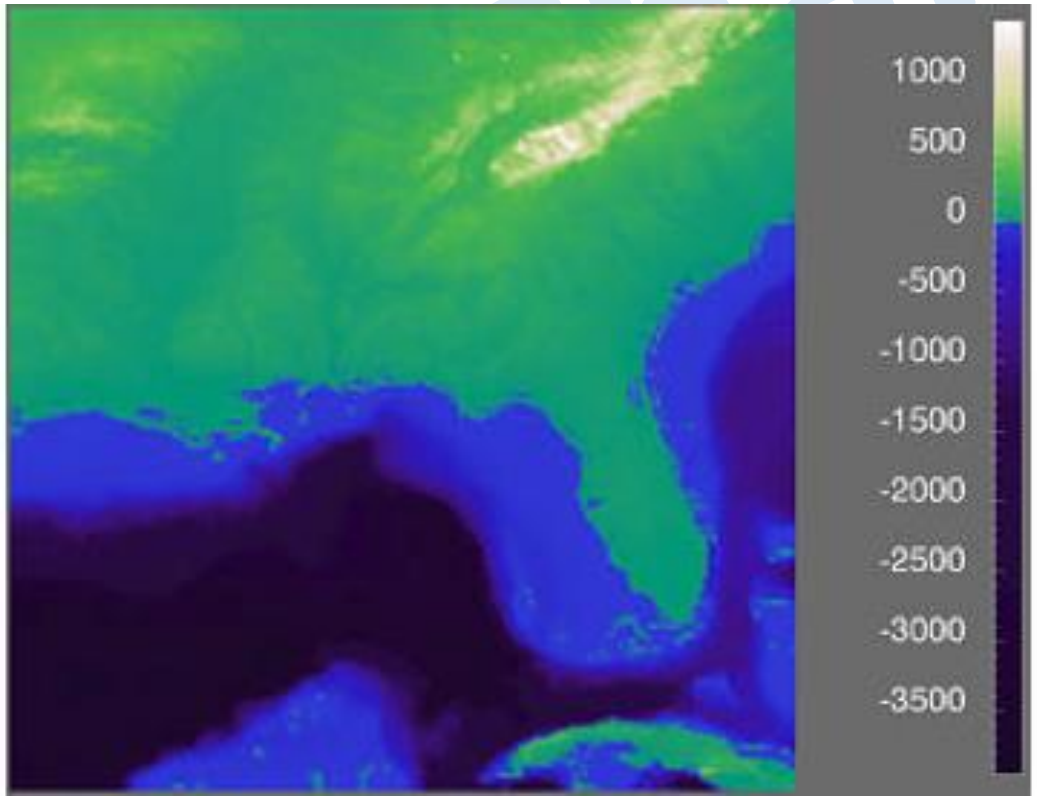


Color Maps: Rainbow colormap

- As it is explain in several visualisation articles the rainbow color map can be misleading when visualising data. The main issues with this kind of color map are:
  - structures in the data can be hidden, since not all data variations are represented visually,
  - the fact the luminance (how much white or black is mixed into the color. Higher the luminance, the brighter your color will be and as you lower your luminance, the color will appear to be darker) is not controlled can hide data,
  - it introduces gradients (a directional change in the intensity or color in an image) not related to the data,
  - it artificially divides the data into a small number of categories, one for each color.
- Despite these problems the color map is still the most widely used in pseudo-color representations.

Color Maps: Rainbow colormap

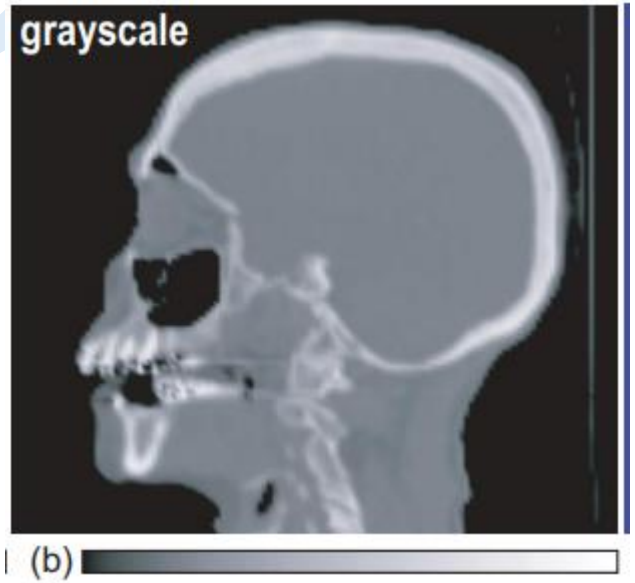




The one on left uses a rainbow color map whereas the one on right uses a more appropriate map. The rainbow map hides the well known structure of the Florida coast line. In many cases, an analyst examines a visualization to discover features. Using the Rainbow colormap can distort the perception of these features and be very misleading

Color Maps: Grayscale colormap

- Here, we map data values f linearly to luminance, or gray value, with f_{min} corresponding to black and f_{max} corresponding to white.
- Most medical specialists, but also nonspecialists, would agree that the grayscale produces a much easier-to-follow, less-confusing visualization on which details are easier to spot than when using the rainbow colormap.
- The grayscale colormap has several advantages.
  - First, it directly encodes data into luminance, and thus is has no issues regarding the discrimination of different hues.
  - Second, color ordering is natural (from dark to bright). For medical visualizations, for example, the black-to-white colormap can be more effective than the rainbow one, as these grayscales map intuitively to the ones visible in X-ray photographs.
  - Finally, rendering grayscale images is less sensitive to color reproduction variability issues.

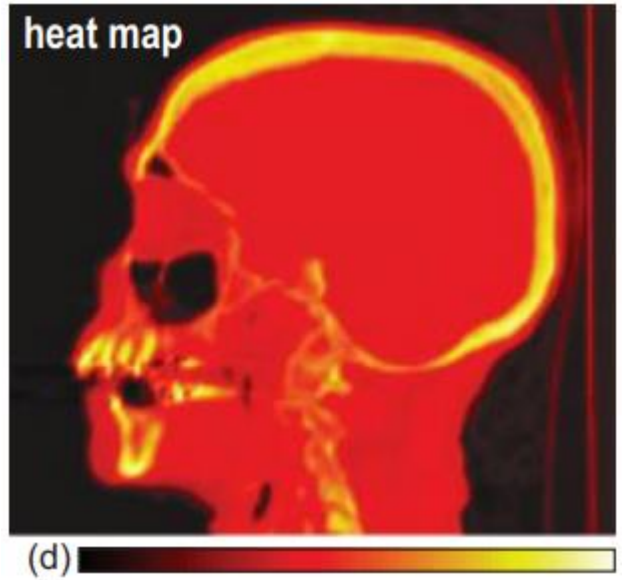


Color Maps: Two Hue colormap

- The colormap entries are obtained by linearly interpolating between two user-selected colors, blue and yellow in our case.
- The two-hue colormap can be seen as a generalization of the grayscale colormap, where we interpolate between two colors, rather than between black and white.
- If the two colors used for interpolation are perceptually quite different (such as in our case, where they differ both in hue but also in perceived luminance), the resulting colormap allows an easy color ordering, and also produces a perceptually more linear result than the rainbow colormap.
- Also, in case none of the two colors that define this colormap is too bright, the result is suitable to be used for non-compact data displayed on a white background.
- However, a disadvantage of this type of colormap is that it arguably offers less dynamic range: In general, we can distinguish between more hues (such as in the case of the rainbow colormap) than between an equal number of mixes of the same two hues.

Color Maps: Heatmap colormap

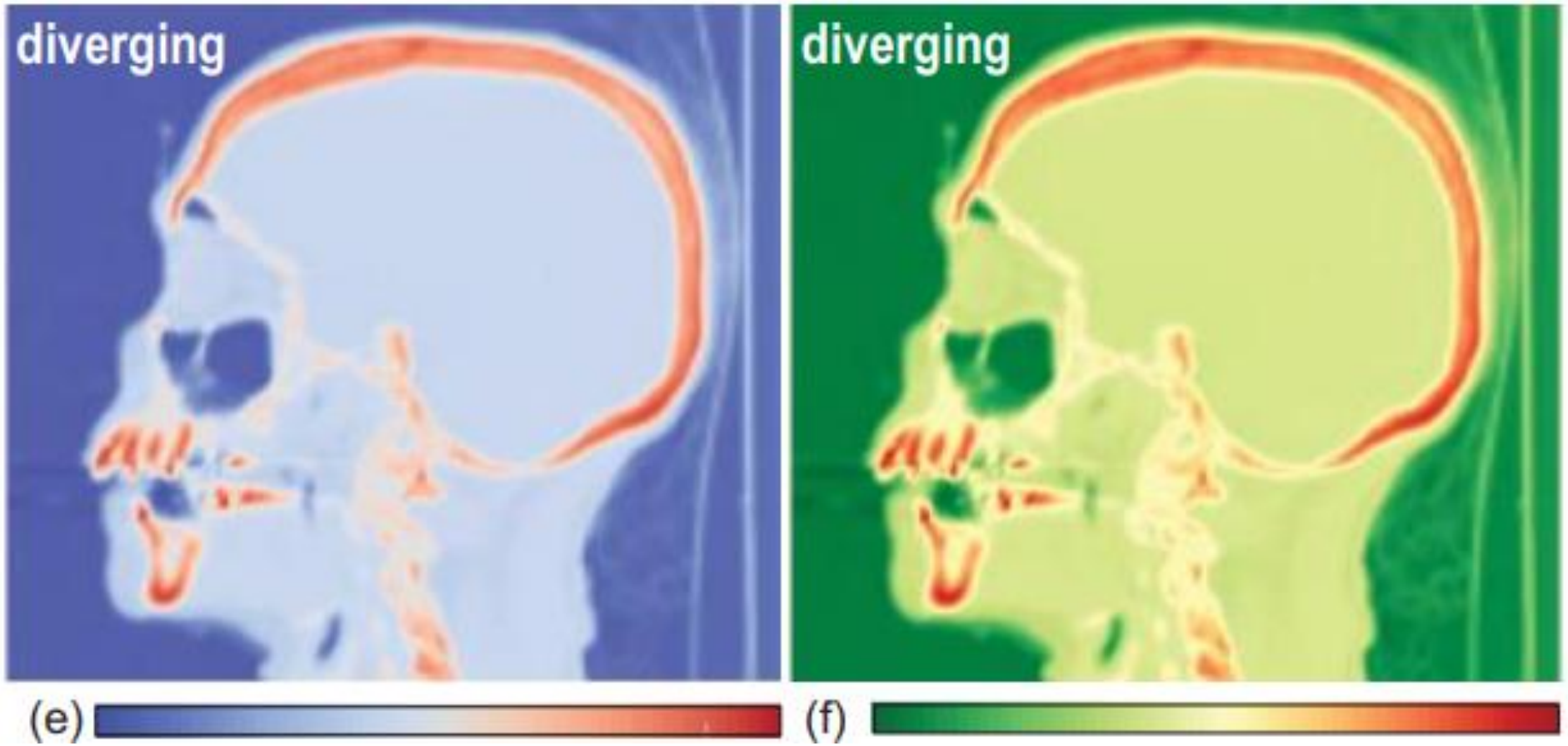
- The intuition behind its colors is that they represent the color of an object heated at increasing temperature values, with black corresponding to low data values, red-orange hues for intermediate data ranges, and yellow-white hues for the high data values respectively.
- Compared to the rainbow colormap, the heat map uses a smaller set of hues, but adds luminance as a way to order colors in an intuitive manner.
- Compared to the two-hue colormap, the heat map uses more hues, thus allowing one to discriminate between more data values.
- Eliminating the right end of the colormap (thus, using yellow rather than white for the highest data value) allows using this colormap also for non-compact domains displayed on a white background.
- Together with the grayscale map, the heat map is a popular choice for medical data visualization



Color Maps: Diverging colormap

- Diverging colormaps are constructed starting from two hues.
- However, rather than interpolating between the end colors c_{min} and c_{max}, we now add a third color c_{mid} for the data value $f_{mid} = (f_{min} + f_{max})/2$ located in the middle of the considered data range $[f_{min}, f_{max}]$, and use two piecewise-linear interpolations between c_{min} and c_{mid} and between c_{mid} and c_{max}, respectively.
- Additionally, c_{mid} is chosen so that it has a considerably higher luminance than c_{min} and c_{max}.

Color Maps: Diverging colormap



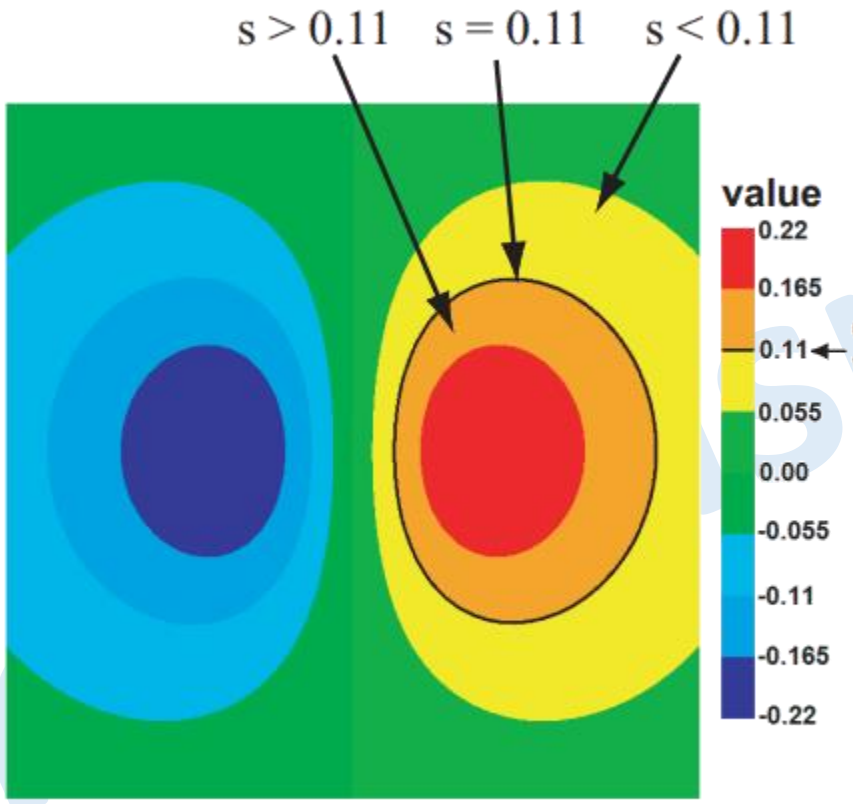
c_{min} = blue, c_{max} = red, and c_{mid} = white.

c_{min} = green, c_{max} = red, and c_{mid} = brightyellow

Contouring

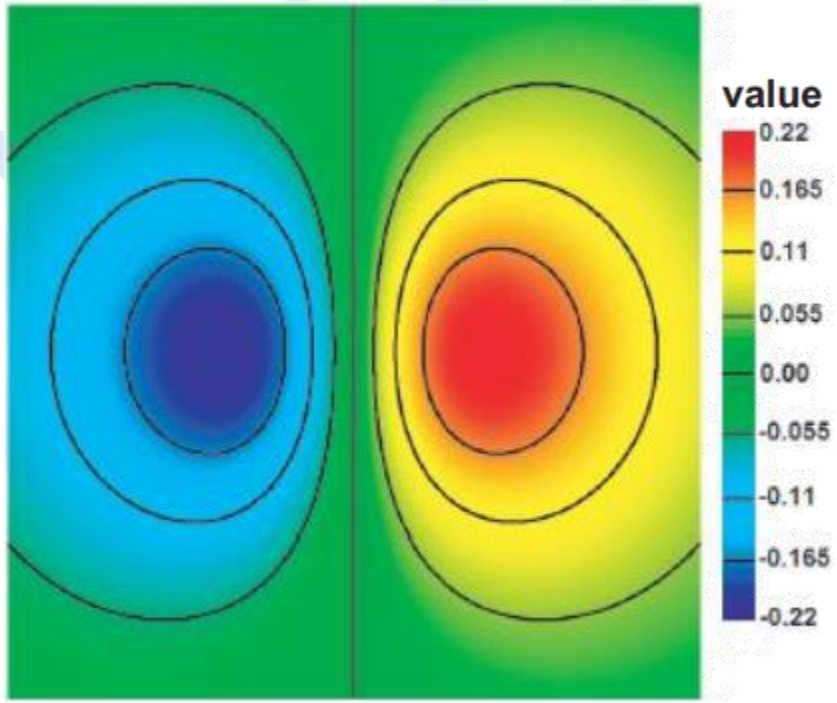
- To understand contouring, think of the meaning of the sharp color transitions that separate the color bands.
- Consider, for instance, the transition between the yellow and orange bands; that is, all points in the figure that are on the border separating these two colors.
- As can be seen from the associated color legend, points in the yellow band have scalar values s below 0.11, whereas the points in the orange band have scalar values s above 0.11.
- Hence, the points located on the color border itself have the scalar value s = 0.11.
- For this reasoning to hold, we must assume that our dataset does not exhibit a sudden “jump”.
- This holds for all datasets that represent the sampling of a continuous signal.
- Points located on such a color border, drawn in black in Figure, are called a contour line, or isoline.

Contouring



(a) one contour (s=0.11)

Contouring and Color Banding



(b) 7 contours (s ∈ [-0.165,0.165])

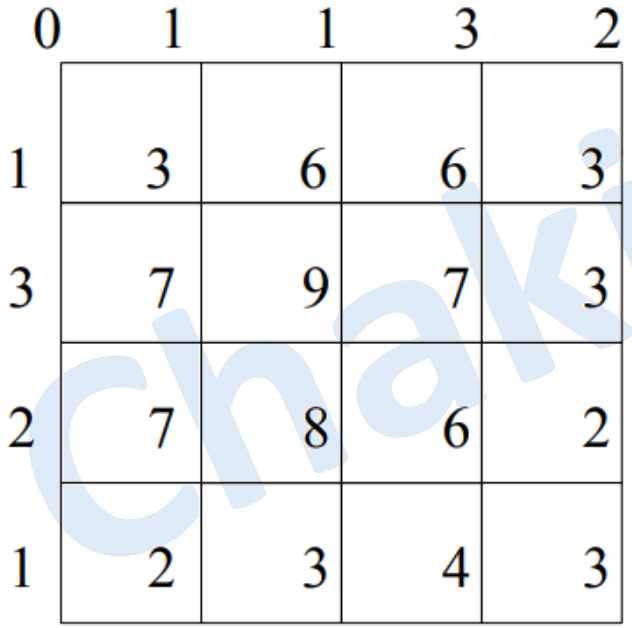
Contouring and Colormapping

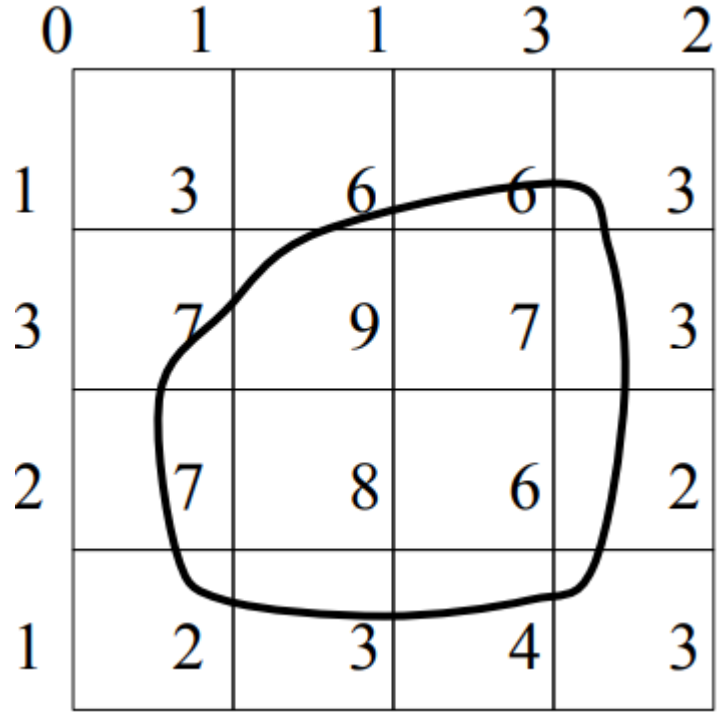
Contouring

- Formally, a contour line C is defined as all points p in a dataset D that have the same scalar value, or isovalue s(p) = x:
  - $C(x) = \{p \in D | s(p) = x\}$.
- Contour lines are drawn on land maps to explicitly indicate all points that have the same altitude.
- For 2D dataset, a contour line is called an **isoline**.
- For 3D datasets, contours are 2D surfaces called **isosurfaces**.

Contouring: Computation: Edge Tracking

- Data : 2D structured grid of scalar values (top)
- Difficult to visualise transitions in data
  - – use contour at specific scalar value to highlight transition
- Approach 1: Edge Tracking
  - Select scalar value
    - – corresponds to contour line – i.e. contour value, e.g. 5
  - Interpolate contour line through the grid corresponding to this value (bottom)
  - Advantages : produces correctly shaped line
  - Dis-advantages : need to search for other contours





Contour: Computation: Edge Tracking

- How can we compute contours, given a discrete, sampled dataset D?
- Since this dataset is defined as a set of cells carrying node or cell scalar data, plus additional basis functions, it is natural to try to construct contours in the same discrete cell space.
- In the actual construction technique, we shall use an important property of isolines.
- The basic algorithm for constructing an isoline is quite simple.
- The principle is illustrated by the simple 5 × 5 cell grid in Figure, where we construct the isoline for the value v = 0.48.

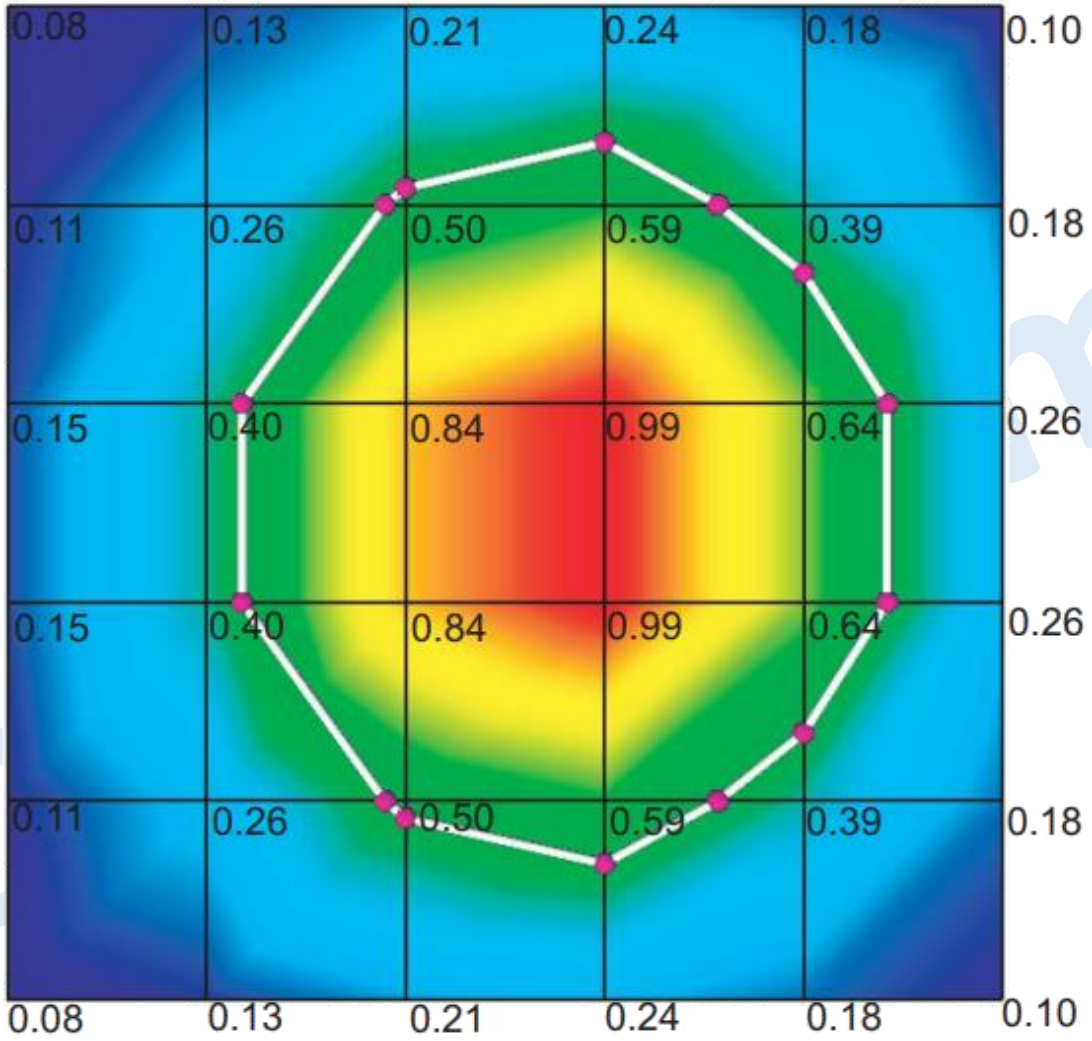
Contour: Computation: Edge Tracking

- For every cell c of the dataset, we test whether the isoline intersects the respective cell, as follows.
  - For every edge e = (pi, pj) of the cell c, we test whether the isoline value v is between the scalar vertex attributes vi and vj corresponding to the edge end points pi and pj.
  - If the test succeeds, the isoline intersects e at a point q.

$$q = \frac{p_i(v_j - v) + p_j(v - v_i)}{v_j - v_i}.$$

  - We repeat the previous procedure for all edges of our current cell and finally obtain a set of intersection points S = {qi} of the isoline with the cell.
  - Next, we must connect these points together to obtain the actual isoline.

Contour: Computation: Edge Tracking



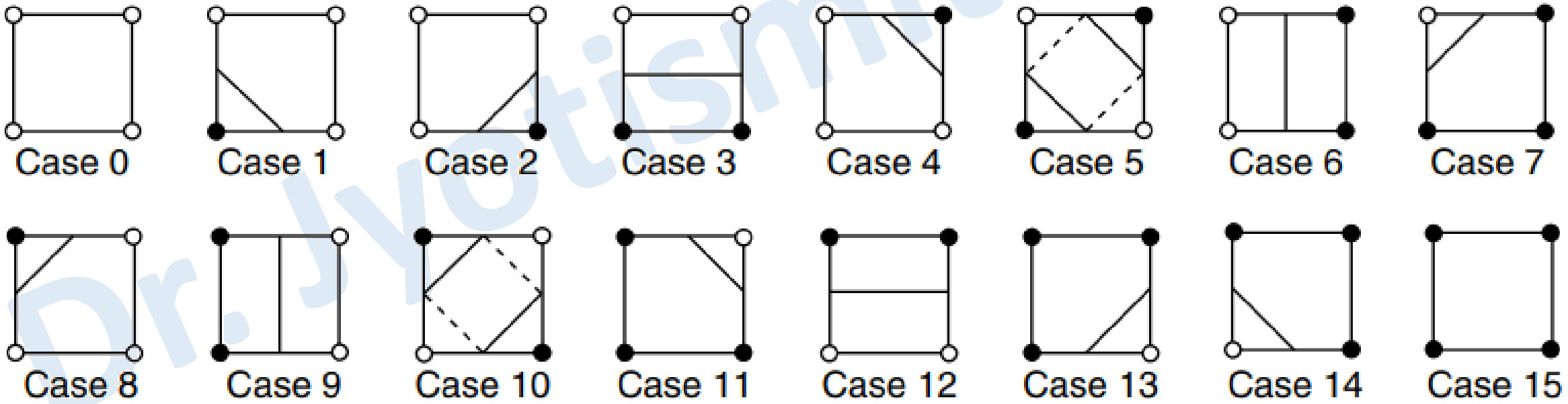
Constructing the isoline for the scalar value v = 0.48. The numbers in the figure indicate scalar values at the grid vertices.

Contour: Computation: Marching Squares (2D)

- Works only on structured data
- The basic assumption of these techniques is that a contour can pass through a cell in only a finite number of ways.
- A **case table** is constructed that enumerates all possible **topological states** of a cell, given combinations of scalar values at the cell points.
- The **number of topological states** depends on the number of cell vertices and the number of inside/outside relationships a vertex can have with respect to the contour value.
- A vertex is considered **inside** a contour if its scalar value is larger than the scalar value of the contour line.
- Vertices with scalar values less than the contour value are said to be **outside** the contour.

Contour: Computation: Marching Squares (2D)

- For example, if a cell has four vertices and each vertex can be either inside or outside the contour, there are 24 ¼ 16 possible ways that the contour passes through the cell.



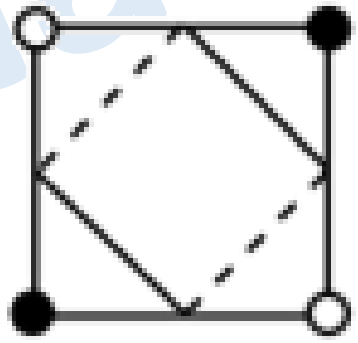
Sixteen different marching squares cases. Dark vertices indicate scalar value is above contour value. Cases 5 and 10 are ambiguous.

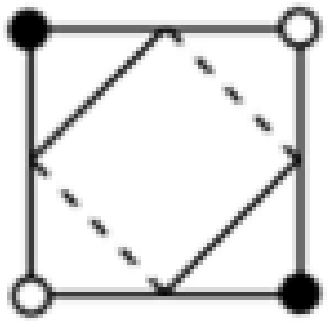
Contour: Computation: Marching Squares (2D)

- In summary, the marching algorithms proceed as follows:
  1. Select a cell.
  2. Calculate the inside/outside state of each vertex of the cell.
  3. Create an index by storing the binary state of each vertex in a separate bit.
  4. Use the index to look up the topological state of the cell in a case table.
  5. Calculate the contour location (via interpolation) for each edge in the case table.

Contour: Computation: Marching Squares (2D)

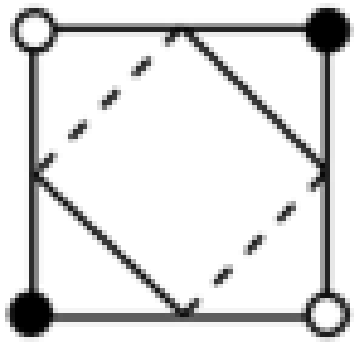
- An important issue is **contouring ambiguity**.
- Careful observation of marching squares cases 5 and 10 shows the ambiguity.
- For each ambiguous case, we implement one of the two possible cases. The choice for a particular case is independent of all other choices. Depending on the choice,
  - The contour may either extend or break the current contour.
- Either choice is acceptable since the resulting contour lines will be continuous and closed (or will end at the dataset boundary).


Case 5

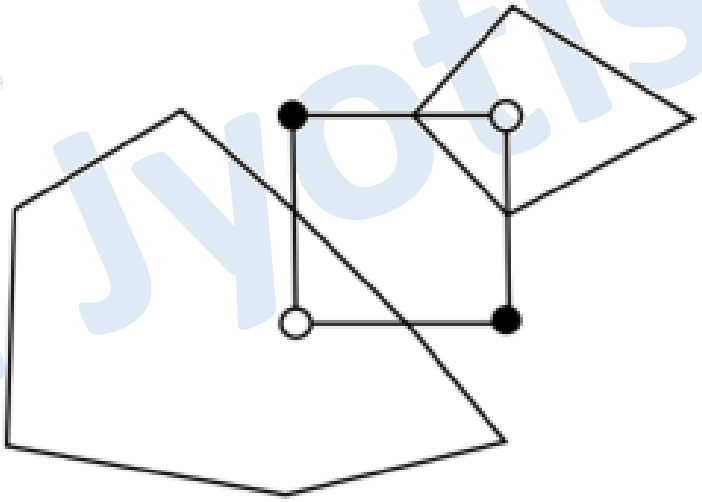

Case 10

Contour: Computation: Marching Squares (2D)

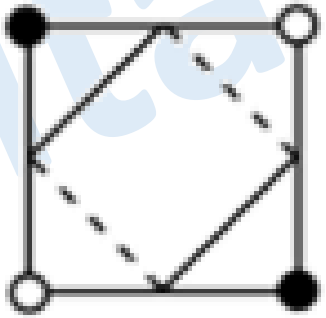
- Choosing a particular contour case will (a) break or (b) join the current contour.



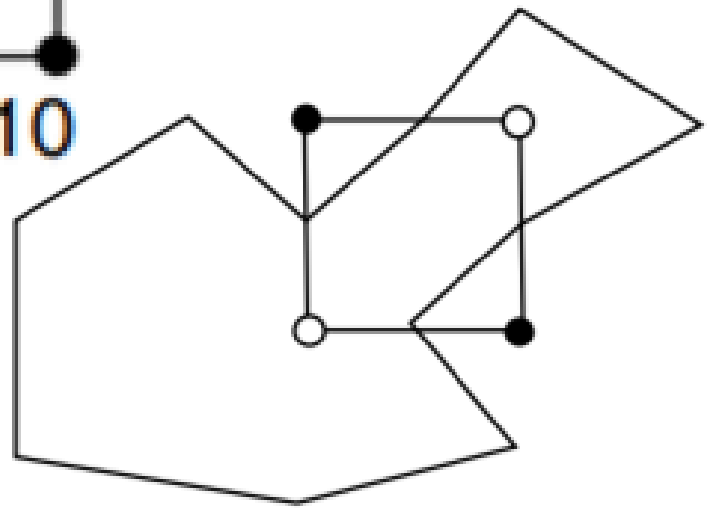
Case 5



(a) Break contour



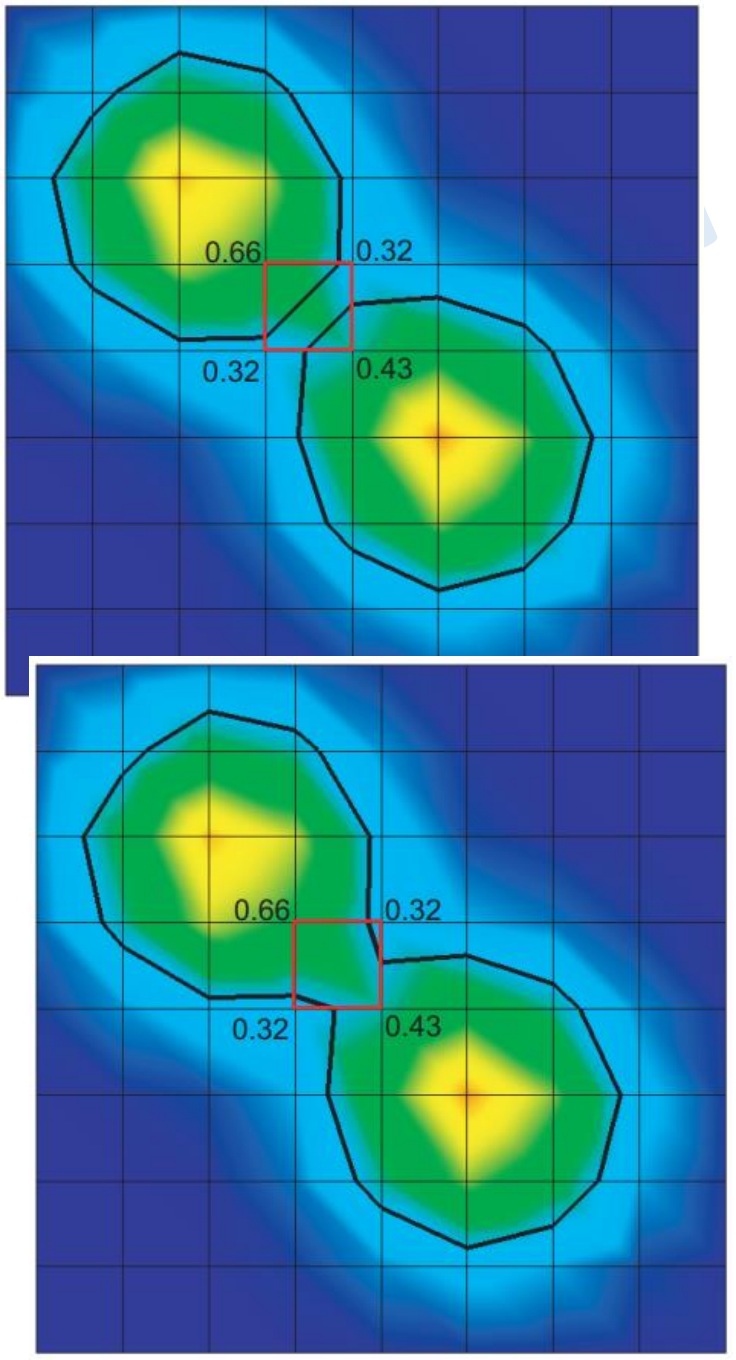
Case 10



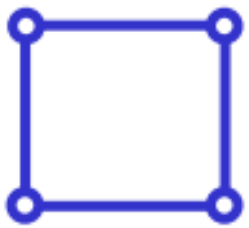
(b) Join contour

Contour: Computation: Marching Squares (2D)

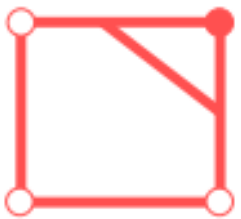
- If S contains exactly two points, there is no problem.
- However, S can contain more points, as illustrated in the Figure for the quad cell marked in red, which has four intersection points.
- In this case, exactly two possibilities exist for connecting the four intersection points, shown in the top and bottom images in Figure.
- Contour ambiguity for a quad cell (drawn in red). The isovalue is equal to 0.37.



Contour: Computation: Marching Squares (2D)



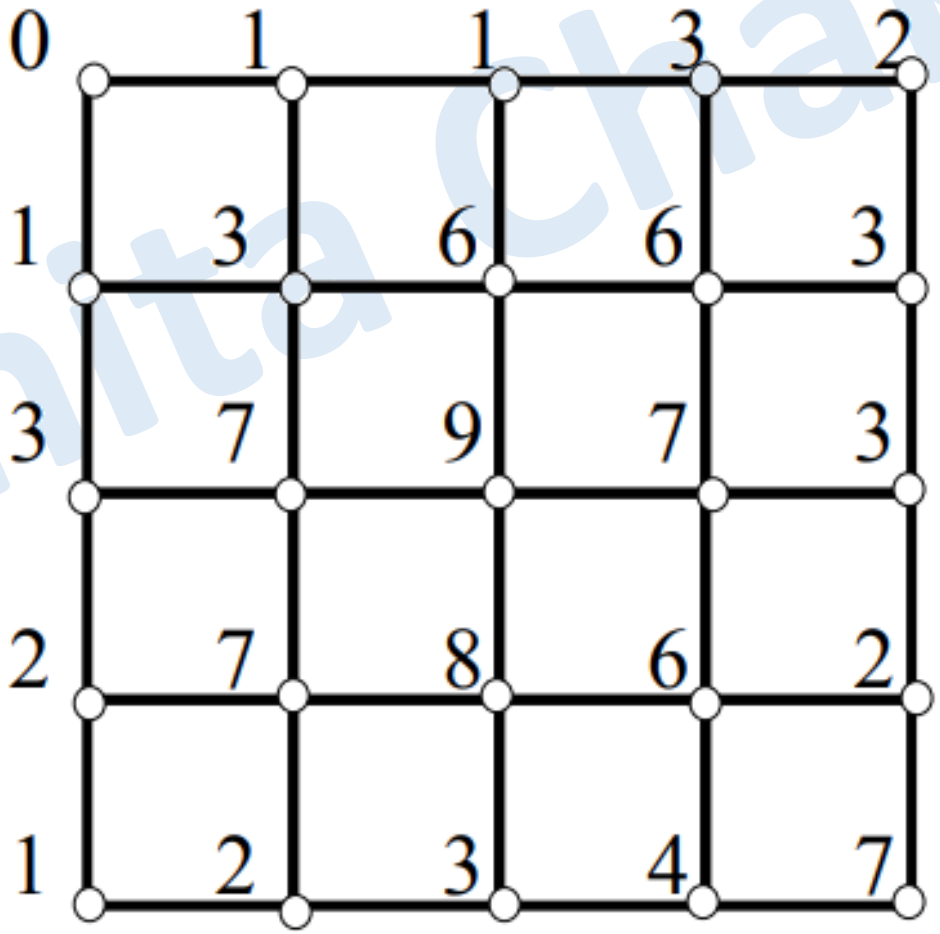
No intersection.



Contour intersects 1 edge

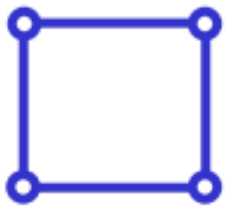
Contour intersects 2 edges

Ambiguous case.

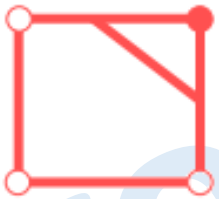


Contour: Computation: Marching Squares (2D)

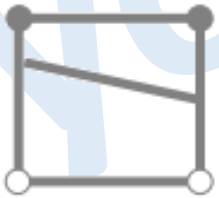
- Decide whether each vertex is inside or outside contour



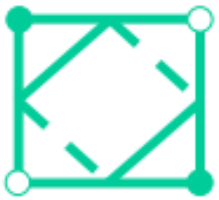
No intersection.



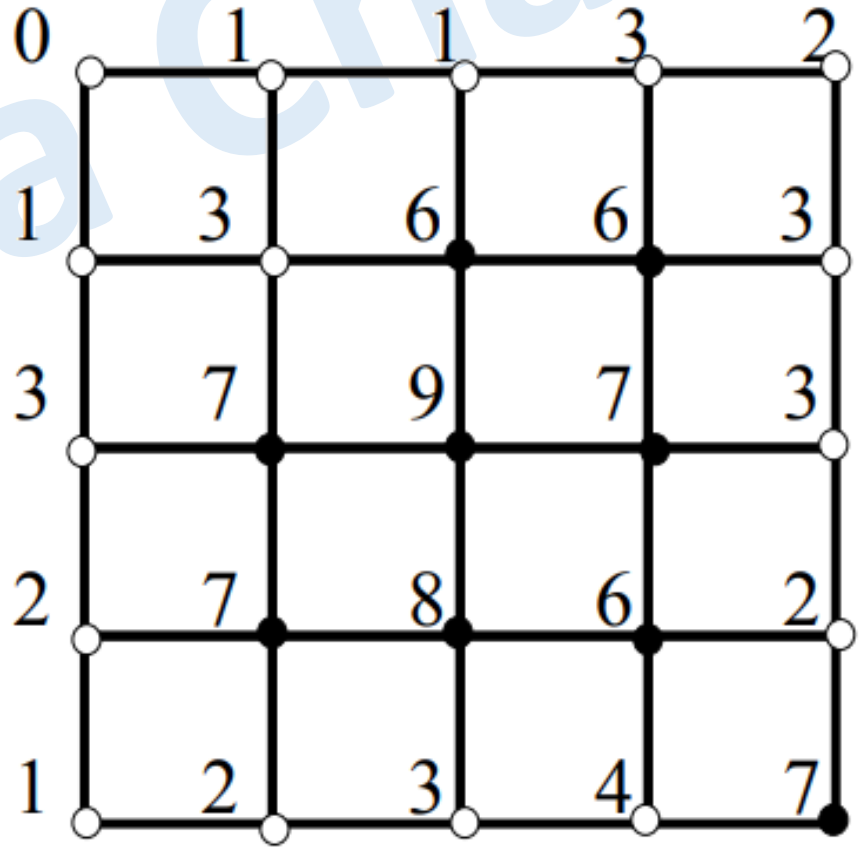
Contour intersects 1 edge



Contour intersects 2 edges



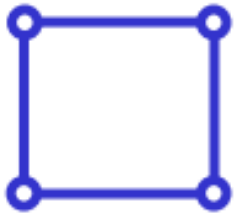
Ambiguous case.



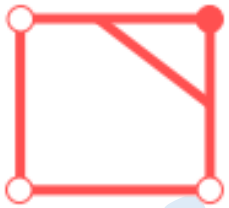
Contour value =5

Contour: Computation: Marching Squares (2D)

- Classify each cell as one of the cases

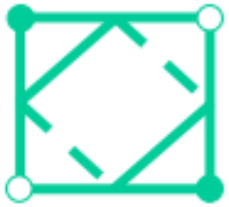


No intersection.

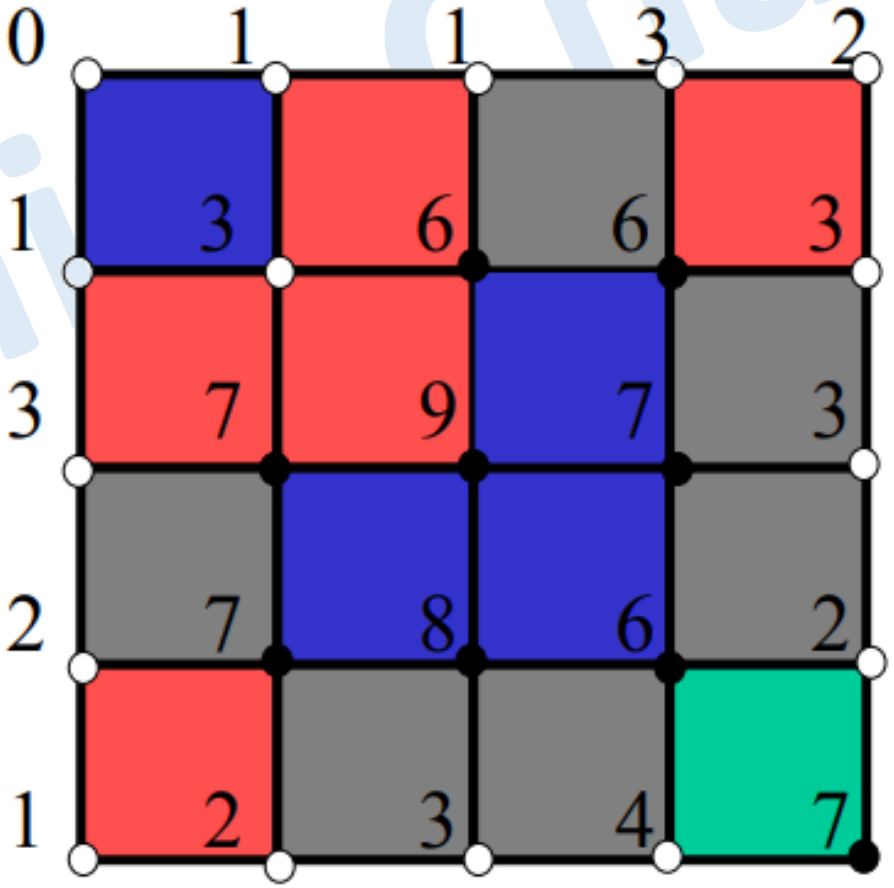


Contour intersects 1 edge

Contour intersects 2 edges



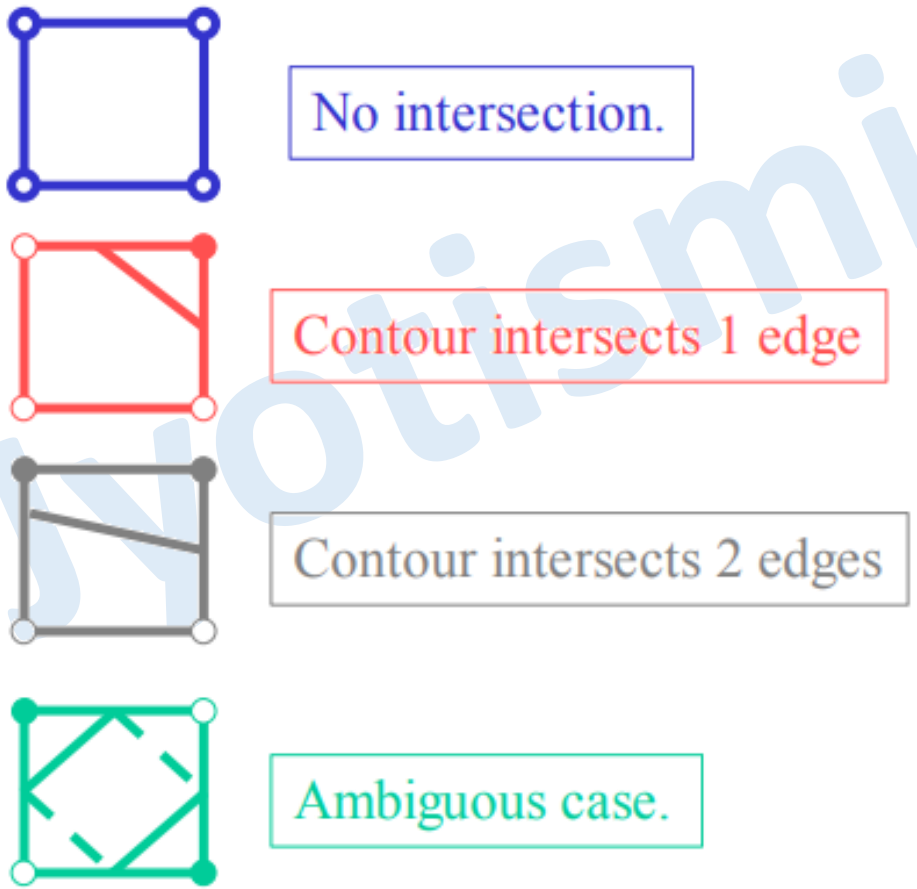
Ambiguous case.

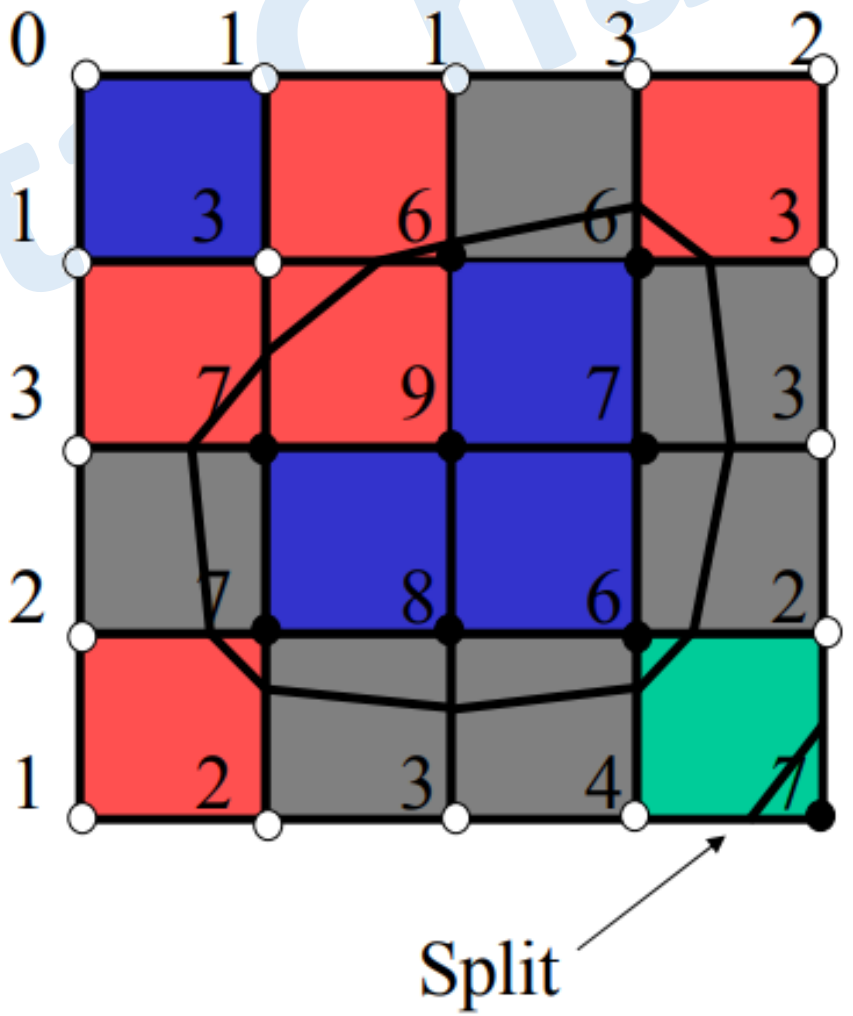


Contour value =5

Contour: Computation: Marching Squares (2D)

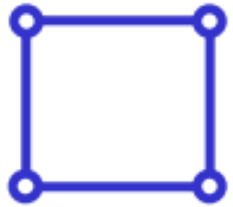
- Determine the edges that are intersected



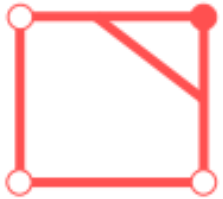


Contour: Computation: Marching Squares (2D)

- Finally : resolve any ambiguity

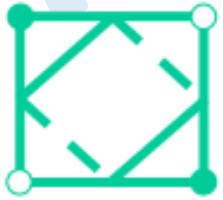


No intersection.

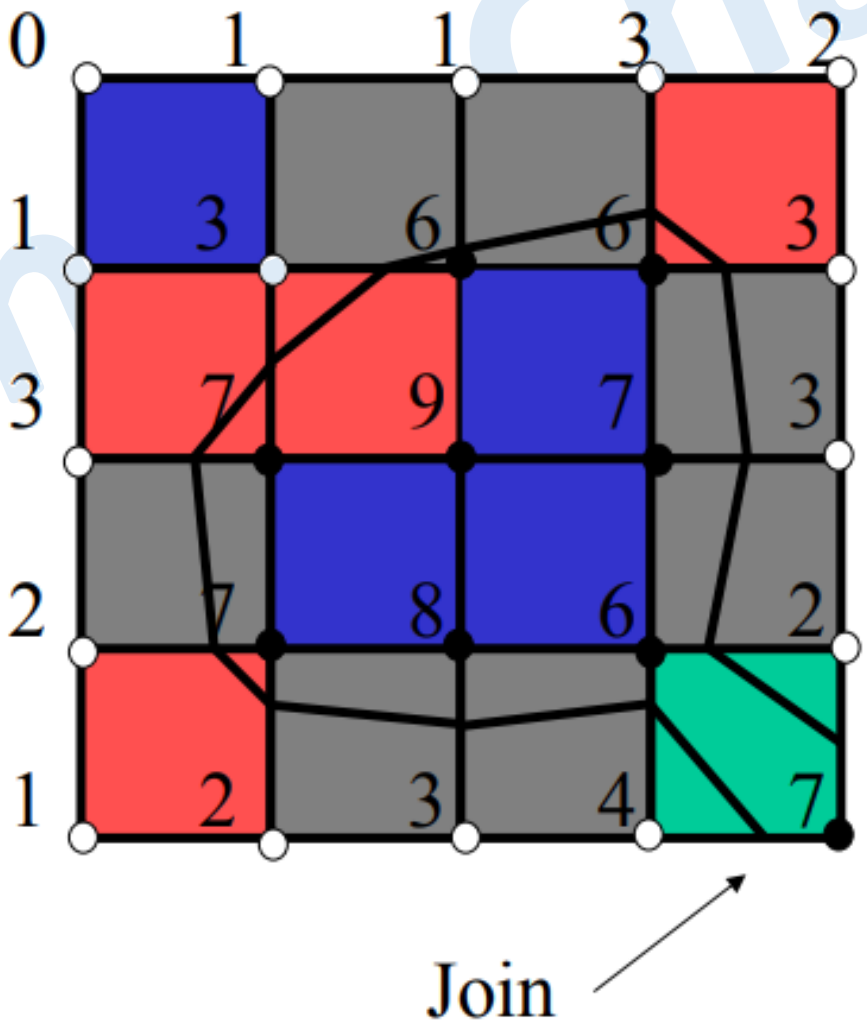


Contour intersects 1 edge

Contour intersects 2 edges



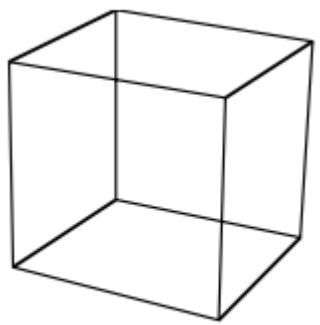
Ambiguous case.



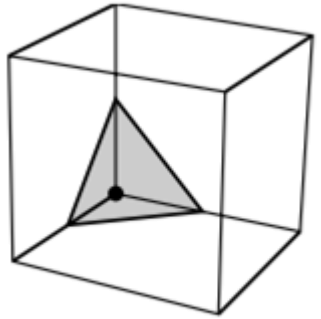
Contour: Computation: Marching Cubes (3D)

- The 3D analogy of marching squares is marching cubes.
- Here, there are 256 different combinations of scalar value, given that there are eight points in a cubical cell (i.e., 28 combinations).
- Figure shows these combinations reduced to 15 cases by arguments of symmetry.
- This is the so-called marching cubes case table.

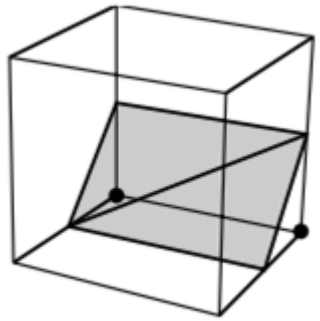
Contour: Computation: Marching Cubes (3D)



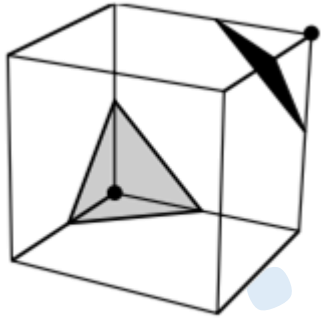
Case 0



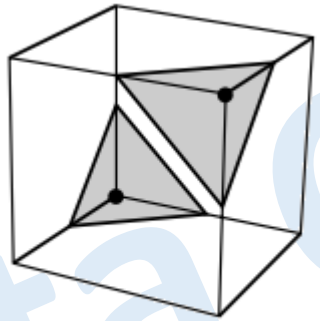
Case 1



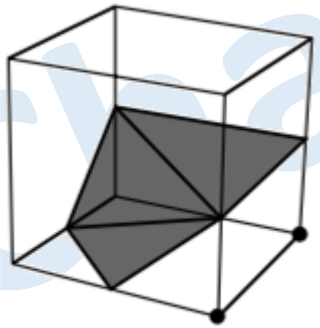
Case 2



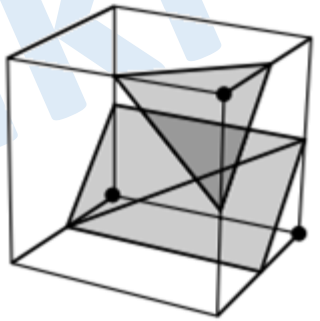
Case 3



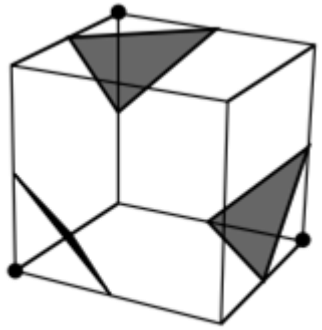
Case 4



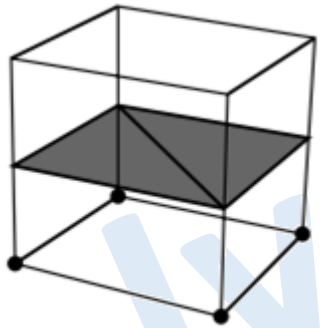
Case 5



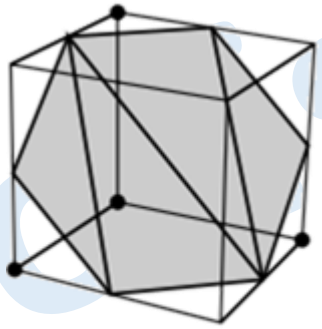
Case 6



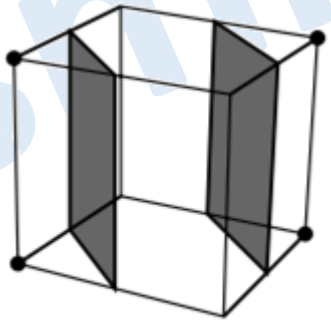
Case 7



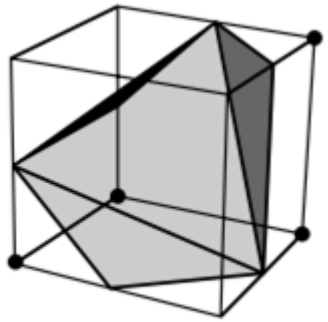
Case 8



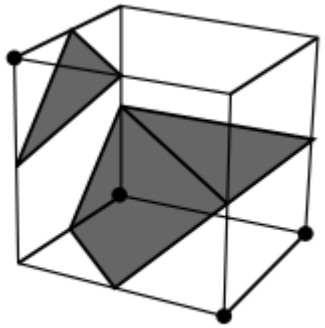
Case 9



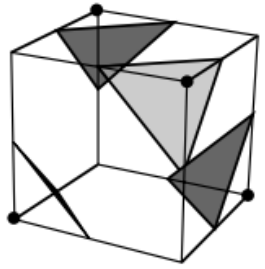
Case 10



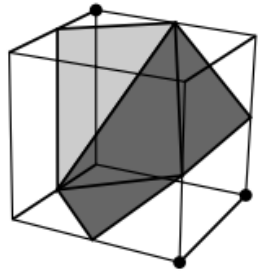
Case 11



Case 12



Case 13

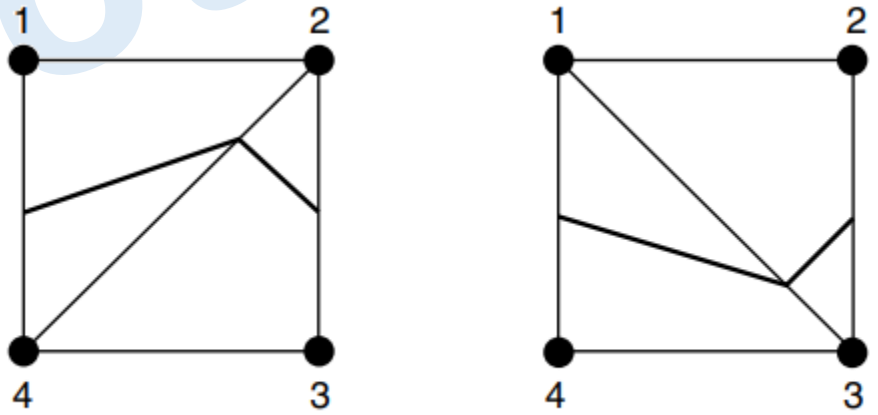


Case 14

Marching cubes cases for 3D isosurface generation

Contour: Computation: Marching Cubes (3D)

- Several different approaches have been taken to remedy the problem of **contouring ambiguity** in 3D.
- One approach tessellates the cubes with **tetrahedra** and uses a marching **tetrahedra technique**.
- This choice may result in artificial ‘‘**bumps**’’ in the **isosurface** because of interpolation along the face diagonals.

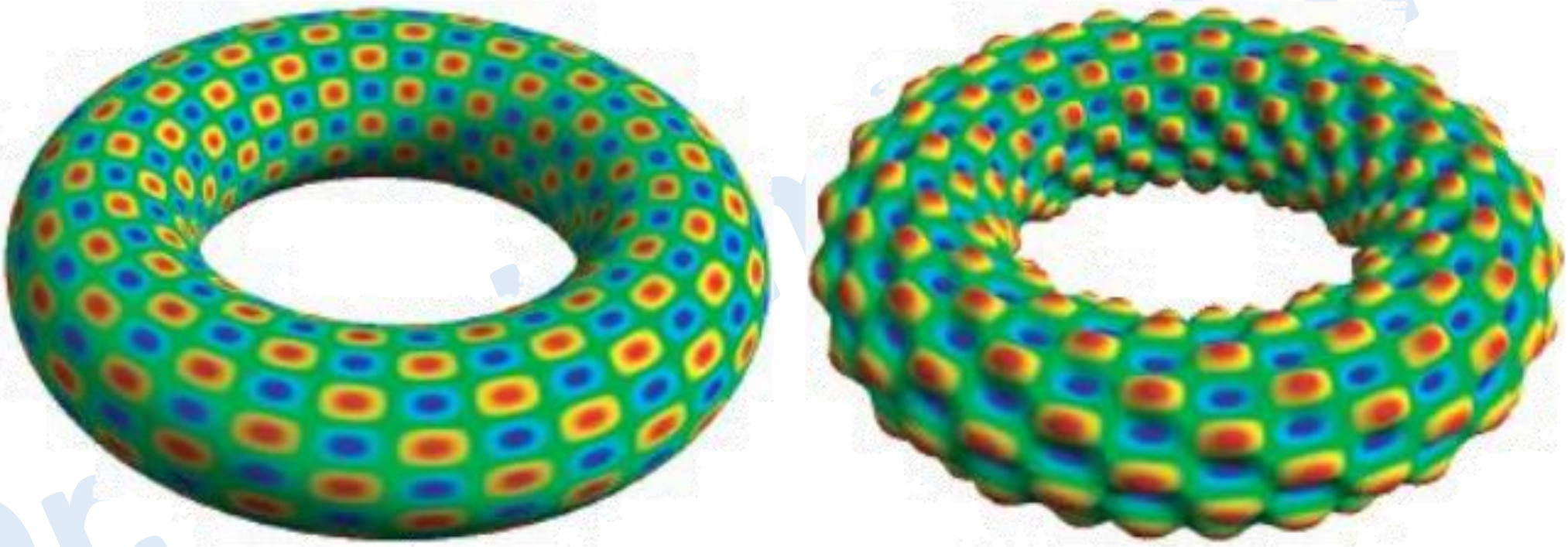


Iso-value = 2.5

Height plots

- Height plots, also called **elevation** or **carpet plots.**
- Given a two-dimensional surface Ds ∈ D, part of a scalar dataset D, height plots can be described by the mapping operation
  - $m : Ds \rightarrow D, m(x) = x + s(x)n(x), \forall x \in Ds$
- where s(x) is the scalar value of D at the point x and n(x) is the normal to the surface Ds at x.
- In their most common variant, height plots warp a planar surface Ds.
- The values of the scalar dataset on the torus surface are shown with a blue-to-red colormap in Figure (a).
- Figure (b) shows the height plot of these scalar values, done by warping the torus surface with the scalar values along the surface normal.

Height plots

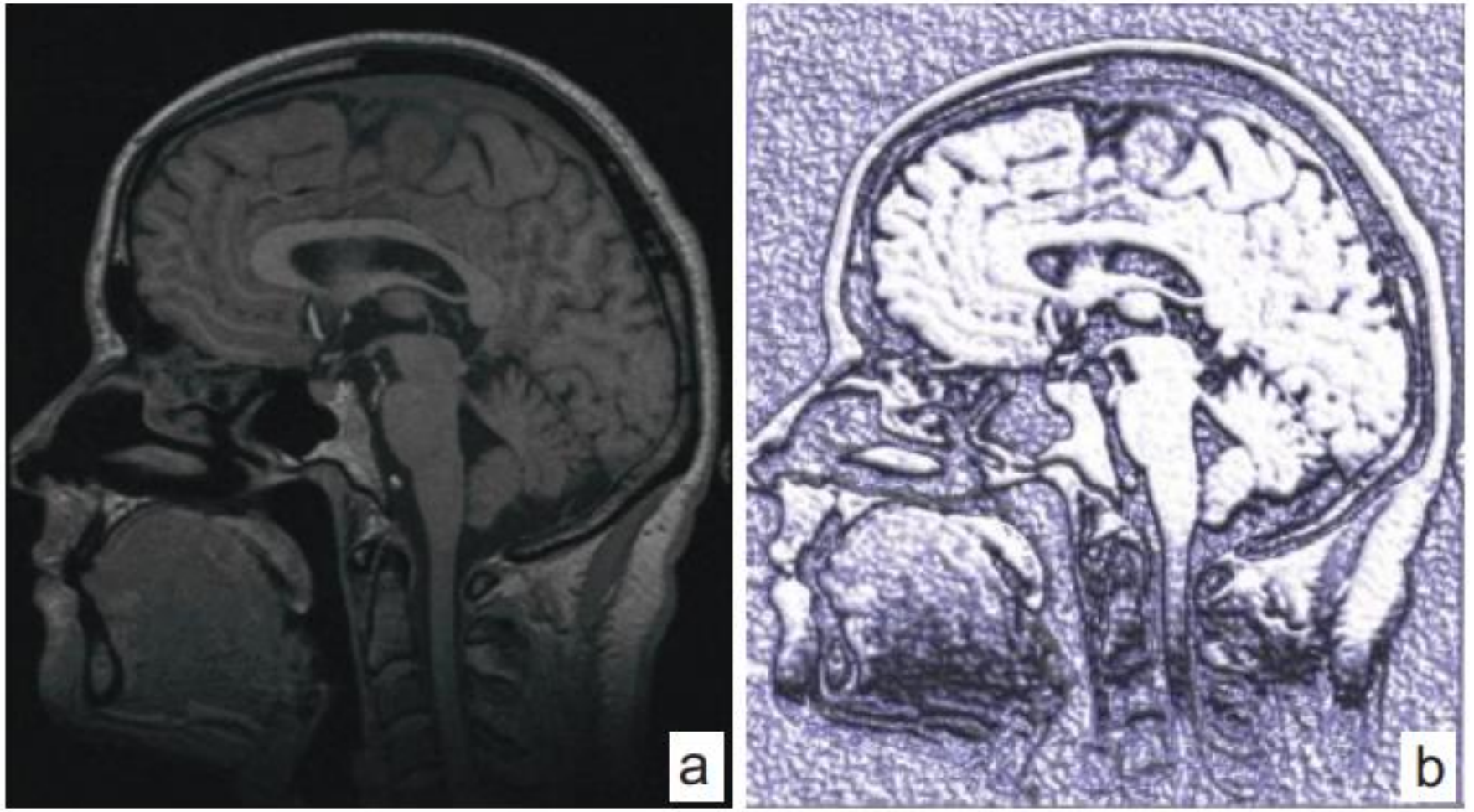


(a) (b)

Height plots

- Figure (a) shows a 2D scalar plot of a brain CT slice, displayed with a grayscale colormap.
- Here, scalar values encode tissue density, with white corresponding to the hardest structures (bone) and black corresponding to air.
- Since a linear scalar-to-grayscale colormap was used here, high-contrast borders between regions of different densities are easily seen.
- Also, large differences of absolute data values are easily seen—we can, for instance, locate the hardest and softest tissues within the given scan by looking for the brightest, respectively darkest, areas in the image.
- However, small-scale data variations are not easily visible, since the dynamic range of a grayscale colormap on a typical computer screen has only 256 different values.
- Figure (b) shows the same dataset, this time displayed using a height plot, visualized from above, and with a directional value from a slightly oblique direction.

Height plots



Plotting Scalar Data

- Table 1: CDC/WHO categories for BMI

BMI range	Category
< 18.5	Underweight
18.5–24.9	Normal weight
25.0–29.9	Overweight
30.0 and above	Obese

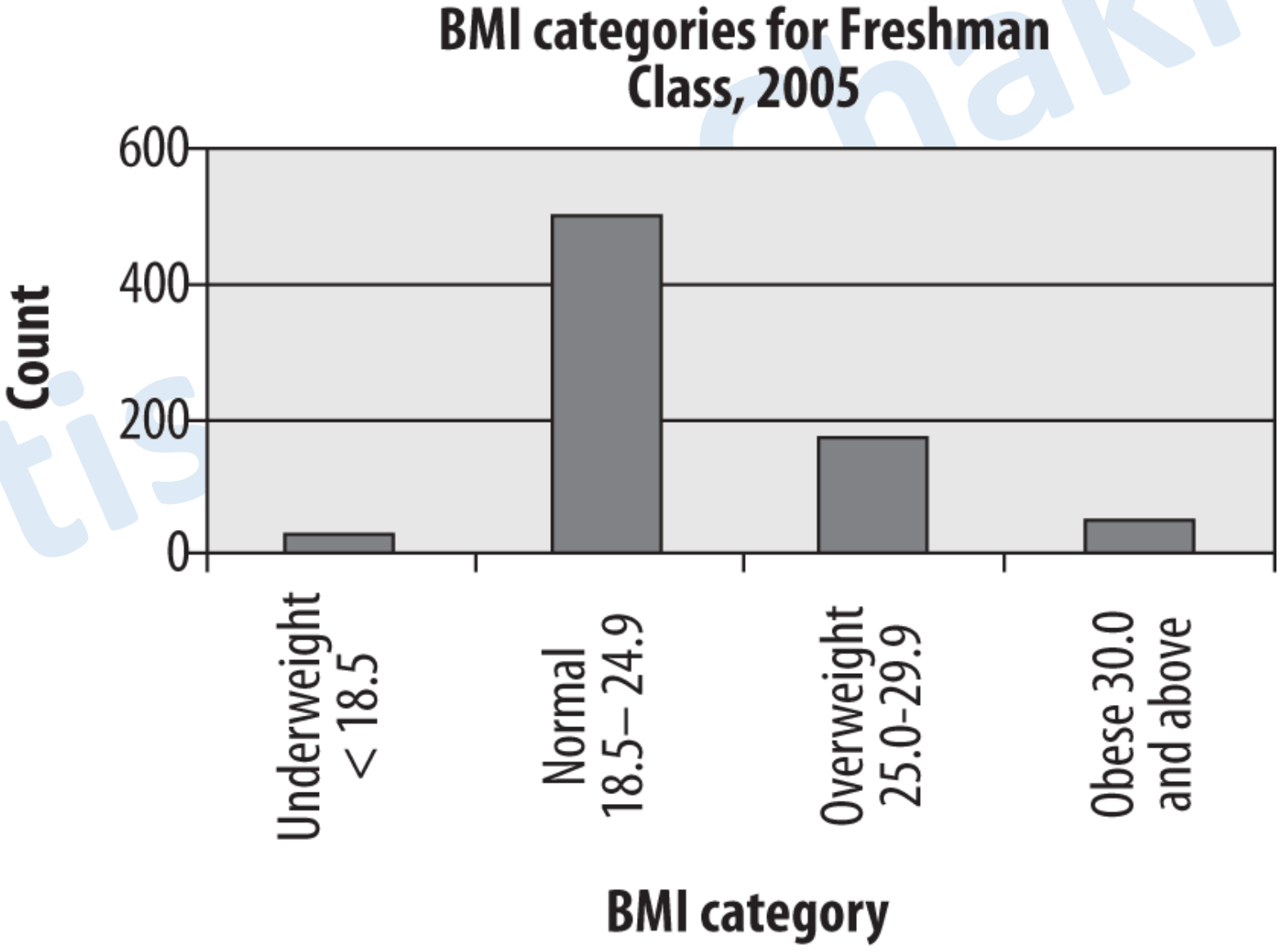
- Table 2: Distribution of BMI in the freshman class of 2005

BMI range	Number
< 18.5	25
18.5–24.9	500
25.0–29.9	175
30.0 and above	50

This table presents raw numbers or counts for each category, which are sometimes referred to as *absolute frequencies*

Plotting Scalar Data: Barplot

- The freshman BMI information presented in a bar chart: Absolute frequency of BMI categories in freshman class



Plotting Scalar Data

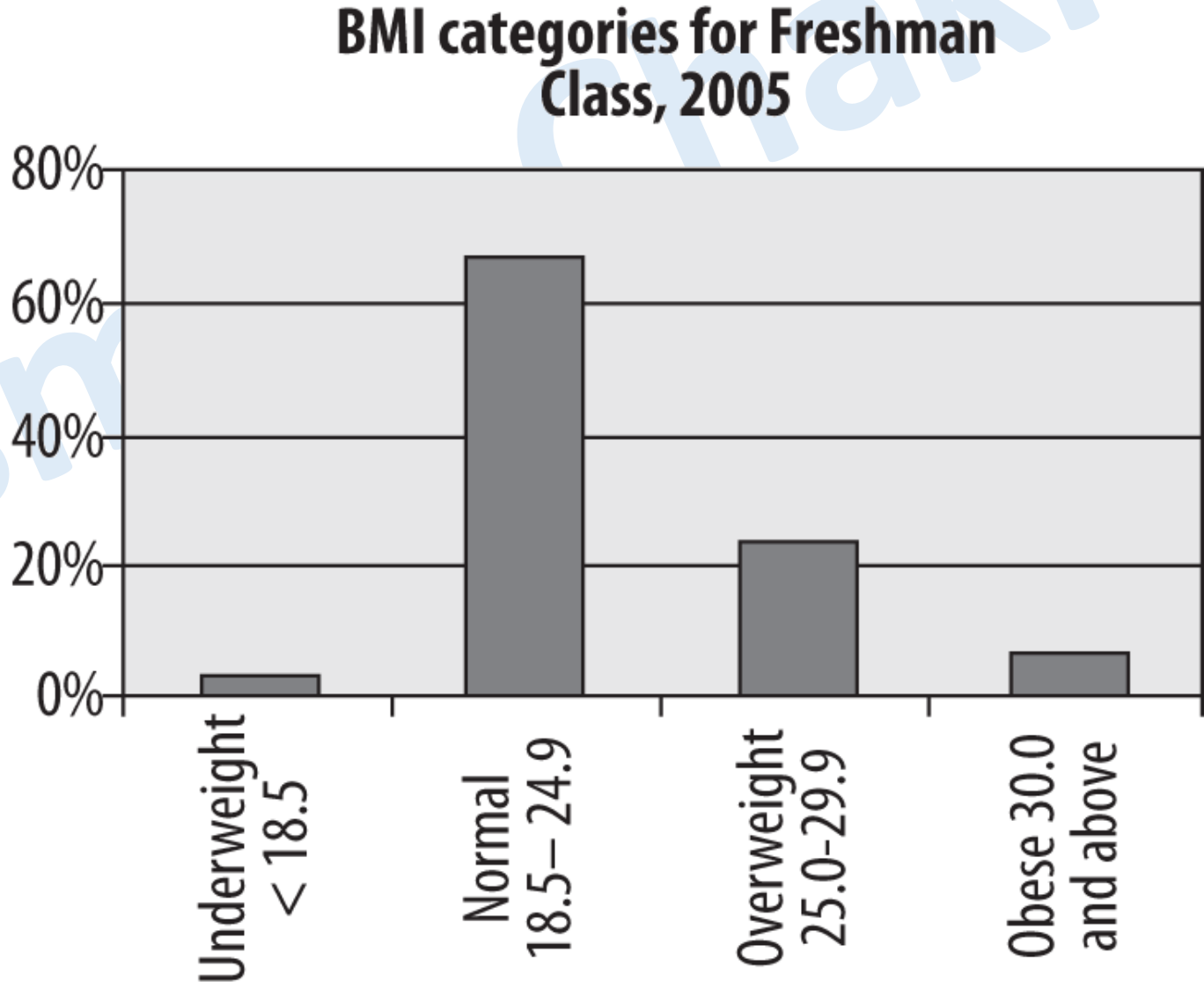
- The relative frequency is calculated by dividing the number of cases in each category by the total number of cases (750) and multiplying by 100.
- Absolute and relative frequency of BMI categories for the freshmen class of 2005

BMI range	Number	Relative frequency
< 18.5	25	3.3%
18.5–24.9	500	66.7%
25.0–29.9	175	23.3%
30.0 and above	50	6.7%

Relative frequencies should add up to approximately 100%, although the total might be slightly higher or lower due to rounding error

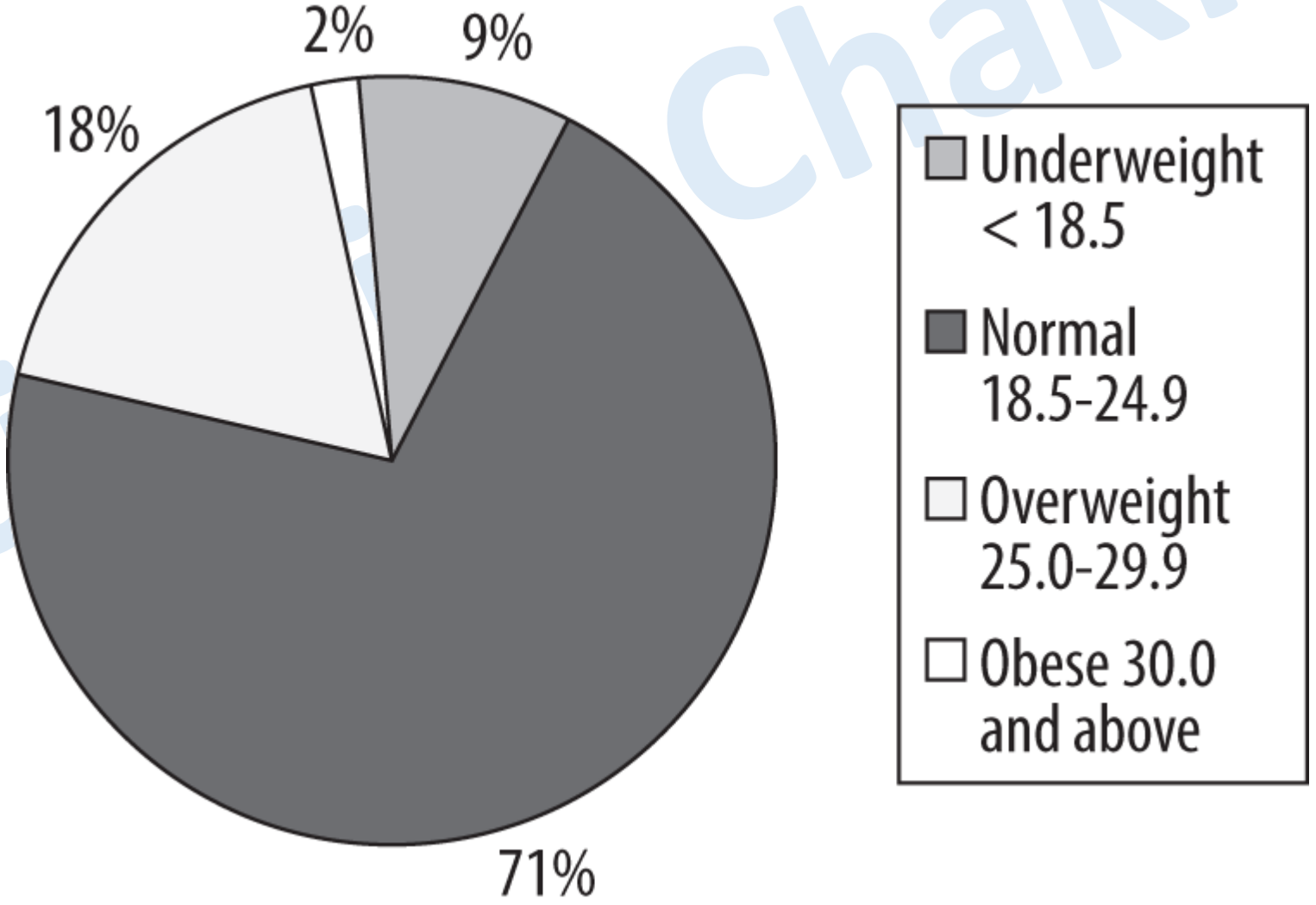
Plotting Scalar Data: Barplot

- Relative frequency of BMI categories in freshman class.
- Note that the two charts are identical except for the y-axis (vertical axis) labels, which are frequencies and percentages



Plotting Scalar Data: Pie chart

- Pie charts, are most useful when there are only a few categories of information and the differences among those categories are fairly large.
- Pie chart showing BMI distribution for freshmen entering in 2005



Plotting Scalar Data: Pie chart

- Imagine you survey your friends to find the kind of movie they like best:

Table: Favorite Type of Movie

Comedy	Action	Romance	Drama	SciFi
4	5	6	1	4

- First, put your data into a table (like above), then add up all the values to get a total:

Table: Favorite Type of Movie

Comedy	Action	Romance	Drama	SciFi	TOTAL
4	5	6	1	4	**20**

Plotting Scalar Data: Pie chart

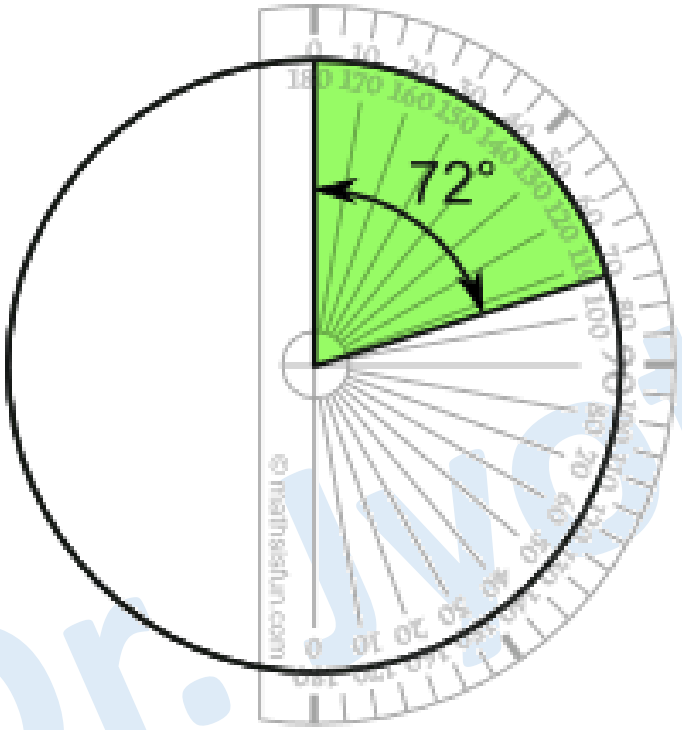
- Next, divide each value by the total and multiply by 100 to get a percent:

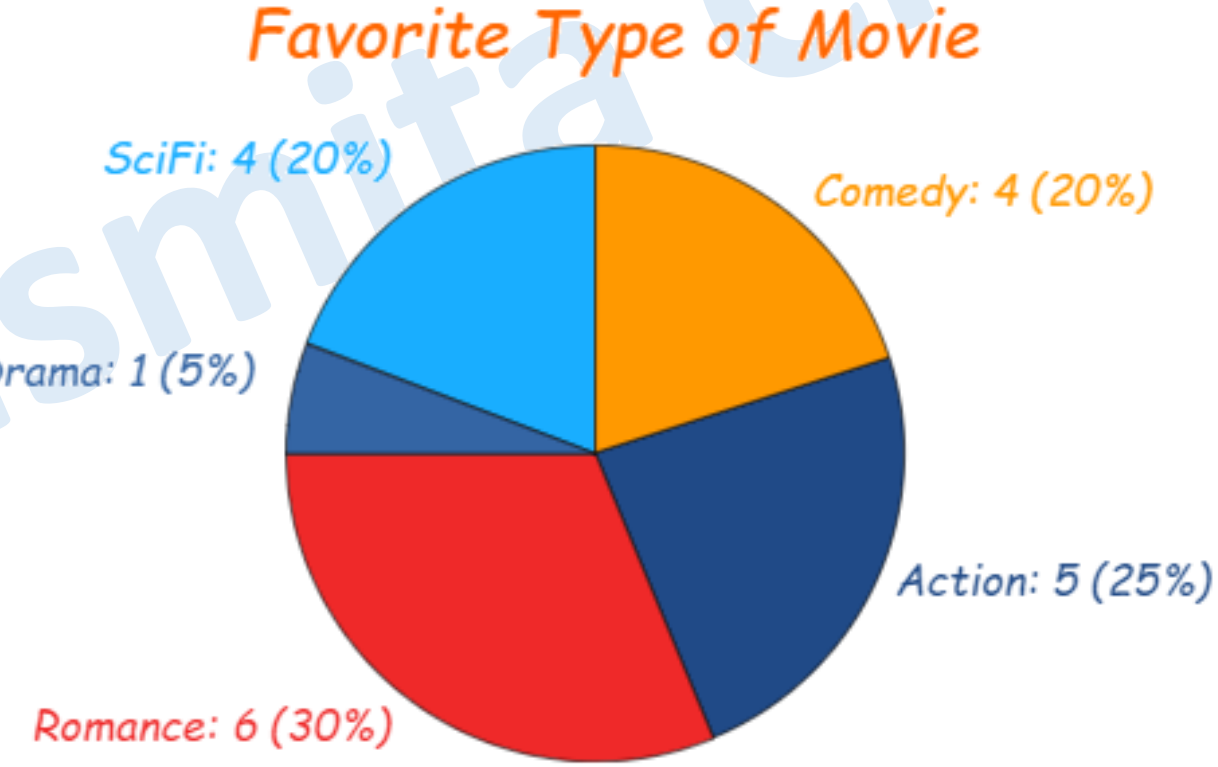
Comedy	Action	Romance	Drama	SciFi	TOTAL
4	5	6	1	4	**20**
4/20 = **20%**	5/20 = **25%**	6/20 = **30%**	1/20 = **5%**	4/20 = **20%**	**100%**

- Now to figure out how many degrees for each "pie slice"

Comedy	Action	Romance	Drama	SciFi	TOTAL
4	5	6	1	4	**20**
20%	**25%**	**30%**	**5%**	**20%**	**100%**
4/20 × 360° = **72°**	5/20 × 360° = **90°**	6/20 × 360° = **108°**	1/20 × 360° = **18°**	4/20 × 360° = **72°**	**360°**

Plotting Scalar Data: Pie chart





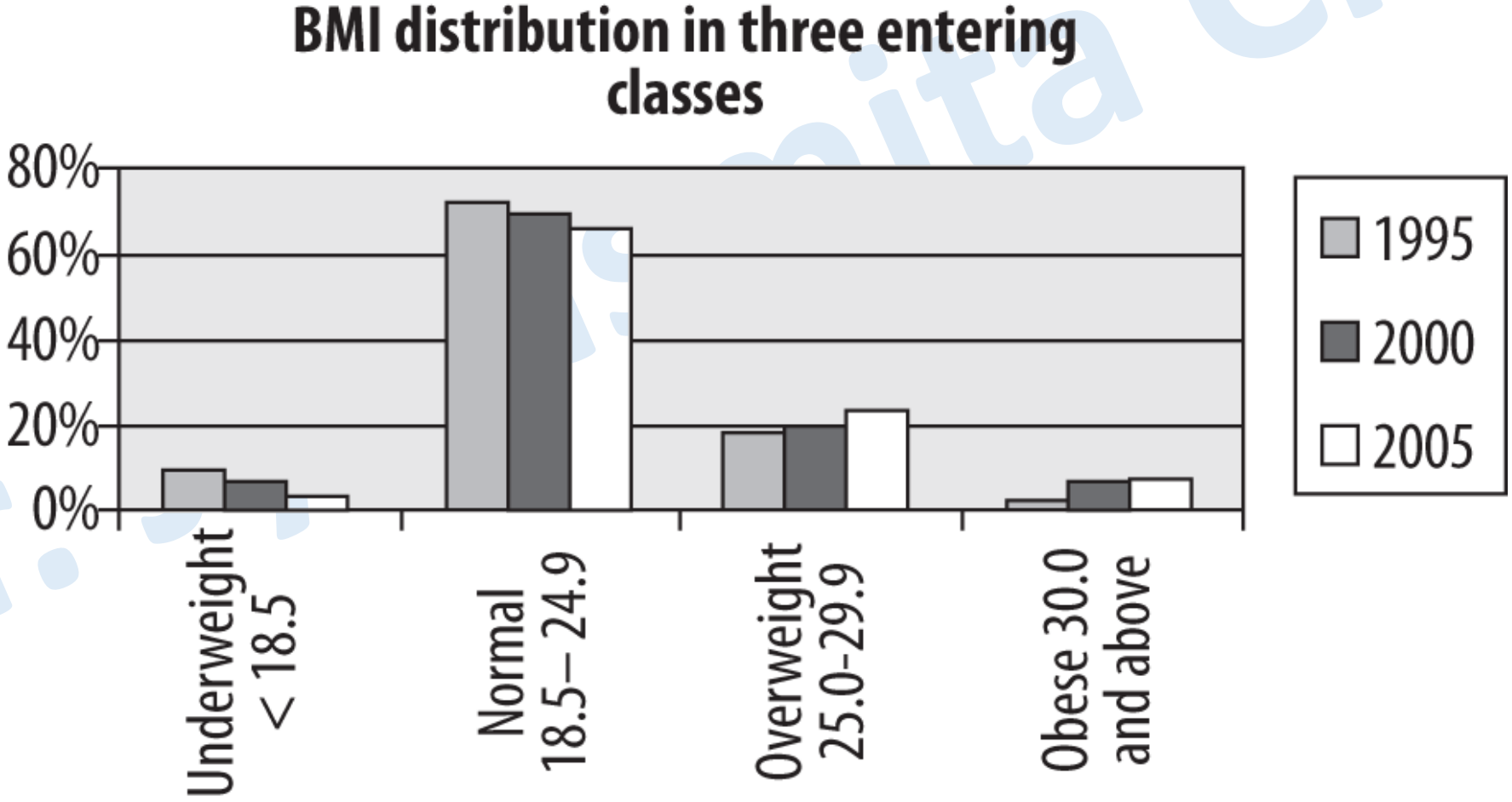
Plotting Scalar Data

- Absolute and relative frequencies of BMI for three entering classes

BMI range	1995		2000		2005	
Underweight < 18.5	50	8.9%	45	6.8%	25	3.3%
Normal 18.5–24.9	400	71.4%	450	67.7%	500	66.7%
Overweight 25.0–29.9	100	17.9%	130	19.5%	175	23.3%
Obese 30.0 and above	10	1.8%	40	6.0%	50	6.7%
Total	560	100.0%	665	100.0%	750	100.0%

Plotting Scalar Data: Grouped Bar chart

- Grouped Bar chart of BMI distribution in three entering classes



Plotting Scalar Data: Boxplot

- Let **n** be the number of data values in the data set.
- The **Median (Q2**) is the middle value of the data set.

$$\textbf{Median (Q2)} = \frac{1}{2}\ (n+1)\ th\ term$$

- The **Lower quartile [25th quartile] (Q1)** is the median of the lower half of the data set

$$\textbf{Lower Quartile (Q1)} = \frac{1}{4}\ (n+1)\ th\ term$$

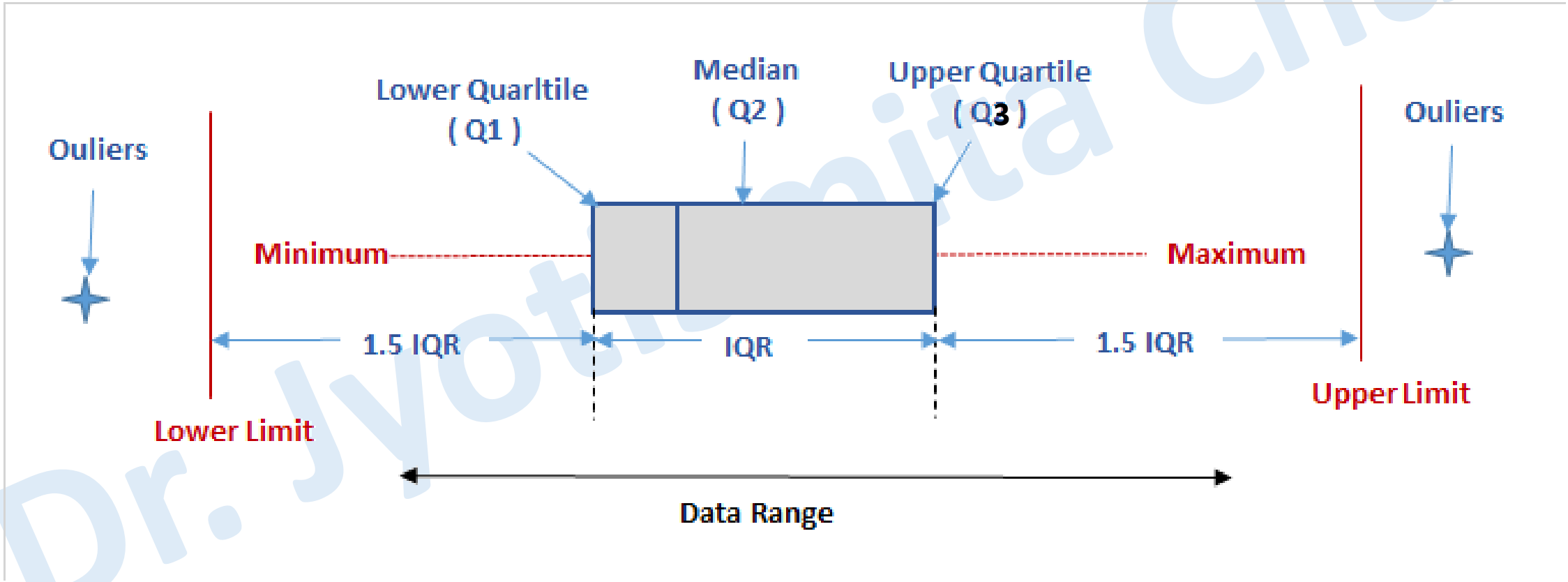
- The **Upper quartile [75th quartile] (Q3)** is the median of the upper half of the data set.

$$\textbf{Upper Quartile (Q3)} = \frac{3}{4}\ (n+1)\ th\ term$$

Plotting Scalar Data: Boxplot

- The **Interquartile range (IQR)** is the spread of the middle 50% of the data values.
- Interquartile Range (IQR) = Upper Quartile (Q3) – Lower Quartile (Q1)
- **IQR = Q3 – Q1**
- **Lower Limit = Q1 – 1.5 IQR.**
- **Upper Limit = Q3 + 1.5 IQR**
- So any value that will be more than the upper limit or lesser than the lower limit will be the outliers. Only the data that lies within Lower and upper limit are statistically considered normal and thus can be used for further observation or study.

Plotting Scalar Data: Boxplot



Plotting Scalar Data: Boxplot

- Let the data range be 100, 151, 236, 269,271,278,283,291, 301, 303, and 400
- Therefore **n = 11**

$$\text{Median (Q2)} = \frac{1}{2}\,(11+1)\,th\ term = 6th\ Term$$

$$\mathbf{Q2 = 278}$$

Plotting Scalar Data: Boxplot

$$\text{Lower Quartile (Q1)} = \frac{1}{4}\ (11+1)\ th\ term = 3rd\ Term$$

$$\mathbf{Q1\ = 236}$$

$$\text{Upper Quartile (Q3)} = \frac{3}{4}\ (11+1)\ th\ term = 9th\ Term$$

$$\mathbf{Q3\ = 301}$$

Inter Quartile Range (IQR) = Q3 – Q1 = 301 - 236 = 65

IQR = 65

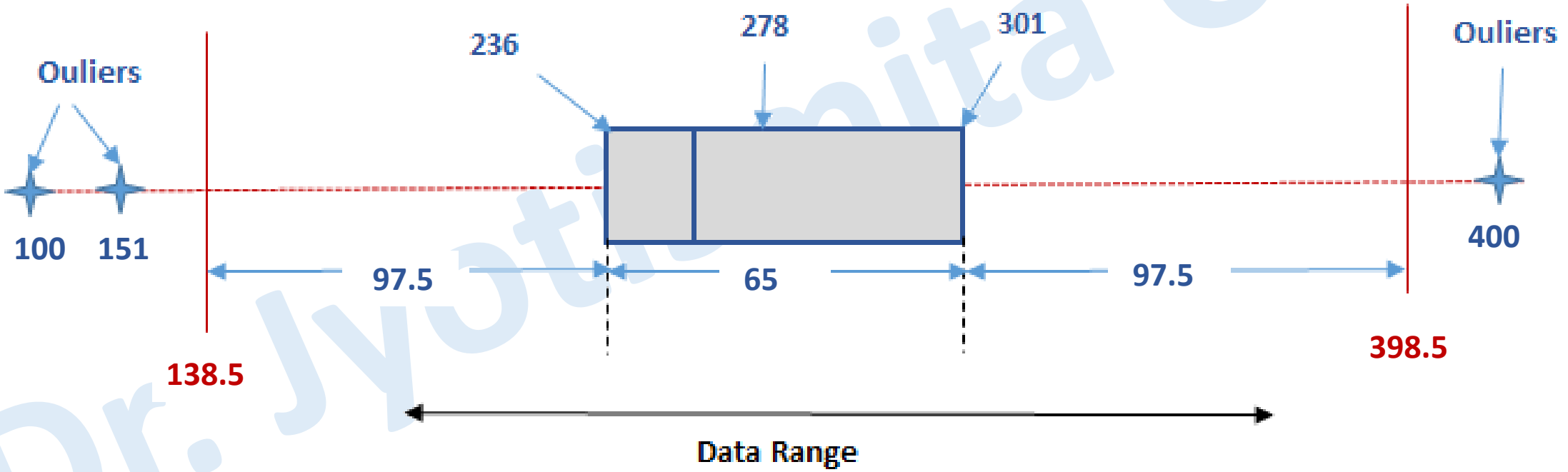
Lower Limit = Q1 – 1.5 IQR = 236 – 1.5 (65)

Lower Limit = 138.5

Upper Limit = Q3 + 1.5 IQR = 301 + 1.5 (65)

Upper Limit = 398.5

Plotting Scalar Data: Boxplot



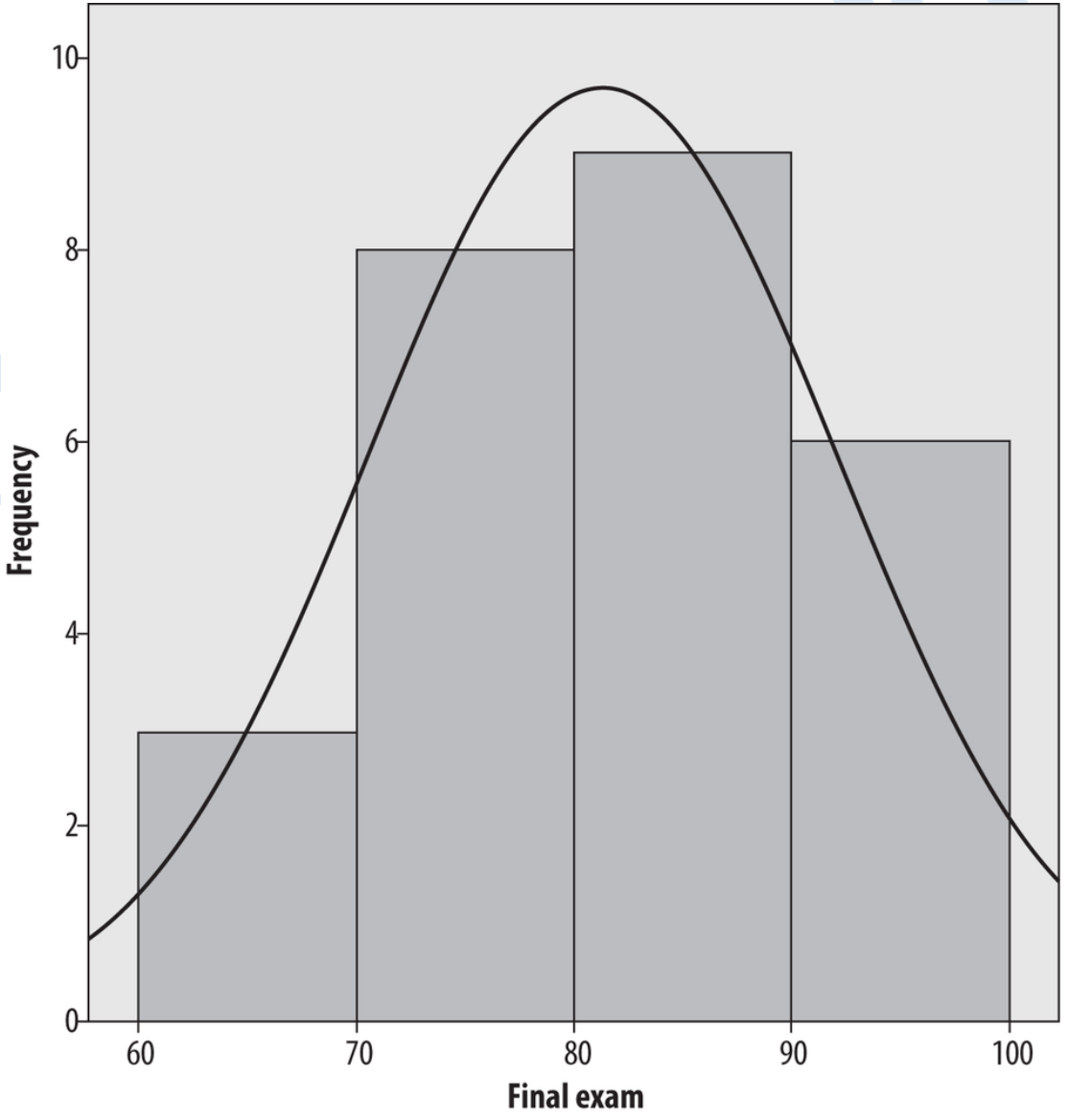
Plotting Scalar Data: Boxplot

- Hence it is clear that any range above 333.5 or below 201.5 are outliers. Hence in the data series 199, 201, 236, 269,271,278,283,291, 301, 303, 341, **outliers are 199, 201 and 341.** These 3 values which lies on either of the **extremes** can be considered **abnormal** and should be discarded from the entire series so that any analysis made on this series is not influenced by these extreme values. So the data series that should be considered for further observation or study after discarding the outliers are as below.

  236, 269,271,278,283,291, 301, 303

Plotting Scalar Data: Histogram

- Suppose we have the final exam grades for 26 students and want to present them graphically. These are the grades:
- 61, 64, 68, 70, 70, 71, 73, 74, 74, 76, 79, 80, 80, 83, 84, 84, 87, 89, 89, 89, 90 92, 95, 95, 98, 100
- *Histogram with a bin width of 10*



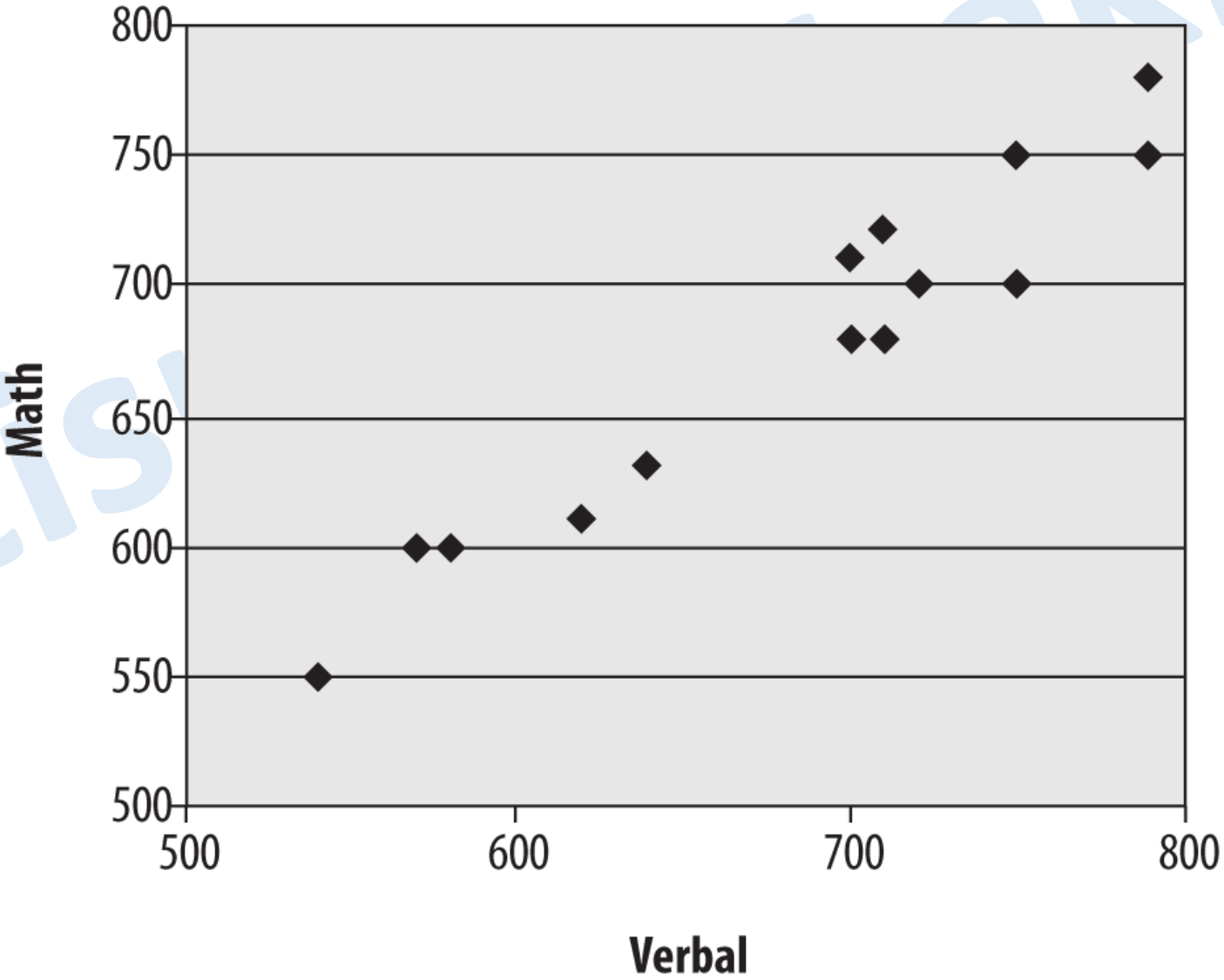
Plotting Scalar Data: Scatter plot

- SAT scores for 15 students

Math	Verbal
750	750
700	710
720	700
790	780
700	680
750	700
620	610
640	630
700	710
710	680
540	550
570	600
580	600
790	750
710	720

Plotting Scalar Data: Scatter plot

Math SAT score on the *y*-axis (vertical axis) and verbal SAT score on the *x*-axis (horizontal axis), makes the relationship between scores much clearer.

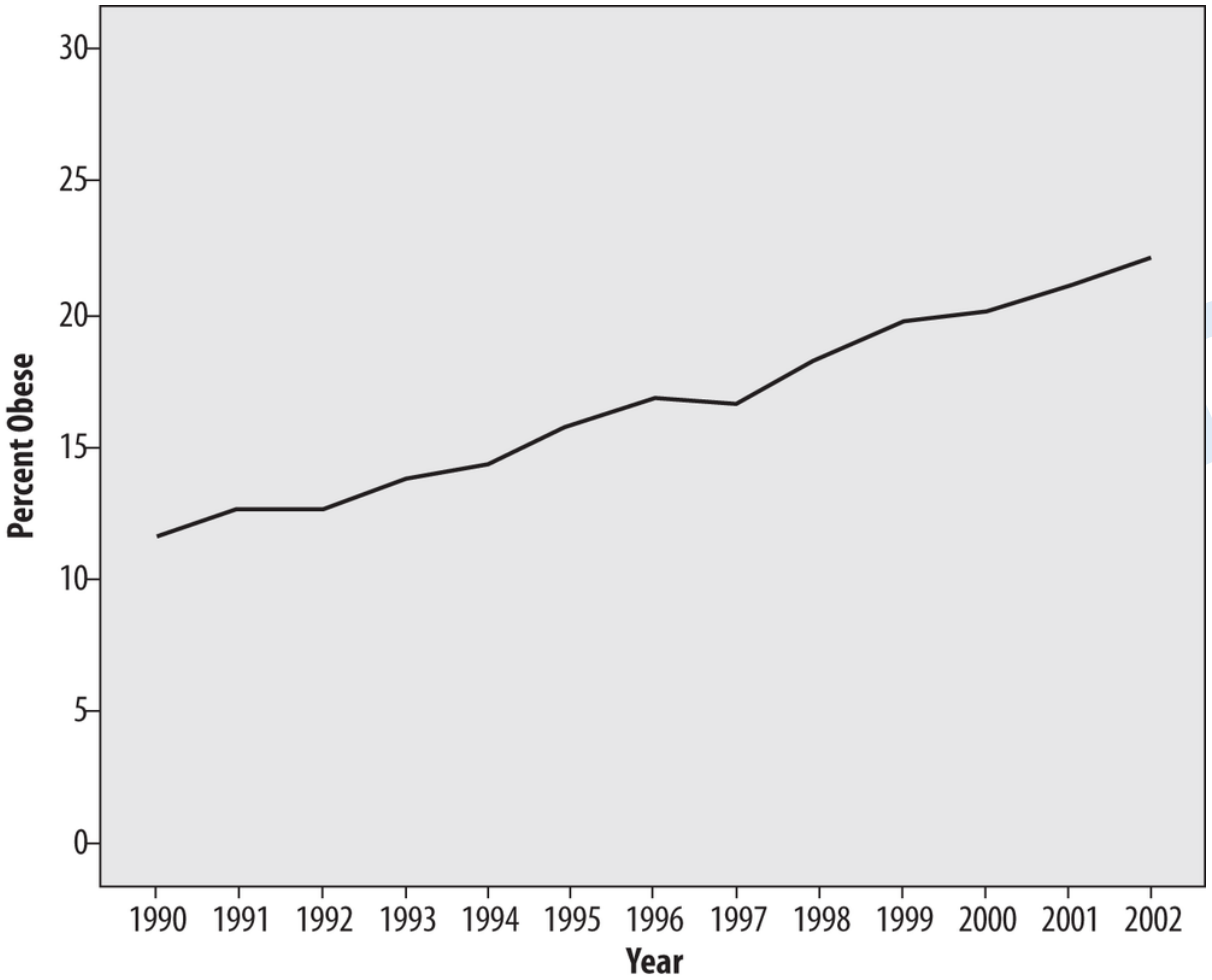


Plotting Scalar Data: Line Graphs

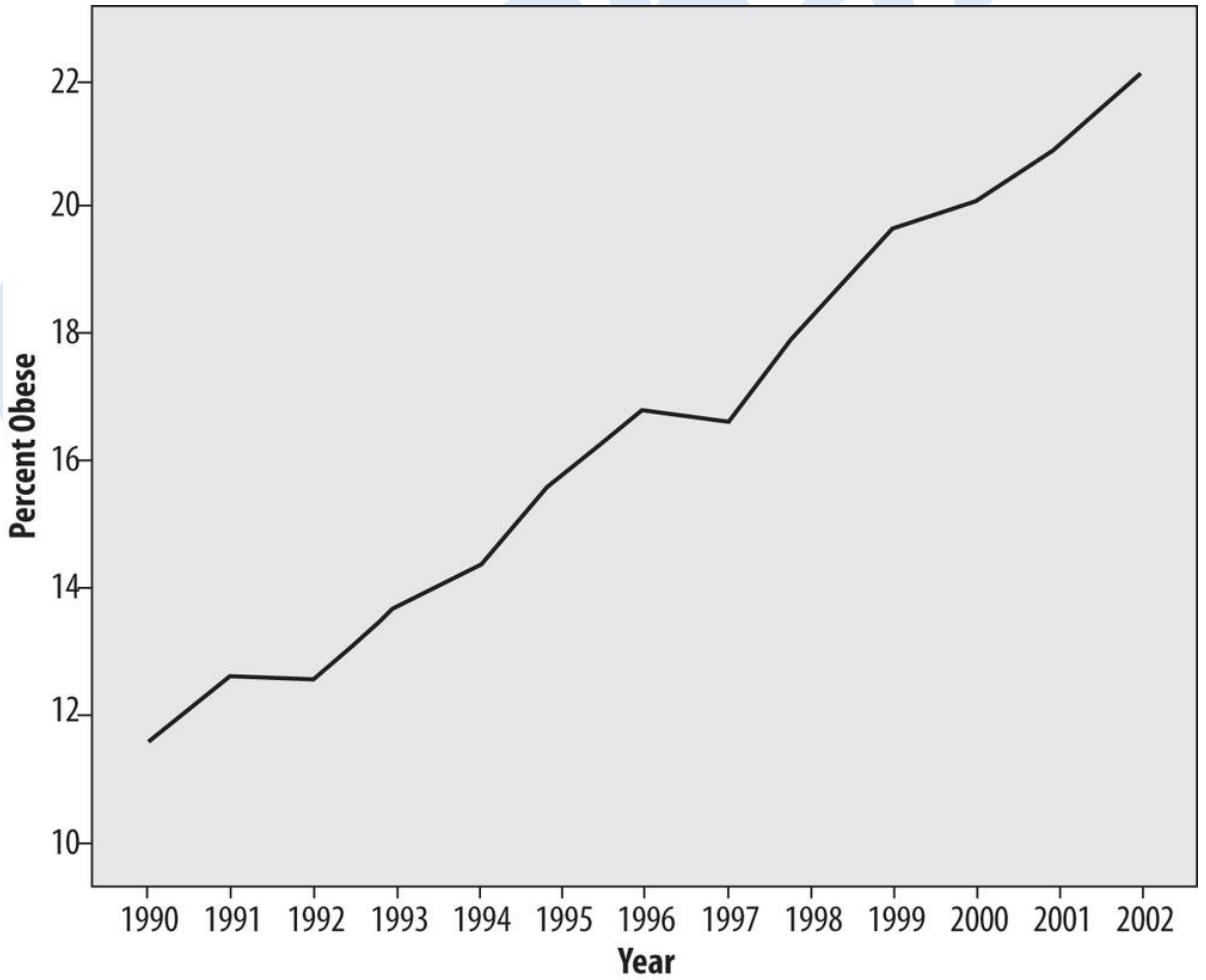
- Percentage of obesity among U.S. adults, 1990–2002 (CDC)

Year	Percent obese
1990	11.6%
1991	12.6%
1992	12.6%
1993	13.7%
1994	14.4%
1995	15.8%
1996	16.8%
1997	16.6%
1998	18.3%
1999	19.7%
2000	20.1%
2001	21.0%
2002	22.1%

Plotting Scalar Data: Line Graphs



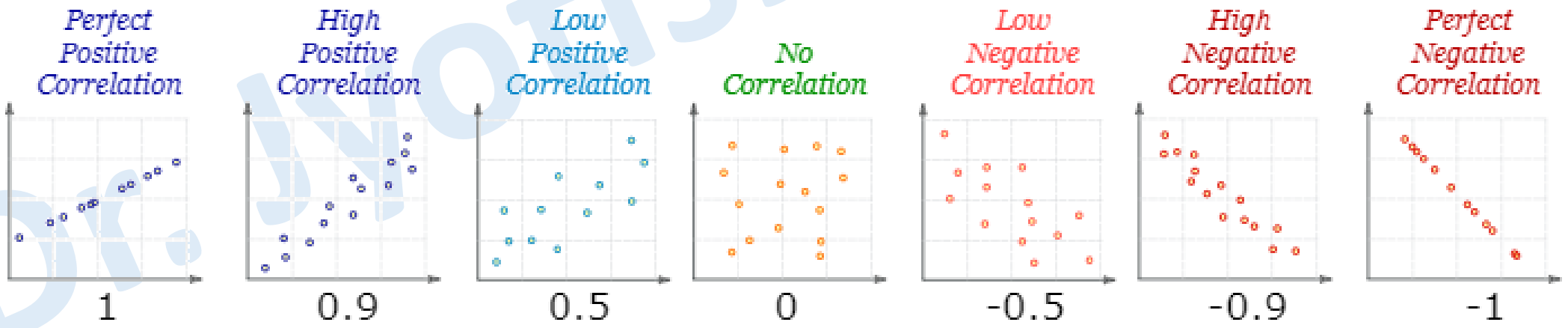
Sensible representation of the data



A larger scale and smaller range for the *y*-axis

Correlation

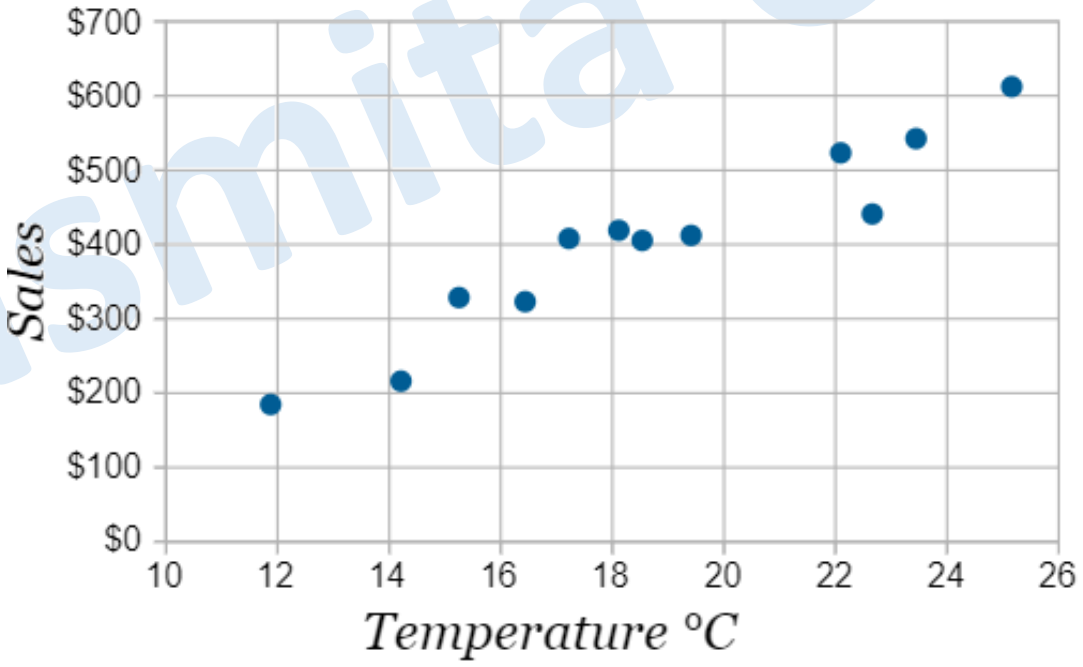
- A correlation is assumed to be **linear** (following a line).
- Correlation is **Positive** when the values **increase** together, and
- Correlation is **Negative** when one value **decreases** as the other increases



Correlation

Ice Cream Sales vs Temperature	
Temperature °C	Ice Cream Sales
14.2°	$215
16.4°	$325
11.9°	$185
15.2°	$332
18.5°	$406
22.1°	$522
19.4°	$412
25.1°	$614
23.4°	$544
18.1°	$421
22.6°	$445
17.2°	$408

The local ice cream shop keeps track of how much ice cream they sell versus the temperature on that day. Here are their figures for the last 12 days:



We can easily see that warmer weather and higher sales go together. The relationship is good but not perfect. In fact the correlation is **0.9575**

Correlation

- Here is how to calculated the first Ice Cream example

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

2 *Subtract Mean*

3 *Calculate ab, a^2 and b^2*

Temp °C	Sales	"a"	"b"	a×b	a^2	b^2
14.2	$215	-4.5	-$187	842	20.3	34,969
16.4	$325	-2.3	-$77	177	5.3	5,929
11.9	$185	-6.8	-$217	1,476	46.2	47,089
15.2	$332	-3.5	-$70	245	12.3	4,900
18.5	$406	-0.2	$4	-1	0.0	16
22.1	$522	3.4	$120	408	11.6	14,400
19.4	$412	0.7	$10	7	0.5	100
25.1	$614	6.4	$212	1,357	41.0	44,944
23.4	$544	4.7	$142	667	22.1	20,164
18.1	$421	-0.6	$19	-11	0.4	361
22.6	$445	3.9	$43	168	15.2	1,849
17.2	$408	-1.5	$6	-9	2.3	36
18.7	**$402**			**5,325**	**177.0**	**174,757**

1 *Calculate Means*

4 *Sum Up*

5 $\frac{5{,}325}{\sqrt{177.0 \times 174{,}757}} = 0.9575$

Example

- Visualize the salary data distribution after transforming the data using the following function:

$$Z = \frac{x - \mu}{\sigma}$$

where x = value in feature, μ = feature mean, and σ = feature standard deviation.

EID	Salary
604399	445000
988334	110000
301647	255000
582313	420000
339001	200000
609356	440000
1081649	150000
610842	105000
1183070	195000
794062	200000

Example

- Salary Mean: 252000
- Salary STD: 134023.2

After Transformation Plot EID and Z Value.

EID	Salary	Z Value
604399	445000	1.440049
988334	110000	-1.05952
301647	255000	0.022384
582313	420000	1.253514
339001	200000	-0.38799
609356	440000	1.402742
1081649	150000	-0.76106
610842	105000	-1.09683
1183070	195000	-0.4253
794062	200000	-0.38799

Visualization Techniques

Dr. Jyotismita Chaki

Scalar Techniques

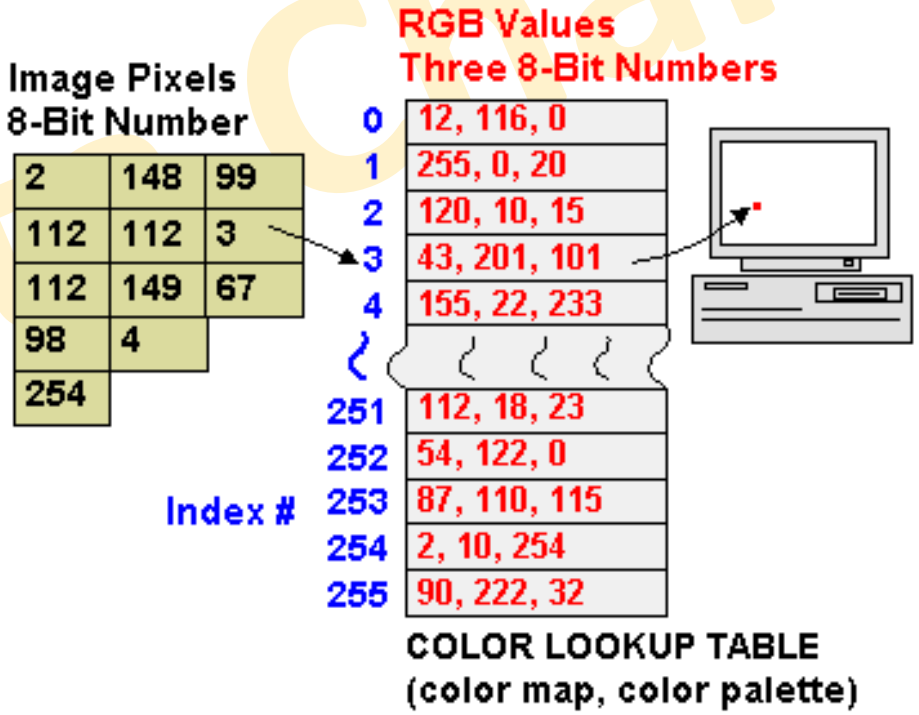
- Visualizing scalar data is frequently encountered in science, engineering, and medicine, but also in daily life.
- There exist many scalar visualization techniques, both for two-dimensional (2D) and three-dimensional (3D) datasets.
- The most popular scalar visualization techniques are: color mapping, contouring, and height plots.

Color Maps

- Color mapping is probably the most widespread visualization method for scalar data.
- Color mapping is a mapping function m : $D \rightarrow D_V$. The geometry of D_V is the same as D, but with a color that depends on the scalar data defined on D.
- Color mapping is not concerned with creating specific shapes to visualize data, but with coloring such shapes on which scalar data is defined to show the data values.
- There are several ways to define such a scalar-to-color function

Color Maps: Color look-up tables

- Color look-up tables are the simplest way to implement color mapping.
- A matrix of color data that is searched in order to change a source set of colors to a destination set.
- Color lookup tables (CLUTs) are found in graphics cards (display adapters) in order to translate the colors in an image to the colors in the hardware.
- They are also found in graphics formats, such as GIFs.



The pixels in the image contain index numbers that point to the RGB value in the color lookup table. The RGB values are the ones used by the display system.

Color Maps: Color look-up tables

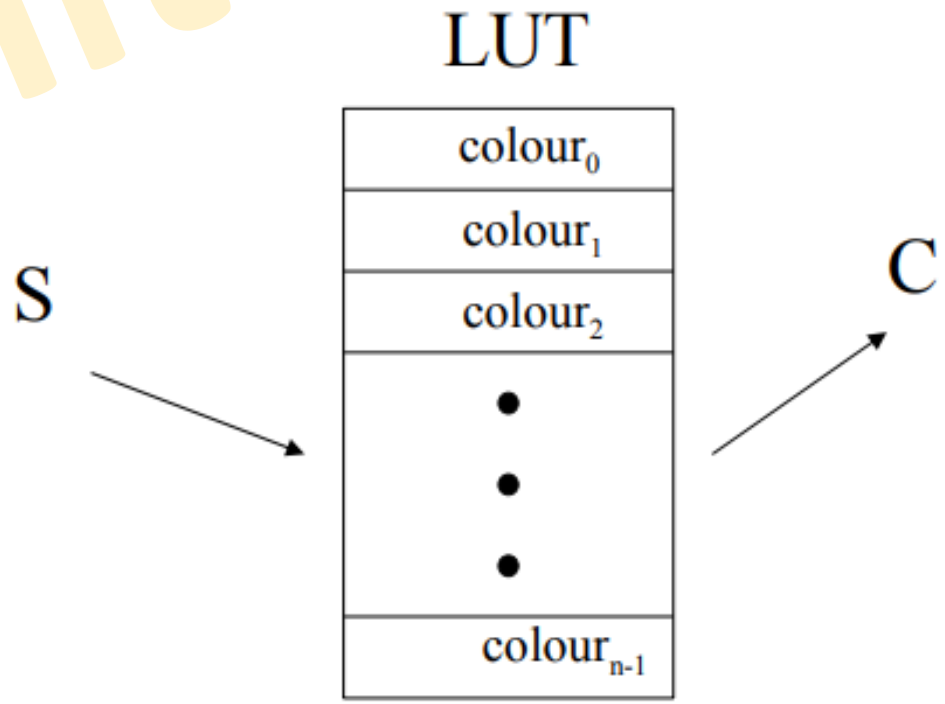
- A color look-up table C, also called a colormap, is a uniform sampling of the color-mapping function c:

$$C = \{c_i\}_{i=1..N}, \qquad \text{where } c_i = c\left(\frac{(N-i)f_{\min} + i f_{\max}}{N}\right)$$

- This equation is implemented as a table of N colors $c_1,\ldots,c_N$, which are associated with the scalar dataset values f, assumed to be in the range $[f_{min}, f_{max}]$.
- Knowing the scalar range is important, as it allows us to construct a color mapping with a clear and simple meaning:
  - the colors c_i with low indices i in the colormap represent low scalar values close to f_{min}, whereas colors with indices close to N in the colormap represent high scalar values close to f_{max}.

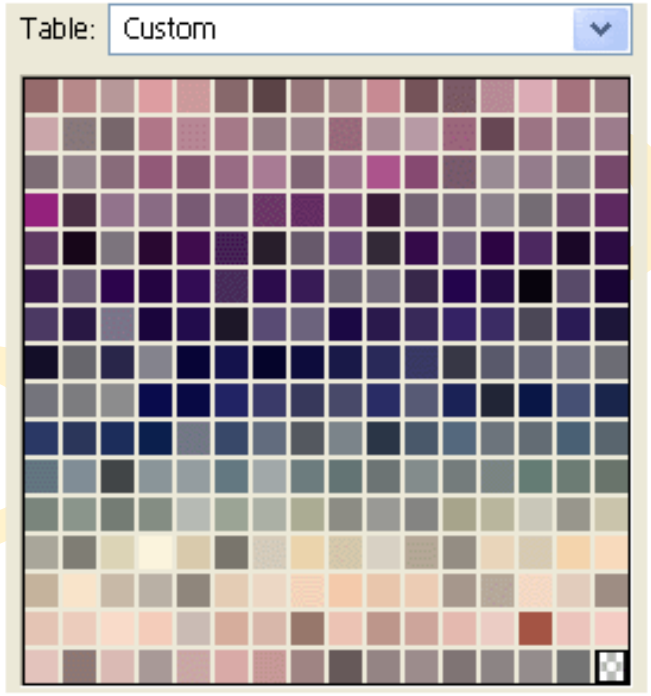
Color Maps: Color look-up tables

- Assume –
  - scalar values S_i in range {min→max}
  - n unique colours, {colour 0 ... colour n-1 } in Look-Up-Table (LUT)
- Define mapped colour C:
  - if S< min then C = $colour_{min}$
  - if S> max then C = $colour_{max}$
  - Else
    - For (j = 0; j < n ; j++)
      - if (Cj min < S < Cj max) C = Cj



Color Maps: Color look-up tables

- **24-Bit vs. 8-Bit:** Indexed color images can look nearly identical to their 24-bit originals, because the 256 most frequently used colors are identified. The 256-color palette (left) was created in Photoshop from the original 24-bit image (middle) to generate the image as 8 bits (right).

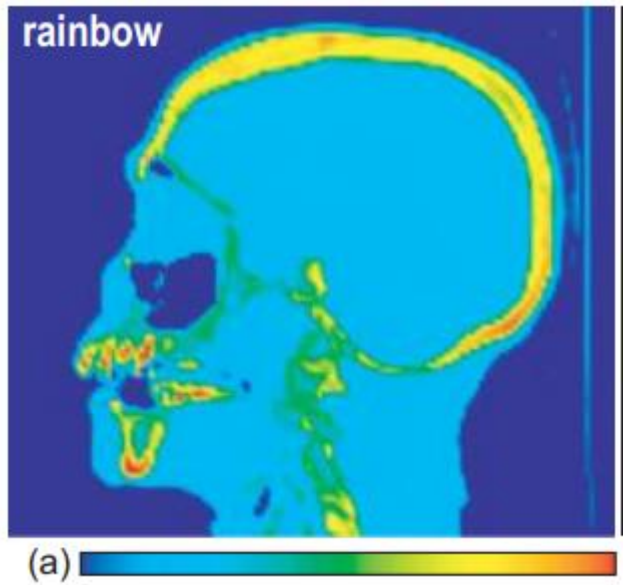


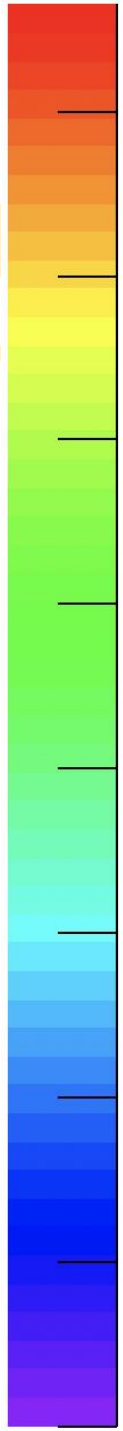
16.8 Million Colors

256 Colors

Color Maps: Rainbow colormap

- The rainbow color map is named that way because it goes through all the rainbow's colors.
- The lower values are in the deep blue range and the higher values in the reds. In between it passes trough light blue green, yellow, orange…
- It is used as a default in many visualization systems since it is easy to calculate (it is a linear interpolation between (0,0,255) and (255,0,0) in RGB color space), and because the bright colors are visually appealing.

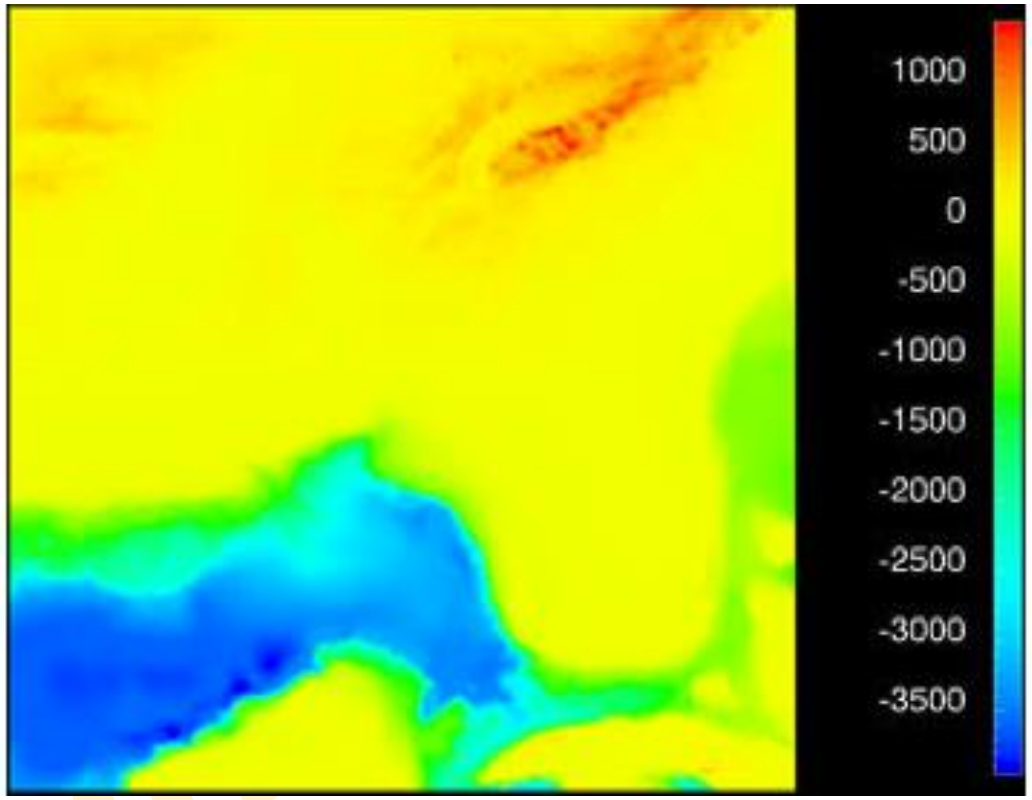


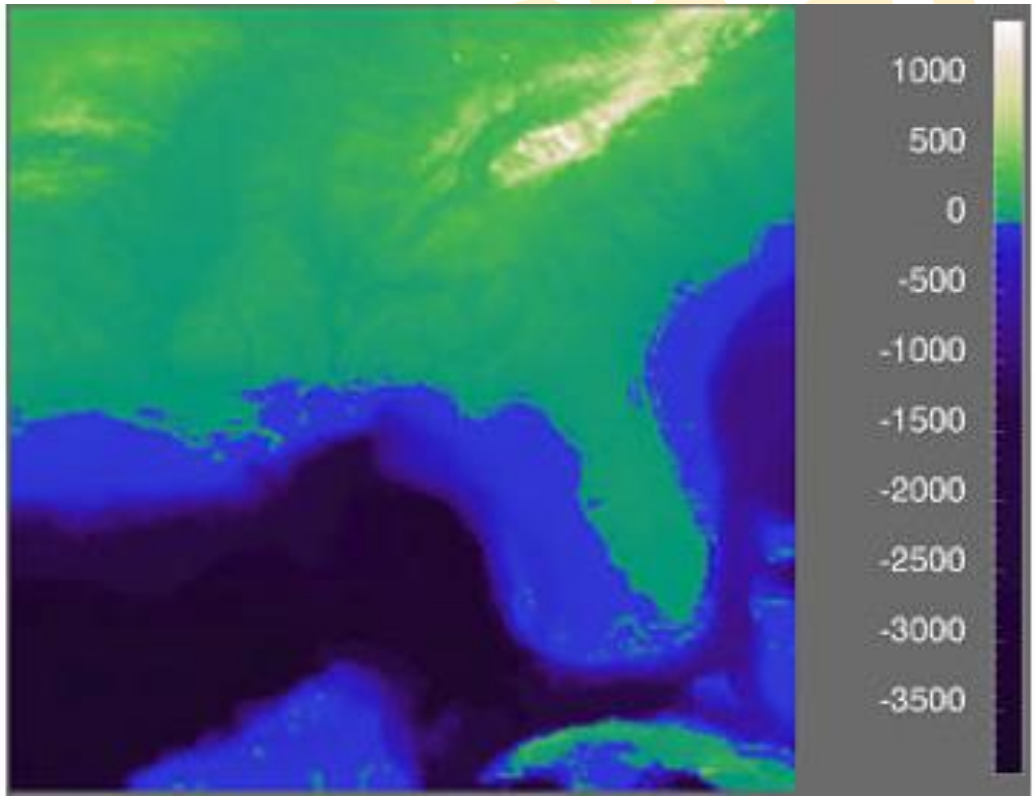


Color Maps: Rainbow colormap

- As it is explain in several visualisation articles the rainbow color map can be misleading when visualising data. The main issues with this kind of color map are:
  - structures in the data can be hidden, since not all data variations are represented visually,
  - the fact the luminance (how much white or black is mixed into the color. Higher the luminance, the brighter your color will be and as you lower your luminance, the color will appear to be darker) is not controlled can hide data,
  - it introduces gradients (a directional change in the intensity or color in an image) not related to the data,
  - it artificially divides the data into a small number of categories, one for each color.
- Despite these problems the color map is still the most widely used in pseudo-color representations.

Color Maps: Rainbow colormap

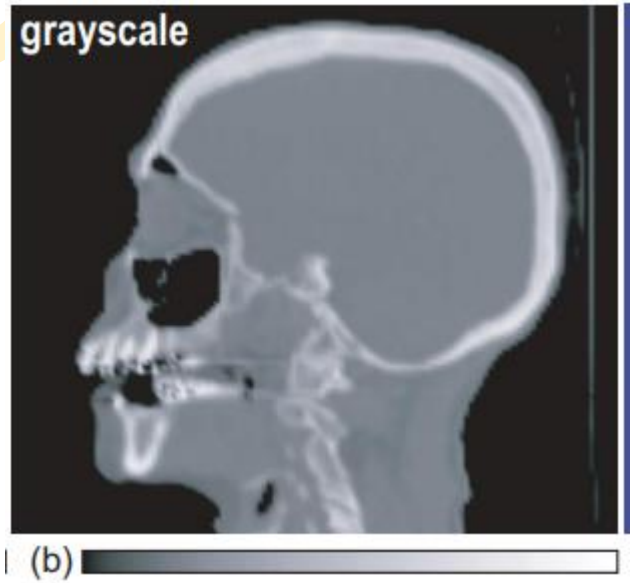




The one on left uses a rainbow color map whereas the one on right uses a more appropriate map. The rainbow map hides the well known structure of the Florida coast line. In many cases, an analyst examines a visualization to discover features. Using the Rainbow colormap can distort the perception of these features and be very misleading

Color Maps: Grayscale colormap

- Here, we map data values f linearly to luminance, or gray value, with f_{min} corresponding to black and f_{max} corresponding to white.
- Most medical specialists, but also nonspecialists, would agree that the grayscale produces a much easier-to-follow, less-confusing visualization on which details are easier to spot than when using the rainbow colormap.
- The grayscale colormap has several advantages.
  - First, it directly encodes data into luminance, and thus is has no issues regarding the discrimination of different hues.
  - Second, color ordering is natural (from dark to bright). For medical visualizations, for example, the black-to-white colormap can be more effective than the rainbow one, as these grayscales map intuitively to the ones visible in X-ray photographs.
  - Finally, rendering grayscale images is less sensitive to color reproduction variability issues.

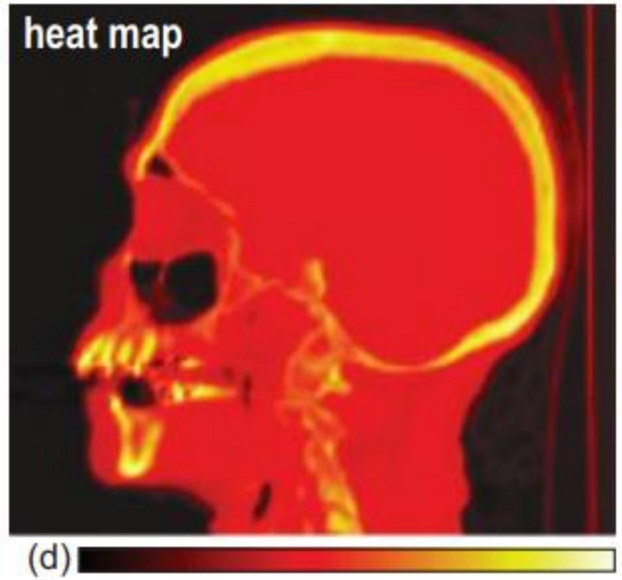


Color Maps: Two Hue colormap

- The colormap entries are obtained by linearly interpolating between two user-selected colors, blue and yellow in our case.
- The two-hue colormap can be seen as a generalization of the grayscale colormap, where we interpolate between two colors, rather than between black and white.
- If the two colors used for interpolation are perceptually quite different (such as in our case, where they differ both in hue but also in perceived luminance), the resulting colormap allows an easy color ordering, and also produces a perceptually more linear result than the rainbow colormap.
- Also, in case none of the two colors that define this colormap is too bright, the result is suitable to be used for non-compact data displayed on a white background.
- However, a disadvantage of this type of colormap is that it arguably offers less dynamic range: In general, we can distinguish between more hues (such as in the case of the rainbow colormap) than between an equal number of mixes of the same two hues.

Color Maps: Heatmap colormap

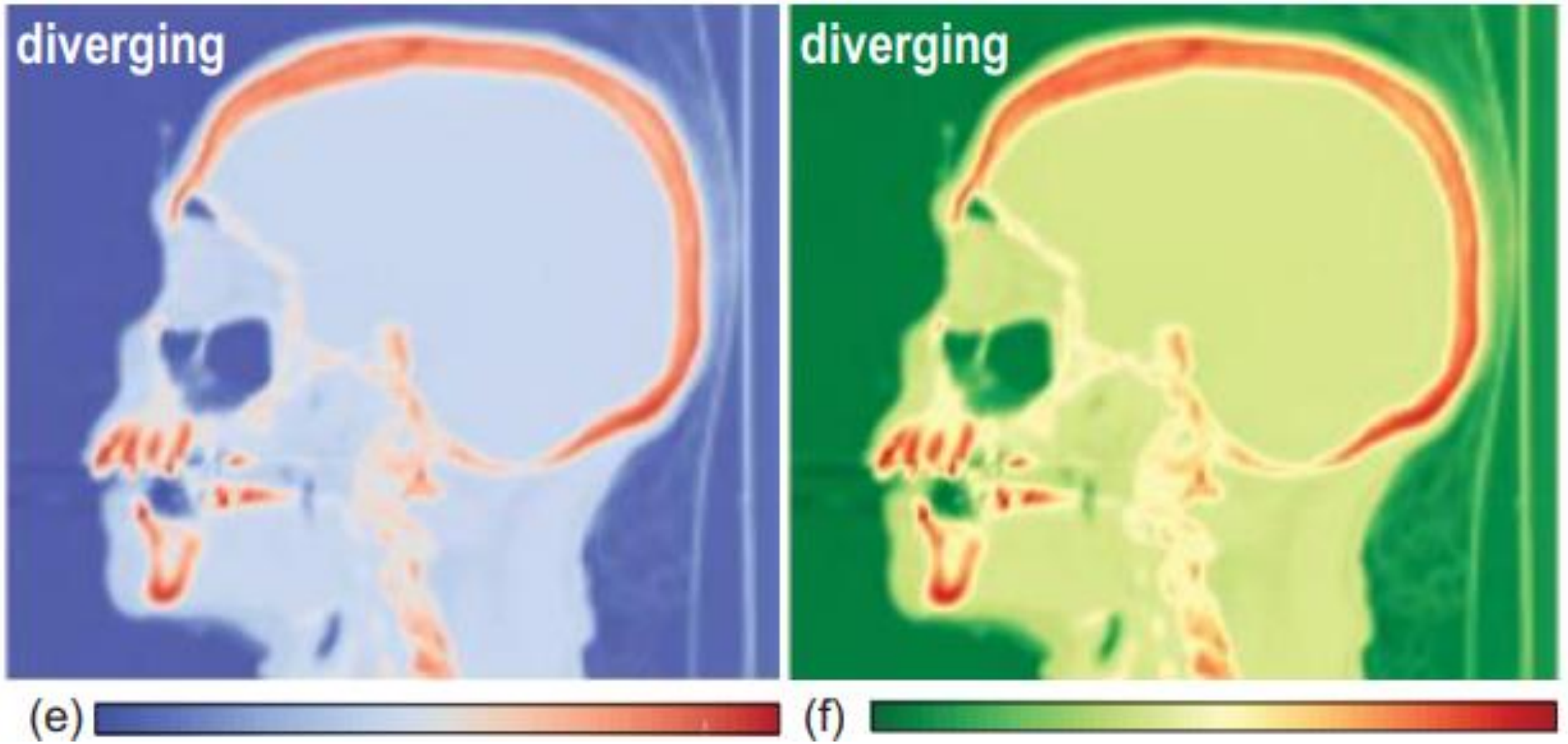
- The intuition behind its colors is that they represent the color of an object heated at increasing temperature values, with black corresponding to low data values, red-orange hues for intermediate data ranges, and yellow-white hues for the high data values respectively.
- Compared to the rainbow colormap, the heat map uses a smaller set of hues, but adds luminance as a way to order colors in an intuitive manner.
- Compared to the two-hue colormap, the heat map uses more hues, thus allowing one to discriminate between more data values.
- Eliminating the right end of the colormap (thus, using yellow rather than white for the highest data value) allows using this colormap also for non-compact domains displayed on a white background.
- Together with the grayscale map, the heat map is a popular choice for medical data visualization



Color Maps: Diverging colormap

- Diverging colormaps are constructed starting from two hues.
- However, rather than interpolating between the end colors c_{min} and c_{max}, we now add a third color c_{mid} for the data value $f_{mid} = (f_{min} + f_{max})/2$ located in the middle of the considered data range $[f_{min}, f_{max}]$, and use two piecewise-linear interpolations between c_{min} and c_{mid} and between c_{mid} and c_{max}, respectively.
- Additionally, c_{mid} is chosen so that it has a considerably higher luminance than c_{min} and c_{max}.

Color Maps: Diverging colormap



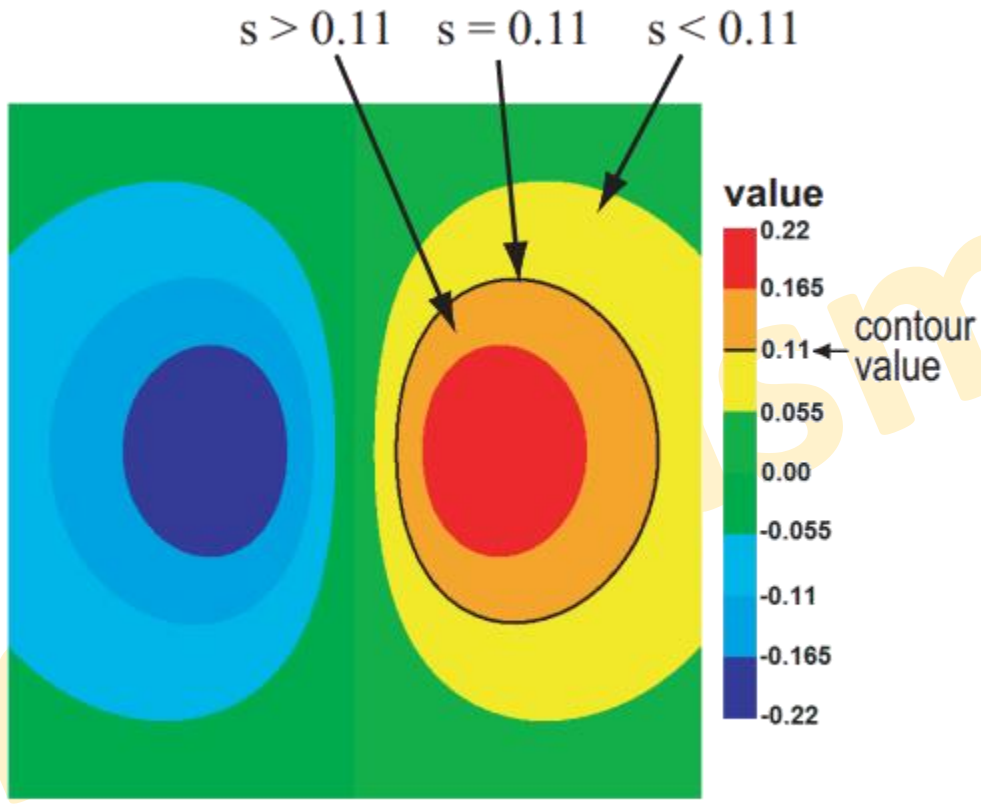
c_{min} = blue, c_{max} = red, and c_{mid} = white.

c_{min} = green, c_{max} = red, and c_{mid} = brightyellow

Contouring

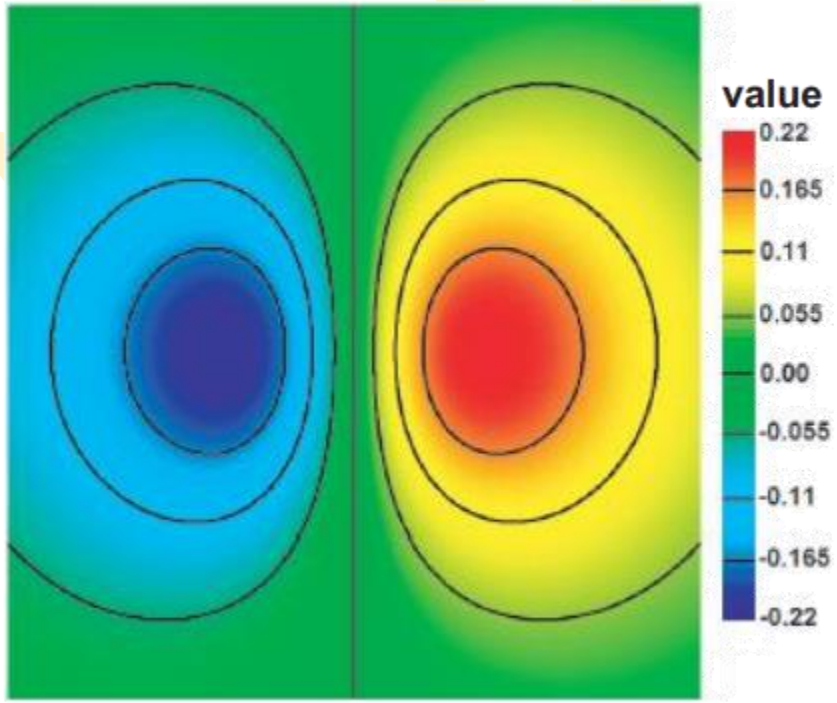
- To understand contouring, think of the meaning of the sharp color transitions that separate the color bands.
- Consider, for instance, the transition between the yellow and orange bands; that is, all points in the figure that are on the border separating these two colors.
- As can be seen from the associated color legend, points in the yellow band have scalar values s below 0.11, whereas the points in the orange band have scalar values s above 0.11.
- Hence, the points located on the color border itself have the scalar value s = 0.11.
- For this reasoning to hold, we must assume that our dataset does not exhibit a sudden “jump”.
- This holds for all datasets that represent the sampling of a continuous signal.
- Points located on such a color border, drawn in black in Figure, are called a contour line, or isoline.

Contouring



(a) one contour (s=0.11)

Contouring and Color Banding



(b) 7 contours (s ∈ [-0.165,0.165])

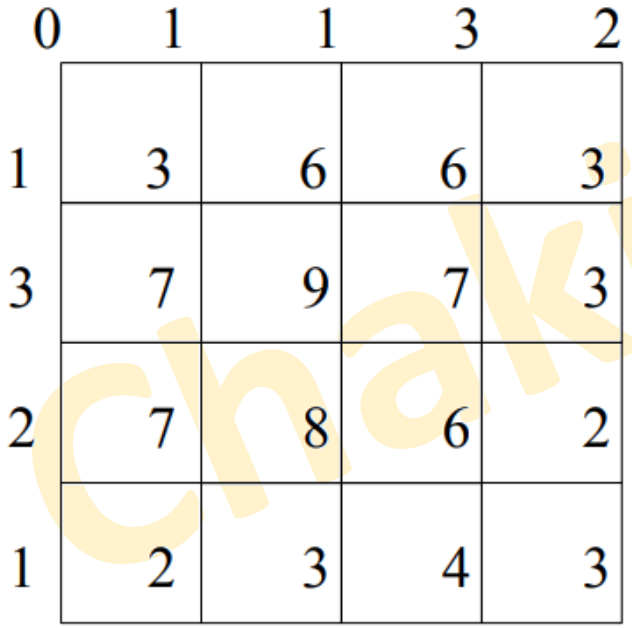
Contouring and Colormapping

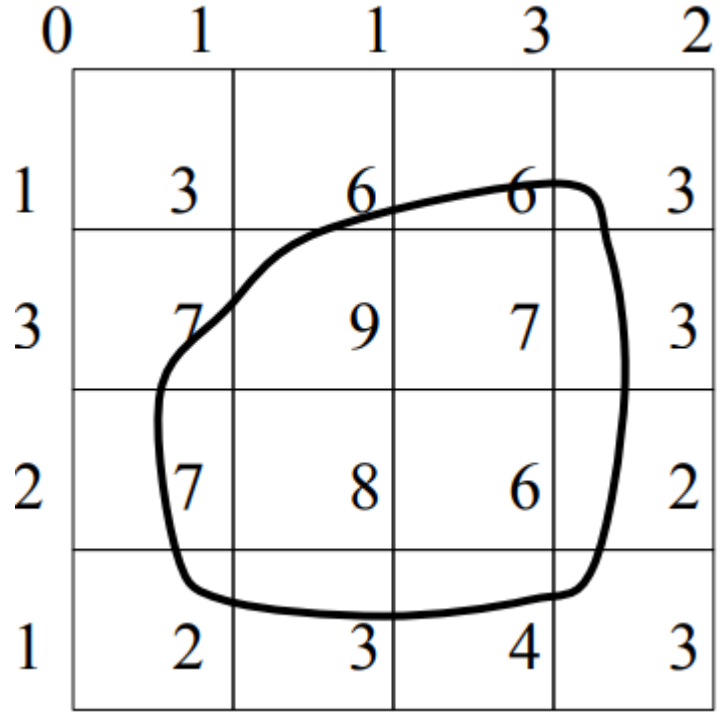
Contouring

- Formally, a contour line C is defined as all points p in a dataset D that have the same scalar value, or isovalue s(p) = x:
  - $C(x) = \{p \in D | s(p) = x\}$.
- Contour lines are drawn on land maps to explicitly indicate all points that have the same altitude.
- For 2D dataset, a contour line is called an **isoline**.
- For 3D datasets, contours are 2D surfaces called **isosurfaces**.

Contouring: Computation: Edge Tracking

- Data : 2D structured grid of scalar values (top)
- Difficult to visualise transitions in data
  - – use contour at specific scalar value to highlight transition
- Approach 1: Edge Tracking
  - Select scalar value
    - – corresponds to contour line – i.e. contour value, e.g. 5
  - Interpolate contour line through the grid corresponding to this value (bottom)
  - Advantages : produces correctly shaped line
  - Dis-advantages : need to search for other contours





Contour: Computation: Edge Tracking

- How can we compute contours, given a discrete, sampled dataset D?
- Since this dataset is defined as a set of cells carrying node or cell scalar data, plus additional basis functions, it is natural to try to construct contours in the same discrete cell space.
- In the actual construction technique, we shall use an important property of isolines.
- The basic algorithm for constructing an isoline is quite simple.
- The principle is illustrated by the simple 5 × 5 cell grid in Figure, where we construct the isoline for the value v = 0.48.

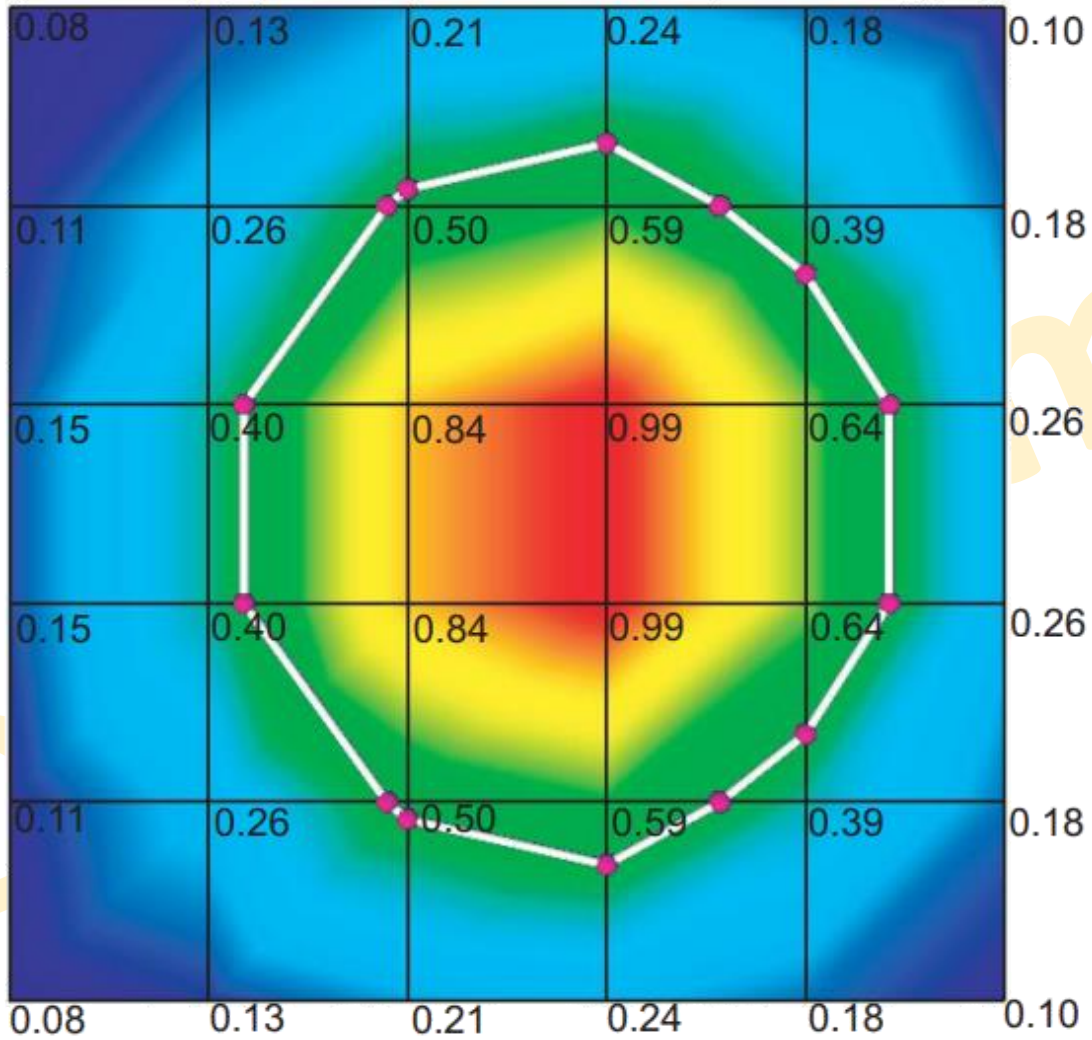
Contour: Computation: Edge Tracking

- For every cell c of the dataset, we test whether the isoline intersects the respective cell, as follows.
  - For every edge e = (pi, pj) of the cell c, we test whether the isoline value v is between the scalar vertex attributes vi and vj corresponding to the edge end points pi and pj.
  - If the test succeeds, the isoline intersects e at a point q.

$$q = \frac{p_i(v_j - v) + p_j(v - v_i)}{v_j - v_i}.$$

  - We repeat the previous procedure for all edges of our current cell and finally obtain a set of intersection points S = {qi} of the isoline with the cell.
  - Next, we must connect these points together to obtain the actual isoline.

Contour: Computation: Edge Tracking



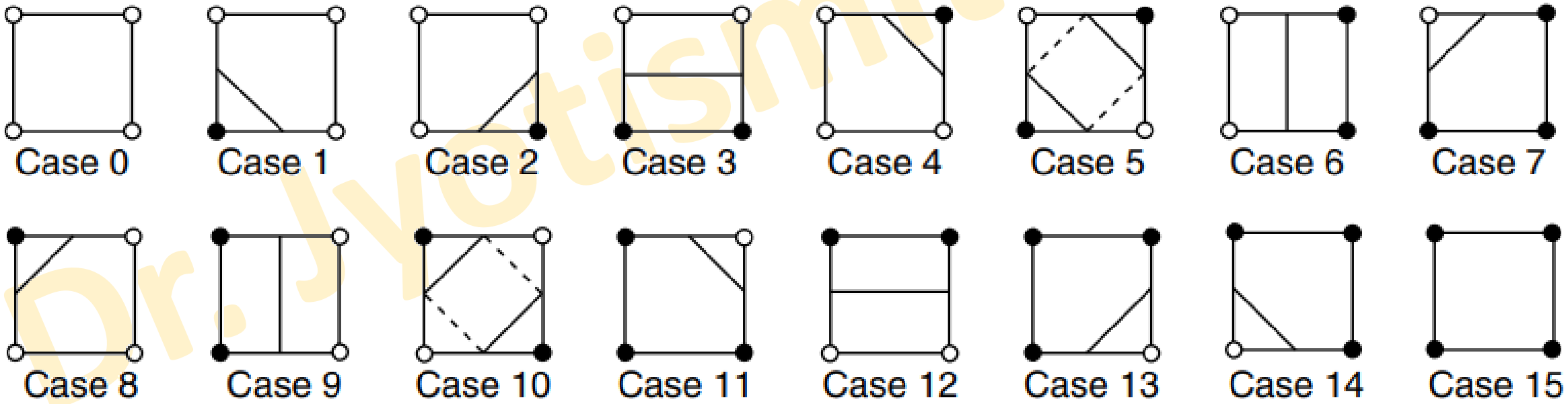
Constructing the isoline for the scalar value v = 0.48. The numbers in the figure indicate scalar values at the grid vertices.

Contour: Computation: Marching Squares (2D)

- Works only on structured data
- The basic assumption of these techniques is that a contour can pass through a cell in only a finite number of ways.
- A **case table** is constructed that enumerates all possible **topological states** of a cell, given combinations of scalar values at the cell points.
- The **number of topological states** depends on the number of cell vertices and the number of inside/outside relationships a vertex can have with respect to the contour value.
- A vertex is considered **inside** a contour if its scalar value is larger than the scalar value of the contour line.
- Vertices with scalar values less than the contour value are said to be **outside** the contour.

Contour: Computation: Marching Squares (2D)

- For example, if a cell has four vertices and each vertex can be either inside or outside the contour, there are 24 ¼ 16 possible ways that the contour passes through the cell.



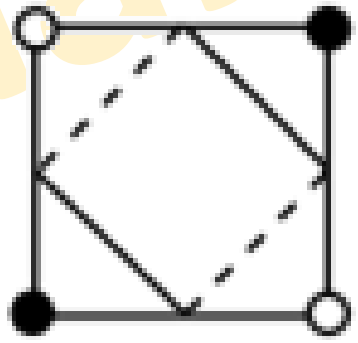
Sixteen different marching squares cases. Dark vertices indicate scalar value is above contour value. Cases 5 and 10 are ambiguous.

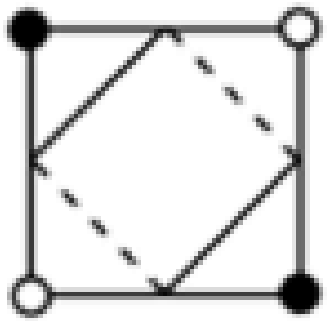
Contour: Computation: Marching Squares (2D)

- In summary, the marching algorithms proceed as follows:
  1. Select a cell.
  2. Calculate the inside/outside state of each vertex of the cell.
  3. Create an index by storing the binary state of each vertex in a separate bit.
  4. Use the index to look up the topological state of the cell in a case table.
  5. Calculate the contour location (via interpolation) for each edge in the case table.

Contour: Computation: Marching Squares (2D)

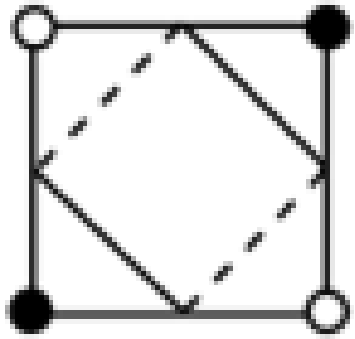
- An important issue is **contouring ambiguity**.
- Careful observation of marching squares cases 5 and 10 shows the ambiguity.
- For each ambiguous case, we implement one of the two possible cases. The choice for a particular case is independent of all other choices. Depending on the choice,
  - The contour may either extend or break the current contour.
- Either choice is acceptable since the resulting contour lines will be continuous and closed (or will end at the dataset boundary).


Case 5

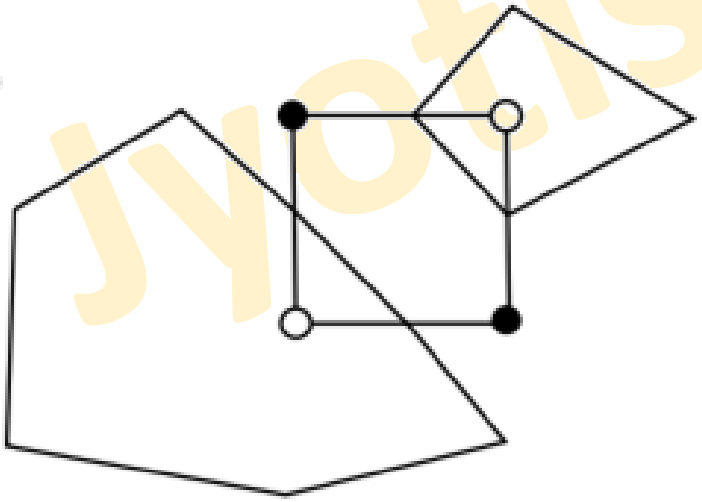

Case 10

Contour: Computation: Marching Squares (2D)

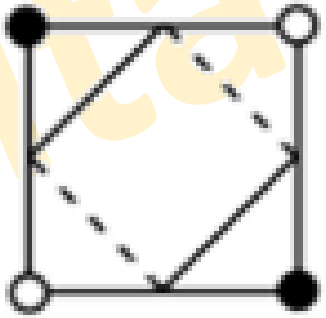
- Choosing a particular contour case will (a) break or (b) join the current contour.



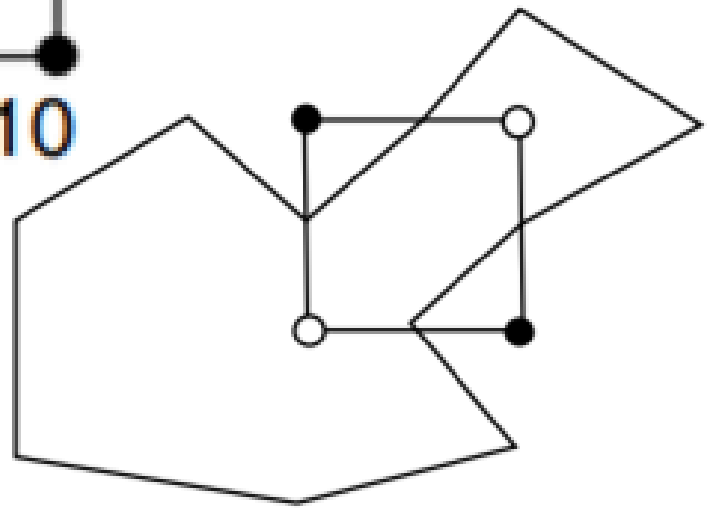
Case 5



(a) Break contour



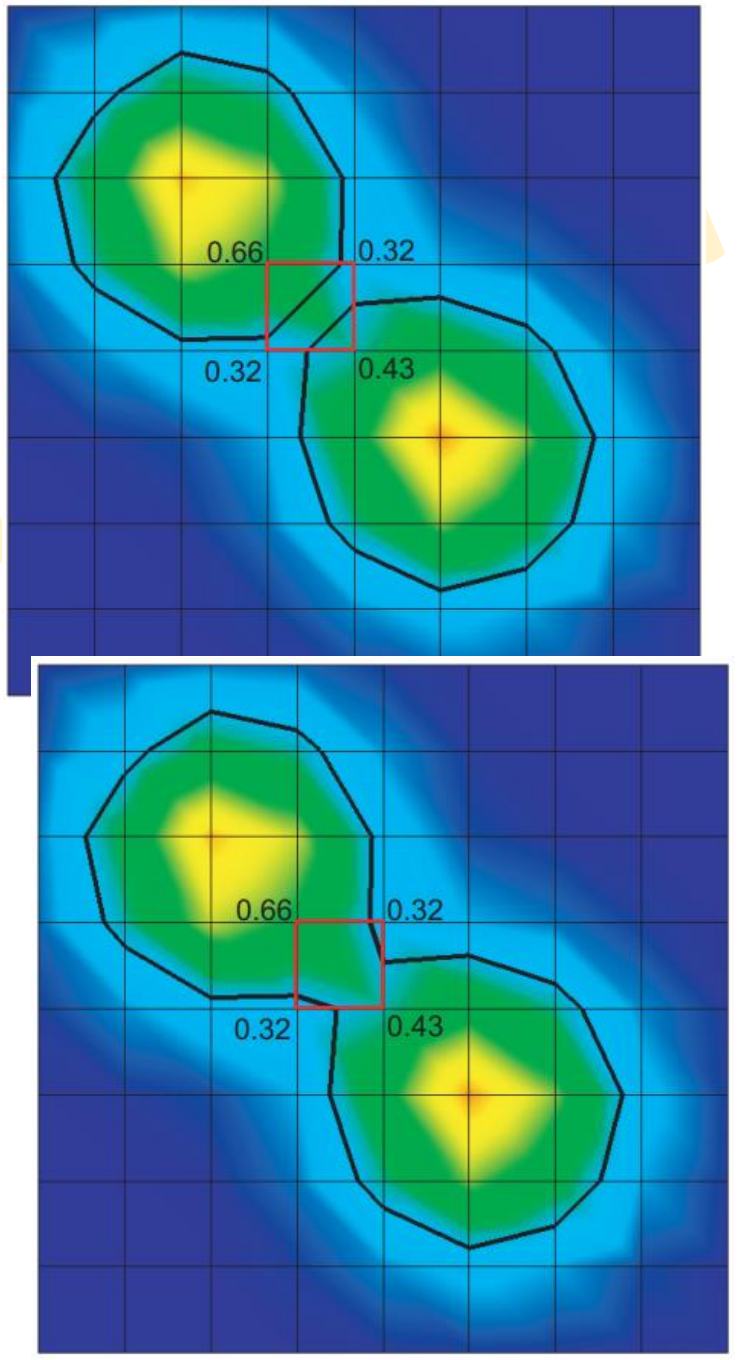
Case 10



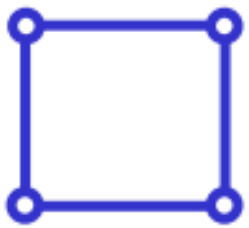
(b) Join contour

Contour: Computation: Marching Squares (2D)

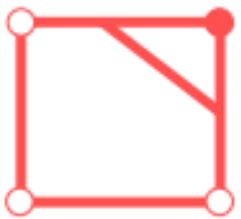
- If S contains exactly two points, there is no problem.
- However, S can contain more points, as illustrated in the Figure for the quad cell marked in red, which has four intersection points.
- In this case, exactly two possibilities exist for connecting the four intersection points, shown in the top and bottom images in Figure.
- Contour ambiguity for a quad cell (drawn in red). The isovalue is equal to 0.37.



Contour: Computation: Marching Squares (2D)



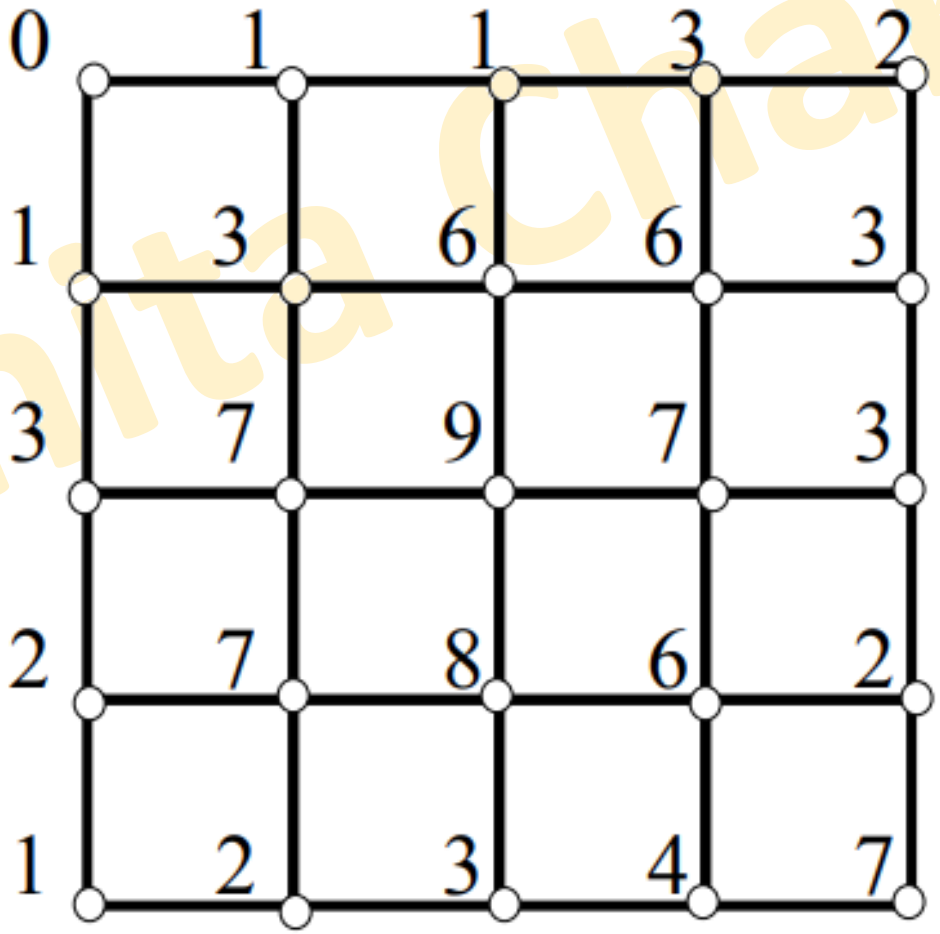
No intersection.



Contour intersects 1 edge

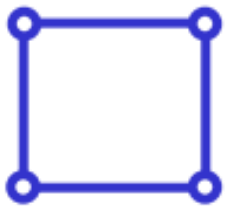
Contour intersects 2 edges

Ambiguous case.

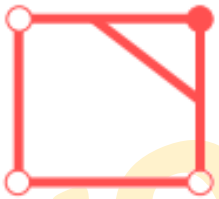


Contour: Computation: Marching Squares (2D)

- Decide whether each vertex is inside or outside contour

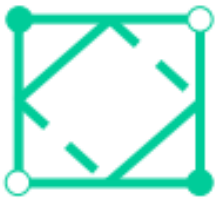


No intersection.

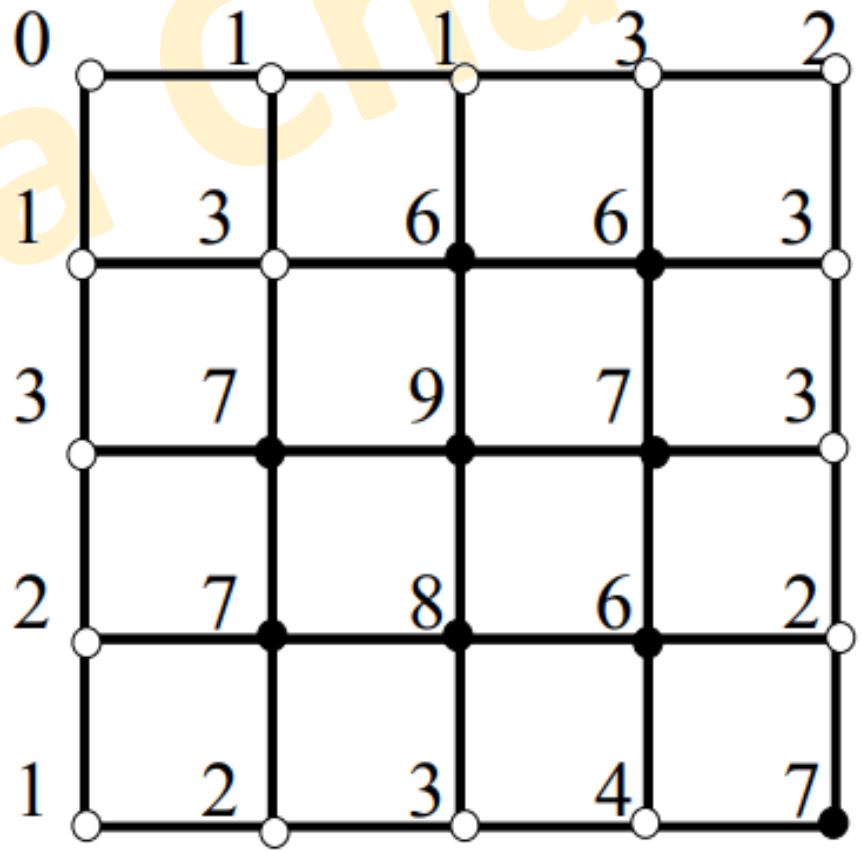


Contour intersects 1 edge

Contour intersects 2 edges



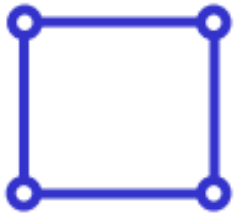
Ambiguous case.



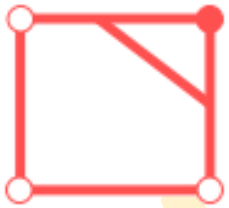
Contour value =5

Contour: Computation: Marching Squares (2D)

- Classify each cell as one of the cases

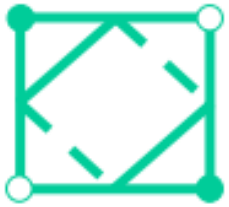


No intersection.

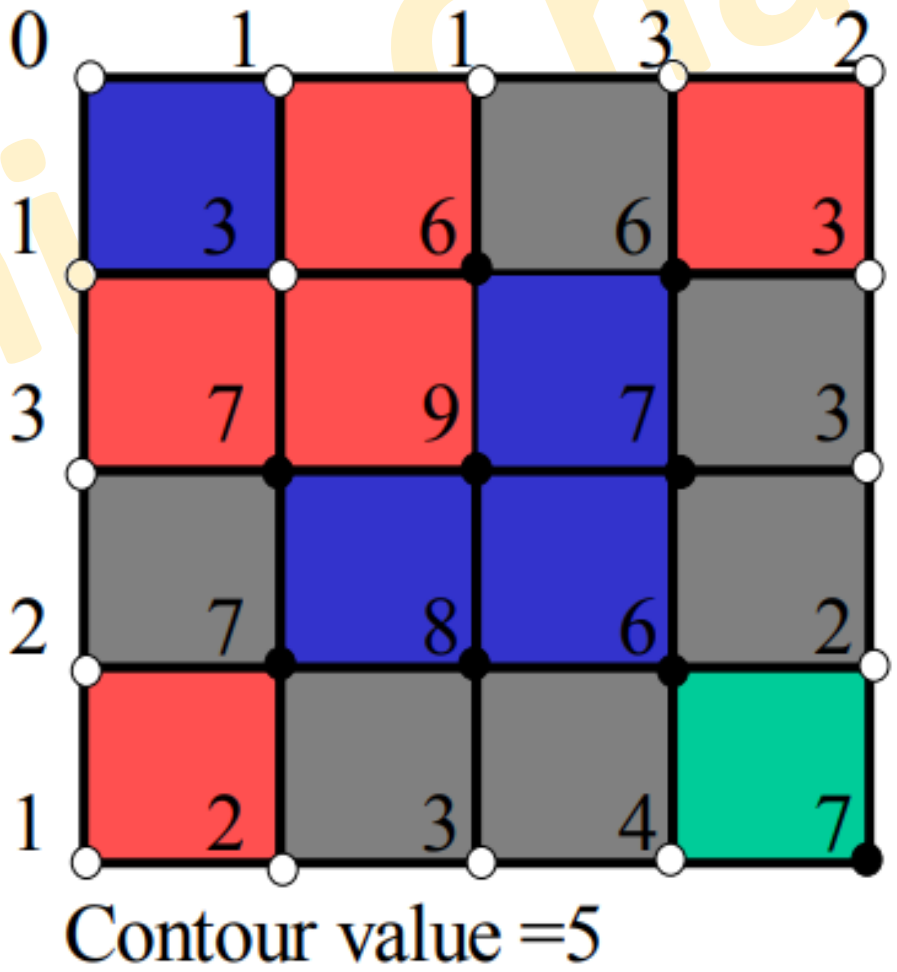


Contour intersects 1 edge

Contour intersects 2 edges



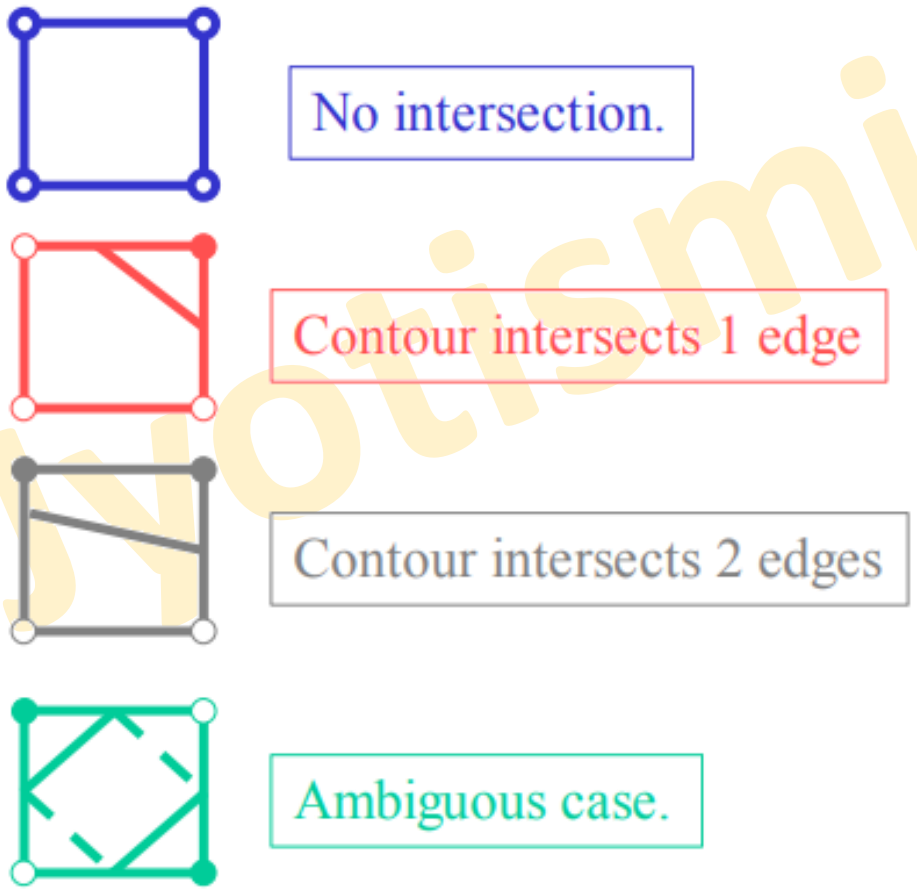
Ambiguous case.

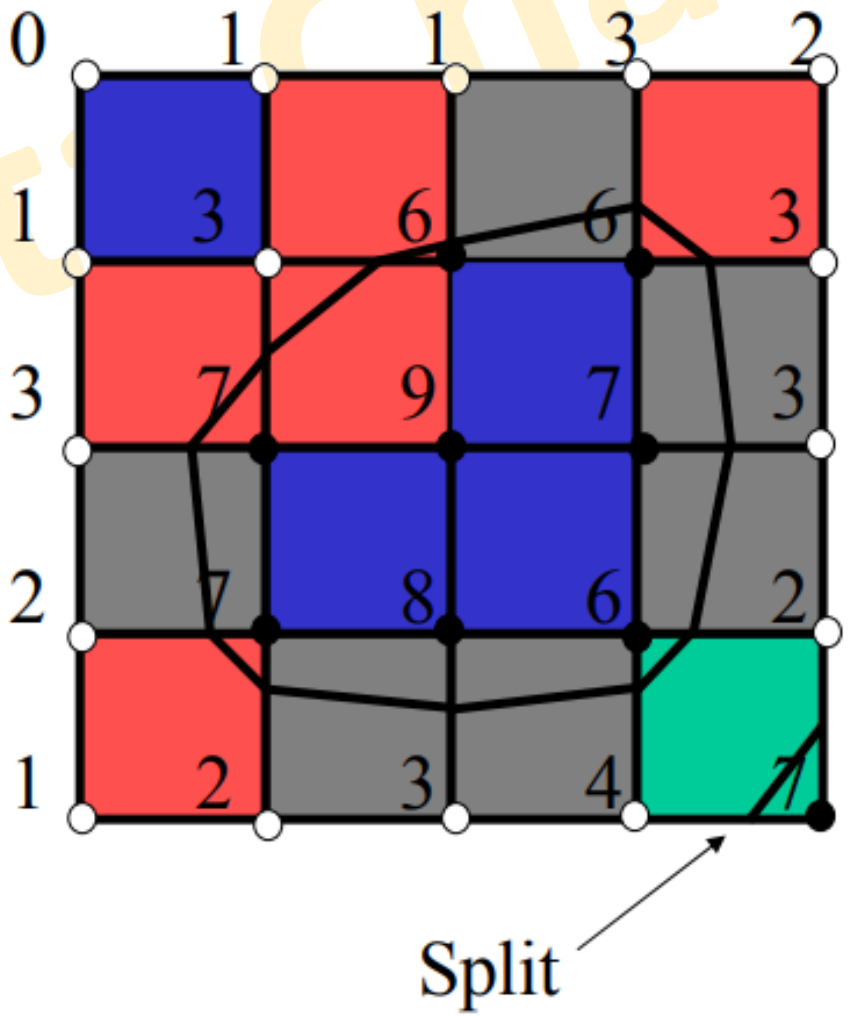


Contour value =5

Contour: Computation: Marching Squares (2D)

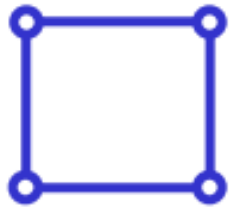
- Determine the edges that are intersected



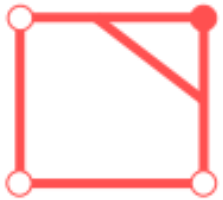


Contour: Computation: Marching Squares (2D)

- Finally : resolve any ambiguity

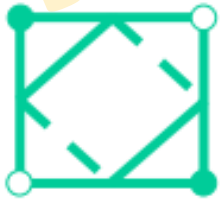


No intersection.

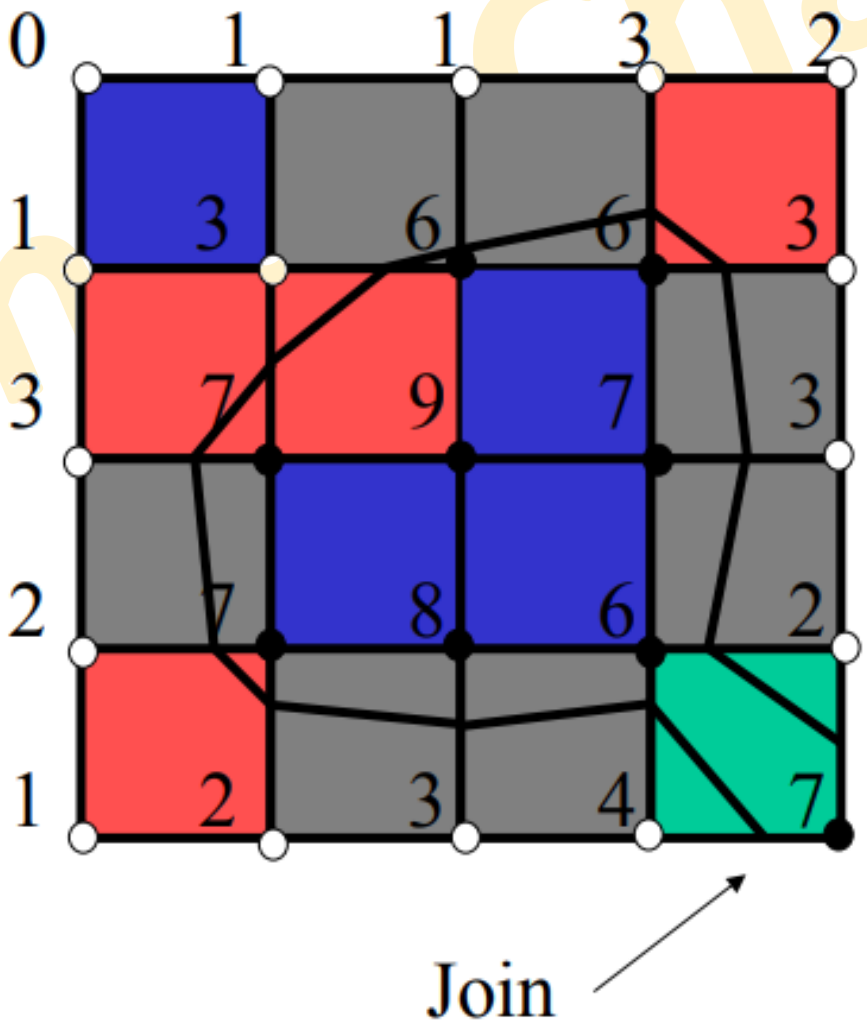


Contour intersects 1 edge

Contour intersects 2 edges



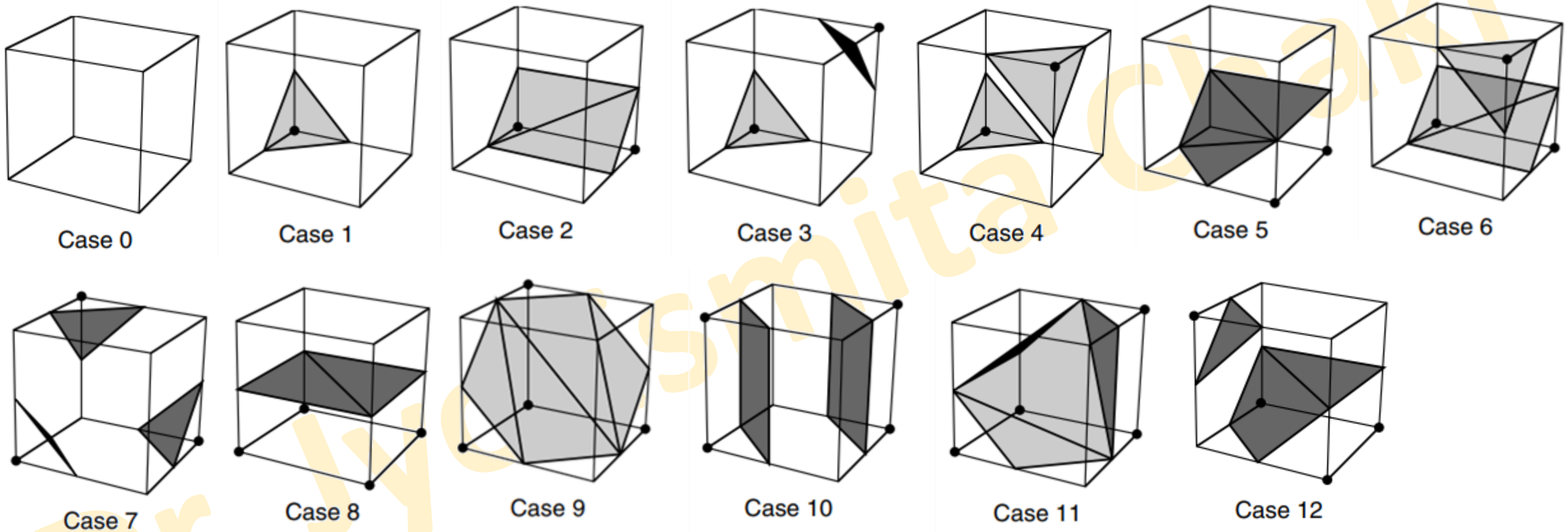
Ambiguous case.



Contour: Computation: Marching Cubes (3D)

- The 3D analogy of marching squares is marching cubes.
- Here, there are 256 different combinations of scalar value, given that there are eight points in a cubical cell (i.e., 28 combinations).
- Figure shows these combinations reduced to 15 cases by arguments of symmetry.
- This is the so-called marching cubes case table.

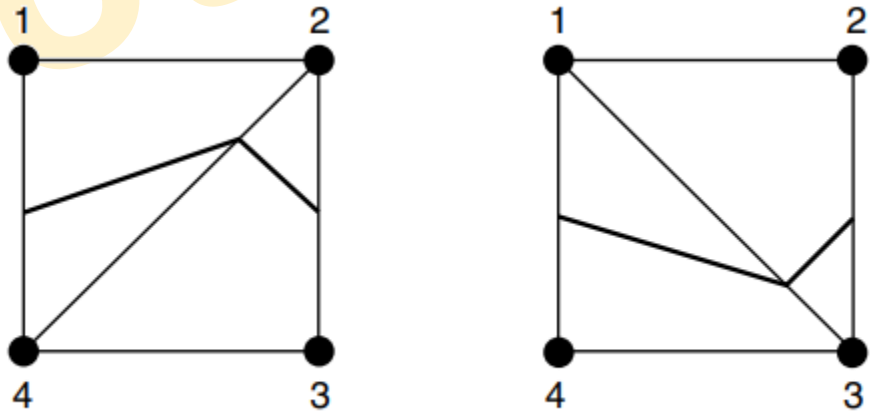
Contour: Computation: Marching Cubes (3D)



Marching cubes cases for 3D isosurface generation

Contour: Computation: Marching Cubes (3D)

- Several different approaches have been taken to remedy the problem of **contouring ambiguity** in 3D.
- One approach tessellates the cubes with **tetrahedra** and uses a marching **tetrahedra technique**.
- This choice may result in artificial ‘‘**bumps**’’ in the **isosurface** because of interpolation along the face diagonals.

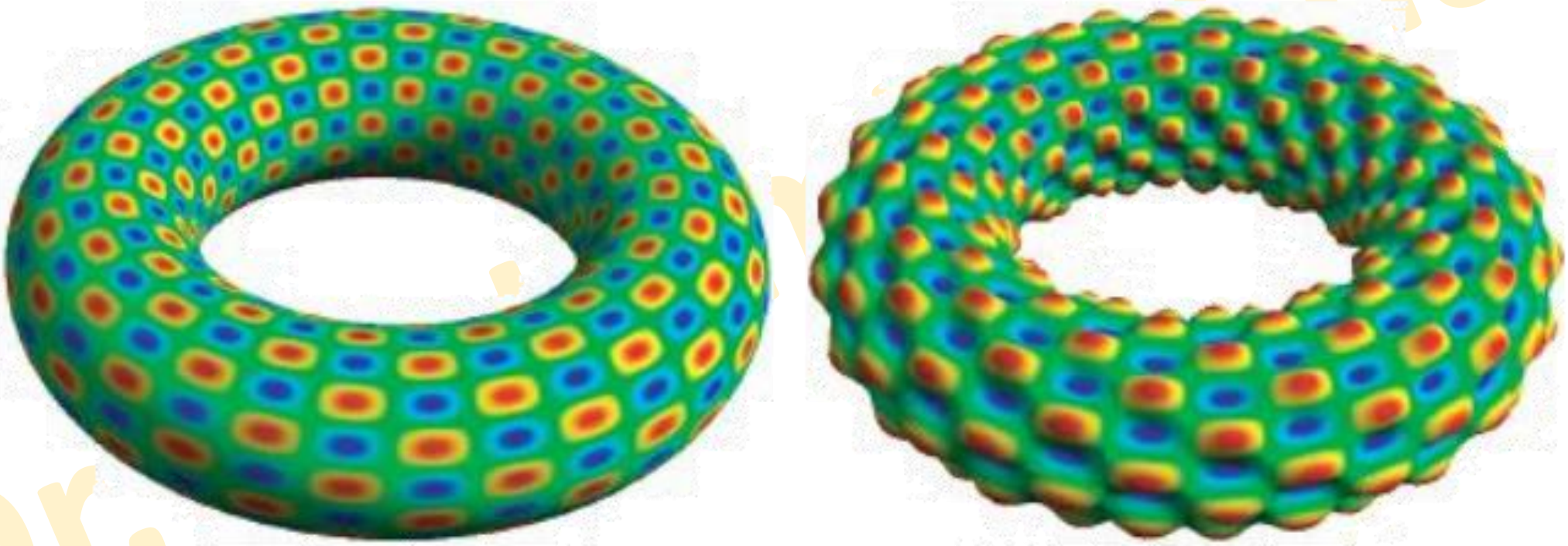


Iso-value = 2.5

Height plots

- Height plots, also called **elevation** or **carpet plots.**
- Given a two-dimensional surface Ds ∈ D, part of a scalar dataset D, height plots can be described by the mapping operation
  - $m : Ds \rightarrow D, m(x) = x + s(x)n(x), \forall x \in Ds$
- where s(x) is the scalar value of D at the point x and n(x) is the normal to the surface Ds at x.
- In their most common variant, height plots warp a planar surface Ds.
- The values of the scalar dataset on the torus surface are shown with a blue-to-red colormap in Figure (a).
- Figure (b) shows the height plot of these scalar values, done by warping the torus surface with the scalar values along the surface normal.

Height plots

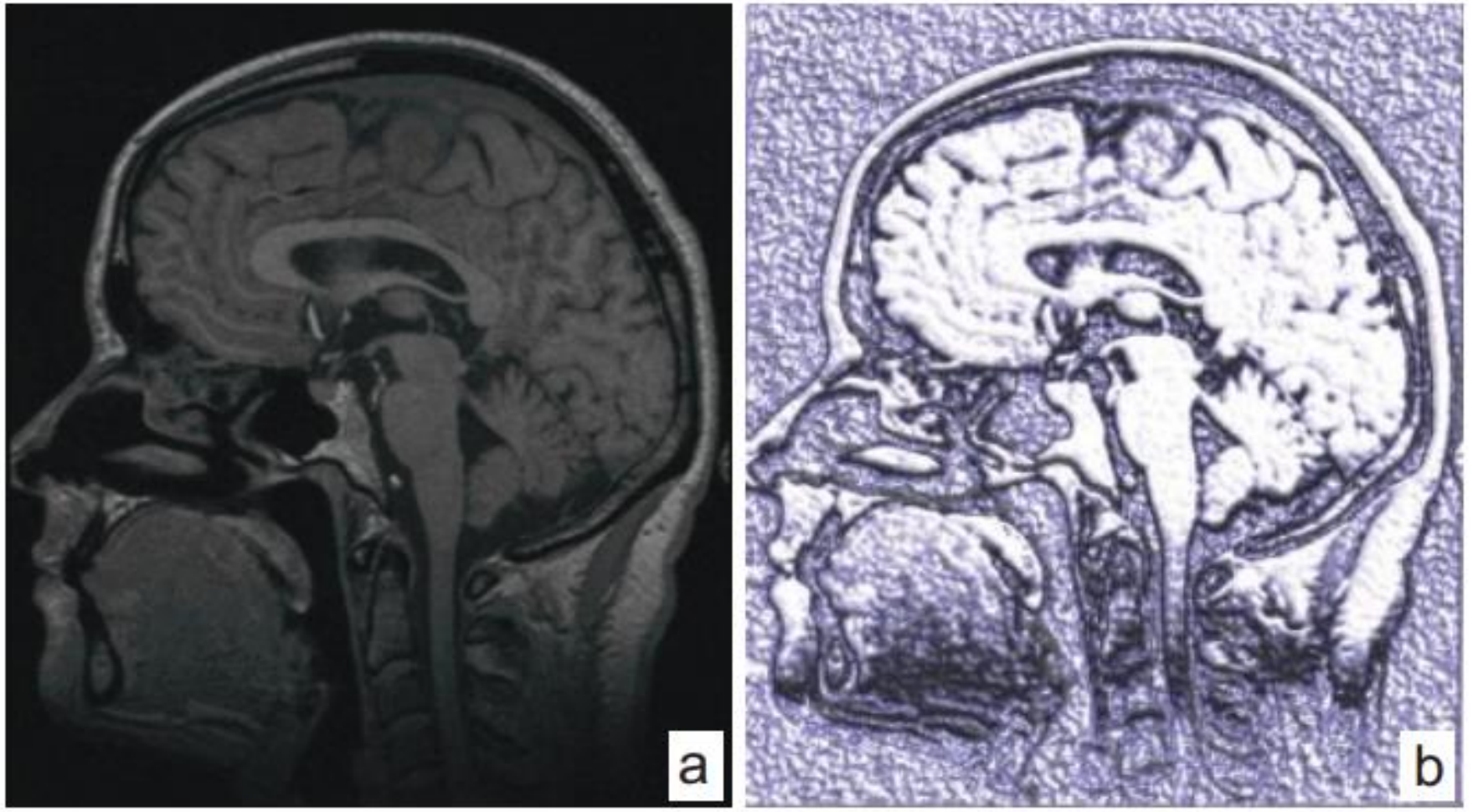


(a) (b)

Height plots

- Figure (a) shows a 2D scalar plot of a brain CT slice, displayed with a grayscale colormap.
- Here, scalar values encode tissue density, with white corresponding to the hardest structures (bone) and black corresponding to air.
- Since a linear scalar-to-grayscale colormap was used here, high-contrast borders between regions of different densities are easily seen.
- Also, large differences of absolute data values are easily seen—we can, for instance, locate the hardest and softest tissues within the given scan by looking for the brightest, respectively darkest, areas in the image.
- However, small-scale data variations are not easily visible, since the dynamic range of a grayscale colormap on a typical computer screen has only 256 different values.
- Figure (b) shows the same dataset, this time displayed using a height plot, visualized from above, and with a directional value from a slightly oblique direction.

Height plots



Plotting Scalar Data

- Table 1: CDC/WHO categories for BMI

BMI range	Category
< 18.5	Underweight
18.5–24.9	Normal weight
25.0–29.9	Overweight
30.0 and above	Obese

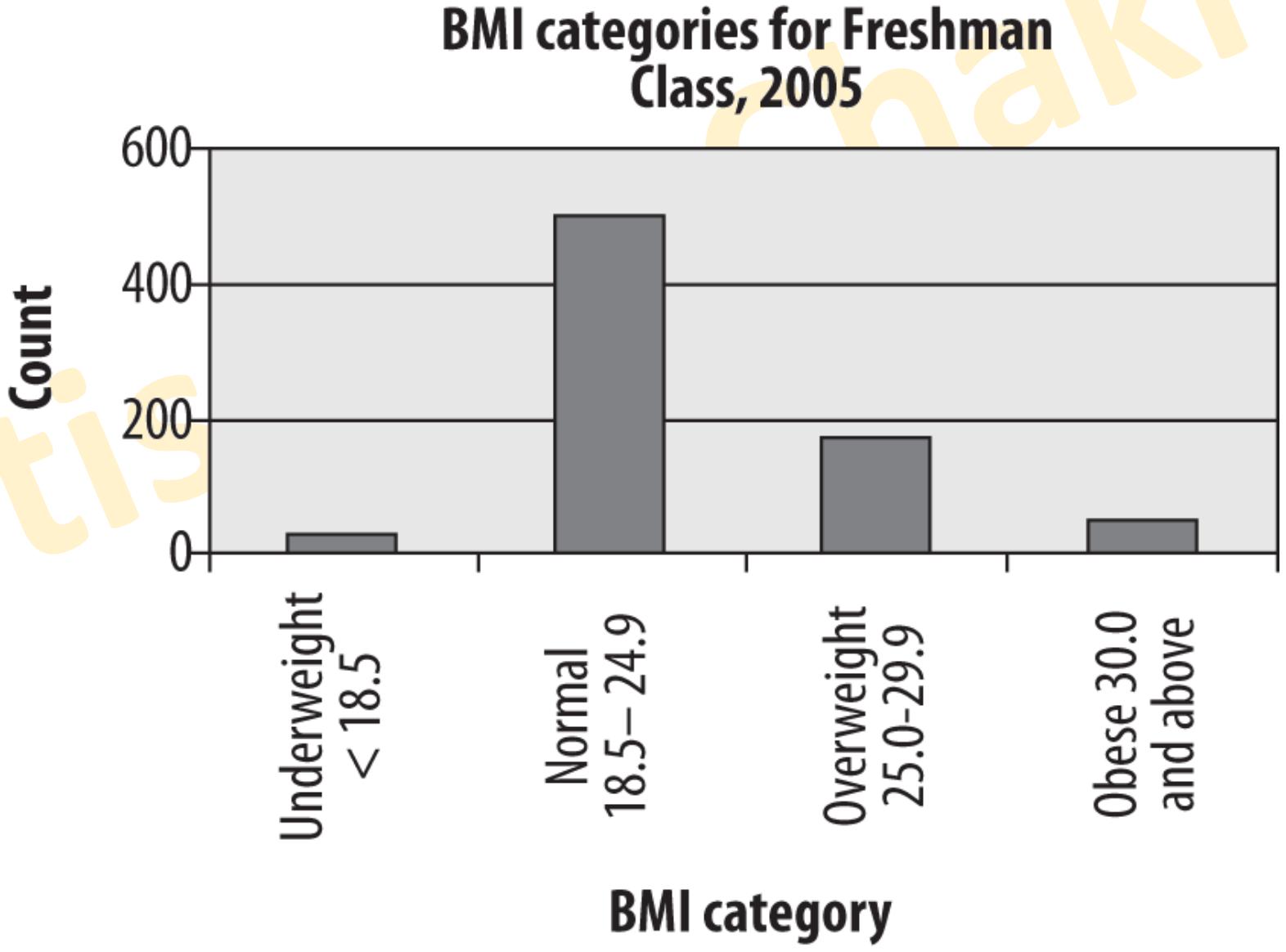
- Table 2: Distribution of BMI in the freshman class of 2005

BMI range	Number
< 18.5	25
18.5–24.9	500
25.0–29.9	175
30.0 and above	50

This table presents raw numbers or counts for each category, which are sometimes referred to as *absolute frequencies*

Plotting Scalar Data: Barplot

- The freshman BMI information presented in a bar chart: Absolute frequency of BMI categories in freshman class



Plotting Scalar Data

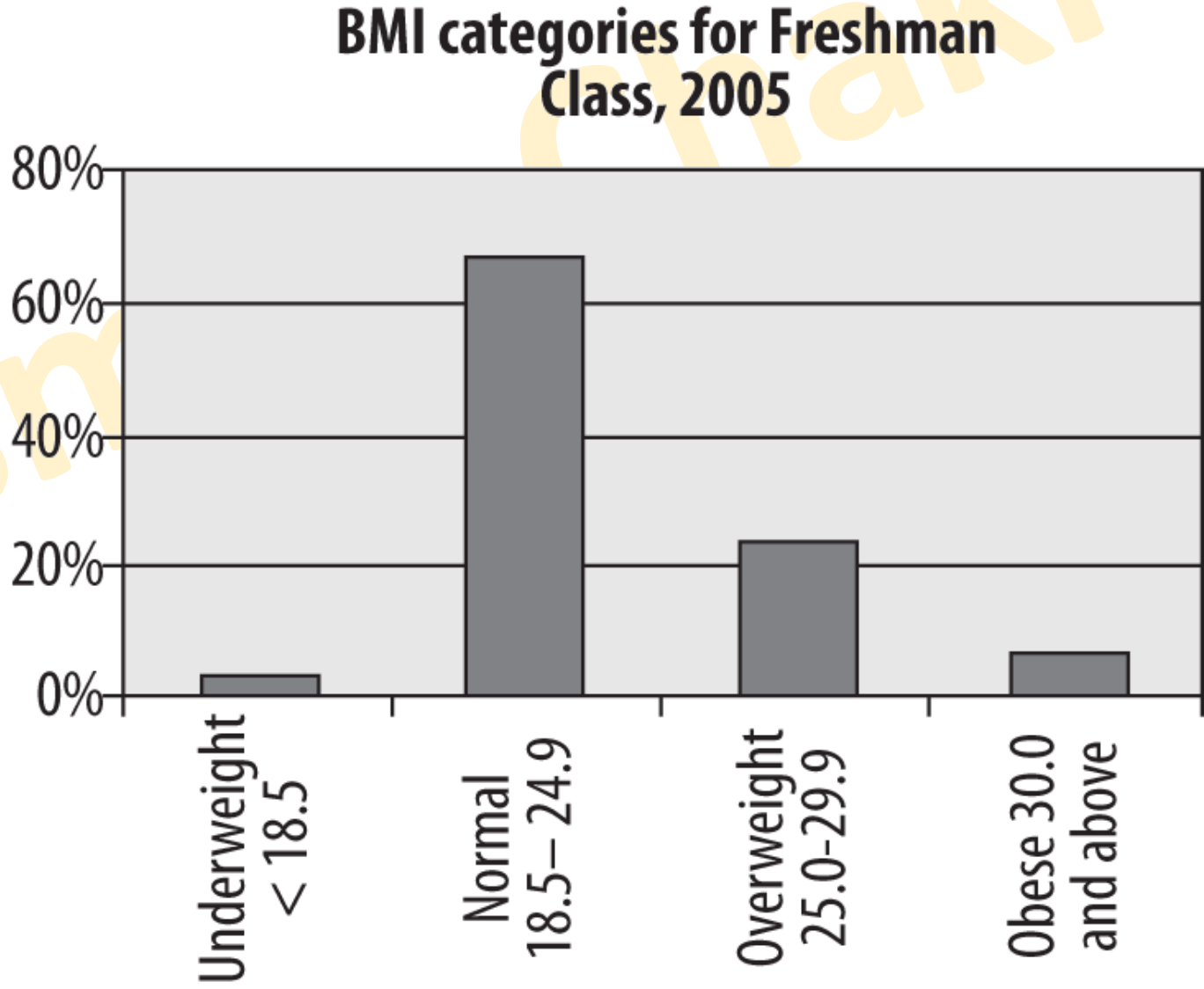
- The relative frequency is calculated by dividing the number of cases in each category by the total number of cases (750) and multiplying by 100.
- Absolute and relative frequency of BMI categories for the freshmen class of 2005

BMI range	Number	Relative frequency
< 18.5	25	3.3%
18.5–24.9	500	66.7%
25.0–29.9	175	23.3%
30.0 and above	50	6.7%

Relative frequencies should add up to approximately 100%, although the total might be slightly higher or lower due to rounding error

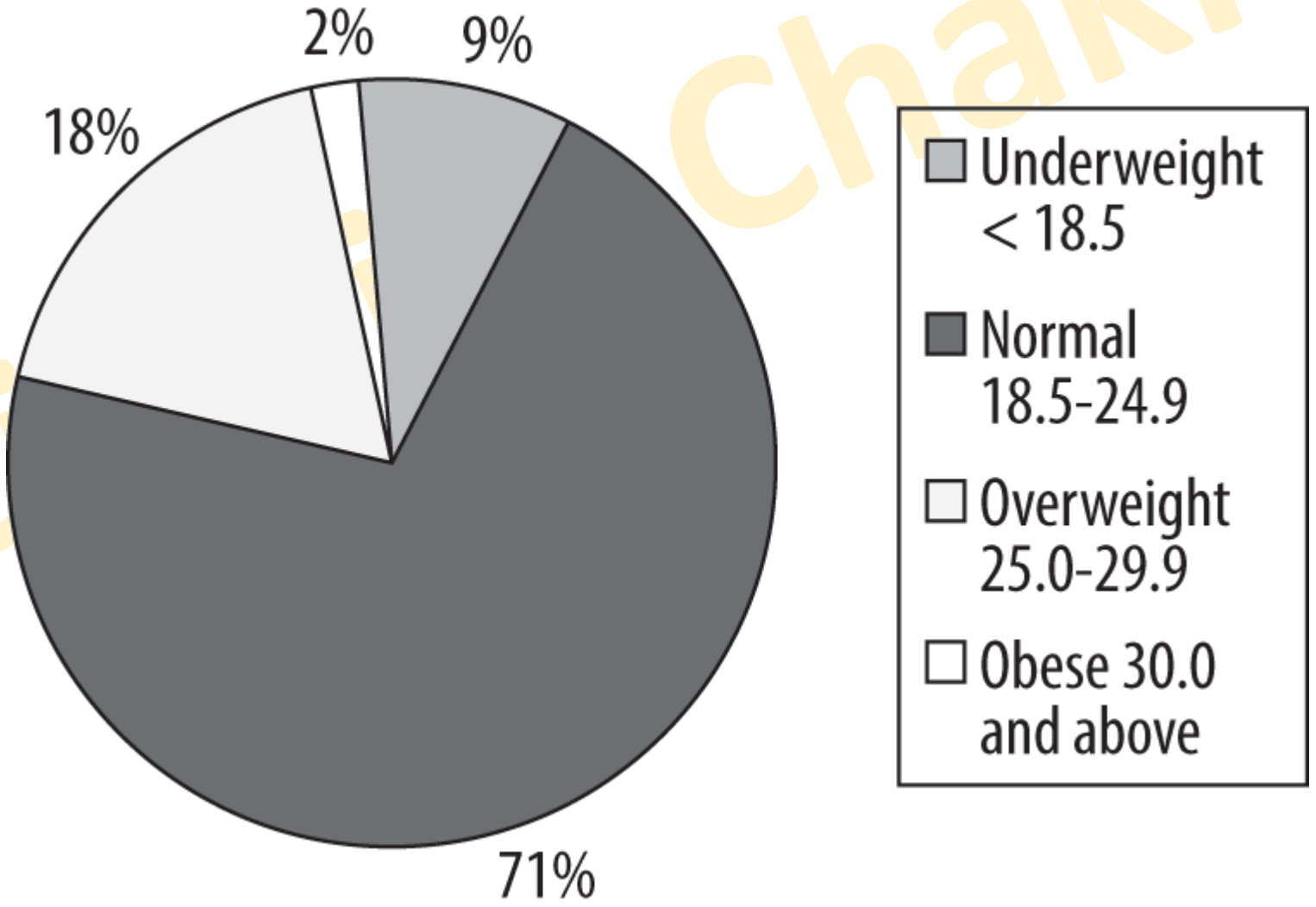
Plotting Scalar Data: Barplot

- Relative frequency of BMI categories in freshman class.
- Note that the two charts are identical except for the y-axis (vertical axis) labels, which are frequencies and percentages



Plotting Scalar Data: Pie chart

- Pie charts, are most useful when there are only a few categories of information and the differences among those categories are fairly large.
- Pie chart showing BMI distribution for freshmen entering in 2005



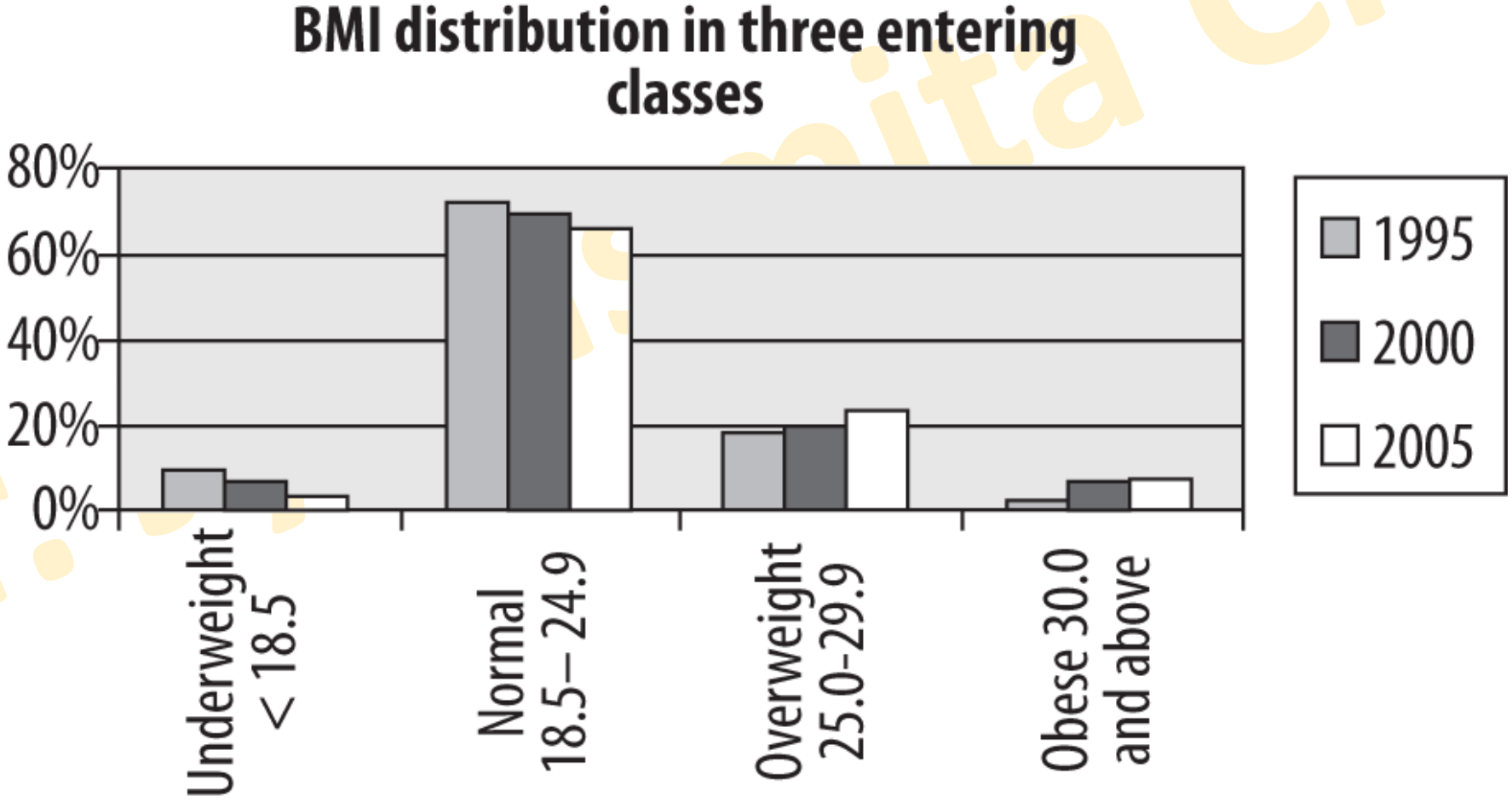
Plotting Scalar Data

- Absolute and relative frequencies of BMI for three entering classes

BMI range	1995		2000		2005	
Underweight < 18.5	50	8.9%	45	6.8%	25	3.3%
Normal 18.5–24.9	400	71.4%	450	67.7%	500	66.7%
Overweight 25.0–29.9	100	17.9%	130	19.5%	175	23.3%
Obese 30.0 and above	10	1.8%	40	6.0%	50	6.7%
Total	560	100.0%	665	100.0%	750	100.0%

Plotting Scalar Data: Grouped Bar chart

- Grouped Bar chart of BMI distribution in three entering classes



Plotting Scalar Data: Boxplot

- Let **n** be the number of data values in the data set.
- The **Median (Q2**) is the middle value of the data set.

$$\textbf{Median (Q2)} = \frac{1}{2}\ (n+1)\ th\ term$$

- The **Lower quartile [25th quartile] (Q1)** is the median of the lower half of the data set

$$\textbf{Lower Quartile (Q1)} = \frac{1}{4}\ (n+1)\ th\ term$$

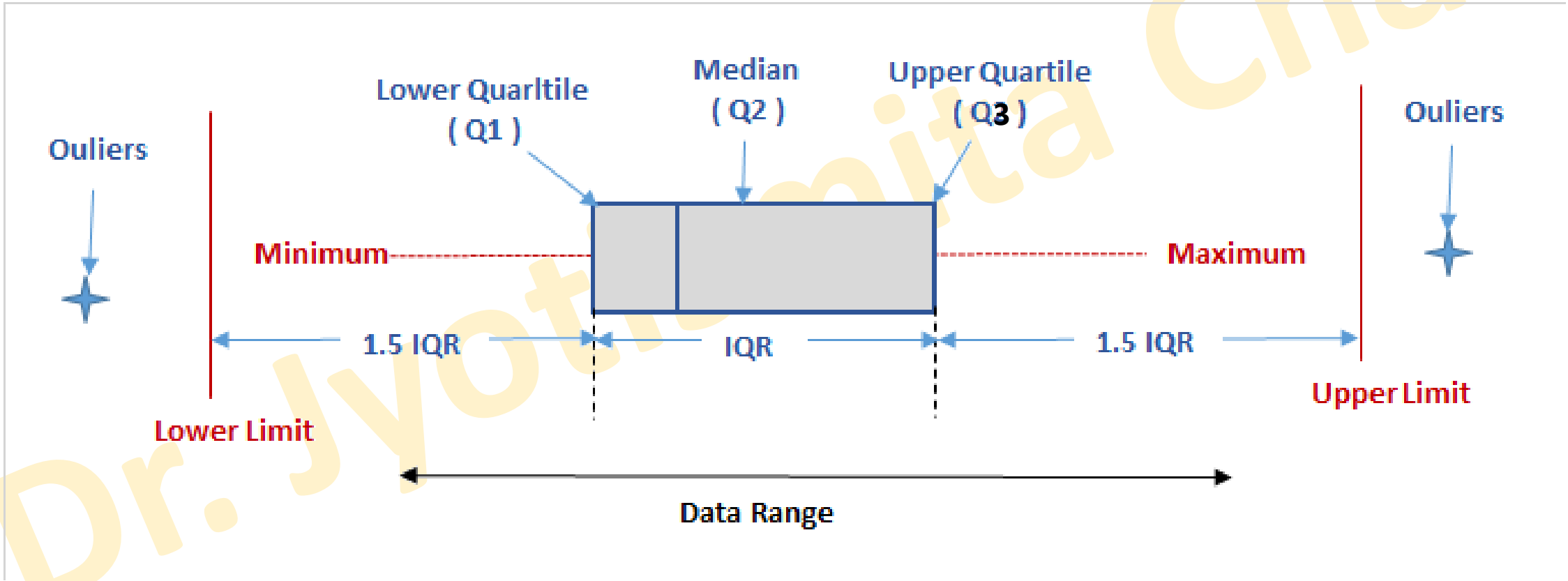
- The **Upper quartile [75th quartile] (Q3)** is the median of the upper half of the data set.

$$\textbf{Upper Quartile (Q3)} = \frac{3}{4}\ (n+1)\ th\ term$$

Plotting Scalar Data: Boxplot

- The **Interquartile range (IQR)** is the spread of the middle 50% of the data values.
- Interquartile Range (IQR) = Upper Quartile (Q3) – Lower Quartile (Q1)
- **IQR = Q3 – Q1**
- **Lower Limit = Q1 – 1.5 IQR.**
- **Upper Limit = Q3 + 1.5 IQR**
- So any value that will be more than the upper limit or lesser than the lower limit will be the outliers. Only the data that lies within Lower and upper limit are statistically considered normal and thus can be used for further observation or study.

Plotting Scalar Data: Boxplot



Plotting Scalar Data: Boxplot

- Let the data range be 100, 151, 236, 269,271,278,283,291, 301, 303, and 400
- Therefore **n = 11**

$$\text{Median (Q2)} = \frac{1}{2}\,(11+1)\,th\ term = 6th\ Term$$

$$\mathbf{Q2 = 278}$$

Plotting Scalar Data: Boxplot

$$\text{Lower Quartile (Q1)} = \frac{1}{4}\ (11+1)\ th\ term = 3rd\ Term$$

$$\mathbf{Q1\ = 236}$$

$$\text{Upper Quartile (Q3)} = \frac{3}{4}\ (11+1)\ th\ term = 9th\ Term$$

$$\mathbf{Q3\ = 301}$$

Inter Quartile Range (IQR) = Q3 – Q1 = 301 - 236 = 65

IQR = 65

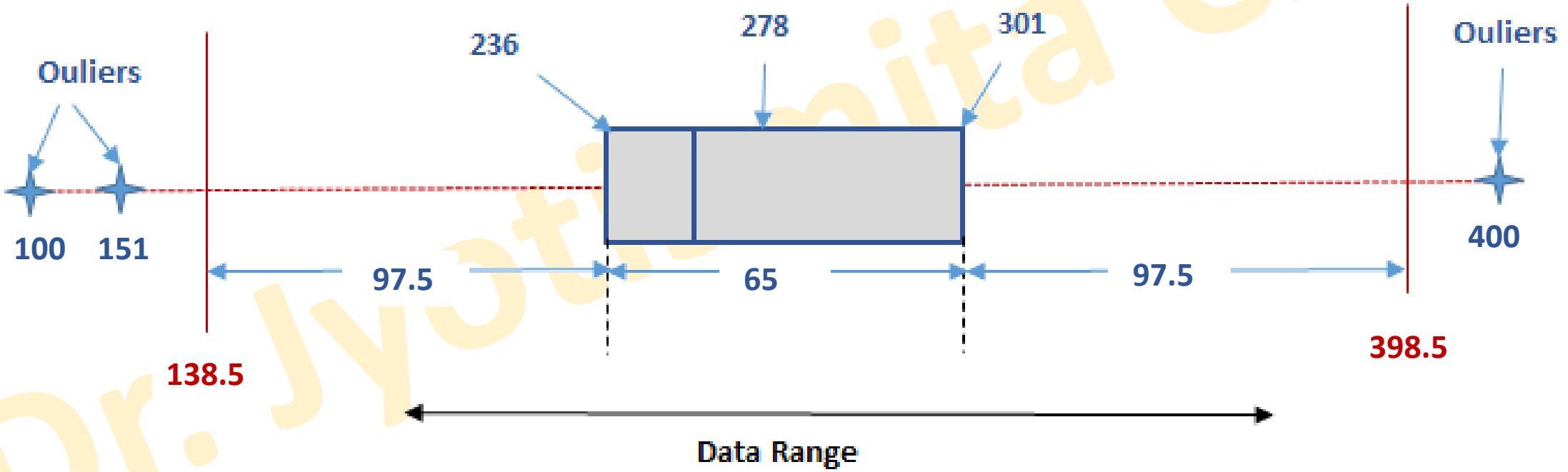
Lower Limit = Q1 – 1.5 IQR = 236 – 1.5 (65)

Lower Limit = 138.5

Upper Limit = Q3 + 1.5 IQR = 301 + 1.5 (65)

Upper Limit = 398.5

Plotting Scalar Data: Boxplot



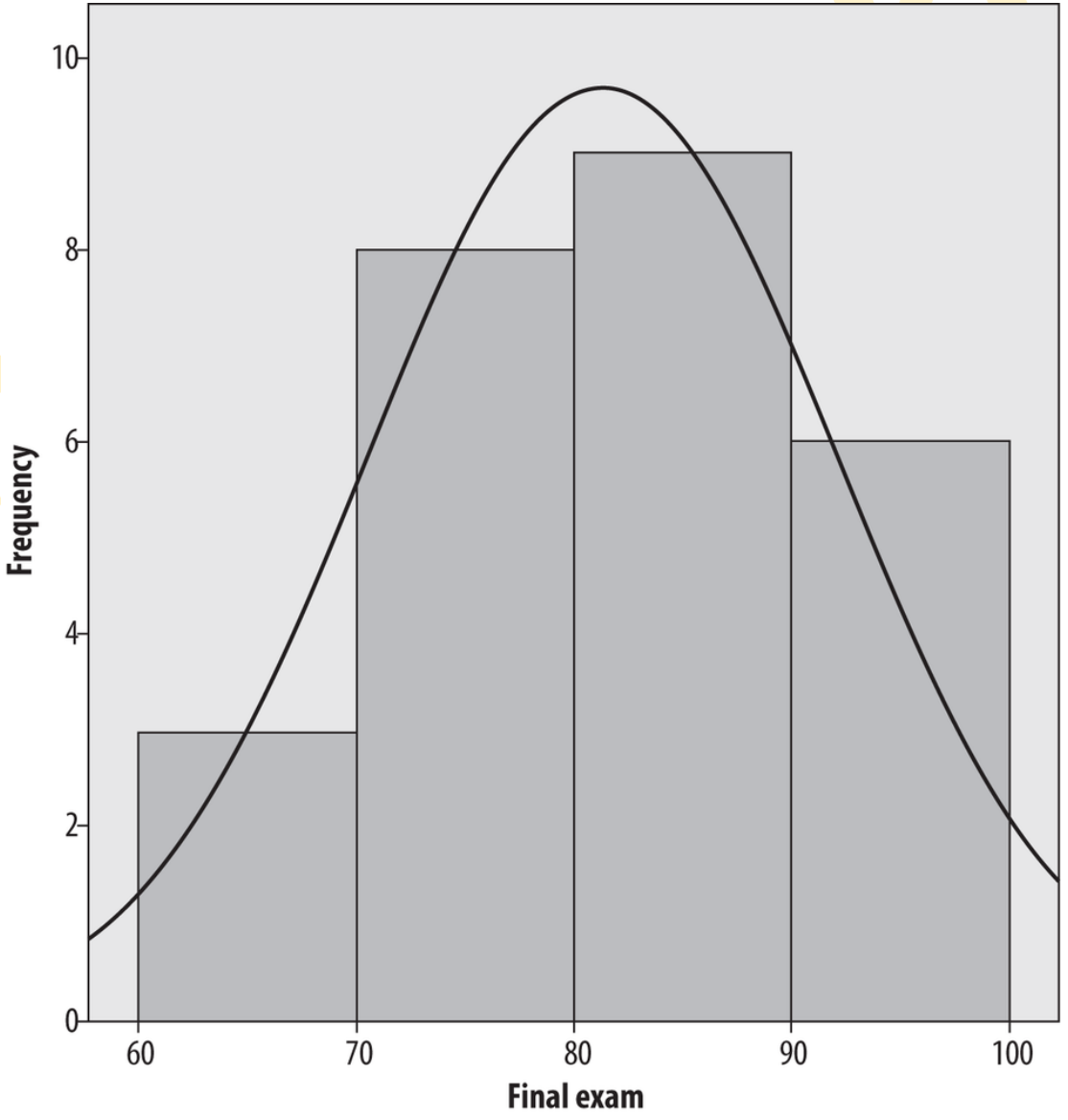
Plotting Scalar Data: Boxplot

- Hence it is clear that any range above 333.5 or below 201.5 are outliers. Hence in the data series 199, 201, 236, 269,271,278,283,291, 301, 303, 341, **outliers are 199, 201 and 341.** These 3 values which lies on either of the **extremes** can be considered **abnormal** and should be discarded from the entire series so that any analysis made on this series is not influenced by these extreme values. So the data series that should be considered for further observation or study after discarding the outliers are as below.

  236, 269,271,278,283,291, 301, 303

Plotting Scalar Data: Histogram

- Suppose we have the final exam grades for 26 students and want to present them graphically. These are the grades:
- 61, 64, 68, 70, 70, 71, 73, 74, 74, 76, 79, 80, 80, 83, 84, 84, 87, 89, 89, 89, 90 92, 95, 95, 98, 100
- *Histogram with a bin width of 10*



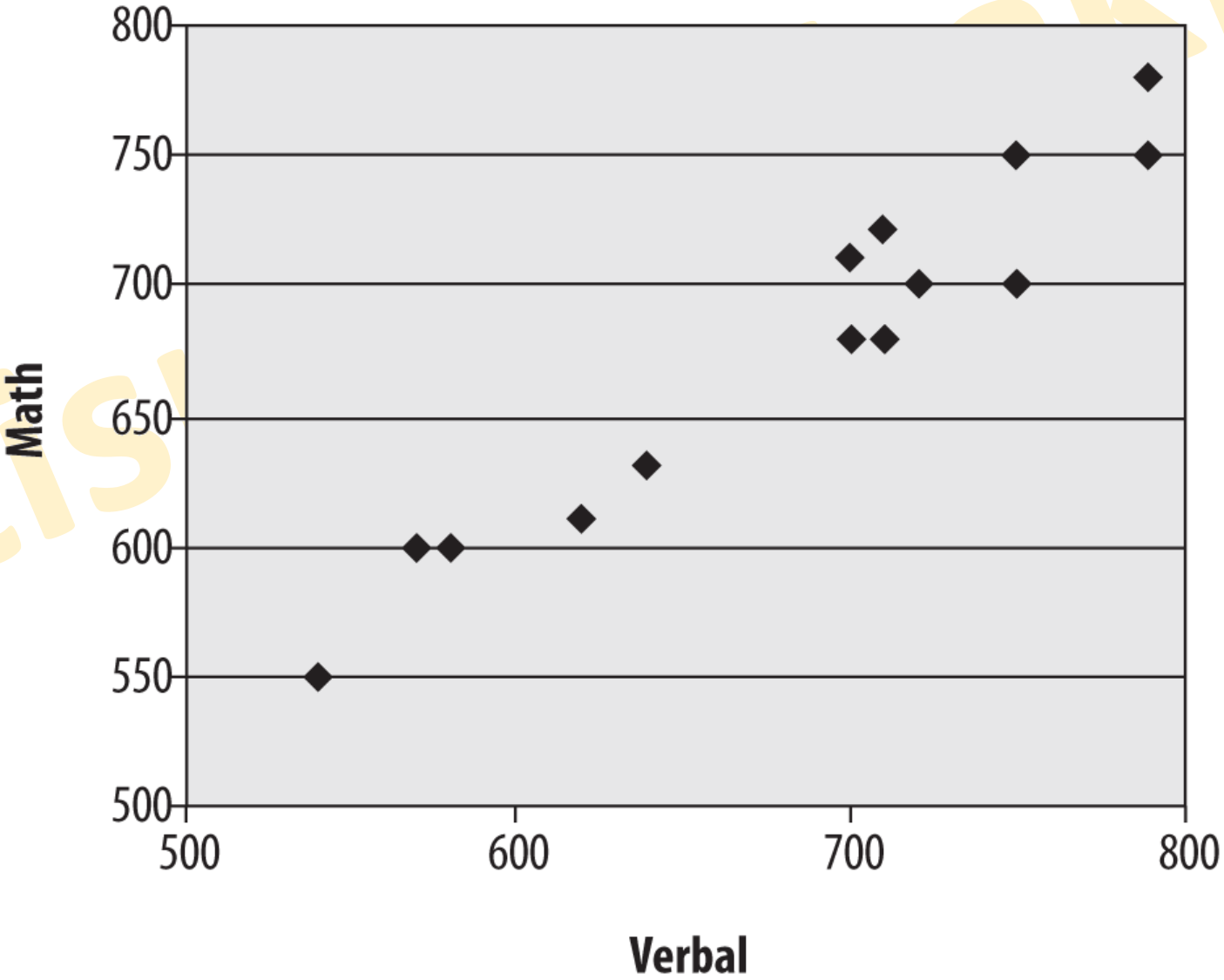
Plotting Scalar Data: Scatter plot

- SAT scores for 15 students

Math	Verbal
750	750
700	710
720	700
790	780
700	680
750	700
620	610
640	630
700	710
710	680
540	550
570	600
580	600
790	750
710	720

Plotting Scalar Data: Scatter plot

Math SAT score on the *y*-axis (vertical axis) and verbal SAT score on the *x*-axis (horizontal axis), makes the relationship between scores much clearer.

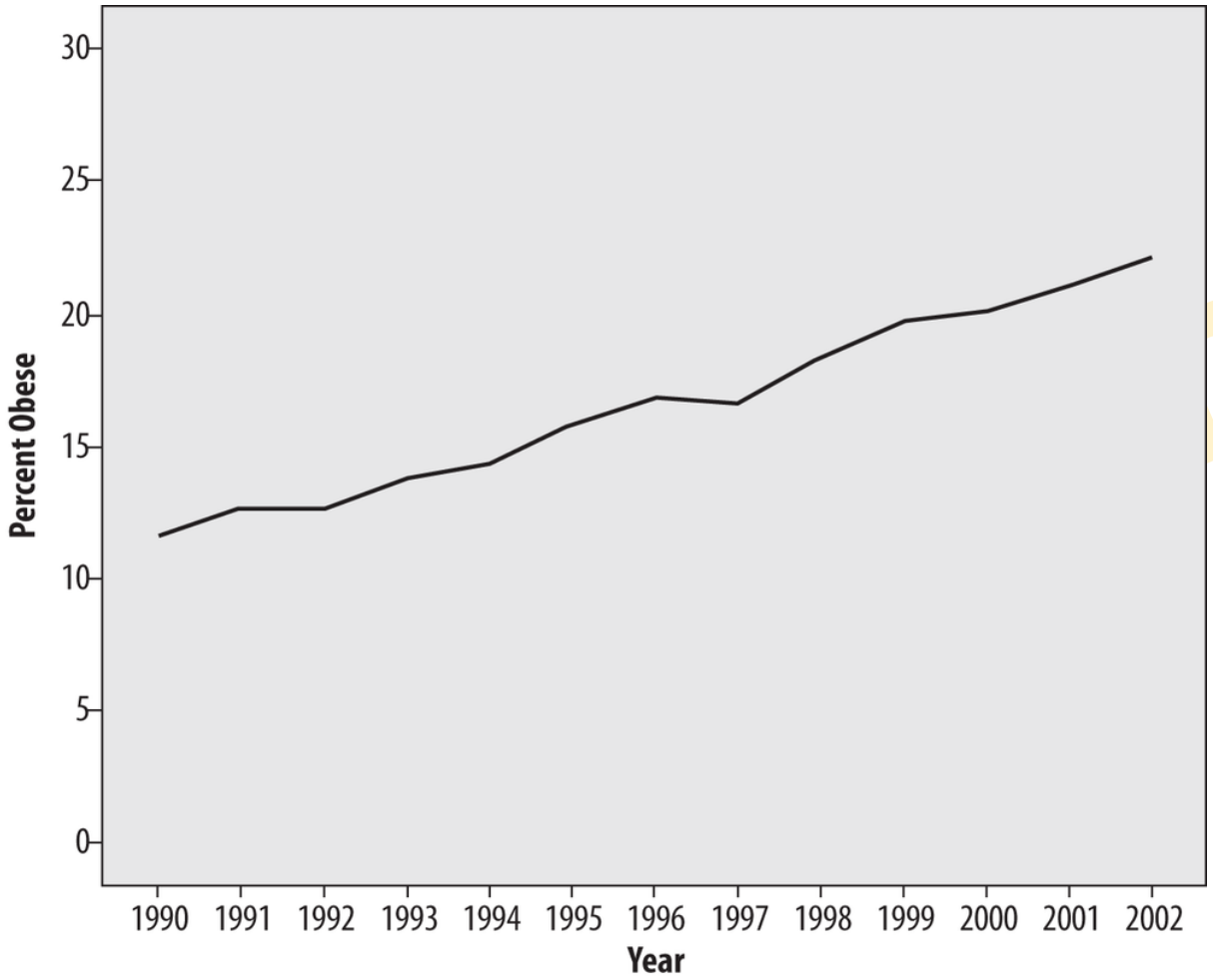


Plotting Scalar Data: Line Graphs

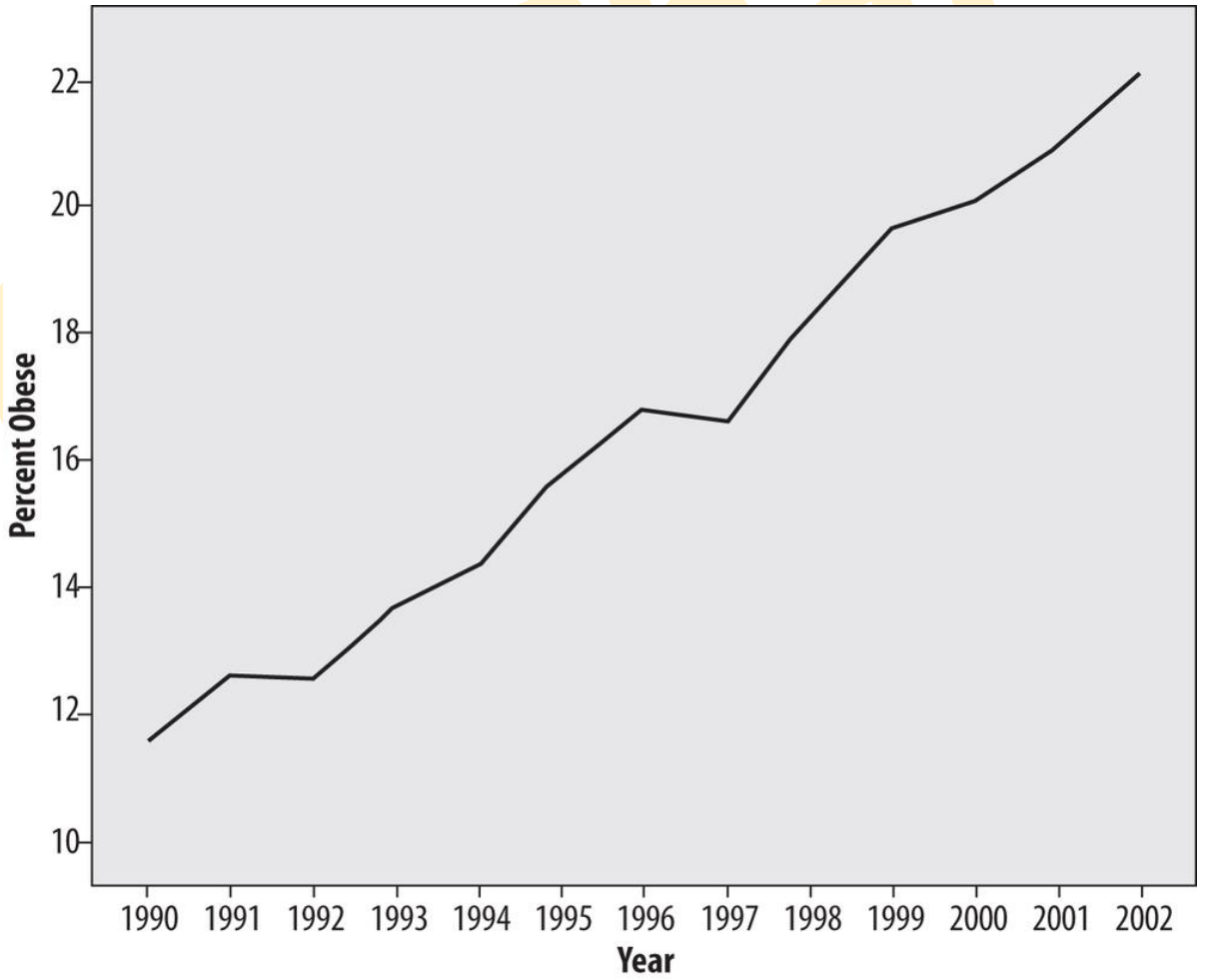
- Percentage of obesity among U.S. adults, 1990–2002 (CDC)

Year	Percent obese
1990	11.6%
1991	12.6%
1992	12.6%
1993	13.7%
1994	14.4%
1995	15.8%
1996	16.8%
1997	16.6%
1998	18.3%
1999	19.7%
2000	20.1%
2001	21.0%
2002	22.1%

Plotting Scalar Data: Line Graphs



Sensible representation of the data



A larger scale and smaller range for the *y*-axis

Example

- Visualize the salary data distribution after transforming the data using the following function:

$$z = \frac{x - \mu}{\sigma}$$

where x = value in feature, μ = feature mean, and σ = feature standard deviation.

EID	Salary
604399	445000
988334	110000
301647	255000
582313	420000
339001	200000
609356	440000
1081649	150000
610842	105000
1183070	195000
794062	200000

Example

- Salary Mean: 252000
- Salary STD: 134023.2

After Transformation Plot EID and Z Value.

EID	Salary	Z Value
604399	445000	1.440049
988334	110000	-1.05952
301647	255000	0.022384
582313	420000	1.253514
339001	200000	-0.38799
609356	440000	1.402742
1081649	150000	-0.76106
610842	105000	-1.09683
1183070	195000	-0.4253
794062	200000	-0.38799

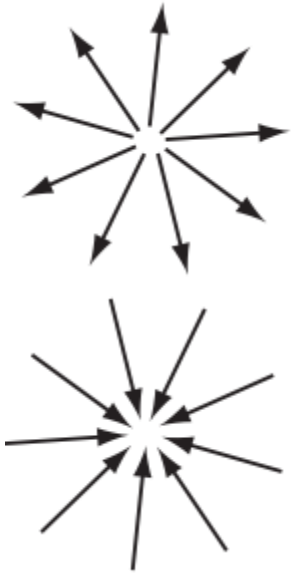
Vector Visualization Techniques

- The majority of visualization applications deal with data that describes physical phenomena in two- or three-dimensional space.
- As a consequence, most visualization software defines all vectors to have three components.
- 2D vectors are modeled as 3D vectors with the third (z) component equal to null.
- A very important application domain for vector visualization is computational fluid dynamics (CFD).
- CFD simulations are able to predict the time-dependent behavior of compressible 3D fluid flows consisting of several potentially interacting substances, or species, having different densities and pressures, over complex spatial geometries.
- The solution of a CFD simulation consists of several datasets, each for a different time step.
- For each time step, several attributes are computed and stored into the solution dataset, such as velocity, pressure, density, flow divergence, and vorticity.

Vector Properties: Divergence

- Given a vector field v : $R^3 \rightarrow R^3$, the divergence of v = (v_x, v_y, v_z) is the scalar quantity:

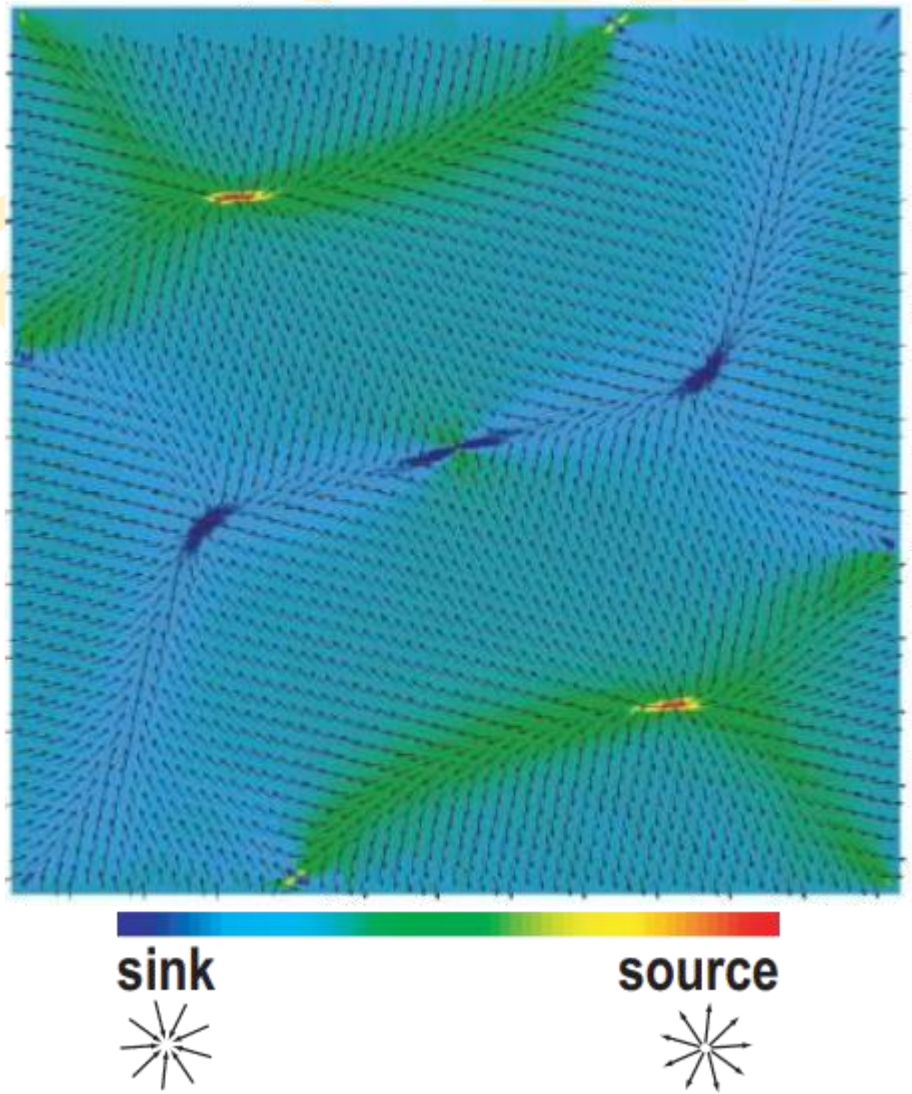
$$\operatorname{div}\ \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}.$$



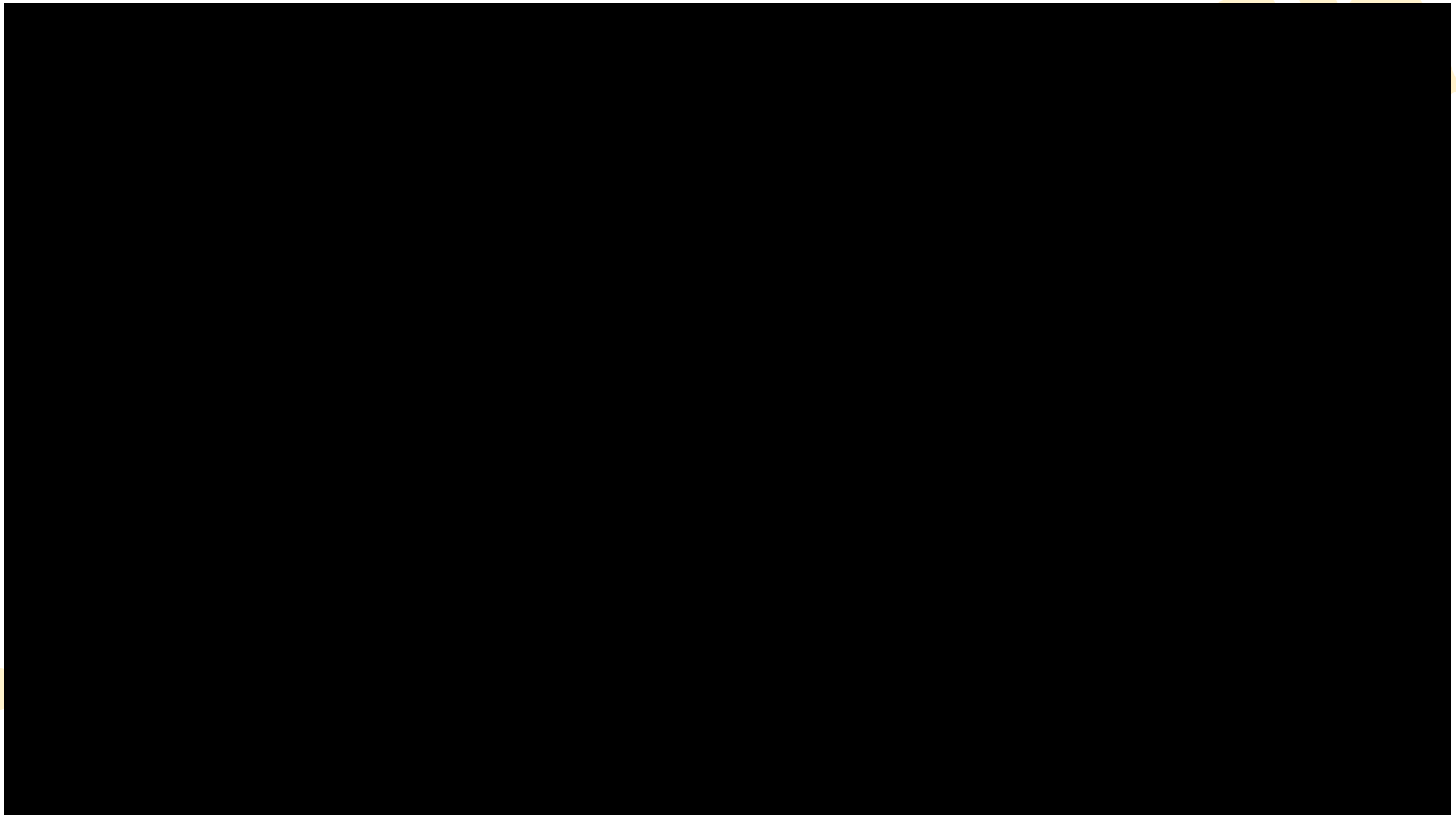
- A positive divergence at p denotes that mass would spread from p outward. Positive divergence points are called **sources**.
- A negative divergence at p denotes that mass gets sucked into p. Negative divergence points are called **sinks**.
- A zero divergence at p denotes that mass is transported without getting spread or sucked, i.e., without compression or expansion

Vector Properties: Divergence

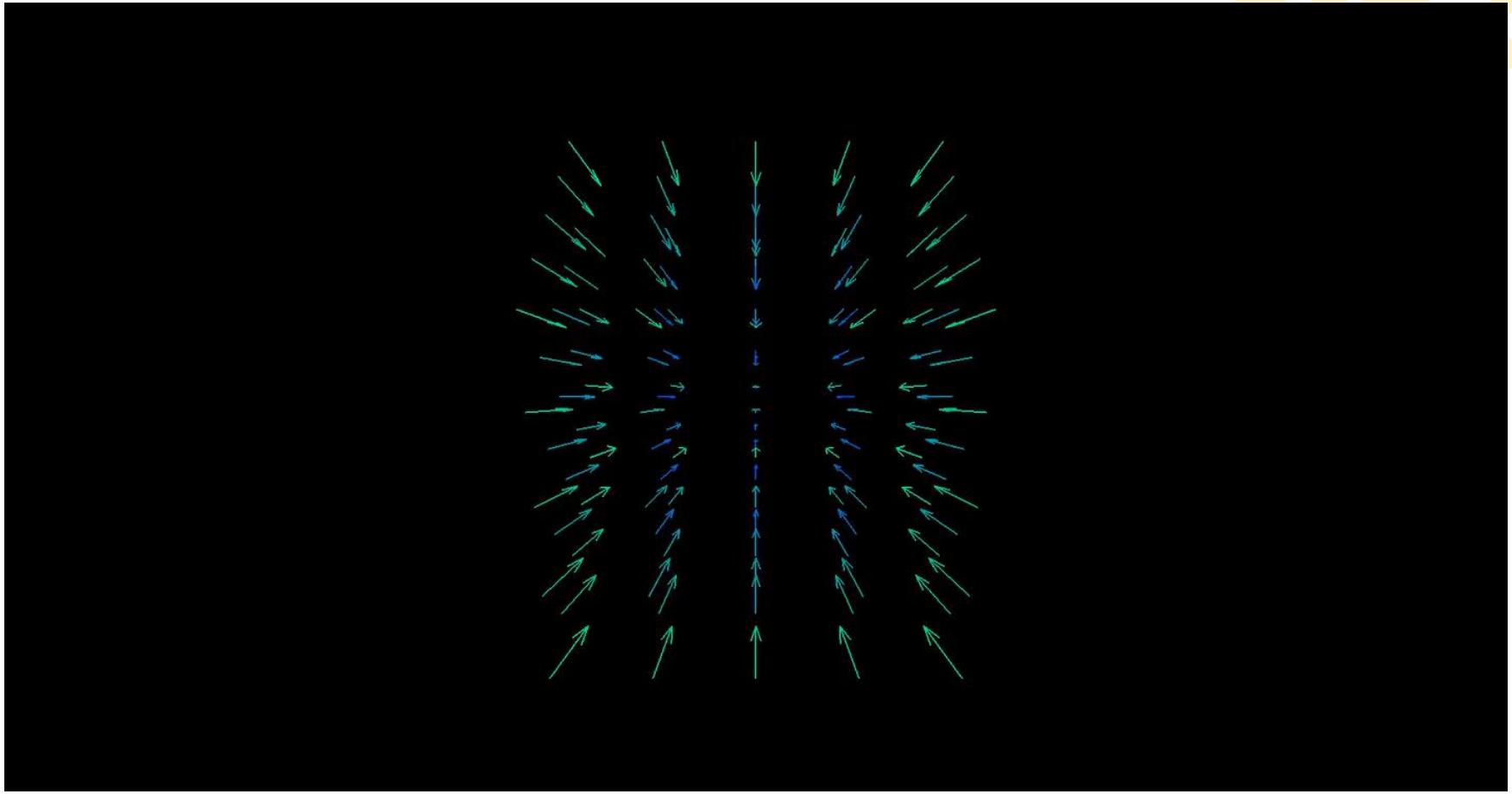
- Figure shows the divergence of a 2D flow field using a blue-to-red colormap.
- The vector field is visualized with arrow glyphs for illustration purposes.
- Red areas indicate high positive divergence, or sources.
- Two such sources are clearly visible. Blue areas indicate high negative divergence, or sinks.
- Within the dark blue area, two pronounced sinks are visible.



Vector Properties: Divergence



Vector Properties: Divergence: Negative divergence in 3D



Vector Properties: Vorticity or Curl

- The vorticity 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 the curl.
- Hence, the vorticity vector characterizes the speed and direction of rotation of a given vector field at every point.
- The curl of a vector field captures the idea of how a fluid may rotate.
- In two dimensions, if a vector field is given by a function:

$$\vec{\mathbf{v}}(x,y) = v_1(x,y)\hat{\mathbf{i}} + v_2(x,y)\hat{\mathbf{j}},$$

- $\hat{\mathbf{i}}$ represents the unit vector in the x-direction.
- $\hat{\mathbf{j}}$ represents the unit vector in the y-direction.

- This rotation is given by the formula:

$$\text{2d-curl } \vec{\mathbf{v}} = \frac{\partial v_2}{\partial x} - \frac{\partial v_1}{\partial y}$$

Vector Properties: Vorticity or Curl

- A particular vector field is defined with the following function:

$$\vec{\mathbf{v}}(x,y) = \begin{bmatrix} y^3 - 9y \\ x^3 - 9x \end{bmatrix} = (y^3 - 9y)\hat{\mathbf{i}} + (x^3 - 9x)\hat{\mathbf{j}}$$

- When you evaluate curl, a **positive** number will indicate a general tendency to rotate **counterclockwise** around a point, a negative quantity indicates the opposite, **clockwise rotation**. If it equals 0, there is **no rotation** in the fluid around a point

Vector Properties: Vorticity or Curl

- Consider the vector field defined by the function

$$\vec{\mathbf{v}}(x,y) = \begin{bmatrix} \cos(x+y) \\ \sin(xy) \end{bmatrix}$$

- Determine whether a fluid flowing according to this vector field has clockwise or counterclockwise rotation at the point

$$p = \left(0, \frac{\pi}{2}\right)$$

Vector Properties: Vorticity or Curl: Step 1

- We apply the **2d-curl** formula we just found to this function.
- This will give us a new, scalar-valued function that indicates the rotation near each point.
- Then, we'll plug in the point $(0, \pi/2)$ to see whether the rotation there is positive (counterclockwise), zero, or negative (clockwise).

$$\begin{aligned}
\text{2d-curl } \vec{\mathbf{v}} &= \frac{\partial v_2}{\partial x} - \frac{\partial v_1}{\partial y} \\
&= \frac{\partial}{\partial x}(\sin(xy)) - \frac{\partial}{\partial y}(\cos(x+y)) \\
&= \cos(xy)y - (-\sin(x+y)) \\
&= \boxed{\cos(xy)y + \sin(x+y)}
\end{aligned}$$

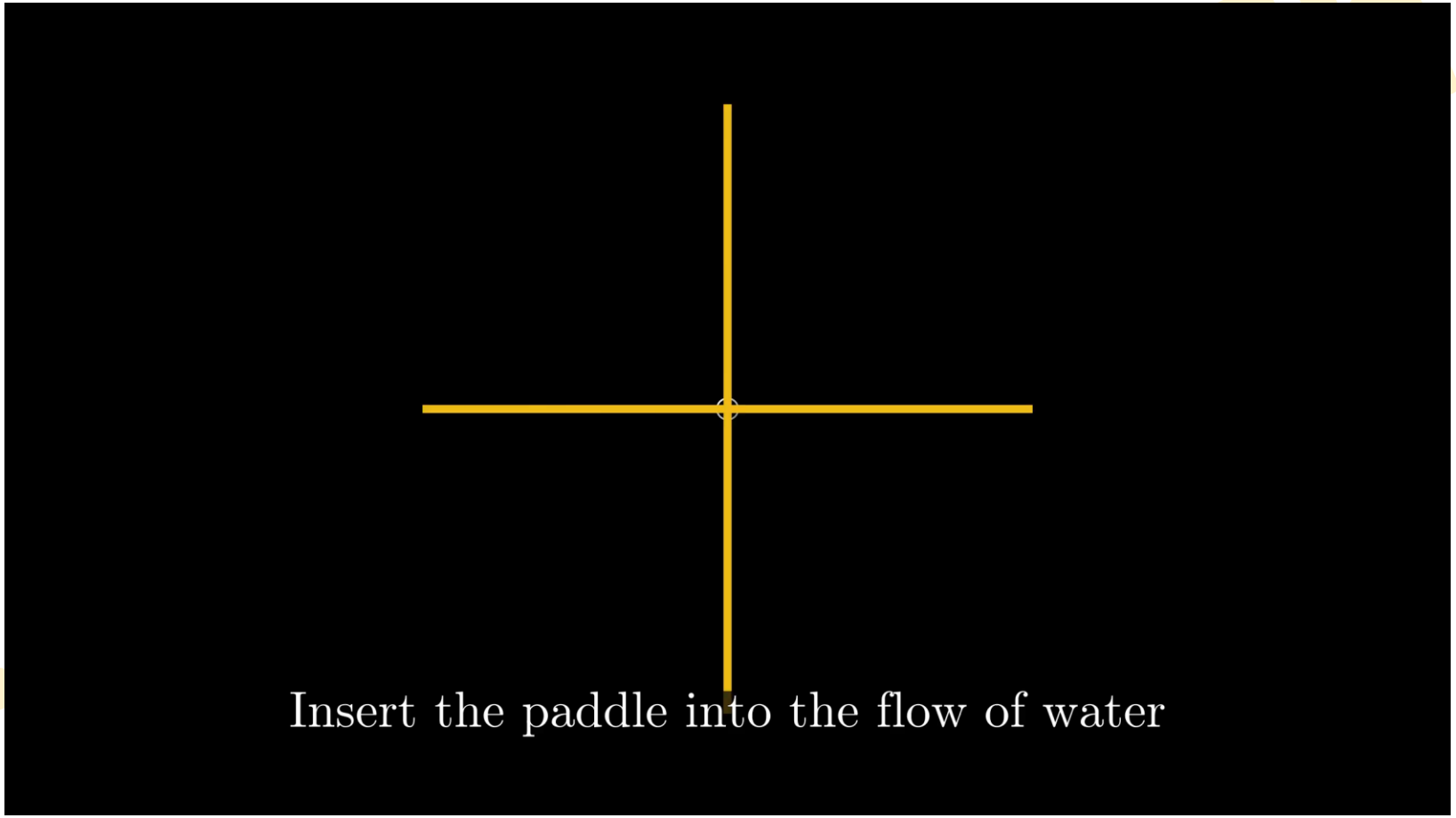
Vector Properties: Vorticity or Curl: Step 2

- Plug in the point $(0, \pi/2)$:

$$p = \left(0, \frac{\pi}{2}\right) \rightarrow \cos\left(0 \cdot \frac{\pi}{2}\right) \frac{\pi}{2} + \sin\left(0 + \frac{\pi}{2}\right)$$
$$= 1 \cdot \frac{\pi}{2} + 1$$
$$= \frac{\pi + 2}{2}$$

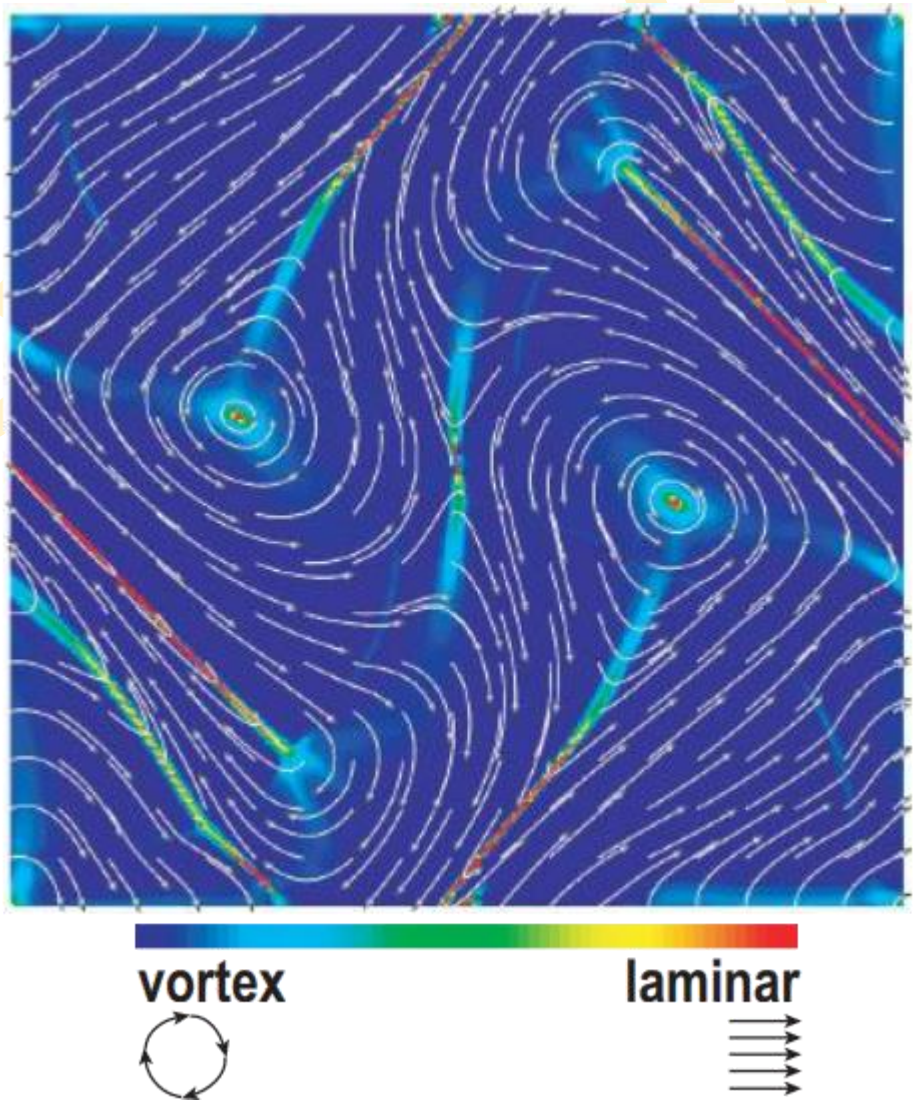
- Interpretation: Because this is a positive value, **the fluid will tend to flow counterclockwise at the point *p***.

Vector Properties: Vorticity or Curl

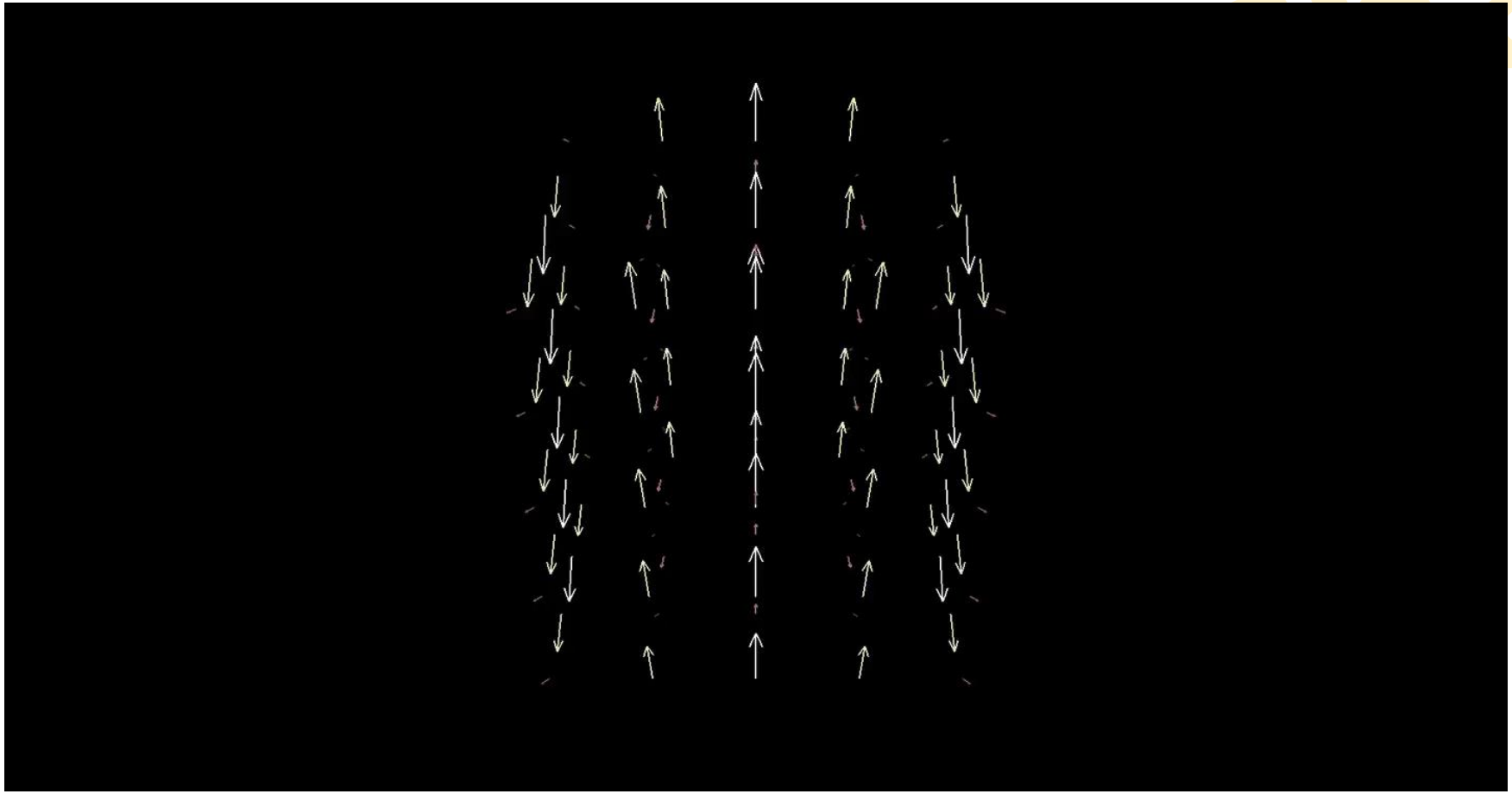


Vector Properties: Vorticity or Curl

- Figure shows the absolute value of the vorticity of a velocity field from a simulation, using a blue-to-red colormap.
- Blue areas indicate low-vorticity, **laminar** regions.
- Red areas indicate high-vorticity regions.
- We can see two types of such regions.
- Two small circular red spots indicate localized vortices, which are also outlined by the circling streamlines.
- Several elongated thin red strips indicate areas where the vector field quickly changes direction.
- Given the shape, these are not vortices, but separation lines that divide regions where the flow has opposite directions.
- If one is interested in locating such high-vorticity areas, a simple method of reasonable accuracy is to contour the vorticity field for high isovalues and select the data points inside such a contour.



Vector Properties: Vorticity or Curl: 3D



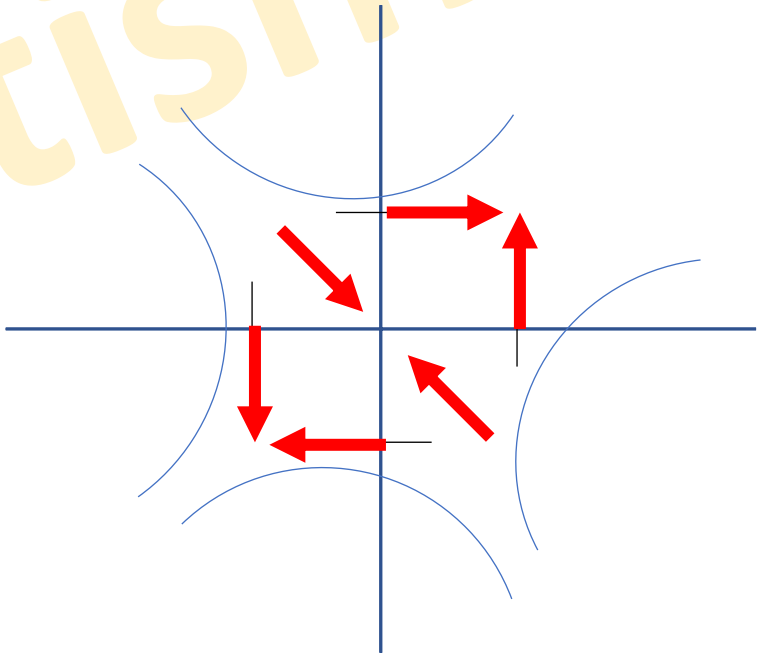
Vector Properties: Diversity and Curl

Classification of divergence and curl

Example

- Consider the vector field <y, x>. Visualize the vector by considering some vector data points. Calculate the divergence and curl at an arbitrary point.
- ANS: Represent the vector using the function $\vec{F} = y\hat{i} + x\hat{j}$

x	y	F
1	0	(0, 1)
0	1	(1, 0)
-1	0	(0, -1)
0	-1	(-1, 0)
1	1	(1, 1)
-1	1	(1, -1)
-1	-1	(-1, -1)
1	-1	(-1, 1)



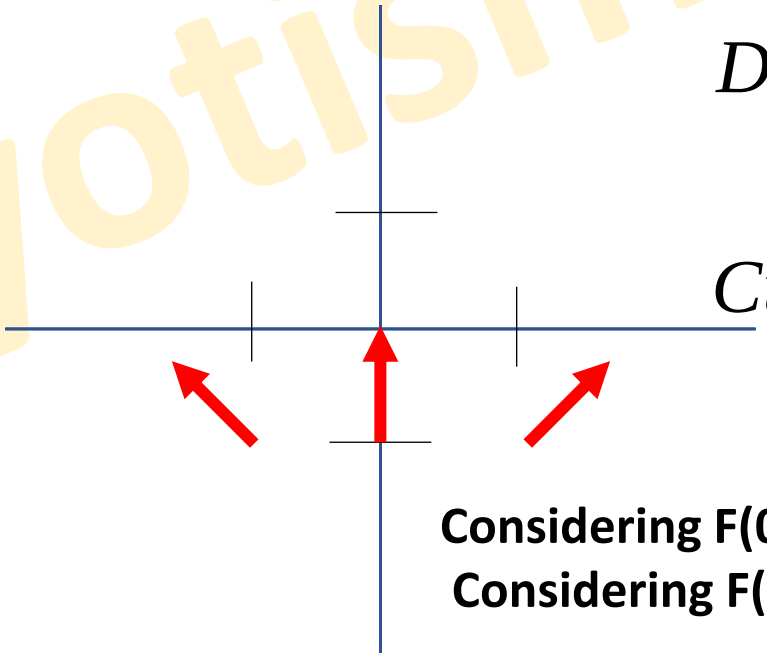
$$Div\vec{F} = \frac{\partial}{\partial x} y + \frac{\partial}{\partial y} x = 0$$

$$Curl\vec{F} = \frac{\partial}{\partial x} x - \frac{\partial}{\partial y} y = 0$$

Example

- Consider the vector field <x, y^2>. Visualize the vector by considering some vector data points. Calculate the divergence and curl at an arbitrary point.
- ANS: Represent the vector using the function $\vec{F} = x\hat{i} + y^2\,\hat{j}$

x	y	F
1	0	(1, 0)
0	1	(0, 1)
-1	0	(-1, 0)
0	-1	(0, 1)
1	1	(1, 1)
-1	1	(-1, 1)
-1	-1	(-1, 1)
1	-1	(1, 1)



$$Div\vec{F} = \frac{\partial}{\partial x}x + \frac{\partial}{\partial y}y^2 = 1 + 2y$$

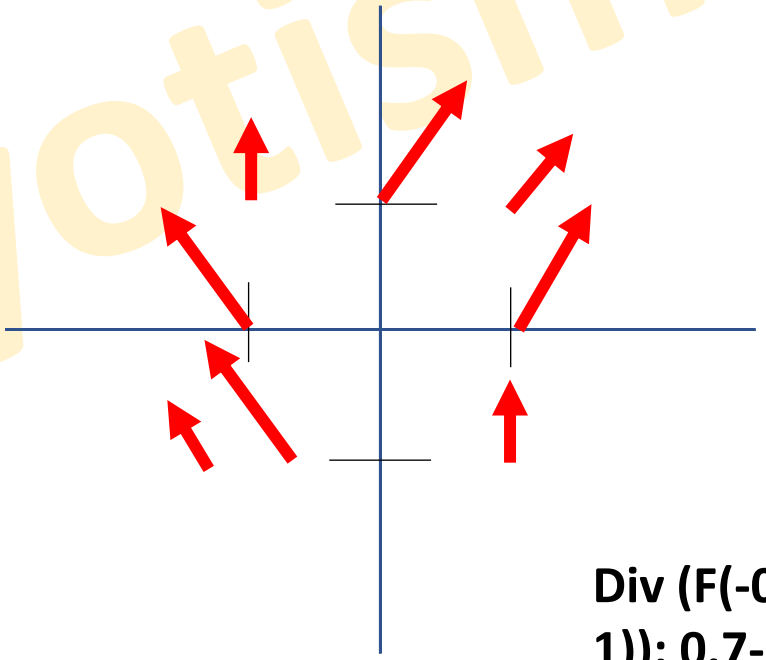
$$Curl\vec{F} = \frac{\partial}{\partial x}y^2 - \frac{\partial}{\partial y}x = 2y\frac{\partial y}{\partial x} - \frac{\partial x}{\partial y}$$

Considering F(0, 1) point: Div(F) will be 2. (Source)
Considering F(0, 1) point: Curl(F) will be 2. (CounterClockwise)

Example

- Consider the vector field <sin(x+y), cos(xy)>. Visualize the vector by considering some vector data points. Calculate the divergence and curl at an arbitrary point.
- ANS: Represent the vector using the function $\vec{F} = \sin(x+y)\hat{i} + \cos(xy)\hat{j}$

x	y	F
1	0	(0.8, 1)
0	1	(0.8, 1)
-1	0	(-0.8, 1)
0	-1	(-0.8, 1)
1	1	(0.9, 0.5)
-1	1	(0, 0.5)
-1	-1	(-0.9, 0.5)
1	-1	(0, 0.5)



$$Div\vec{F} = \frac{\partial}{\partial x}\sin(x+y) + \frac{\partial}{\partial y}\cos(xy)$$

$$= \cos(x+y) + (-\sin(xy)x)$$

$$Curl\vec{F} = \frac{\partial}{\partial x}\cos(xy) - \frac{\partial}{\partial y}\sin(x+y)$$

$$= (-\sin(xy)y) - \cos(x+y)$$

Div (F(-0.8, 1)): 0.98-0.56 = 0.42 (Source), Curl (F((-0.8, 1)): 0.7-0.9 = -0.2 (clockwise)

Vector Glyphs

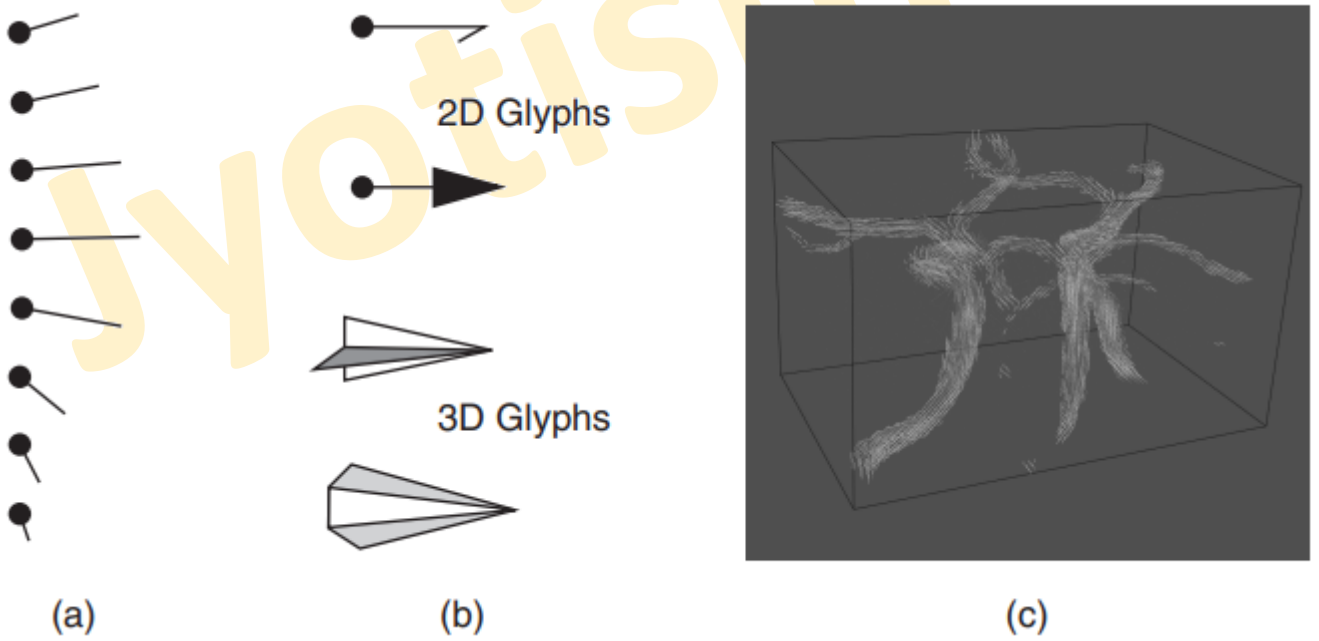
- Vector glyphs are probably the simplest, and fastest, and most popular technique for visualizing vector fields.
- The vector glyph mapping technique essentially associates a vector glyph, or vector icon, with every sample point of the vector dataset.
- Various properties of the icon, such as location, direction, orientation, size, and color, are adjusted to reflect the value of the vector attribute it represents.
- The name glyph, meaning “sign” in Greek, reflects 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.

Vector Glyphs: Line Glyphs

- Lines essentially show the position, direction, and magnitude of a set of vectors.
- Given a vector dataset defined on a sampled domain D, we associate a line $l = (x, x+kv(x))$ with every sample point $x \in D$ 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.

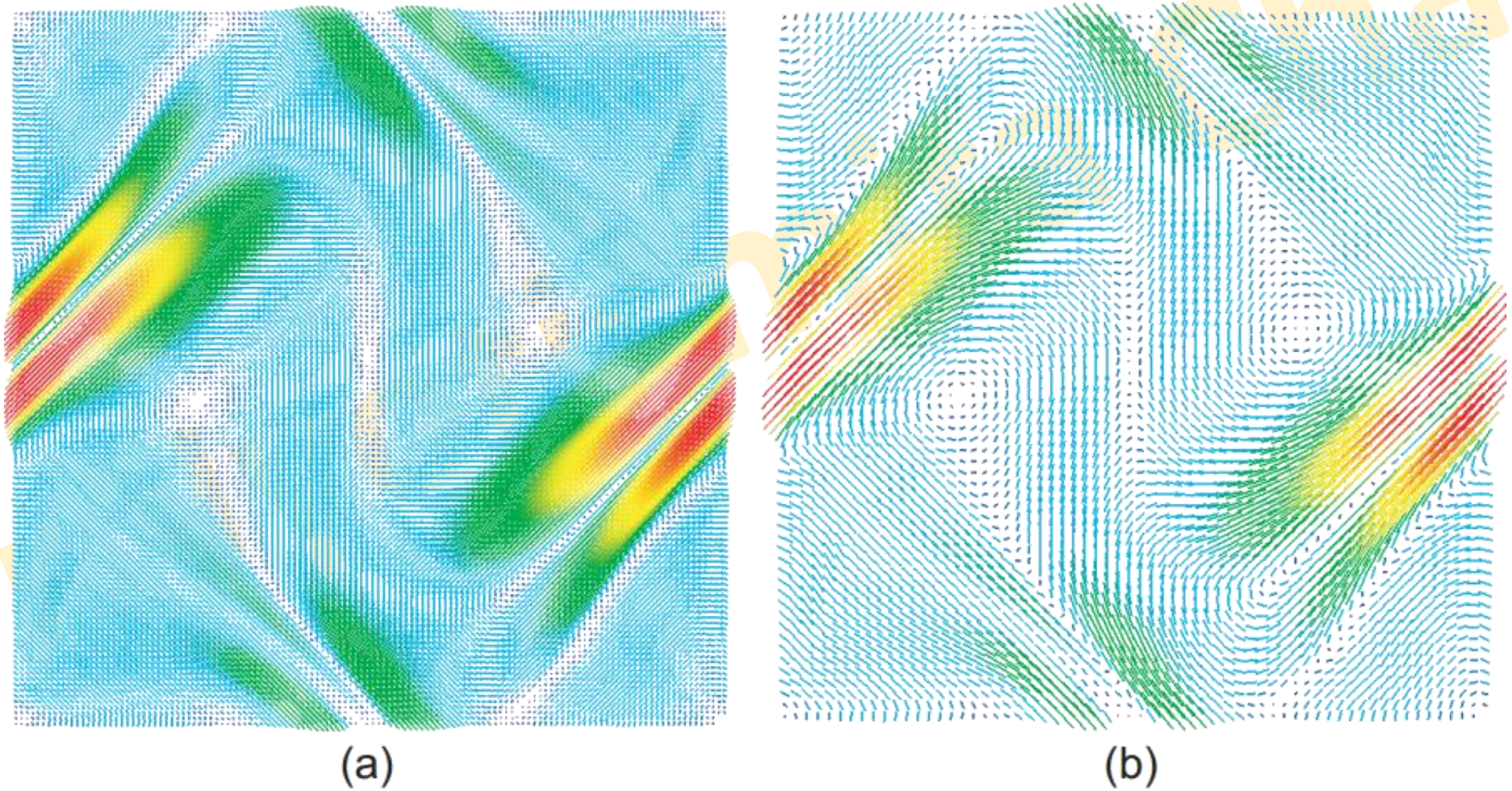
Vector Glyphs: Line Glyphs

- There are many variations of this technique. Arrows may be added to indicate the direction of the line. The lines may be colored according to vector magnitude or some other scalar quantity (e.g., pressure or temperature). Also, instead of using a line, oriented ‘‘glyphs’’ can be used.



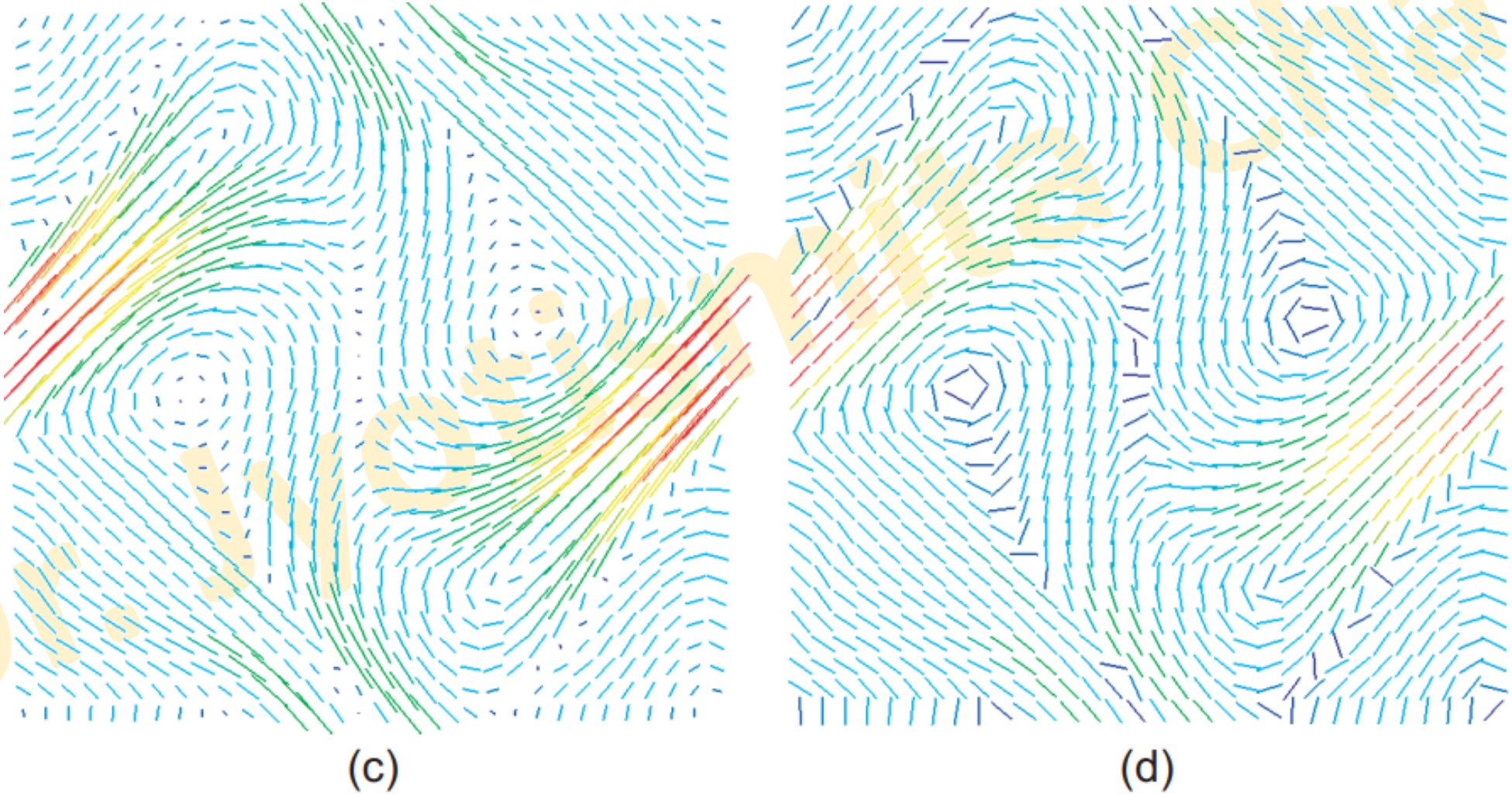
Vector visualization techniques. (a) Oriented lines; (b) oriented glyphs; (c) complex vector visualization.

Vector Glyphs: Line Glyphs



(a) (b)

Vector Glyphs: Line Glyphs



Vector Glyphs: Line Glyphs

- 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 (see Figure (a)), a rate of 4 (Figure (b)), and a rate of 8 (Figure (c)).
- 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 (d), 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.

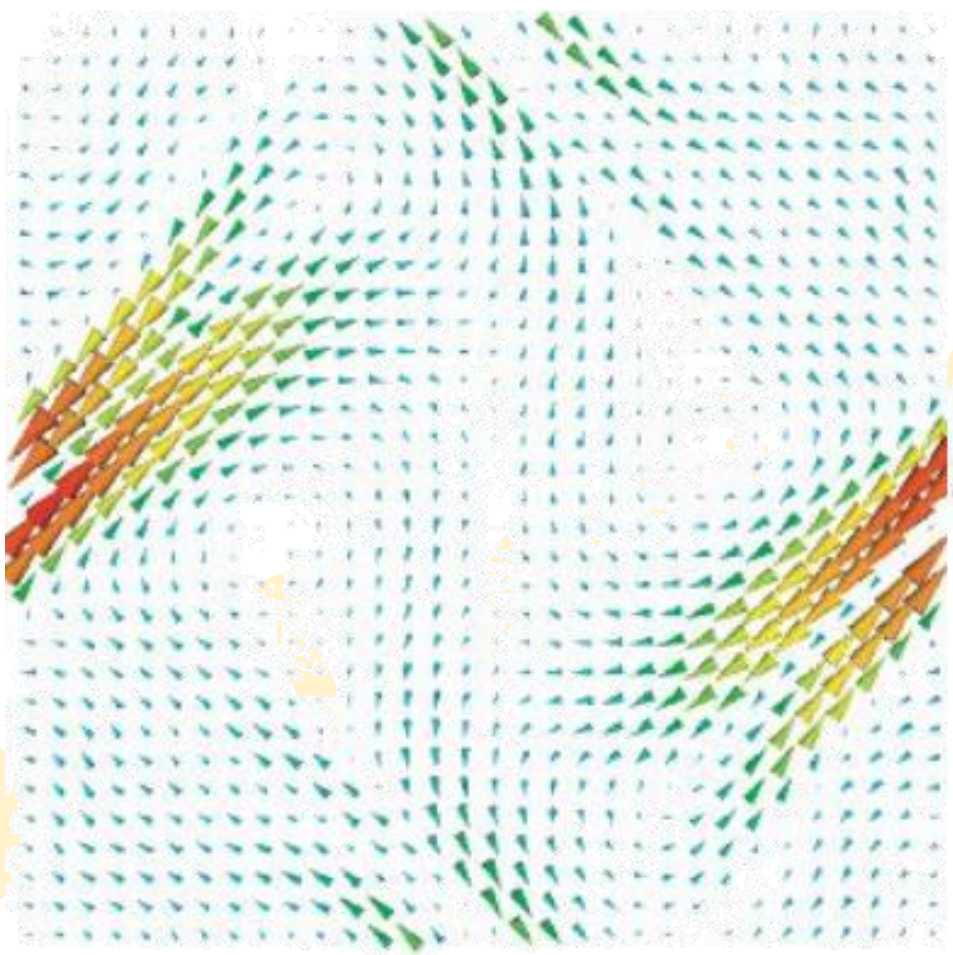
Vector Glyphs: Line Glyphs

- Looking at Figure (a) – (d), we can make a number of important observations.
  - It is clear that high-resolution vector datasets must be subsampled in order 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.
  - 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 not too large, so it would not intersect neighboring glyphs.

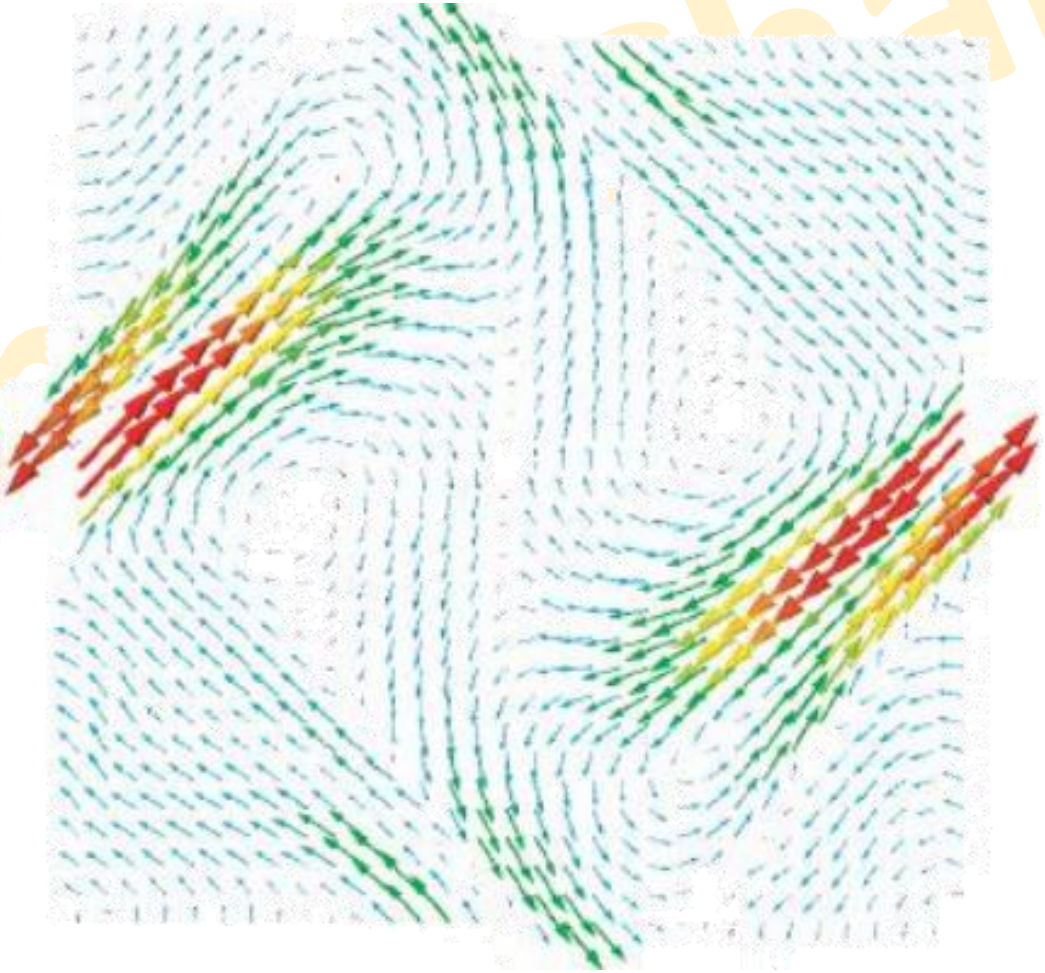
Vector Glyphs: Cone and Arrow glyphs

- More complex shapes can be used for glyphs besides lines.
- Figure (next slide) shows the same 2D vector field as in the previous Figure, this time visualized with 3D cone and arrow glyphs.
- Such glyphs have the advantage of being able to convey a signed direction, whereas lines convey an unsigned direction only.
- However, these glyphs also take more space to draw, so they increase the clutter or require lower-resolution datasets.

Vector Glyphs: Cone and Arrow glyphs



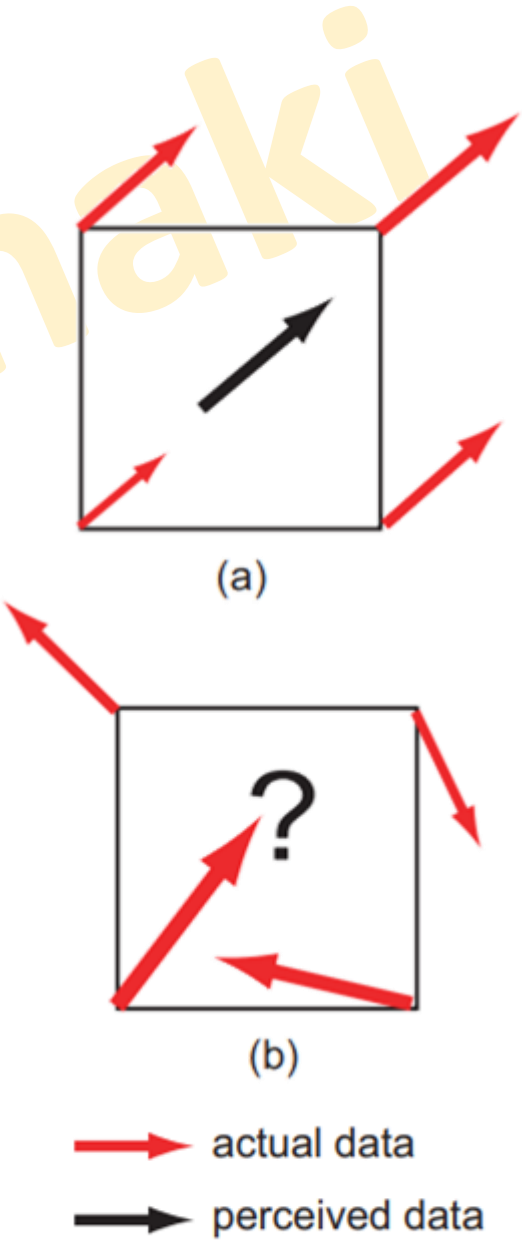
Cones



Arrows

Vector Glyphs

- In the first case (see Figure (a)), the vector field variation over the displayed cell is quite small.
- We can easily interpolate mentally the displayed arrow glyphs and arrive at the conclusion that the vector field has an upper-right direction and orientation, and increases in magnitude in this direction.
- In the second case (see Figure b), the situation is more problematic.
- The vector field varies greatly between the vertices of the considered cell, so it is harder to mentally interpolate between these four vector glyphs and get an idea of how the field actually behaves over the considered surface.
- Clearly, the interpretation can get very confusing when we have hundreds of cells in this situation.

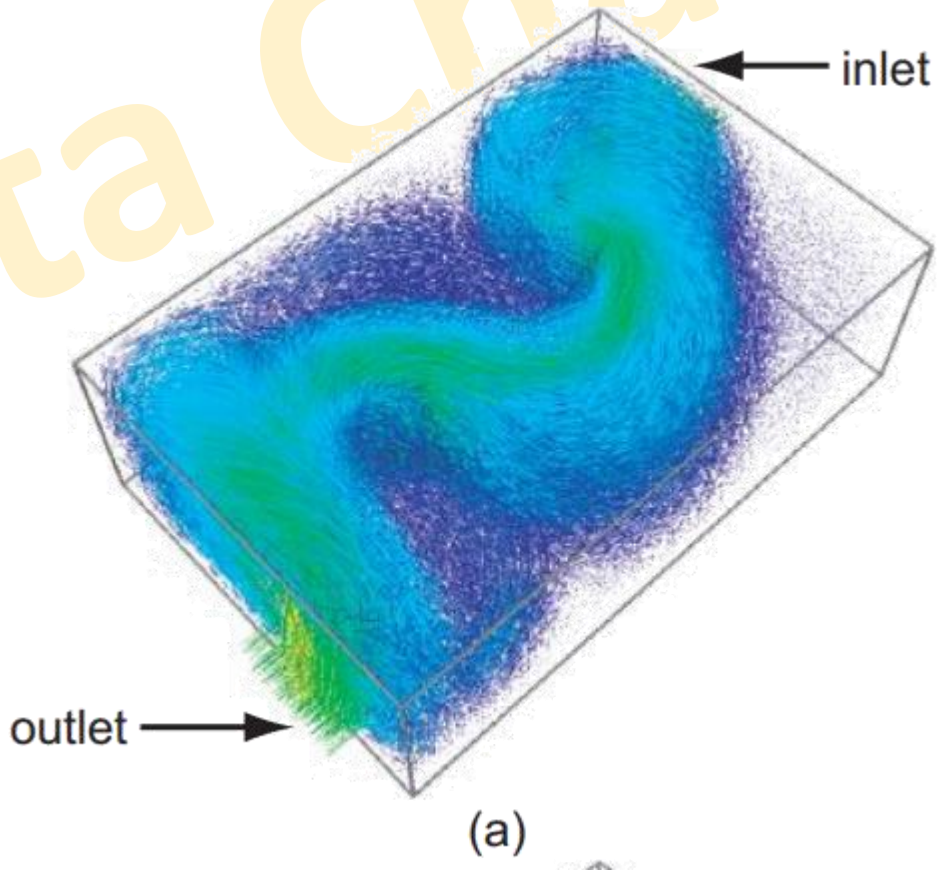


Vector glyphs in 3D

- Figure shows an arrow glyph visualization of a 3D vector dataset sampled on a uniform grid containing 128 × 85 × 42 data points that describes the flow of water in a box-shaped basin that has an inlet, located upper-right, an outlet, located lower-left, and two obstacles (not drawn in the figure) that cause the sinuous behavior of the flow.
- Visualizing such a dataset with vector glyphs at full resolution would produce a completely cluttered result.
- Randomly subsampling the dataset to 100,000 points and visualizing it with line glyphs produces the result shown in Figure (a).

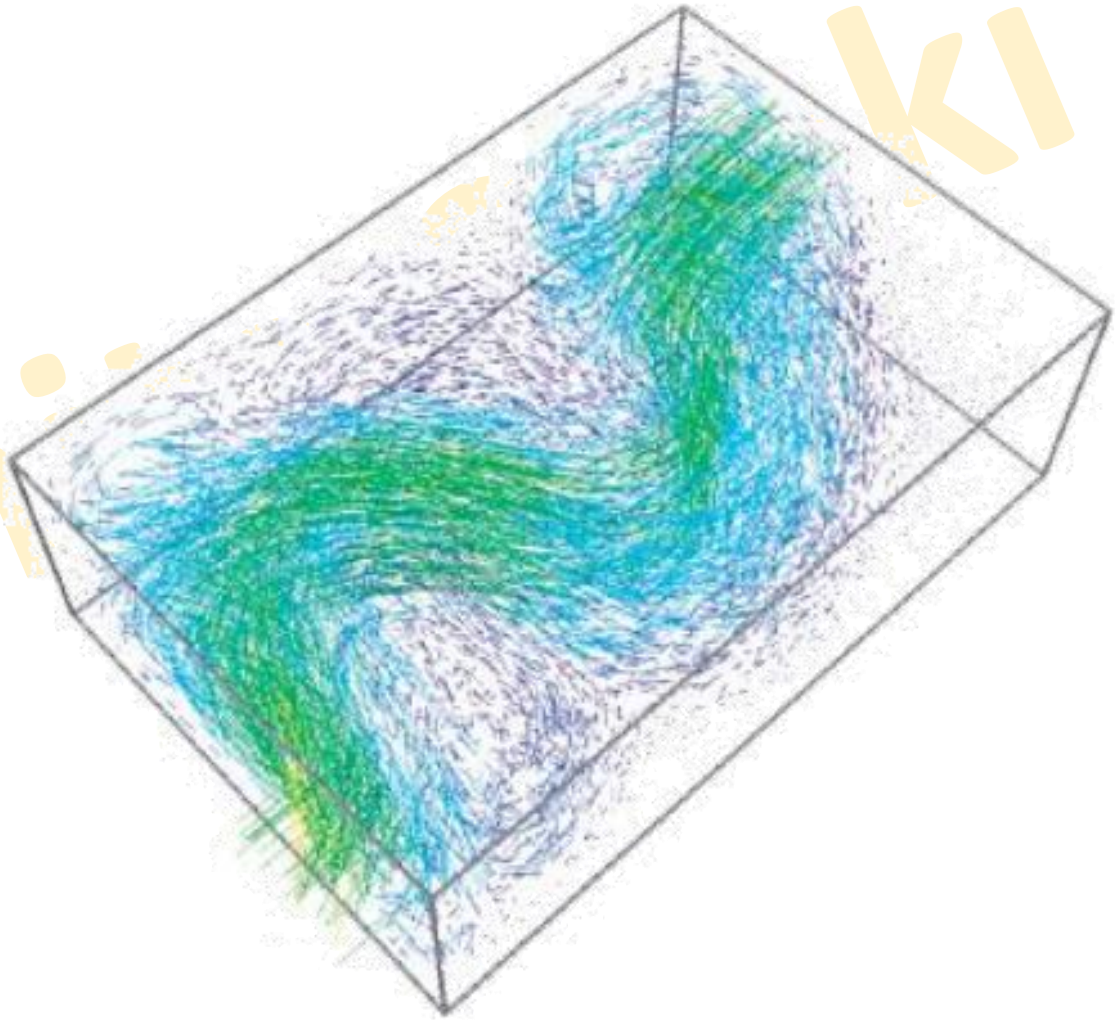
Vector glyphs in 3D

- Besides the known problems of glyphs in 2D, an additional problem of 3D glyph visualization becomes apparent here: occlusion.
- Closer glyphs obscure further ones, which makes understanding the flow behavior deep inside the dataset quite difficult.
- Note that using arrow instead of line glyphs only increases the occlusion problem, as arrows have a larger screen area than lines.



Vector glyphs in 3D

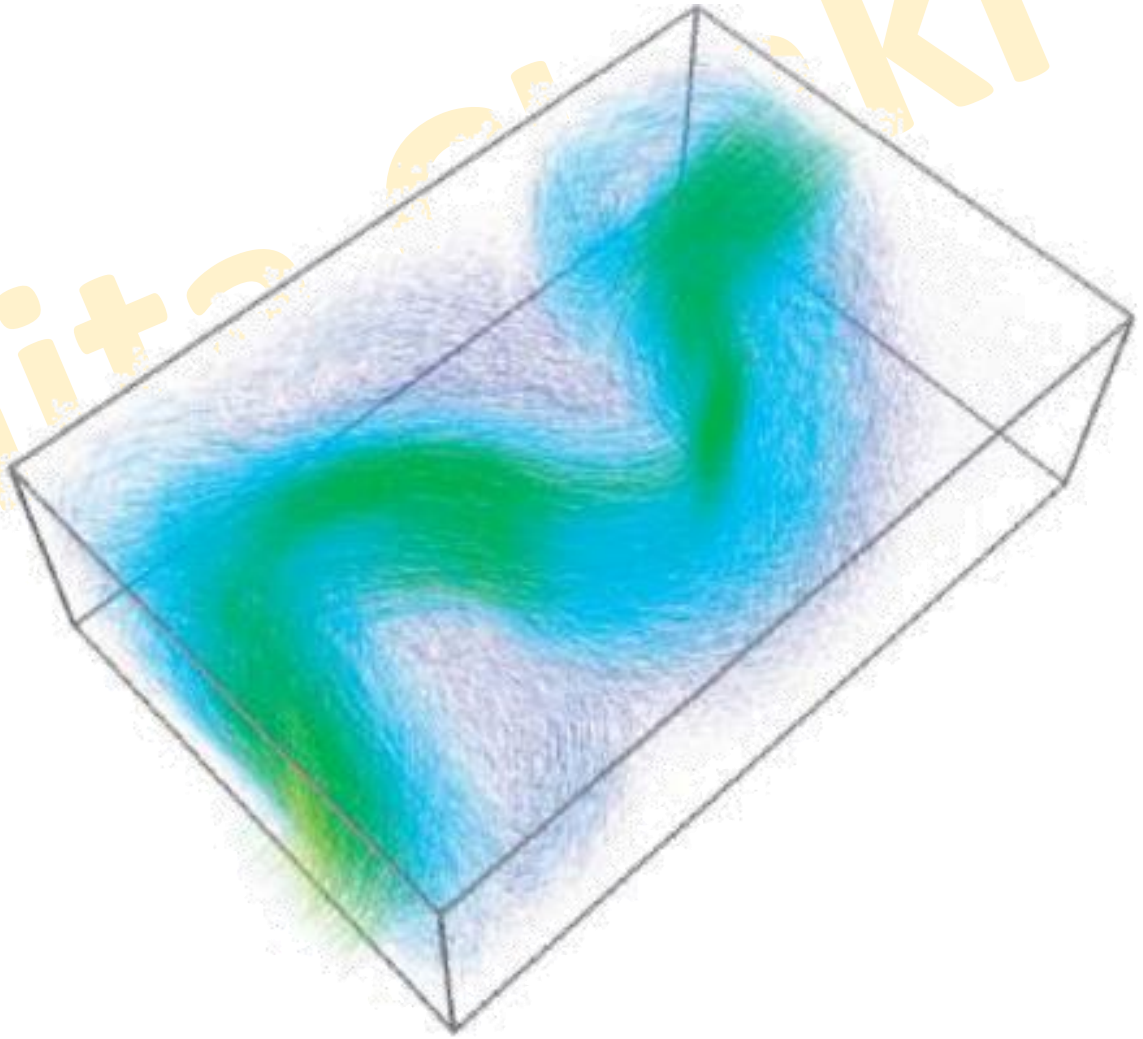
- We can alleviate the occlusion problem by further subsampling the dataset to only 10,000 data points (see Figure (b)).
- However, the dataset is now too sparse to be able to distinguish local details.



(b)

Vector glyphs in 3D

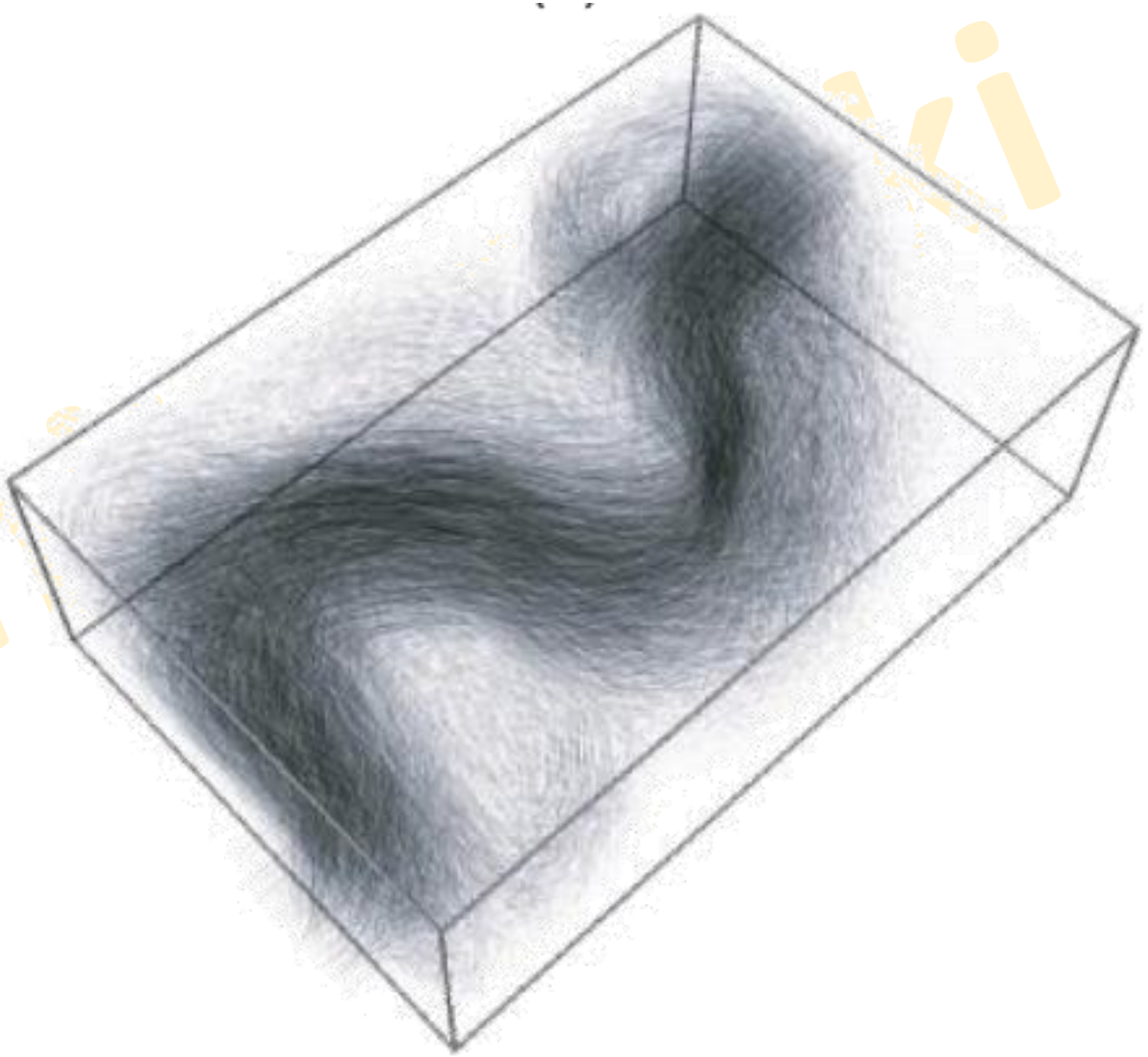
- A different way to tackle the occlusion problem is to draw the glyphs transparently. Figure (c) shows the same visualization as in Figure (a), but this time using line glyphs with a transparency of 0.15.
- Closer glyphs now cause less occlusion, allowing us to “see” deeper inside the dataset.



(c)

Vector glyphs in 3D

- An interesting visual effect is achieved by using monochrome, instead of color mapped, transparent line glyphs (see Figure (d)).
- Here, a single color (black) is blended, so the resulting visualization is easier to interpret.
- The high velocity “flow core” located at the center of the fluid flow is now easily visible as a dark region.



(d)

Scalar Vs Vector Visualization

- This difference between scalar (color-mapped) visualizations and vector (glyph) visualizations can be explained in sampling terms.
  - Scalar color-mapping techniques produce a piecewise linear visualization.
  - Glyph techniques produce a purely discrete visualization.
  - In the first case, we do not have to mentally interpolate between drawn pixels, as the graphics hardware has done this task for us.
  - In the second case, we only have visual indication at the sample points (e.g., cell vertices), so we must do this interpolation ourselves.