UNIT-2

Contents

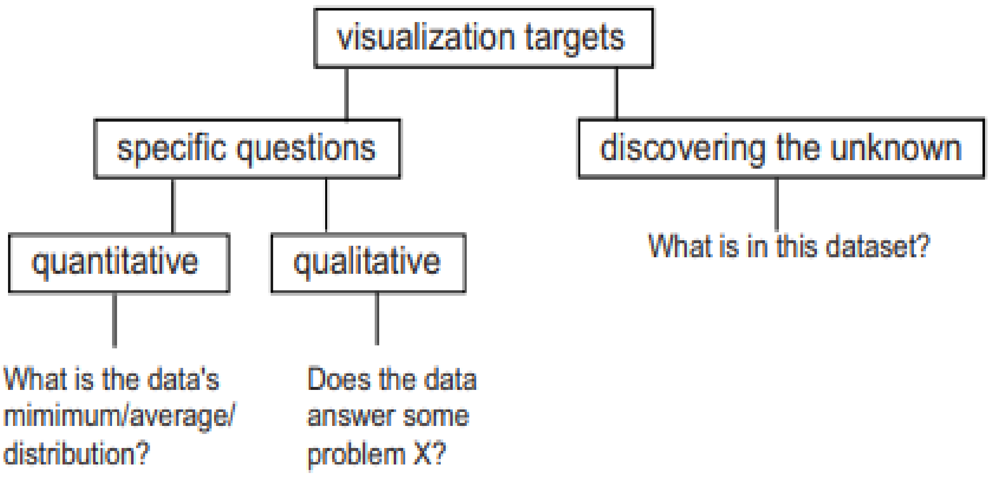


Fig.: Types of questions targeted by the visualization process

The Visualization Pipeline

Conceptual perspective:

- The role of visualization is to create images that convey various types of insight into a given process.
- The visualization process consists of the sequence of steps, or operations, that manipulate the data produced by the process under study and ultimately deliver the desired images

Conceptual Perspective of Visualization process

Functional view on the visualization pipeline

Visualization Pipeline

The visualization process consists of several stages. Each stage is modelled by a specific data transformation operation.

The input data "flows" through this pipeline, being transformed in various ways, until it generates the output images.

Given this model, the sequence of data transformations that take place in the visualization process is often called the visualization pipeline.

The visualization pipeline typically has four stages: data importing, data filtering and enrichment, data mapping, and data rendering

On design level: Visualizations allows one to manage the complexity of the whole process

On implementation level: construct visualizations by assembling reusable and modular data-processing operations

Visualization pipeline as a function V is that maps between D_I , the set of all possible types of raw input data, and the set I of produced images: Vis : $D_I \rightarrow I$

Also model this process by a function Insight, in inverse direction to the V is function: Insight : $I \rightarrow D_I$

Insight maps from the produced images to the actual questions the user has about the raw data

Monitoring live process – need for users in changing its parameters.

Applications provide close the loop between the visualization output and the application's inputs.

If the user completes round trip time effectively steers the process at hand by means of visual feedback. This process, called computational steering,

Data Importing

To import our input data into the visualization process.

Functional terms: importing the input data maps the raw information D_I that is available at the beginning of the visualization process to a dataset $D \in D$,

$$\text{Import} : D_I \rightarrow D$$

D consists of uniform, rectilinear, structured, and unstructured datasets.

Data Importing: One-to-one mapping, Reading the input data from some external storage, Direct mapping, translating between different data storage formats, resampling the data from the continuous to discrete. Choice made during data importing determine the quality and effectiveness of visualization. Preserve available input information

Data Filtering and Enrichment

After importing, decision about important and interested features. Distil our raw dataset into more appropriate representations; also called enriched datasets that encode our features of interest in a more appropriate form for analysis and visualization. This process is called data filtering or data enriching.

This process performs two tasks: On one hand, data is filtered to extract relevant information. On the other hand, data is enriched with higher-level information that supports a given task.

Data filtering can be described by the function Filter: $D \rightarrow D$.

Both the input (domain) and output (codomain) of the filtering function are **datasets**

Contexts where filtering is important

See what is relevant: medical specialists- CT or MRI scanners, financial analysts-stock prices

Handle large data: Problem for efficient visualization: Limited output resolution

* Solution- Subsampling(zooming)-subsets of pixels that captures the all characteristics of the complete dataset
* Panning-selecting subset of the input image at its original resolution

Ease of use: Convenience and Need to transform – apply necessary operation

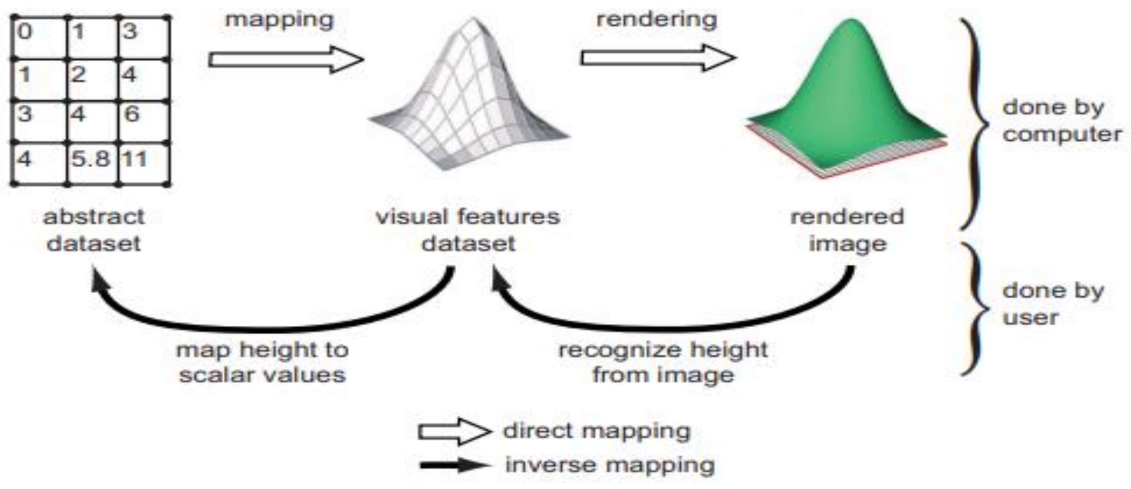
Data Mapping

The filtering operation produces an enriched dataset that should directly represent the features of interest for a specific exploration task. Visualization process -mapping can be modelled by the function Map: $D \rightarrow D_V$

This function takes a dataset $D \in D$ and maps it to a dataset of visual features D_V .The visual domain D_V is a multidimensional space whose axes, or dimensions, are those elements that we perceive as quasi-independent visual attribute. The visual feature is a colored, shaded, textured, and animated 2D or 3D shape.

Map function: used in a specific application, depend on purpose of the visualization, specifics of the data, the preference of the designer of that visualization. Example: mapping 2D and 3D coordinates. The Direct and Inverse Mapping in the visualization process

Why to extract visual features?



Rendering is applying computer graphics techniques, such as coordinate transformations, lighting, texture mapping etc., Data mapping targets in making the invisible and multidimensional data visible and low-dimensional. In implementation practice, Filter and Import may be not invertible.
if Map and Render are invertible, make judgments about the enriched datasets D that model our problem domain, using the rendered images, For example, a 3D scene rendered from a bad angle and with low lighting will produce an bad image an image which does not tell us anything insightful. Map should be injective, it is invertible over its whole value range, which is what we want.

Data Mapping:

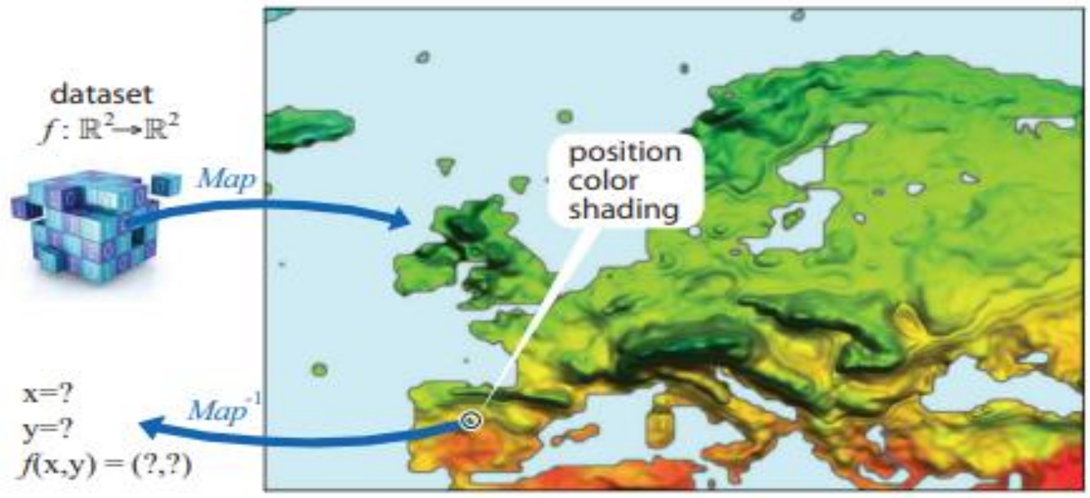
- Encodes explicit design decisions of what and how
- Converts invisible data to visible representations
- Specifies those attributes that encode actual data

Rendering

- Simulates the physical process to visible 3D scene
- Specifies visual attributes to tune to their taste

Modularity

- Both operations implementation is complex and has numerous sub-steps
- 3D rendering libraries are reused in a visualization application once the rendering step is separated from the mapping step.



Data Mapping

Distance preservation

- The distance d(x1, x2) between any two values x1 and x2 in the dataset D should suggest the distance d'(Map(x1), Map(x2)) in the visual feature dataset D_V.
- Linear Mapping: Numerical value to visual attribute
- Measurement mappings: the empirical relations preserve and are preserved by the numerical relations
- To map rainfall values: a color map that translates scalar values to hues that works
- Organization levels: Types of operations that perform on variables.
- Associative: if v allows a categorical attribute mapped by v to be perceived independently on the presence of other visual variables in the same image. Example: Shape

Data Rendering

- The final step of the visualization process.
- Takes the 3D scene created by the mapping operation, together with several user specified viewing parameters such as the viewpoint and lighting, and renders it to produce the desired images: Render : $D_V \rightarrow I$
- This allows users to interactively navigate and examine the rendered result of a given visualization by rendering the 3D scene without having to re-compute the mapping operation.
- If the view point changes, mapping remains same but have to do render with new viewing parameters

Implementation Perspective:

- Visualization pipeline as a composition of functions that have dataset arguments and values.

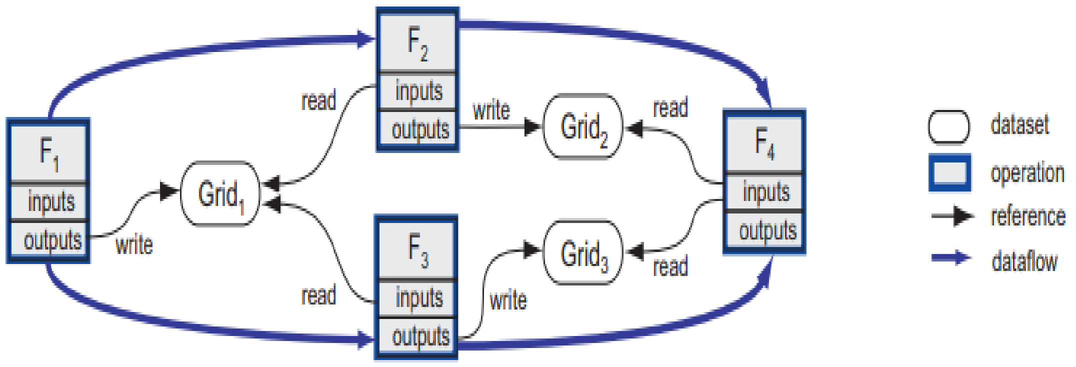
$$Vis = F1 \circ F2 \circ ... \circ Fn, \text{ where } Fi : D \rightarrow D.$$

Where Fi perform the data rendering, mapping, filtering, and importing and in inverse order
This model allows us to decompose each of the four stages into many sub-functions.
First, complex operations, such as filtering, in terms of a composition of simple filter-like atomic operations that each address a specific task.
Second, this favors modular, reusable software design and allows us to assemble visualization applications from a set of predefined functional components.

- Dataflow design: Implement the functions Fi as classes that have three properties:
- They read one or more input datasets $D^{inp}{}_i$.
- They write one or more output datasets $D^{out}{}_j$.
- They have an execute() operation that computes $D^{out}{}_j$ given $D^{inp}{}_i$.
- The choice of letting the function F have several datasets as input (arguments) and output (results)

```
class F
{
public:
    void              setInput(Grid*,int);
    Grid*             getOutput(Grid*,int);
    virtual void      execute() =0;
protected:
    vector<Grid*>     inputs;
    vector<Grid*>     outputs;
};
```

- Input raw data into the pipeline, first execute F_1
- If the application graph is acyclic, the execution is equivalent to calling the execute() method of all operations Fi in the order of the topological sorting of the graph
- The sequence of operation executions in the "flow" of data from the importing operation to the final rendering operation.
- Progressive updation and serialization



Implementation Perspective

- This application model follow the "flow" of data from the importing operation to the final rendering operation, called a dataflow application model.
- a visualization application can be implemented as a network of operation objects that have dataset objects as inputs and outputs. To execute the application, the operations are invoked in the dataflow order, starting with the data importing and ending with the rendering.
- most notable additions:
- reference counting
- automatic memory management
- ensure the dataset-operation compatibility, smart pipeline traversal,
- parallelization-distribution of execution on one or several machines,
- progressive update mechanisms that allow users to stop the pipeline execution at any desired moment
- serialization facilities for both the datasets and the application

Implementation Perspective
Dataflow implementation:
Visualization Toolkit (VTK)- a professional visualization framework in both academic and industrial contexts.

- VTK is an open-source product, easily modified and embedded in different development. **C++,**Java, Python
- VTK toolkit great flexibility, genericity, efficiency and price with complexity.
- **Insight Toolkit (ITK):** general-purpose data visualization
- ITK focuses on the more specific field of image segmentation, processing, and registration.

- ITK can handle multidimensional images and offers algorithms for thresholding, edge detection, smoothing, denoising, distance computations, segmentation, and registration.
- ITK offers the same wrapper concepts, open-source development model as VTK.

Visual dataflow programming:

- Constructs dataflow application – assembling **iconic** representations to visualize operations.
- **Icons** - describing the operations- their inputs and outputs are connected by means of mouse manipulations.
- **Graphical user interfaces (GUIs**) are provided by the environment to let users interactively control the parameters of the various visualization operations, achieving the goal of interactive data exploration.
- When the user modifies such a parameter, the environment triggers a dataflow execution engine that updates the complete application network from the affected operations onward until a new image is rendered in the visualization window

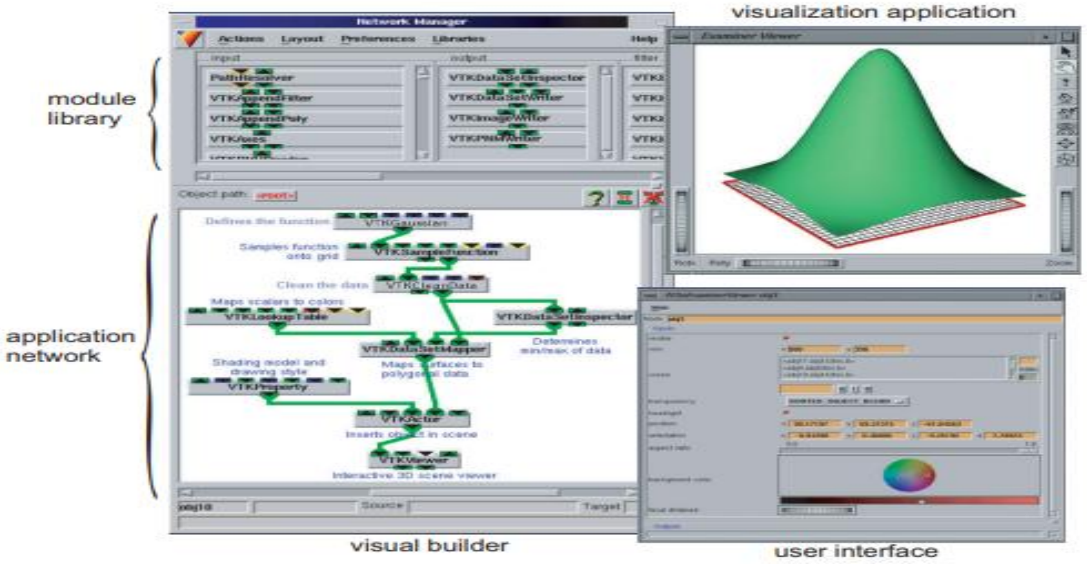


Fig.: The height-plot application in the VISSION application builder

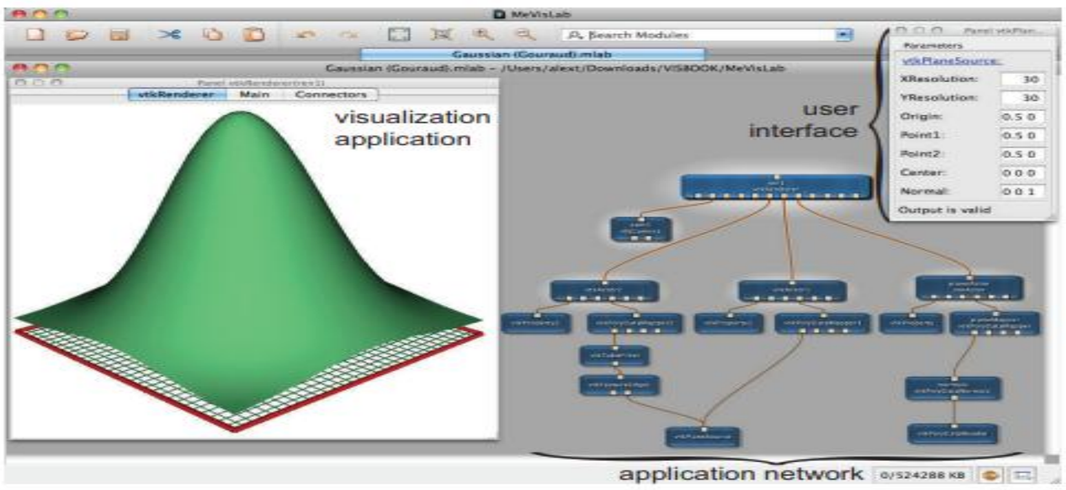


Fig.:The height-plot application in the MeVisLab application builder

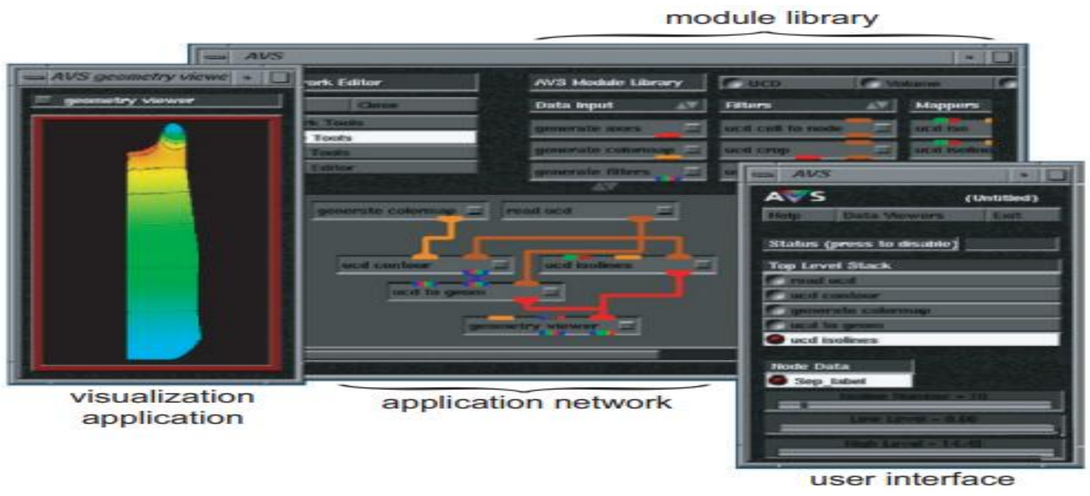


Fig.: A visualization application in the AVS application builder

Simplified visual programming:

- visual application builder is **ParaView** environment.
- uses the **VTK** library and its underlying machinery to provide the actual implementation of visualization operations.
- ParaView features a more **beginner-friendly end-user** interface.
- Constructs application networks of fully general graph topology
- GUI menus that is easier and faster to learn and use.
- A similar trade-off of design freedom for utilization simplicity is used in other toolkits, such as MayaVi
- **Issues** in visual programming environments are:
- At initial stages **rapid Prototyping of users** with no programming skills.
- **Less effective** for complex visualization applications
- Need complex custom code **to Intricate control flow**
- Structure of application changes **Creation of final applications**
- Algorithm Classification
- To start learning about number of specific visualization techniques with theoretical and practical parameters are called visualization algorithms
- Schroeder et al. propose a structural classification that groups visualization techniques by the type of dataset ingredient they change.
- Their classification includes
- geometric techniques (that alter the geometry, or locations, of sample points),
- topological techniques (that alter the grid cells),
- attributes techniques (that alter the attributes only), and
- combined techniques (that alter several of a dataset's ingredients)

Algorithm Classification

- Marcus et al. in terms of a five-dimensional model containing the following dimensions:
- Task: What is the task to be completed?
- Audience: Which are the users?
- Target: What is the data to visualize?
- Medium: What is rendering (drawing) support?
- Representation: What are the graphical attributes (shapes, colors, textures) used?
- This classification is applicable not only for visualization algorithms but also for visualization Applications.
- Many users think of (scientific) visualization methods in terms of scalar, vector, tensor, and domain modeling algorithms.

Scalar Visualization

- Visualizing scalar data is encountered in science, engineering, and medicine, but also in daily life.
- scalar datasets, or scalar fields, represent functions $f : D \rightarrow R$, where D is usually a subset of R^2 or R^3. There exist many scalar visualization techniques, both 2D and 3D datasets.
- Color Mapping: associates a color with every scalar value, which is a mapping function $m : D \rightarrow D_V$
- Not concerned with creating shapes to visualize data.
- For every point of the domain of interest D,
- color mapping applies a function $c : R \rightarrow$ Colors that assigns to that point a color $c(s) \in$ Colors which depends on the scalar value s at that point.
- Ways to define scalar-to-color function c

Color look-up tables

- **Transfer function: A color look-up table C**, also called a colormap, is a uniform sampling of the color-mapping function c:
- A table of N colors $c_1,...,c_N$, which are associated with the scalar dataset values f, assumed to be in the range $\mathbf{[f_{min}, f_{max}]}$.
- 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}.
- Dataset values outside the prescribed range [fmin, fmax] are clamped to this range to yield valid colors in the given colormap
- To visualize a time-dependent scalar field f(t) with $t \in [t_{min}, t_{max}]$.
- Solution is to visualize the color-mapped values of the scalar field f(t) for consecutive values of t in $[t_{min}, t_{max}]$.
- If the range $[f_{min}(t_i), f_{max}(t_i)]$ of a time step $t_i \in [t_{min}, t_{max}]$ is much smaller than the absolute range $[f_{min}, f_{max}]$, normalizing f to the absolute range at the individual frames.
- Another solution is to normalize the scalar range separately for every time frame f(t). This implies drawing different color legends for every time frame.
- colors are usually represented as triplets in either the RGB or HSV (hue-saturation-value) color systems, this is usually done by defining three scalar functions
- $c_R : R \rightarrow R$, $c_G : R \rightarrow R$, and $c_B : R \rightarrow R$, whereby c = (cR, cG, cB). The functions c_R , c_G, and c_B are also called transfer functions.

Designing Effective Colormaps

- color-mapping visualization is **effective** if, by looking at the generated colors, we can easily and accurately **make statements about the original scalar dataset** that was color mapped.
- Different **analysis statements and goals** require different types of colormaps:
  1. Absolute values: Tell the absolute data values at all points in the displayed dataset. **(absolute)**
  2. Value ordering: Given two points in the displayed dataset, tell which of the corresponding two data values is greater. **(large or small**)
  3. Value difference: Given two points in the displayed dataset, tell what is the difference of data values at these points.**(far apart)**
  4. Selected values: Given a particular data value $f_{interest}$, tell which points in the displayed data take the respective value $f_{interest}$. A variation of this goal replaces $f_{interest}$ by a compact interval of data values. **(chosen value**)
  5. Value change: Tell the speed of change, or first derivative, of the data values at given points in the displayed **dataset(quick change of values)**

Color Legends

- Invert the color-mapping function c; that is, look at a color of some point in the visual domain D_V and tell its scalar value f.
- A color strip containing all the colors c_i in our colormap, annotated with labels that indicate the values f for all or a number of the depicted colors.
- By looking at an actual visualization and comparing its colors with the labeled colors in the colormap, we are able to infer the scalar values of the depicted dataset at desired points in the drawn image.
- Color function c must be invertible.

This function must be injective- every scalar value in the range $[f_{min}, f_{max}]$ is associated with a unique color.

1. Able to map the colors to scalars using the color legend that can be perceived visually.
2. Visually distinguish separate regions having different colors.(spatial resolution compared to the speed of variation of scalar data).

Goal 1: Color legend is required to map a color to a data-related quantity.

Goal 2: Color legend is required to tell how colors are ordered with respect to the ordering of the data values.

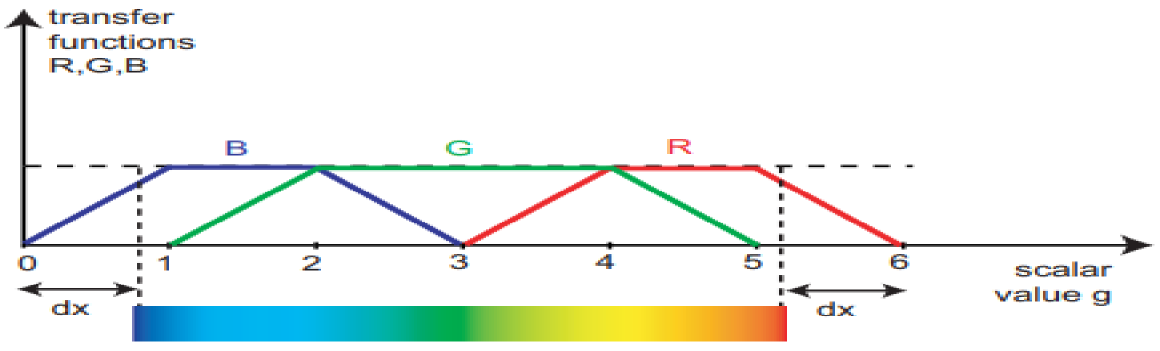
Goal 3: Compare distances between data values

Goal 4: Data points take a given value of interest

Goal 5: tell the speed of data variation the magnitude of the gradient ∇f of our scalar signal f, we then need a color legend for ∇f .

Rainbow Color Map

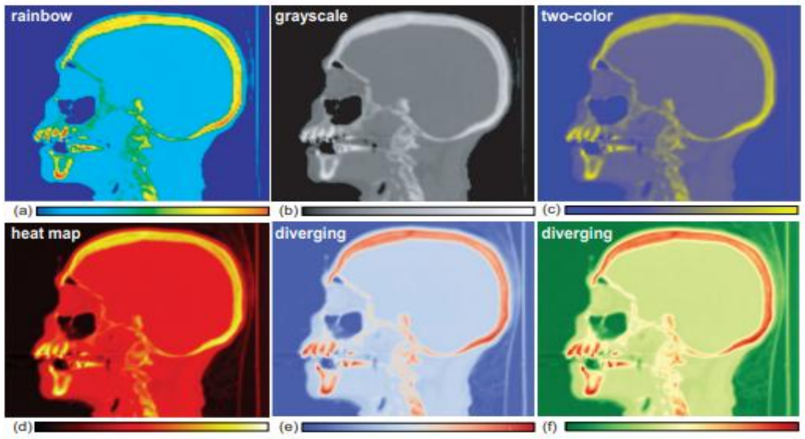
- Many engineering and weather forecast applications use a blue-to-red colormap, often called the rainbow colormap (see Figure 5.1).
- This colormap is based on intuition that blue -"cold" color and red – "hot " color.
- construct a rainbow colormap using three transfer functions R, G, B



Rainbow Color Map

- **Limitations of Rainbow colormap:**
- Focus: Warm colors attract attention more than cold colors. Depending on the application, these may not be the values where we want to focus on.
- Luminance: In the figure the luminance is slightly increasing, respectively slightly decreasing luminance of the rainbow colormap entries vary non-monotonically. This leads to users being potentially attracted more to certain colors. This issue can be corrected by adjusting the rainbow colormap entries so that they use the same hues, but have maximal luminance.
- Context: Hues can have application-dependent semantics.
- Assumption of rainbow colormap is "warm" colors, such as yellow and red, are perceived as being associated with higher data values, whereas "cold" colors such as blue suggest low values.
- Ordering: The rainbow colormap assumes that users can easily order hues from blue to green to yellow to red, such as used by this colormap.

- • Linearity: Besides the colormap invertibility requirement, visualization applications often also require a linearity constraint to be satisfied.



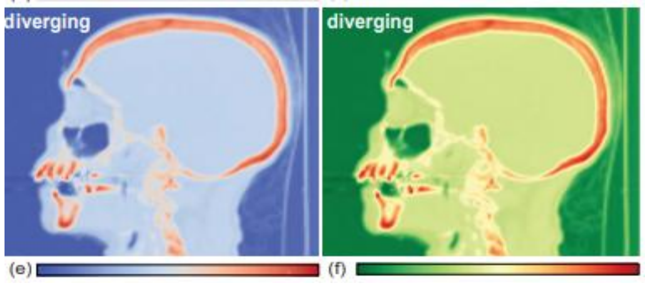
Gray Scale:
- First, it directly encodes data into luminance, and thus is has no issues regarding the discrimination of different hues.
- Second, color ordering is natural (from dark to bright), which helps goal 2.
- Finally, rendering grayscale images is less sensitive to color reproduction variability issues when targeting a range of display or print devices.

on the negative side, telling differences between two gray values, or addressing goal 3, is harder than when using hue-based colormaps.
Two-hue: Figure (c) shows a two-hue colormap. The colormap entries are obtained by linearly interpolating between two user-selected colors, blue and yellow in our case.
If the two colors used for interpolation are perceptually quite different , a disadvantage offers less dynamic range.
- The disadvantage of this design is that less colors can be individually perceived, leading to challenges for goals 1, 4, and 5

Heat map: Figure 5.2(d) shows a heated body colormap.
- 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.
- Compared to the two-hue colormap, the heat map uses more hues, thus allowing one to discriminate between more data values(using yellow at the highest end rather than white)
- Not suitable for color shading data on 3D shaded surfaces due strong dependence on luminance.
- Combination of grayscale map and the heat map is a popular choice for medical data visualization.

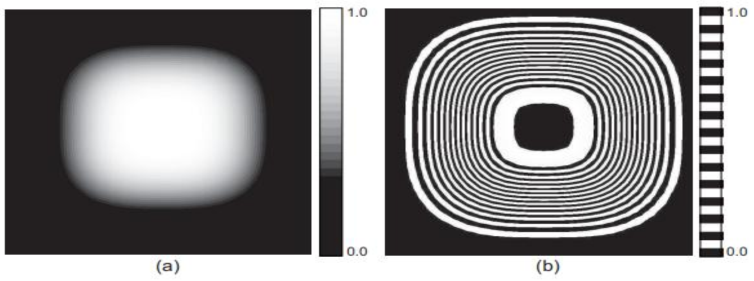
Diverging
- Diverging: Figures 5.2(e,f) show two final colormap examples, known under the name of diverging, or double-ended scale, colormaps.
- Diverging colormaps are constructed starting from two typically isoluminant hues, just as the isoluminant two-hue colormaps. However, rather than interpolating 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} .
- Figure(e) a diverging colormap with c_{min} = blue, c_{max} = red, and c_{mid} = white.
- Figure(f) used a diverging color map with c_{min} = green, c_{max} = red, and c_{mid} = bright yellow.

- Diverging colormaps consist of two two-hue colormaps for the left, respectively right, halves of the considered data range.
- to emphasize the deviation of data values from the average value f_{mid}, and also effectively support the task of assessing value differences.



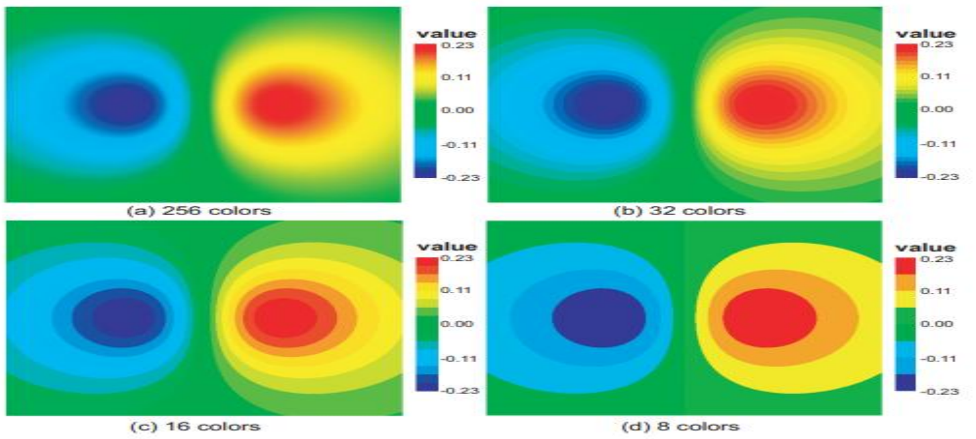
Zebra colormap:

- To emphasize the variations of the data rather than absolute data values.
- the continuous colormaps presented in Figure 5.2 on the magnitude of the gradient ∇f of our scalar dataset f.
- the absolute value of our data's rate of change, not the direction in which that rate of change is maximal.
- use a colormap on the scalar dataset f containing two or more alternating colors that are perceptually very different.
- In the left image, we visualize a scalar function $f(x, y) = e^{-10(x4+y4)}$, whose shape is quite similar to the Gaussian.
- Thin, dense stripes indicate regions of high variation speed of the function, whereas the flat areas in the center and at the periphery of the image indicate regions of slower variation



Contouring

- Color banding: Colormap design is the choice of the number of colors N. Choosing a small N would inevitably lead to the color banding effect. As the number of different colors in the look-up table decreases, equal color bands become visible in both the image and the color legend. Color banding is related to a different usage of color mapping—visualizing categorical data.
- Color banding is related to fundamental and widely used visualization technique called contouring.
- Meaning of the sharp color transitions that separate the color bands.
- The transition between the yellow and orange bands; that is, all points in the figure that are on the border separating these two colors.
- 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.
- The points located on the color border itself have the scalar value s = 0.11.
- Points located on such a color border, drawn in black in Figure, are called a contour line, or isoline. Formally, a contour line C is defined as all points p in a dataset D that have the same scalar value, or isovalue $s(p) = x$, or $C(x) = \{p \in D | s(p) = x\}$.

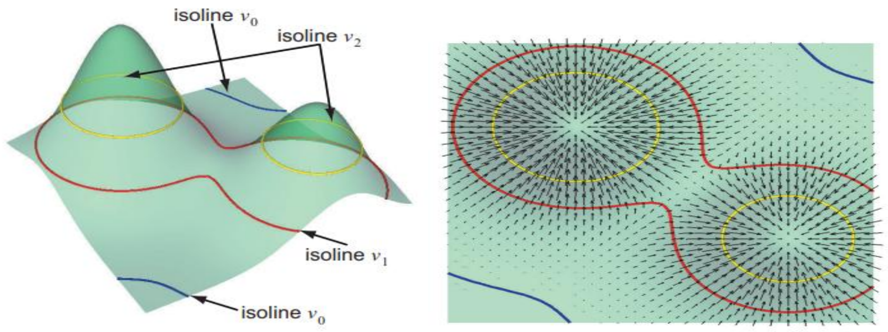


- Areas where contours are closer to each other, such as in the center of the image, indicate higher variations of the scalar data. Indeed, the scalar distance between consecutive contours (which is constant) divided by the spatial distance between the same contours (which is not constant) is exactly the derivative of the scalar signal.

Contour properties:

- A two-variable function z = f(x, y) with the familiar elevation plot technique. Over the function graph, three isolines are drawn, for three different values, v0 (blue), v1 (red), and v2 (yellow).
- First, isolines can be either closed curves, such as the yellow isoline or open curves.
- Second an isoline can never intersect

The scalar function and its isolines are viewed along the z-axis. The vector field displayed in the image shows the gradient of the scalar function.



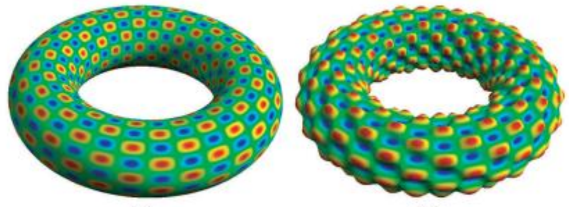
- The gradient of a function is the direction of the function's maximal variation, whereas contours are points of equal function value, so the tangent to a contour is the direction of the function's minimal (zero) variation.
- These properties hold for a continuous dataset

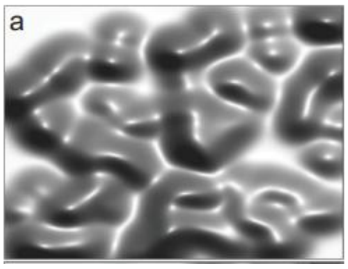
Height Plots

- Height plots, also called elevation or carpet plots.
- Given a two-dimensional surface $D_s \in D$, part of a scalar dataset D, height plots can be described by the mapping operation

$m : D_s \rightarrow D$, $m(x) = x + s(x)n(x)$, $\forall x \in D_s$, 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.

The height-plot mapping operation "warps" a given surface D_s included in the dataset along the surface normal, with a factor proportional to the scalar values.


(a) (b)



- Consider a scalar dataset given by a function $f : D \to R$, and its height plot given by the mapping
- $z(x, y) = sf(x, y)$, for all points $(x, y) \in D$. Here, $s > 0$ is the plot's scaling factor.
- Here, $s > 0$ is the plot's scaling factor. Figure (a) can view the height plot from above, looking along the $-z$-axis.

- Next, instead of the linear mapping $z = sf$, we use a non-linear mapping given by
  $z(x, y) = sf(x, y) + sh\, g\,(f(x, y) \bmod h/h)$ where $g(u) = au(1-u)$ is a parabolic function to add parabolic bumps of height a to consecutive intervals in the range of f of size h. The parameter $a \in [0, 1]$ sets the strength of the bump effect. The parameter h controls how "thick" the resulting bands are, and is similar in function to the distance between consecutive contours in isoline plots.