Computer Graphics

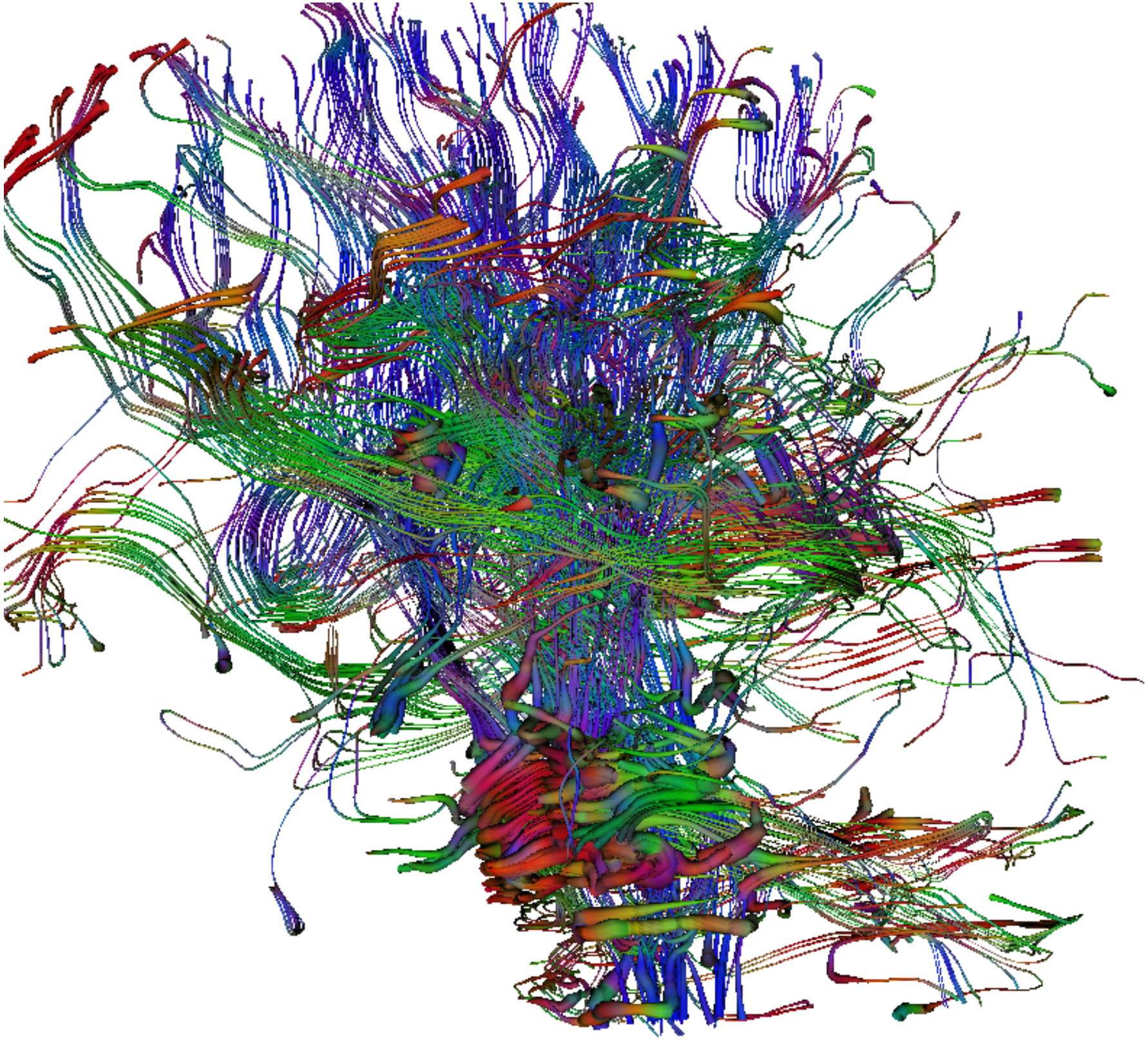
-Scientific Data Visualization-

Oliver Bimber

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

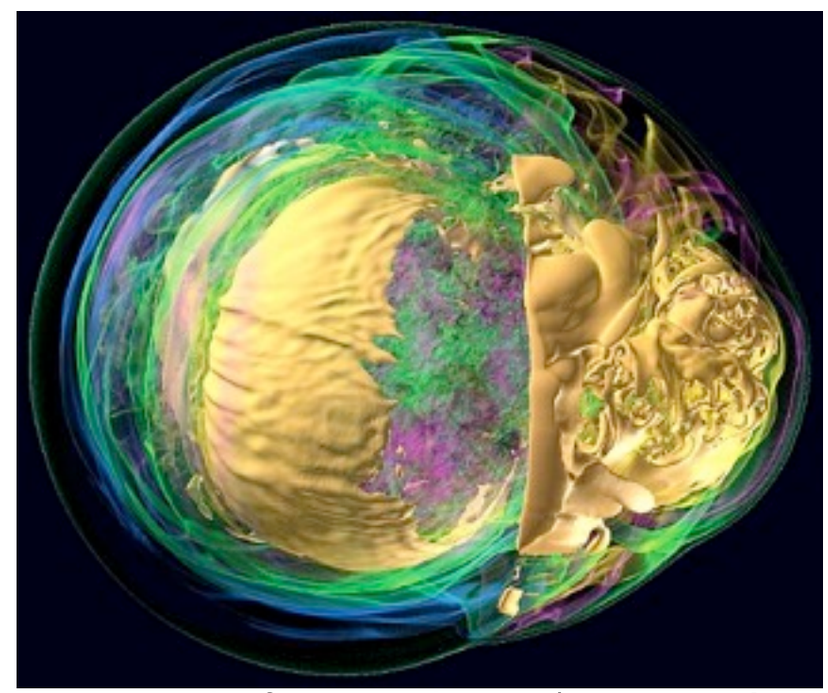
Scientific Data Visualization



Scientific Data Visualization

- So far, we have mainly talked about rendering (i.e., how to graphically display synthetic information, such as modeled scenes)
- Today, we will more look at visualization (i.e., how to graphically display data)
- Volume rendering should better be called volume visualization, since we visualized measured or simulated data
- The goal of data visualization (as it was the case with volumes) is to make it easy to interpret complicated dataset visually
- In general, such data can have different forms, such as scalars, vectors, and tensors
- We will start with the visualization of scalar data
- But we have to consider that we are dealing with sampled data!

example for rendering modeled scenes



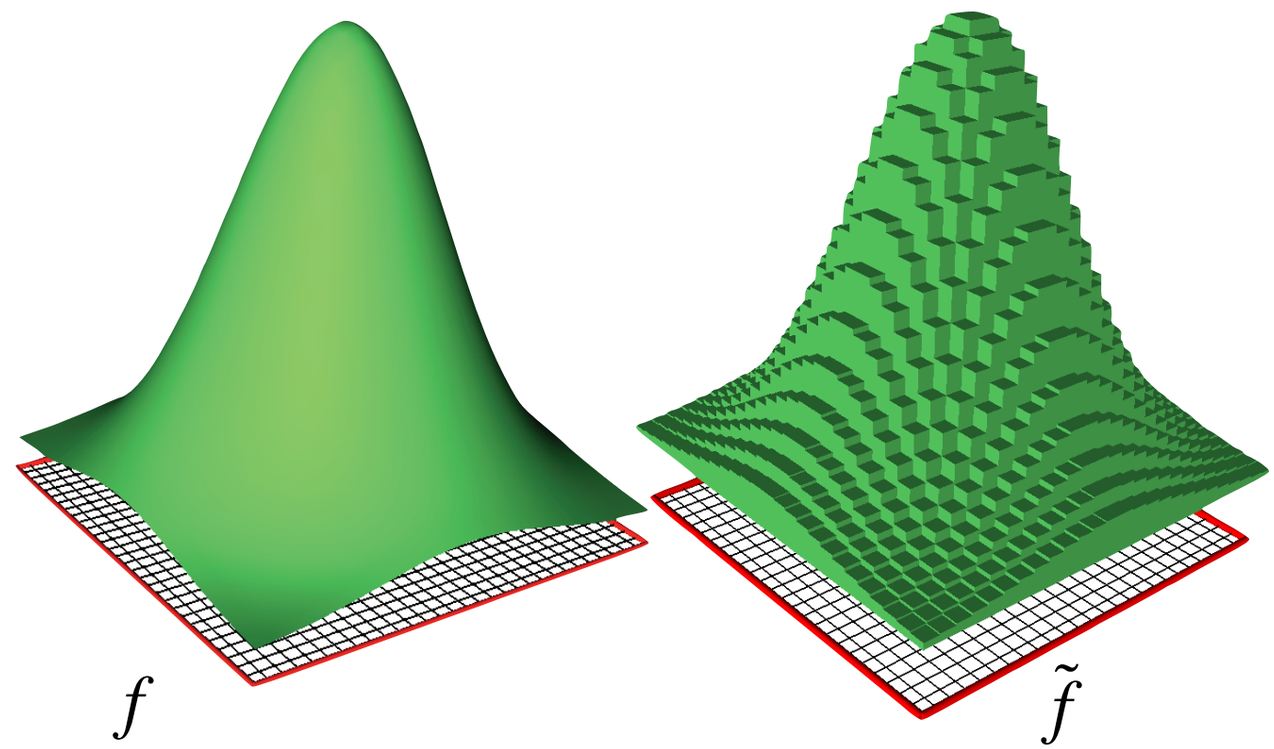
example for visualizing (measured or computed) data

Sampled Data

- Two operations relate sampled and continuous data:
  - sampling - converting continuous data into samples
  - reconstruction - recover (approximate) continuous data from sampled data (e.g., interpolation)
- Interpolation is achieved through a weighted sum of basis functions (interpolation functions)
  - different basis functions exist (most are orthonormal: orthogonal and normal)
  - choice of basis function: computational complexity vs. quality of reconstruction
- Grid (or mesh) is a subdivision into cells (or elements)

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

sampling of continuos function (f) with basis function

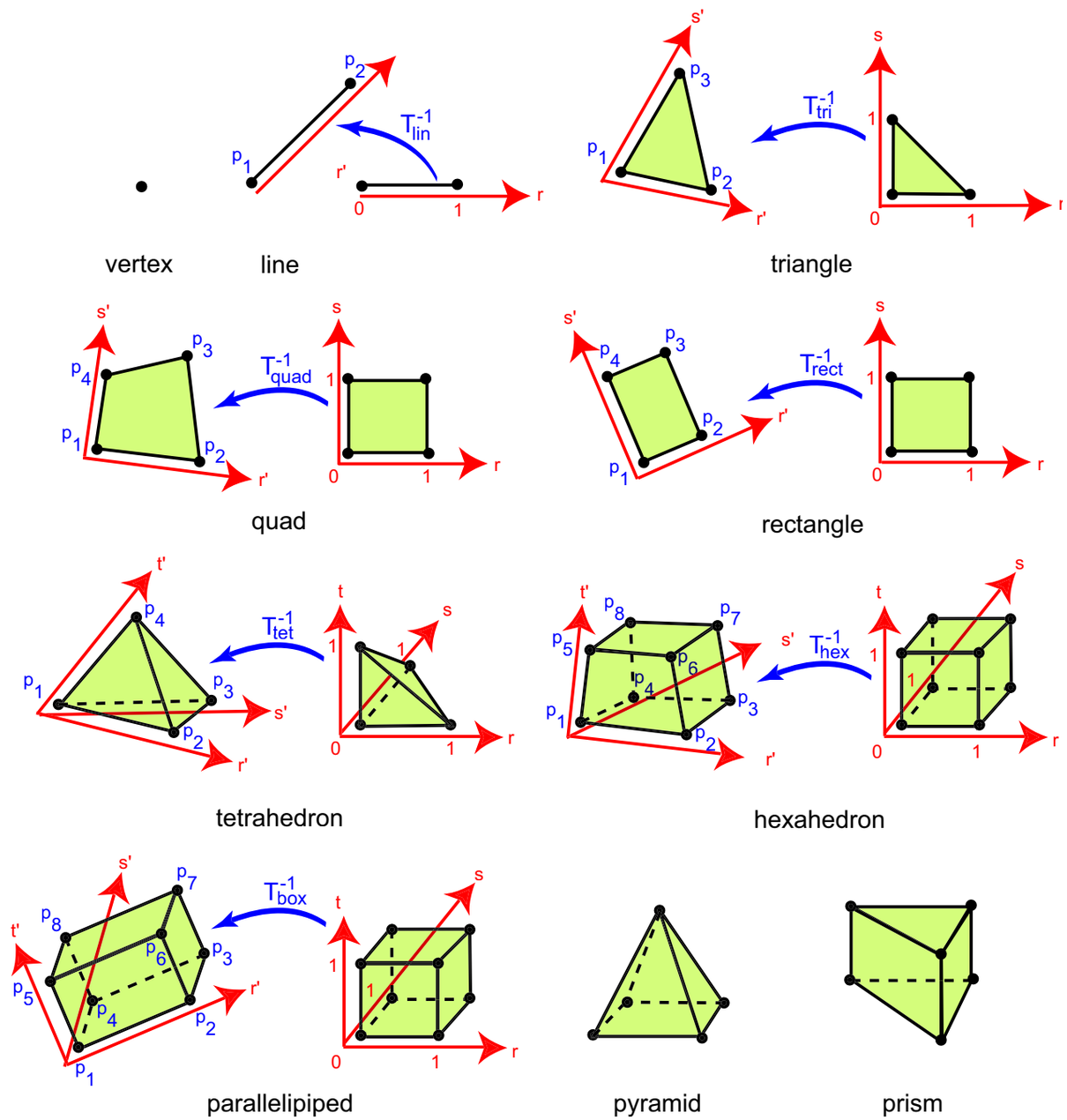


$$\phi_i^0(x) = \begin{cases} 1, x \in c_i \\ 0, x \notin c_i \end{cases}$$

example for a constant orthonormal basis function (piecewise constant interpolation = nearest-neighbour interpolation)

Cell Types

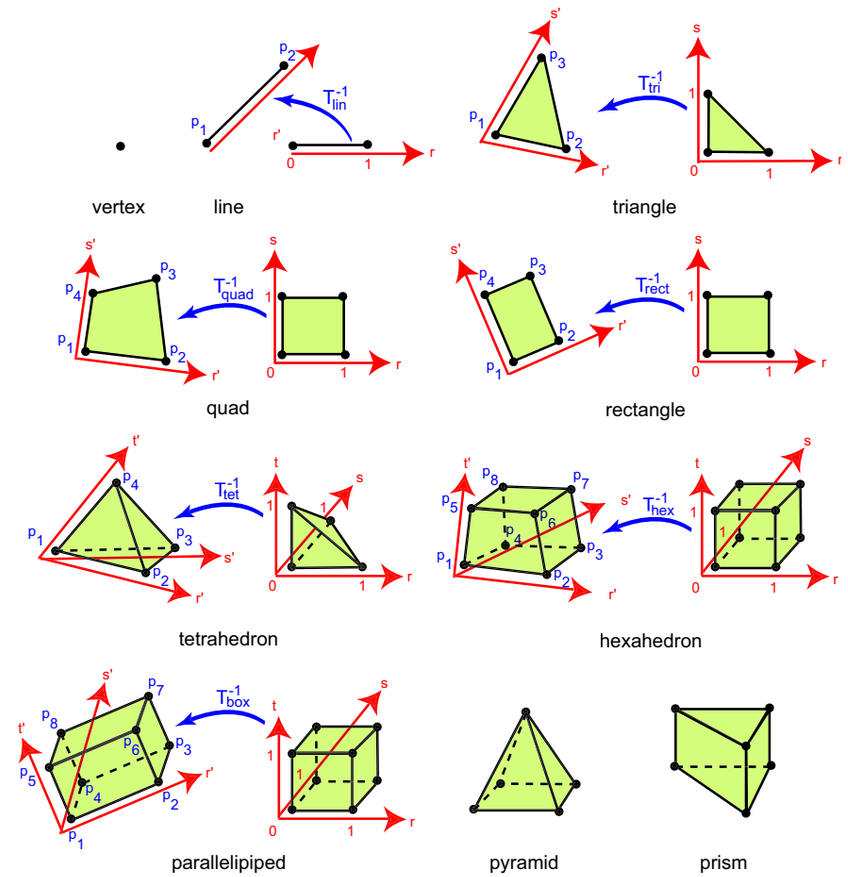
- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
  - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points



cell types in world and reference coordinate systems

Cell Types

- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
  - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points



cell types in world and reference coordinate systems

$$\phi_1^0(r) = 1$$

vertex

$$\phi_1^1(r) = 1 - r \qquad \phi_2^1(r) = r$$

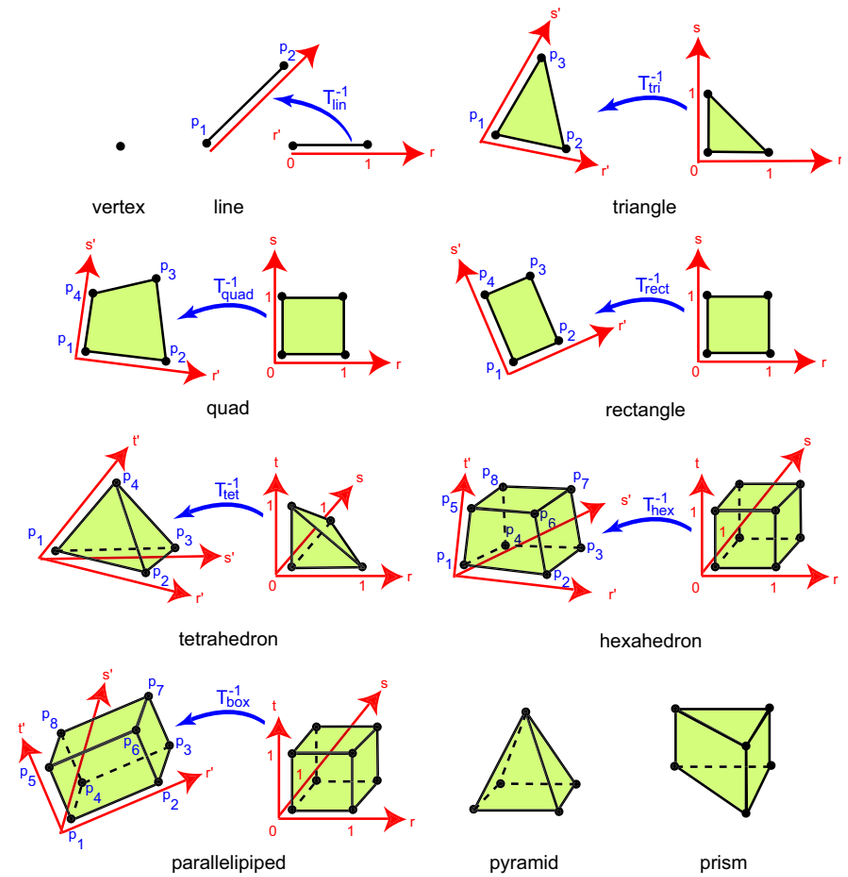
line

$$\phi_1^1(r,s) = 1 - r - s \quad \phi_2^1(r,s) = r \quad \phi_3^1(r,s) = s$$

triangle

Cell Types

- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
  - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points



cell types in world and reference coordinate systems

$$\phi_1^1(r,s) = (1-r)(1-s) \qquad \phi_2^1(r,s) = r(1-s)$$

$$\phi_3^1(r,s) = rs \qquad \phi_4^1(r,s) = (1-r)s$$

quad

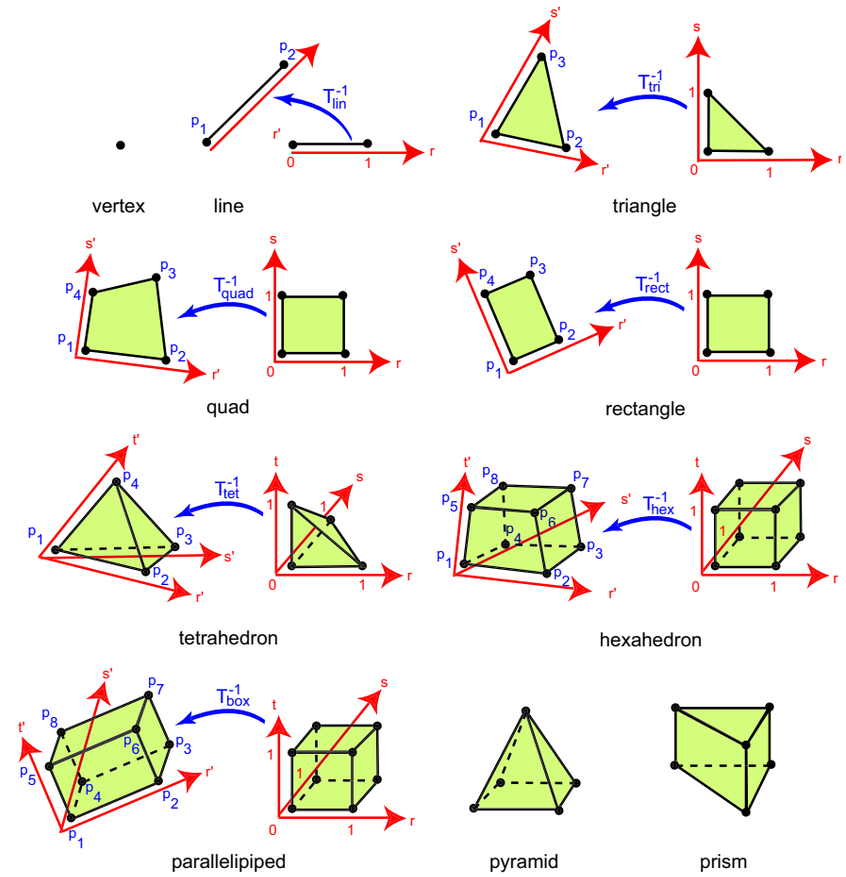
$$\phi_1^1(r,s,t) = 1-r-s-t \qquad \phi_2^1(r,s,t) = r$$

$$\phi_3^1(r,s,t) = s \qquad \phi_4^1(r,s,t) = t$$

tetrahedron

Cell Types

- Constant basis functions are simple and have virtually no computational cost, but provide only low quality approximations
- Higher-order basis functions provide more continuos reconstruction of the original data
- Linear basis functions are the next best choice, but they require some knowledge about the cell types used in the grid
  - example: trilinear interpolation in cube-like cell (defined by its 8 vertices)
- But cube-like cells are not the only cell choice - different types exist
- Vertices are sample points



cell types in world and reference coordinate systems

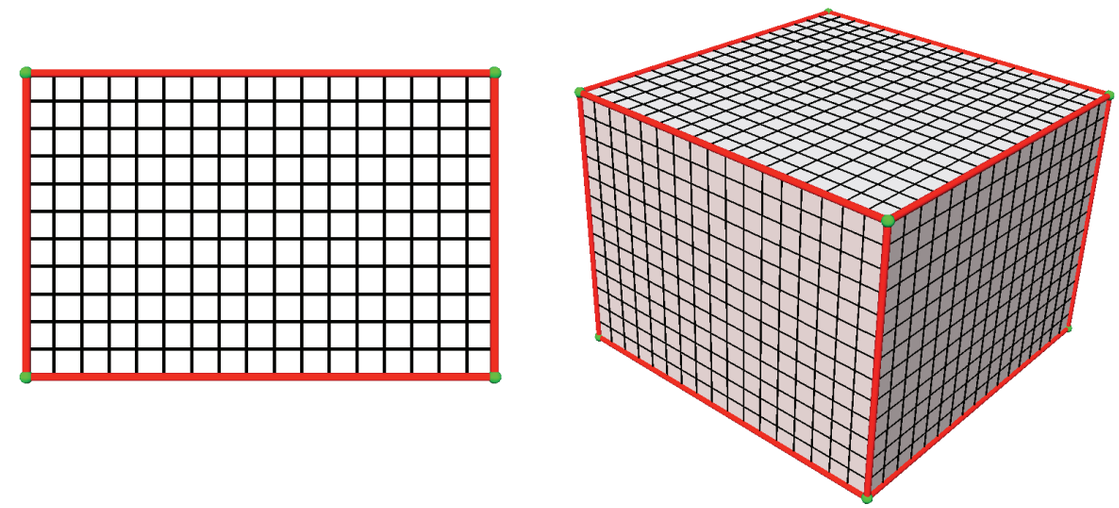
$$\phi_1^1(r,s,t) = (1-r)(1-s)(1-t)$$
$$\phi_2^1(r,s,t) = r(1-s)(1-t)$$
$$\phi_3^1(r,s,t) = rs(1-t)$$
$$\phi_4^1(r,s,t) = (1-r)s(1-t)$$

hexahedron

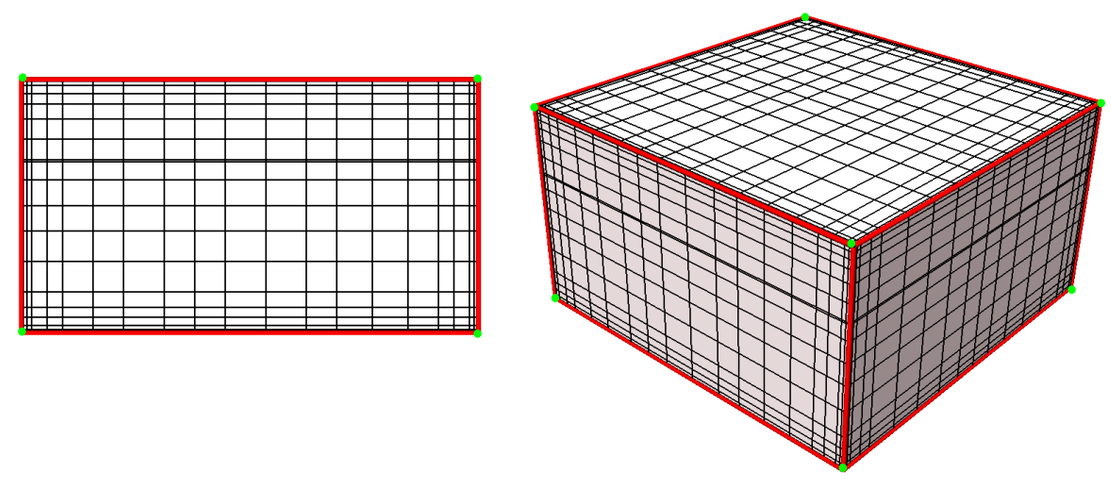
$$\phi_5^1(r,s,t) = (1-r)(1-s)t$$
$$\phi_6^1(r,s,t) = r(1-s)t$$
$$\phi_7^1(r,s,t) = rst \quad \phi_8^1(r,s,t) = (1-r)st$$

Grid Types

- Different grid types can be constructed from the various cell types
  - uniform grids (uniform sampling not necessarily optimal)
  - rectilinear grids (non-uniform sampling but still axis / domain aligned = sampling resolution can only be changed along axes)



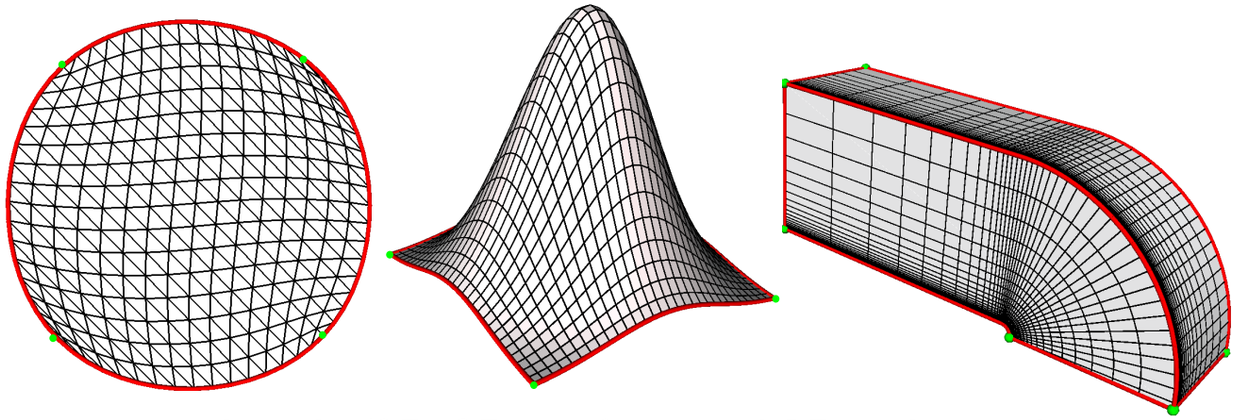
uniform grids in 2D and 3D domains



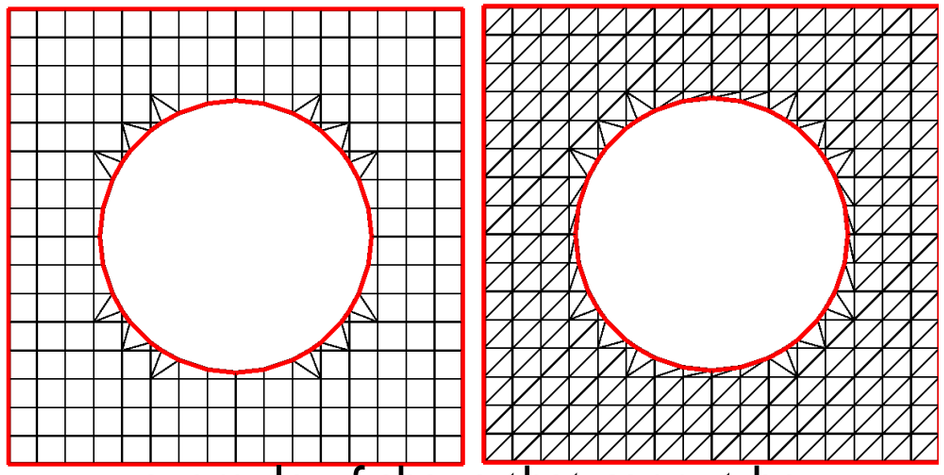
rectilinear grids in 2D and 3D domains

Grid Types

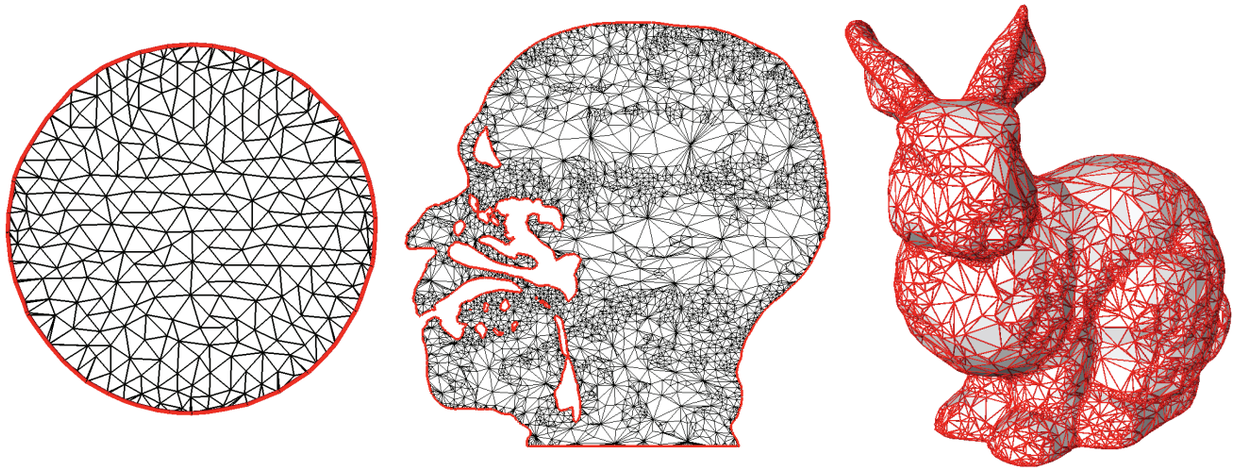
- Different grid types can be constructed from the various cell types
  - uniform grids (uniform sampling not necessarily optimal)
  - rectilinear grids (non-uniform sampling but still axis / domain aligned = sampling resolution can only be changed along axes)
  - structured grids (all sampling points can be freely specified, but topological order of vertices remains) - not all shapes can be efficiently modelled with uniform grids
  - unstructured grids (topological order of vertices does not remain)



structured grids (2D, 3D surface, and 3D volume domains)



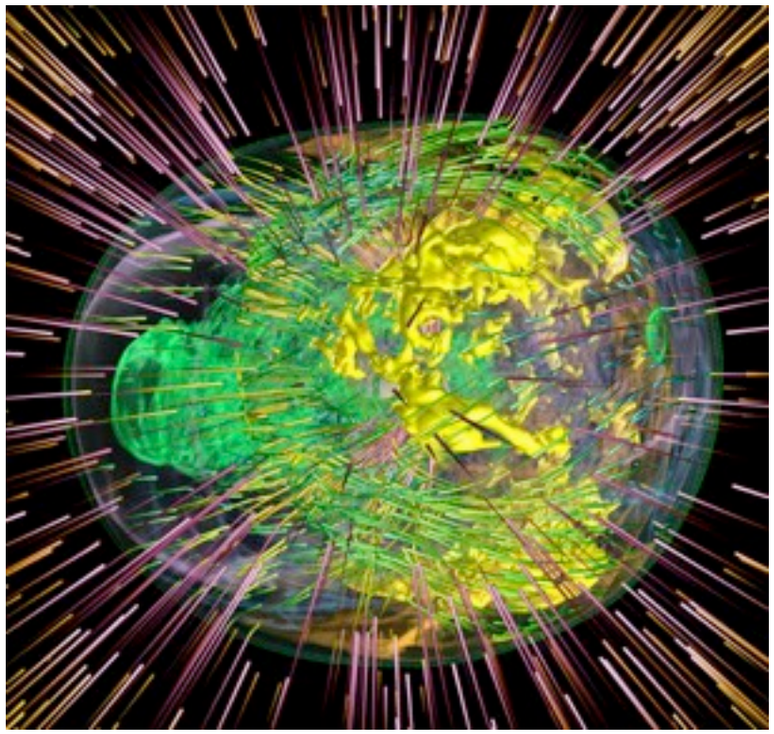
example of shape that cannot be modeled with structured grid



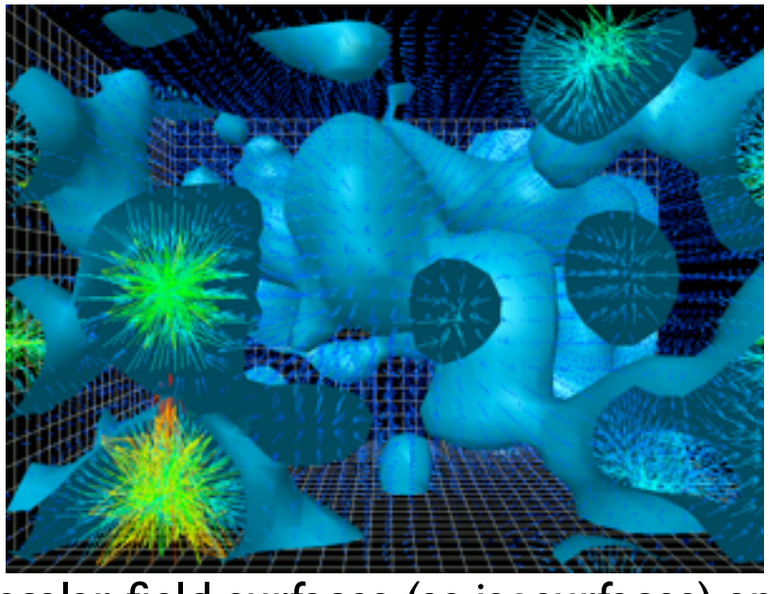
unstructured grids (2D and 3D surface domains)

Scalar Fields

- Scalar fields associate a value (eg., mathematical number of physical quantity) to every point in space
- Grids can be 1D, 2D, 3D, ... ,nD
- How can we visualize scalar fields?
  - the scalar values have to be mapped to some form of graphical visualization
  - intensity, color, texture, isosurfaces, ...
  - features (eg. contours) can be highlighted
  - can be combined with vector / tensor visualization
- We have to consider sampling and reconstruction, and with that, cell types and grid types



visualization of a velocity field superimposed with volume rendering of a scalar field from a supernova simulation



constant scalar field surfaces (as isosurfaces) and scalar-field gradient using the action energy density of the vacuum gluon field of quantum chromodynamics

Recap: Marching Cubes

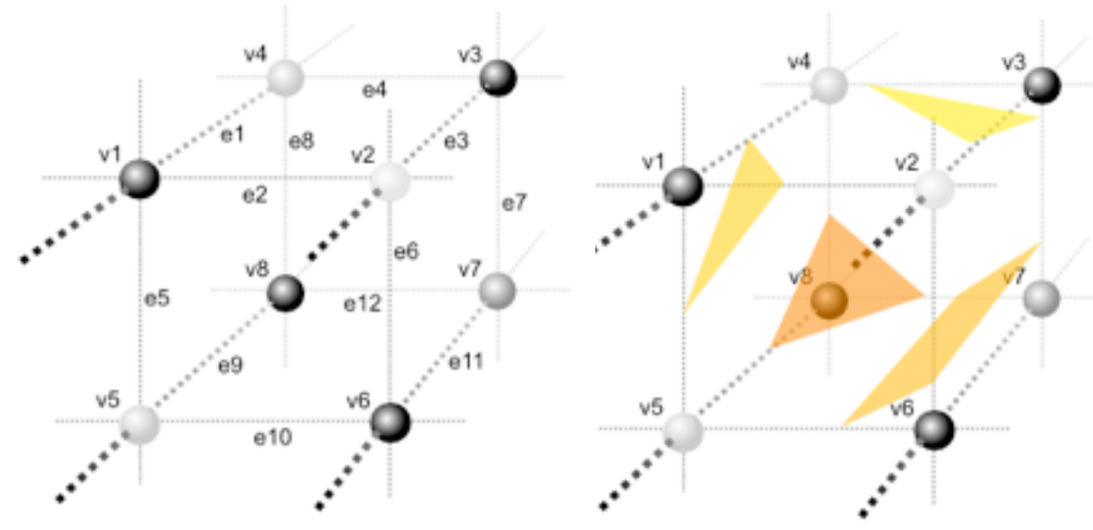
- Depending on a user-defined threshold, corners with a field value above the threshold are separated from corners that are below the threshold (i.e., we separate the corners that are in front of the surface from the ones that are behind the surface)

- The polygons are placed within the cube in such a way these two sets of corners are separated

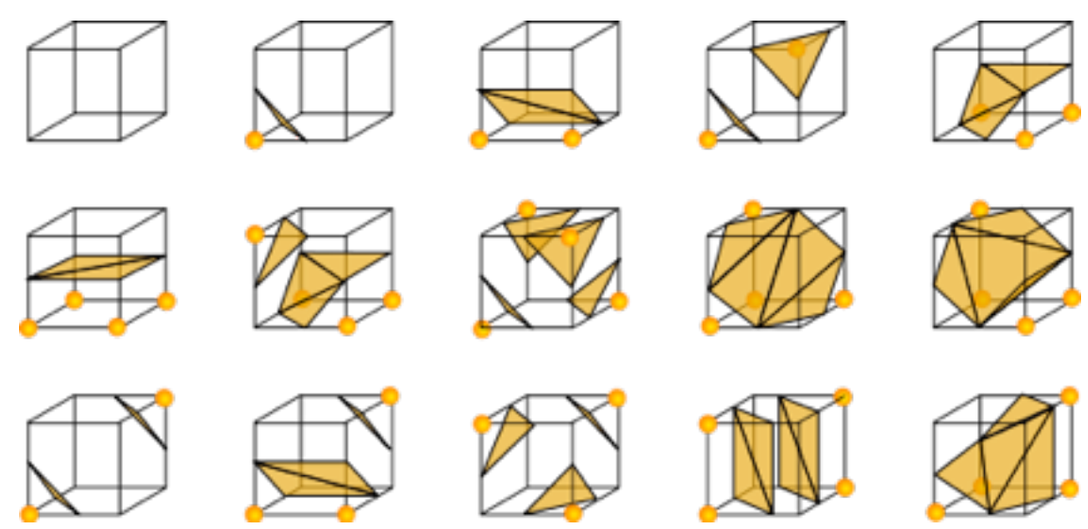
- There are 2^8=256 different separation - but due to symmetry (i.e., if symmetric cases are not considered) then only 15 different possibilites remain

- The exact positions and orientations of the polygons (i.e., their accurate intersections with their cubes) are determined form interpolating between the field values at the corners

- The final polygons represent fragments of the final surface mesh (normals are computed for each fragment in addition)



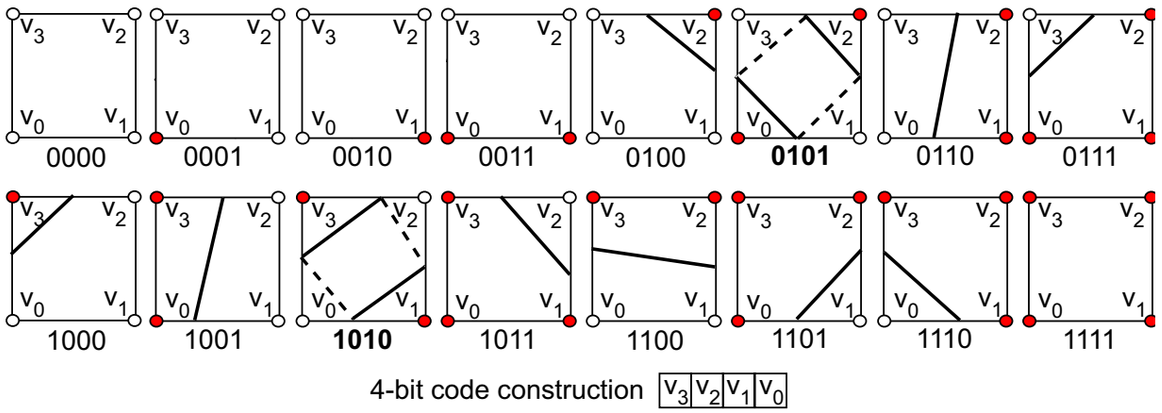
separating cube corners with polygons that are on one side of a surface from corners that are on the other side (based on the corners field values)



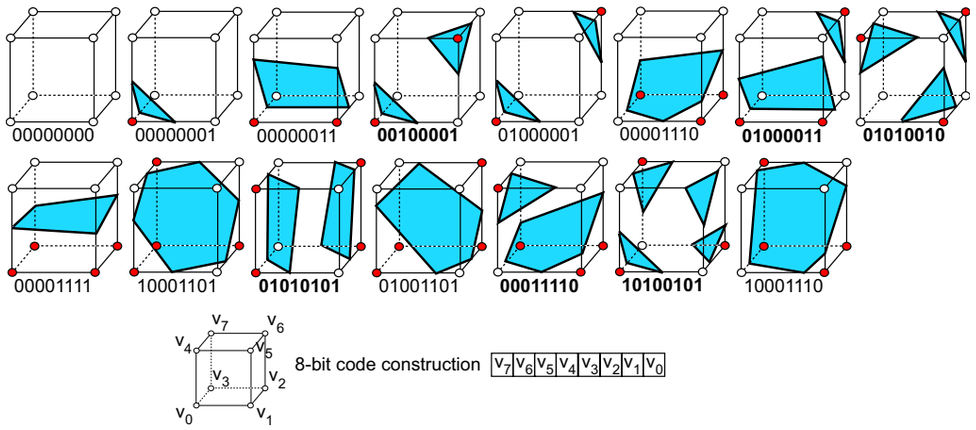
the 15 possibilites in the marching cube algorithm

Marching Squares

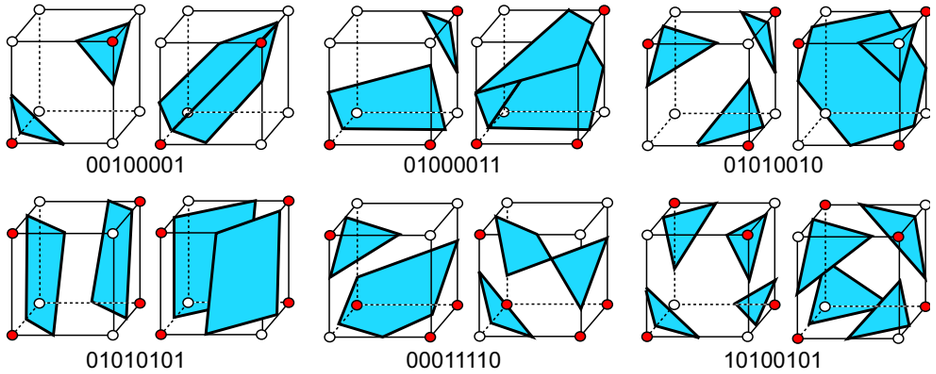
- It is the 2D version of marching cubes (we discussed this in volume rendering), and is called marching squares
- While marching cubes finds isosurfaces (e.g., by finding voxels in the volume with the same density), marching squares finds isolines (e.g., by finding same hight values in a height map)
- As we have seen for isolines (and therefore for marching squares), there is also ambiguity for isosurfaces (and therefore for marching cubes) - how to resolve them is depending on the application / implementation
- There are more advanced techniques that compute isolines and isosurfaces that do not cause ambiguity (marching tetrahedra or marching triangles)



marching square cases (redundancy not removed, there are 4 unique cases from a total of 16 cases)



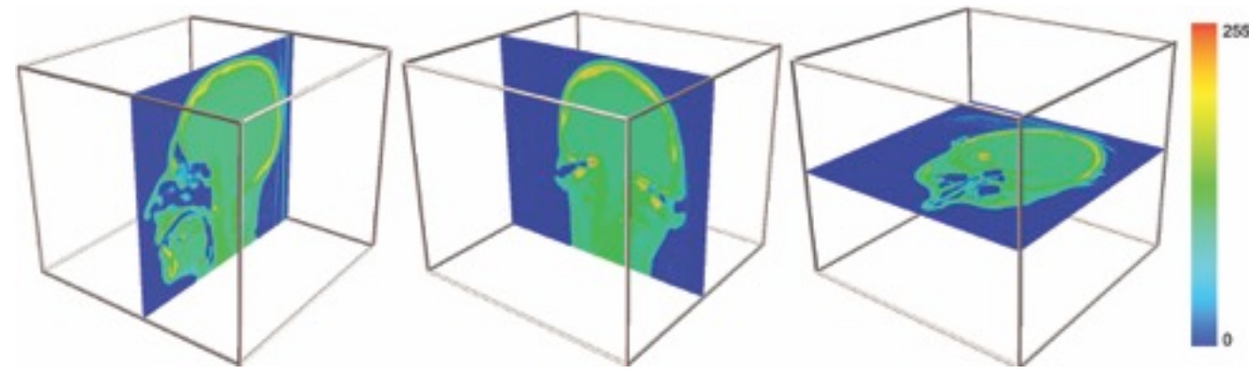
marching cube cases (redundancy removed, there are 15 unique cases from a total of 256 cases)



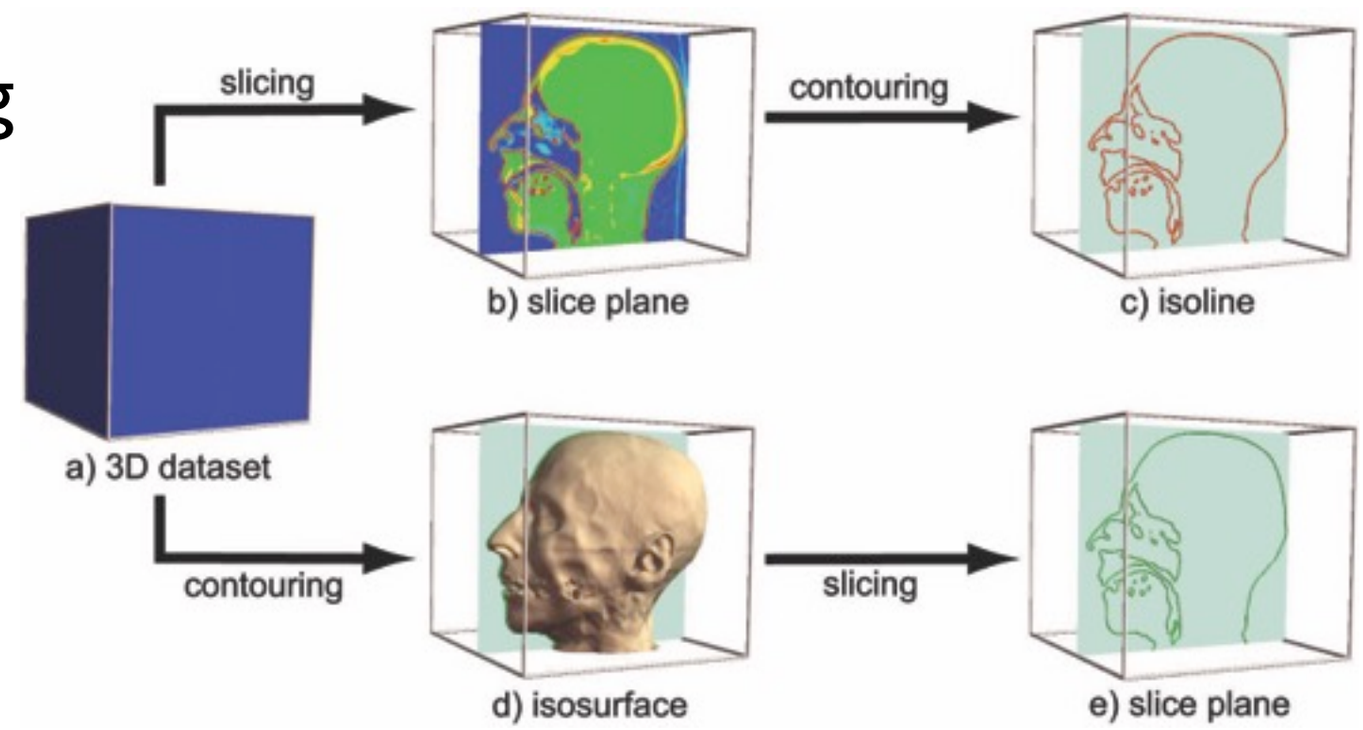
ambiguity cases for marching cubes

Isosurfaces vs. Isolines

- Slicing and contouring operations are commutative
- Slicing a volume at a particular plane and computing the contour lines at this plane is equivalent to computing the isosurafce and slicing it at the same plane
- This observation is important, as it suggests that 3D isosurfaces can be constructed by assembling a set of 2D isolines on a set of parallel slicing planes



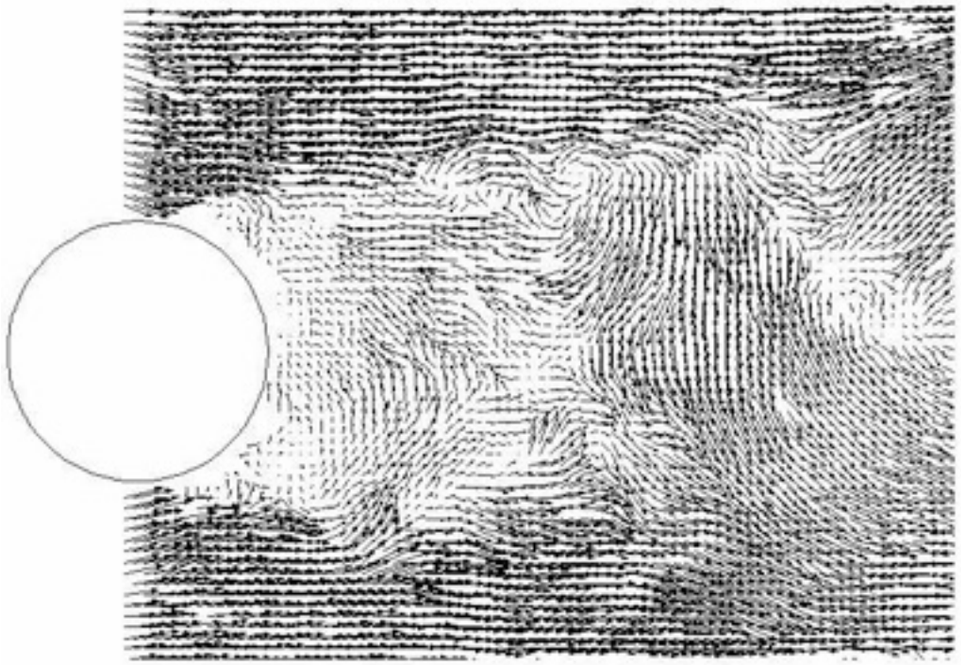
color coded slice planes in XYZ directions in a volume

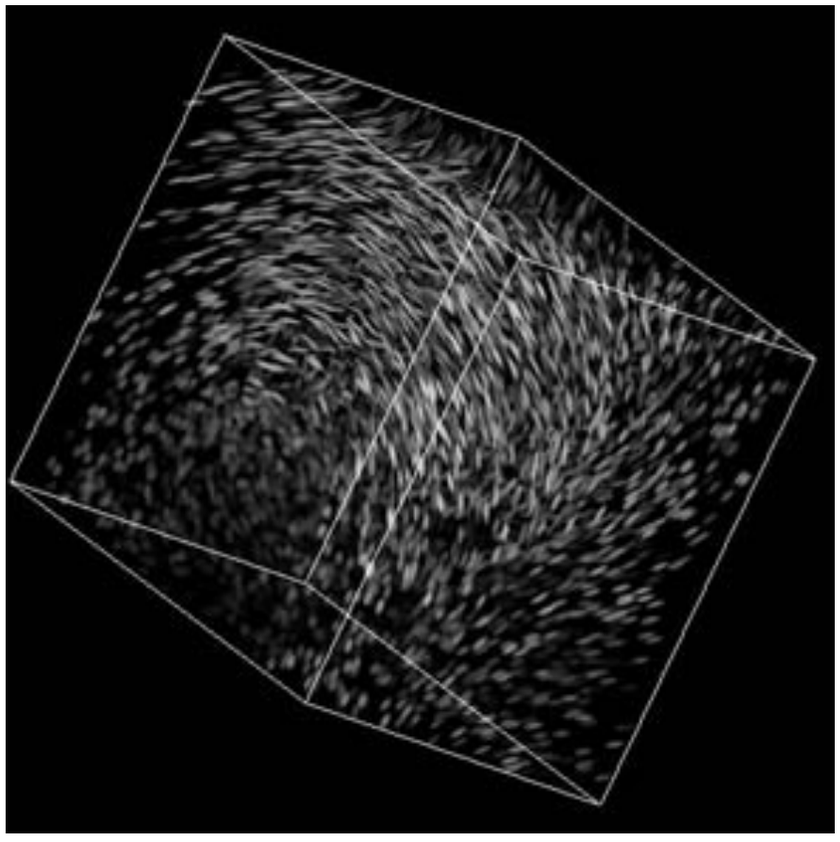


slicing the volume and computing the isolines is equivalent to computing the isosurface and slicing it at the same plane

Vector Fields

- So far, we have seen how scalar values can be visualized
- In many applications vector fields have to be visualized
- A vector is a tuple of n (usually 2 or 3) scalars
- An important domain for vector visualization is computational fluid dynamics (CFD)
- The solution of a CFD simulation is usually several datasets, each for a different time step
- For each time step, different attributes are computed: velocity, pressure, density, flow divergence, vorticity, etc.

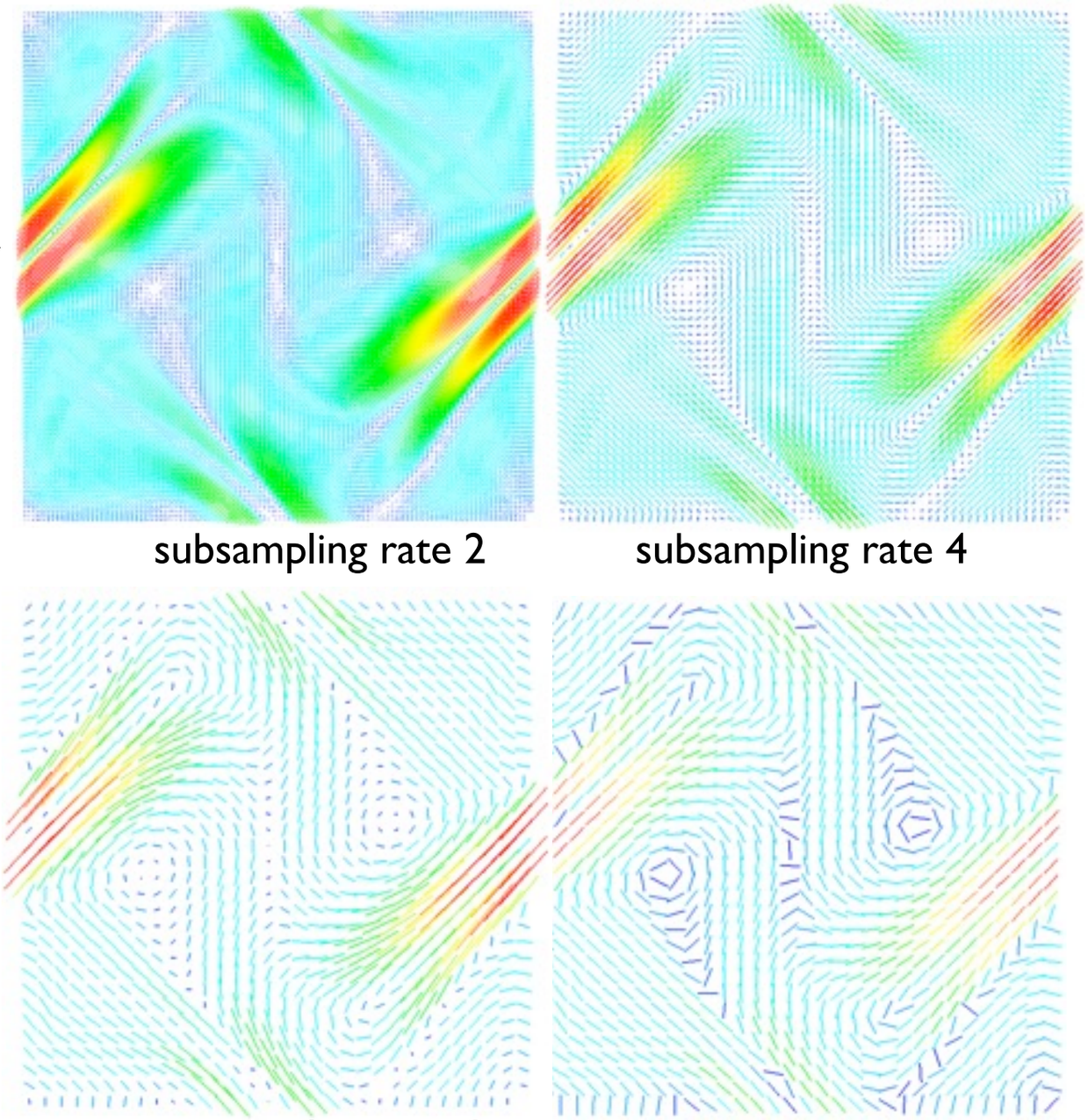




examples for vector fields in 2D (top) and 3D (bottom)

Vector Glyphs

- Vector glyphs (Greek for „sign“) are also called vector icons
- They can encode different properties, such as position, orientation, direction, size and color
- They can be of different types, such as lines (oriented lines are called hedgehogs), arrows, etc
- A vector field is subsampled, and vector glyphs with corresponding parameters are displayed for visualizing the vector field
- Ideally, vector glyphs should be as large as possible without intersecting with neighboring glyphs
- Consequently, there is a tradeoff between sampling rate of the data and interpretability of the vector glyphs

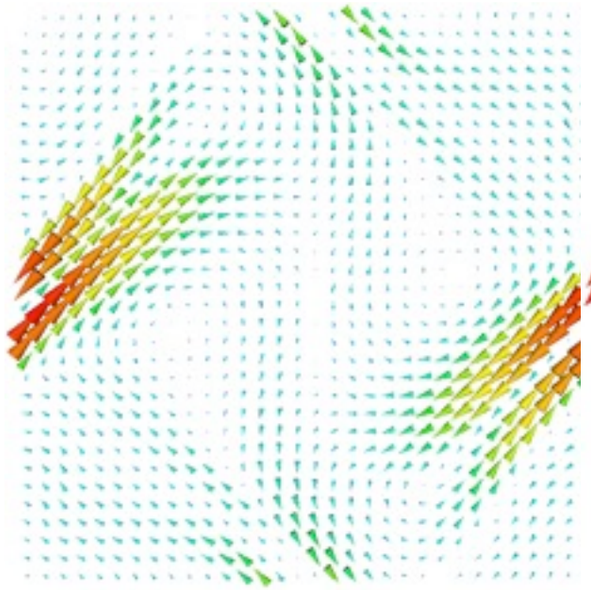


subsampling rate 2 subsampling rate 4

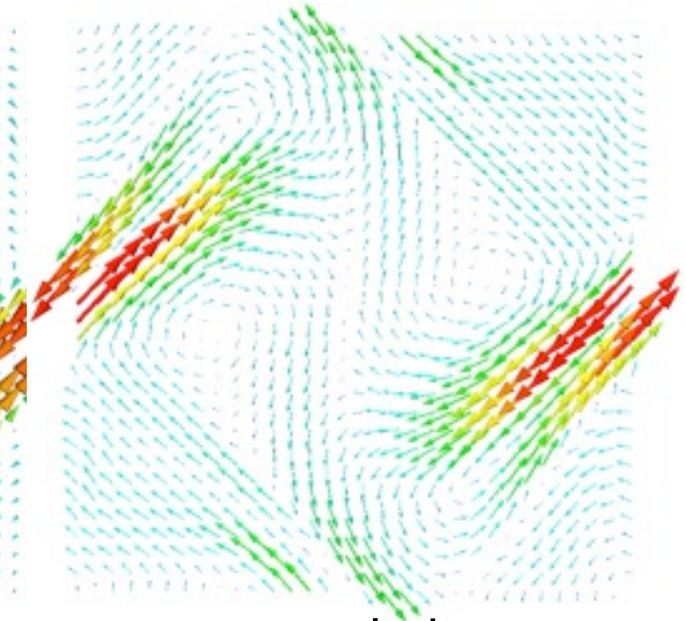
subsampling rate 8 subsampling rate 8 with constant line lengths

Vector Glyphs

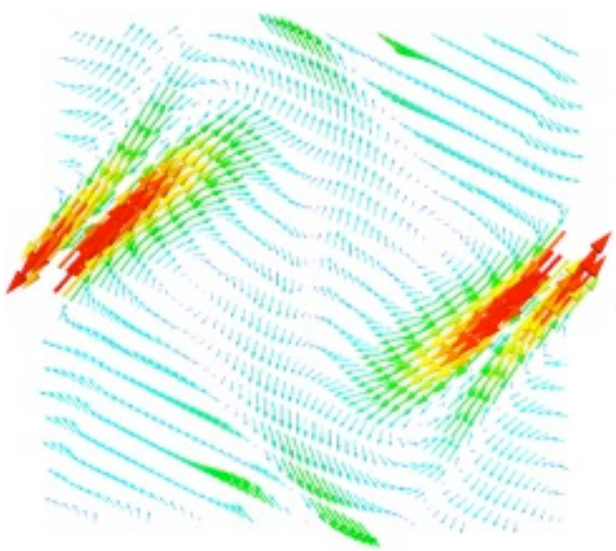
- Other glyph types might be better suited - such as cones or arrows
- Arrows can encode orientation and direction, while lines cannot encode direction
- However, the larger the glyphs the lower the sampling rate can be (i.e., the less dense is the visualization)
- While scalar color mapping produces a piecewise linear visualization (data is sampled at grid points, but is interpolated in between) vector glyphs produce a purely discrete visualization (data has to be interpolated cognitively - and this might be difficult)
- There is an undesired visual interference between the regular sampling grid and glyph directions - which can be alleviated by sampling with a random distribution



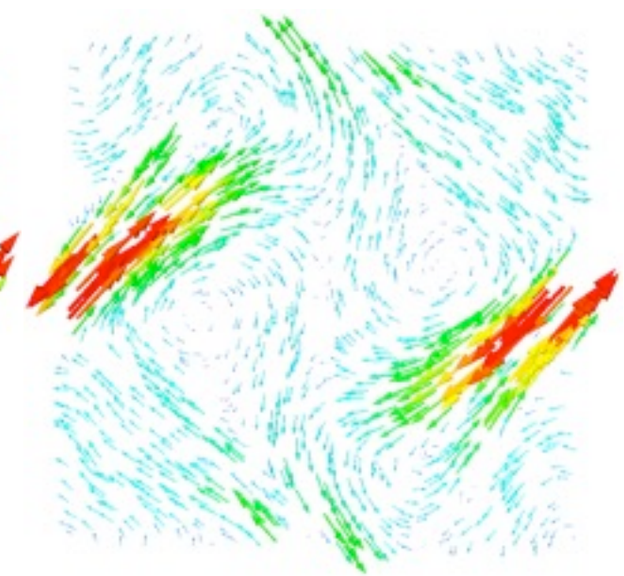
cone glyphs



arrow glyphs



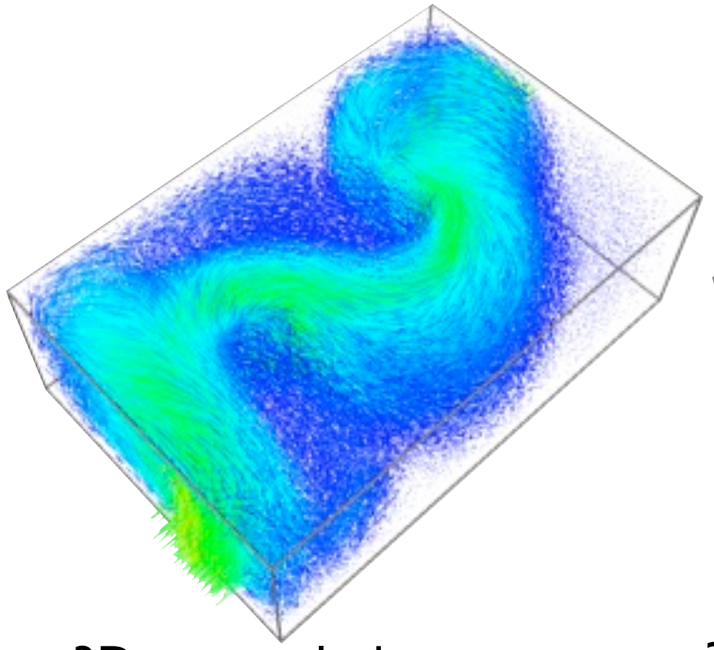
regular sampling distribution



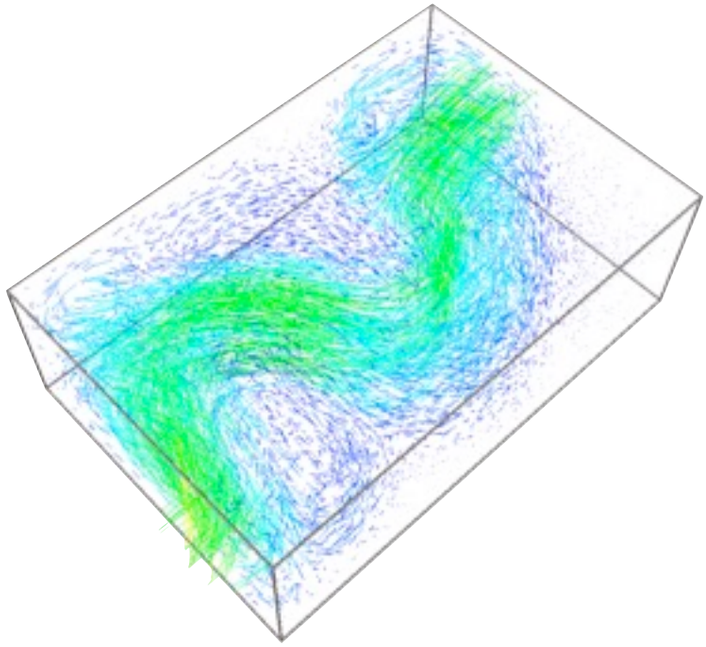
random sampling distribution

3D Vector Glyphs

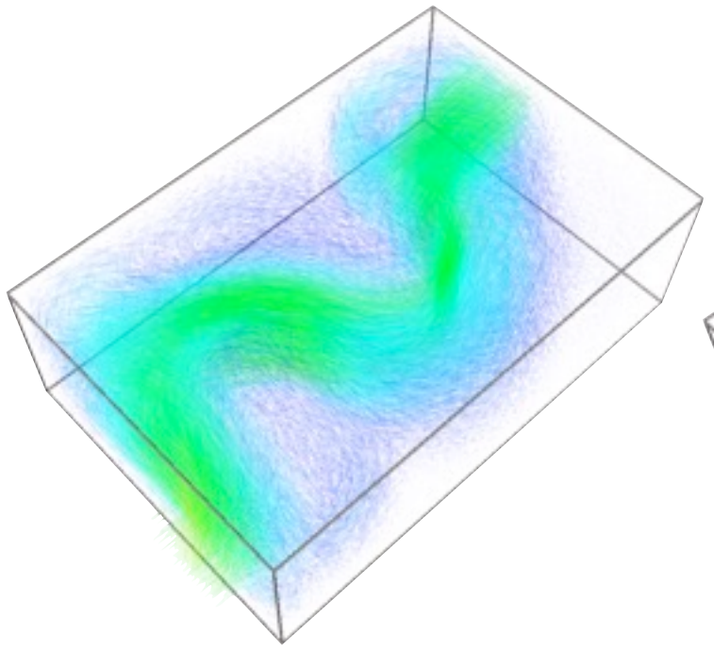
- Vector glyphs can also be used for 3D vector fields
- They basically share the same properties as vector glyphs in 2D - yet occlusion is an additional limitation
- Transparency can be used to reduce occlusion problems
- In particular if a grayscale color map is used in combination with transparency, 3D vector glyphs are effective to interpret for some applications



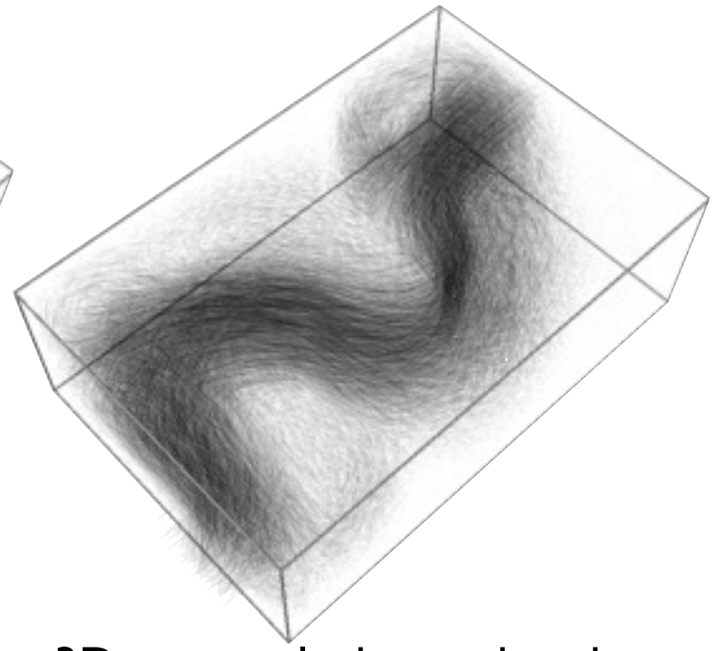
3D vector glyphs



3D vector glyphs at sparser sampling rate



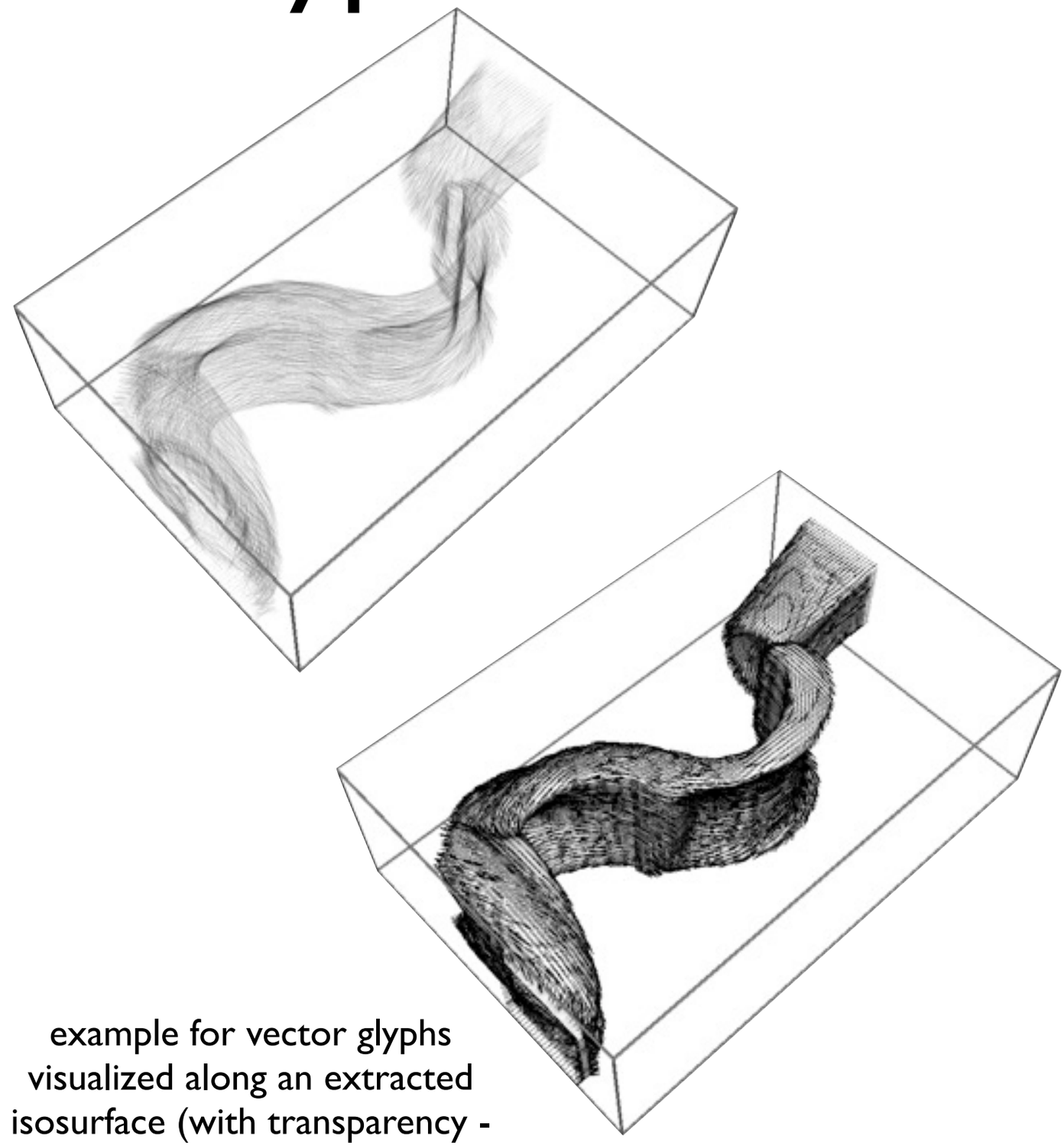
3D vector glyphs rendered with transparency



3D vector glyphs rendered with transparency and grayscale color map

3D Vector Glyphs

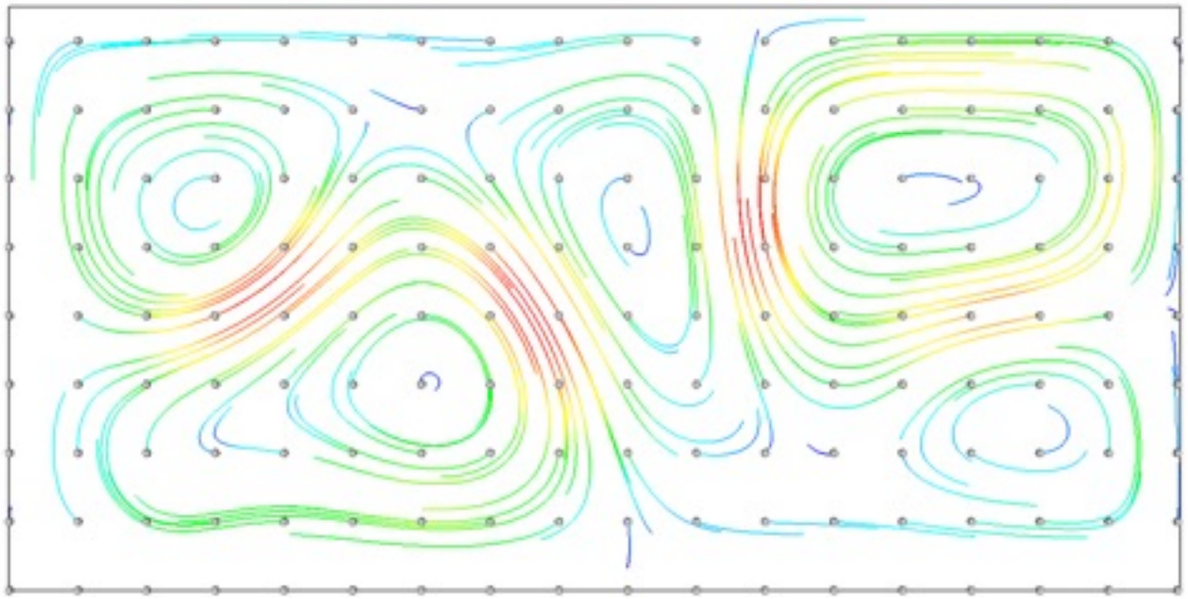
- Vector glyphs can also be used for 3D vector fields
- They basically share the same properties as vector glyphs in 2D - yet occlusion is an additional limitation
- Transparency can be used to reduce occlusion problems
- In particular if a grayscale color map is used in combination with transparency, 3D vector glyphs are effective to interprete for some applications
- In addition to 2D surfaces and 3D volumes, glyph-based visualization can also be applied along extracted isosurfaces (the vectors do not have to be tangent to the surface)



example for vector glyphs visualized along an extracted isosurface (with transparency - top, and without transparency - bottom)

Stream Lines

- Stream lines are computed as follows:
  - we start at a given seed point
  - we compute a new point in the direction of the corresponding vector as an offset from the seed point (the distance is scaled by the parameter delta_t, but the vector length itself can also influence the distance), and connect the two points
  - we repeat this for each new point until we reach an exit criteria (e.g., maximum length of stream line, maximum time / number of iterations, etc.)



stream lines through 2D flow field with seed points (gray)

$$S = \{ p(\tau), t \in [0,T] \}, p(\tau) = \int_{t=0}^{\tau} v(p)\,dt$$

with $p(0)=p_0$ - seed point

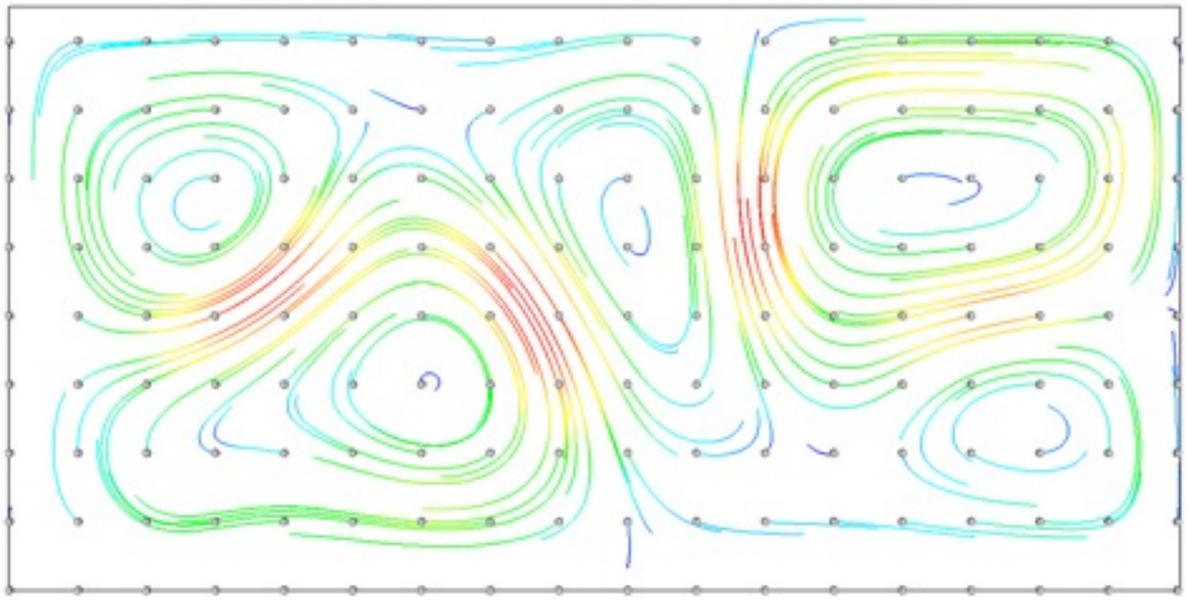
as Euler integration:

$$\int_{t=0}^{\tau} v(p)\,dt \approx \sum_{i=0}^{\tau/\Delta t} v(p_i)\Delta t, p$$

$$p_i = p_{i-1} + v_{i-1}\Delta t$$

Stream Lines

- This is equivalent of integrating the vector field by starting at the seed point
- Note, that delta_t is not a physical time (the vector field itself remains constant) but the integration time (integration parameter)
- The integral can be approximated on a discrete basis via Euler integration (or better Runge-Kutta integration)
- The quality of stream line visualizations is mainly influenced by:
  - sampling (density, regularity) of the seed points
  - discretization step width (delta_t)
- As for vector glyphs, they should be chosen in such a way that the data can be sampled adequately, and their visualization does not produce extreme clutter
- Both parameters can be locally adapted to achieve regional optimized effects



stream lines through 2D flow field with seed points (gray)

$$S = \{p(\tau), t \in [0,T]\}, p(\tau) = \int_{t=0}^{\tau} v(p)\,dt$$

with $p(0)=p_0$ - seed point

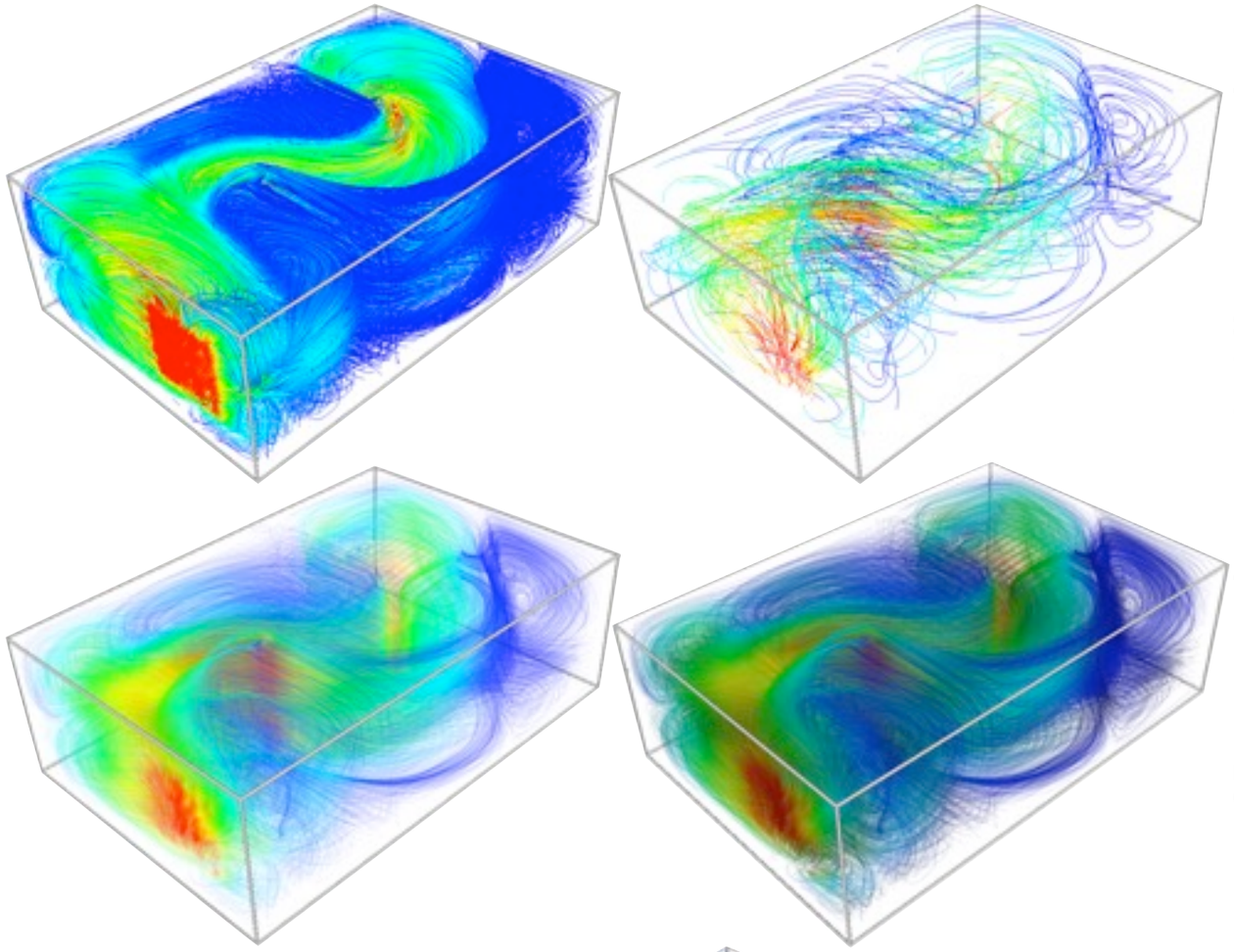
as Euler integration:

$$\int_{t=0}^{\tau} v(p)\,dt \approx \sum_{i=0}^{\tau/\Delta t} v(p_i)\Delta t, p$$

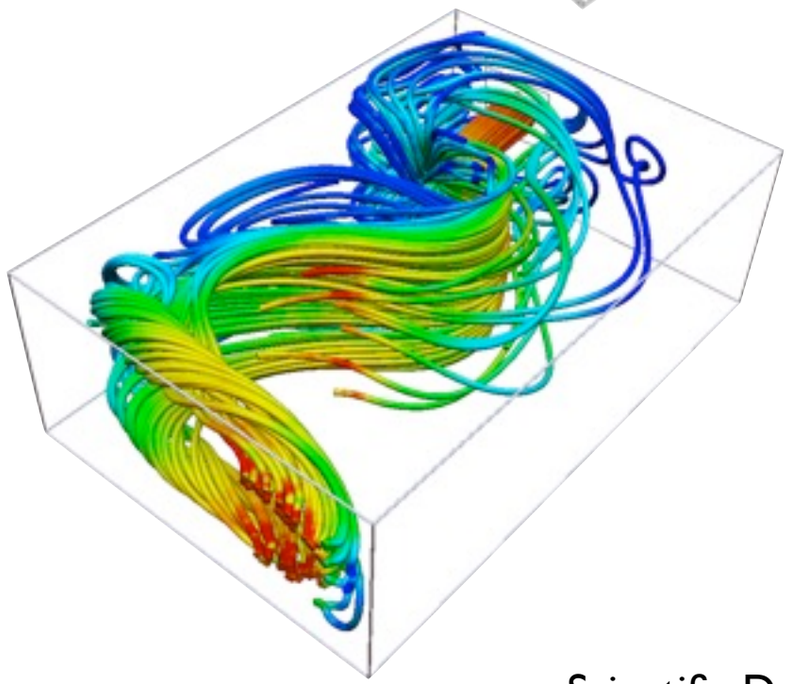
$$p_i = p_{i-1} + v_{i-1}\Delta t$$

3D Stream Lines and Tubes

- Instead of building lines with a piecewise addition of points, we can build tubes by sweeping a circular cross-section (orthogonal to the stream line tangent vector)
- The circles diameter can be varied to encode an additional parameter (in addition to color)
- Stream lines and stream tubes can be sampled from the seed point in direction of the vector field (downstream) or in the opposite direction by using a negative delta_t (upstream) - or in both directions
- Of course, stream lines and stream tubes can be implemented in 3D (with similar occlusion limitations as 3D vector glyphs)



3D stream lines with different sampling rate and opacity / transparency



3D stream tubes

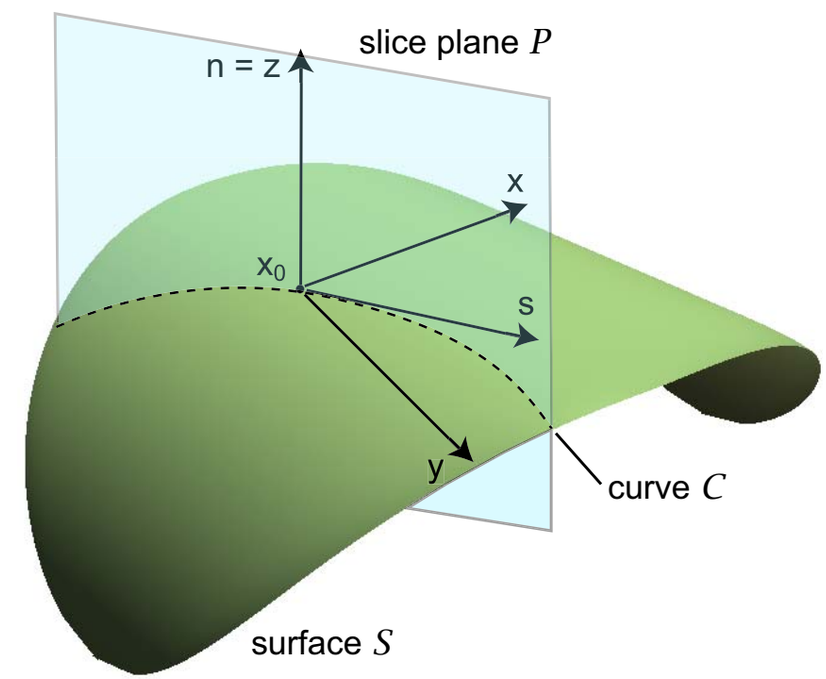
Tensor Attributes

- Tensor attributes are high-dimensional generalizations of vectors
- We describe them by example
- The curvature of a planar curve f at point x_0 can be computed with the 2nd derivative of f at x_0
- The same applies to surfaces - but we have to specify in which direction s we want to compute the curvature
- The curvature at point x_0 of a surface in direction s is the curvature at x_0 of the planar curve that is created by intersecting a plane (spanned by normal n and s) with the surface

$$y = f(x), f(x_0) = 0$$

$$c = \frac{\partial^2 f}{\partial x^2}(x_0) = f''(x_0)$$

curvature c of a planar curve f at neighborhood point x_0 (local coordinate system at origin x_0)



curvature of surface

$$z = f(x,y), f(x_0) = 0$$

at neighborhood of x_0 (2D now!) - again with local coordinate system around x_0

Tensor Attributes

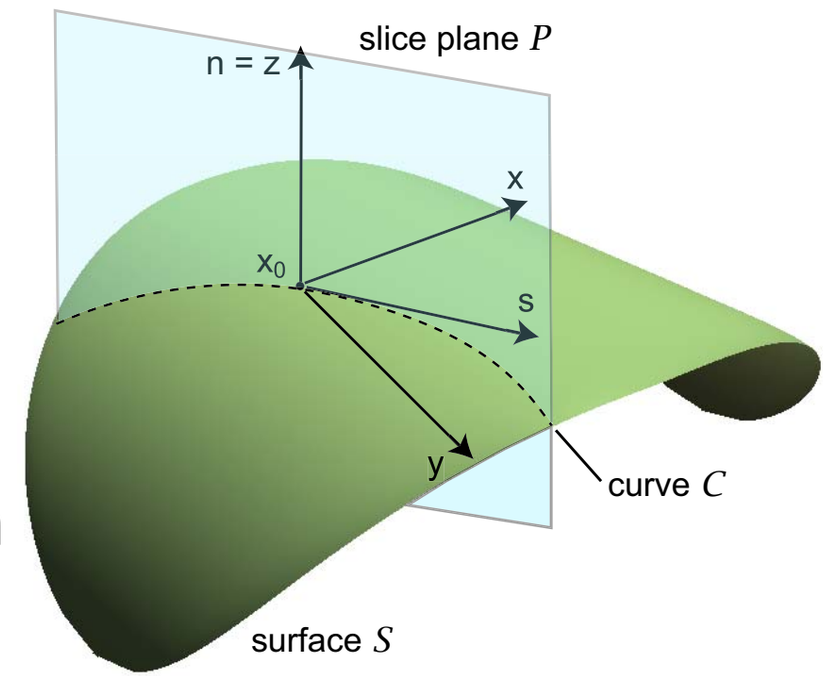
- In contrast to planar curves (only two directions at each point), there is an infinite number of directions s for which we can compute the curvature at a point of the surface

- Considering the gradient g at a point x_0 on the surface f(x,y), the rate of variation in some direction s can be computed using the Hessian H of f

- The Hessian H of f combines all 2nd-order partial derivatives of an n-dimensional function

- Second derivative test: H allows classifying critical points of a function (e.g., local minimum/maximum, saddle points)

- H also allows to express the curvature of of f at x_0 in direction s with $s^T Hs$ (this is known as normal curvature)



curvature of surface
(local coordinate system)

$$z = f(x,y), f(x_0) = 0$$

$$g(x_0 + s) = g(x_0) + H(x_0)s$$

rate of gradient variation along s

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

Hessian of f

$$\frac{\partial^2 f}{\partial s^2}(x_0) = s^T Hs$$

curvature of surface at x_0 in direction s

Tensor Attributes

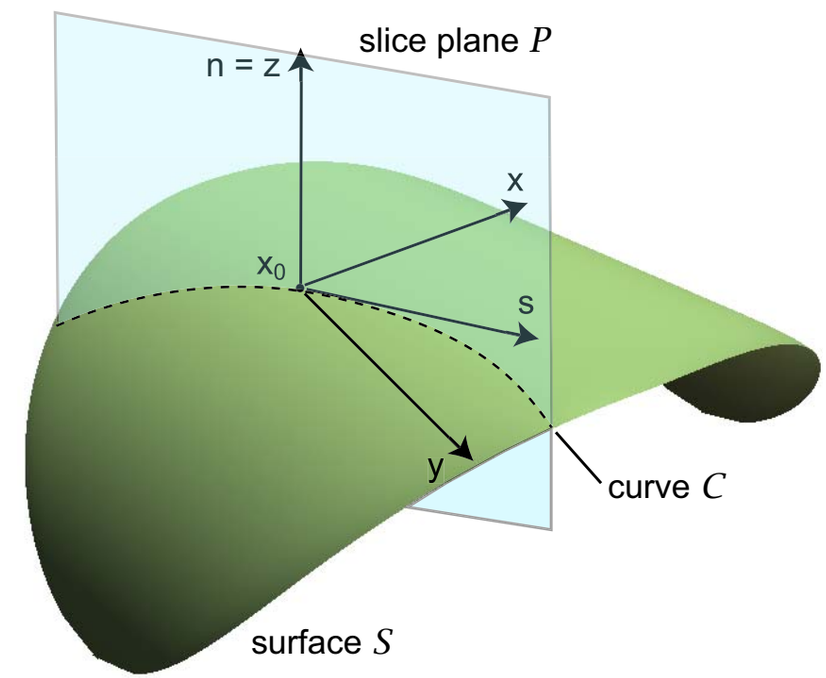
- If the surface is defined implicitly f(x,y,z)=0 in a global coordinate system instead of a local one, we have to apply its 3x3 Hessian matrix of partial derivatives of f(x,y,z) in the global coordinate system

- In this example, the Hessian matrix H is called the curvature tensor of the surface f

- Besides curvature, tensors can describe other physical quantities that depend on direction (e.g., water diffusion, stress/stain of material, etc.)

- Tensors are characterized by their rank (scalars are tensors of rank 0, vectors are tensors of rank 1, Hessian curvature tensor is rank 2 symmetric tensor, general tensor of rank k is k-dimensional)

- We focus in rank 2 tensors only



curvature of surface
(global coordinate system)

$$f(x,y,z)=0$$

$$H=\begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}$$

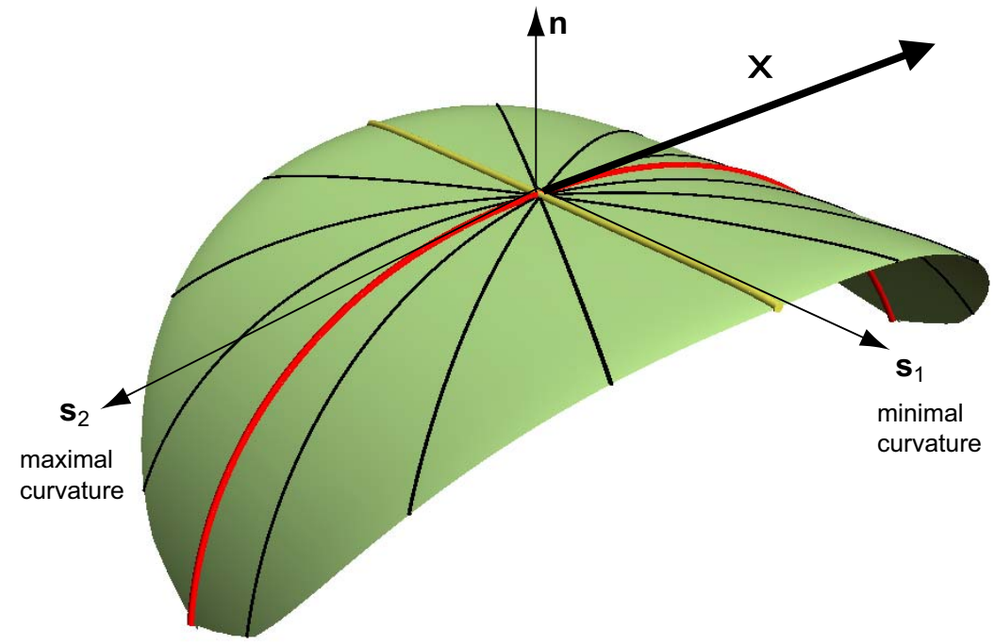
Hessian of f

$$\frac{\partial^2 f}{\partial s^2}(x_0)=s^T H s$$

curvature of surface at x_0 in direction s

Principle Component Analysis

- For the visualization of tensor fields, we might be interested in only the extreme directions of a tensor (e.g., the directions with minimal / maximal curvature / diffusion speed)
- These directions can be found using principle component analysis (PCA)
- A PCA on 2x2 tensor matrix leads to two principle directions (called eigenvectors) s_1 and s_2 as well as to the two corresponding principle magnitudes λ_1, λ_2 (called eigenvalues)
- Eigenvectors represent the directions towards extremal (minimal and maximal) curvature / diffusion, and eigenvalues represent the extremal values in these directions
- For a nxn symmetric matrix, the principle directions are perpendicular to each other



$s^T = (\cos\alpha, \sin\alpha)$ with alpha being the angle between s and x and |s|=1

directions s for which the normal curvature has extremal values are the solutions of the equation system:

$$Hs = \lambda s$$

where H is 2x2 and lambda is a real number

we can rewrite this (I is 2x2 identity matrix):

$$(H - \lambda I)s = 0 = \det(H - \lambda I), s \neq 0$$

solving this yield two solutions: λ_1, λ_2

as well as the corresponding directions: s_1, s_2

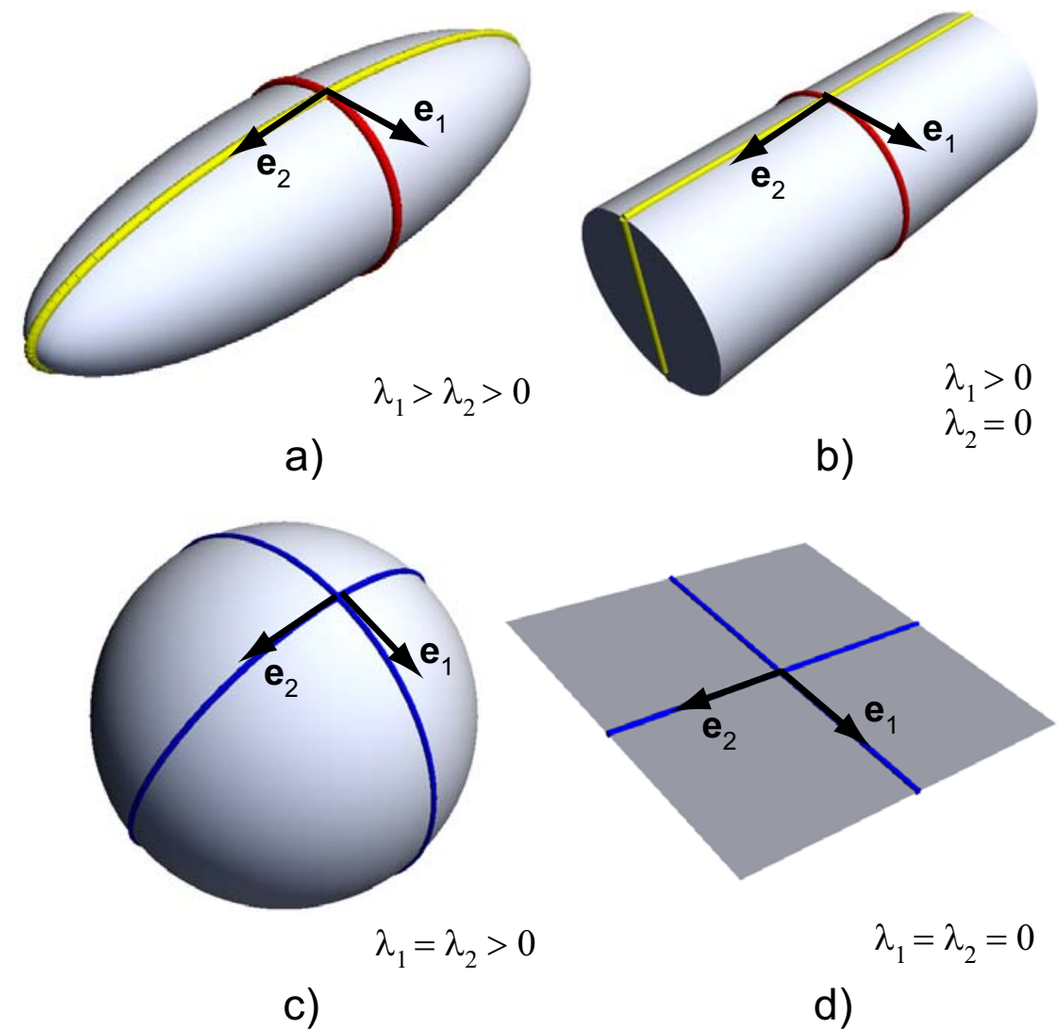
Principle Component Analysis

- This also applies in exactly the same way if the surface is defined implicitly f(x,y,z)=0 within a global coordinate system - yet, the 3x3 Hessian matrix has to be used
- In this case, we get three solutions (i.e., eigenvectors and eigenvalues)
- Let's call the eigenvectors $e_{1,2,3}$ from now (major, medium, minor eigenvector)
- While e_1 and e_2 are tangent to the surface and represent the minimal and maximal directions, e_3 is the surface normal n

a 3x3 tensor has three eigenvectors and eigenvalues:

$$e_1, e_2, e_3$$

$$\lambda_1 > \lambda_2 > \lambda_3$$



comparing eigenvalues with respect to surface curvature

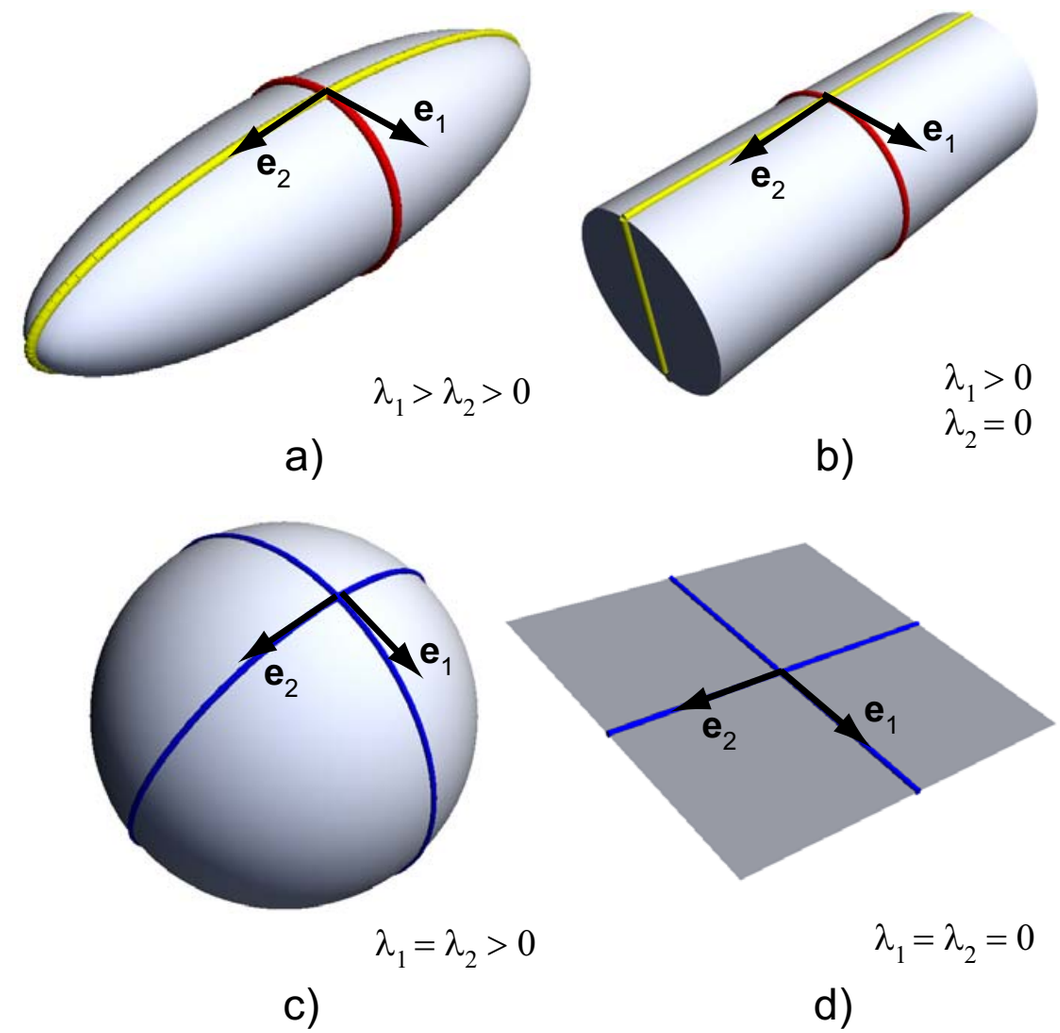
Principle Component Analysis

- The eigenvalues can be compared to identify the surface topology
- For curvature tensors, it can be shown that the third eigenvalue is always zero (but not in general, such as diffusivity)
- Applying this to diffusion means that the eigenvectors represent the extreme diffusion directions and the eigenvalues represent the degree of isotropy/anisotropy
- How can we apply this to visualize diffusion?
- How would you visualize the degree of isotropy/ anisotropy?

a 3x3 tensor has three eigenvectors and eigenvalues:

$$e_1, e_2, e_3$$

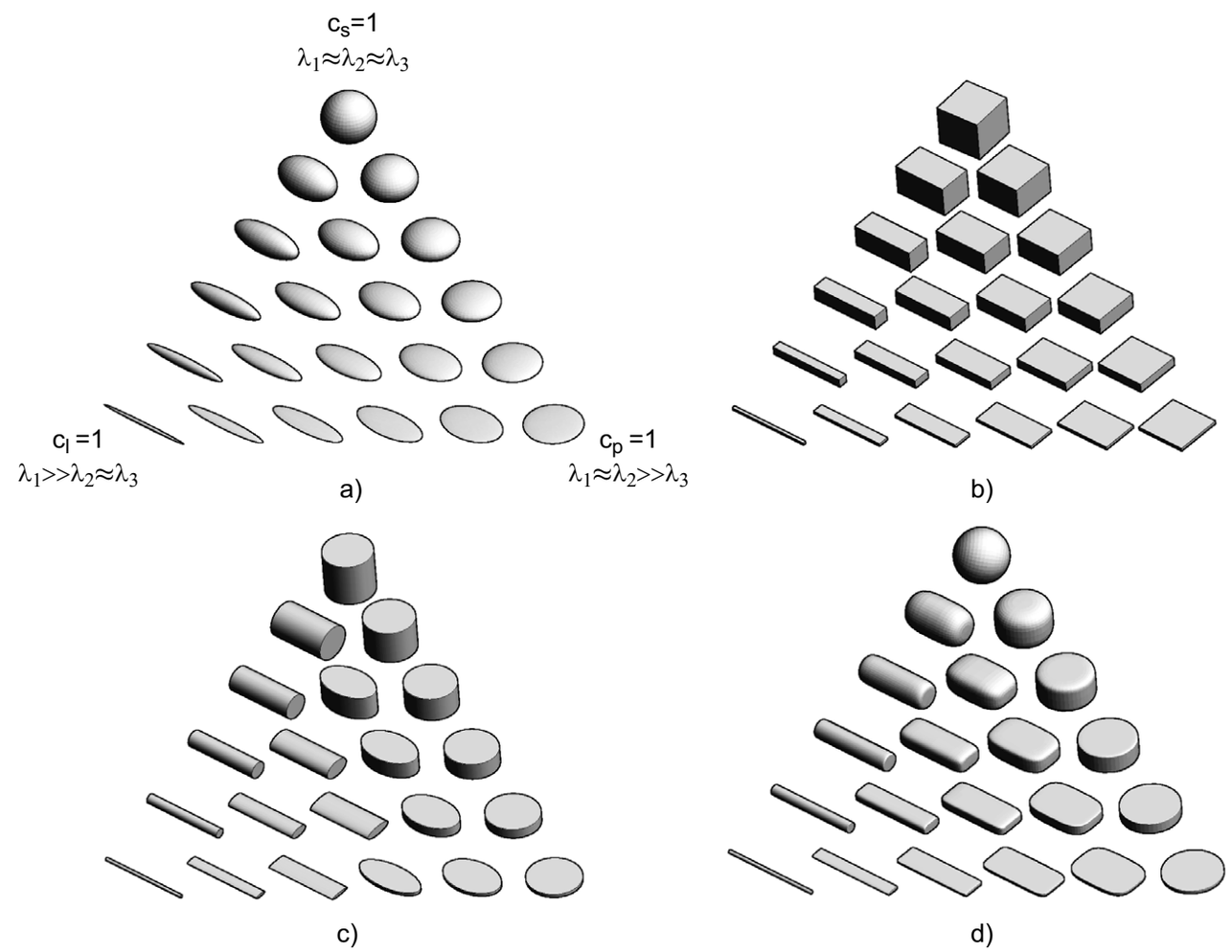
$$\lambda_1 > \lambda_2 > \lambda_3$$



comparing eigenvalues with respect to surface curvature

Tensor Glyphs

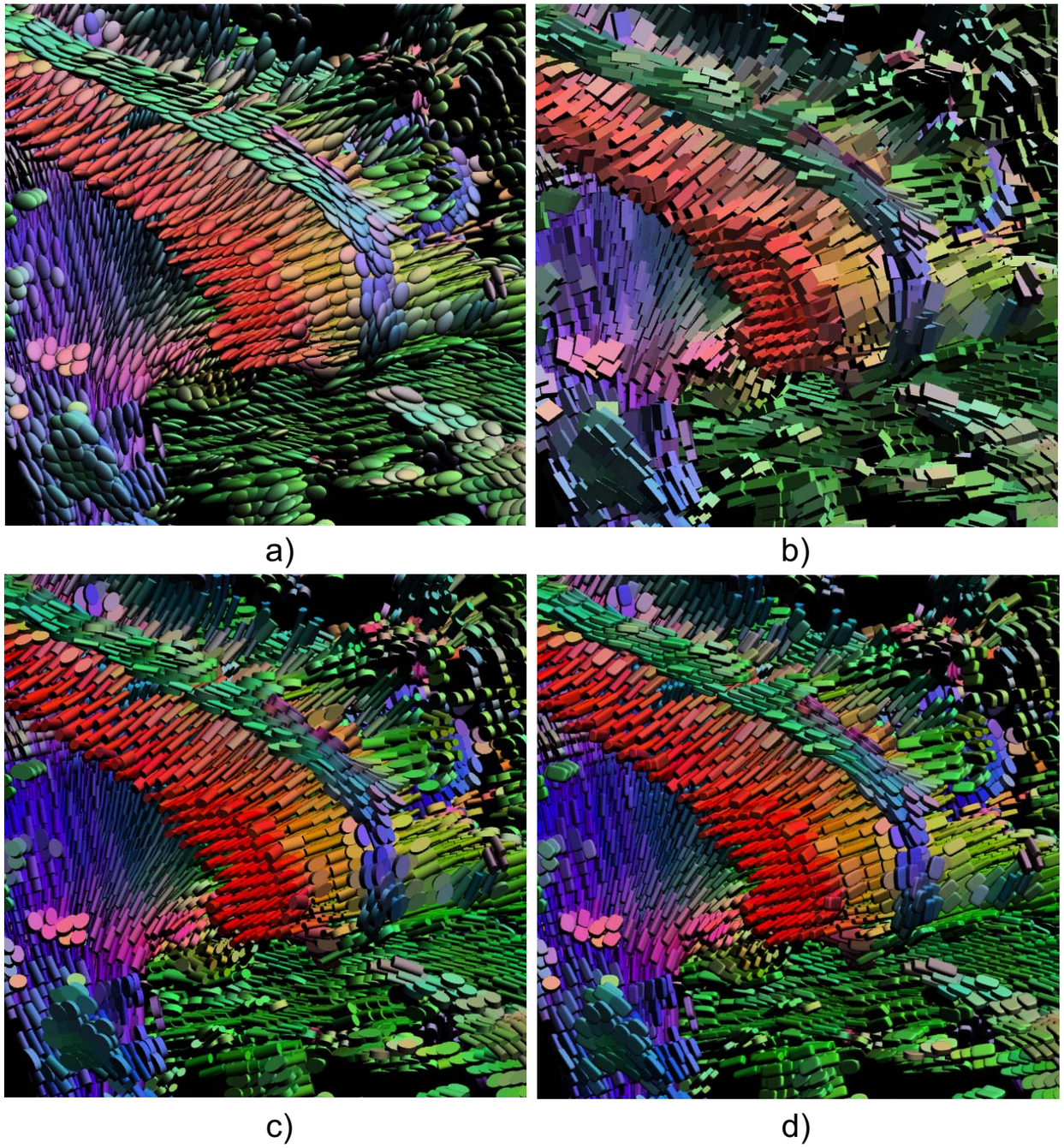
- Tensor glyphs is a more general form of vector glyphs
- They encode eigenvectors with their two directions and corresponding eigenvalues with their size (also in two directions)
- In addition, the glyphs can be color coded (e.g., with their direction)
- Tensor glyphs suffer from similar problems as vector glyphs



different tensor glyph shapes: ellipsoids (a), cubolids (b), cylinders (c), superqaudrics (d)

Tensor Glyphs

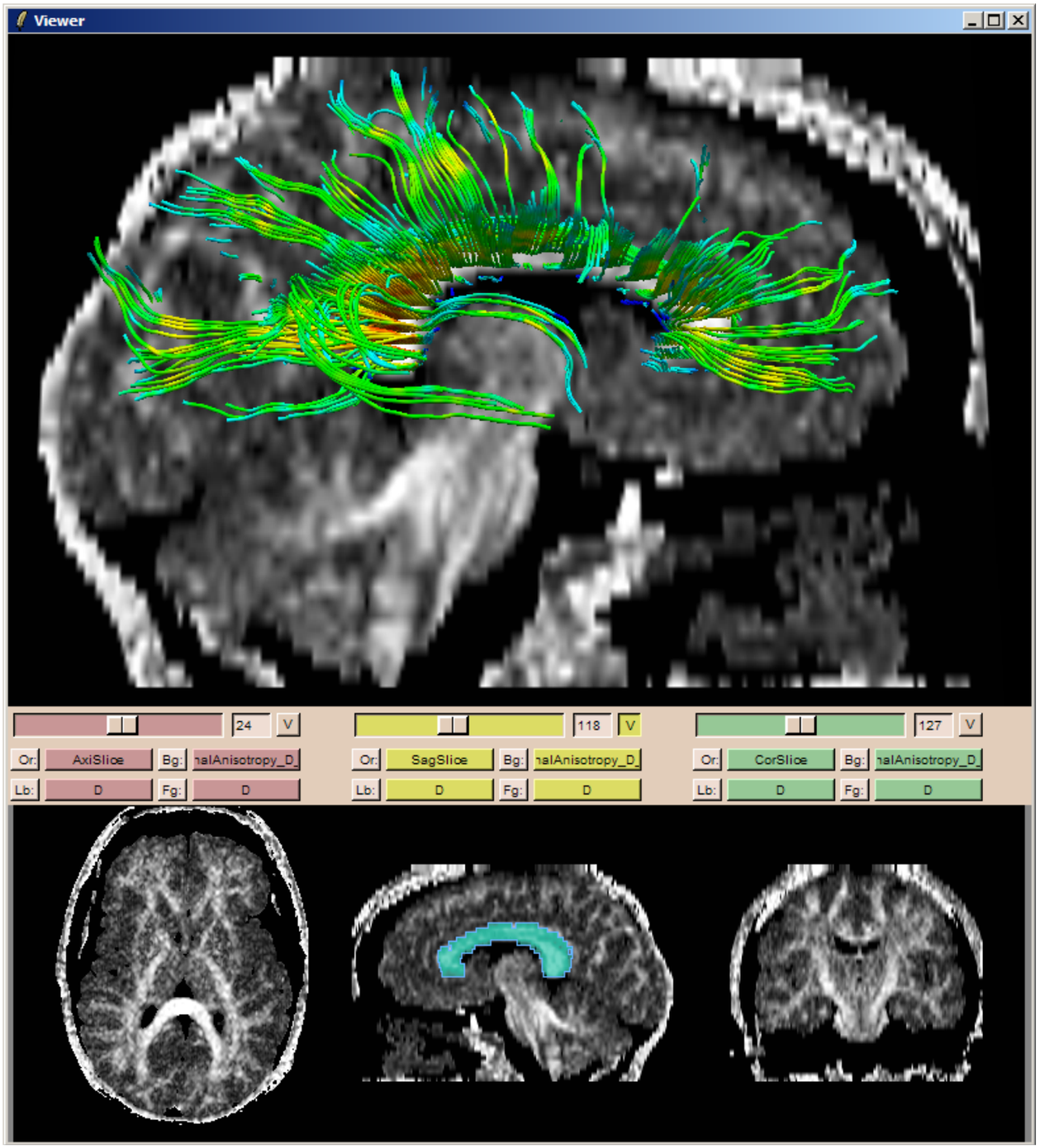
- Tensor glyphs is a more general form of vector glyphs
- They encode eigenvectors with their two directions and corresponding eigenvalues with their size (also in two directions)
- In addition, the glyphs can be color coded (e.g., with their direction)
- Tensor glyphs suffer from similar problems as vector glyphs



close-up examples for DT-MRI data visualized with ellipsoids (a), cubolids (b), cylinders (c), superqaudrics (d)

Fiber Tracking

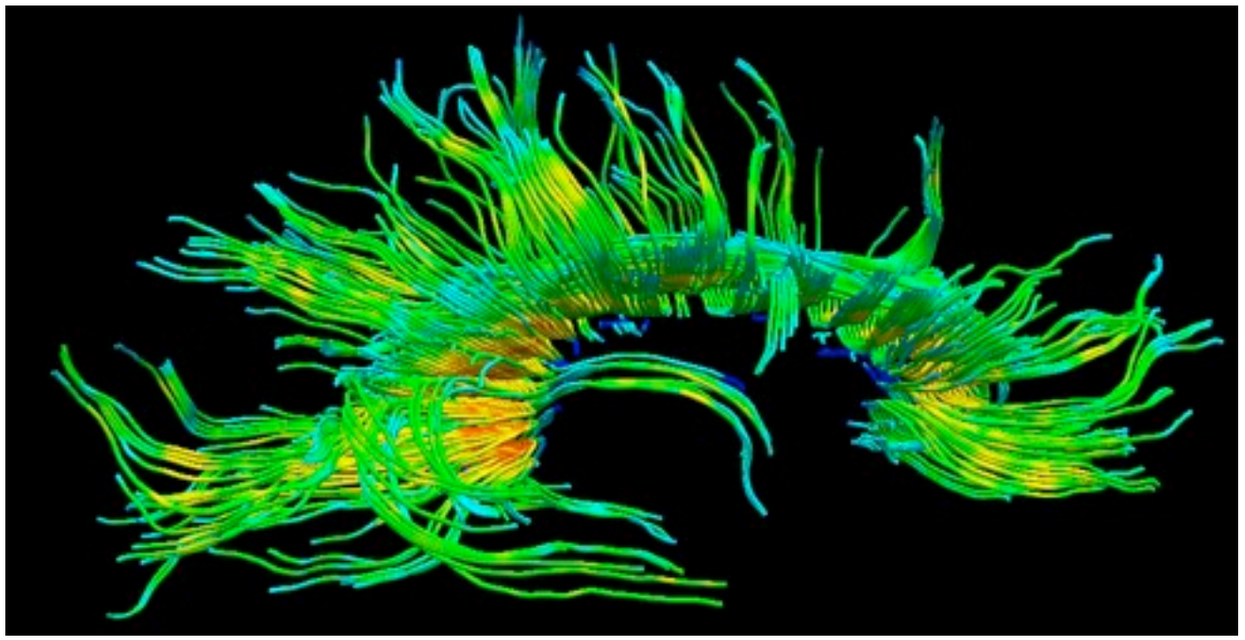
- Neural fibers can be tracked by constructing streamlines along high anisotropic regions (i.e., major eigenvectors)
- A strong distinction between the major eigenvectors and all the others is necessary for making streamlines efficient (and low of noise)
- As before, choosing appropriate seed points is essential for a good visualization (a lot experience is needed)

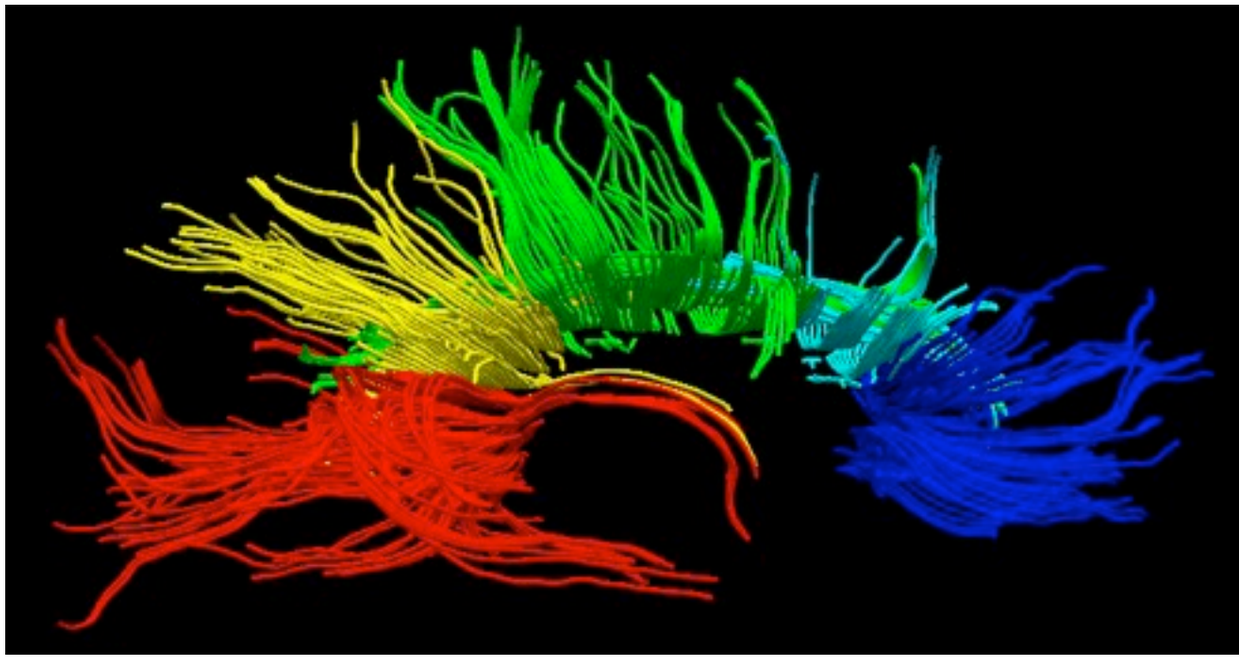


fiber tracking example

Fiber Tracking

- Neural fibers can be tracked by constructing streamlines along high anisotropic regions (i.e., major eigenvectors)
- A strong distinction between the major eigenvectors and all the others is necessary for making streamlines efficient (and low of noise)
- As before, choosing appropriate seed points is essential for a good visualization (a lot experience is needed)
- However, these fibers / streamlines do not encode the eigenvalues
- Do you have a better idea?

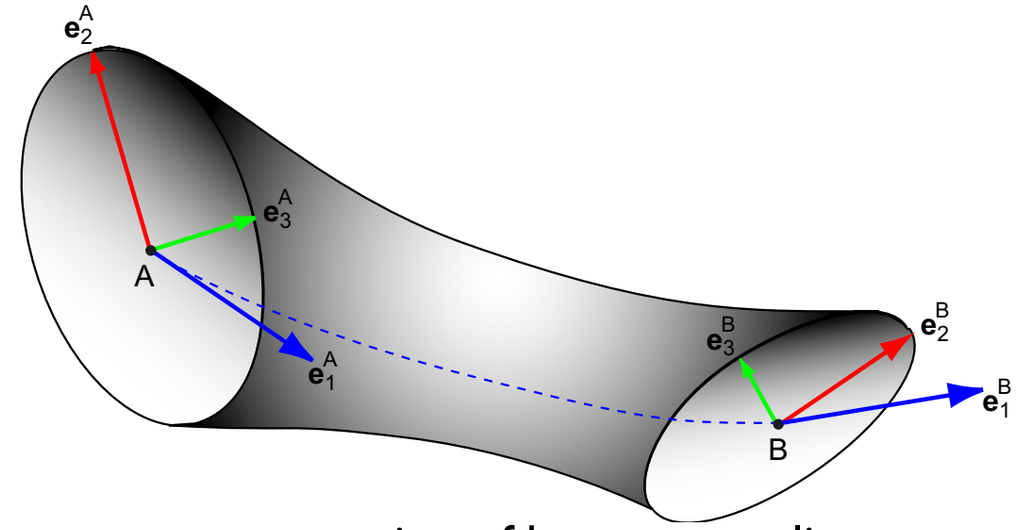




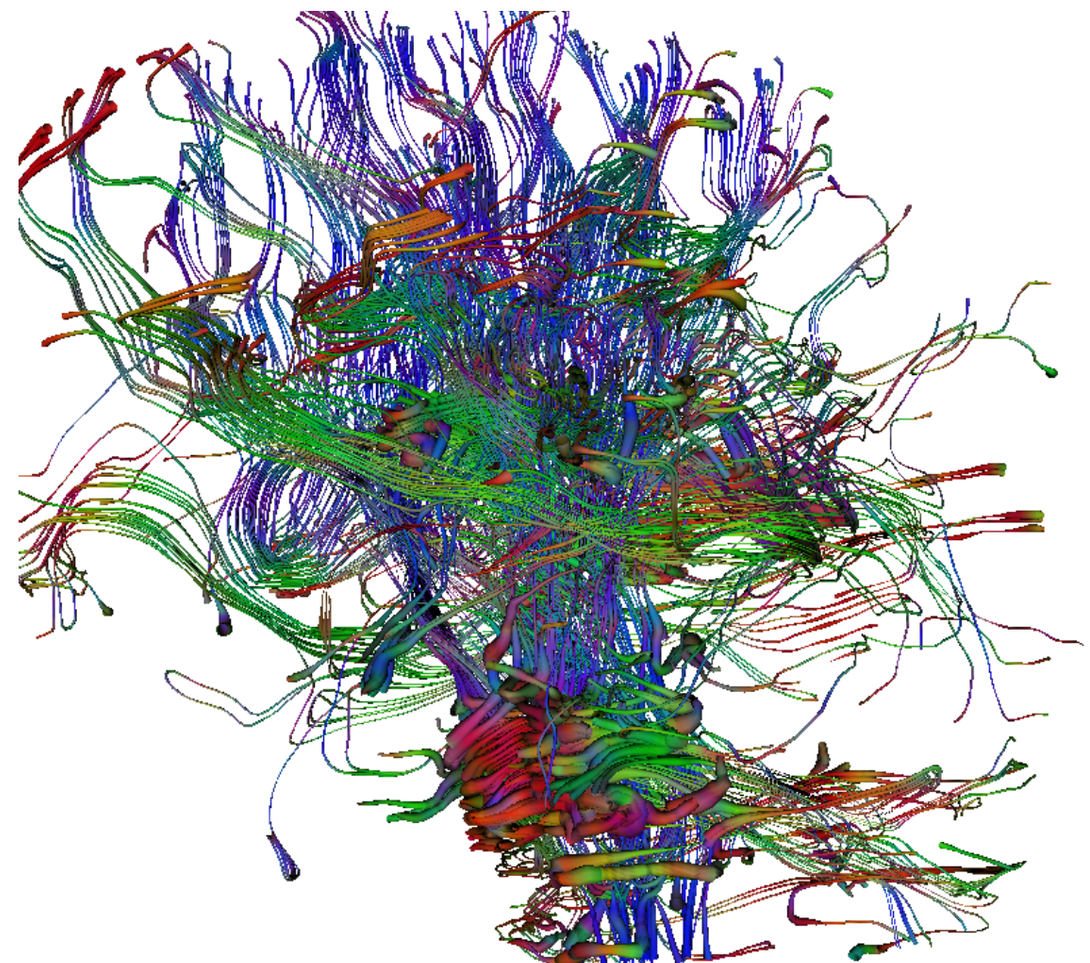
fiber tracking examples with different color codes

Hyper-Streamlines

- Stream tubes can be used for encoding eigenvalues
- Stream tubes are constructed in the major eigenvector field of e_1 (directions)
- At each point along each stream tube, we visualize the medium and minor eigenvectors e_2 and e_3 by sweeping an elliptic cross-section along e_1
- The two radii of the ellipses are then scaled by the medium and minor eigenvalues λ_2, λ_3
- These are called hyper-streamlines



construction of hyper-streamlines



example DT-MRI dataset visualized with hyper-streamlines

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

Thank You!