Data Representation and Scalar Algorithms

Anders Hast

with some slides by
Alexandru C. Telea
(indicated)
Scientific Visualization Module 2

Data representation

prof. dr. Alexandru (Alex) Telea

Department of Mathematics and Computer Science
University of Groningen, the Netherlands
The Visualization Pipeline - Recall

Dataset → Process → Dataset → Process → Dataset → Process → Dataset → Process → Dataset

Any kind of data → formatted data → filtered data → spatial data → 2D image

Data acquisition → data enriching, transformation, resampling... → map abstract data to visual representations → draw visual representations

End user

Insight into the original phenomenon
## The Visualization Pipeline - Recall

<table>
<thead>
<tr>
<th>any kind of data</th>
<th>formatted data</th>
<th>filtered data</th>
<th>spatial data</th>
<th>2D image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>→ Input</td>
<td>→ Filtering</td>
<td>→ Mapping</td>
<td>→ Rendering</td>
</tr>
</tbody>
</table>

### 1. Input data
- your primary “raw” source of information
- can be anything (measurements, simulations, databases, …)

### 2. Formatted data
- converted to points, cells, attributes (discussed next in this module)
- Ready to use for visualization algorithms

### 3. Filtered data
- eliminates the unneeded data, adds the needed information
- read and written by visualization algorithms

### 4. Spatial (mapped) data
- has spatial embedding → can be drawn

### 5. 2D Image
- final image you look at to get your answers
Scientific Visualization - The Dataset

Dataset

- key notion in visualization (SciVis, InfoVis, SoftVis)
- captures all relevant characteristics of a data collection
  - structure
  - data values
  - data operations

We’ll detail all these next
Visualization data properties

Sampling (data importing)

Reconstruction

Domain $f : D \rightarrow C$

Co-domain

Continuous data

$f(p_i) = f(p_i) = f_i$

Sampling

$\{p_i, f_i\}$ Measurements (samples) at discrete set of points

Reconstruction

$f : D \rightarrow C$ Continuous data, as close as possible to input
Continuous data

Figure 3.1. Function continuity. (a) Discontinuous function. (b) First-order $C^0$ continuous function. (c) High-order $C^k$ continuous function.

Cauchy definition of continuity

A function $f$ is continuous iff

$$\forall \epsilon > 0, \exists \delta > 0 \text{ such that if } ||x - p|| < \delta, x \in C \text{ then } ||f(x) - f(p)|| < \epsilon.$$ 

$C^{-1}$ discontinuous (graph of function has “holes”)

$C^0$ first-order continuous (graph of function has “kinks”)

$C^k$ first $k$ derivatives of the function are continuous
Sampled data

Functional properties

• **finite**
  • captures continuous signal at a finite set of points (measurements)

• **accurate**
  • can reconstruct a signal close to input accurately
  • reconstruction guarantees continuity properties

Non-functional properties

• **efficient**
  • reconstruction is fast

• **compact**
  • store Gbytes of sample points compactly

• **generic**
  • few data structures cover most dataset types

• **simple**
  • learn to create & use such data structures quickly
Why Interpolate?

- Interpolation usually gives a better representation of the sampled data.
- Interpolation is always a “guess” of what the “missing” data would be like.

https://en.wikipedia.org/wiki/Interpolation
**Interpolation**

Fundamental tool for signal reconstruction

1. **Reconstruction** formula

\[
\tilde{f} = \sum_{i=1}^{N} f_i \phi_i \\
\phi_i : D \rightarrow C 
\]

are basis (or interpolation) functions

Doesn’t necessarily pass through sample values = approximation

2. **Interpolation**: reconstruction passes through (interpolates) the sampled values

\[
\sum_{i=1}^{N} f_i \phi_i(p_j) = f_j, \forall j. 
\]

because \( \tilde{f}(p_i) = f(p_i) = f_i \)

3. **Orthogonality** of basis functions

\[
\phi_i(p_j) = \begin{cases} 
1, & i = j, \\
0, & i \neq j.
\end{cases}
\]

why? Just apply (2) to \( f = \begin{cases} 
1, & p = p_j \\
0, & p \neq p_j
\end{cases} \)

4. **Normality** of basis functions

\[
\sum_{i=1}^{N} \phi_i(x) = 1, \forall x \in D 
\]

why? \( \sum_{i=1}^{N} \phi_i(p_j) = 1, \forall p_j \) (sum (3) over \( i = 1 \ldots N \)) and apply above to all \( p_i \in D \)
Interpolation, Examples

Linear

Polynomial

https://en.wikipedia.org/wiki/Interpolation
Linear Interpolation

- Linear Interpolation:
  \[ I = I_0 (1-u) + I_1 u \]

- \( u \) varies from 0 to 1.

- Expand
  \[ I = I_0 - uI_0 + I_1 u \]

The line equation!!!

- Basis (blending) Functions
Practical interpolation: Cells

Recall the interpolation formula

\[ \tilde{f} = \sum_{i=1}^{N} f_i \phi_i \]

This becomes very inefficient if

- \( N \) is very large and we have to evaluate \( \phi_i \) at all these \( N \) points
- \( \phi_i \) have complicated expressions

Practical basis functions

- are non-zero over small spatial ‘pieces’ of \( D \) only (limited support)
- have the same simple formula at all sample points \( p_i \)

Note:
Cubic Polynomials are often used in Computer graphics

We will discretize our spatial domain \( D \) into cells
Cells: 1D space

Consider a simple 1D function $f : \mathbb{R} \rightarrow \mathbb{R}$

1. Sample the 1D axis at some points $p_i$

2. Define cells $c_i = (p_i, p_{i+1})$

3. Consider the reference basis functions for a reference cell $(0,1)$

   $\phi_{0,1} : [0,1] \rightarrow [0,1], \; \phi_0(r) = 1-r, \; \phi_1(r) = r$

4. Define a linear transformation $T_i$ from the reference to actual cell $c_i$

5. For $c_i$, define the actual basis functions $\Phi_{0,1}$ using $\phi_{0,1}$ and $T_i^{-1}$

   and rewrite the final interpolation

   $\tilde{f}(x, y) = \sum_{i=1}^{n} f_i \Phi_i^1(T^{-1}(x, y))$

   • Apply (5) to interpolate all points in $c_i$ using only samples at vertices $p_i, p_{i+1}$ of $c_i$
   • Repeat from 4 for next cell $c_{i+1}$

   This is covered in the CG course!

Note: see Sec. 3.4 for expressions for all $T^{-1}$
Hermite Splines

- A cubic Hermite curve is defined by four constraints, the two endpoints $p_1$, $p_2$ and the tangents at these points $t_1$ and $t_2$. 
How to Derive the Equations

- We can use the following equations:

\[
P(u) = au^3 + bu^2 + cu + d
\]
\[
P'(u) = 3au^2 + 2bu + c
\]

- Let \( u = 0 \) and \( u = 1 \) and solve:

\[
\begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 \\
3 & 2 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix}
= 
\begin{bmatrix}
p_1 \\
p_2 \\
t_1 \\
t_2
\end{bmatrix}
\]

Compute the inverse of the matrix to get the coefficients!
Basis and Geometry

- Basis matrix and Geometry matrix

\[
\begin{bmatrix}
a \\ b \\ c \\ d \\
\end{bmatrix} =
\begin{bmatrix}
2 & -2 & 1 & 1 \\
-3 & 3 & -2 & -1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
p_1 \\ p_2 \\ t_1 \\ t_2 \\
\end{bmatrix}
\]
Solution

- Insert vector with different degrees of $u$

$$\mathbf{P}(u) = \begin{bmatrix} u^3, u^2, u, 1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} u^3, u^2, u, 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ t_1 \\ t_2 \end{bmatrix}$$

$$\mathbf{P}(u) = u \mathbf{MG}$$
Blending functions

- Blends the geometry together

\[ P(u) = \begin{bmatrix} u^3, u^2, u, 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ t_1 \\ t_2 \end{bmatrix} \]
Blending Functions

- The polynomials functions that blends the geometry
  - i.e. blends the control points
- Do you remember linear Interpolation?

\[ p(u) = p_0(1-u) + p_1(u) \]
Blending Functions
Hermite Basis Functions

\[ H_0(t) = 2t^3 - 3t^2 + 1 \]
\[ H_1(t) = -2t^3 + 3t^2 \]
\[ H_2(t) = t^3 - 2t^2 + t \]
\[ H_3(t) = t^3 - t^2 \]

\[ M_x = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \]

\( t \) instead of \( u \) and transposed!
Cells: 1D example (cont’d)

Remarks
- interpolation & reconstruction goes cell-by-cell
- only need sample points at a cell vertices to interpolate over that cell
- reconstruction is piecewise $C^1$ because $\phi_i$ are $C^1$
2D cells: Quads

Same as in 1D case, but
• we have to decide on different cells; say we take quads
• quads → 4 vertices, 4 basis functions
• particular case: square cells = pixels

Bilinear basis functions

\[
\begin{align*}
\Phi_1^1(r, s) &= (1 - r)(1 - s), \\
\Phi_2^1(r, s) &= r(1 - s), \\
\Phi_3^1(r, s) &= rs, \\
\Phi_4^1(r, s) &= (1 - r)s;
\end{align*}
\]

Remember that these functions “support” the points! I.e. when equal to 1 we have the point \( V_x \)

\[
\sum_{i=1}^{N} f_i \Phi_i(p_j) = f_j, \forall j.
\]
2D cells: Quads

Bilinear interpolation

- 4 functions, one per vertex
- result: $C^0$ but never $C^1$ (why?)
- good for vertex-based samples

Constant interpolation

- 1 functions per whole cell
- result: not even $C^0$
- good for cell-based samples

\[ \Phi_1(r, s) = (1 - r)(1 - s), \]
\[ \Phi_2(r, s) = r(1 - s), \]
\[ \Phi_3(r, s) = rs, \]
\[ \Phi_4(r, s) = (1 - r)s; \]

\[ \phi_i^0(x) = \begin{cases} 1, & x \in c_i, \\ 0, & x \notin c_i. \end{cases} \]
Intermezzo

What is the difference between flat and Gouraud (smooth) shading?

- Flat shading:
  - surface: bilinear interpolation
  - colors: constant interpolation

- Gouraud shading:
  - surface: bilinear interpolation
  - colors: bilinear interpolation

Note: do not confuse Gouraud shading (color interpolation) with the Phong lighting model (color computation from normals)
2D cells: Quads

Images (color or grayscale)
• use constant basis functions
• cells = pixels
• data (color) is defined at the center of pixels, not corners
• we’ll see why this is important in Module 3
2D cells: Triangles

Remarks
• triangles and quads offers largely same pro’s and con’s
• quad basis functions are not planes (they are bilinear)
• in graphics/visualization, triangles used more often than quads
  • easier to cover complex shapes with triangles than quads
  • same computational complexity

\[ \Phi_1^1(r, s) = 1 - r - s, \]
\[ \Phi_2^1(r, s) = r, \]
\[ \Phi_3^1(r, s) = s. \]
3D cells: Tetrahedra

Remarks

- counterparts of triangles in 3D
- interpolate volumetric functions \( f : \mathbb{R}^3 \rightarrow \mathbb{R} \)
- three parametric coordinates \( r, s, t \)
- trilinear interpolation

\[
\begin{align*}
\Phi_1^{\text{tet}}(r,s,t) &= 1 - r - s - t, \\
\Phi_2^{\text{tet}}(r,s,t) &= r, \\
\Phi_3^{\text{tet}}(r,s,t) &= s, \\
\Phi_4^{\text{tet}}(r,s,t) &= t.
\end{align*}
\]
Remarks
• counterparts of quads in 3D
• interpolate volumetric functions $f : \mathbb{R}^3 \rightarrow \mathbb{R}$
• trilinear interpolation
• particular case: cubic cells or voxels (studied later in Module 7)

\[ \Phi_1(r, s, t) = (1 - r)(1 - s)(1 - t), \]
\[ \Phi_2(r, s, t) = r(1 - s)(1 - t), \]
\[ \Phi_3(r, s, t) = rs(1 - t), \]
\[ \Phi_4(r, s, t) = (1 - r)s(1 - t), \]
\[ \Phi_5(r, s, t) = (1 - r)(1 - s)t, \]
\[ \Phi_6(r, s, t) = r(1 - s)t, \]
\[ \Phi_7(r, s, t) = rst, \]
\[ \Phi_8(r, s, t) = (1 - r)st. \]
Cell types for constant/linear basis functions

0D
- point

1D
- line

2D
- triangle, quad, rectangle

3D
- tetrahedron, parallelepiped, box, pyramid, prism, …
Quadratic cells

- allow defining quadratic basis functions
- higher precision for interpolation
- however, we need data samples at extra midpoints, not just vertices
- used in more complex numerical simulations (e.g. finite elements)
- split into linear cells for visualization purposes

Figure 3.6. Converting quadratic cells to linear cells.
Non Linear Interpolation

- Splines are often used to interpolate data
- They are polynomials
  - often second or third degree Polynomials
- Two Points: Linear Interpolation
- Three points: Quadratic Interpolation
- Four points: Cubic Interpolation and so forth
- Larger degree does not necessarily give better interpolation!
Quadratic interpolation

- \( p(u) = au^2 + bu + c \)
- Solve the system of equations to obtain the coefficients

\[
\begin{bmatrix}
0 & 0 & 1 \\
1/4 & 1/2 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
a \\ b \\ c \\
\end{bmatrix}
= 
\begin{bmatrix}
p_0 \\
p_1 \\
p_2 \\
\end{bmatrix}
\]

- This curve is defined between \( p_0 \) and \( p_1 \) for \( u=[0..1] \)
Quadratic Interpolation

- Can be used on triangles
  - Must have 6 points of data
- Quads
  - needs 8 points of data
From cells to grids

Cells
• provide interpolation over a small, simple-shaped spatial region

Grids
• partition our complex data domain D into cells
• allow applying per-cell interpolation (as described so far)

Given a domain D…

A grid \( G = \{c_i\} \) is a set of cells such that

\[
c_i \cap c_j = \emptyset, \forall i \neq j
\]

no two cells overlap (why? Think about interpolation)

\[
\bigcup_i c_i = D
\]

the cells cover all our domain (why? Think about our end goal)

The dimension of the domain \( D \) constrains which cell types we can use: see next
Uniform grids

- all cells have identical size and type (typically, square or cubic)
- cannot model non-axis-aligned domains

- Efficient Storage!
Rectilinear grids

- all cells have same type
- cells can have different dimensions but share them along axes
- cannot model non-axis-aligned domains

Figure 3.8. Rectilinear grids. 2D rectangular domain (left) and 3D box domain (right).
Structured grids

- All cells have the same type.
- Cell vertex coordinates are freely (explicitly) specifiable…
- …as long as cells assemble in a matrix-like structure.
- Can approximate more complex shapes than rectilinear/uniform grids.

Figure 3.9. Structured grids. Circular domain (left), curved surface (middle), and 3D volume (right). Structured grid edges and corners are drawn in red and green, respectively.
Unstructured grids

Consider the domain $D$: a square with a hole in the middle

We cannot cover such a domain with a structured grid (why?)
• it’s not of genus 0, so cannot be covered with a matrix-like distribution of cells

For this, we need unstructured grids (see next)
Unstructured grids

- different cell types can be mixed (though it’s not usual)
- both vertex coordinates and cell themselves are freely (explicitly) specifiable
- implementation
  - vertex set \( V = \{ v_i \} \)
  - cell set \( C = \{ c_i = (\text{indices of vertices in } V) \} \)
- most flexible, but most complex/expensive grid type
Topology vs. Geometry

- Each type has its topology
  - Example:
    - A triangle has three vertices but a line has only two, etc

- Geometry
  - Can differ within the same type
Data Representation

- Cells
  - Linear
  - Non linear
- Topology vs. Geometry
- Attribute Data
  - Scalar
  - Vectors
  - Normals
  - Texture Coordinates
  - Tensors
Recapitulation: Dataset

- Points: $f: \mathbb{R}^2 \rightarrow \mathbb{R}^n$
- Cells: $f: \mathbb{R}^3 \rightarrow \mathbb{R}^n$

Structure:
- Scalars: $f: \mathbb{R}^m \rightarrow \mathbb{R}$
- Vectors: $f: \mathbb{R}^m \rightarrow \mathbb{R}^3$
- Tensors: $f: \mathbb{R}^m \rightarrow \mathbb{R}^{6..9}$
- $m$-dimensional $n$-variate

Attributes:
- Operations: $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$
- Reconstruction:
  - piecewise constant
  - piecewise linear

We discussed about these (grids, interpolation, reconstruction)
We discuss next about attributes
Data attributes

\[ f : \mathbb{R}^m \rightarrow \mathbb{R}^n \]

- \( n=0 \) no attributes (we model a shape only e.g. a surface)
- \( n=1 \) scalars (e.g. temperature, pressure, curvature, density)
- \( n=2 \) 2D vectors
- \( n=3 \) 3D vectors (e.g. velocity, gradients, normals, colors)
- \( n=6 \) symmetric tensors (e.g. diffusion, stress/strain – Modules 5..6)
- \( n=9 \) assymmetric general tensors (not very common)

Remarks

- an attribute is usually specified for all sample points in a dataset (why?)
- different measurements will generate different attributes
- each attribute is interpolated separately
- different visualization methods for each \( n \) (see Module 3 next)
Summary

Data Representation (book Chapter 2)

• reconstruct continuous representations of sampled signals
  • efficiently
  • accurately

• interpolation, grids, and cells

• data attributes (scalars, vectors, tensors)

• advanced issues (resampling, grid-less interpolation)

• read Ch. 2 in detail to understand all the math!

Next module

• visualization algorithms

Happy so far?
Scientific Visualization Module 3
Scalar Algorithms

prof. dr. Alexandru (Alex) Telea

Department of Mathematics and Computer Science
University of Groningen, the Netherlands
The Visualization Pipeline - Recall

Dataset → Process → Dataset → Process → Dataset → Process → Dataset → Process → Dataset

data formatting → data filtering → data mapping → 3D to 2D rendering

raw data

\[ f(x,y) \rightarrow \mathbb{R}^3 \]
\[ \{0,2,-5,...\} \]

imported dataset

filter

enriched dataset

map abstract data to visual representations

2D/3D shape

draw visual representations

final image

end user

measuring device or simulation

insight into the original phenomenon
Algorithm classification

1. Scalar algorithms
   • operate on scalar data
   • color mapping, contouring, height plots

2. Vector algorithms
   • operate on vector data
   • hedgehogs, glyphs, derived quantities, stream surfaces, image-based methods

3. Tensor algorithms
   • operate on symmetric 3x3 tensors
   • tensor glyphs, hyperstreamlines, fiber tracing, principal component analysis

4. Modeling algorithms
   • change attributes and/or underlying grid
   • implicit functions, distance fields, cutting, selection, grid-less interpolation, grid processing
Color mapping

Basic idea
• Map each scalar value \( f \in \mathbb{R} \) at a point to a color via a function \( c : [0,1] \rightarrow [0,1]^3 \)

Color tables
• precompute (sample) \( c \) and save results into a table \( \{c_i\}_{i=1..N} \)
• index table by normalized scalar values

\[
\begin{align*}
\text{input data} & \quad \text{scalar value } f \\
\text{determine input range} & \quad \text{scalar value range } [f_{\text{min}}, f_{\text{max}}] \\
\text{normalize input to } [0,N] & \quad i = N \frac{f - f_{\text{min}}}{f_{\text{max}} - f_{\text{min}}} \\
\text{Color mapping} & \quad c_i = c \left( \frac{i}{N} \right) \\
\text{desired color transfer function} & \quad c : [0,1] \rightarrow [0,1]^3 \\
\end{align*}
\]
Colormap design

What makes a good colormap?

• map scalar values to colors intuitively…
• …so we can visually invert the mapping to tell scalar values from colors

Recall example in Module 1

Data values mapped to RGB colors via a colormap

Invert mapping:
1. Look at some point \((x,y)\) in the image → color \(c\)
2. Locate \(c\) in colormap at some position \(p\)
   • use the colormap legend to derive data value \(s\) from \(p\)

\(\text{blue}=0\) \(\rightarrow\) \(\text{red}=100\)

\(s = 90\)
Rainbow colormap

- probably the most (in)famous in data visualization
- intuitive ‘heat map’ meaning
  - cold colors = low values
  - warm colors = high values

```c
void c(float f, float& R, float& G, float& B) {
    const float dx = 0.8;
    f = (f<0)? 0 : (f>1)? 1 : f; //clamp f in [0,1]
    g = (6-2*dx)*f + dx; //scale f to [dx, 6-dx]
    R = max(0,(3-fabs(g-4)-fabs(g-5))/2);
    G = max(0,(4-fabs(g-2)-fabs(g-4))/2);
    B = max(0,(3-fabs(g-1)-fabs(g-2))/2);
}
```

Simple to implement (see Sec. 5.2)
Gray-value colormap

- brightness = value
- natural in some domains (X-ray, angiography)

2D slice in 3D CT dataset
Scalar value: tissue density

Gray-value colormap
- white = hard tissues (bone)
- gray = soft tissues (flesh)
- black = air

Rainbow colormap
- red = hard tissues (bone)
- blue = air
- other colors = soft tissues
Colormap comparison

2D slice in 3D hydrogen atom potential field

Heat colormap
• maxima highlighted well
• lower values better
• separable than with gray-value colormap

Rainbow colormap
• maxima not prominent
• lower values better
• separable

Gray-value colormap
• maxima are highlighted well
• lower values are unclear

Which is the better colormap? Depends on the application context!
Colormap comparison

2D slice in 3D pressure field in an engine

A. Gray-value colormap
• maxima highlighted well
• low-contrast

B. Purple-to-green colormap
• maxima highlighted well
• good high-low separation

C. Red-to-green colormap
• luminance not used
• color-blind problems

D. ‘Random’
• equal-value zones visible
• little use for the rest

Which is the better colormap? Depends on the application context!
Colormap design techniques

We cannot give universal design rules
• but some technical guidelines/tricks still exist

1. Fully use the perceptual spectrum
• colormap entries should differ in more, rather than less, HSV components

   scalar value ~ V; H,S not used

   scalar value ~ H; S,V not used

   scalar value ~ H,V; S not used

2. Colormap should be easily invertible
• avoid colormap entries with
  • similar HSV entries
  • which are perceived as similar (see color blindness issues)
  • which are hard to perceive (e.g. dark or strongly desaturated colors)

Good design guidelines: www.colorbrewer.org
Colormap design techniques

3. Design based on what you need to emphasize
• specific value ranges
• specific values
• value change rate (1st derivative of scalar data)
• ...

2D function \( f(x, y) = e^{-10(x^4+y^4)} \)

**Gray-scale colormap**
• highlights plateaus
• value transitions hard to see

**Zebra colormap**
• highlights value variations (1st derivative)
• dense, thin bands: fast variation
• thick bands: slow variation
Colormap implementation details

Where to apply the colormap?
• per grid-cell vertex

Note:
Compare to Gouraud Shading

2D periodic high-frequency function

64x64 points  32x32 points  16x16 points

As we decrease the sampling frequency, strong colormapping artifacts appear
Why is this so?
Colormap implementation details

Where to apply the colormap?
• per pixel drawn – better results than per-vertex colormapping
• done using 1D textures

2D periodic high-frequency function

64x64 points  32x32 points  16x16 points

Explanation
• per-vertex: \( f \rightarrow c(f) \rightarrow \text{interpolation}(c(f)) \)  color interpolation can fall outside colormap!
• per-pixel: \( f \rightarrow \text{interpolation}(f) \rightarrow c(\text{interpolation}(f)) \)  colors always stay in colormap

See Sec. 5.2 for details

Note: Compare to Phong Shading
Color banding

How many distinct colors $N$ to use in a color table?

• more colors: better sampled $c$ thus smoother results
• fewer colors: color banding appears

Question

• can we see sharp color banding with per-vertex colormapping? Why (not)?
Colour Mapping

- Maps scalar data to colour
- Can be done by a colour look up table
Height / displacement plots

Displace a given surface \( S \subseteq D \) in the direction of its normal. Displacement value encodes the scalar data \( f \).

\[
S_{\text{displ}}(x) = x + n(x) f(x), \quad \forall x \in S
\]

Height plot:
- \( S = xy \) plane
- displacement always along \( z \)

Displacement plot:
- \( S = \) any surface in \( \mathbb{R}^3 \)
- useful to visualize 3D scalar fields
Conclusion

- Data interpolation
  - Fills in “missing data”
- Simple visualisation of scalar data
  - Colour Mapping
- Next time scalar continued and vector visualisations