Scientific Visualisation Summary FS 2013

Pascal Spörri
pascal@spoerri.io

July 31, 2013

This summary is based on the course slides of the Scientific Visualisation course at ETH Zürich[1] from spring semester 2013.

[1] http://www.scivis.ethz.ch/education/scivis_course/notes

Contents

1 Introduction

SciVis is interdisciplinary the fields of application include engineering, natural sciences and medical sciences. There's a common application to all fields: There are *numerical datasets* providing an abstraction from the particular application. The characteristics of such datasets include:

Dimension of domain: Number of coordinates or parameters

Dimension of values: Scalar, vector or tensor fields

Type of data: Discrete values versus discretised data

Type of discretisation: (Un-)structured grid, scattered data

Time dependencies: Static versus time-dependent.

1.0.1 SciVis and InfoVis

Scientific Visualisation is mostly concerned with

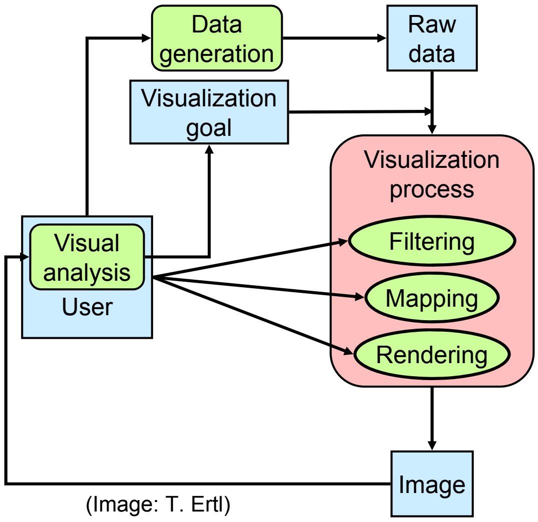
- 2,3,4 dimension spatial or spatio-temporal data
- discretised data

Information Visualisation focuses on:

- High-dimensional, abstract data
- Discrete data
- Financial, statistical, etc.
- Visualisation of large trees, networks, graphs
- Data mining:
  - Finding patterns
  - Clusters
  - Voids
  - Outliers

1.1 Visualisation Scenarios

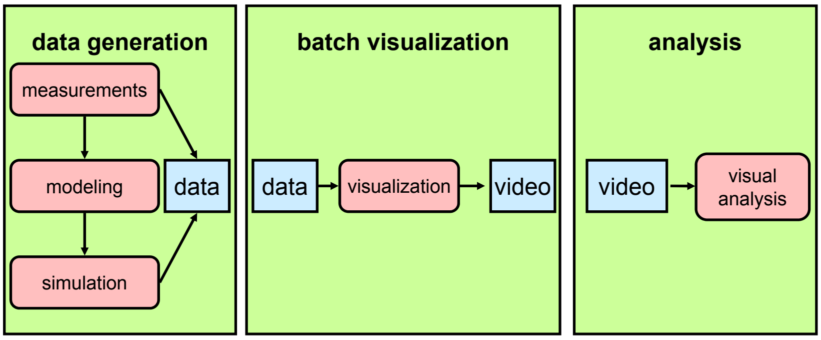
The reference model for visualisation:



(Image: T. Ertl)

1.1.1 Video/Movie

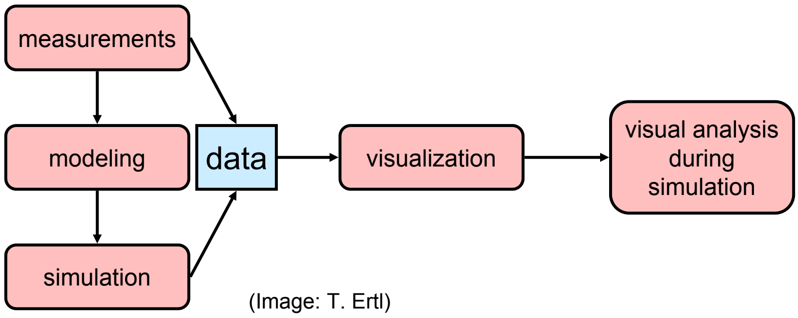
In a first step the data is generated. Then the data is visualised during a batch visualisation step and in the end the video is analized.



(Image: T. Ertl)

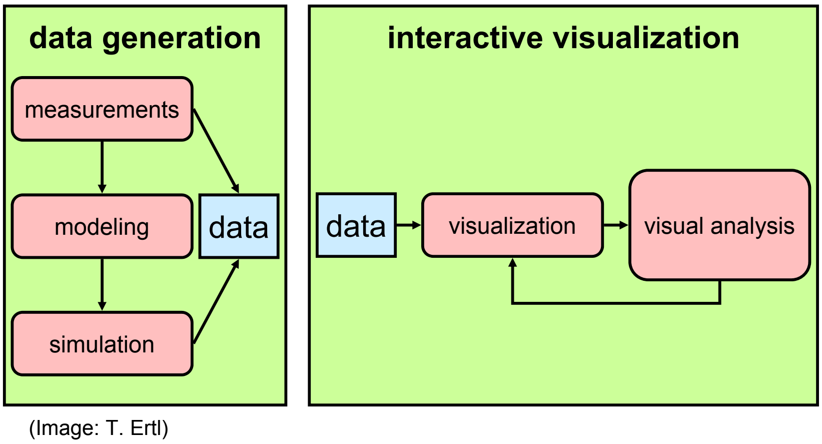
1.1.2 Tracking

The gathered data is directly visualised and analysed.



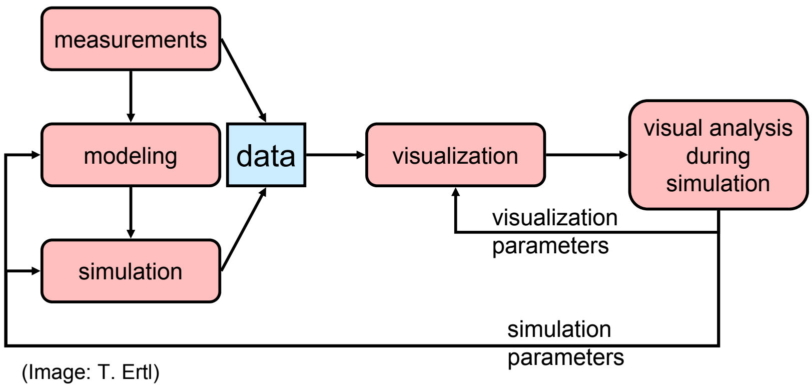
1.1.3 Interactive Post Processing/Visualisation

The data generation step is split from the visualisation step.



1.1.4 Interactive Steering/Computational Steering

The visualisation has a direct impact on the simulation and the visualisation.



1.2 Data Discretisations

Types of data sources have typical types of discretisations:

Measurment Data Typically scattered (”mesh-less”, no grid)

Numerical Simulation Data

- Structured, block-structured or unstructured grids
- Adaptively refined meshes
- Multi-Zone grids with relative motion
- ...

Imaging Methods Uniform grids

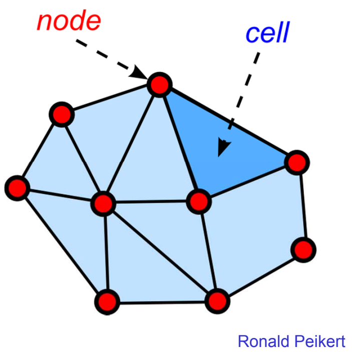
Mathematical Functions (and functionally represented data) can be sampled by demand:

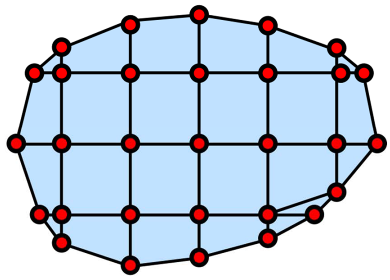
- Uniform
- Adaptive

1.3 Unstructured Grids

1.3.1 2D Unstructured Grids

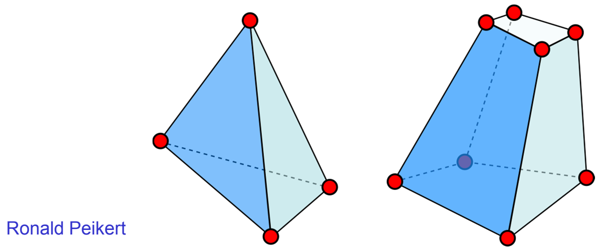
Cells are *triangles* and/por quadrangles. The domain can be a surface embedded in 3-space (distinguish n-dimensional from n-space).



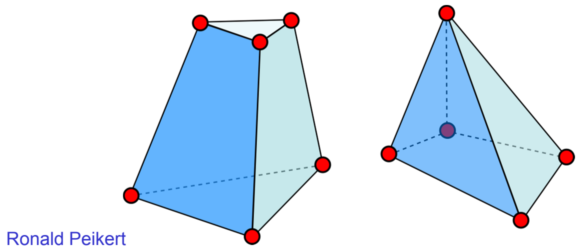


1.3.2 3D Unstructured Grids

Cells are *tetrahedra* or *hexahedra*.



Mixed grids ("zoo meshes") require additional types: *wedge* (3 sided prism), and a *pyramid* (4-sided).



1.4 Structured Grids

Curvilinear Grid (general case) Nodes are given in an array $N_i \times N_j \times N_k$ and the cells are implicit.

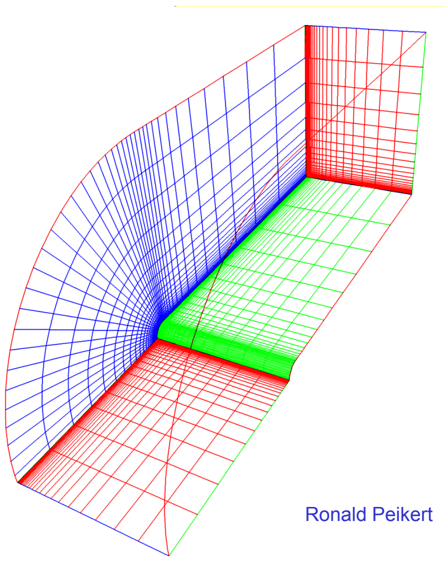


Figure 1: Curvilinear Grid

Rectilinear Grid (special case) The coordinate functions are simpler:

$$x = x(i) \qquad y = y(j) \qquad z = z(k)$$

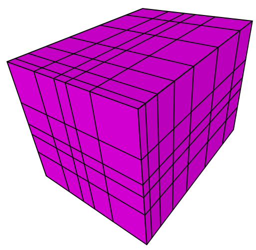


Figure 2: Rectilinear Grid, Source: Wikipedia

Uniform Grid (more special) The coordinates are defined by an *axis-aligned*bounding box.

1.4.1 Point-Sampled Data/Scattered Data

Point sampled data returns only nodes and no cells. Typical data sources are measurement data for example meteorological data.
Options for visualisation include:

Point-Based Methods (relatively few algorithms)

Triangulation for example constrained Delaunay (difficult in 3D)

Resampling onto uniform grid.

1.5 Elementary Visualisation Methods

Scalar Fields can be visualised by plotting its *function graphs*:

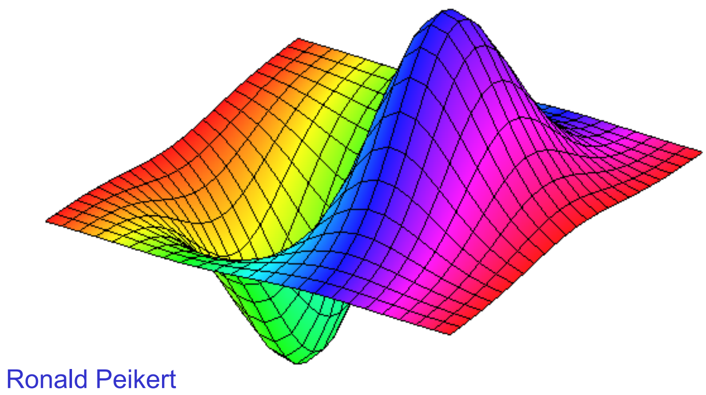
1D Field: The Graph is a curve:

$$y = f(x)$$

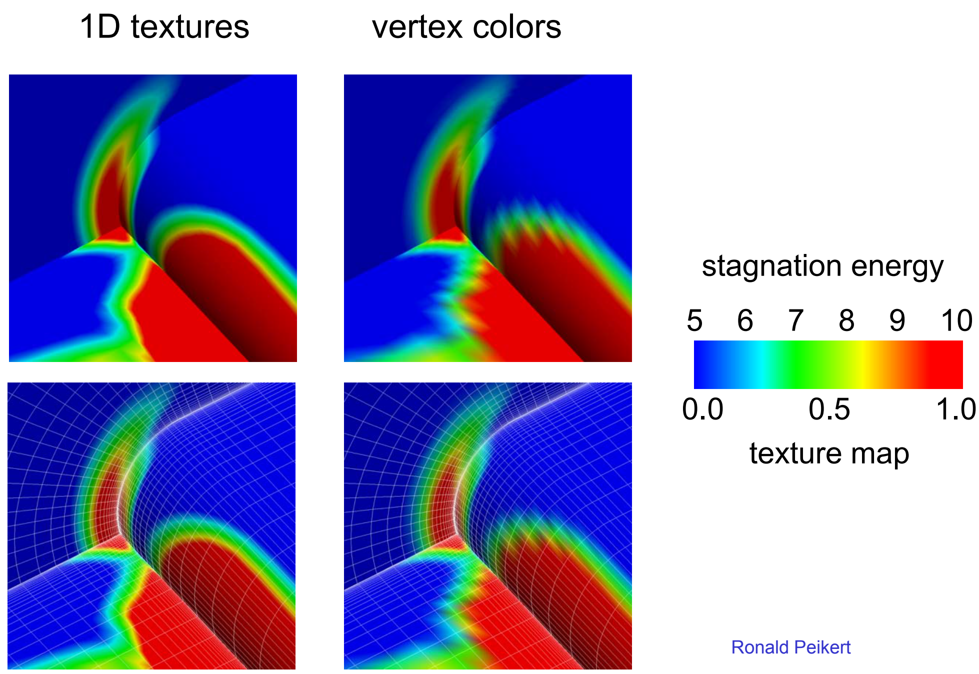
2D Field: The Graph is a *height field*:

$$z = f(x, y)$$

Visualisation is easy for rectilinear grids:
Painter's algorithm (hidden surface removal in software): Draw cells row by row from back to front.



Scalar fields can also be visualised using *color coding* using *1D texture mapping*. Don't use *vertex colors* and Gouraud shading!



- Problem of RGB colouring mode: The Interpolation is in the wrong colour space (RGB vs. colour table).
- Problem of Colour Index mode: Lighting is not possible.

2 Contouring and Isosurfaces

2.1 Contours

Contours are a set of points where the scalar field s has a given value c:

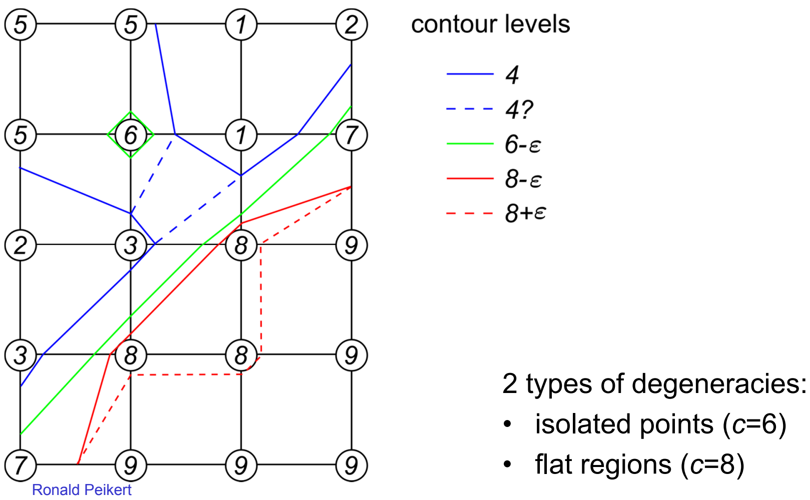
$$\{x \in \mathbb{R}^n : s(x) = c\}$$

Examples in 2D:

- Height contours on maps
- Isobars on weathermaps

Contouring algorithm:

- Find intersection with grid edges
- Connect points in each cell



Topological consistency To avoid degeneracies, use *symbolic perturbations*:

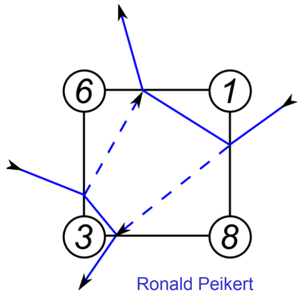
If level c is found as a node value, set the level to $c + \varepsilon$ where ε is an infinitesimal, i.e., $\varepsilon > 0$ and $\varepsilon < x \; \forall x \in \mathbb{R}$.

Then:

- Contours intersect edges at some (possibly infinitesimal) distance from end points.
- Flat regions can be visualised by a pair of contours at $c - \varepsilon$ and $c + \varepsilon$.
- Contours are *topologically consistent*, meaning:

  Contours are *closed*, *orientable*, *nonintersecting lines*.

Ambiguities of contours What is the *correct* contour of $c = 4$?



Two possibilities (from which both are orientable):

- Connecting the high values ———
- Connecting the low values – – – – – –

2.1.1 Contours in a quadrangle cell

Local Coordinates $(0,0)$ $(1,0)$ $(0,1)$ $(1,1)$

Function Values s_{00} s_{10} s_{01} s_{11}

Bilinear Interpolant

$$\begin{aligned} s(x,y) &= (1-x)(1-y)\ s_{00} + x(1-y)\ s_{10} + (1-x)y\ s_{01} + xy\ s_{11} \\ &= Axy + Bx + Cy + D \\ \text{with} \quad A &= s_{11} - s_{01} - s_{10} - s_{00} \\ B &= s_{10} - s_{00} \\ C &= s_{01} - s_{00} \\ D &= s_{00} \end{aligned}$$

- If $A = 0$, then the contour equation is $c = Bx + Cy + D$ and the contours are *straight lines*, all parallel.
- If $A \neq 0$, then the contour equation is

  $$c = A\left(x + \frac{C}{A}\right)(y + \frac{B}{A}) + D - \frac{BC}{A}$$

  and the contours are *hyperbola* except for the level

  $$c = D - \frac{BC}{A}.$$

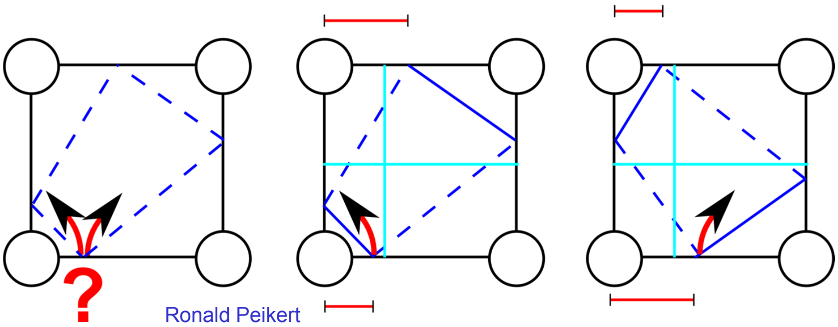
For the special level the contour equation is

$$0 = A\left(x + \frac{C}{A}\right)\left(y + \frac{B}{A}\right),$$

and the contour is a pair of axis-aligned straight lines with

$$x = -\frac{C}{A},$$
$$y = -\frac{B}{A}.$$

Decision can be made without computing special level or saddle points by just comparing fractions of edges:



By using local coordinates, this works also for curvilinear and unstructured grids.
Not that drawing hyperbola instead of straight lines does not lead to better contours: The piecewise bilinear function is not in C^1.

2.1.2 Basic contouring algorithms

Cell-By-Cell algorithms: Simple structure, but generate disconnected segments and require post-processing.

Contour Propagation methods: Complicated, but generate connected contours.

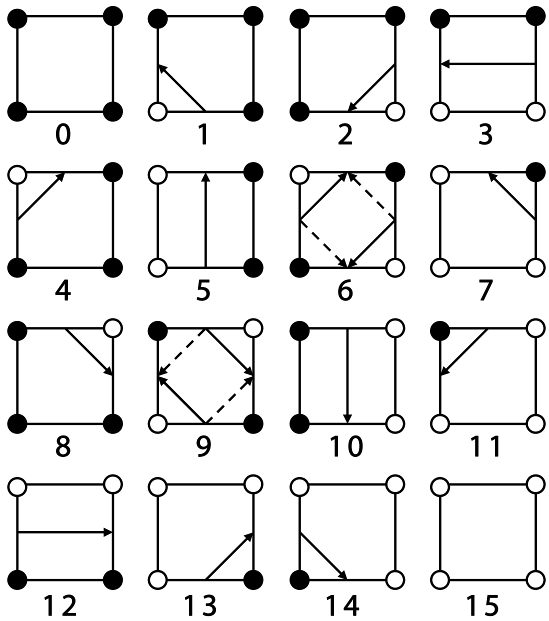
Marching Squares algorithm: Systematic cell-by-cell algorithm. (See below)

2.1.3 Marching Squares

Process nodes in ccw (counter-clockwise order), denoted here as x_0, x_1, x_2 and x_3. At each node x_i compute the reduced field

$$\tilde{s}(x_i) = s(x_i) - (c - \varepsilon) \qquad \text{(which is forced to be nonzero).}$$

Take it's signe as the i^{th} bit of a 4-bit integer. Use this as an index for the lookup table containing the connectivity information.



- • $\tilde{s}(x_i) < 0$
- ∘ $\tilde{s}(x_i) > 0$

Alternating signs exist in cases 6 and 9. Chose the solid or the dashed line based on topological *consitency*.

Contours in triangle/tetrahedral cells Linear interpolation of cells implies piece-wise linear contours. Since contours are unambiguous let us introduce a ”marching triangles” method. This however introduces periodic artefacts.

2.2 The Marching Cubes Algorithm

Contours of 3D scalar fields are known as *isosurfaces*. Before 1987, isosurfaces were computed as contours on planar *slices*, followed by ”contour stitching”.
The *marching cubes* algorithm computes contours *directly in 3D*:

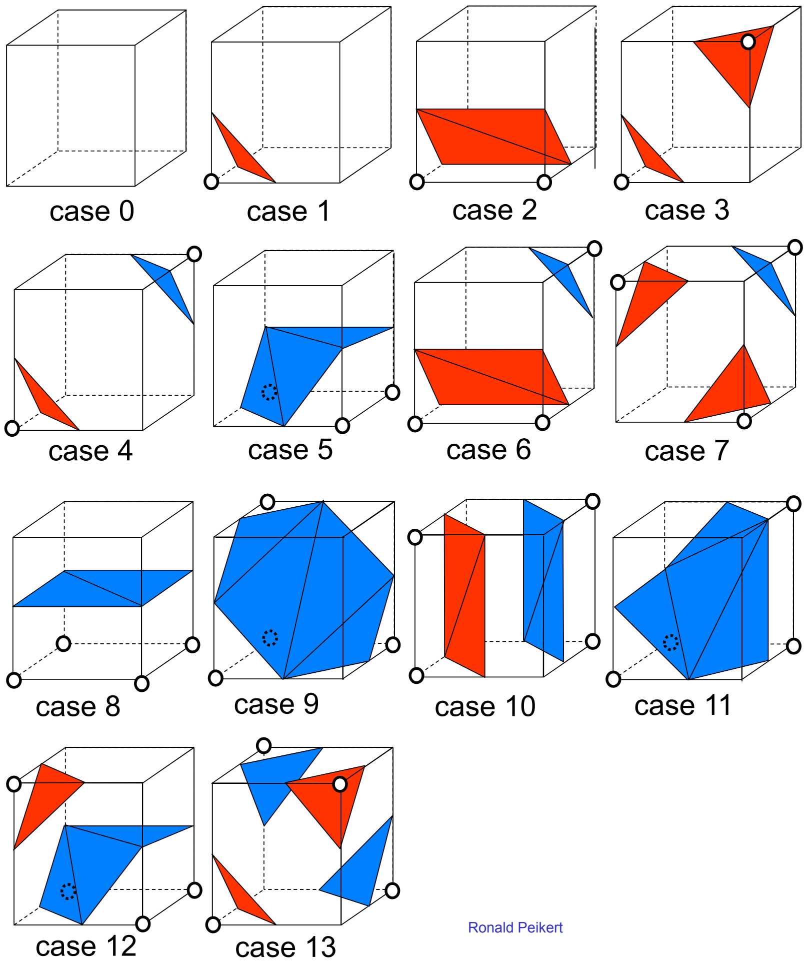
- Pieces of the isosurfaces are generated on a cell-by-cell basis.
- Similar to marching squares, an 8-bit number is computed from the 8 signs of $\tilde{s}(x_i)$ on the corners of a hexahedral cell.
- The isosurface piece is looked up in a table with 256 entries.

How to build up the table of 256 cases?
Lorensen and Cline (1987) exploited 3 types of symmetries:

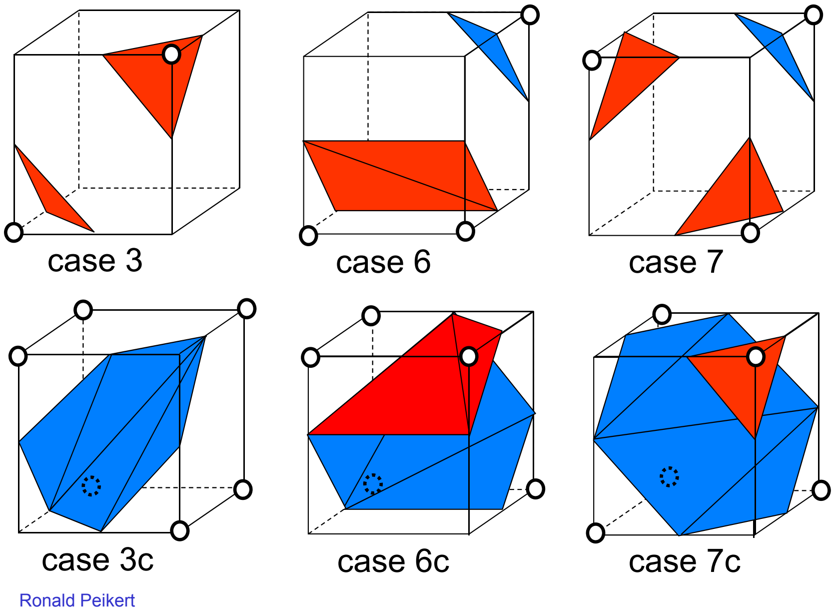
- Rotational symmetries of the cube
- Reflective symmetries of the cube
- Sign changes of $\tilde{s}(x)$.

They published a reduced set of 14 cases:

- White circle indicate positive signs of $\tilde{s}(x)$,
- The positive side of the isosurface is drawn in red, the negative side in blue.



Unfortunately not all pieces fit together (same problem as with marching squares) and additional cases need to be introduced.



The remaining complementary cases are simply obtained by changing the the orientation. Based on these 28 cases, the full 256 cases are obtained by rotations of the cube.

Summary of the algorithm:

1. Pre-processing:
   - Build a table of the 28 cases.
   - Derive a table of the 256 cases, containing info on
     - Intersected cell edges
     - Triangles based on these points
2. Loop over cells:
   - Find sign of $\tilde{s}(x)$ for the 8 corner nodes, giving 8-bit integer.
   - Use as index into lookup table.
   - Find intersection points on edges listed in table, use linear interpolation.
   - Generated triangles according to table.
3. Post-processing steps:
   - Connect triangles (share vertices)
   - Compute normal vectors:
     - By averaging triangle normals (problem: thin triangles!).
     - By estimating the gradient of the field $s(x)$ (better).

2.3 The Asymptotic Decider Algorithm

Motivation for a different isosurface algorithm: Marching cubes can produce "bad" topology.
Asymptotic decider algorithm (Nielson and Hamann 1991):

- Generate topologically *correct* contours (as oriented straight line segments) on the cell surfaces.
- Connect these around the cell, resulting in one or more polygons.
- Triangulate the polygons.

In general, the AD algorithm generates better isosurfaces. However,

- It cannot be easily implemented with a table like Marching Cubes (too many cases).
- It generates polygons with up to 12 sides (Marching Cubes: up to 7).
- The topology is correct with respect to the trilinear interpolant, but the geometry can deviate.
- Some polygons cannot be "cleanly" triangulated.

2.4 The Dividing Cubes Algorithm

An early *point-based* algorithm (Crawford et al. '87): For each cell

- Check whether it is intersected by the isosurface:

$$\min_{i \in \text{cell}} s_i < c < \max_{i \in \text{cell}} s_i$$

- Subdivide the intersected cell into $m \times m \times m$ subcells using trilinear interpolation.
- Draw the centers of all intersected subcells.
- To light the points estimate the gradient and use it as the normal vector.

2.5 Optimised Isosurface Algorithms

Approaches to speeding up isosurface computation:

View Dependent algorithms:

- Occluded triangles are not computed
- GPU-based isosurface computation and redenring

Data Processing for fast computation of *multiple* isosurfaces (multiple levels), for example for interactive exploration of the data.

- Many methods: Octree, Extrema Graph, Span Space.
- Common Goal: Avoid computation in non-intersected cells.

2.5.1 The Span-Space Algorithm

Method by Livnat (1996).

1. Pre-Processing. For each element
   - Compute min and max ,
   - Treat (min, max) as a point in the *span space* (Euclidean plane).
   - Store points in boxes, non-empty boxes are stored in a linked list.
2. Computation of the isosurface level c:

   Find the intersected cells in the *quadrant* $\min < c$, $\max > c$.

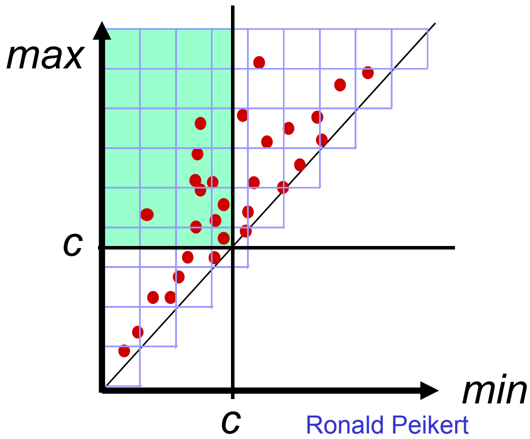


Figure 3: Node distribution in span space.

This algorithm yields a performance gain for datasets with a small local variation, i.e. points in the span space are distributed on the diagonal.

2.6 Selecting Contour Levels

Several types of isosurface statistics can help with level selection.

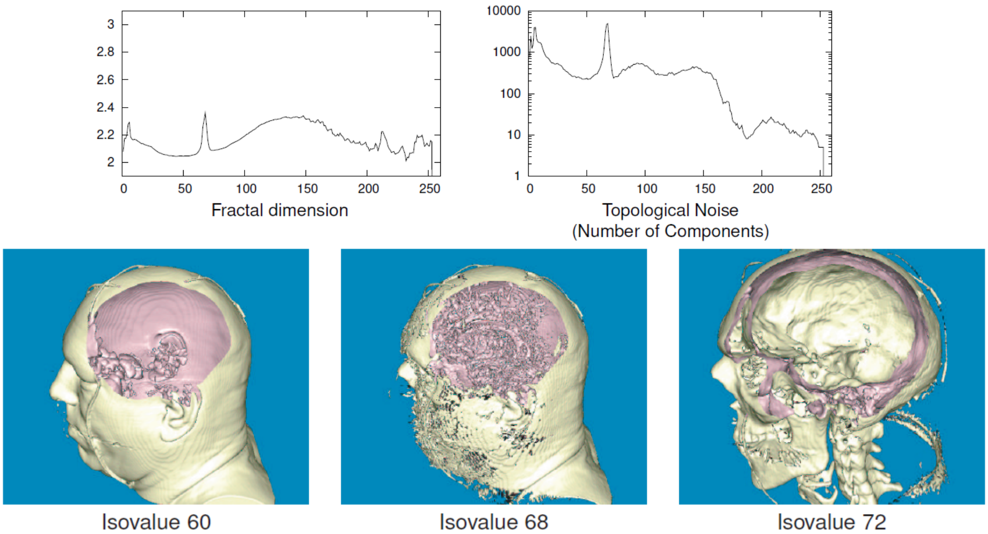


Image credit: M. Khoury

2.7 Limitations of Isosurfaces

Isosurfaces represent only a single level within the data range. In practical data, there is often not a single ”interesting” level.
Transparent rendering of multiple isosurfaces is possible, but

- Limited to a small number of surfaces by visibility
- Alpha-blending require depth sorting.

Alternatives:

- *Feature Extraction methods*, e.g. detecting ”blobs” (maximal ellipse-like contours)
- *Volume rendering* can show ranges of ”interesting levels of the field and/or its gradient.

3 Raycasting

3.1 Direct Volume Rendering

Volume rendering (sometimes called *direct volume rendering*) stands for methods that generate images directly from 3D scalar data. "Directly" means: *no intermediate geometry* (such as an isosurface) is generated.
Volume rendering techniques

- Depend strongly on the grid type.
- Exist for structured and unstructured grids.
- Are predominantly applied to uniform grids with 2D or 3D image data with *cell-centered* data. Cell-centered data
  - are attributed to cells (pixels, voxels) rather than nodes,
  - can also occur in (finite volume) CFD datasets,
  - are converted to node data:
    * By taking the *dual grid* (easy for uniform grids: n cells $\rightarrow n-1$ cells!),
    * or by interpolating.

3.2 Raycasting

Raycasting is historically the first volume rendering technique. It is very similar to *raytracing*:

- *image-space* method: The main loop iterates over the pixels of the output image,
- a *view ray* per pixel (or per subpixel) is traced backward,
- and samples are taken along the ray and *composited* to a single color.

The differences are

- No secondary (reflected, shadow) rays,
- the transmitted ray is not refracted,
- more elaborate compositing functions,
- and samples are taken at intervals (not at object intersections).

3.2.1 Compositing

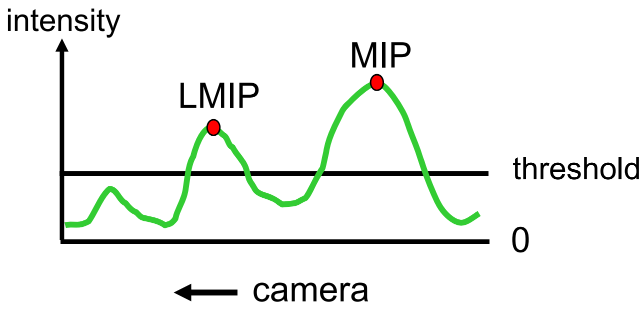
Two simple compositing functions can be used for previewing:

Maximum intensity projection (MIP):

- Maximum of sampled values
- Result resembles X-ray image.

Local maximum Intensity projection (LMIP):

- Choose first local maximum which is above a prescribed threshold.
- Process approximates occlusion.
- It is faster and better than MIP.



3.2.2 α-compositing

Assume that each sample on a view ray has a *color* and *opacity*.

$$(C_0, \alpha_0), \ldots, (C_N, \alpha_N) \qquad C_i \in [0,1]^3, \quad \alpha_i \in [0,1]$$

where the 0^{th} sample is next to the camera and the N^{th} one is a (fully opaque) background sample:

$$\begin{aligned} C_N &= (r,g,b)_{\text{background}} \\ \alpha_N &= 1. \end{aligned}$$

α-compositing can be defined recursively:
Let C_f^b denot the *composite color* of samples f, $f+1$, ..., b. The recursion formula for *back-to-front* compositing yields:

$$\begin{aligned} C_b^b &= \alpha_b C_b \\ C_f^b &= \alpha_f C_f + (1-\alpha_f) C_{f+1}^b. \end{aligned}$$

With *transparency* set to $T_i = 1 - \alpha_i$ we get a *closed formula* for α-compositing:

$$C_f^b = \sum_{i=f}^{b} \alpha_i C_i \prod_{j=f}^{j-1} T_j$$

The *front-to-back* compositing can be derived from the closed formula: Let T_f^b denote the *composite transparency* of samples f, $f+1$, …, b:

$$T_f^b = \prod_{j=f}^{b} T_j.$$

Then the *simultaneous recursion* for front-to-back composition is:

$$\begin{aligned}
C_f^f &= \alpha_f C_f \\
T_f^f &= 1 - \alpha_f \\
C_f^{b+1} &= C_f^b + \alpha_{b+1} C_{b+1} T_f^b \\
T_f^{b+1} &= (1 - \alpha_{b+1}) T_f^b.
\end{aligned}$$

The emission-absorption model How realistic is α compositing? The *emission-absorption* model (Sabella 1988) yields a basic *volume rendering equation*"

$$L(x) = \int_x^{x_b} \varepsilon(x') \exp\left(- \int_x^{x'} \tau(x'') dx'' \right) dx'$$

The equation describes the *radiance* arriving along a ray at the position x on this ray. The *emission* function $\varepsilon(x)$ describes the photons "emitted" by the volume along the ray. The *absorbtion* function $\tau(x)$ is the probability that that photon traveling over a unit distance is lost by absorption.
The emission-absorption model is based on the Boltzmann transport equation in statistical physics, but completely *ignores scattering*. In more general models $\tau(x)$ is an *extinction function* having both an absorption term and a scattering term.
Instead, in the emission-absorption model:

- *Incident scattering* is modelled by the emission function
- *Loss by scattering* can be thought to be part of the absorption.

The discrete version of the emission absorption model:

$$L(x) = \sum_{i=0}^{n} \varepsilon_i \Delta x \exp\left(- \sum_{j=0}^{i-1} \tau_j \Delta x \right) = \sum_{i=0}^{n} \varepsilon_i \Delta x \prod_{j=0}^{i-1} e^{-\tau_j \Delta x},$$

matches the α-compositing formula

$$C_f^b = \sum_{i=f}^{b} \alpha_i C_i \prod_{j=f}^{j-1} (1 - \alpha_i)$$

and gives interpretations of "opacity" and "color":

$$\begin{aligned}
\alpha_i &= 1 - e^{\tau_j \Delta x} \approx \tau_j \Delta x & \text{if } \Delta x << 1 \\
\alpha_i C_i &= \varepsilon_i \Delta x.
\end{aligned}$$

The product $\tilde{C} = \alpha_i C_i$ is called a *premultiplied* or *associated* colour.

3.3 Transfer Functions

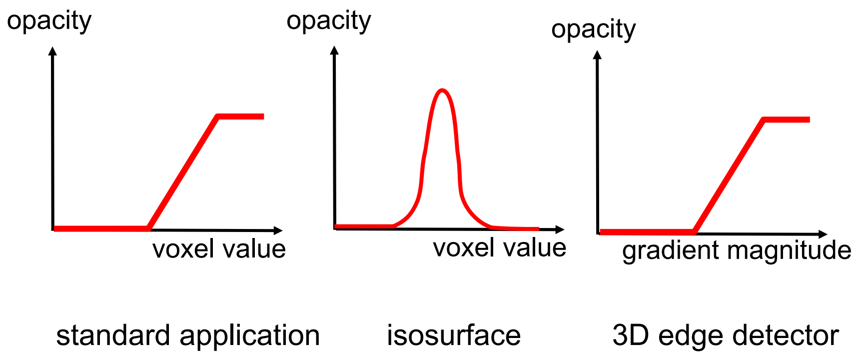
Transfer functions map raw voxel data to opacities and colours as needed for the α-compositing. Inputs of the TF (one or more):

- Voxel value $s(x)$
- Gradient magnitude $\|\nabla s(x)\|$
- Higher derivatives of $s(x)$.

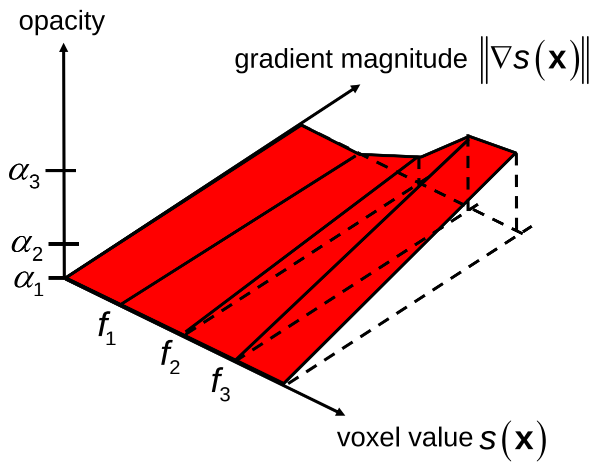
Opacity transfer function $\alpha(s(x), \|\nabla s(x)\|, \ldots)$

Color transfer function $C(s(x), \|\nabla s(x)\|, \ldots)$

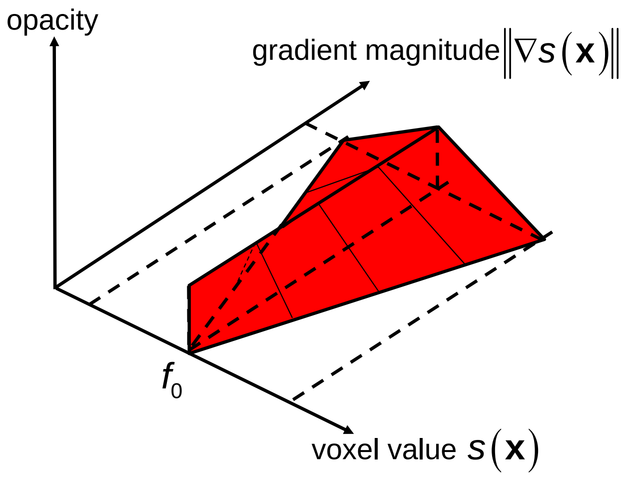
In general TF don't depend on *spatial location*, exception for *focus and context* techniques.
By choosing different opacity transfer functions different types of applications can be achieved.



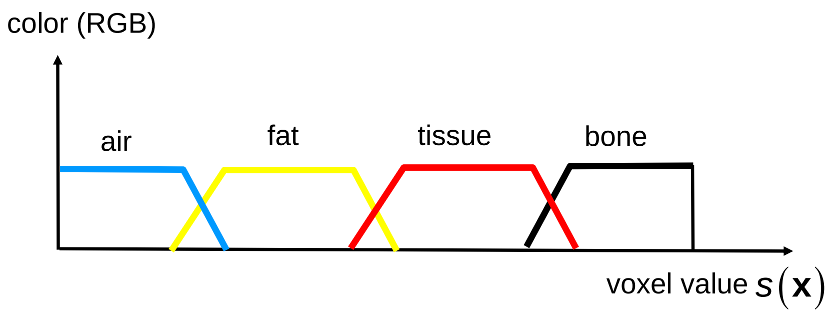
Example of a *bivariate* (=2D) transfer function:



Example of a bivariate transfer function for an isosurface of constant thickness.



The color transfer function allows to make a simple *classification*.



Pre-classification In pre-classification, the voxels can also be lit:

- The gradient is perpendicular to the local isosurface. It can be used as a normal vector for *Phong lighting* (without rendering the isosurface itself).
- *Reflection coefficients* can be assigned by a separate transfer function ("materials" instead of only colors).
- *Diffuse lighting* can be applied to the entire volume dataset as a pre-processing since it's independent of the viewing direction.

3.3.1 Pre- vs. Post-classification

For quality reasons, current volume rendering implementations often use *post-classficiation*.

Pre-Classification

1. Transfer functions are applied to voxels.
2. Results are interpolated to sample locations.

Post-Classification

1. Raw data are interpolated to sample locations.
2. Transfer functions are applied to sampled data.

3.4 Preintegration

Idea (Engel 2001): Simulate *infinitely many* interpolated samples between two successive samples $s_i = s(x_i)$ and $s_{i+1} = s(x_{i+1})$. Assuming that

- Field $s(x)$ varies *linearly* between samples
- and that the transfer functions don't depend on derivatives.

The discrete formula for opacity at a sample was

$$\alpha_i = 1 - e^{-\tau_i \Delta x}.$$

The continuous version for a sample interval $[x_i, x_{i+1}]$ is

$$\alpha_i = 1 - e^{-\int_{x_i}^{x_{i+1}} \tau(s(x))dx}.$$

Assuming now $s(x)$ to be linear between samples, we get:

$$\alpha_i = 1 - \exp\left(-\frac{d}{s_{i+1} - s_i} \int_{s_i}^{s_{i+1}} \tau(s)ds\right) \qquad \text{with } d = \|x_{i+1} - x_i\|,$$

which is called a *preintegrated opacity transfer function*.
The integral

$$\int_{s_i}^{s_{i+1}} \tau(s)ds = \int_0^{s_{i+1}} \tau(s)ds - \int_0^{s_i} \tau(s)ds$$

can be evaluated by two lookups in a precomputed table of

$$\int_0^s \tau(s')ds'.$$

The composite colour of the same interval

$$C_i = \int_{x_i}^{x_{i+1}} \varepsilon(s(x)) \exp\left(-\int_{x_i}^{x} \tau(s(x'))dx'\right) dx,$$

simplifies for linear $s(x)$ to:

$$C_i = \frac{d}{s_{i+1} - s_i} \int_{s_i}^{s_{i+1}} \varepsilon(s) \exp\left(-\frac{d}{s_{i+1} - s_i} \int_{s_i}^{x} \tau(s')ds'\right) ds,$$

which can be precomputed for all combinations of s_i, s_{i+1} and d.

3.4.1 Extinction-based volume rendering

Instead of an opacity Transfer function, use an extinction TF [Schlegel 2011]. Advantage: Extinction is *additive*.

- Riemann sums for numerical integration give better accuracy.
- Additivity can be used for efficient lighting.

Screen-Space ambient occlusion Approximate the fraction of ambient light that is occluded. Method: Compute the total extinction per shell (boxes approximating spheres) b using a *summed area table*.