Visualization techniques

Juan Hernando Vieites
jhernando@fi.upm.es

ECAR 2012

1. Data representations
2. Scalar field visualization
   - Iso-surfaces
3. Volume rendering
   - Mathematical model
   - Sampling
   - Classification
   - Transfer functions
   - Compositing
   - Rendering algorithms
     - Image space algorithms
     - Object-space algorithms
4. Vector field visualization
   - Particle tracing methods
   - Line integral convolution
5. Tensor visualization

Data representation

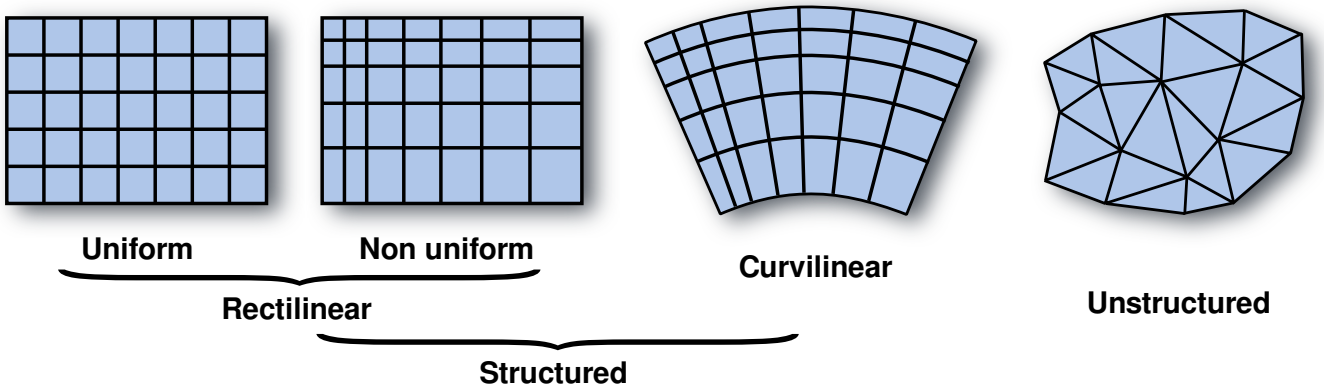
Data sets

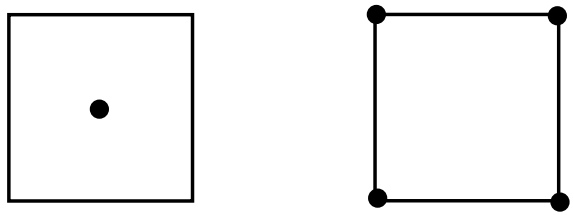
Data sets are comprised of two elements:

- Structure: The spatial organization of the data
  - Topology: properties invariant under most geometrical transformations (e.g. connectivity)
  - Geometry: An instantiation of a topology.

  A square is a topology, the locations of the corners is the geometry.
- Attributes: Data values associated to the structure
  - Scalars
  - Vectors
  - Normals
  - Tensors
  - Coordinates

Data representation





Example of cell oriented topologies

Algorithms for scalar field visualization

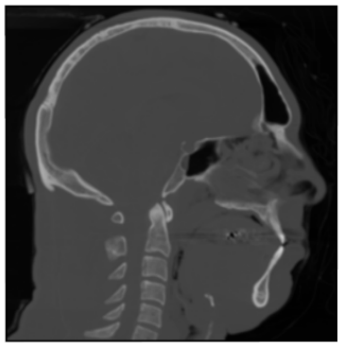
Basic algorithms

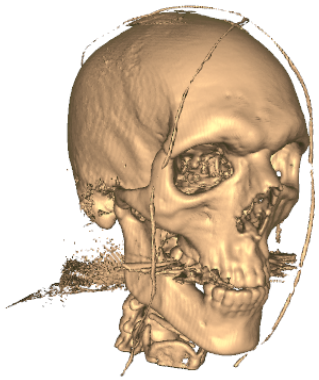
Basic scalar field visualization (for 3D fields) consist of reducing the dimensionality to 2D.

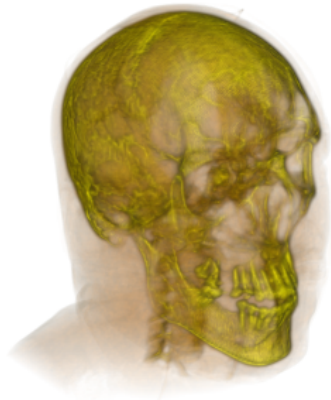
- Slicing
- Iso-surfacing

Direct volume rendering

Direct volume rendering refers to a family of techniques that render scalar fields as 3D geometrical objects.







Iso-surfacing

Iso-contours

- An iso-countour of a N-dimensional scalar field is a set of points that have the same value.
- In 2D, iso-contours are iso-lines. Examples from everyday life are height and isobar maps.
- In 3D, the iso-contours are known as iso-surfaces.

Types of algorithms

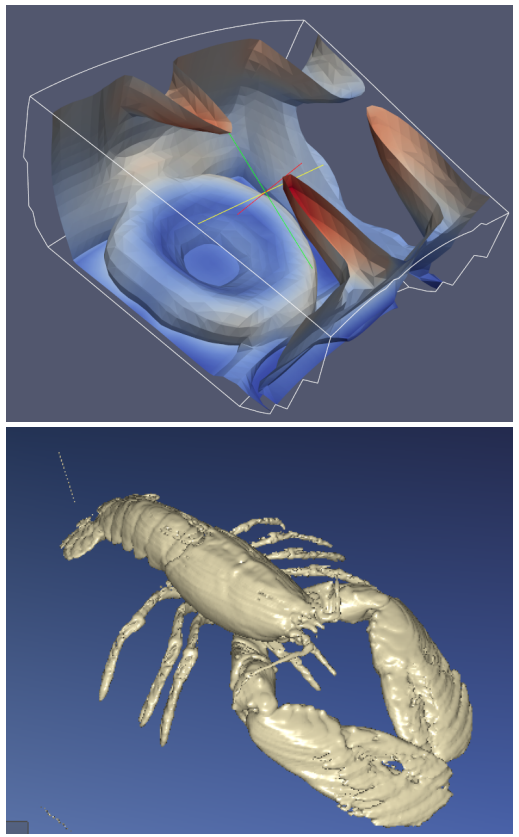
Iso-surfacing algorithms are mainly divided between:

- Object order algorithms
- Image order algorithms

Marching cubes

Marching cubes

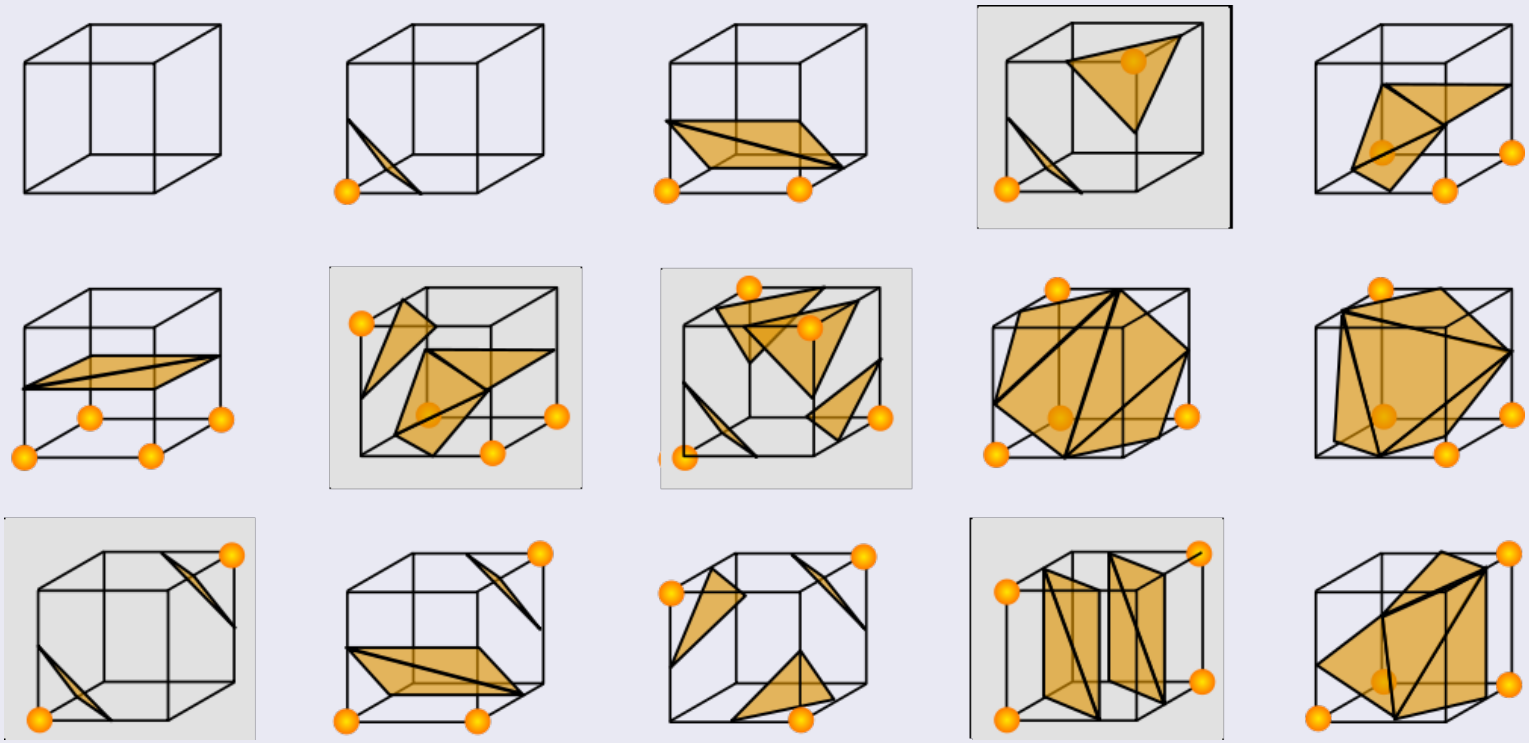
- Marching cubes is an iso-surfacing algorithm for cubic grids published and patented in 1987 [Lorensen:1987:MCH:37401.37422]
- In the original algorithm, each cell is processed independently
  1. The cell vertices are labelled as greater or smaller than the iso-surface value. 2^8 combinations are possible.
  2. A tessellation scheme is chosen based on the vertex labelling
  3. The triangle vertices are found by linear interpolation along the edges of the cell.



Marching cubes

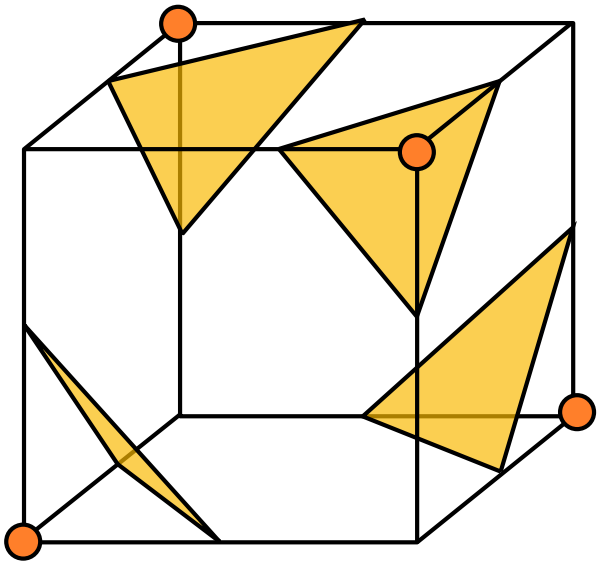
Possible tessellations

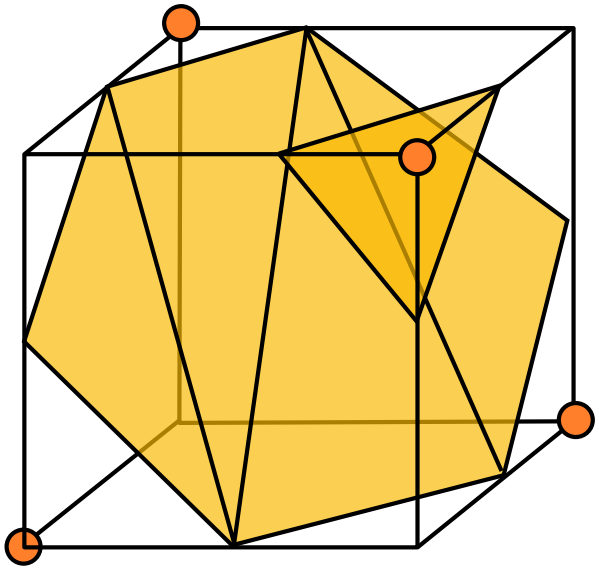
- Below are shown the 15 cases from the original publication.
- 29 cases are possible if equivalency is limited to rigid body rotations.



Marching cubes ambiguities

- 6 of the cases can have 2 tessellations (3, 6, 7, 10, 12 and 13).
- In the 2D equivalent the ambiguities are unimportant.
- In 3D the resulting mesh can present holes and cracks if the topology of the neighbouring cells is not considered.

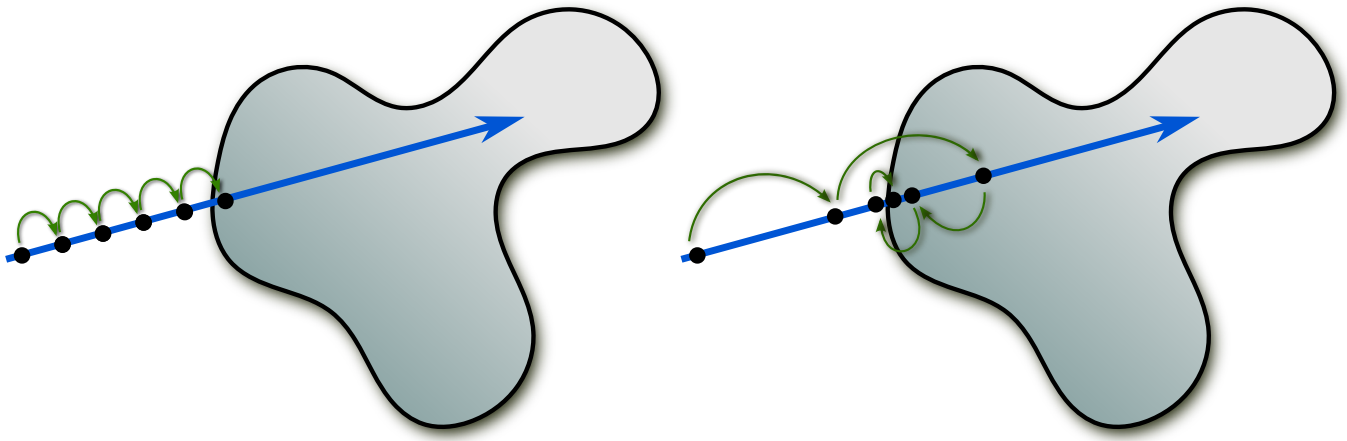




Ray-casting

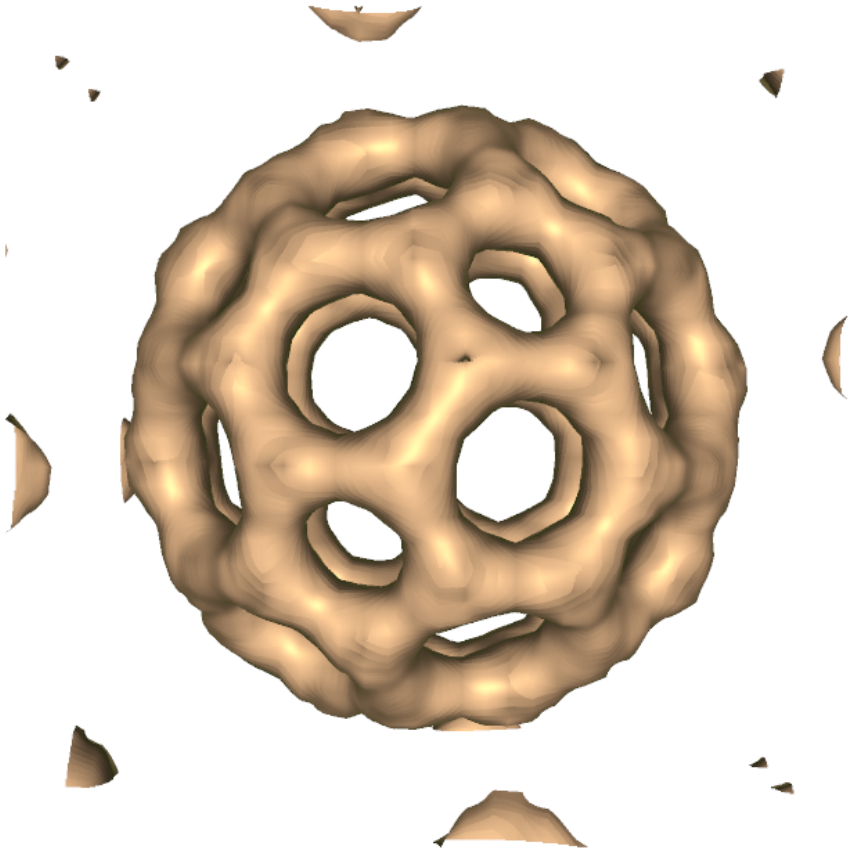
Algorithm description

- In ray-casting, a ray is casted from the camera through the center of each pixel
- As the ray traverses the volume at some step size, the volume is sampled to check whether the previous sample is at a different side of the iso-surface value than the current one.
- When an intersection is found, the gradient is computed to apply a lighting model.



Ray-casting

Example



Acceleration techniques for iso-contours

Cell span space

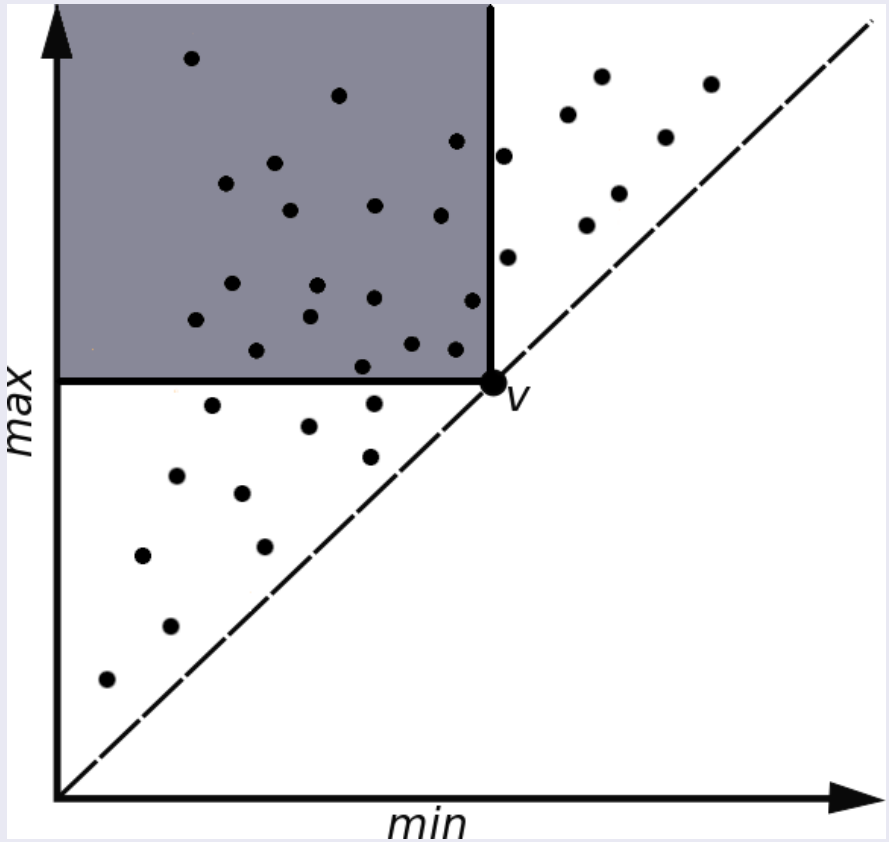
- The marching cubes algorithm is a brute force algorithms with regard to the number of cells processed.
- An octree where each non-leaf node contains the maximum and minimum values of the children can accelerate the search of intersected cells.
- However, octrees are not optimal due to unbalancing.

Span space

- The span space is a representation of intervals as 2D points.
- Spatial data structures for points can be used to speed up queries.

Acceleration techniques for iso-contours

Example of span space

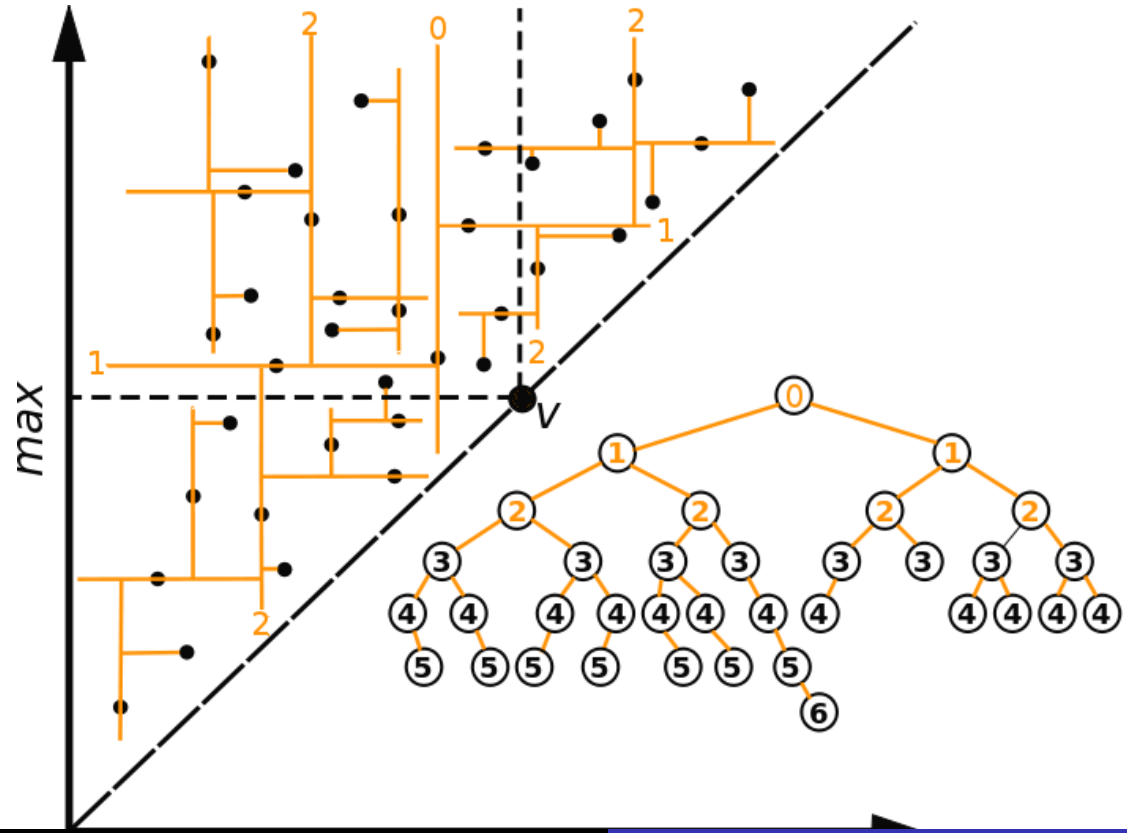


The grayed are area shows the candidate cells for an iso-value equal to V.

Span space algorithms

Example: k-d tree

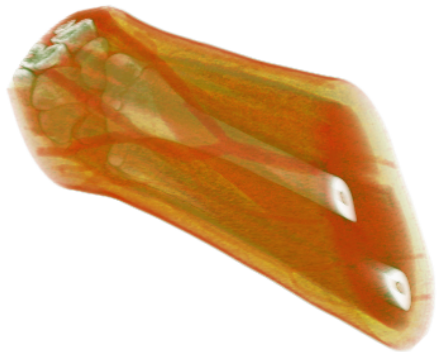
A k-d tree can be used to accelerate searches as proposed in the NOISE (Near optimal iso-surface extraction) algorithm [Livnat-1996]



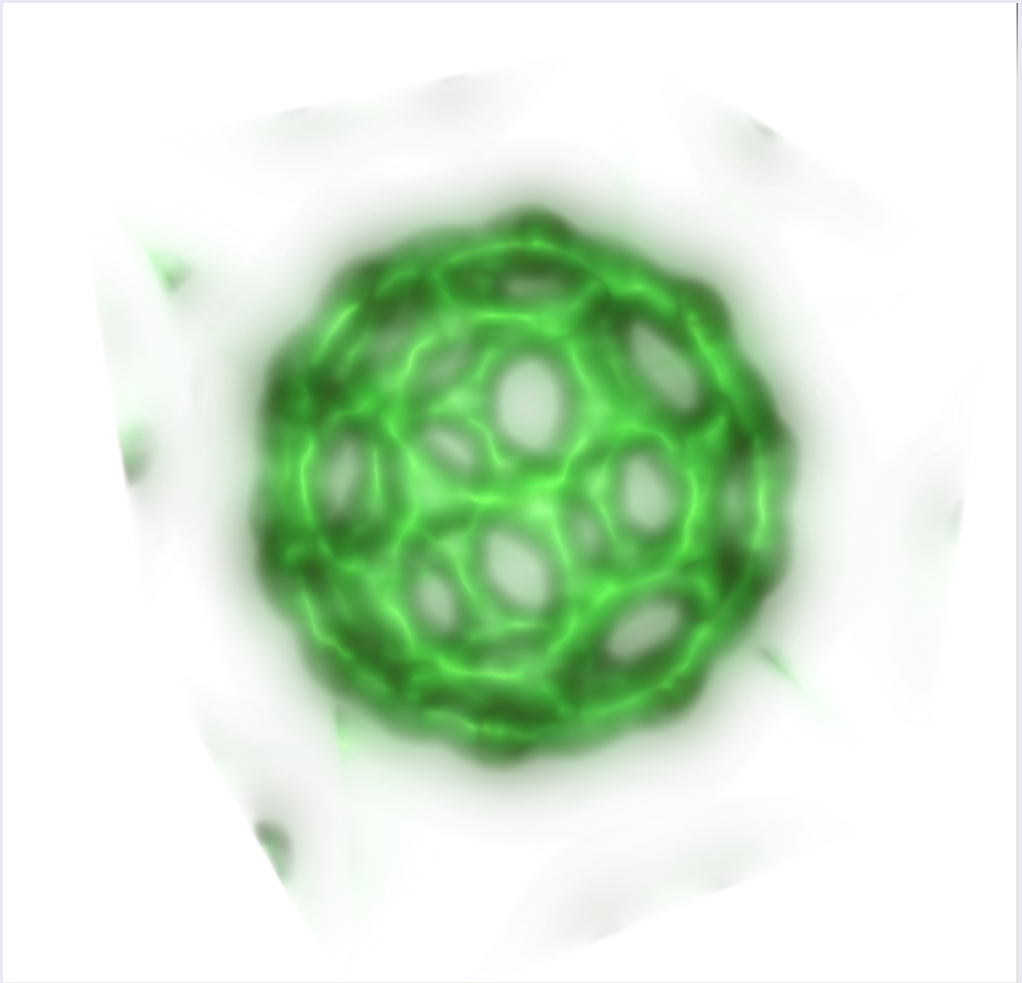
Direct volume rendering

Volume rendering

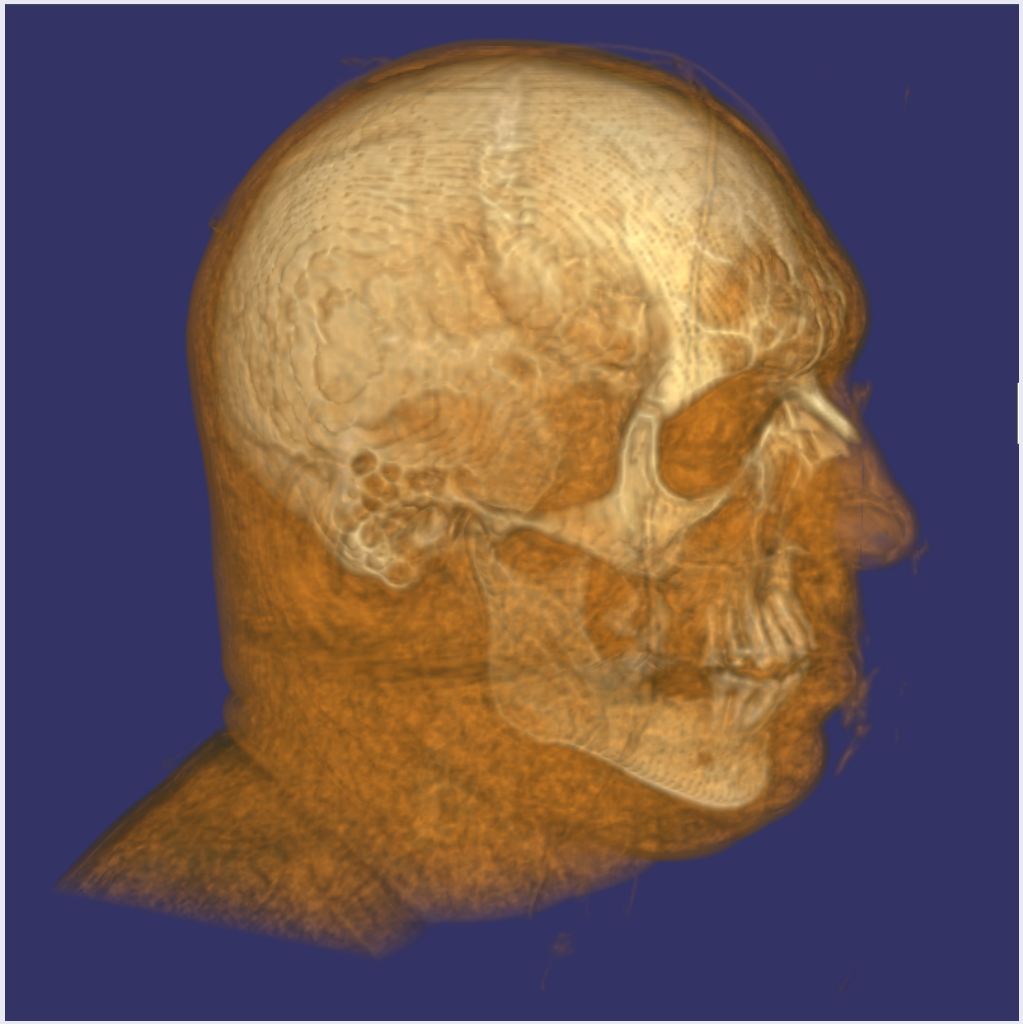
- Direct volume rendering is a visualization technique for scalar fields where all the data participates in the generation of the image.
- Adequate in cases when the iso-surfaces are uncertain, or larger vision of the data is required.
- Usually applied to simulation results obtained from CFD or FEM simulations as well as data obtained from scanners (CAT, MRI, fMRI).
- Applications in: Physics, Engineering, Medicine, and Computer Graphics in general.



Examples



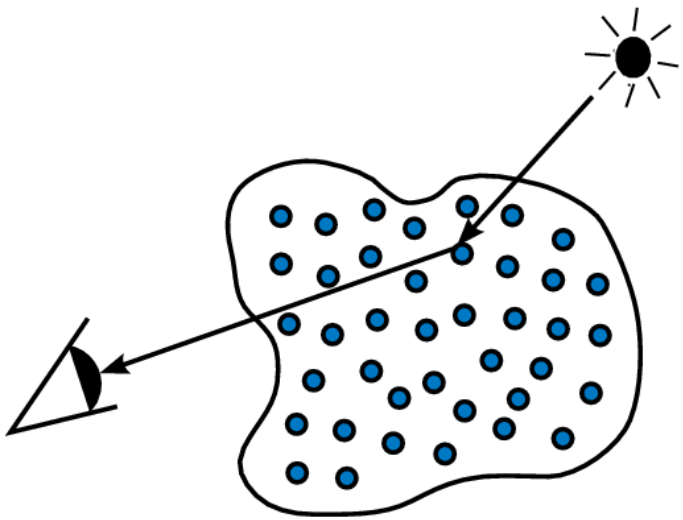
Examples



Optical model

Features of the common model

- A volume is assumed to be composed of particles with which the light interacts.
- The model considers: attenuation, emission and reflection.
- Each ray is considered to be reflected only once inside the volume.
- Multiple scattering or global illumination will not be introduced.



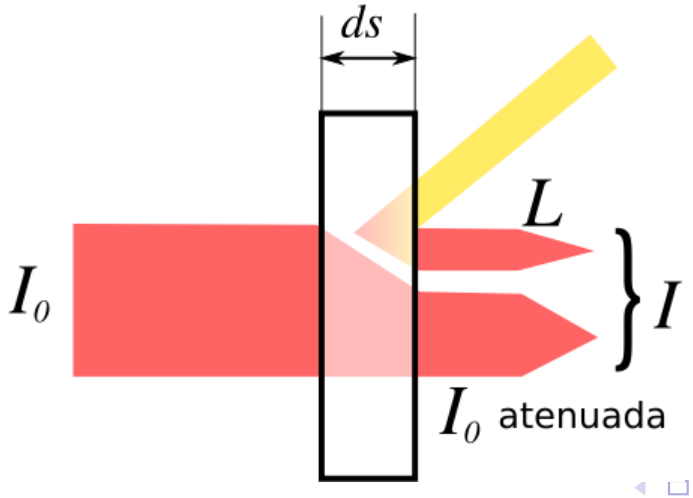
The volume integral equation

The volume integral

Proposed by Max [Max:1995:OMD:614258.614298], the volume integral is the solution of an ODE that states the attenuation, emission and reflection of light in a ray infinitesimal:

$$\frac{dI_\lambda(r)}{ds} = L(s) - \tau(s) I_\lambda(s; r)$$

where $I_0 = 0$.



The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Where:

- L is the ray length
- $C_\lambda(s)$ is a function that computes the outgoing luminance in s due to emission or reflection from a light source.
- τs is the light attenuation coefficient at point s.

The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Numerical form

The analytic equation is solved approximately by a series.

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s) \exp\left(-\int_0^s \tau(t)dt\right) ds$$

The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Numerical form

$$I_\lambda(r) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\tau(i\Delta s)\Delta s \exp\left(-\sum_{j=0}^{i-1}\tau(j\Delta s)\Delta s\right)$$

Transforming the summation in a product series...

The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Numerical form

$$I_\lambda(r) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\tau(i\Delta s)\Delta s \prod_{j=0}^{i-1} \exp(-\tau(j\Delta s)\Delta s)$$

The transparency is $t(i\Delta s) = 1 - \alpha(i\Delta s) = \exp(-\tau(i\Delta s)\Delta s)$. Approximating the exponential at 0 by its two first Taylor series terms it results: $t(i\Delta s) \approx 1 - \tau(i\Delta s)\Delta s$, so $\tau(i\Delta s)\Delta s \approx \alpha(i\Delta s)$. Finally ...

The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Numerical form

$$I_\lambda(r) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\alpha(i\Delta s)\prod_{j=0}^{i-1}(1-\alpha(j\Delta s))$$

This closed form formula can be computed recursively by two different expressions:

The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Numerical form

$$I_\lambda(r) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\alpha(i\Delta s)\prod_{j=0}^{i-1}(1-\alpha(j\Delta s))$$

This closed form formula can be computed recursively by two different expressions:

Front to back composition:

$$c_i = C_\lambda(i\Delta s)\alpha(i\Delta s)(1-\alpha_{i-1}) + c_{i-1}$$
$$\alpha_i = \alpha(i\Delta s)(1-\alpha_{i-1}) + \alpha_{i-1}$$

The volume integral equation

The volume integral

$$I_\lambda(r) = \int_0^L C_\lambda(s)\tau(s)\ \exp\left(-\int_0^s \tau(t)dt\right) ds$$

Numerical form

$$I_\lambda(r) = \sum_{i=0}^{L/\Delta s-1} C_\lambda(i\Delta s)\alpha(i\Delta s)\prod_{j=0}^{i-1}(1-\alpha(j\Delta s))$$

This closed form formula can be computed recursively by two different expressions:

And back to front composition:

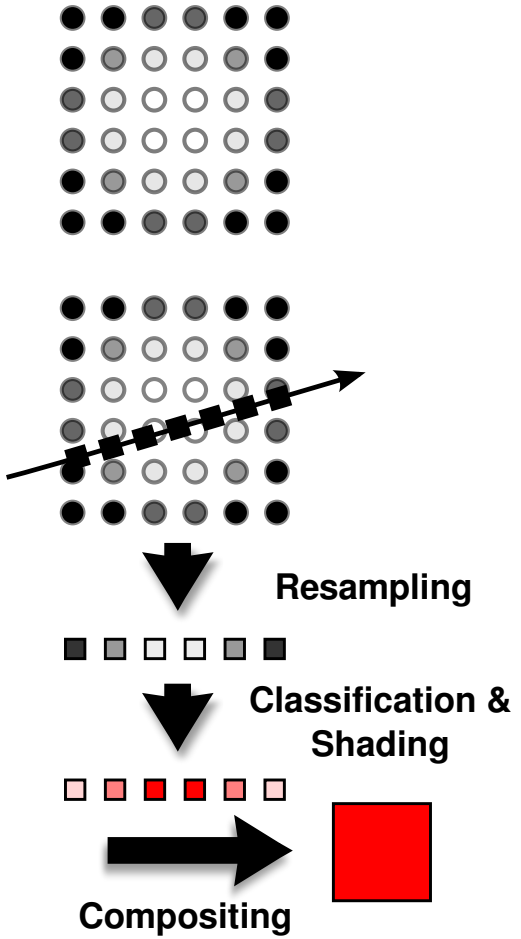
$$c_i = c_{i-1}(1-\alpha(i\Delta s)) + C_\lambda(i\Delta s)\alpha(i\Delta s)$$
$$\alpha_i = \alpha_{i-1}(1-\alpha(i\Delta s)) + \alpha(i\Delta s)$$

Volume rendering pipeline

Rendering stages

In general, direct volume rendering algorithms consist of these steps:

1. Pre-filtering: data sub-setting, resizing, ...
2. Rendering:
   - Resampling: Taking values, gradients from the volume data set
   - Mapping: Colors and opacities from values (classification)
   - Shading: Applying the illumination model
   - Compositing: Blending the shaded samples
3. Display



Sampling - reconstruction

The volume function

- The volume data set v is a discrete sampling of a continuous signal f ($v : V -> \mathbb{R}$ where $V \subset \mathbb{R}^3, |V| \in \mathbb{N}$).
- Finding a value outside the sampling domain implies a reconstruction of the original signal.

The Sampling Theorem

- As in the case of anti-aliasing, the Nyquist theorem tells us that the maximum frequency that can be reconstructed is half the sampling rate.
- This is very important during data acquisition, but also for rendering algorithms: Many rendering algorithms resample the function v.

Sampling - reconstruction

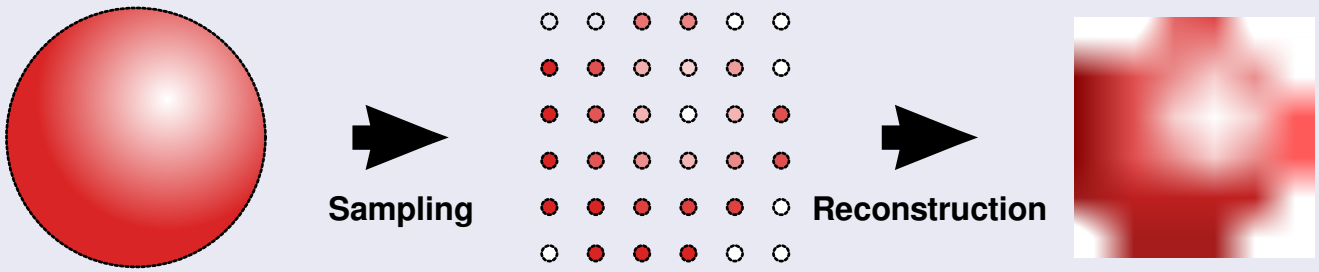
Reconstruction

- Mathematically speaking, reconstructing a signal from a sampling is computing the convolution of a filter with the sampling:

$$f'(\boldsymbol{x}) = \sum_{s} V(\boldsymbol{s}) h(\boldsymbol{x} - \boldsymbol{s})$$

  where h is the reconstruction filter and $\boldsymbol{s} \in V$ are the samples

- In practice, the reconstruction implies interpolation between the samples of V.

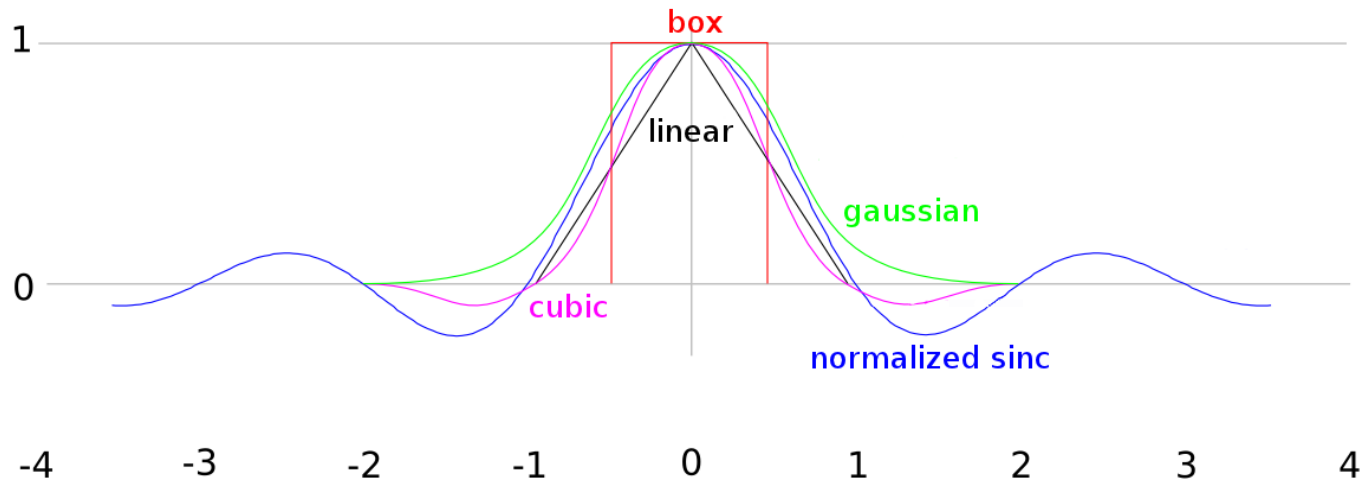


Original signal | Volume dataset | Reconstructed signal

Filters

Types

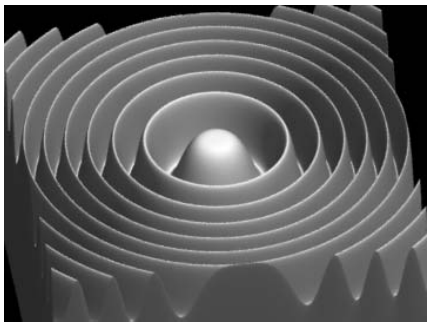
- The ideal filter is the sinc function. However it is unpractical because it has infinite domain.
- Other common filters are: box, linear, cubic, Gaussian, ...
- Gradients need also to be reconstructed for shading. Filters for gradients include: central differences, Sobel, ...



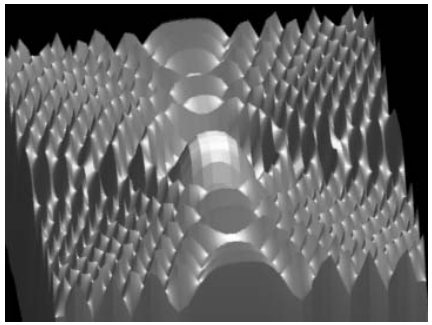
Filters

Features

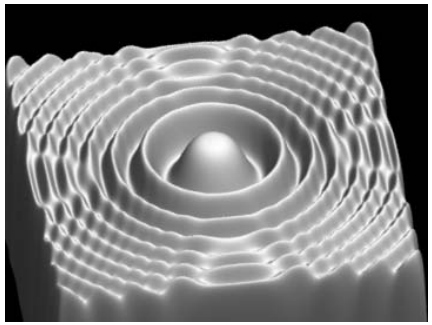
- Different filters have different trade-offs between:
  - Computational cost
  - Blurring
  - Oscillation
- **Radially symmetric filters** are very used in high quality rendering and some types of algorithms.
- GPUs implement **linear filters**



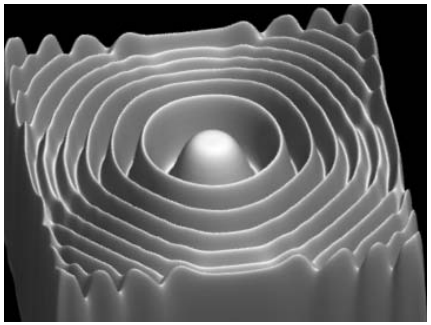
Original



Trilinear



B-spline



Windowed sinc

Images from [Marschner-1994]

Classification

The transfer function

- The formulation of the volume integral refers to two functions, the sample opacity ($alpha$) and the color (C_λ).
- These functions depend on the sample value as well as the gradient if some illumination model is applied.
- Classification is the process of mapping the sample to an alpha an color.
- The function that translate a sample into a pre-shaded color and opacity is called **transfer functions**.
- A transfer function function is typically a single variable function, but higher order transfer functions also exist (extra variables could be gradient, principle curvature, . . .)

Classification

Types or classification

Depending on the order between classification and resampling we have:

- **Pre-classification**: Original and samples are mapped and color and opacities are interpolated during resampling.
- **Post-classification**: Resampling occurs after classification.

Pre-classification

Features

- Classification goes before resampling.
- A color and opacity is assigned to each sample. Resampling interpolates RGBA values.

Pros

- Easy to implement and fast.
- Gradient need only to be interpolated at cell corners.
- The classification can be precomputed.

Cons

- High-frequency details in the transfer function are lost
- Lighting is less accurate (remember Gouraud vs. Phong)
- Magnification causes blurring.

Post-classification

Features

First, the sample value is computed and then the transfer function and shading are applied.

Pros

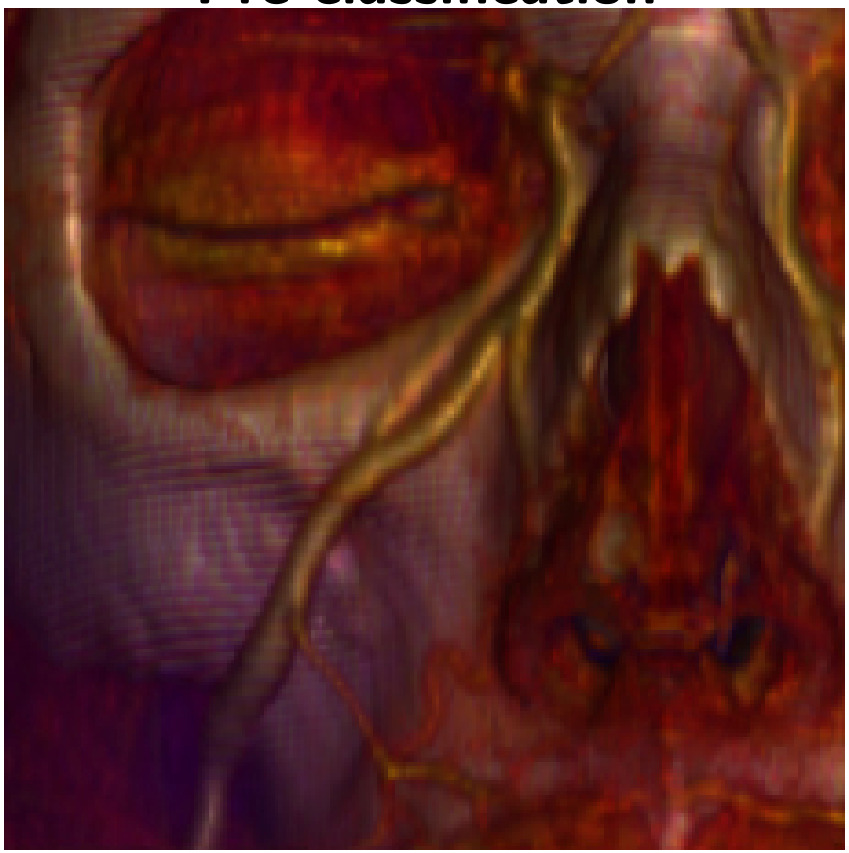
- Higher fidelity transfer function reproduction.
- Per sample shading possible.

Cons

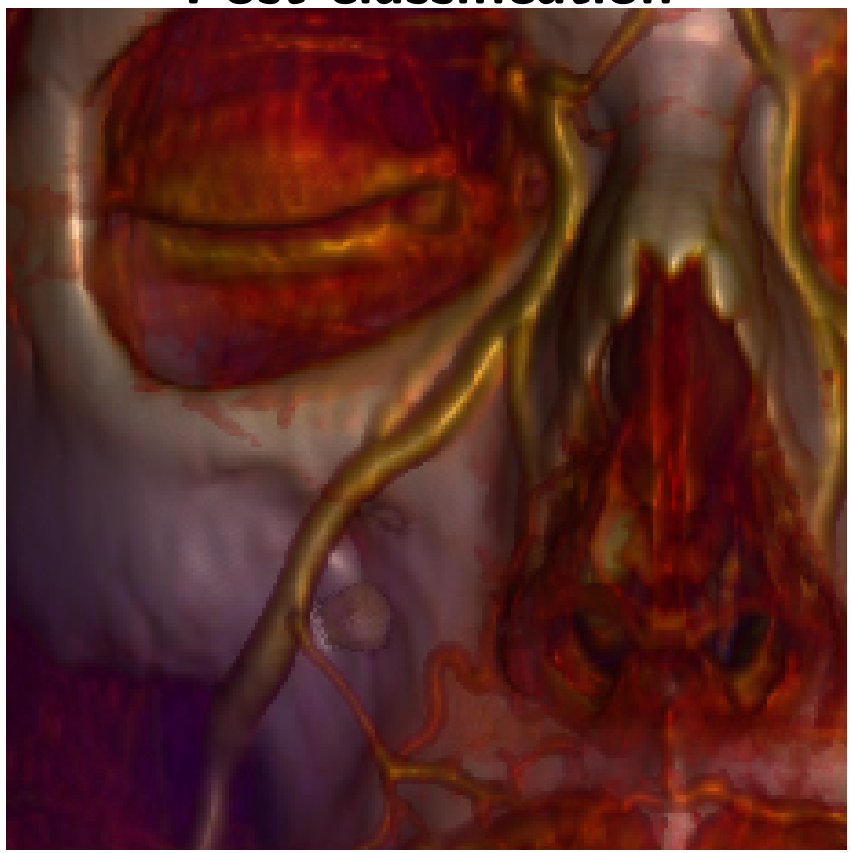
- Higher computation cost because the transfer function and shading cannot be precomputed.

Post-classification vs pre-classification

Pre-classification



Post-classification



Transfer functions

from the volume integral the transfer functions is from value to primary colors and opacities, the formula is then

Multidimensional transfer functions

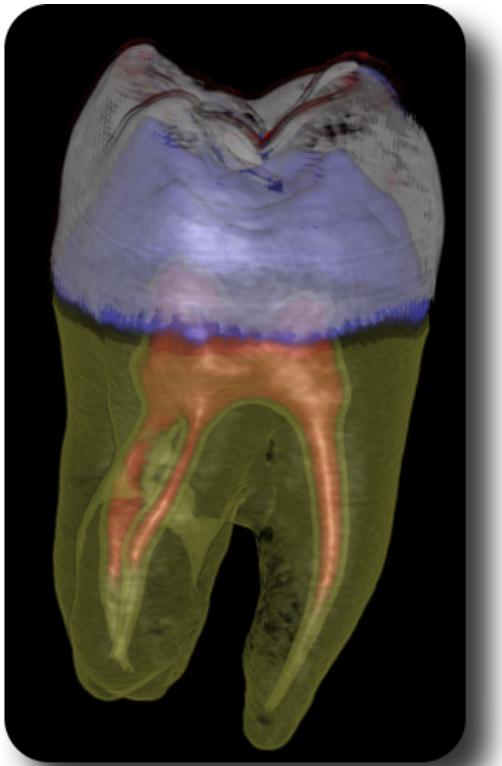
Using only the value of the volume to distinguish materials is very limited.

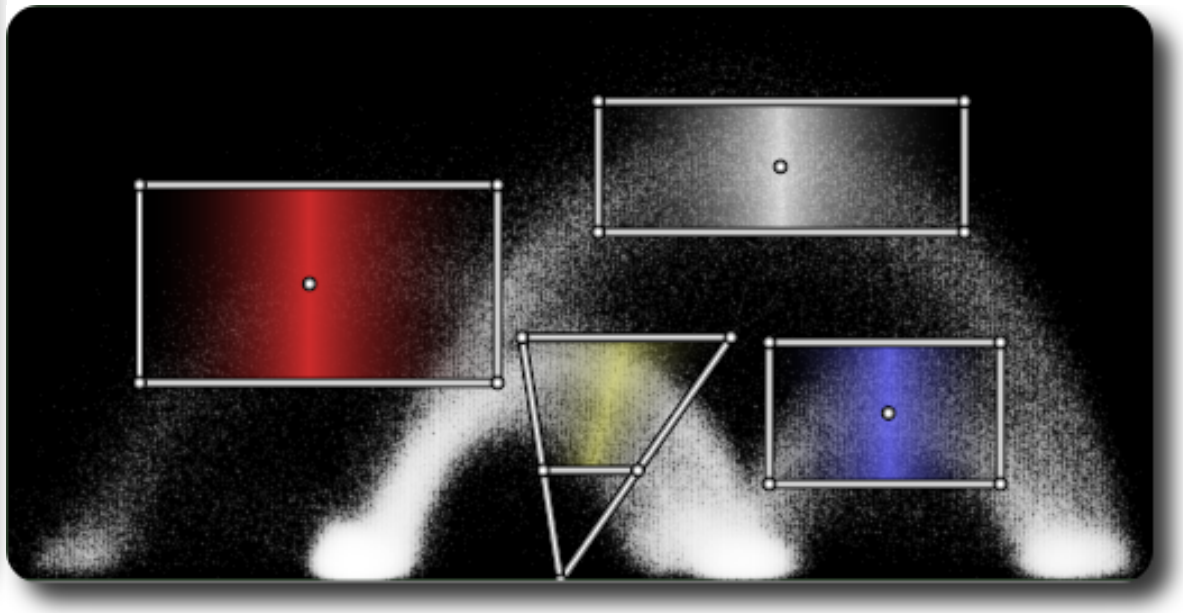
Multidimensional functions

- More elaborate classifications are possible if we provide more information about the sampled point to the transfer.
- The first and second derivatives are straight forward to derive and can be used for this purpose.
- The first derivative help distinguish interfaces between different materials.
- The second derivative can help separate materials based on the curvature of the iso-surfaces.
- As a downside, the task of creating a meaningful transfer function for a data set becomes more complex.

Multidimensional transfer functions

[Kniss:2002:MTF:614287.614529] presented the extension of transfer functions to use the gradient for classification. They also proposed a widget for intuitive edition of the 2D function.





Pre-integrated transfer functions

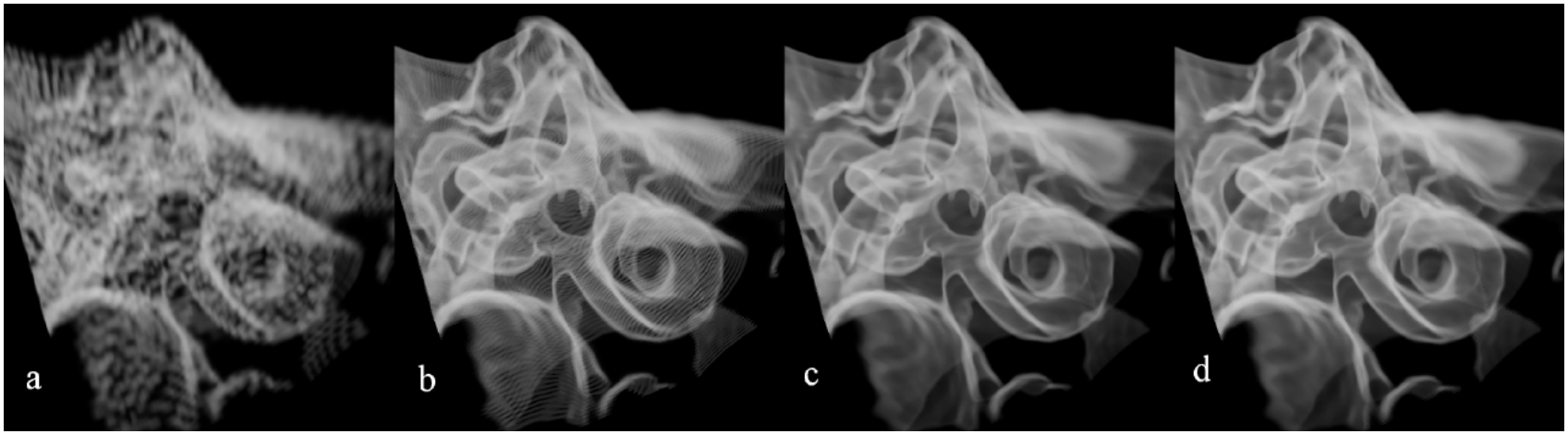
Motivation

- High frequency features in the transfer function can be easily missed if the sampling rate is not high enough.
- However if the volume doesn't change fast enough, there is no need for higher sampling rate.

Pre-integrated classification

- The idea is to separate the integration of volume samples from the transfer function.
- For a ray segment, the classification can be pre-integrated along that segment using the start and end values and the segment length.
- The pre-integration can be used to interpolate the classification for arbitrary parameter values.

Pre-integrated transfer functions



Images from [Engel-2001]

a) pre-shaded b) post-shaded
c) post-shaded with extra slices d) pre-integrated

Compositing

Alpha channel

- RGBA colors are RGB tuples extended with an alpha channel to specify the opacity of the color.
- There are two types of RGBA colors: with or without multiplied alpha: *e.g. pure red with 50% opacity: (1, 0, 0, 0.5) normal, (0.5, 0, 0, 0.5) premultiplied*.

Back-to-front and front-to-back alpha blending

Begin C_1 is in front of C_2.

- Back-to-front
  $C_1 \oplus C_2 = C_1 * \alpha_1 + C_1 * (1 - \alpha_1)$,
  $\alpha_1 \oplus \alpha_2 = \alpha_1 + (1 - \alpha_1) * \alpha_2$
- Front-to-back (with pre-multiplied alpha)
  $C_1 \oplus C_2 = C_1 + C_2 * (1 - \alpha_1)$,
  $\alpha_1 \oplus \alpha_2 = \alpha_1 + (1 - \alpha_1) * \alpha_2$

Compositing

Alpha channel

- RGBA colors are RGB tuples extended with an alpha channel to specify the opacity of the color.
- There are two types of RGBA colors: with or without multiplied alpha: *e.g. pure red with 50% opacity: (1, 0, 0, 0.5) normal, (0.5, 0, 0, 0.5) premultiplied*.

Back-to-front and front-to-back alpha blending

Begin C_1 is in front of C_2.

- **Non commutative operations!**
- But front to back is associative

Alpha correction

Integral discretization

- Given a transfer function, the final pixel color depends on the step size used during integration.
- The alpha value can be adjusted to account for the fact.

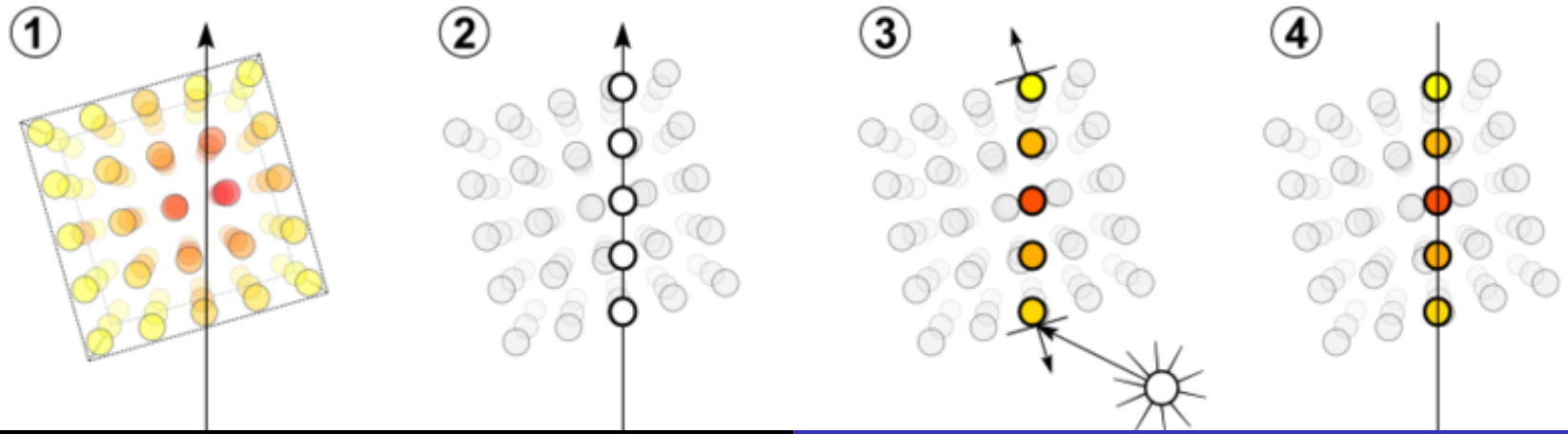
Adjusting the alpha value

Being α the original alpha values to use for a ray marching of step w, for a new step w', the formula to correct the alpha value is:

$$\alpha' = 1 - (1 - \alpha)^{\frac{w}{w'}}$$

Ray casting

- The final color is computed independently for each pixel.
- Similarly to the original problem formulation, for each pixel, a ray is cast from the camera through the volume passing by the pixel center.
- The volume integral is solved by sampling and integrating along the ray.
- In current GPUs, the implementation is straightforward (solving the integral in the fragment shader).



Aliasing

Problem

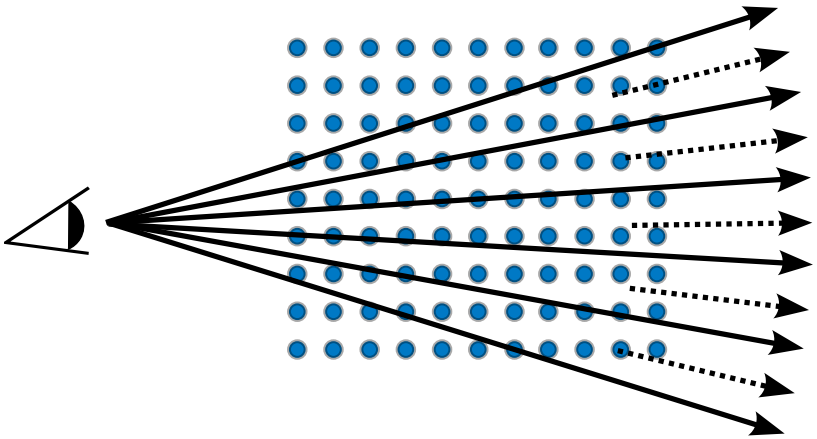
- Divergent rays cause subsampling as the ray moves away from the camera.
- These subsampling can cause aliasing if the the volume has high frequency features.

Aliasing

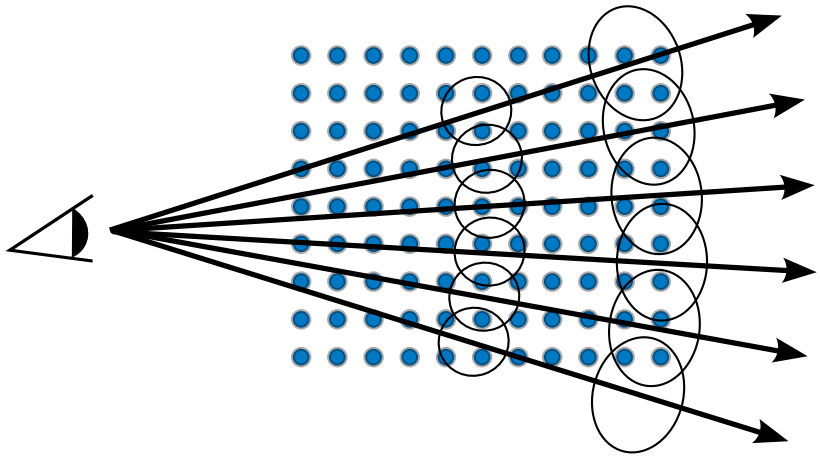
Solutions

Combining reconstructions with low-pass filtering.

- EWA (Elliptical weighted average)
- Ray splitting can combination.
- Pre-processing and for multi-resolution.



Ray division

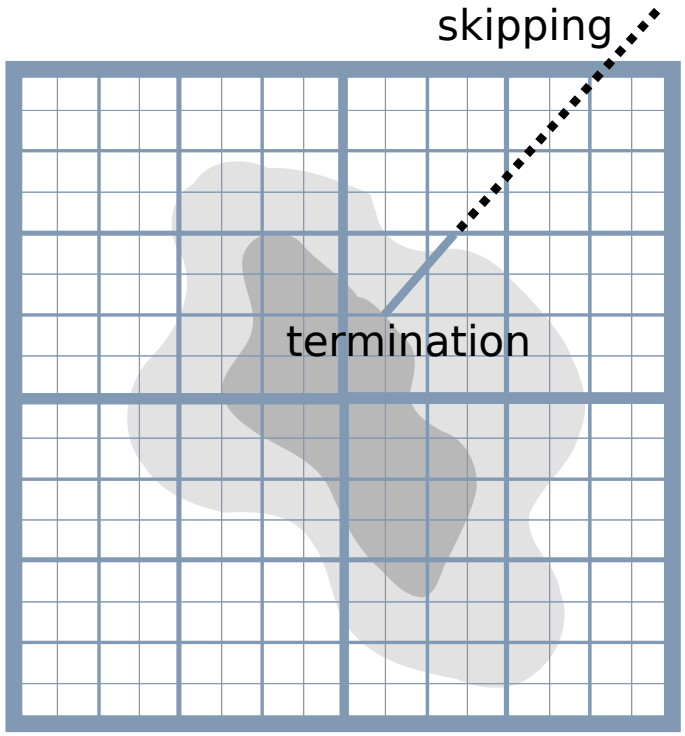


EWA-filtering

Improvements

Examples

- Early ray termination: detect when the opacity has accumulate above a given threshold.
- Skipping empty space: using spatial partitions (*octrees*, *kd-trees*) based on the opacity of the samples.
- Hierarchical subsampling for multi-resolution and larger data sets [].



Object space algorithms. Introduction

Features

- The volume integral is not computed explicitly.
- Instead, the volume is reconstructed and sampled by independent intervals.

Categories

- Texture mapping based
- Kernel splatting
- Tetrahedral

Splatting

Splatting

Splatting consist of center a filter kernel in each sample and integrate the kernels into the final image in some way.

Filter kernels

- Given $f'(\boldsymbol{x}) = \sum_s v(\boldsymbol{s})h(\boldsymbol{x} - \boldsymbol{s})$, integrating $f'(x, y, x)$ along a view ray can be refactorized as

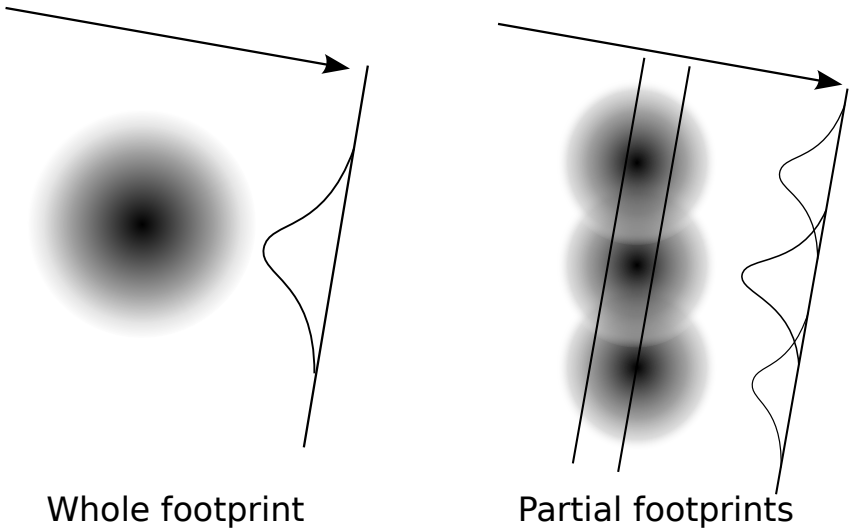
$$\sum_s v(\boldsymbol{s}) \int_0^L h(\boldsymbol{x}(t))dt$$

- The integral of the kernel k projected into a plane is the kernel footprint.
- Kernel footprint are unique in orthographic projections but not in perspective ones.

Kernel splatting

Types of splatting

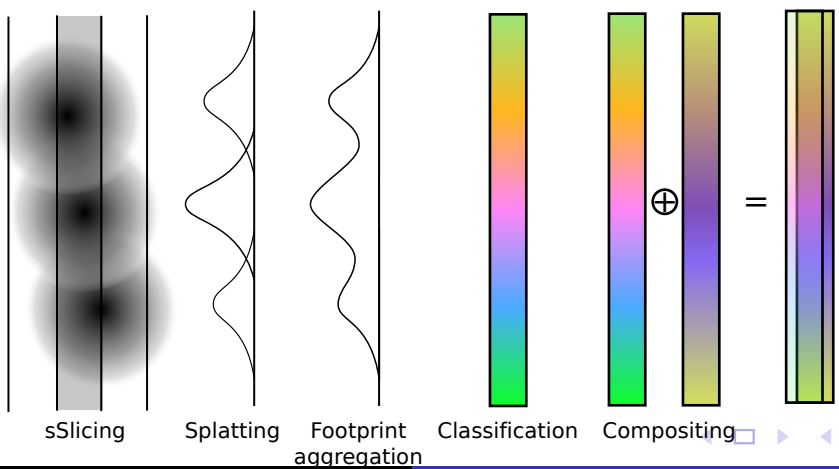
- **Whole splatting**: The whole kernel is integrated along the splatting axis. Only really suitable for iso-surface rendering or maximum intensity projection.
- **Partial spatting**: The filter is integrated in the interval delimited by two slices (slices are to be composited).



Kernel splatting, rendering

Steps

1. The volume is sliced in several screen-aligned slices
2. Slices are processed back to front.
3. For every sample contributing to the slice, the filter kernel centered at its position is splatted and the footprint added to the current slices (kernel footprints could be precalculated).
4. The values in the slice texture are classified and shaded
5. La slice is composed in the buffer where the final image is accumulated.



Texture-mapping based techniques

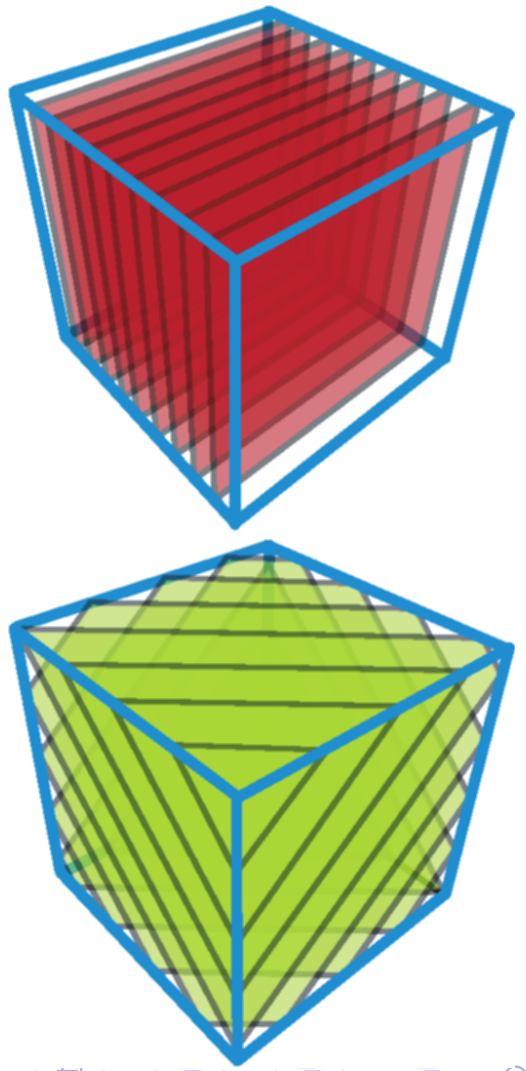
Features

- Similar to slice-based kernel splatting.
- Can take advantage of texture filtering units hardware from consumer level GPUs
- Unstructured volume data sets only supported if resampled into a 3D texture.
- In the basic formulation these techniques are faster but have worse quality.
- The GPU memory limits the size of the biggest volume that can be easily handled.
- May suffer from aliasing in perspective projection.
- Other filters different to the trilinear on can be used but that implies moving texture filtering from the texture units to the GPU cores

Techniques based on hardware texture mapping

Description

- The volume is sliced with screen or axis-aligned slices
- For each slice, a polygon is generated together with its texture coordinates.
- The geometry is dispatched to the GPU in back to front or front to back order.
- The GPU performs bilinear (axis-aligned) or trilinear (screen-aligned) interpolation of the volume, classifies, shades and composites the samples.



More advanced topics

Topics

- Advanced illumination models for global illumination, and participating media (multiple scattering). Still difficult in real-time.
- Illustrative visualization (non-photorealistic rendering).
- Semi/automatic transfer function generation.
- Very large data sets.

Where to start

- `http://www.cs.ucdavis.edu/~ma/VolVis`
- State of the art report in interactive volumetric illumination [Jonsoon-2012]

Vector fields

Definition

- Vector fields are scalar fields with vectors.
- A vector field v is a mapping:

$$\mathbb{S} \rightarrow \mathbb{R}^n$$

where $\mathbb{S}$ is subspace of a Euclidean space.

- Let $u(t)$ be a trajectory in $\mathbb{S}$, The field v can be treated as a derivative of a trajectory:

$$\frac{d\boldsymbol{x}}{dt} = v(\boldsymbol{x})$$

Vector fields

Types

- Steady (the definition above)
- Time-varying: The mapping changes over time: $v : \mathbb{S} \times \mathbb{R} \rightarrow \mathbb{R}^n$

Underlying representations

- Structured grids
- Unstructured grids
- Adaptive Mesh Refinement (AMR)

Types of visualization methods

- Glyphs
- Preprocessing and volume rendering
- **Particle advection** methods
- **Texture based** methods (Line Integral Convolution, LIC)
- Topology based methods
- Lagrangian based: Finite-time Lyapunov exponent

Glyphs

- Straight forward representation in a simple object is used to show the direction and magnitude of the field at a discrete set of location.
- Provides some information in 2D but in 3D the images become quickly cluttered.

(Stream,path,time,streak)-lines

Basic principles

- All these methods are based on line integrals.
- They show the path followed by weightless particles inside the field.
- The path $u(t)$ is computed by solving a line integral.
- And rendered with lines or with generalized cylinders.
- The integration needs a seed point.

(Stream,path,time,streak)-lines

Streamlines

- Streamlines show the trajectory followed by particles assuming the filed is steady:

$$u(t) = \int_0^t v(u(t))dt \qquad u(t_0) = \boldsymbol{x_0}$$

- Useful for static fields, and instant velocity or steady state in time varying fields.

Pathlines

- A pathline shows the trajectory of a single particle when time is considered.

$$u(t) = \int_0^t v(u(t), t)dt \qquad u(t_0) = \boldsymbol{x_0}$$

(Stream,path,time,streak)-lines

Streaklines

- Show the trajectories of particles as they are continuously dropped inside the field. Like ink spilled on the water or smoke in the air.

$$u(t,s) = \int_0^s v(u(t,s),t)ds \qquad u(t_0) = \boldsymbol{x_0}$$

Timelines

- Show the trajectories of particles seeded at the same moment along a continuous line.

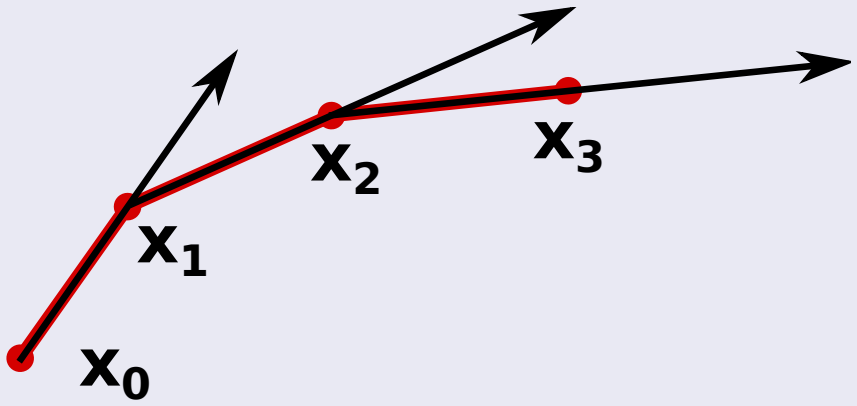
Numerical integration

Euler method

Simple and intuitive iterative solution of the line integral. Fast, but inaccurate and unstable.

$$\begin{aligned} \boldsymbol{x}_{i+i} &= \boldsymbol{x}_i + v(\boldsymbol{x}_i, t_i) * \Delta_t \\ t_{i+1} &= t_i + \Delta_t \end{aligned}$$

where $\boldsymbol{x}_0$ is the seed point.

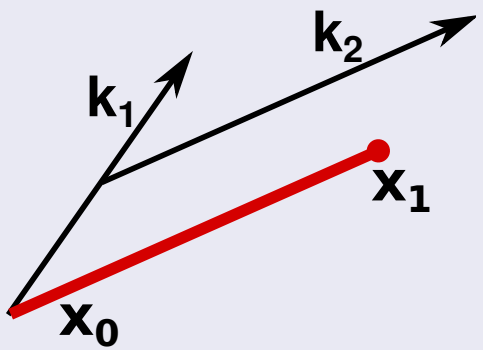


Numerical integration

Runge-Kutta

- Runge-Kutta are a family of methods for solving differential equations with better stability than the Euler method.
- Runge-Kutta of order 2 is stated as:

$$
\begin{aligned}
\boldsymbol{x}_{i+i} &= \Delta_t \boldsymbol{x}_i + \Delta_t \boldsymbol{k}_2 \\
\boldsymbol{k}_1 &= v(\boldsymbol{x}_i, t_i) \\
\boldsymbol{k}_2 &= v(\boldsymbol{x}_i + \tfrac{1}{2} k_1, t_n + \tfrac{1}{2}\Delta_t) \\
t_{i+1} &= t_i + \Delta_t
\end{aligned}
$$



Numerical integration

Runge-Kutta

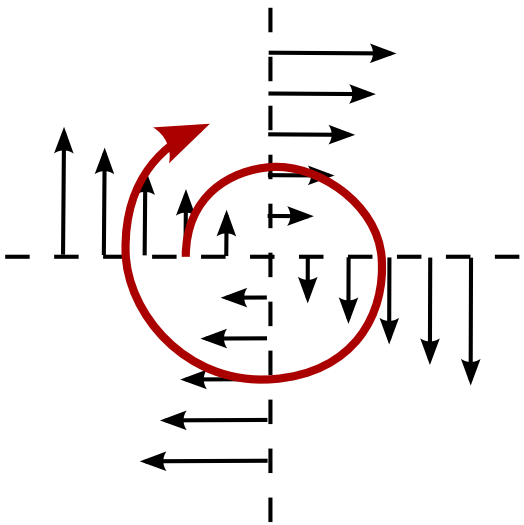
- Runge-Kutta are a family of methods for solving differential equations with better stability than the Euler method.
- Runge-Kutta of order 4 is very common. Its formulation is:

$$
\begin{aligned}
\boldsymbol{x}_{i+i} &= \boldsymbol{x}_i + \tfrac{1}{6}\Delta_t(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4) \\
\boldsymbol{k}_1 &= v(\boldsymbol{x}_i, t_i) \\
\boldsymbol{k}_2 &= v(\boldsymbol{x}_i + \tfrac{1}{2}\Delta_t\boldsymbol{k}_1, t_i + \tfrac{1}{2})\Delta_t \\
\boldsymbol{k}_3 &= v(\boldsymbol{x}_i + \tfrac{1}{2}\Delta_t\boldsymbol{k}_2, t_i + \tfrac{1}{2})\Delta_t \\
\boldsymbol{k}_4 &= v(\boldsymbol{x}_i + \Delta_t\boldsymbol{k}_3, t_i + \tfrac{1}{2})\Delta_t \\
t_{i+1} &= t_i + \Delta_t
\end{aligned}
$$

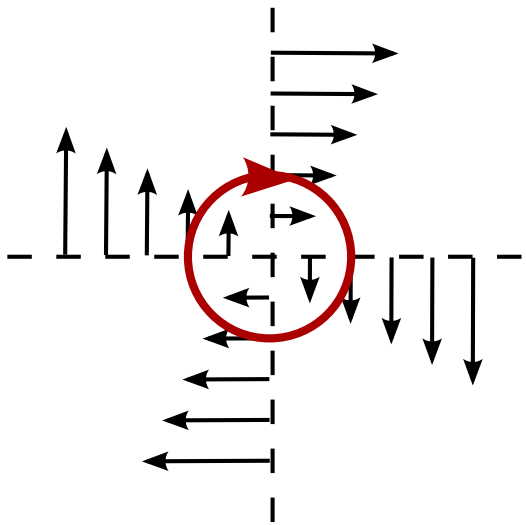
Numerical Integration

Euler vs. Runge-Kutta

- The Euler method is less stable than Runge-Kutta.
- The integration result can diverge from the analytical solution.
- Runge-Kutta is more complex but a larger step size can be used



Euler

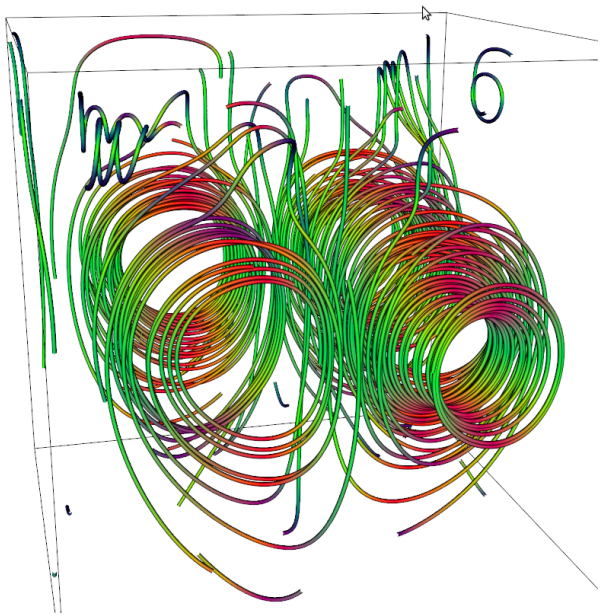


Runge-kutta

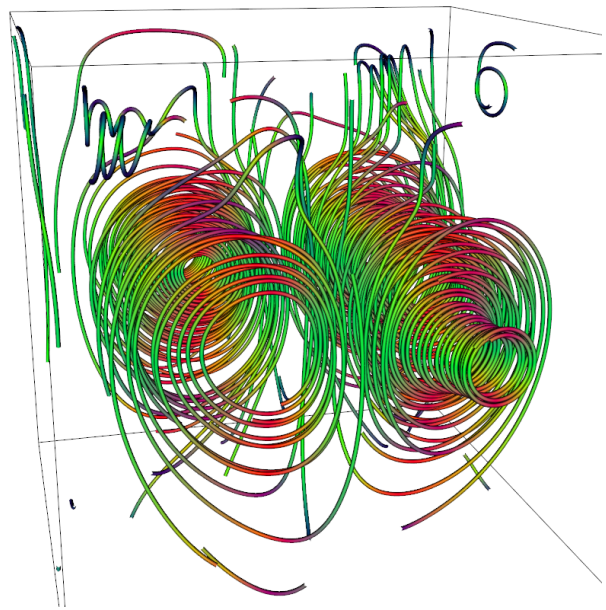
Numerical Integration

Euler vs. Runge-Kutta

- The Euler method is less stable than Runge-Kutta.
- The integration result can diverge from the analytical solution.
- Runge-Kutta is more complex but a larger step size can be used



Euler

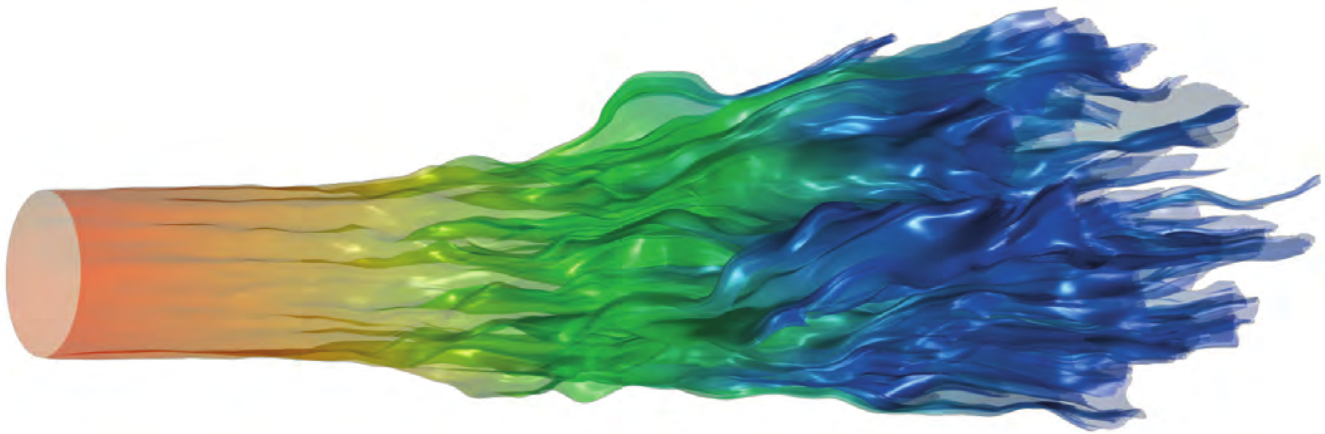


Runge-Kutta

Extension to surfaces

Motivation

- A visualization of trajectories become quickly cluttered as the number of paths shown increases.
- If animated, they are difficult to follow.
- Instead of computing the advection of separated points, we can use a continuous line as seed to obtain a surface



Credit: VACET, SciDAC Visualization and Analytics Center for Enabling Technologies

Extension to surfaces

Benefits

- Better depth and shape perception can be provided by lighting and textures.
- Transparency can be used to increase the information presented (but with care, otherwise it will clutter the image).

Disadvantages

- Much more computational expensive.
- Greater algorithmic complexity.

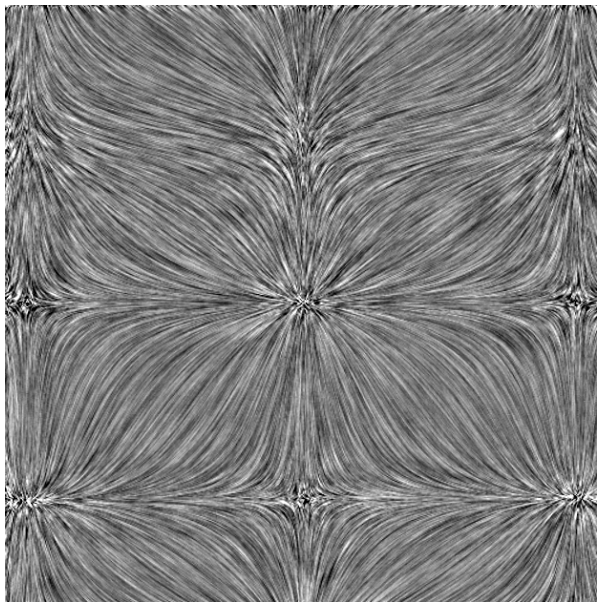
Generation

Created by linking and refining pathlines and then triangulating the geometry (more details in [Garth-2008]).

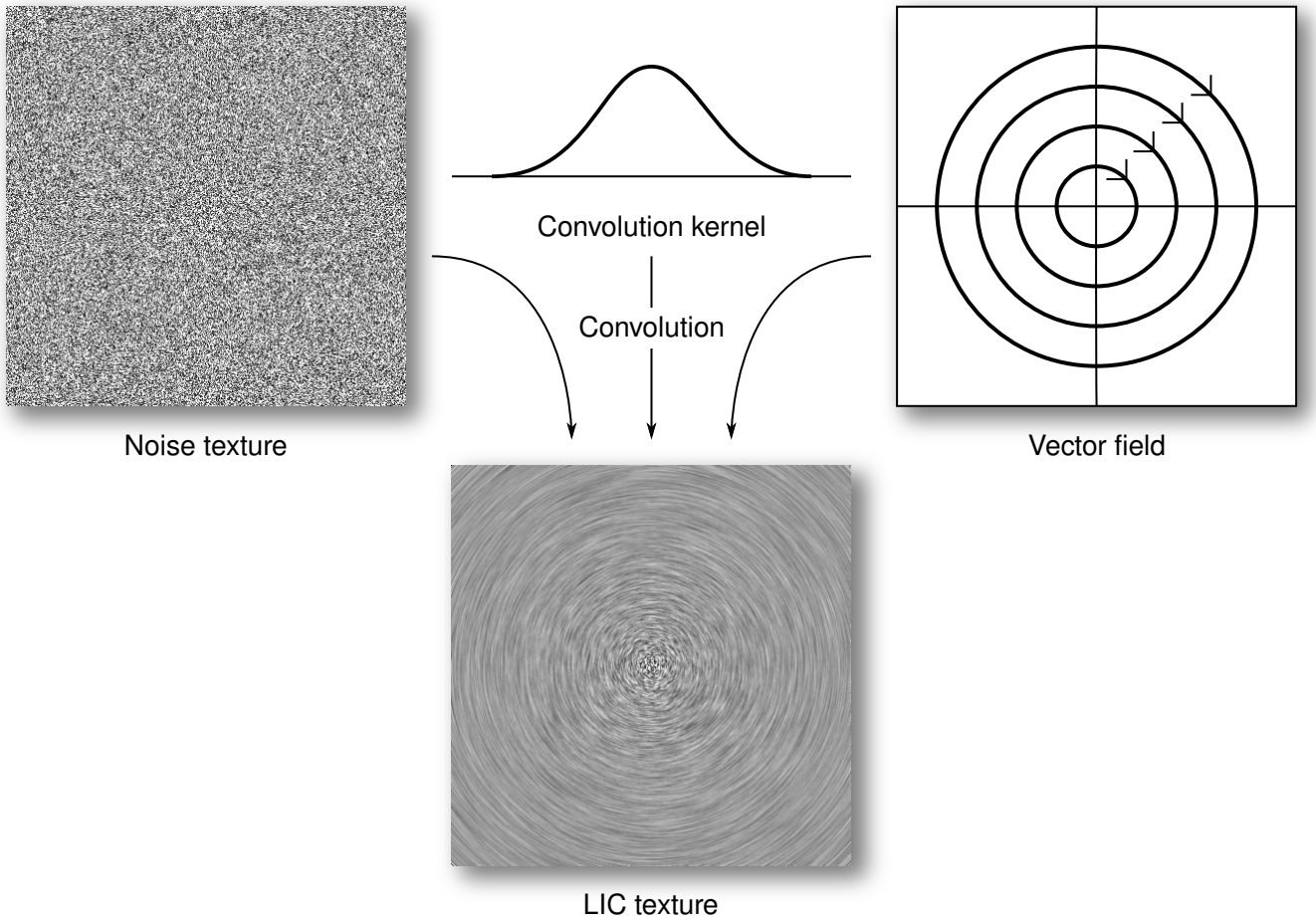
Line integral convolution

Concept

- LIC is a texture-based dense method.
- The technique consist of convolving of a kernel and a random noise texture along streamlines seeded backwards and forward from the center of each pixel.
- The final color of pixels whose streamline are similar will be highly correlated.
- Originally proposed by [Cabral-1993]



Computing LIC



A fast implementation for 2D is presented in [Stalling-1995].

Tensor visualization

Tensors

- Extension of the concept of vectors to higher dimensions.
- 0-order tensors are scalar, first-order are vectors
- Second-order tensors are often represented as matrices.
- Tensors have also algebraic operations defined.

Tensor fields

Some examples are:

- Diffusion tensors.
- Material properties: conductivity, stress, ...

Diffusion tensor fields

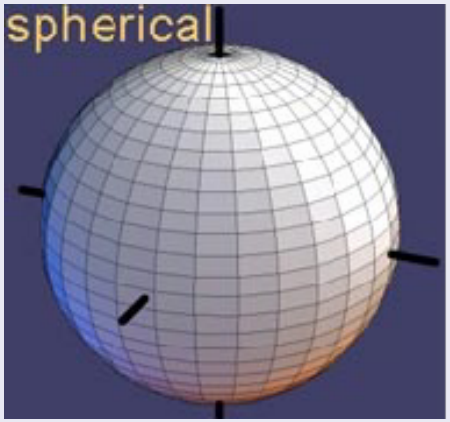
Diffusion tensor fields

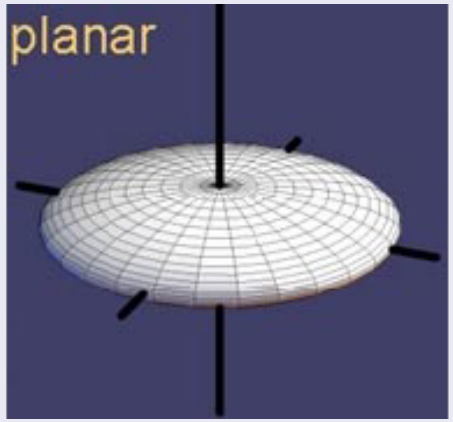
- The diffusion tensor represents the diffusion rate of particles inside a medium along the axis.
- Typical example of second order tensor.
- Output of modern MRI equipments.
- In 3D, a diffusion tensor is represented by a 3×3 symmetric matrix.

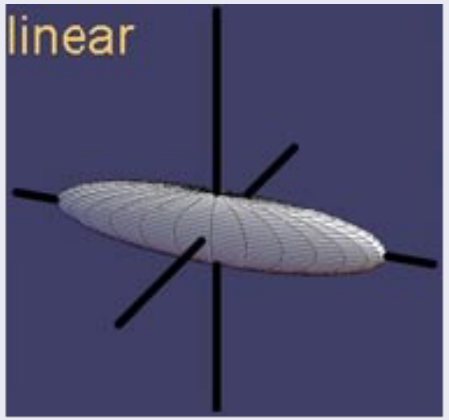
Tensor visualization

Tensor ellipsoids

- Symmetric tensors can be diagonalized (eigenvalues and eigenvectors.
- The diagonalization can be used to create ellipsoid glyphs
- In diffusion tensors, the largest eigenvalue vector represents the direction of maximum diffusion.



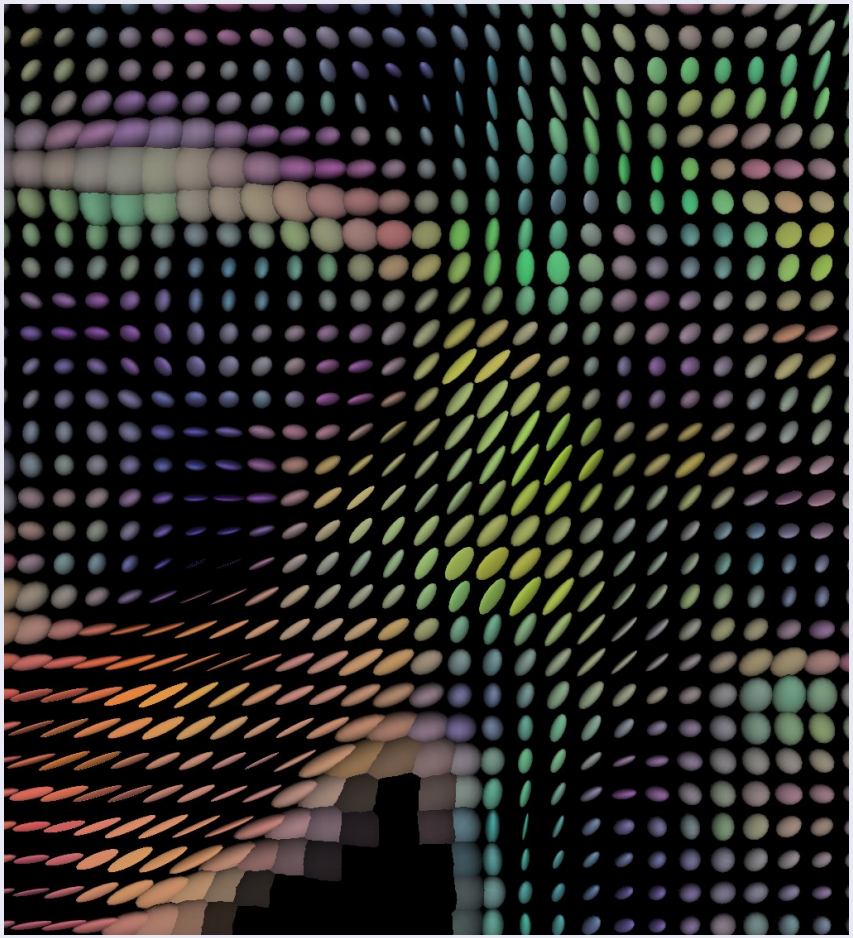




Credit: Weiskopf/Machiraju/Möller

Tensor visualization

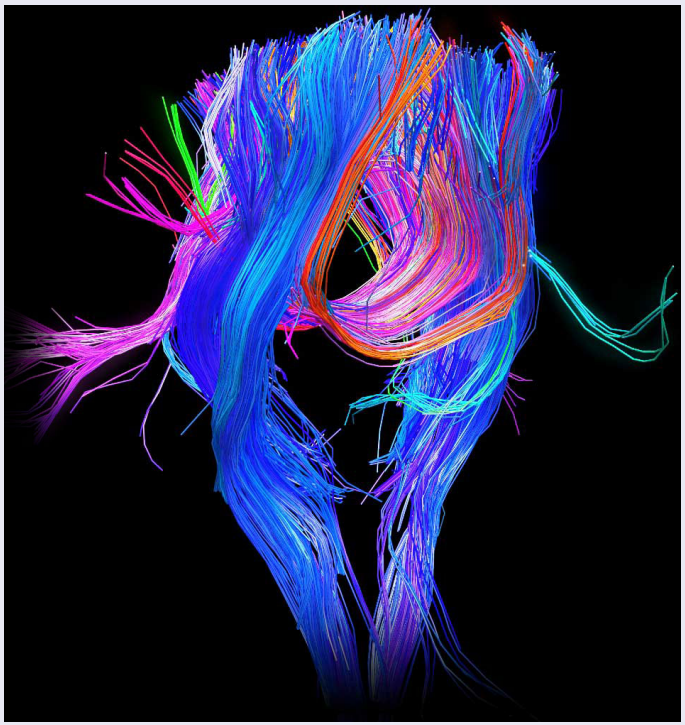
Tensor ellipsoids



Tensor visualization

Hyperstreamlines

- Considering only the major eigenvector for direction and the other two for shape vector field techniques can be applied.



©The Human Connectome Project


Brian Cabral and Leith Casey Leedom.
Imaging vector fields using line integral convolution.
In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 263–270, New York, NY, USA, 1993. ACM.


Klaus Engel, Martin Kraus, and Thomas Ertl.
High-quality pre-integrated volume rendering using hardware-accelerated pixel shading.
In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, pages 9–16, New York, NY, USA, 2001. ACM.


Christoph Garth, Han Krishnan, Xavier Tricoche, Tom Tricoche, and Kenneth I. Joy.
Generation of accurate integral surfaces in time-dependent vector fields.
IEEE Transactions on Visualization and Computer Graphics, 14(6):1404–1411, November 2008.


Daniel Jönsson, Erik Sundén, and Timo Ynnerman, Anders an d Ropinski.
Interactive Volume Rendering with Volumetric Illumination.
In *Eurographics STAR program*, 2012.
accepted.


Joe Kniss, Gordon Kindlmann, and Charles Hansen.
Multidimensional transfer functions for interactive volume rendering.
IEEE Transactions on Visualization and Computer Graphics, 8(3):270–285, July 2002.


Yarden Livnat, Han wei Shen, and Christopher R. Johnson.
A near optimal isosurface extraction algorithm using the span space.
IEEE Transactions on Visualization and Computer Graphics, 2:73–84, 1996.


William E. Lorensen and Harvey E. Cline.

Marching cubes: A high resolution 3d surface construction algorithm.
In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.


Stephen R. Marschner and Richard J. Lobb.
An evaluation of reconstruction filters for volume rendering.
In *Proceedings of the conference on Visualization '94*, VIS '94, pages 100–107, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.


Nelson Max.
Optical models for direct volume rendering.
IEEE Transactions on Visualization and Computer Graphics, 1(2):99–108, June 1995.


Detlev Stalling and Hans-Christian Hege.
Fast and resolution independent line integral convolution.

In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 249–256, New York, NY, USA, 1995. ACM.