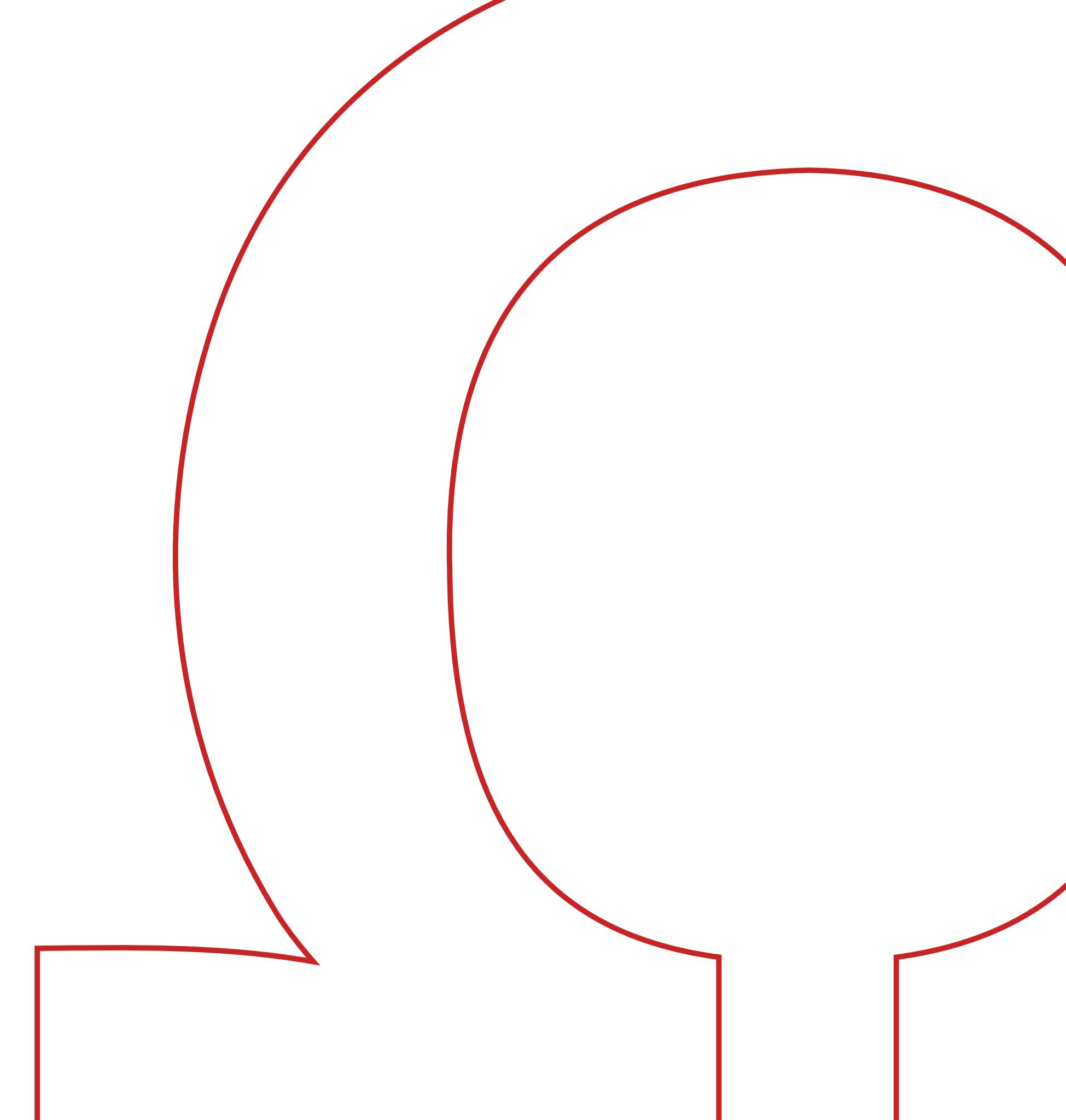




Visualization

Prof. Dr. Matthias Teßmann
Technische Hochschule Nürnberg
Department Computer Science
Summer Term 2023



Moodle

<https://elearning.ohmportal.de/course/view.php?id=12692>

Registration Password: **PiXXL_2023**

Contact

Prof. Dr. Matthias Teßmann

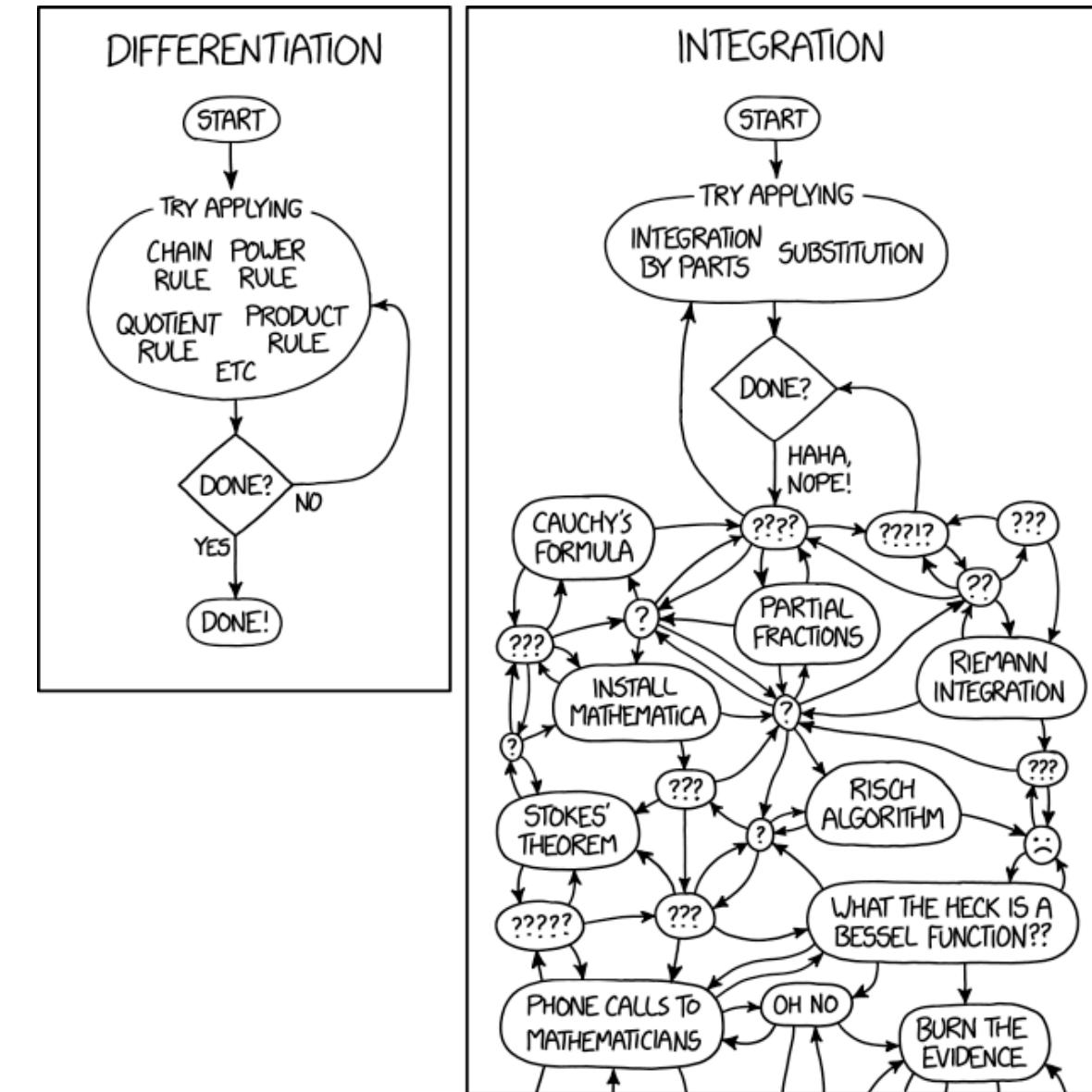
- Mail: matthias.tessmann@th-nuernberg.de
- Tel.: +49 911 5880-1193
- Office: HQ.411 (appointments **online only**)
 - Contact me via mail or MS Teams chat for an appointment if you need to
- You can ask simple questions via MS Teams chat
 - But you can and should also ask during the lecture and especially the exercises
 - Also: use the Q&A forum in Moodle

Organization

- Lecture
 - Tuesday, 13:00h – 14:30h, HQ.405
- Exercises
 - Wednesday, 13:00h – 14:30h, HQ.205
- Exam
 - Written exam / online exam at the end of semester (90 minutes)
 - You can bring 5 pages self-compiled material and a calculator
 - Only students assigned to this FWPF can take the exam!

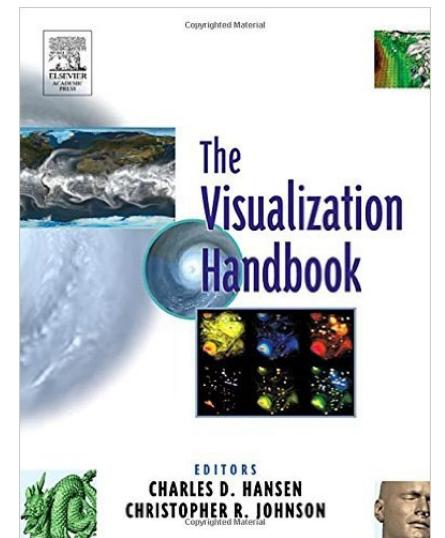
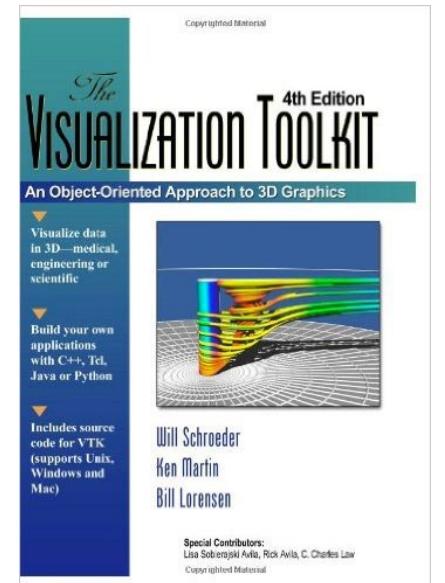
Prerequisites

- Computer graphics
 - There will be an introduction to basic concepts (CG Primer)
- Mathematics
 - Vector calculus and linear algebra
 - Some differential-and integral calculus



Organization

- Exercises
 - Practical implementation and application of rendering and visualization techniques in C++ using VTK
- Literature
 - "The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics", Schröder, Martin, Lorensen (available online as PDF)
 - Focus on algorithms, not implementation
 - Software: <http://www.vtk.org>
- Further reading
 - “The Visualization Handbook”, C. D. Hansen, C. R. Johnson, Academic Press 2004



Lecture contents

- Introduction
- Computer Graphics Primer
- Fundamental Concepts
- 1D and 2D scalar fields
- 3D scalar fields
- 2D and 3D vector fields

1. Introduction

What is visualization?

- “to visualize” (Oxford English Dictionary)
 - Form a mental vision, image or picture of
 - something not visible,
 - or present to sight,
 - or of an abstraction
 - To make visible to the mind or imagination
 - e.g. air pressure
 - Transformation of an abstraction to a picture
 - e.g. molecular structure

"One picture is worth ten thousand words."

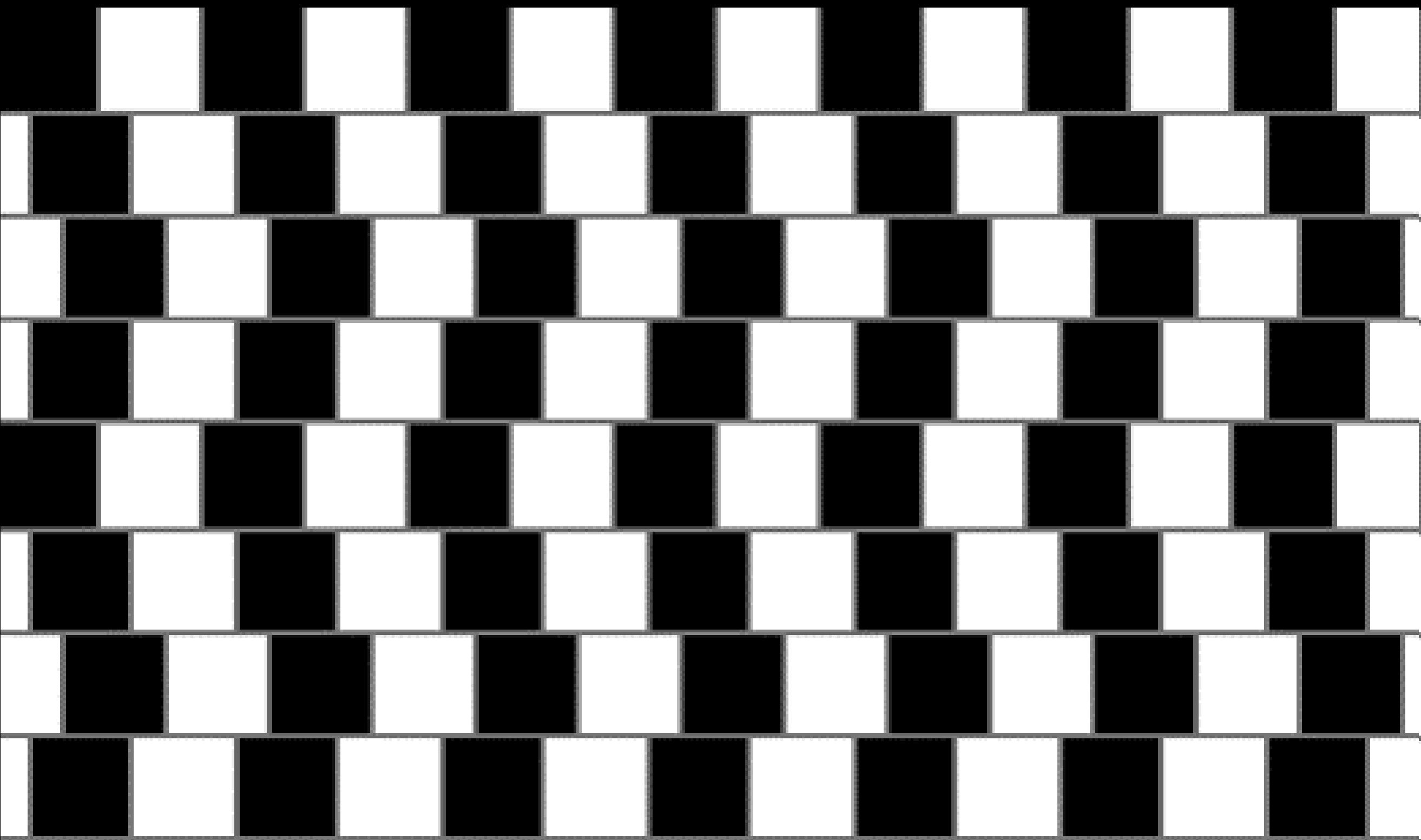
– Fred R. Barnard, Advertisement in *Printers' Ink*, 1921

"Denken ist interessanter als Wissen, aber nicht als Anschauen."

– Johann Wolfgang von Goethe, 1749 - 1832

Importance of the Human Visual System





Color Test

YELLOW

ORANGE

BLUE

BLACK

GREEN

RED

PURPLE

RED

YELLOW

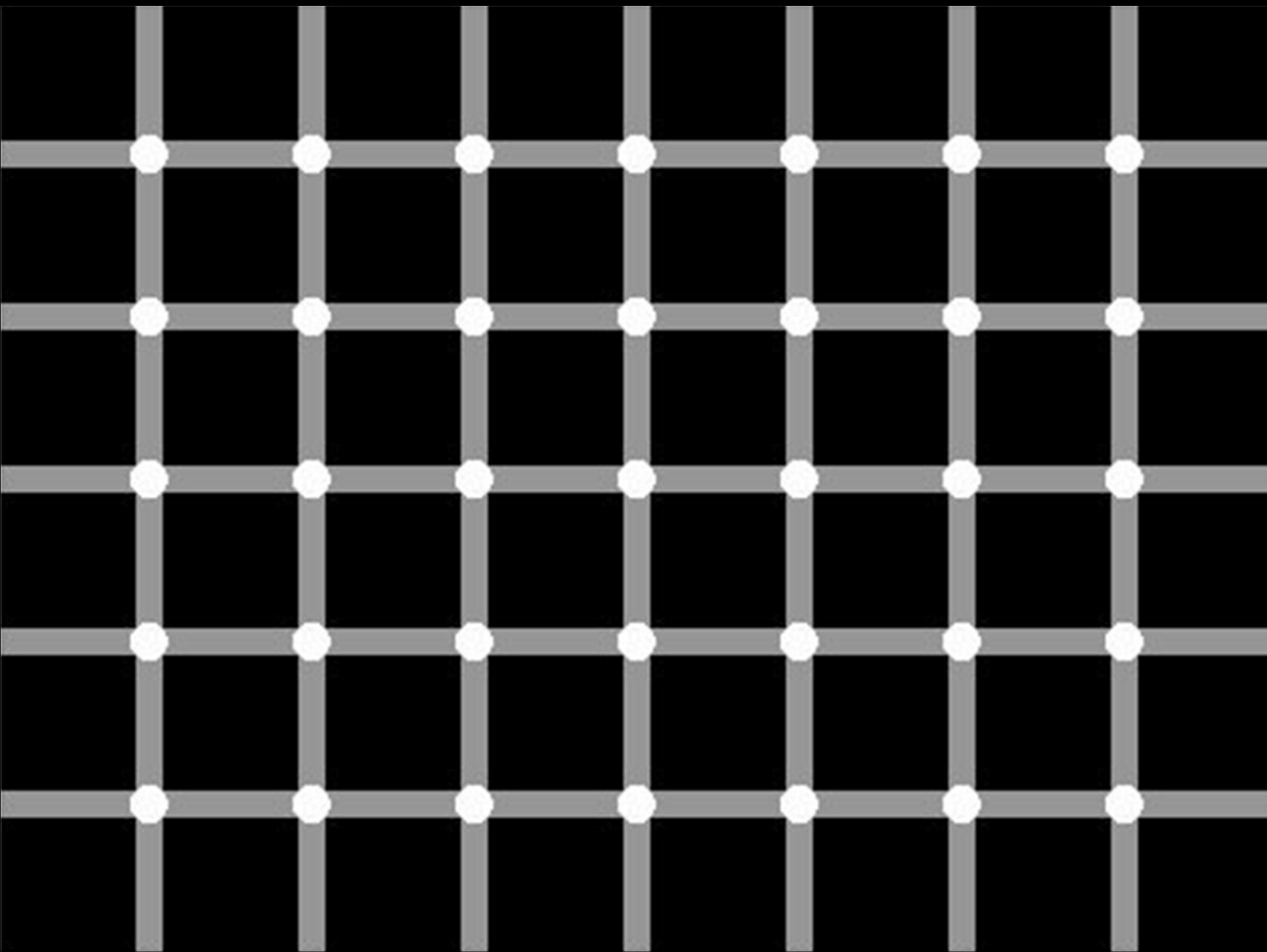
ORANGE

YELLOW

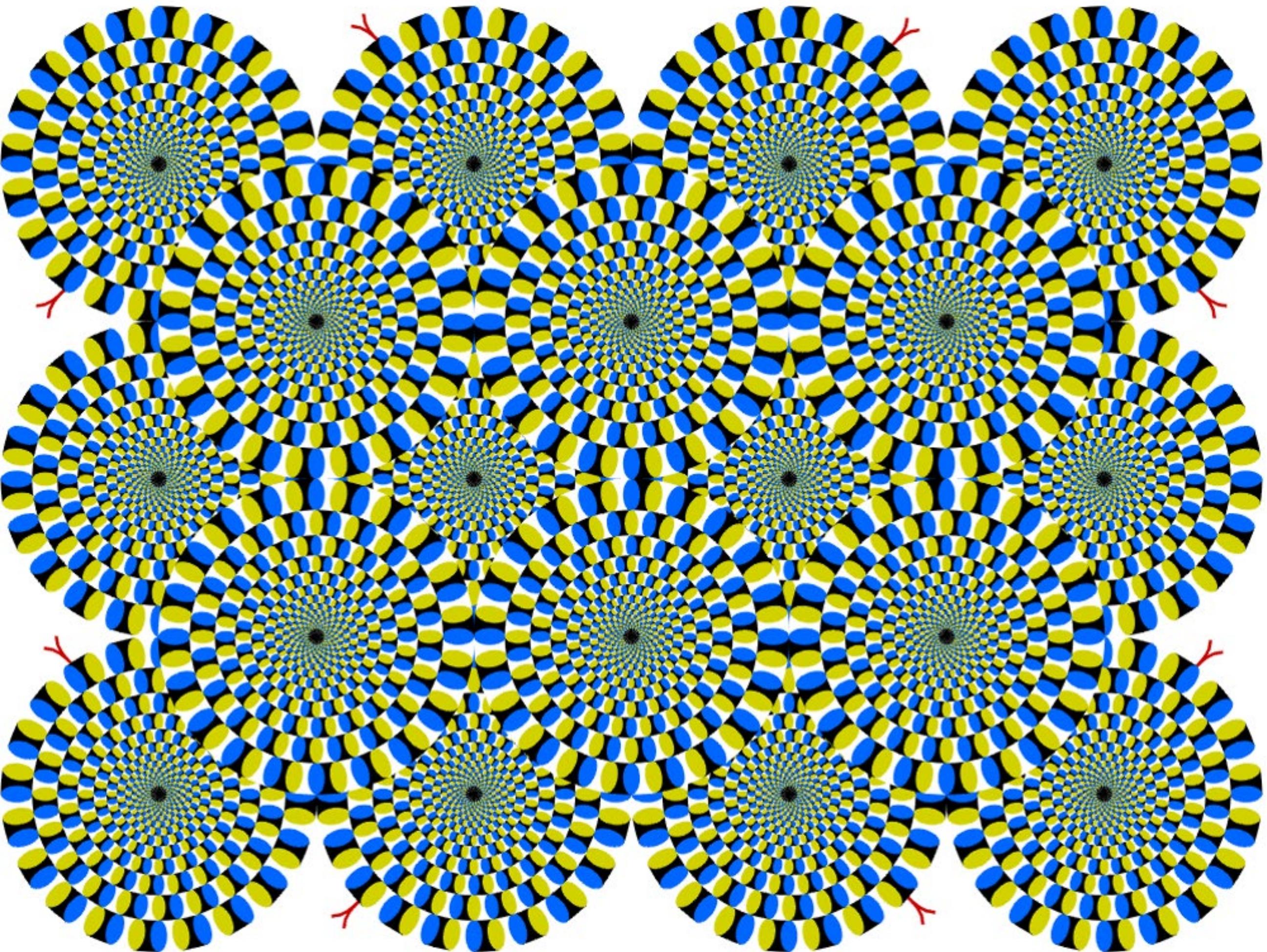
GREEN

Left-Right Conflict

Your right brain tries to say the color, but your left
brain insists on reading the word!



ohm



About this Lecture

Definitions

- Scientific Visualization
 - Visualization in scientific and technical environments
 - Computer aided extraction and display of information
 - from measured or simulated data
 - Not
 - Education
 - Marketing
 - Art
 - ...

Definitions

"Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields it is already revolutionizing the way scientists do science."

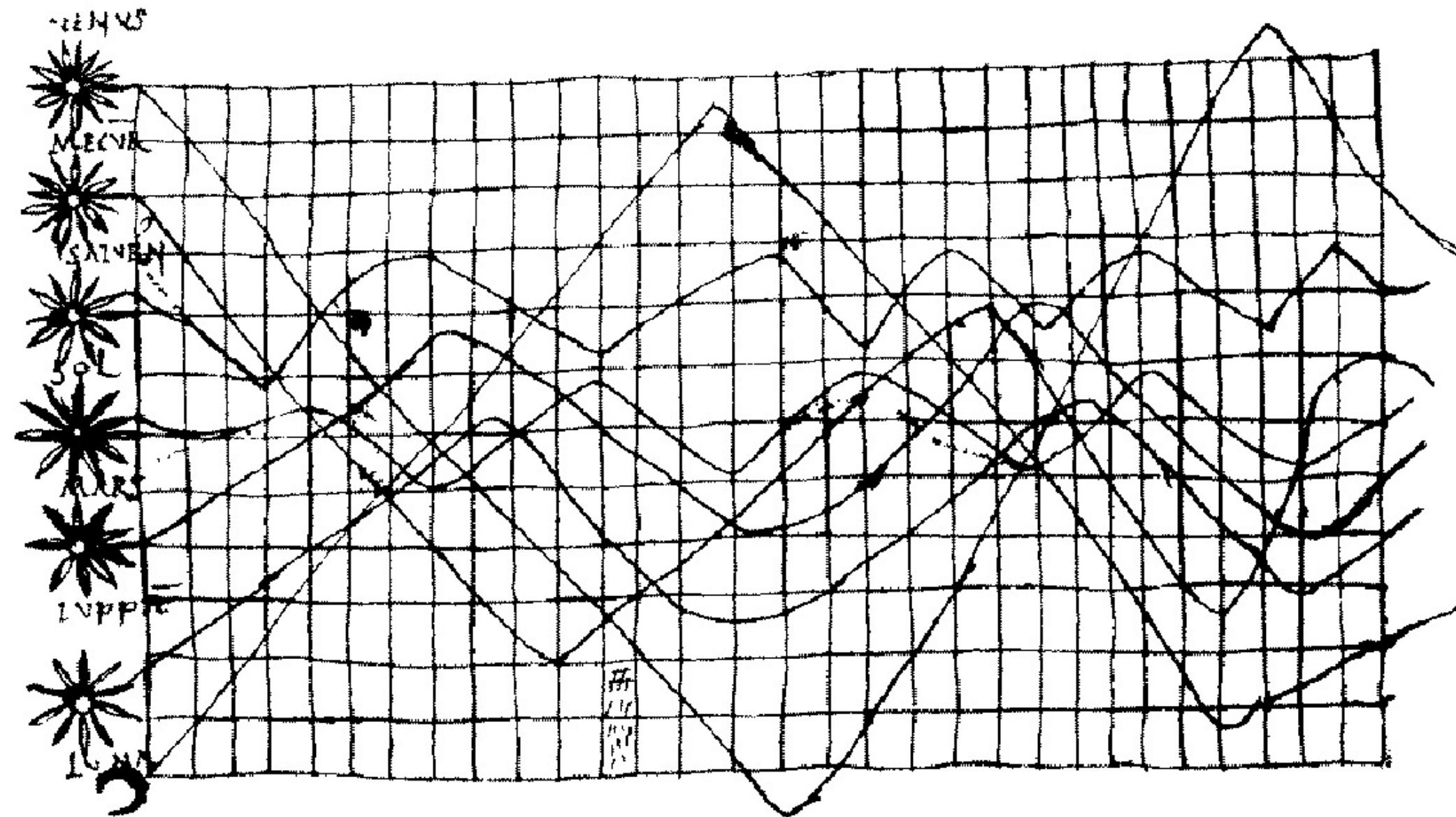
– McCormick, DeFanti, Brown, Visualization in Scientific Computing, Computer Graphics Vol. 21.6, Nov. 1987

Definitions and goals

- Insight and analysis
 - Extract information content
 - Make coherences visible that are not apparent
 - Analyze data by means of visual representation
- Communication
 - Allow the non-expert to understand
 - Guide the expert into the right direction
- Steering
 - Interactively control and drive your application
 - Accelerate the understanding of phenomena

Historic Examples

History



Time series: inclination of planets

History



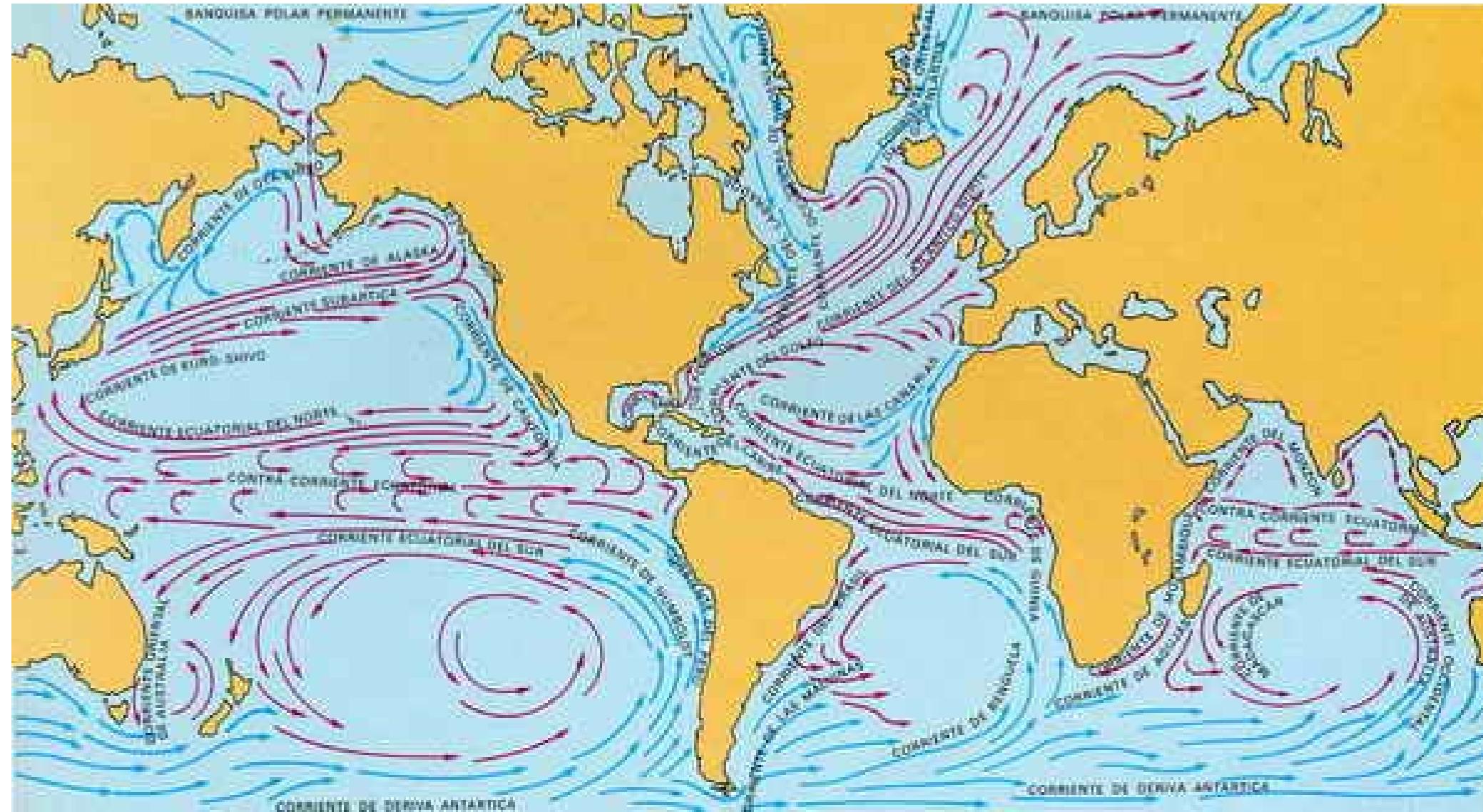
Ptolemaic map of the world by Johan Scotus (1505)
based on the writings of Claudius Ptolemy (87 - 150 A.D.)

History



Dr. John Snows map
(1855) of death from
cholera in the broad
Street area
September 1854.

History



Main ocean currents

Provides better understanding of transoceanic travel
Courtesy of Albatros, Enciclopedia del mar, Barcelona 1974

Interdisciplinarity

Interdisciplinarity

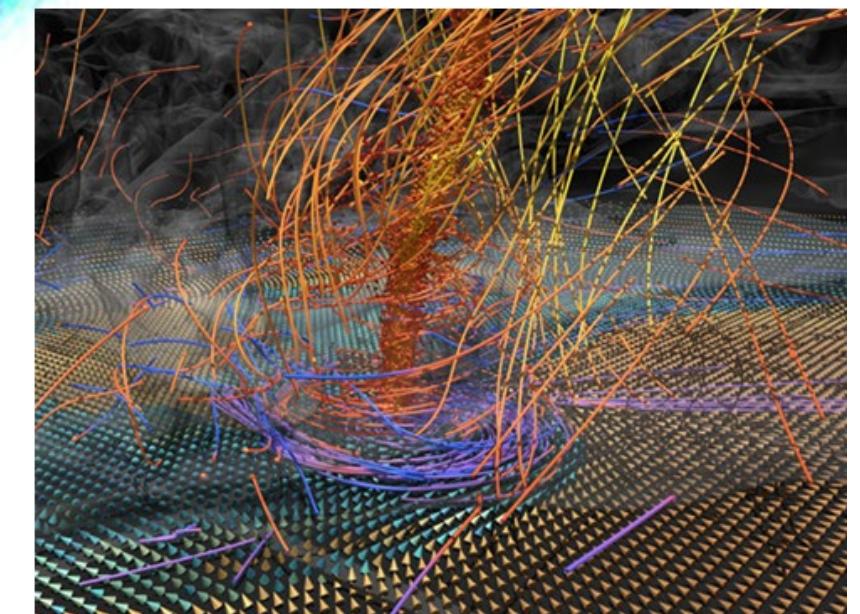
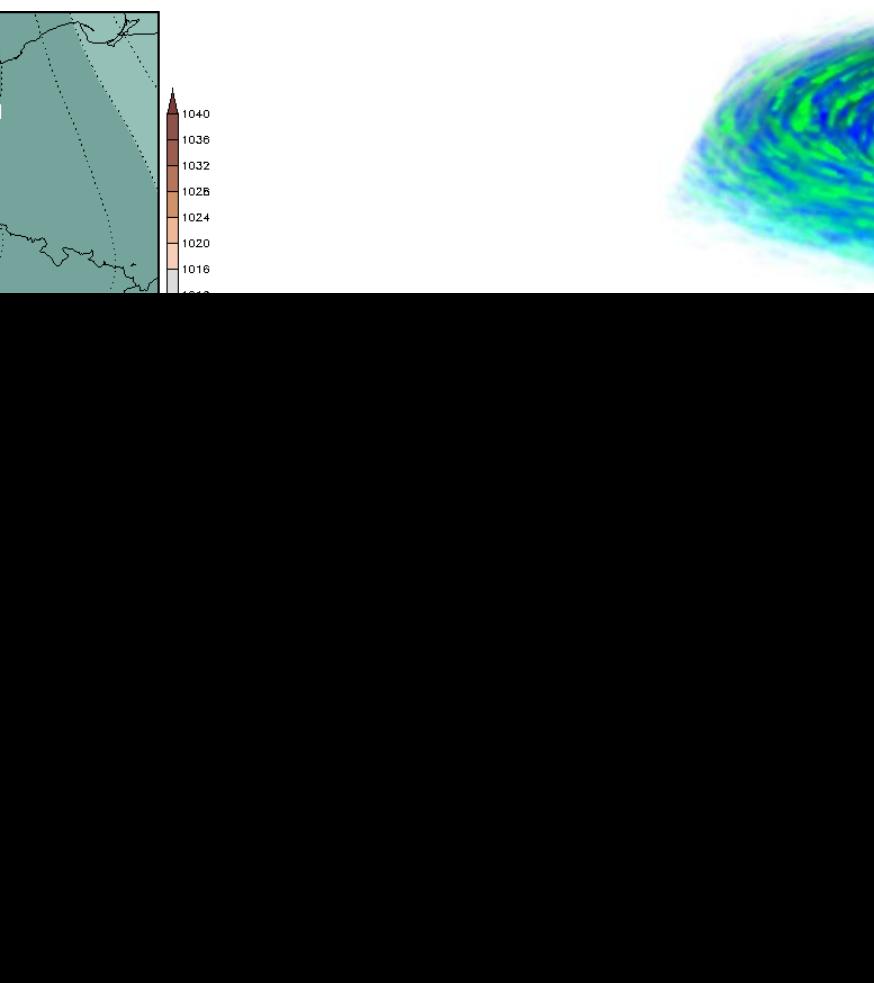
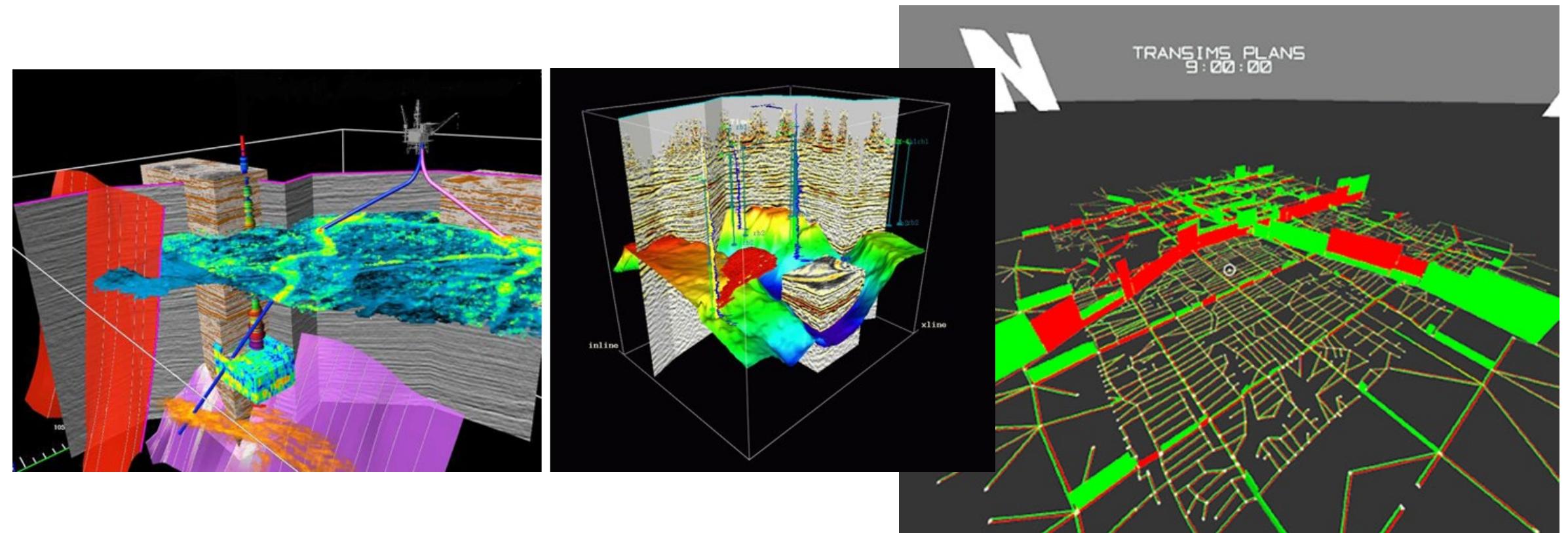
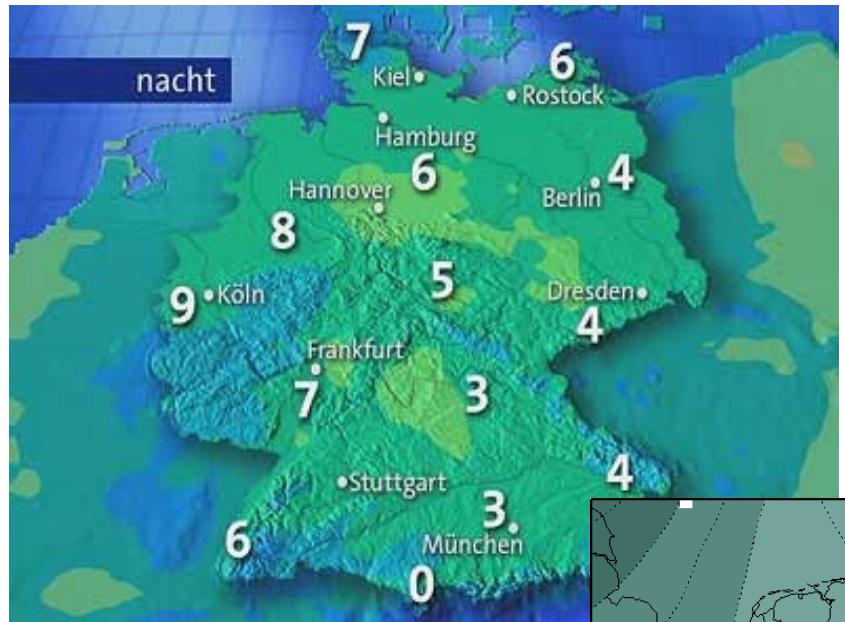
- Measurements and experiments
 - Medical imaging (e.g. computed tomography, ultrasound, ...)
 - Geology (e.g. oil exploration)
 - Meteorology and environment (e.g. weather, satellites)
 - Astronomy (e.g. digital radio telescope @300GB/s)
- Computer simulations
 - Computational fluid dynamics (CFD)
 - Structural mechanics
 - Chemistry (molecular modeling)
 - Physics (computational physics)

Interdisciplinarity

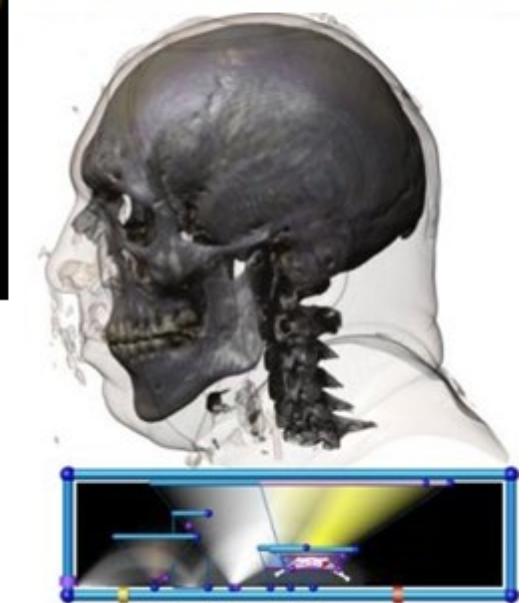
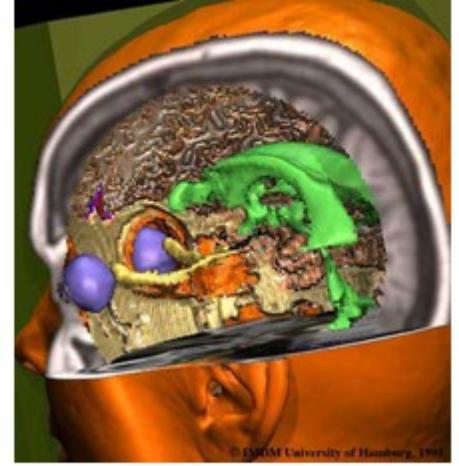
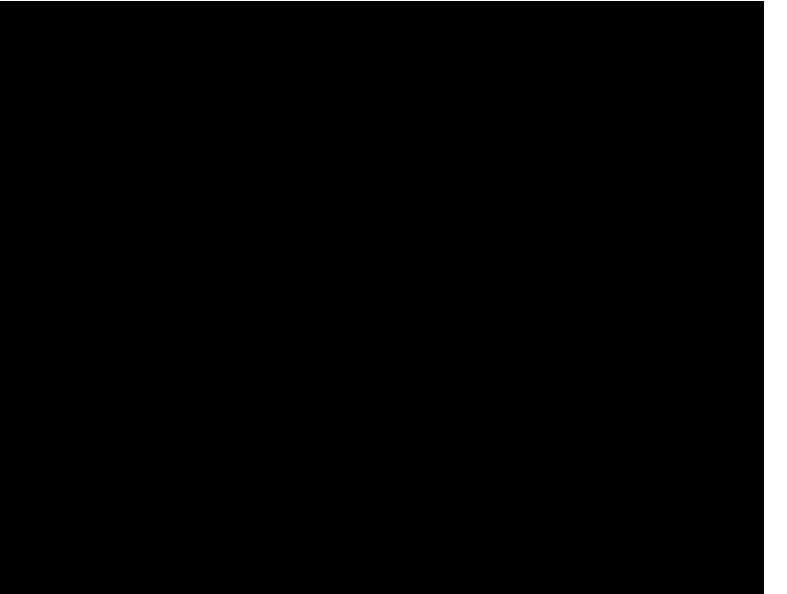
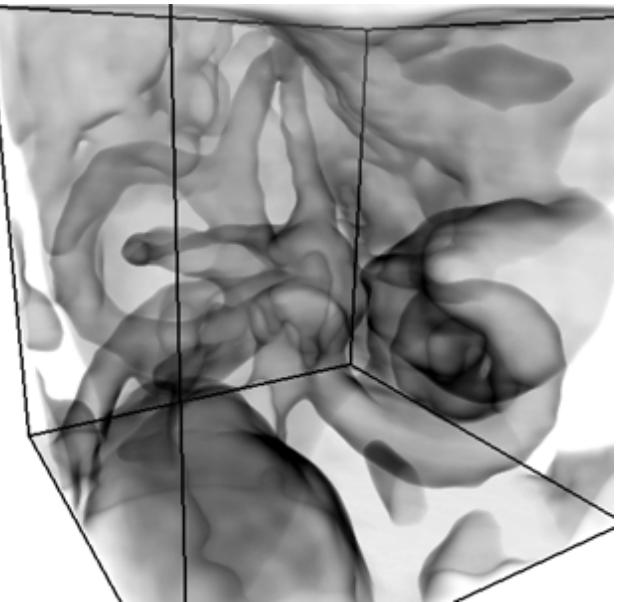
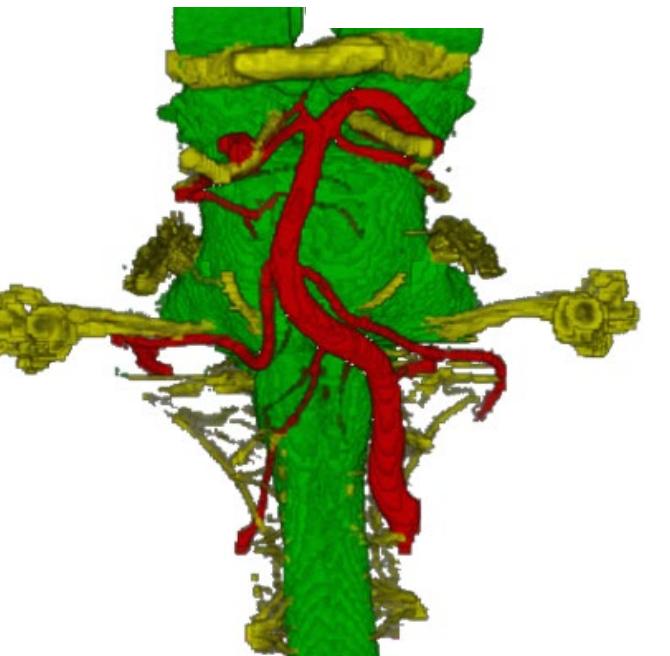
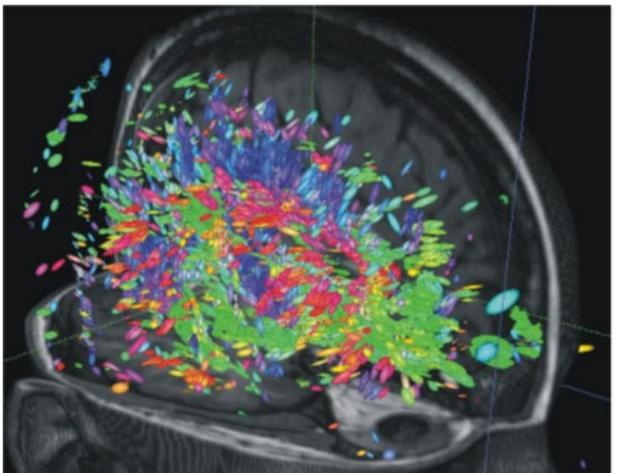
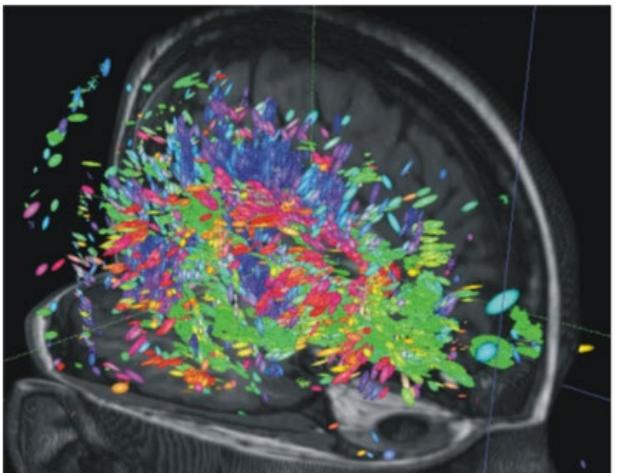
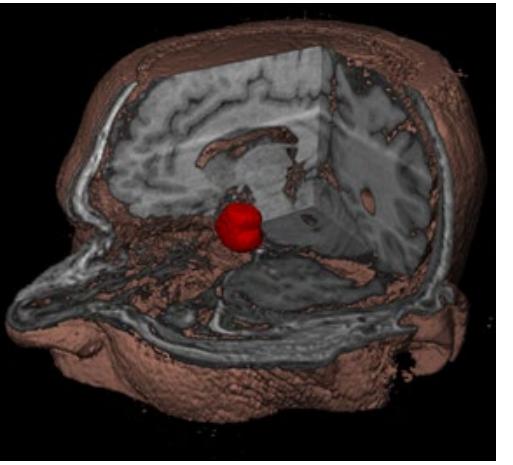
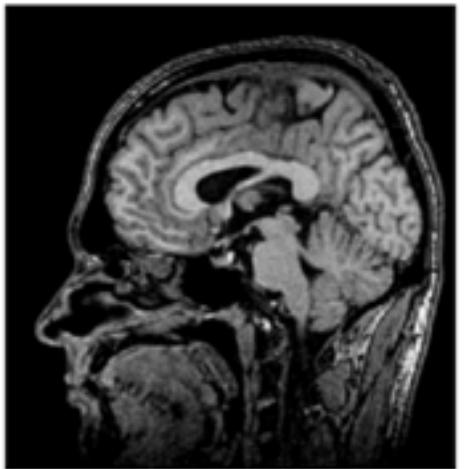
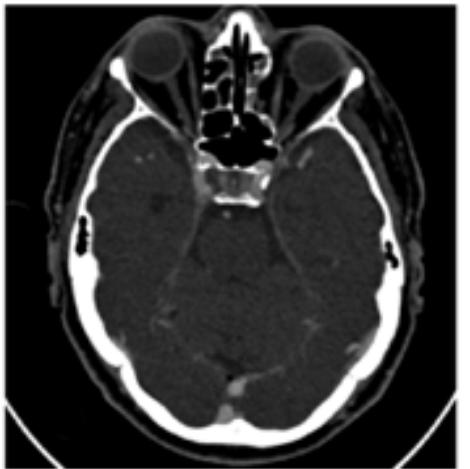
- Links with the following areas
 - Engineering
 - Numerical mathematics
 - Image processing
 - Computer vision
 - Physics
 - Chemistry
 - Medicine
 - Psychology
 - ...

Examples

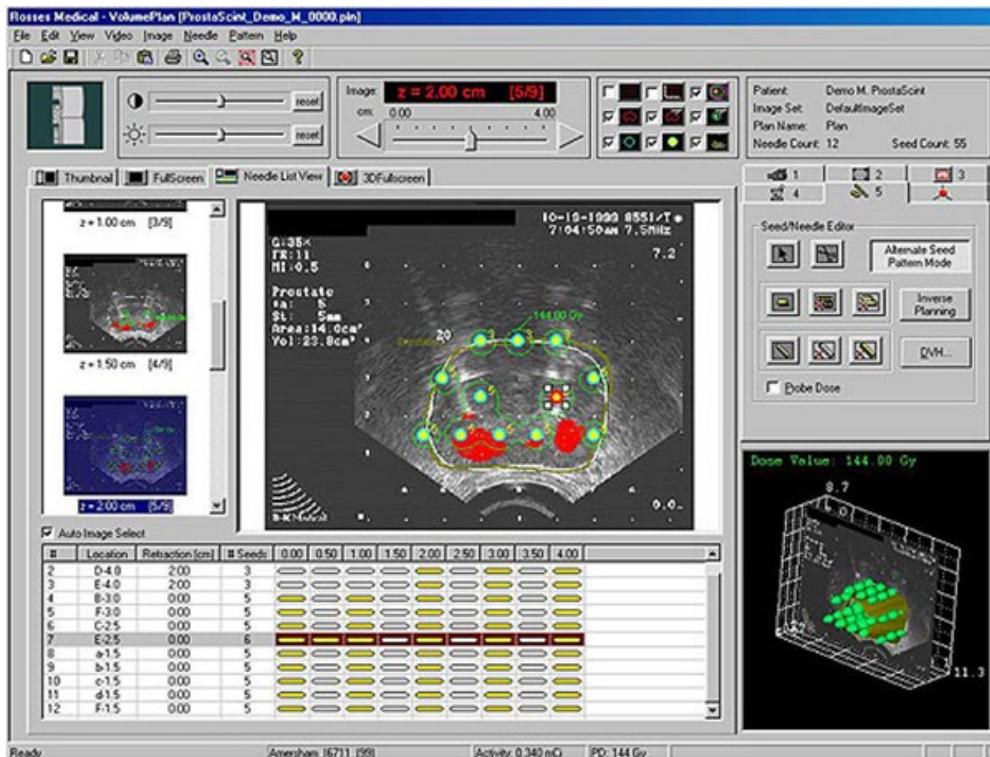
Geosciences



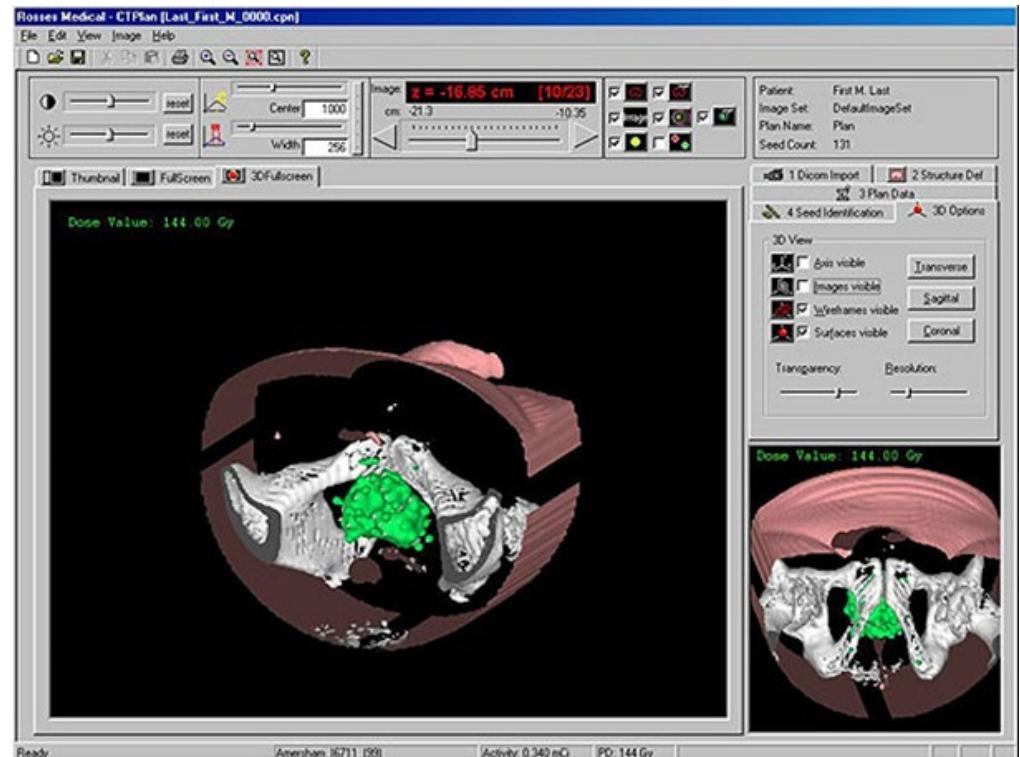
Medicine



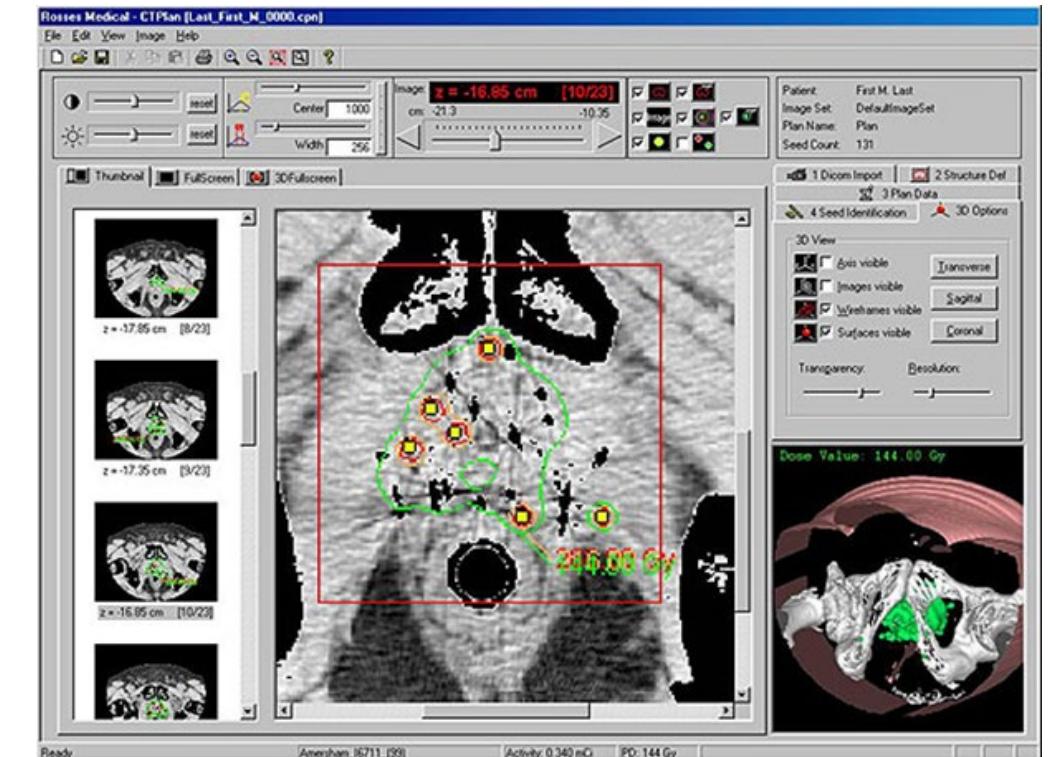
Medicine



Diagnosis

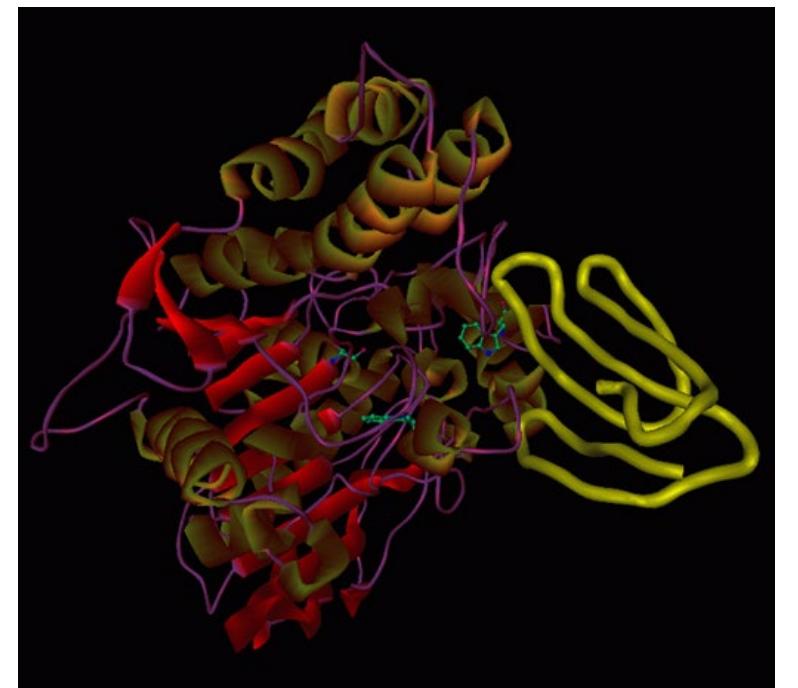
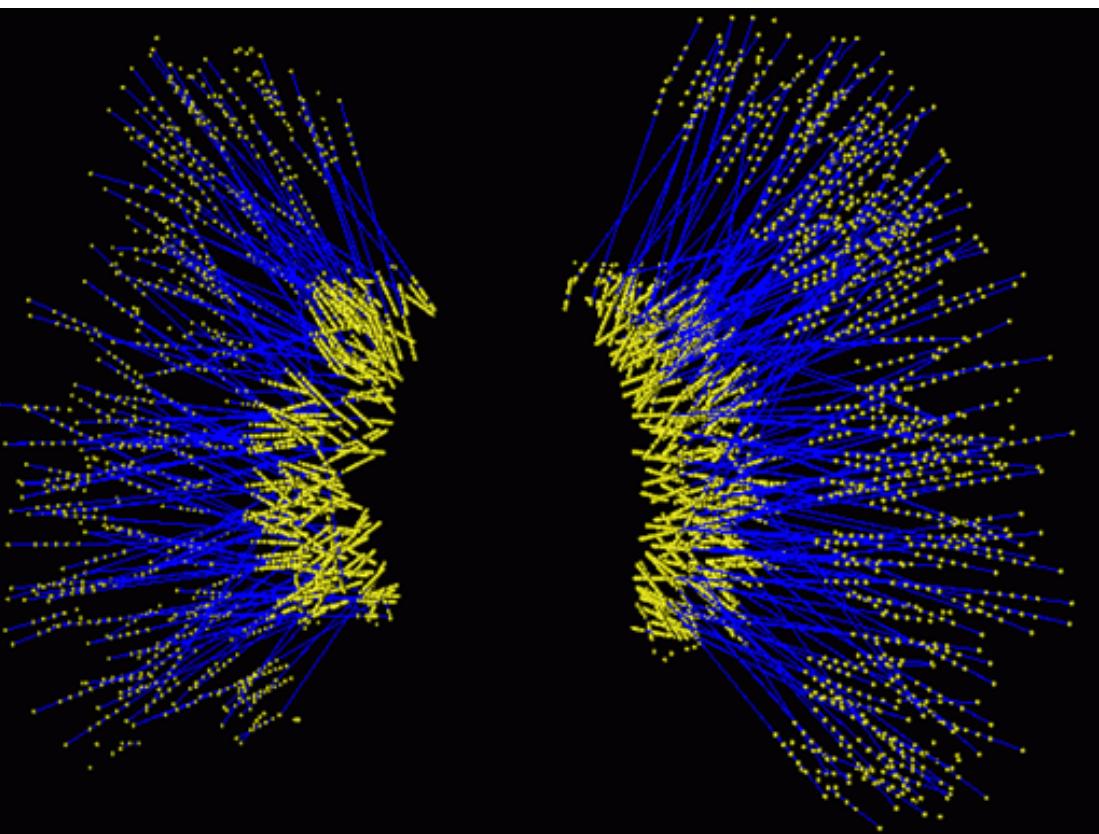
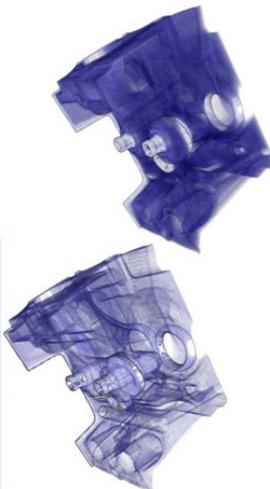
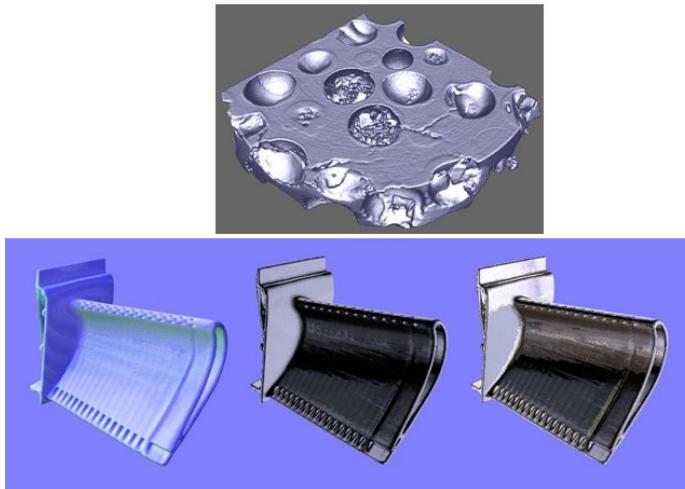


Therapy

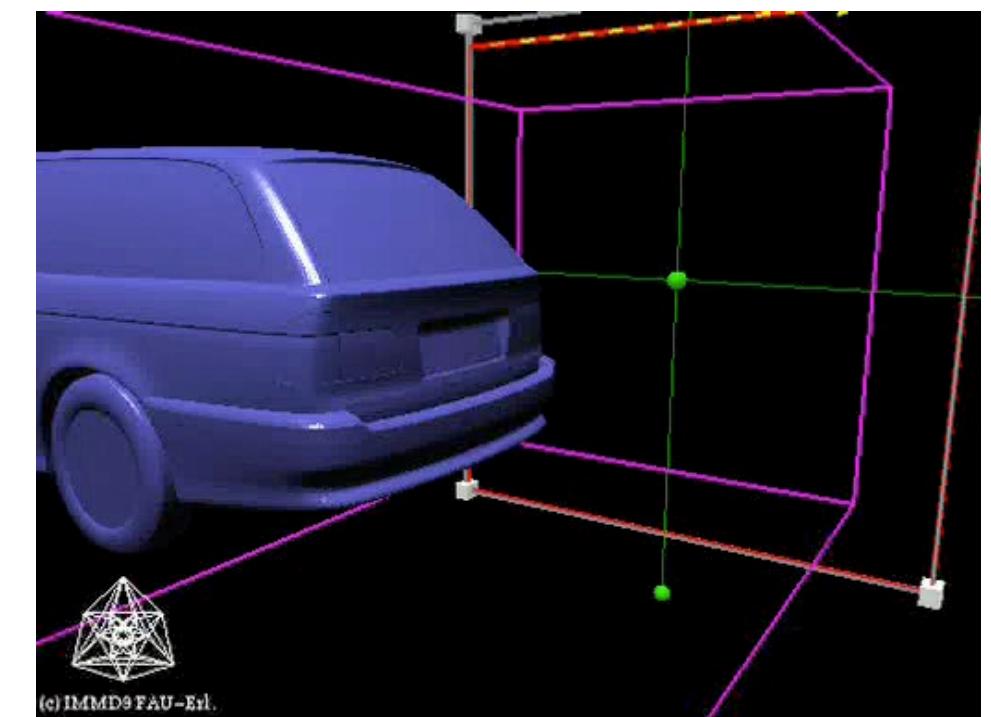
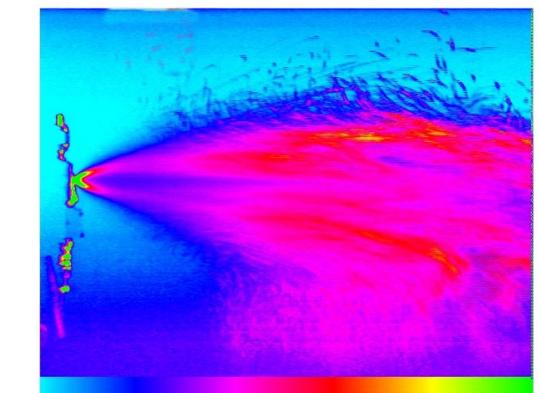
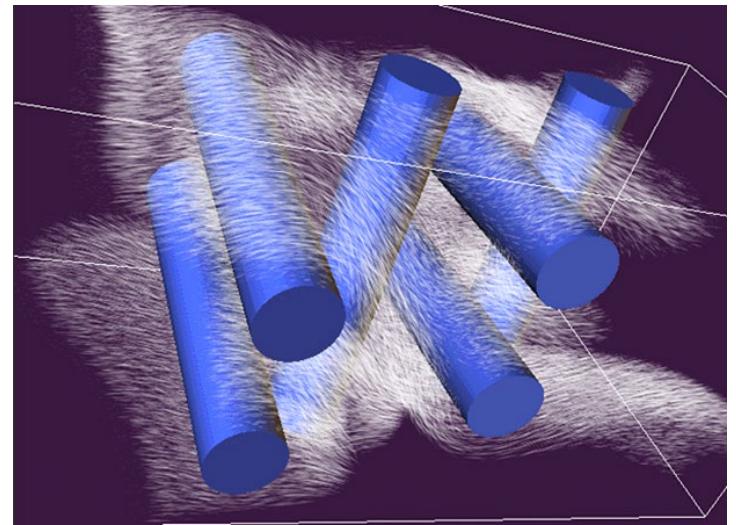
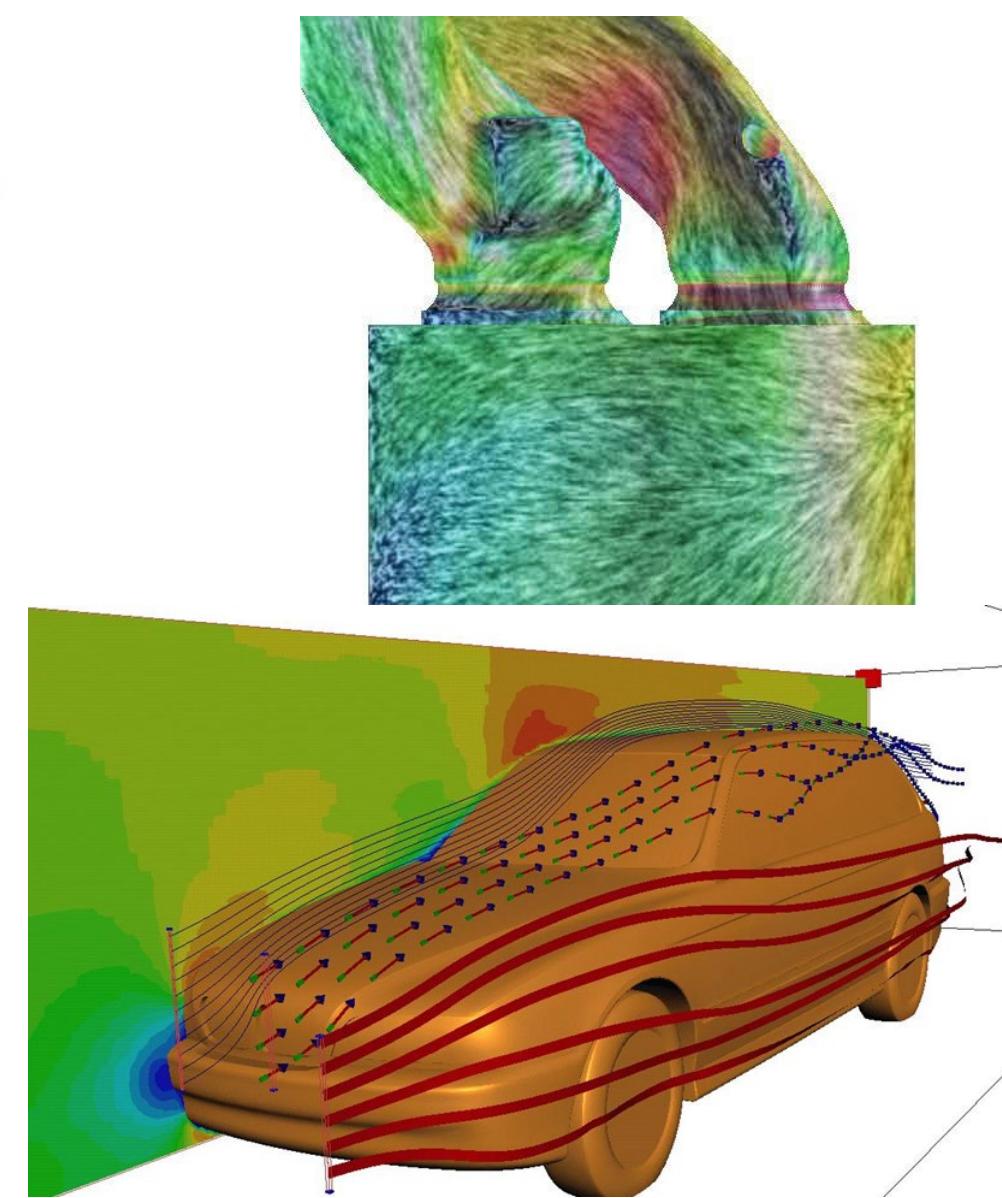
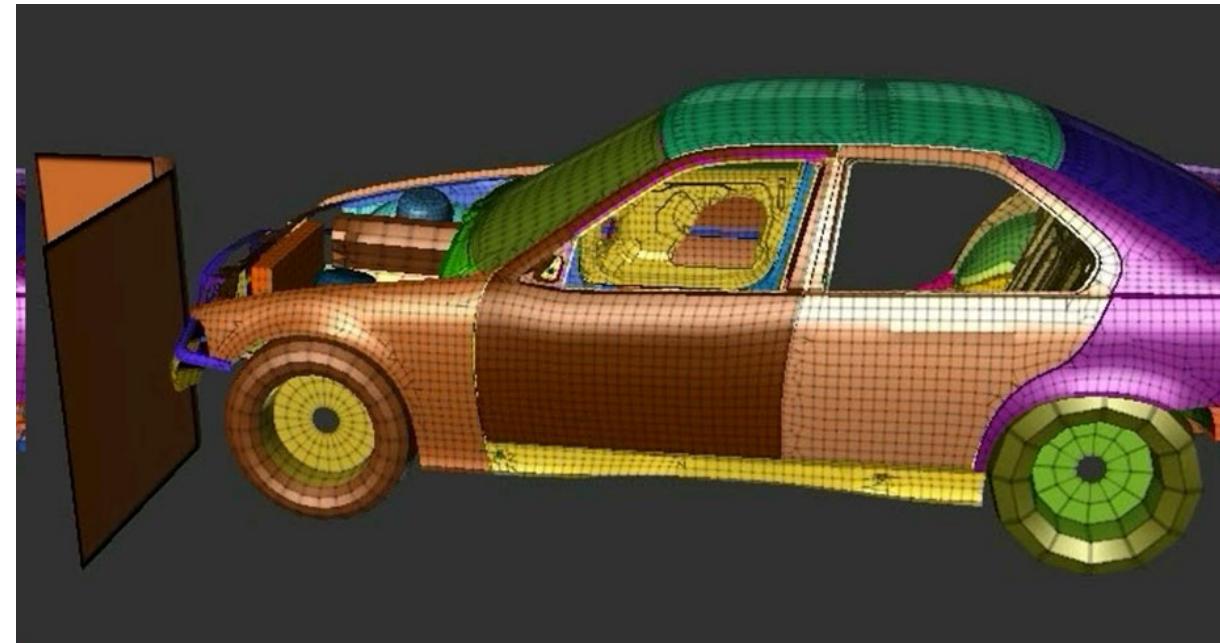
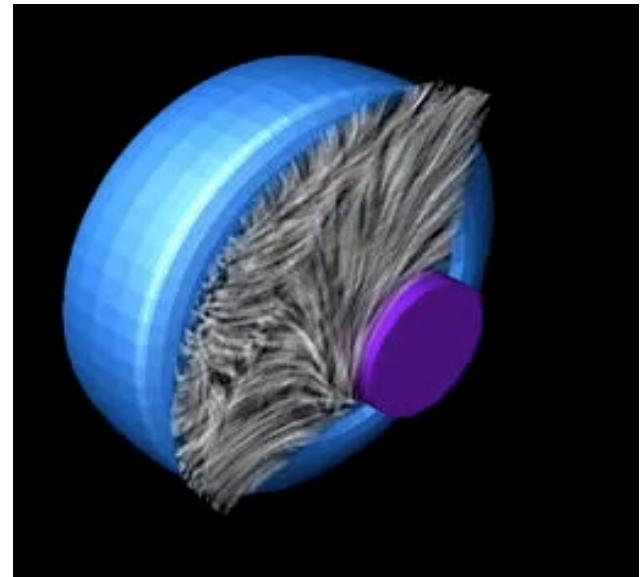
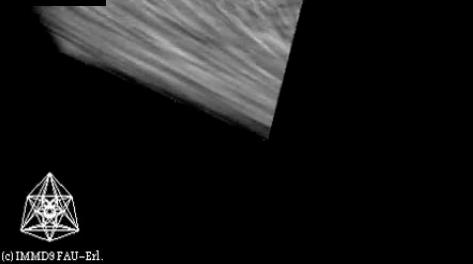
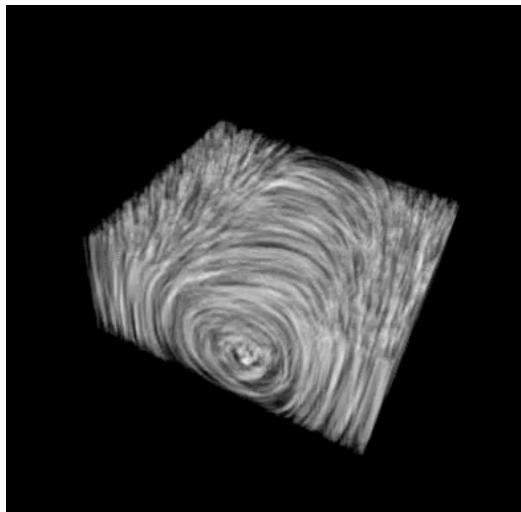
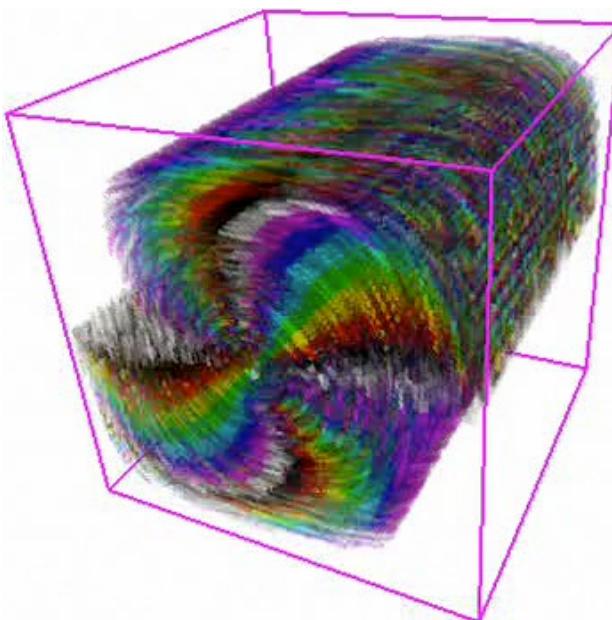


Follow up

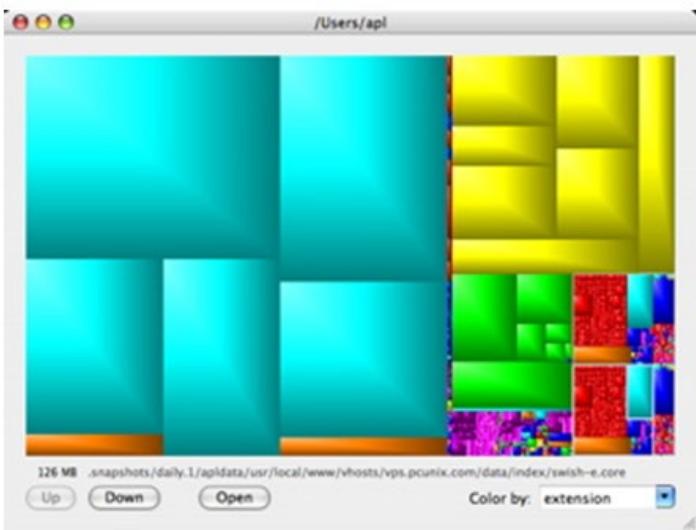
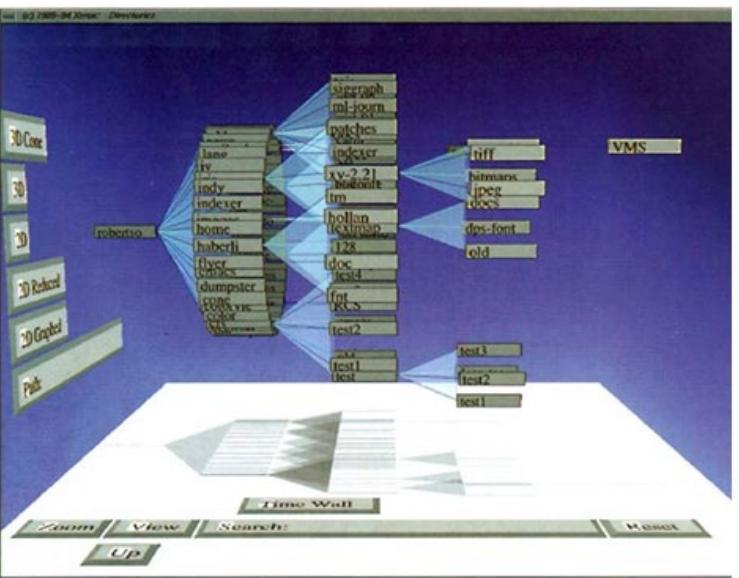
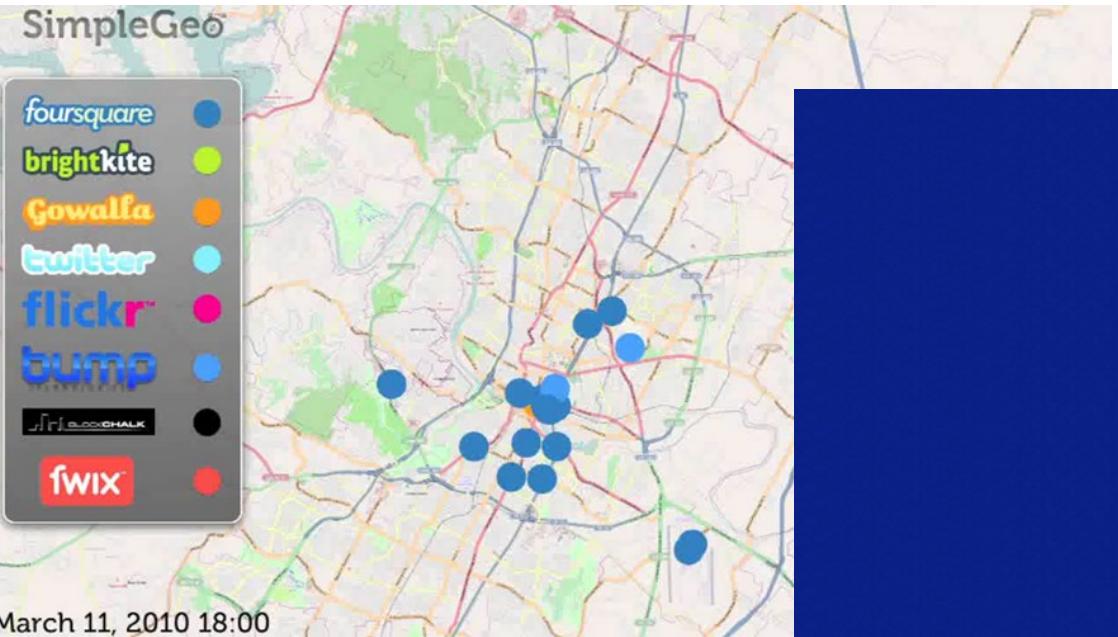
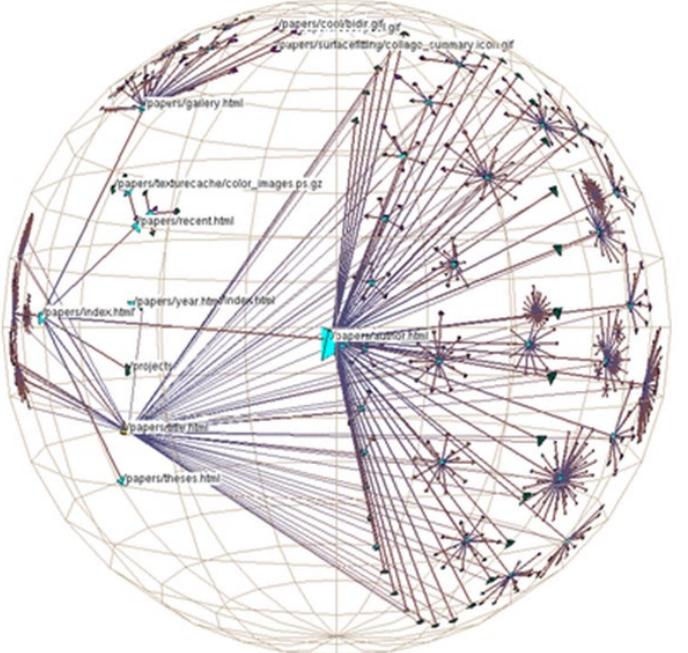
Materials, Archeology, Chemistry, Physics, ...



Engineering



Information and Algorithms



Related Fields

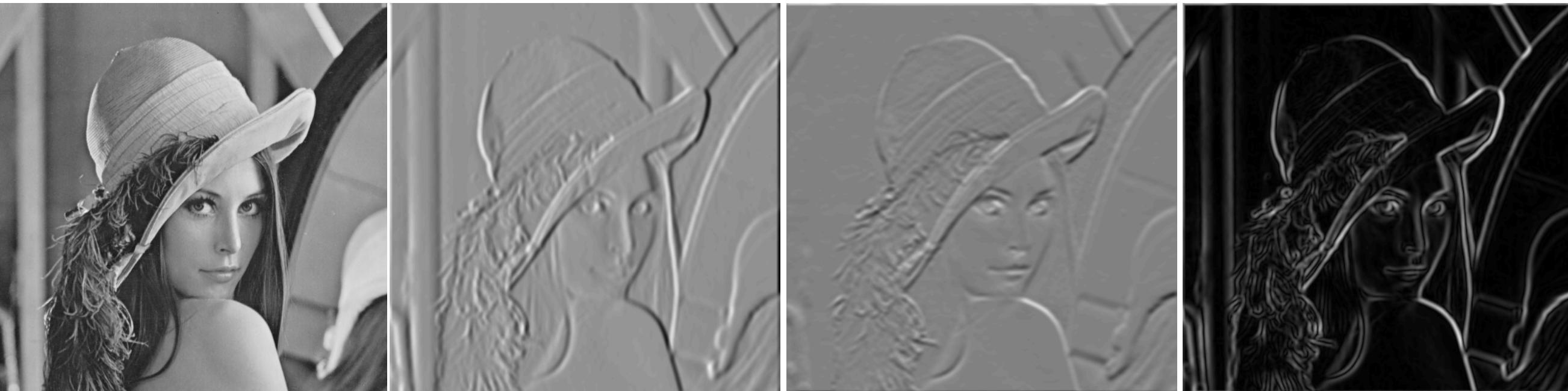
Related Fields

Image Synthesis and Geometric Modelling



Related Fields

Image Processing and Computer Vision



Literature

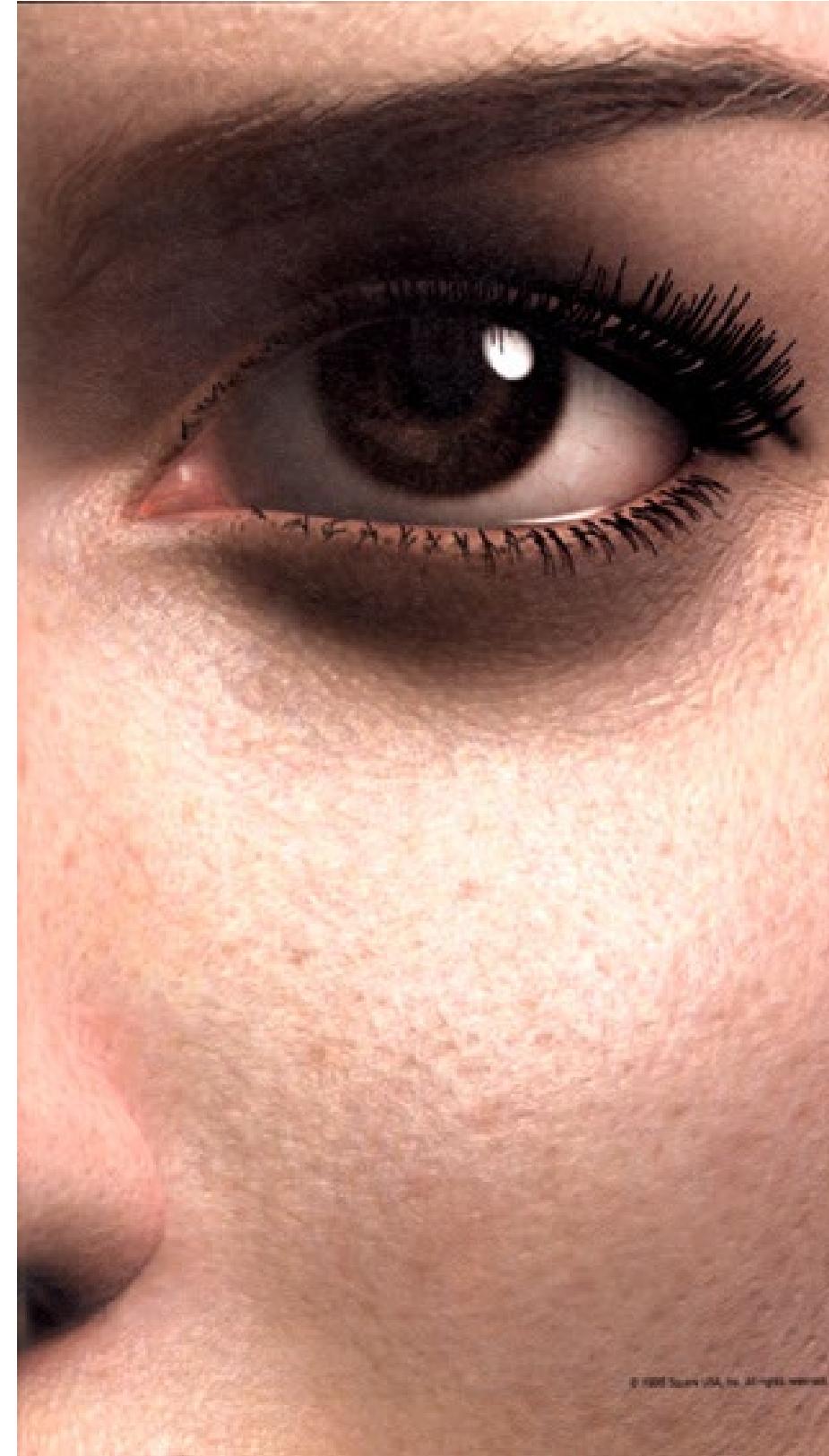
Books

- R. A. Earnshaw, N. Wiseman (Eds.), An Introductory Guide to Scientific Visualization, Springer, 1992
- K.W. Brodlie u.a. (Eds.), Scientific Visualization - Techniques and Applications, Springer 1992
- Richard S. Gallagher (Ed.), Computer Visualization: Graphics Techniques for Scientific and Engineering Analysis, CRC Press, 1995
- G.M. Nielson, H. Hagen, H. Müller, Scientific Visualization, IEEE Computer Society Press, Los Alamitos, 1997
- C.D. Hansen and C.R. Johnson, The Visualization Handbook, Academic Press, 2004
- C. Rezk-Salama, K. Engel, M. Hadwiger, J. Kniss, D. Weiskopf, Real-Time Volume Graphics, AK Peters, 2006
- W. Schroeder, K. Martin, B. Lorensen, Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition, Kitware, 2006
- D. Weiskopf, GPU-Based Interactive Visualization Techniques, Springer; 2006
- B. Preim, D. Bartz, Visualization in Medicine. Theory, Algorithms, and Applications: Theory, Algorithms, and Applications, Morgan Kaufmann, 2007
- AC. Telea, Data Visualization: Principles and Practice, AK Peters, 2008
- Wolfgang Engel, GPU Pro: Advanced Rendering Techniques, A K Peters, 2010
- M. Ward, G.G. Grinstein, D. Keim, Interactive Data Visualization: Foundations, Techniques, and Applications, Taylor & Francis, 2010

Journals and DL

- Journals:
 - IEEE Transactions on Visualization and Computer Graphics
 - IEEE Computer Graphics & Applications
 - The Visual Computer (Springer)
 - Computers & Graphics (Pergamon)
 - Visualization and Computer Animation (Wiley)
- IEEE / ACM digital libraries
 - Full text access from computers within the university

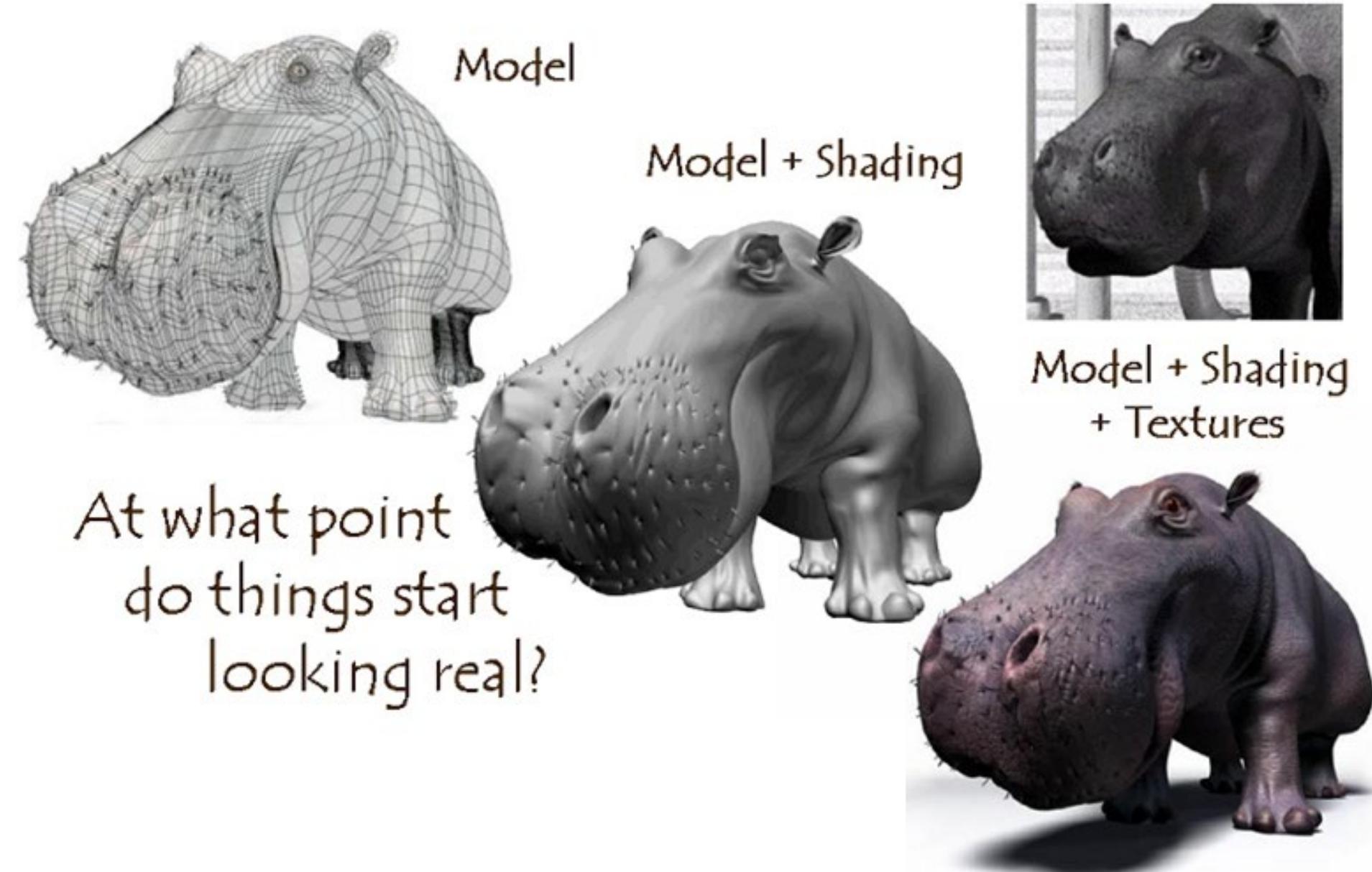
2. Computer Graphics Primer



Source: Various: LGDV, Nvidia, BMW

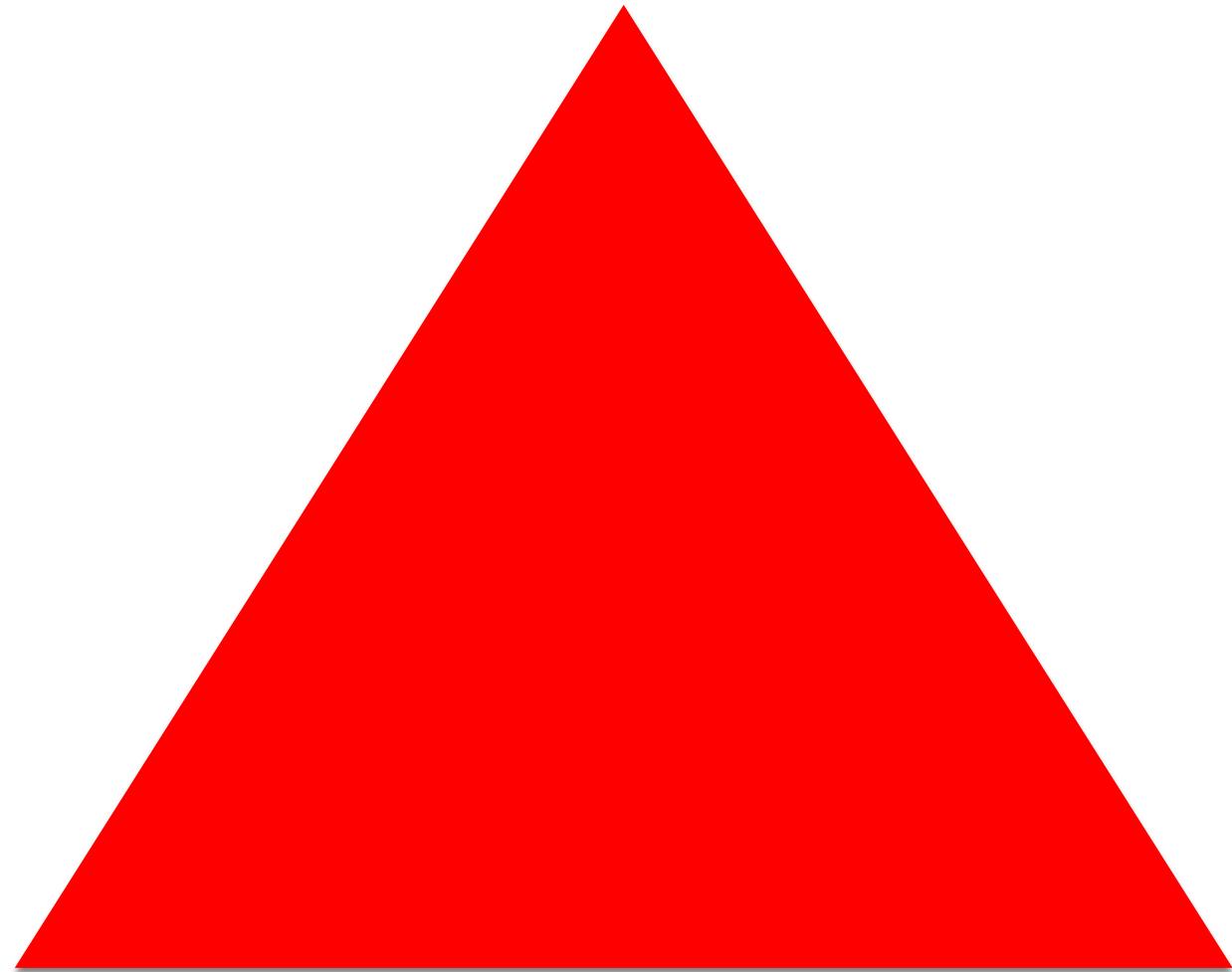
Prof. Dr. Matthias Teßmann

Computer Graphics



Computer Graphics

We start here



Contents

- The rendering pipeline
- Transformations
- Projections
- Color
- Lighting and Shading
- Texture Mapping

2.1 The Rendering pipeline

The Rendering Pipeline

The Rendering Pipeline

Key Graphics Areas

- Modeling
 - Mathematical specification of shape and appearance properties of objects
 - Primitives: Points, lines, curves, surfaces, ...
 - Attributes: color, texture maps, lighting properties, ...
 - Geometric transformations
- Animation
 - Create the illusion of motion: sequence of still images
 - Time as parameter for modeling and rendering
 - Keyframes, physically-based animations, collision detection, ...

The Rendering Pipeline

Key Graphics Areas

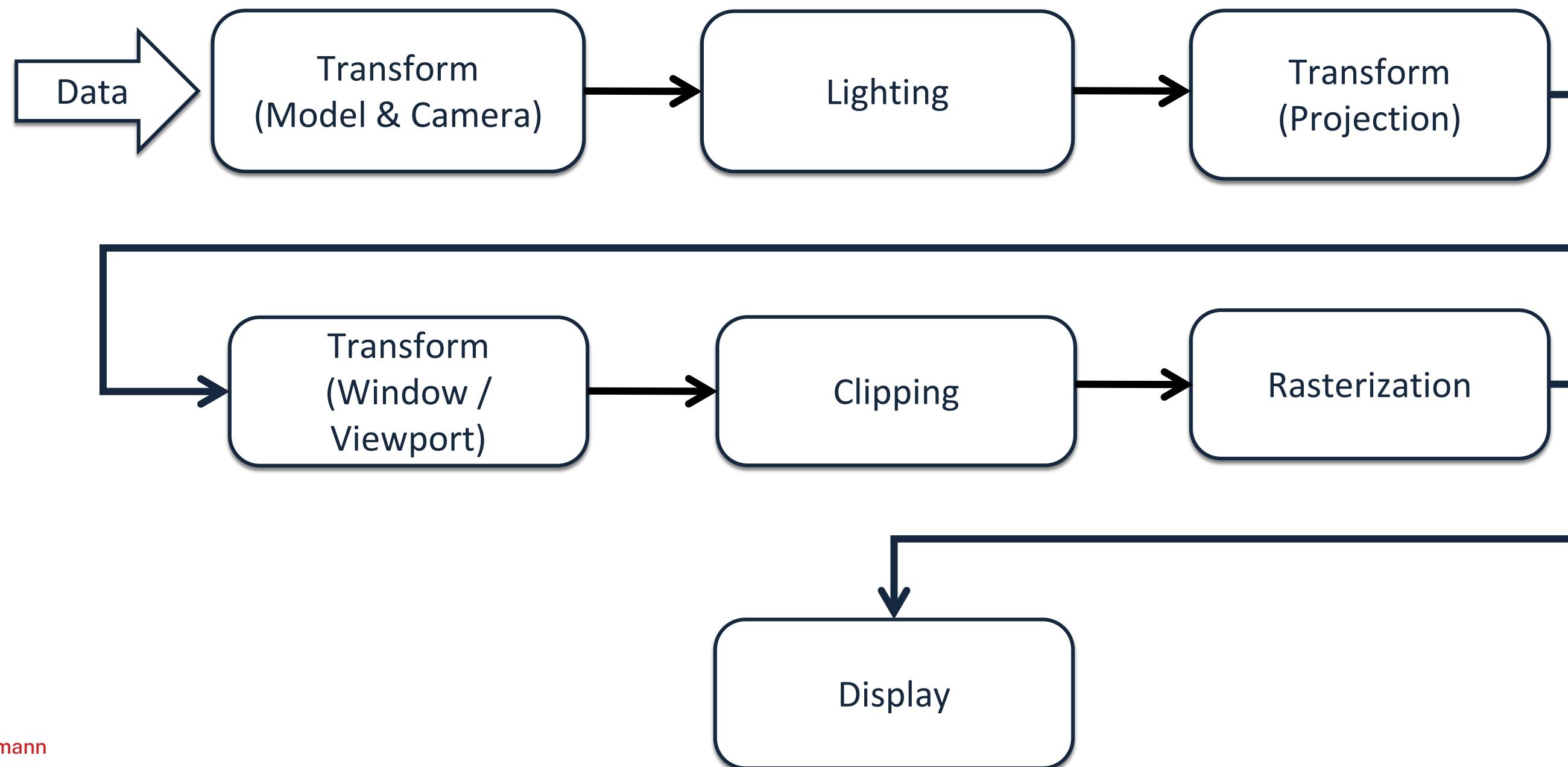
- Rendering
 - Creation of shaded images from 3D computer models given
 - A virtual camera
 - Objects (modeling)
 - Light sources
 - 2D images from 3D scenes
- Important issues
 - Visibility and Projection
 - Simulation of light
 - Interactivity vs. photorealism

The Rendering Pipeline

- **3D input**
 - Virtual camera
 - Position, orientation, focal length, ...
 - Objects
 - Points, lines, polygons and attributes
 - Light sources
 - Position, direction, color, ...
 - Textures
 - Images
- **2D output**
 - Per-pixel color values in the framebuffer

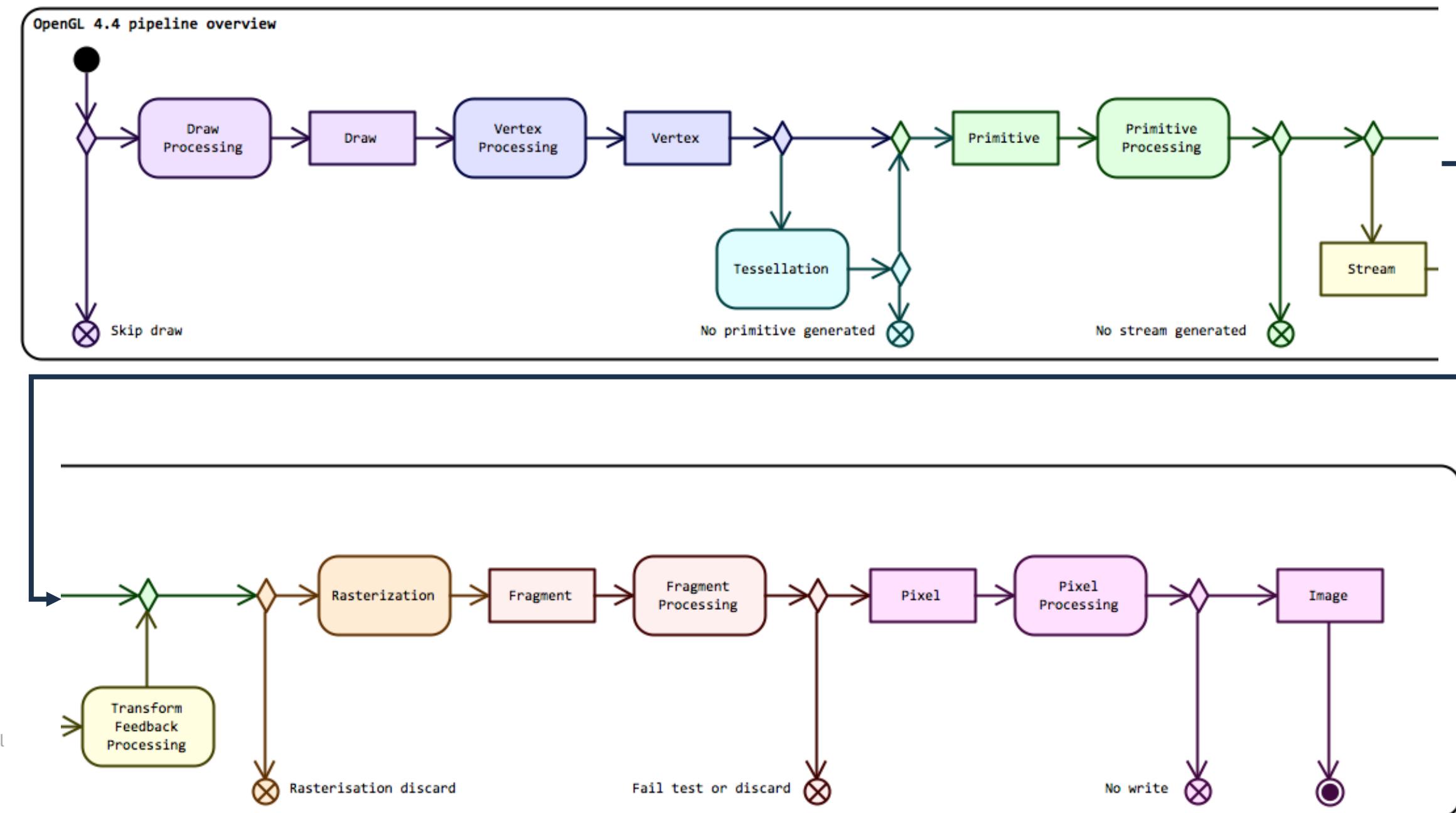
The Rendering Pipeline

Classic pipeline architecture



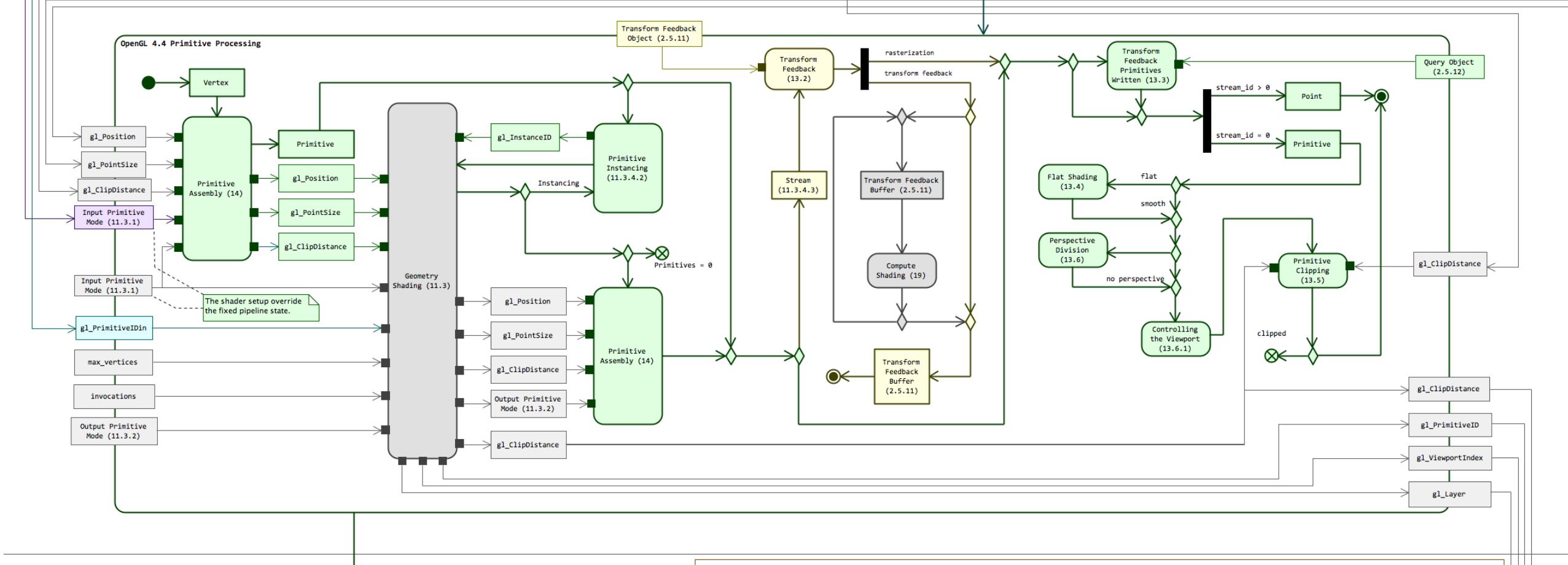
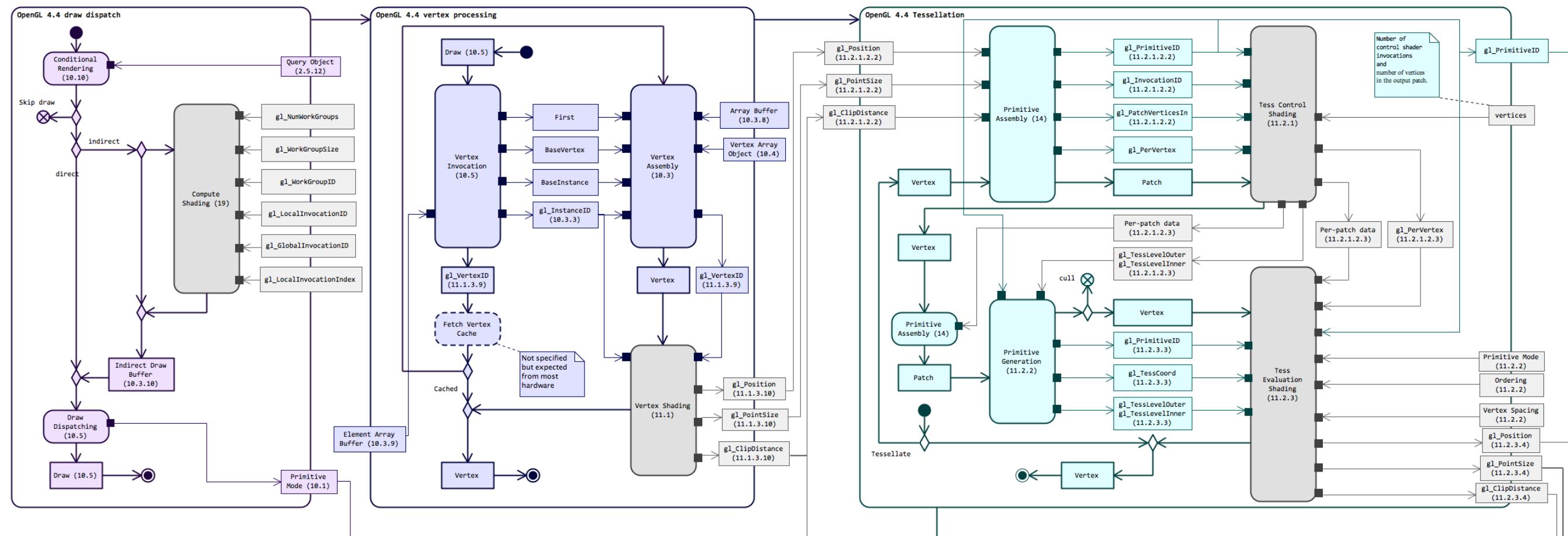
The Rendering Pipeline

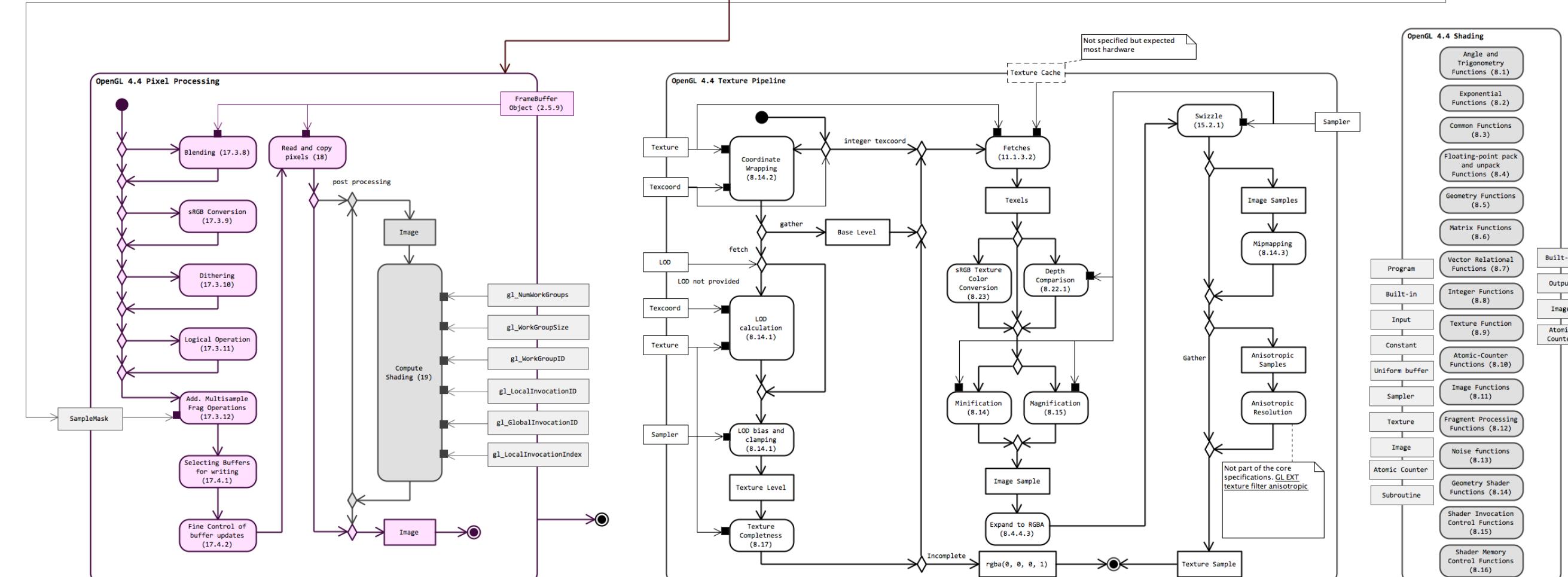
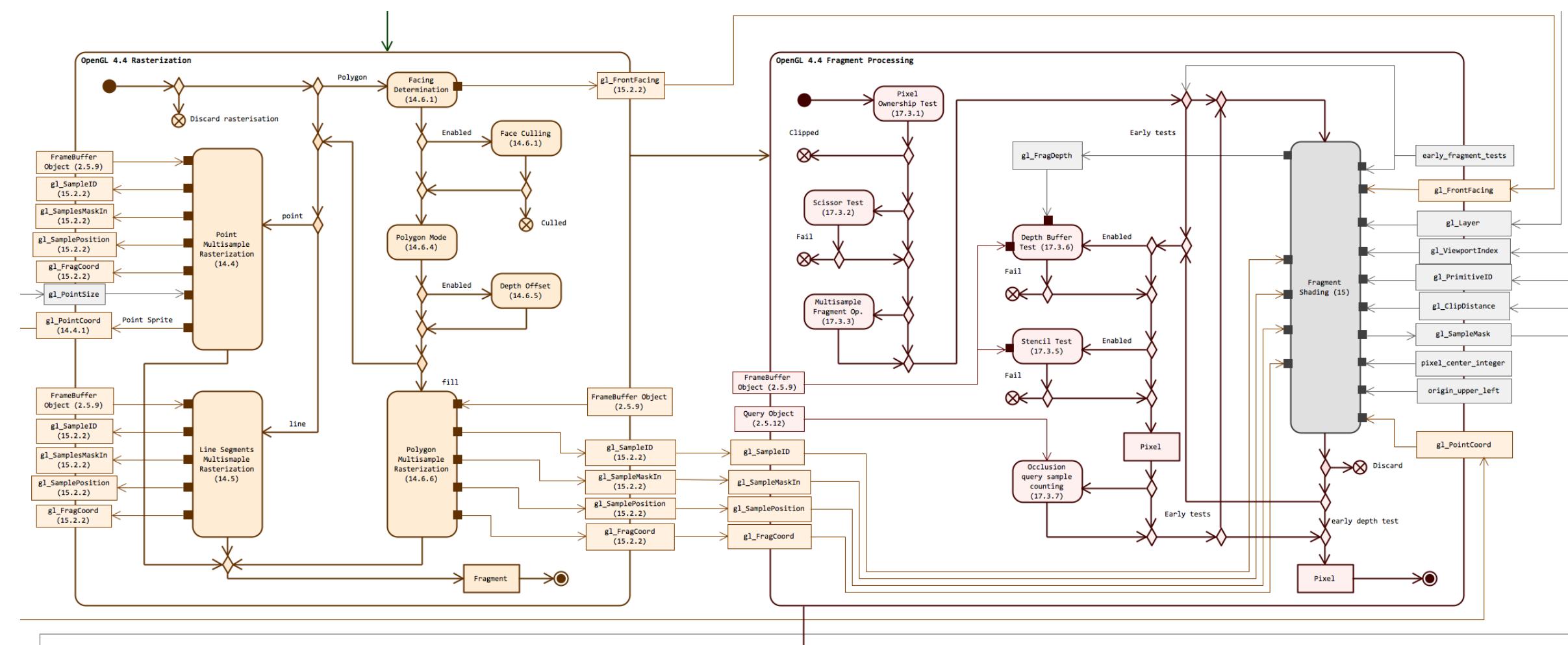
Current pipeline architecture (OpenGL 4.4)



Source: <http://openglinsights.com/pipeline.html>

Prof. Dr. Matthias Teßmann





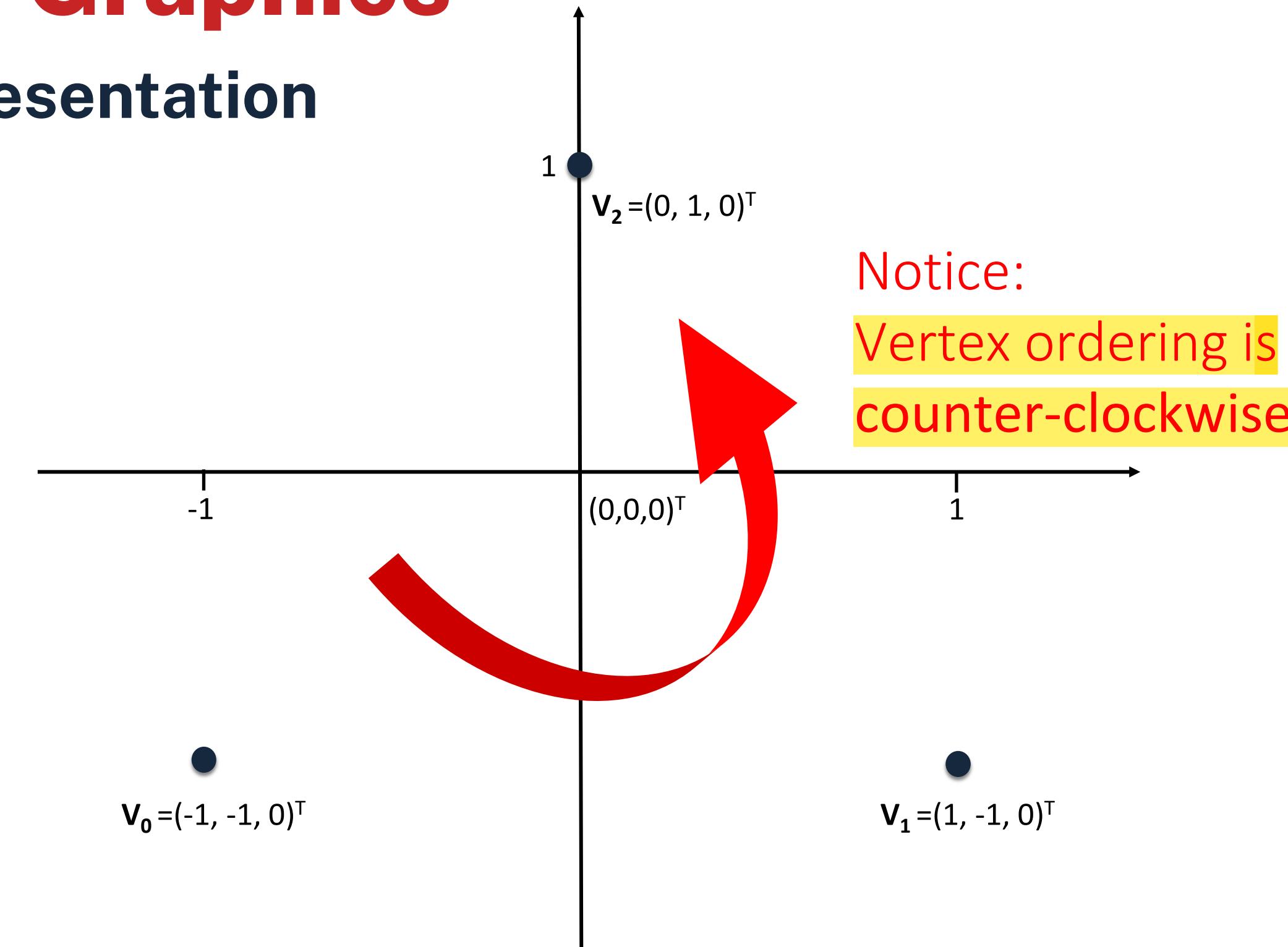
The Rendering Pipeline

Geometry representation

- Points (**vertices**) w.r.t. a known coordinate system
 - Vertices are used for modeling polygons (objects)
- Arbitrary polygons and surfaces are represented as triangles
 - Triangles, Triangle Strips, Triangle Fans - used to build meshes
 - Meshes form objects
- Attributes are associated with vertices
 - Color, material, surface normal vectors, ...
 - Used to calculate final appearance (pixel color)
- **Every single vertex is processed in the same way**
 - The series of processing steps is known as the rendering pipeline
 - This is what graphics hardware is for

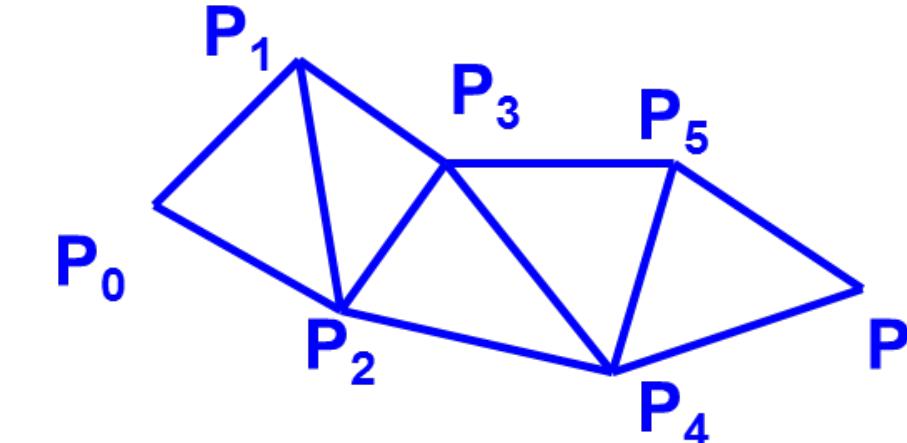
Computer Graphics

Geometry representation

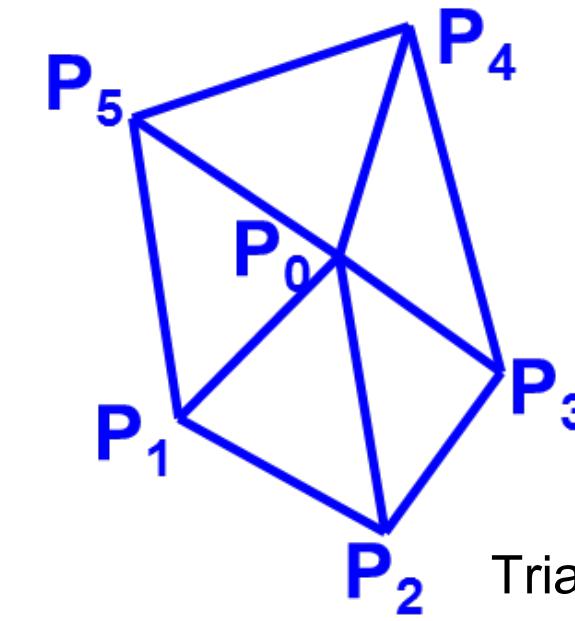


The Rendering Pipeline

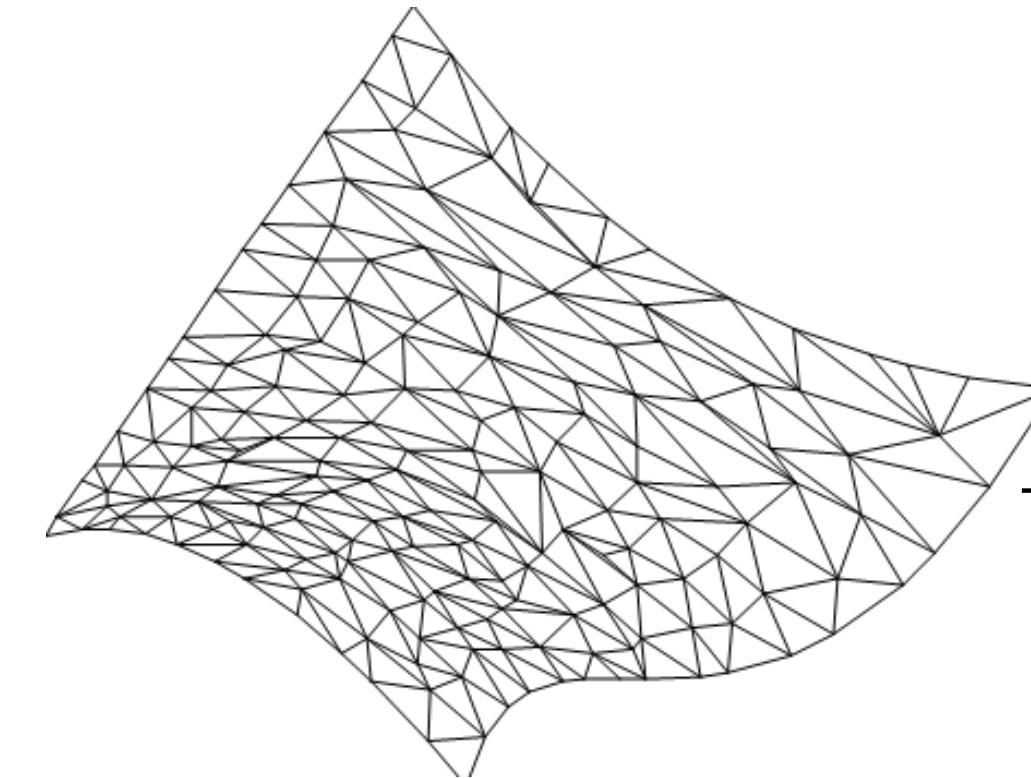
Geometry representation



Triangle strip



Triangle fan



Triangle mesh

The Rendering Pipeline

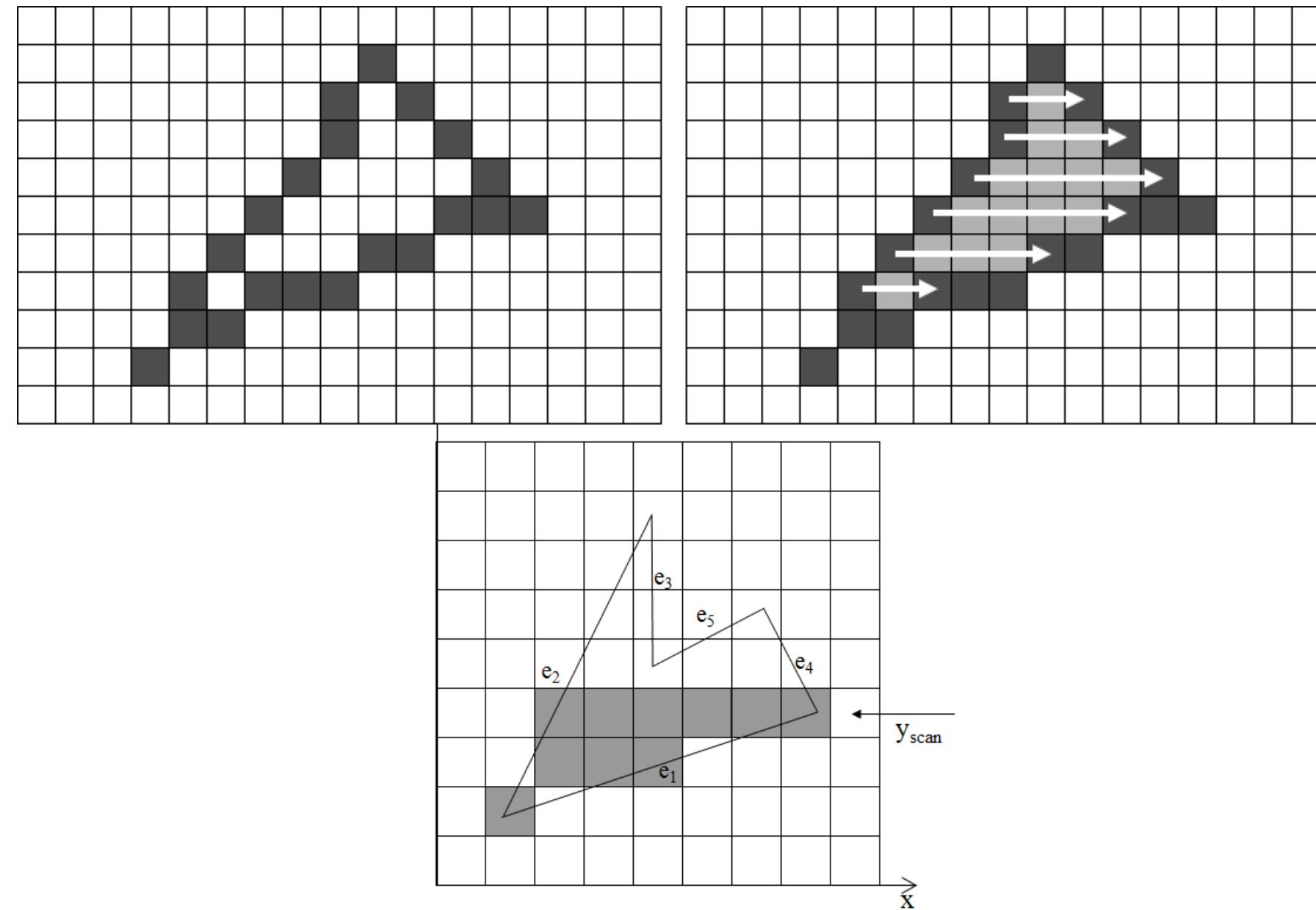
Rasterization

Object space	Image space
for all objects set pixel color	for all pixels for all objects calculate color contribution of object to pixel

- The framebuffer is the intermediate memory space storing the accumulated pixel values

The Rendering Pipeline

Rasterization



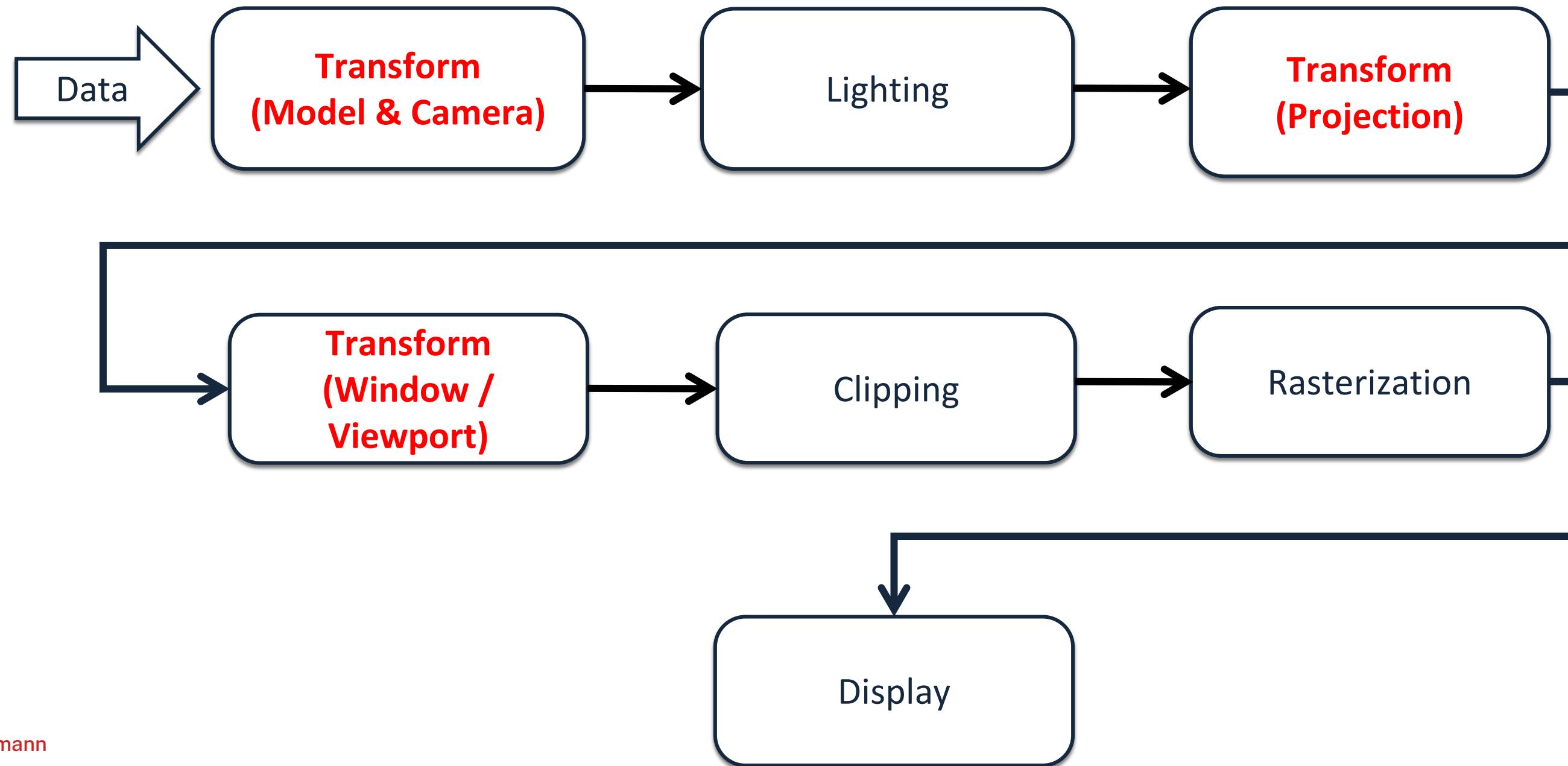
2.2 Transformations

Transformations

- Transformations are very important in CG and visualization!
 - Position objects in a scene (modeling)
 - Change shape of objects
 - Create copies of objects
 - Projections for virtual cameras
 - Position of projection on actual 2D window
 - Change of coordinate systems
 - Animations

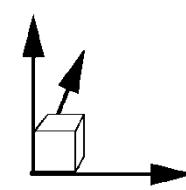
Transformations

Context: Rendering pipeline

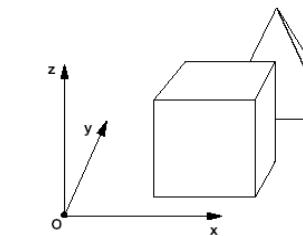
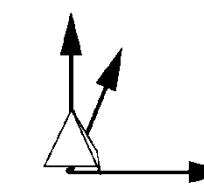


Transformations

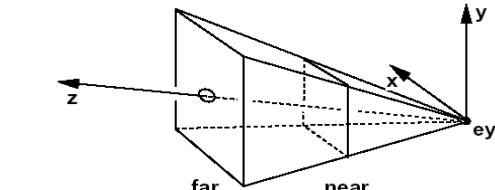
Spaces in CG applications



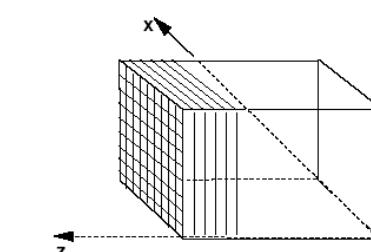
Object Space



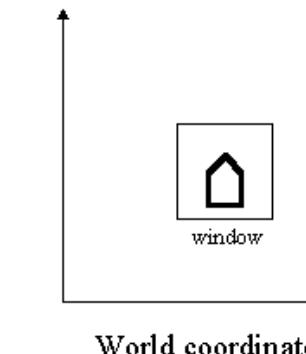
World Space



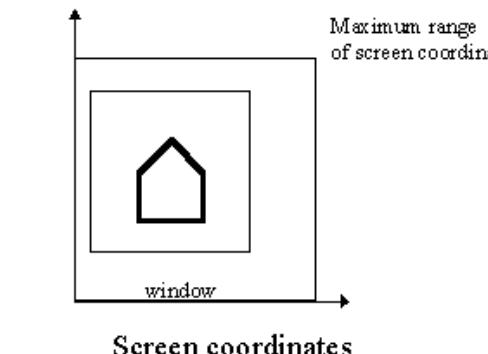
Camera Space



Viewing Space



World coordinates



Screen coordinates

Screen Space

Change spaces using transformations!

Transformations determine what is finally seen (visibility)

Transformations

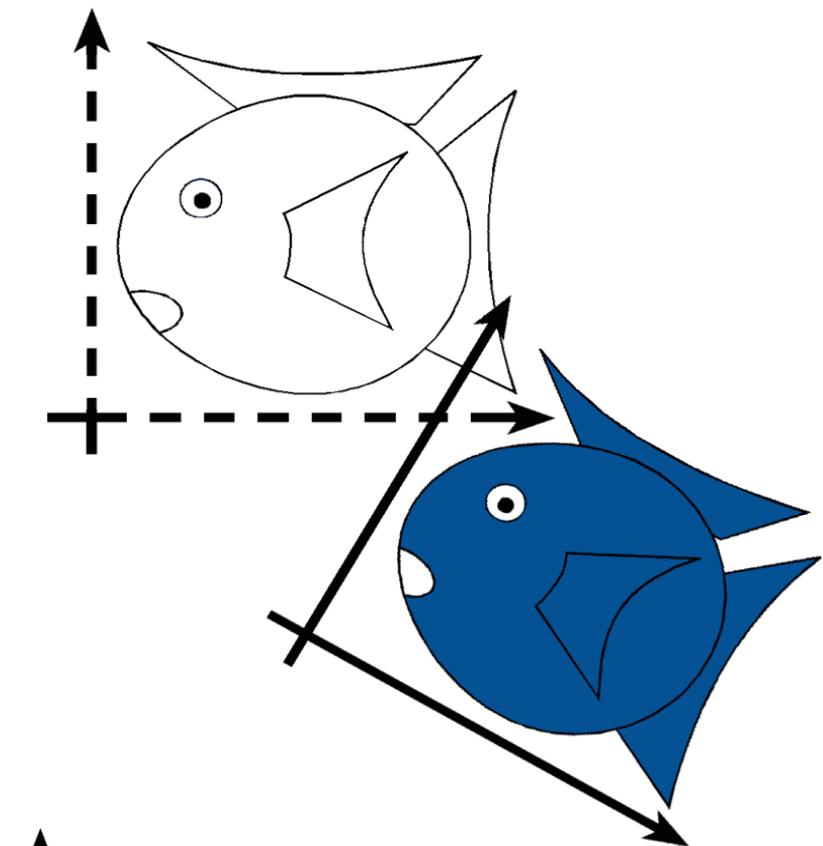
- Rigid Transformations (Euclidean Transform)

- Preserves distances
- Preserves angles

Rigid / Euclidean

Translation

Identity
Rotation



- Similarity Transformations

- Preserves angles
- Changes distances

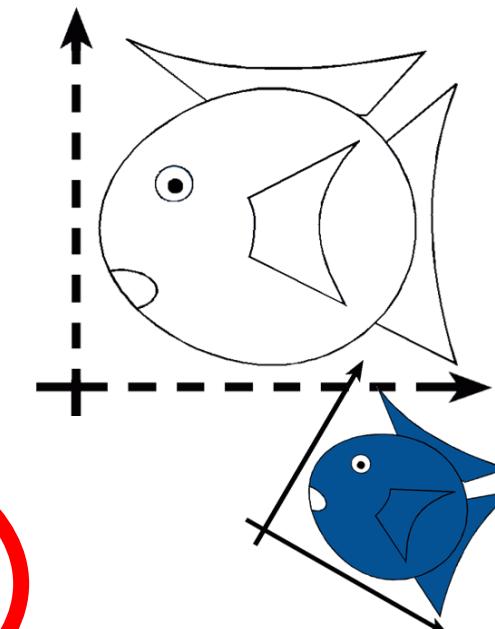
Similitudes

Rigid / Euclidean

Translation

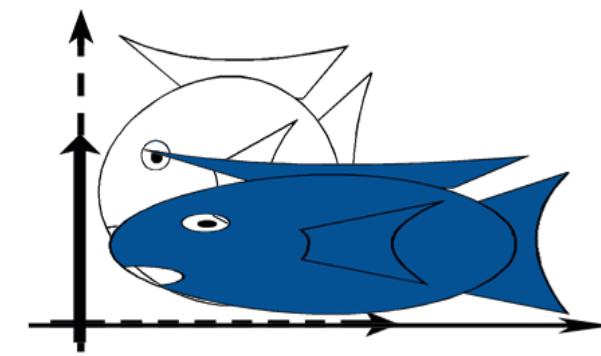
Identity
Rotation

Isotropic Scaling

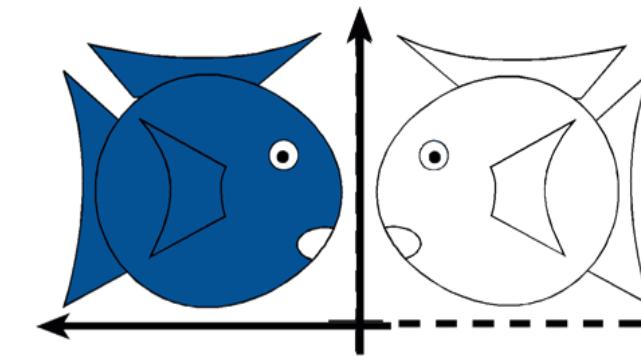


Transformations

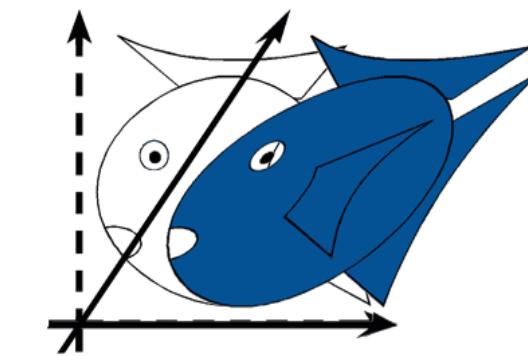
Linear Transformations



Scaling



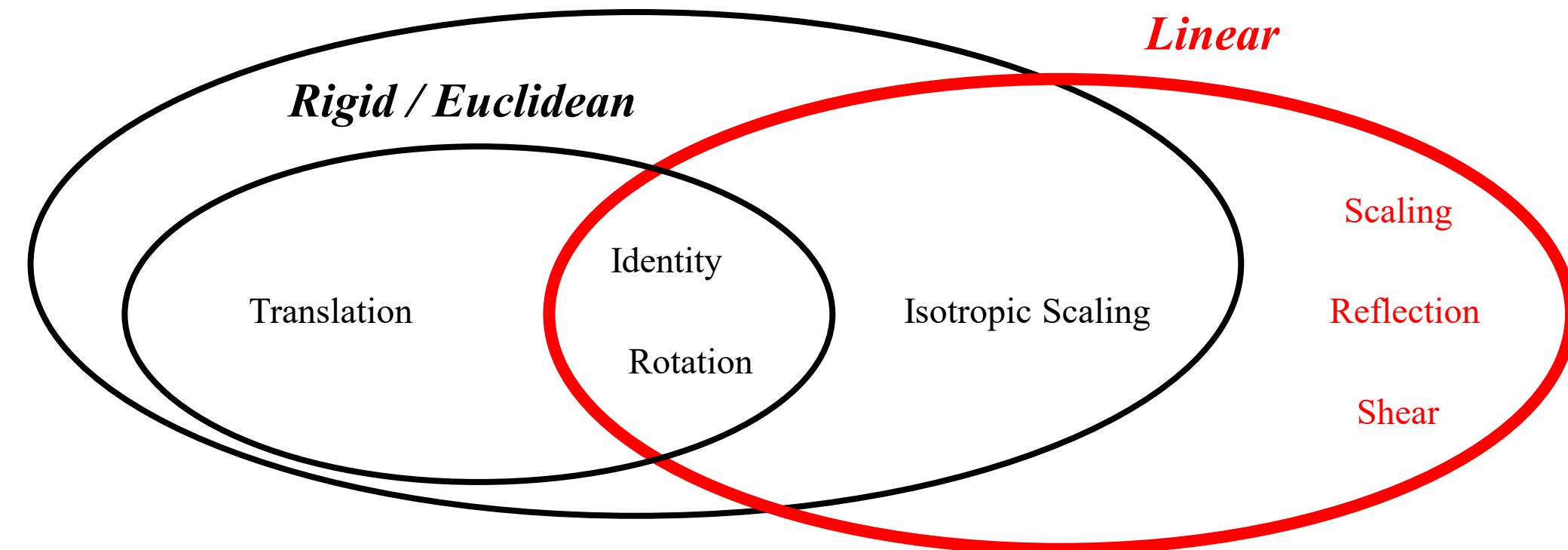
Reflection



Shear

Similarities

Linear



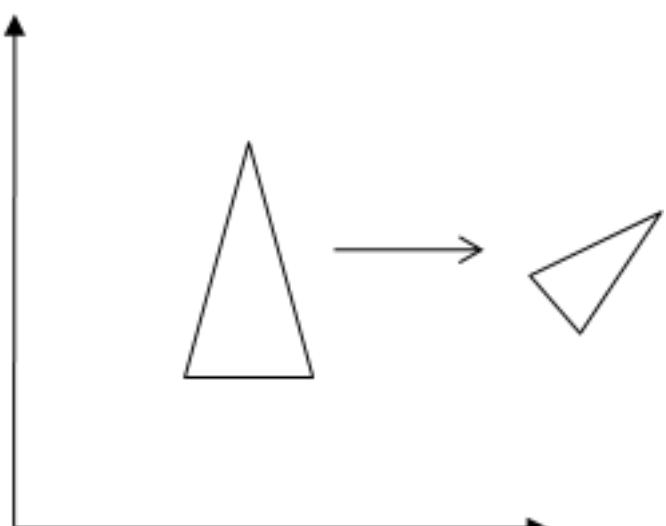
Transformations

Linear Transformations

- Two geometric objects, T_1 and T_2 are congruent if they can be made to coincide by a rigid motion and a positive or negative scaling, i.e. if there exists a scaling factor $c > 0$, a translation vector b and an *orthogonal* matrix Q such that

$$T_1 = b + cQT_2$$

T1 und T2 kongruent wenn sie sich umkehrbar lassen
bzw. vom einen in den anderen transformieren lassen



Transformations

Basic 2D Transformations

- Matrix multiplication
 - Scaling
 - Rotation
- Different operation of transformations on
 - Locations (points)
 - Displacement vectors
 - Normal vectors (surface normals)
- Matrices used for scale, rotation and shear

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{pmatrix}$$

Transformations

Scaling

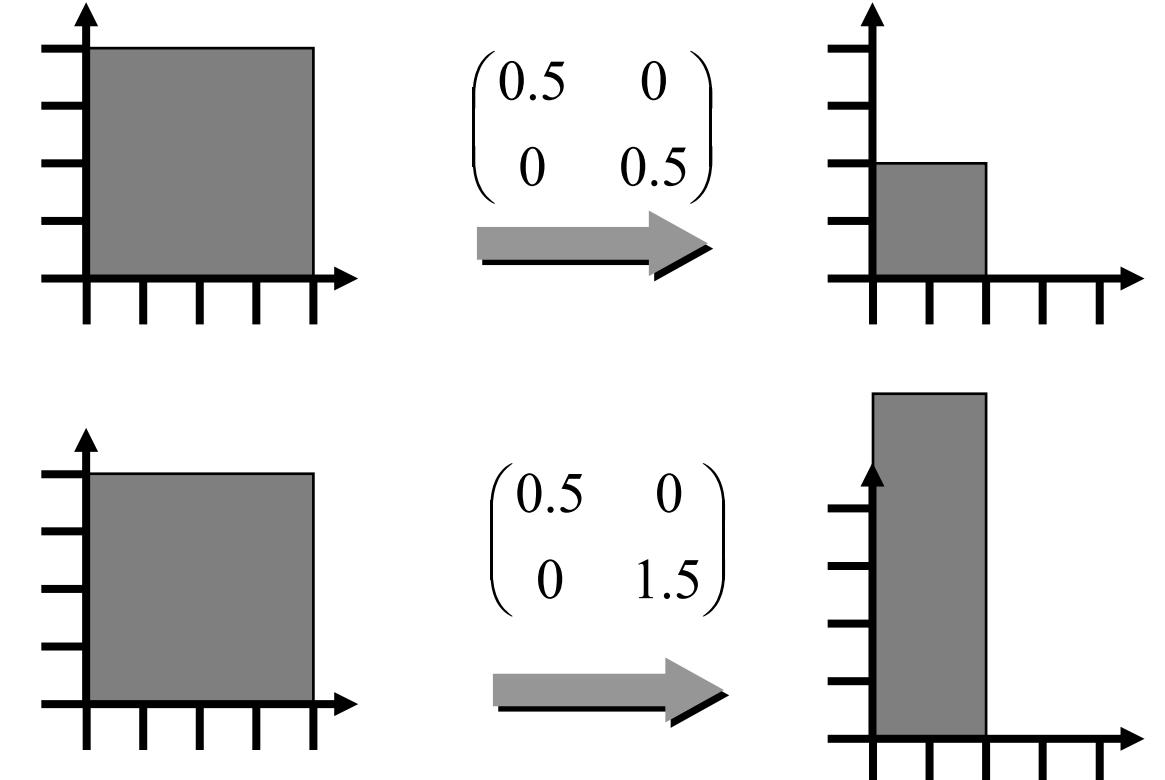
- Most basic transformation

$$scale(s_x, s_y) = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

- Change length and possibly direction

- Transformation of a vector

$$\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \end{pmatrix}$$



Transformations

Shearing

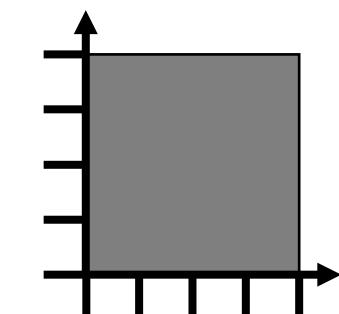
- Pushing things sideways (deck of cards)
- Horizontal (y-coordinate constant) and Vertical (x-coordinate constant)
- Can be expressed as rotation about an angle

$$\text{shear } -x(s) = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix}$$

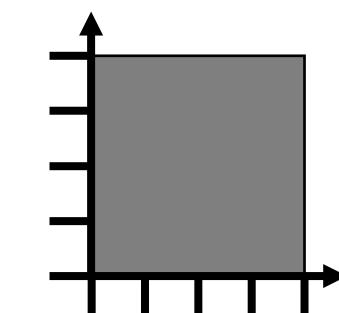
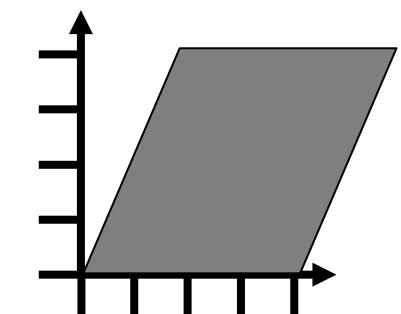
$$\text{shear } -y(s) = \begin{pmatrix} 1 & 0 \\ s & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & \tan\phi \\ 0 & 1 \end{pmatrix}$$

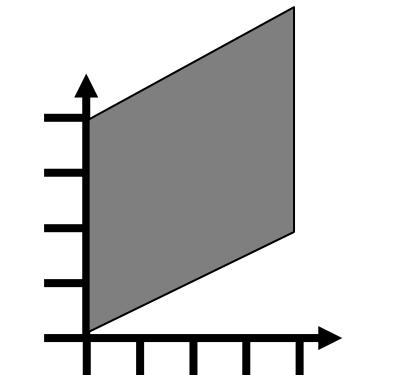
$$\begin{pmatrix} 1 & 0 \\ \tan\phi & 1 \end{pmatrix}$$



$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$



$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$



Transformations

Rotation

- Vector

$$\mathbf{a} = (a_x, a_y)$$

- Length

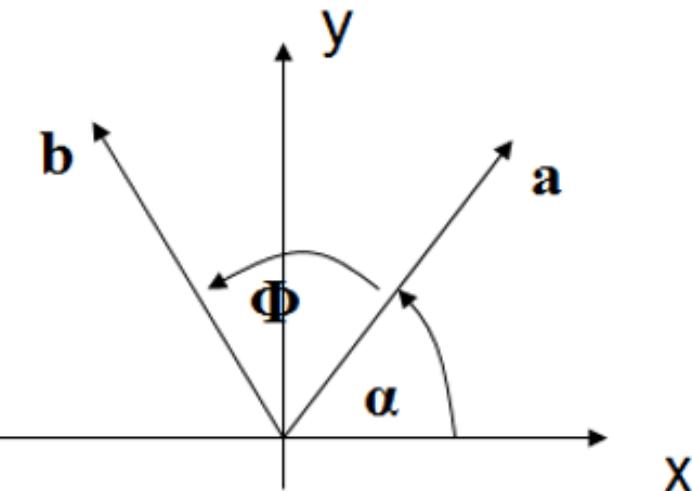
$$r = \sqrt{a_x^2 + a_y^2}$$

- By definition

$$a_x = r \cos \alpha$$

$$a_y = r \sin \alpha$$

- Rotation by an angle Φ , counter-clockwise



$$b_x = r \cos(\alpha + \varphi) = r \cos \alpha \cos \varphi - r \sin \alpha \sin \varphi$$

$$b_y = r \sin(\alpha + \varphi) = r \sin \alpha \cos \varphi + r \cos \alpha \sin \varphi$$

Transformations

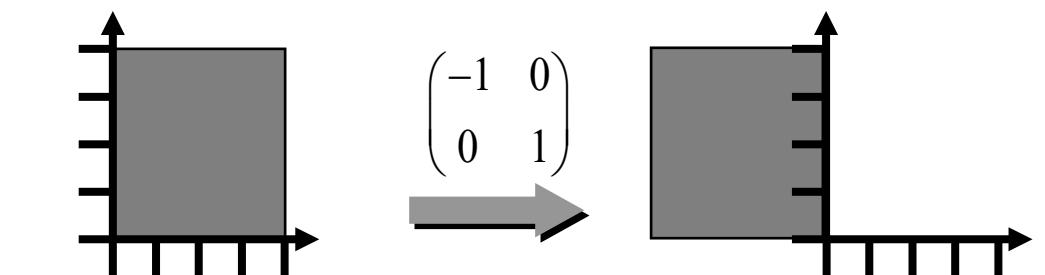
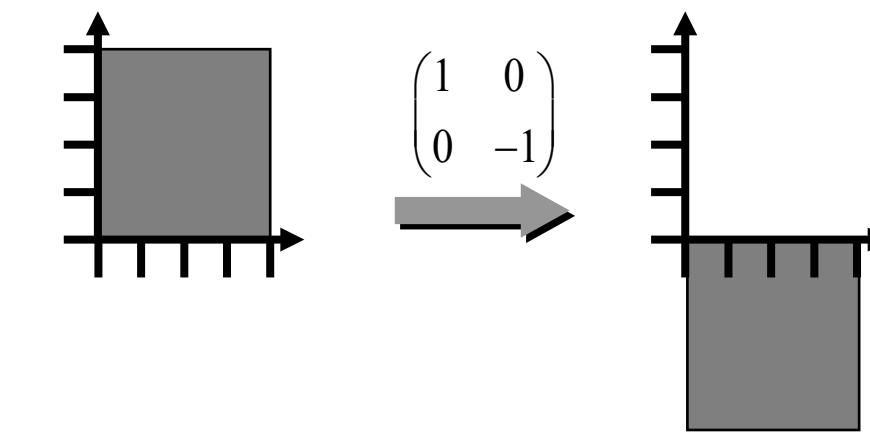
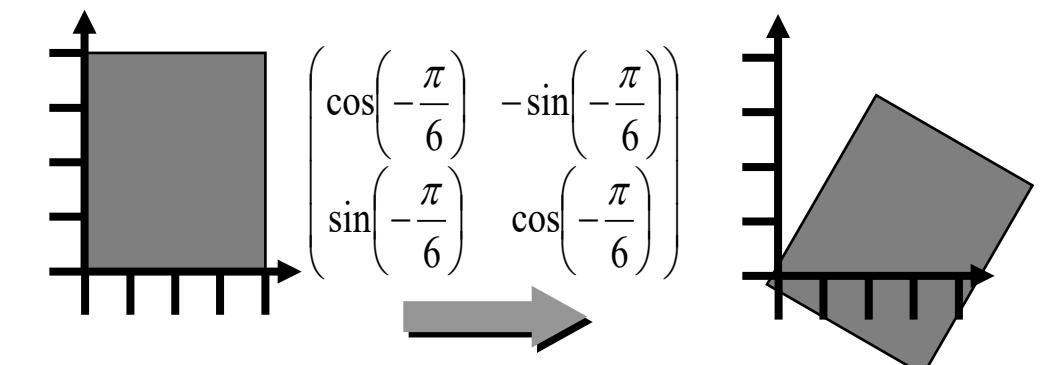
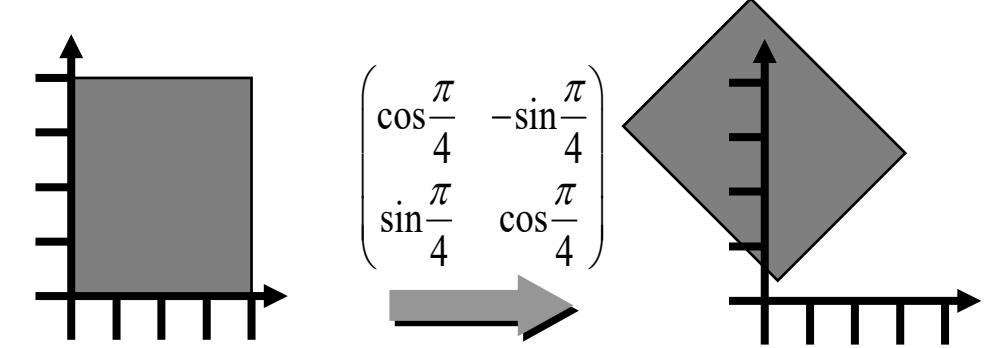
- Substitute

$$b_x = a_x \cos \varphi - a_y \sin \varphi$$

$$b_y = a_y \cos \varphi + a_x \sin \varphi$$

- Matrix from a to b

$$\text{rotate}(\phi) = R(\phi) = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix}$$

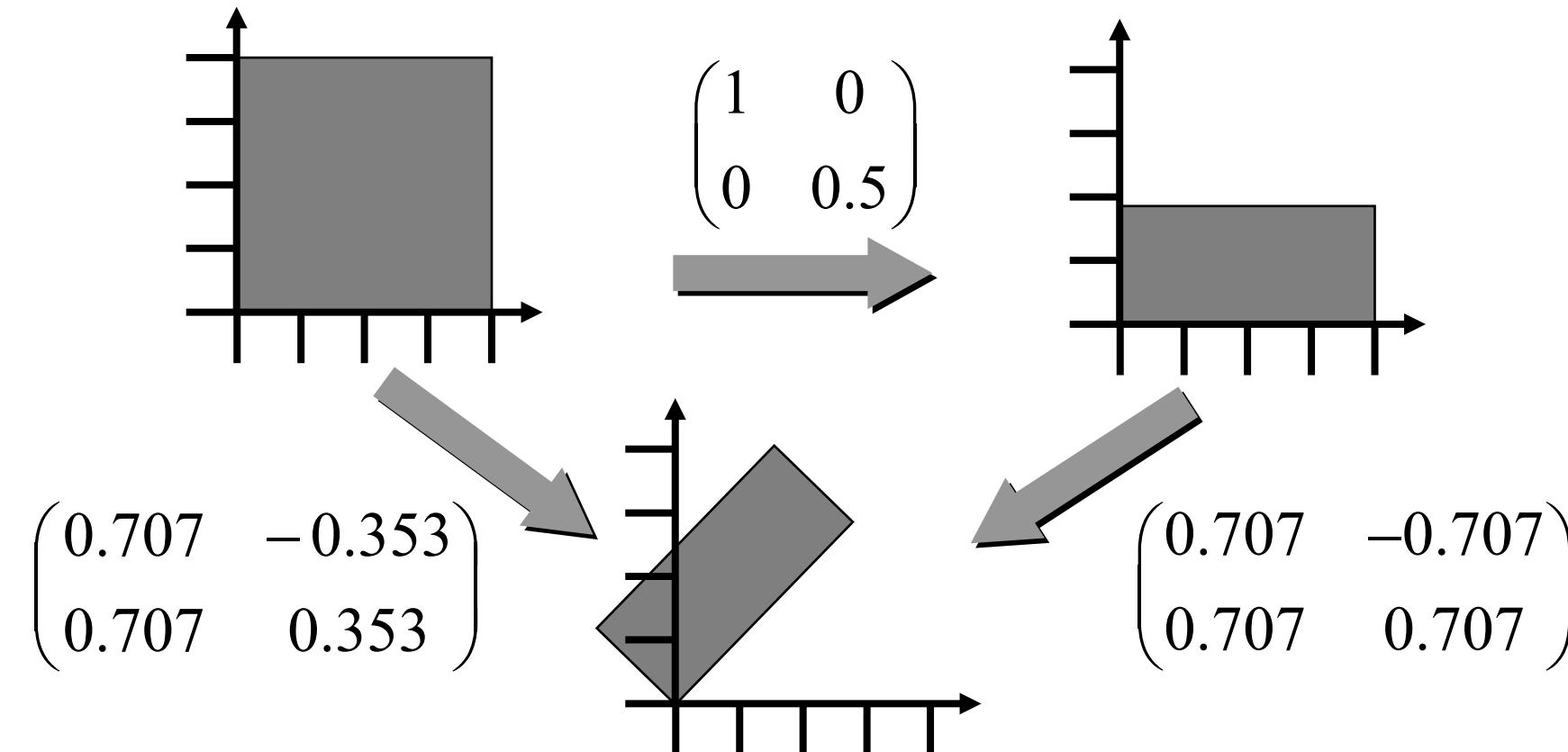


Transformations

Composition of 2D Transformations

- First $v_2 = S v_1$
- Second $v_3 = R v_2$

2. 1.



Transformations

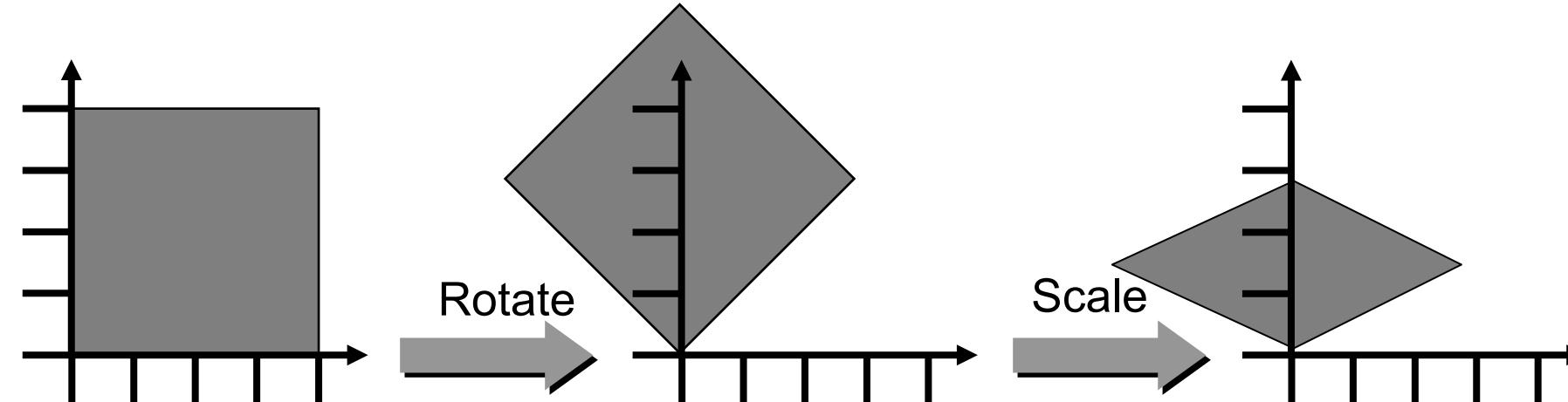
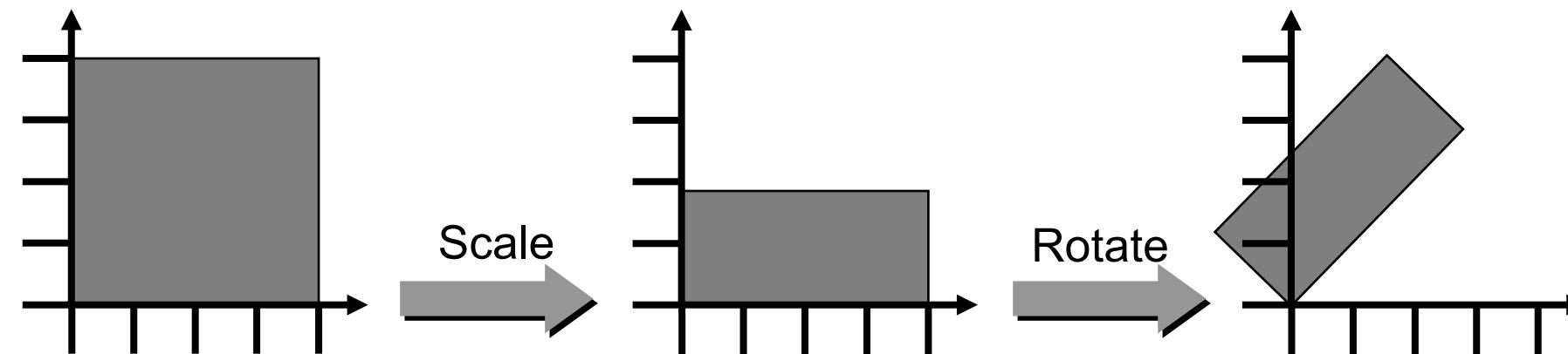
- Matrix multiplications are associative $(R S) T = R (S T)$

$$\Rightarrow \mathbf{v}_3 = (\mathbf{RS})\mathbf{v}_1 = \mathbf{M}\mathbf{v}_1$$

- Effects of two consecutive matrix multiplications by one matrix
 - Applied **from the right side** first! Here: first apply S, then R
- Matrix multiplications are *NOT commutative*
 - **The order of transformations does matter**
 - Note the difference
 - Scaling, then rotating vs. rotating, then scaling

Transformations

Order does matter



Transformations

3D Transformations

- Rotation about the main xyz-axis
 - Same applies to scale / shear

$$Rot_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{pmatrix}$$
$$Rot_y(\phi) = \begin{pmatrix} \cos \phi & 0 & \sin \phi \\ 0 & 1 & 0 \\ -\sin \phi & 0 & \cos \phi \end{pmatrix}$$
$$Rot_z(\phi) = \begin{pmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Transformations

Arbitrary 3D rotations

- 3D rotations by *orthogonal* matrices that preserve orientation
 - A matrix is orthogonal if

$$O^T \cdot O = O \cdot O^T = I_{n \times n} \text{ and } \det(O) = 1$$

- Matrix rows
 - Cartesian coordinates of three **mutually orthogonal** unit vectors
 - Matrix columns
 - Three potentially different **mutually orthogonal** unit vectors

Transformations

Change of base

- Let $\vec{u}, \vec{v}, \vec{w}$ form an orthonormal system

$$\vec{u} \cdot \vec{u} = \vec{v} \cdot \vec{v} = \vec{w} \cdot \vec{w} = 1$$

$$\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{w} = \vec{w} \cdot \vec{u} = 0$$

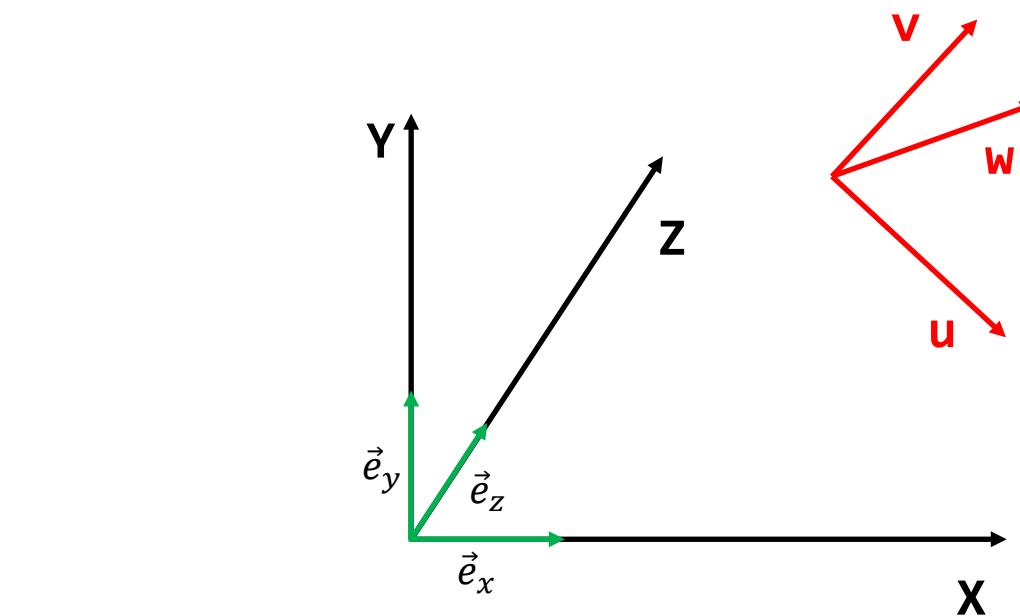
- with

$$\vec{u} = u_x \vec{e}_x + u_y \vec{e}_y + u_z \vec{e}_z$$

$$\vec{v} = v_x \vec{e}_x + v_y \vec{e}_y + v_z \vec{e}_z$$

$$\vec{w} = w_x \vec{e}_x + w_y \vec{e}_y + w_z \vec{e}_z$$

- then the associated rotation matrix is



$$R_{uvw} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$$

Transformations

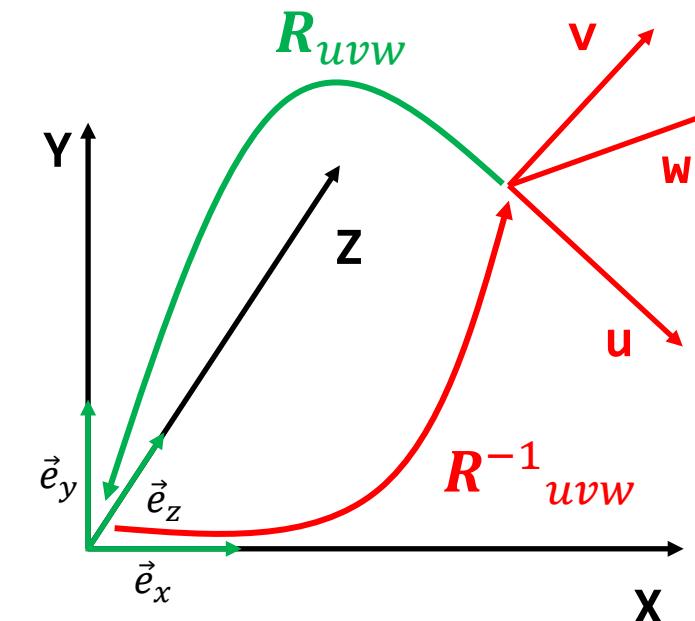
- The matrix R_{uvw}
 - Takes the basis uvw to the corresponding cartesian axis via rotation

$$R_{uvw} \vec{u} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \vec{u} \cdot \vec{u} \\ \vec{v} \cdot \vec{u} \\ \vec{w} \cdot \vec{u} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \vec{e}_x$$

$$R_{uvw} \vec{v} = \vec{e}_y$$

$$R_{uvw} \vec{w} = \vec{e}_z$$

- If R_{uvw} is a rotation matrix with orthonormal rows then R^T_{uvw} is a rotation matrix with orthonormal columns and $\Rightarrow R^T_{uvw} = R^{-1}_{uvw}$
- Consequence: change coordinate systems by using the basis vectors of an orthonormal system as columns of a rotation matrix

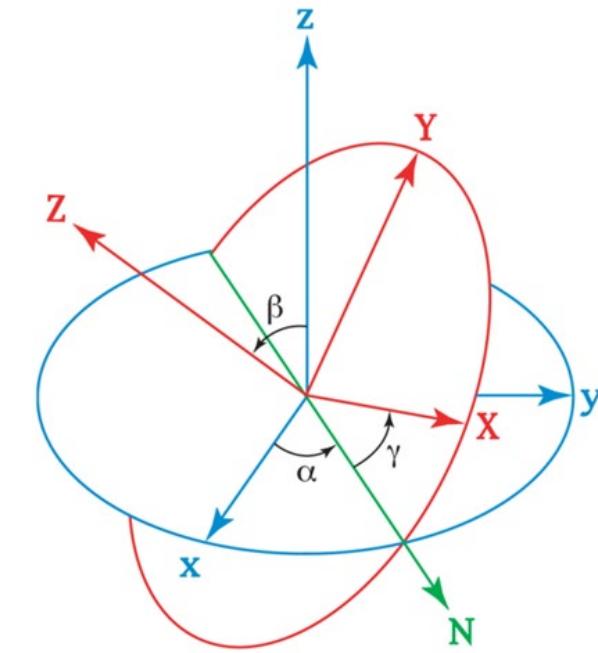


Transformations

Equivalent representations of rotations in 3D

- Orthogonal matrices
- 3 Euler rotations

$$\mathbf{R} = \mathbf{R}_z(\alpha) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\gamma) = \begin{pmatrix} \cos \beta \cos \alpha & \sin \gamma \sin \beta \cos \alpha - \cos \gamma \sin \alpha & \cos \gamma \sin \beta \cos \alpha + \sin \gamma \sin \alpha \\ \cos \beta \sin \alpha & \sin \gamma \sin \beta \sin \alpha + \cos \gamma \cos \alpha & \cos \gamma \sin \beta \sin \alpha - \sin \gamma \cos \alpha \\ -\sin \beta & \sin \gamma \cos \beta & \cos \gamma \cos \beta \end{pmatrix}$$



- Axis of rotation and angle
- Quaternions
- 2 (planar) reflections

Transformations

For orthogonal transformations

- Algebraic inverse = geometric inverse = transpose
- So, if R_{uvw} takes v to y then R_{uvw}^T takes y to v
- Construct rotation about arbitrary vector a
 - Form orthonormal basis with $w = a$
 - Rotate basis to canonical basis xyz
 - Rotate about z -axis
 - Rotate canonical basis back to uvw basis

$$\begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{pmatrix}$$

Transformations

Transformation of surface normal vectors

- Problem: Transformation of surface normals differs from transformation of underlying surface:

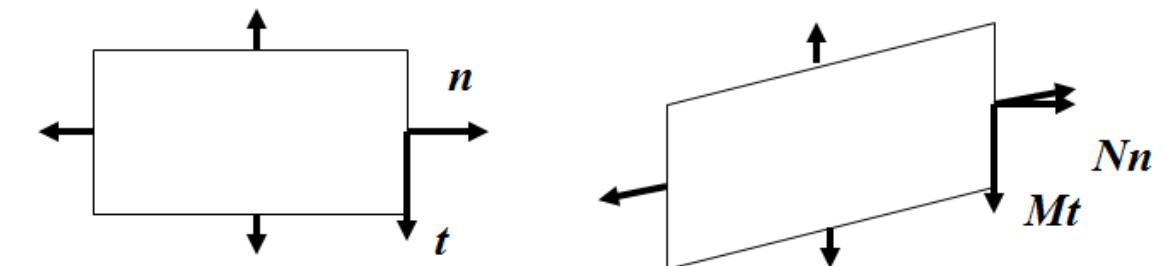
- We have: $\mathbf{n}^T \cdot \mathbf{t} = 0$ and $\mathbf{t}_M = \mathbf{M}\mathbf{t}$ and $\mathbf{n}_N = \mathbf{N}\mathbf{n}$

- Goal: find \mathbf{N} , such that $\mathbf{n}_{\mathbf{N}}^T \cdot \mathbf{t}_M = 0$

- And $\mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$, hence

$$\mathbf{n}^T \cdot \mathbf{t} = \mathbf{n}^T \mathbf{I}\mathbf{t} = \mathbf{n}^T \mathbf{M}^{-1}\mathbf{M}\mathbf{t} = 0$$

$$(\mathbf{n}^T \mathbf{M}^{-1}) \cdot (\mathbf{M}\mathbf{t}) = (\mathbf{n}^T \mathbf{M}^{-1}) \cdot \mathbf{t}_M = 0$$



$$\begin{aligned} &\Rightarrow \mathbf{n}_M^T = \mathbf{n}^T \mathbf{M}^{-1} \\ &\mathbf{n}_M = (\mathbf{n}^T \mathbf{M}^{-1})^T = (\mathbf{M}^{-1})^T \mathbf{n} \\ &\Rightarrow \boxed{\mathbf{N} = (\mathbf{M}^{-1})^T} \end{aligned}$$

Transform with the inverse transpose
of the original transformation matrix!

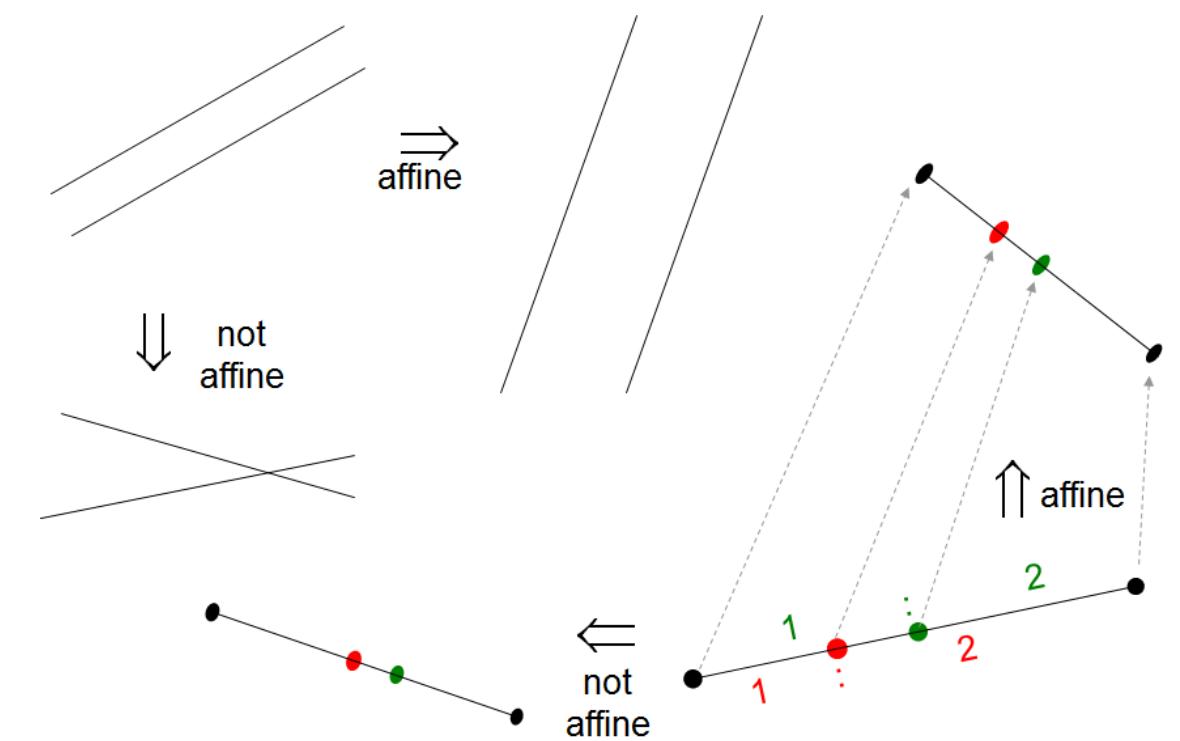
Affine Transformations

- Linear Transformation and Translation

$$\vec{x} \mapsto A\vec{x} + \vec{b} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

- Characterization

- Maps lines to lines
- Parallel lines will be mapped to parallel lines
- Division ratios will be kept
- Angles are not preserved
- Representation in CG: homogenous coordinates



Affine Transformations

- Dilemma 1

- Vector $\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$: describes a direction in space – independent of position
- You cannot "translate" direction vectors!

- Dilemma 2

$$T = T_1 T_2 T_3 \dots$$

- Linear transformations can be simply concatenated by matrix multiplication

$$T_1(\vec{x}) = \mathbf{M}_1 \vec{x} + \vec{t}_1 \quad T_2(\vec{x}) = \mathbf{M}_2 \vec{x} + \vec{t}_2$$

- Affine Transformations

$$\Rightarrow T_2(T_1(\vec{x})) = \mathbf{M}_2 \mathbf{M}_1 \vec{x} + \mathbf{M}_2 \vec{t}_1 + \vec{t}_2$$

Affine Transformations

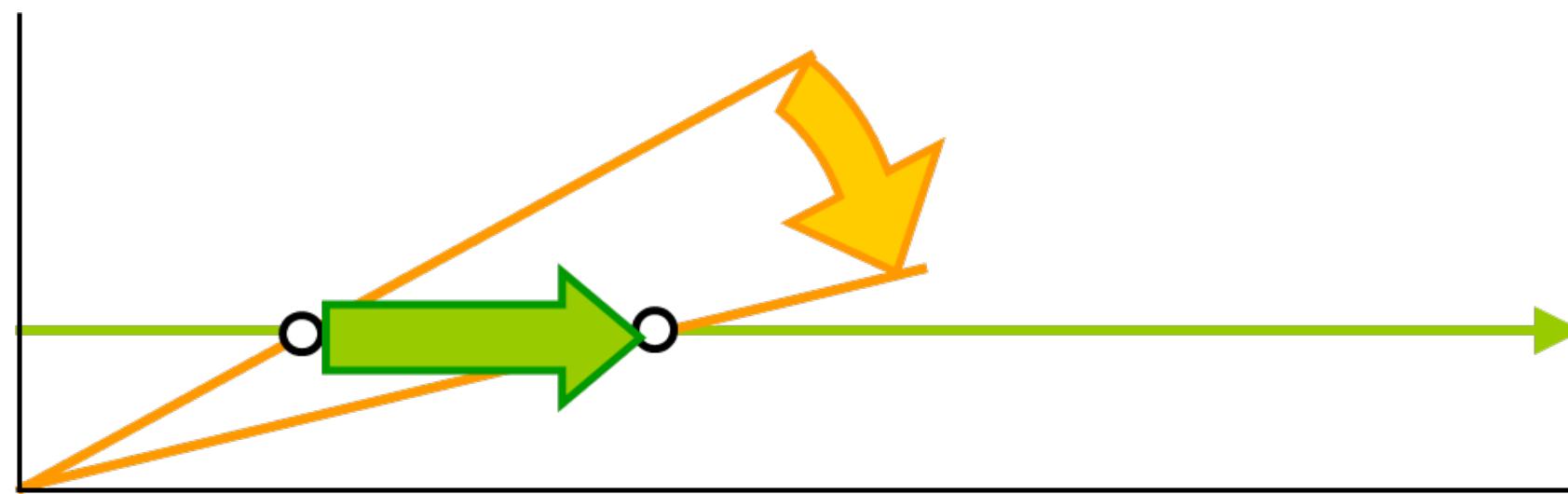
Homogenous coordinates (simple)

- Add third / fourth (2d/3d) coordinate w (1 for point, 0 for vector)
- Homogenous coordinates (advanced)
 - Identify (x,y) with the line $\{\alpha x, \alpha y, \alpha | \alpha \in \mathbb{R}\}$ in 3D or with any non-zero point on this line (e.g. $(x,y,1)$)
 - Consequence: $(x, y, 1)$, $(3x, 3y, 3)$, $(0.5x, 0.5y, 0.5)$ represent the same point!
 - Dehomogenization: $(x, y, w) \Rightarrow (\frac{x}{w}, \frac{y}{w})$
 - Analogous in 3D
- Mathematical foundation
- Projective geometry

$$\vec{p}_H = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \leftrightarrow \vec{p} = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

Affine Transformations

- Example: Point in 1D
 - Corresponds to ray in homogenous coordinates
- Rotation in homogenous coordinates
 - Corresponds to translation in cartesian coordinates



Affine Transformations

- What is w ?
 - For points: $w = 1$
 - For vectors: $w = 0$
 - For matrices
 - Add row $(0 \ 0 \ 0 \ 1)$
 - Rest of remaining column contains \vec{t}
- Result

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Note: homogenous transformation matrices can also be used for projection transformations

Affine Transformations

- Homogenous Coordinates: General form in 3D

$$\vec{x} \mapsto \mathbf{M}\vec{x} + \vec{b} \Rightarrow$$

$$= \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$= \begin{pmatrix} m_{11} & m_{12} & m_{13} & b_1 \\ m_{21} & m_{22} & m_{23} & b_2 \\ m_{31} & m_{32} & m_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Lin. Transf.
Translation

Affine Transformations

- Affine transformation of point \vec{p}

$$\vec{p}' = \mathbf{M}\vec{p} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} m_{00}x + m_{01}y + m_{02}z + t_x \cdot 1 \\ m_{10}x + m_{11}y + m_{12}z + t_y \cdot 1 \\ m_{20}x + m_{21}y + m_{22}z + t_z \cdot 1 \\ 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \end{pmatrix} = \begin{pmatrix} x' + t_x \\ y' + t_y \\ z' + t_z \\ 1 \end{pmatrix}$$

- Affine transformation of vector \vec{v}

$$\vec{v}' = \mathbf{M}\vec{v} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} m_{00}x + m_{01}y + m_{02}z + t_x \cdot 0 \\ m_{10}x + m_{11}y + m_{12}z + t_y \cdot 0 \\ m_{20}x + m_{21}y + m_{22}z + t_z \cdot 0 \\ 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \cdot 0 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 0 \end{pmatrix}$$

- Note how direction vectors are not translated!

Affine Transformations

Concatenation of transformations

- Regular (cartesian coordinates)

$$T_1(\vec{x}) = \mathbf{M}_1 \vec{x} + \vec{t}_1 \quad T_2(\vec{x}) = \mathbf{M}_2 \vec{x} + \vec{t}_2$$

$$\Rightarrow T_2(T_1(\vec{x})) = \mathbf{M}_2 \mathbf{M}_1 \vec{x} + \mathbf{M}_2 \vec{t}_1 + \vec{t}_2$$

- Homogenous coordinates

$$T_1(\vec{x}) = \mathbf{M}_1 \vec{x} \quad T_2(\vec{x}) = \mathbf{M}_2 \vec{x}$$

$$\Rightarrow T_2(T_1(\vec{x})) = \mathbf{M}_2 \mathbf{M}_1 \vec{x}$$

Affine Transformations

Transformation rules

- Multiplication is composition

$$x \xrightarrow{T} Tx = y \xrightarrow{S} Sy = z \quad \equiv \quad z = A_S \cdot A_T \cdot x$$

- *Inverse matrix* will inverse the transformation (inverse transformation)
 - Affine transformations are invertible
 - Caution: projection transformations are not generally invertible!
 - But also expressed as a homogenous matrix

Affine Transformations

Common transformations

- Translation and scaling

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{s}(a, b, c) = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation around the x-, y- and z-axis (note: φ is in radians)

$$\mathbf{R}_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & -\sin \varphi & 0 \\ 0 & \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{R}_y(\varphi) = \begin{pmatrix} \cos \varphi & 0 & \sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{R}_z(\varphi) = \begin{pmatrix} \cos \varphi & -\sin \varphi & 0 & 0 \\ \sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Affine Transformations

Example of a general transformation in 2D

- Rotate with center at $c=(c_x, c_y)$ by angle ϕ
- $\text{Transl}(-c) \rightarrow \text{Rot}(\phi) \rightarrow \text{Transl}(c)$

- in den Ursprung transformieren um $-c$
- rotieren
- zurück verschieben auf c

$$\begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos \varphi & -\sin \varphi & 0 \\ \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{pmatrix}$$

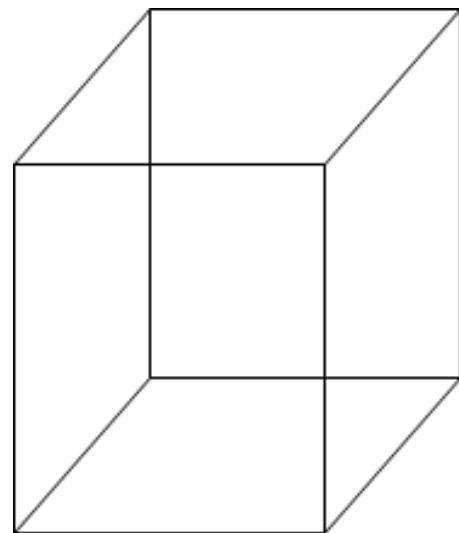
$$\Rightarrow \begin{pmatrix} \cos \varphi & -\sin \varphi & (c_y \sin \varphi - c_x \cos \varphi) + c_x \\ \sin \varphi & \cos \varphi & (-c_x \sin \varphi - c_y \cos \varphi) + c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Projections

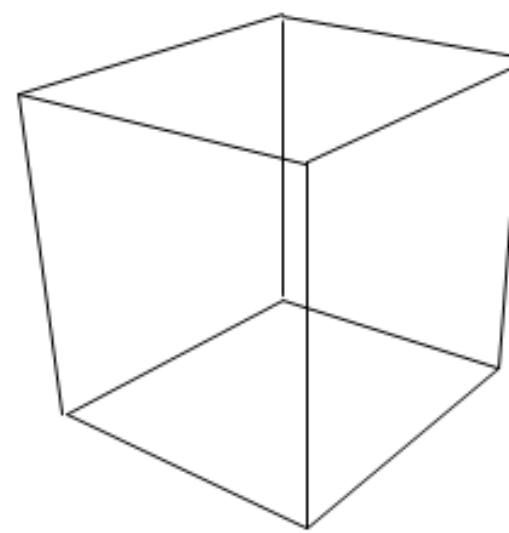
Projections

Viewing

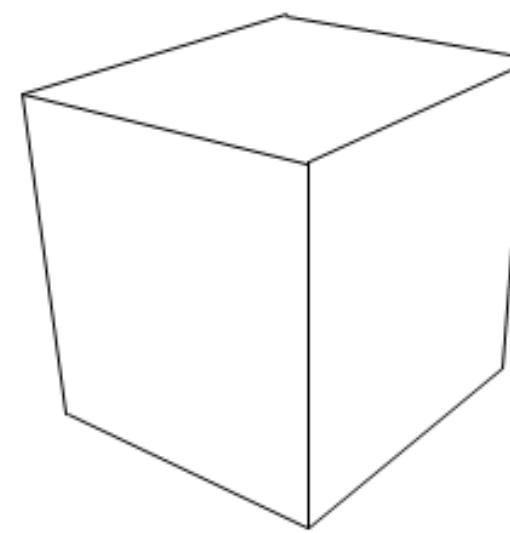
- Orthographic projection: parallel lines map to parallel lines
- Perspective projection: have 1, 2 or 3 vanishing points



orthographic
projection



perspective
projection

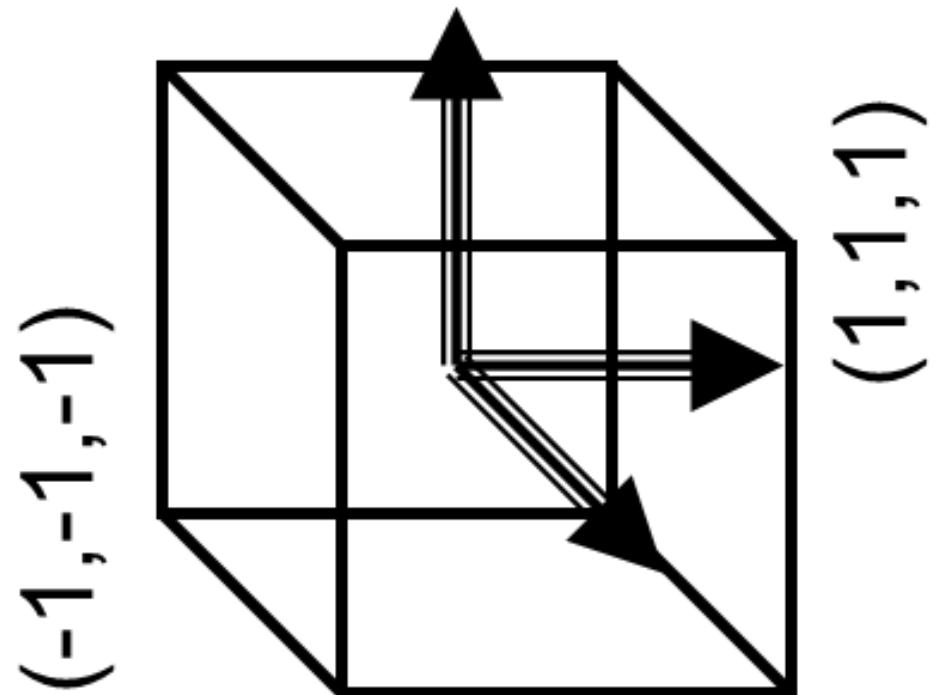


perspective
projection
with hidden line removal

Projections

The canonical view volume

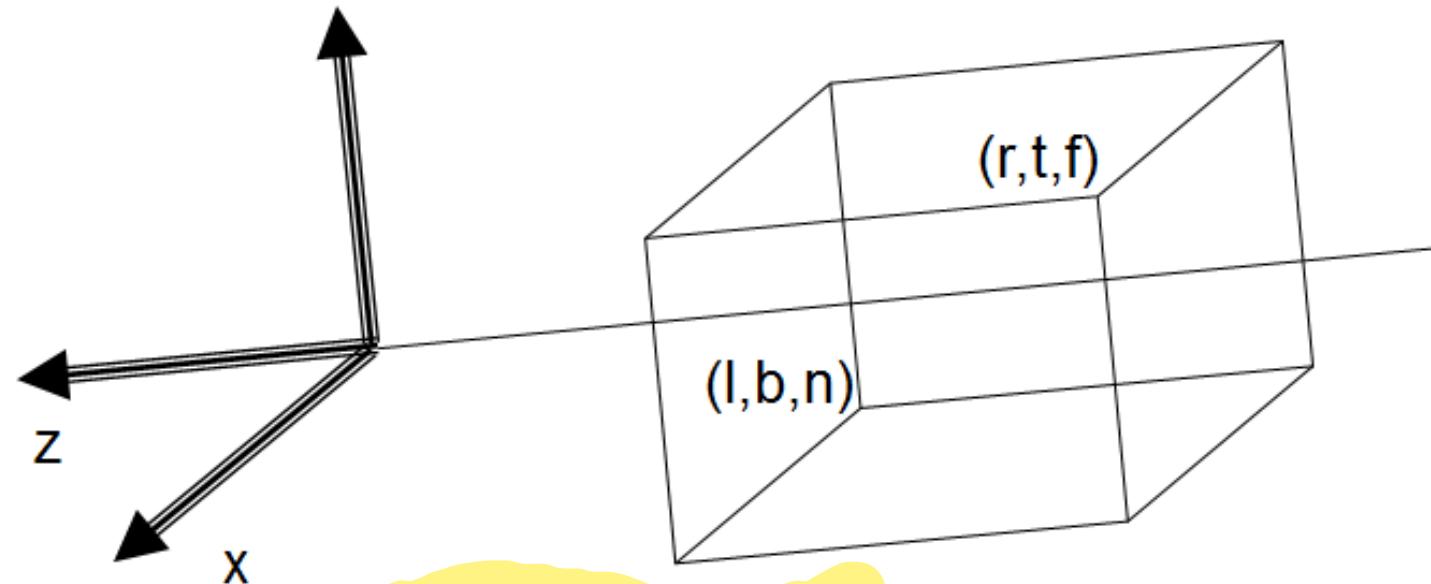
- Basic problem: map lines to the screen
 - Reuse solution for any viewing condition
- Limit to canonical view volume (x,y,z) in $[-1, 1]^3$
- If screen has $nx \times ny$ pixels
 - $x = -1 \Rightarrow$ left side
 - $x = +1 \Rightarrow$ right side
 - $y = +1 \Rightarrow$ top
 - $y = -1 \Rightarrow$ bottom
- Note the mapping: "square" \Rightarrow rectangle
- *Projections transform the scene into the canonical view volume*



Projections

Orthographic projection

- Viewer looks in direction of negative z-axis
- y-direction: view up
- x-axis: pointing right (right-handed coordinate system)



Note $\Rightarrow n > f$

$$M_{ortho} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Projections

Orthographic projection dissected

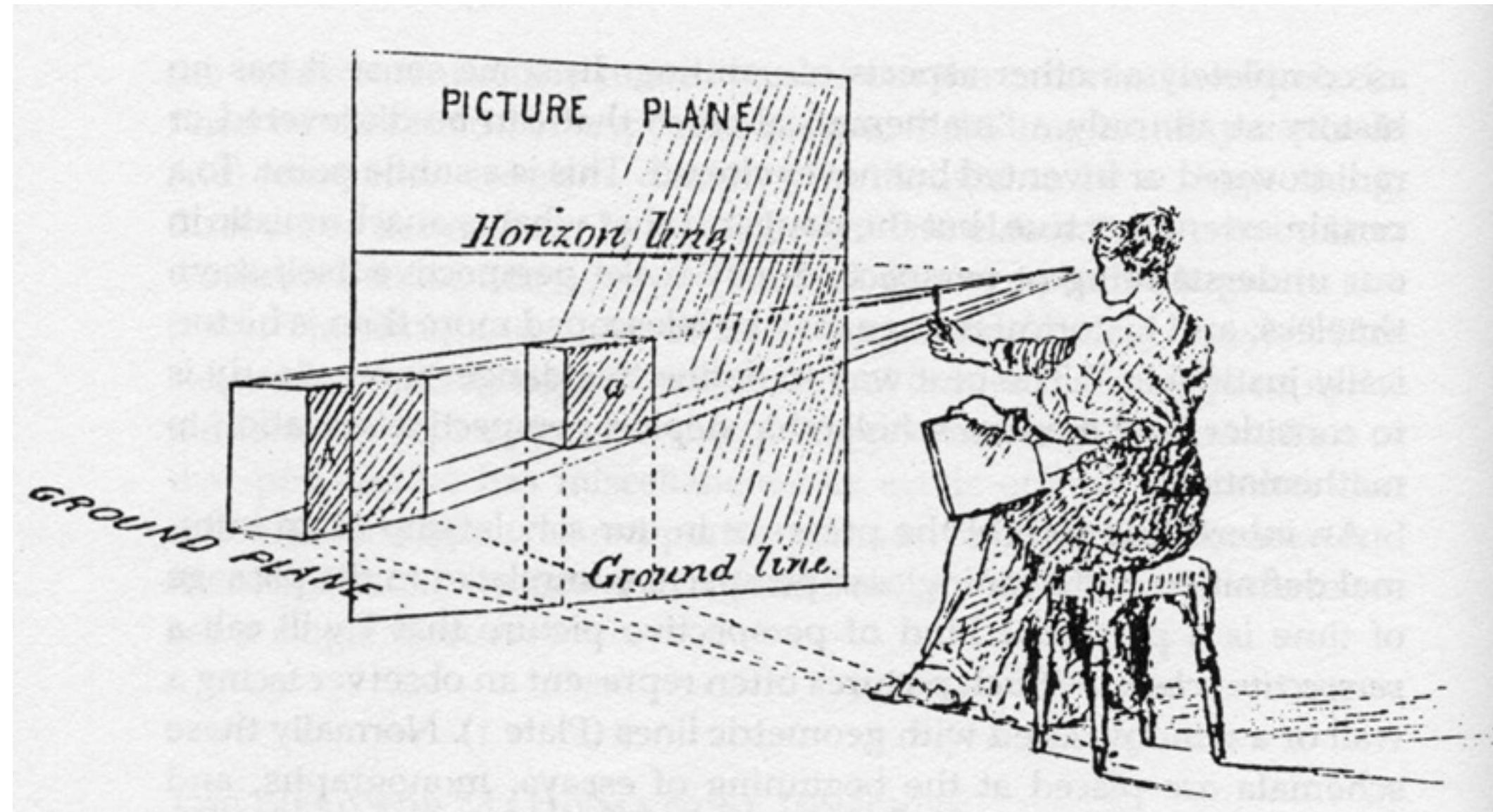
- First: move to the origin
- Second: scale to match canonical view volume

$$\begin{pmatrix} x_{canonical} \\ y_{canonical} \\ z_{canonical} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Scale **Translation**

Projections

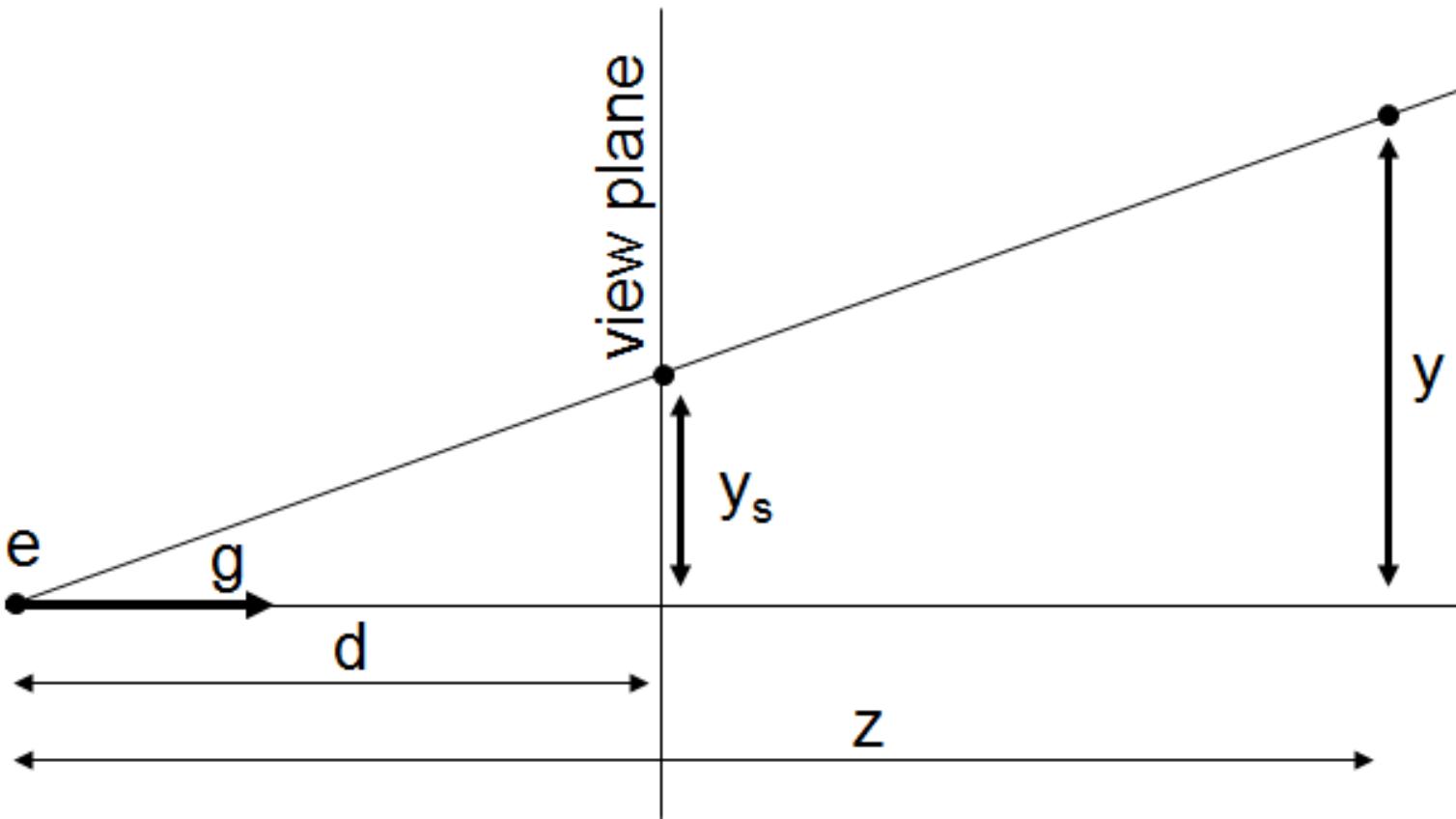
Perspective projection



Projections

Perspective projection

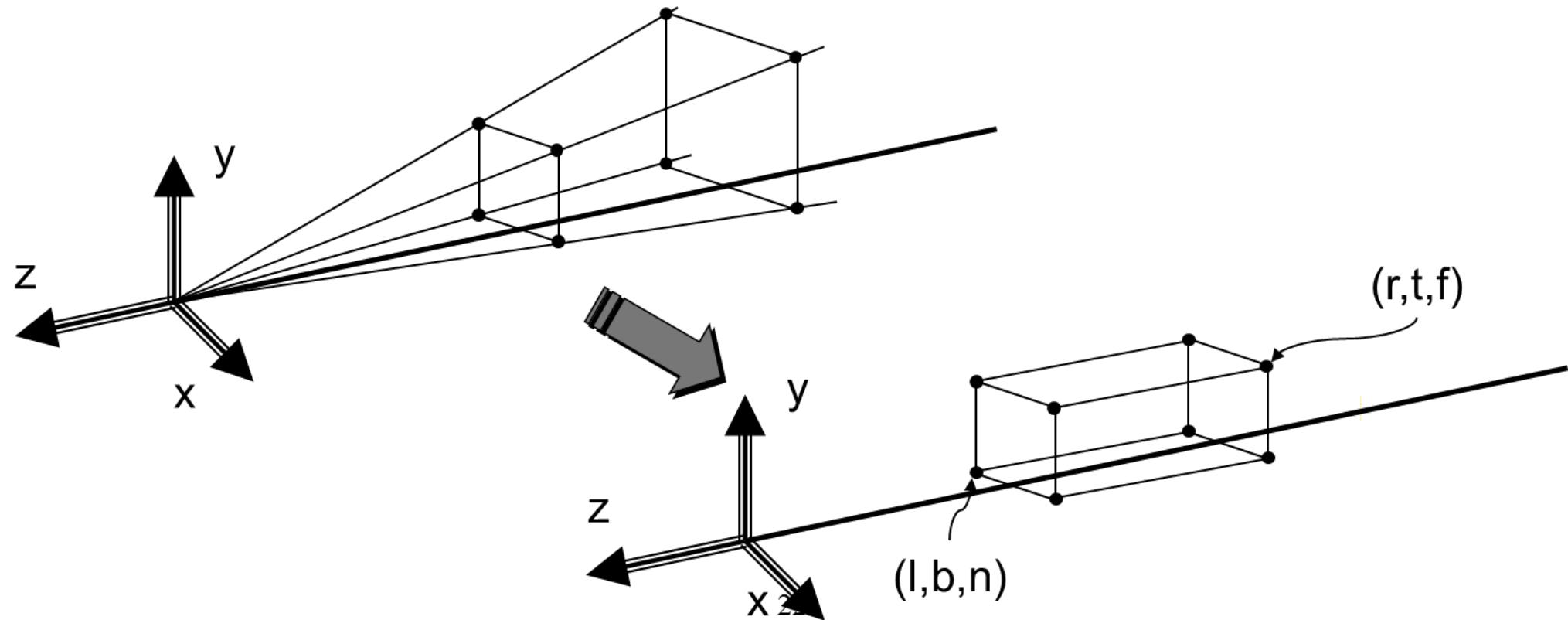
$$y_s = \frac{d}{z} y$$



Projections

Perspective projection

- Leave points on $z = n$ plane (view plane)
- Between $z = n$ and $z = f$ lines through eye point become parallel to z -axis

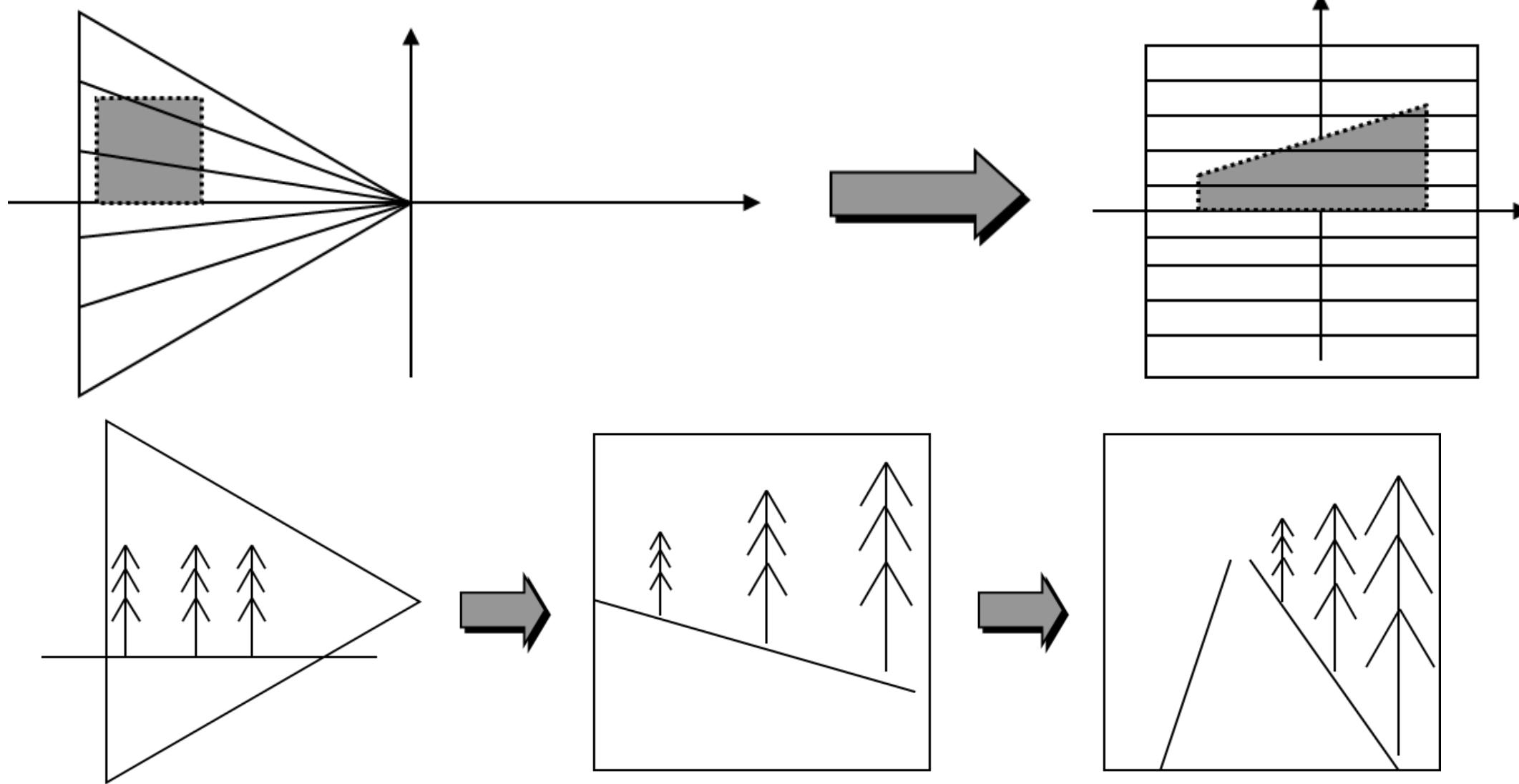


$$M_{\text{perspective}} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Projections

Normalizing transform

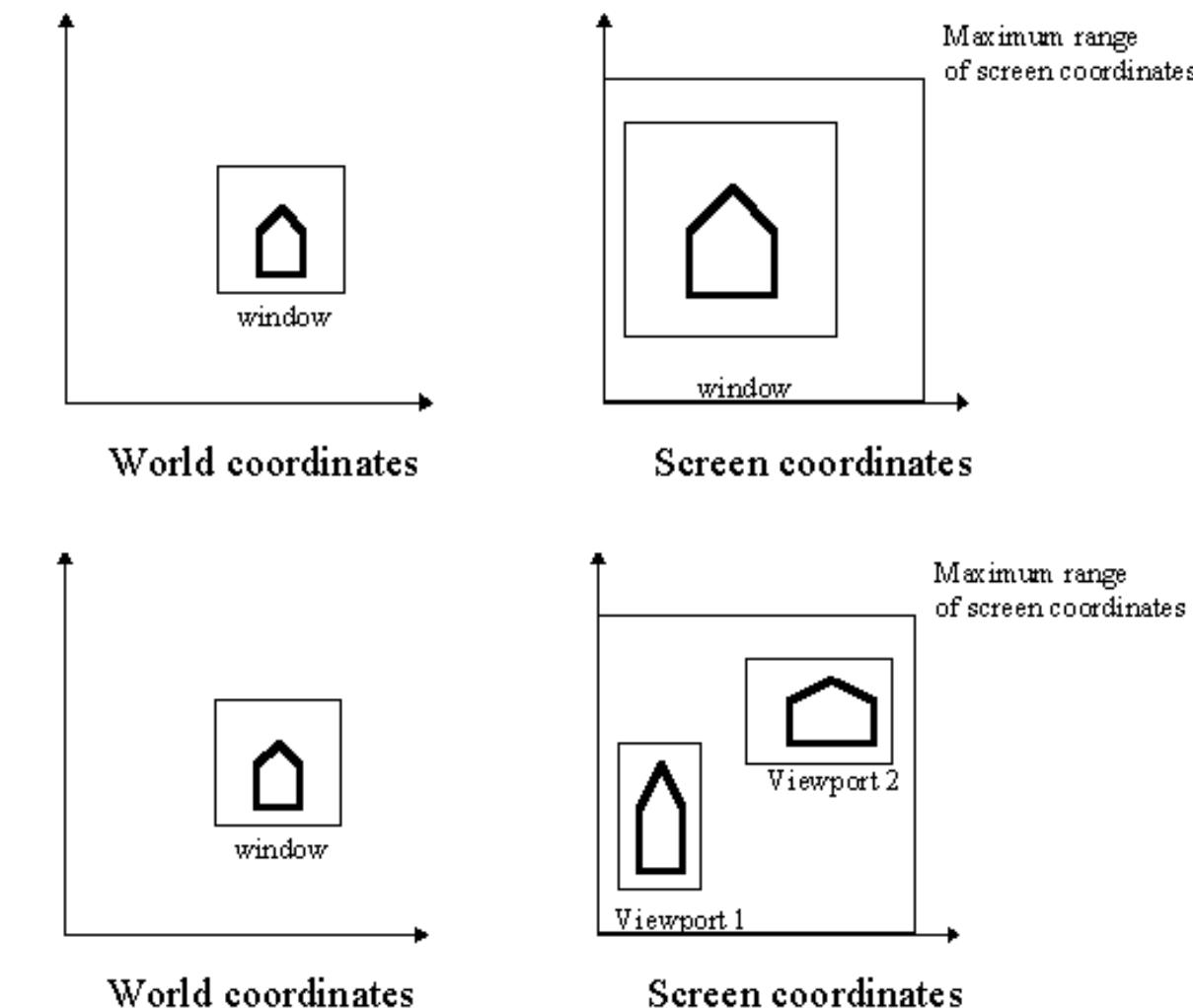
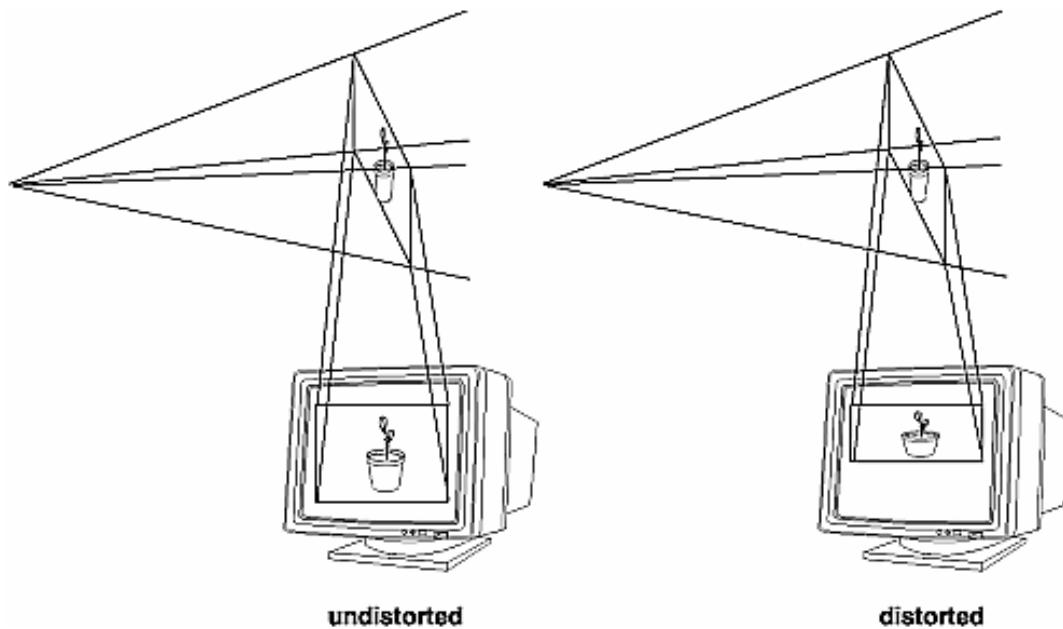
- Regions close to observer enlarged, distant regions shrink
- Perspective distortion



Projections

Viewport transformation

- Maps the projected 2D coordinates to a screen area in the window

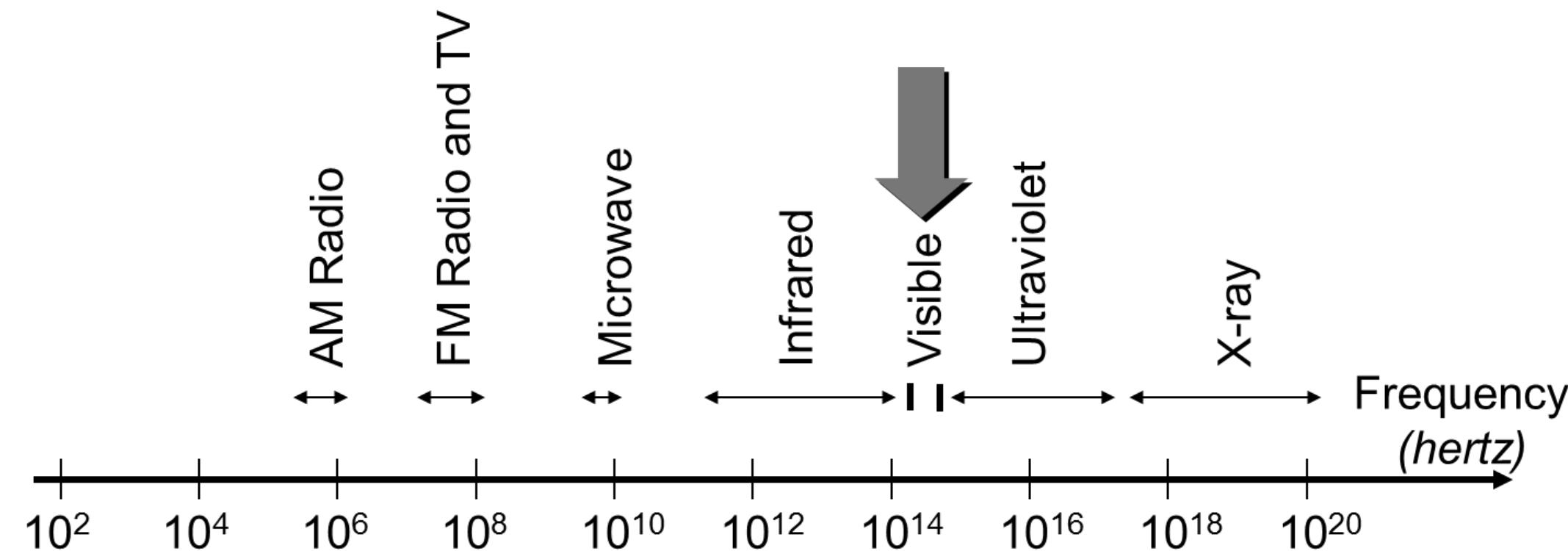


ohm

2.3 Color

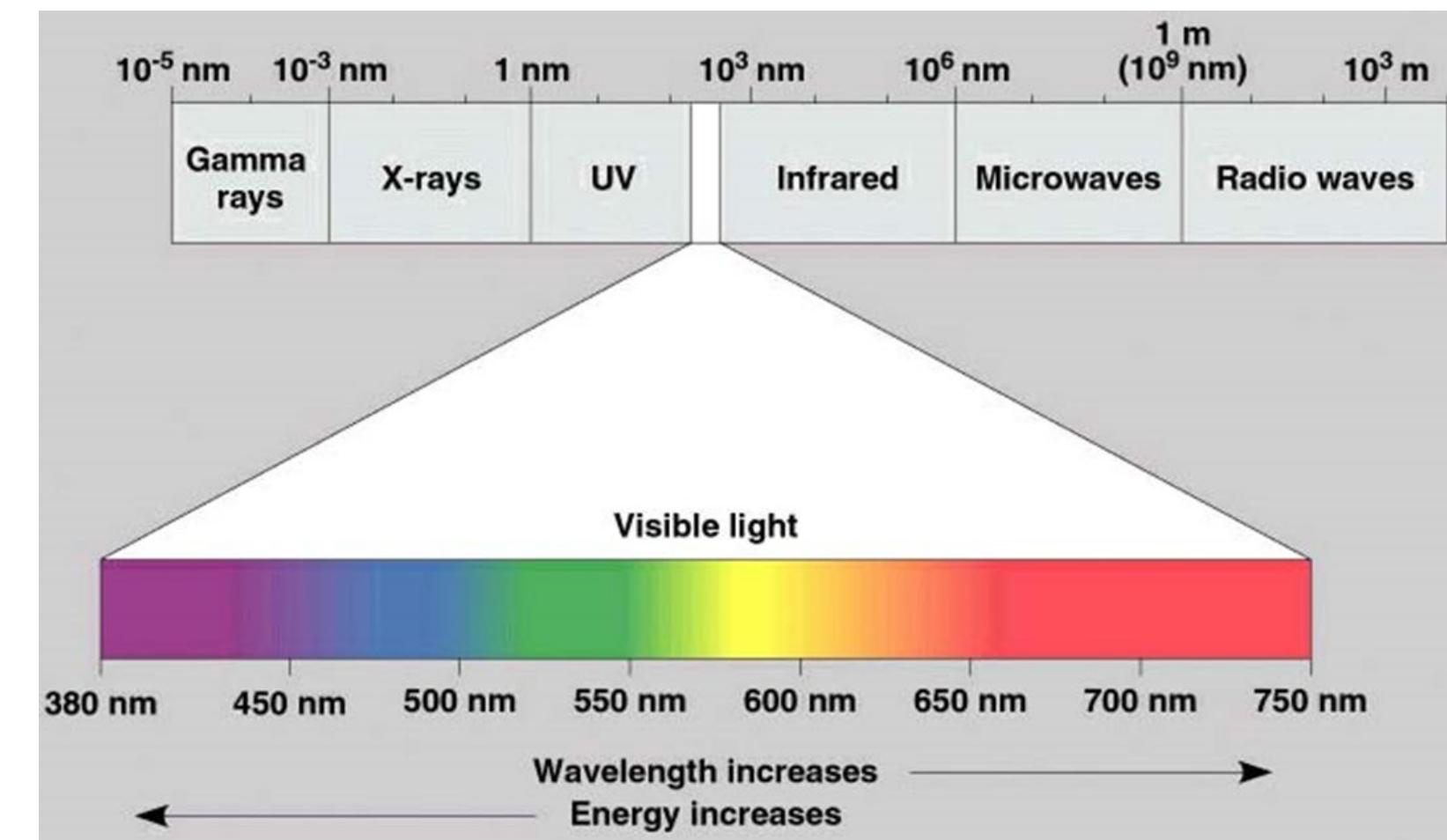
Color

- What is light?
 - Narrow frequency band of electromagnetic spectrum
 - Red color: $\sim 4.3 \times 10^{14}$ Hz
 - Violet color: $\sim 7.5 \times 10^{14}$ Hz



Color

- Light can originate from
 - Emission
 - Scattering
 - Absorption / reflection



Color

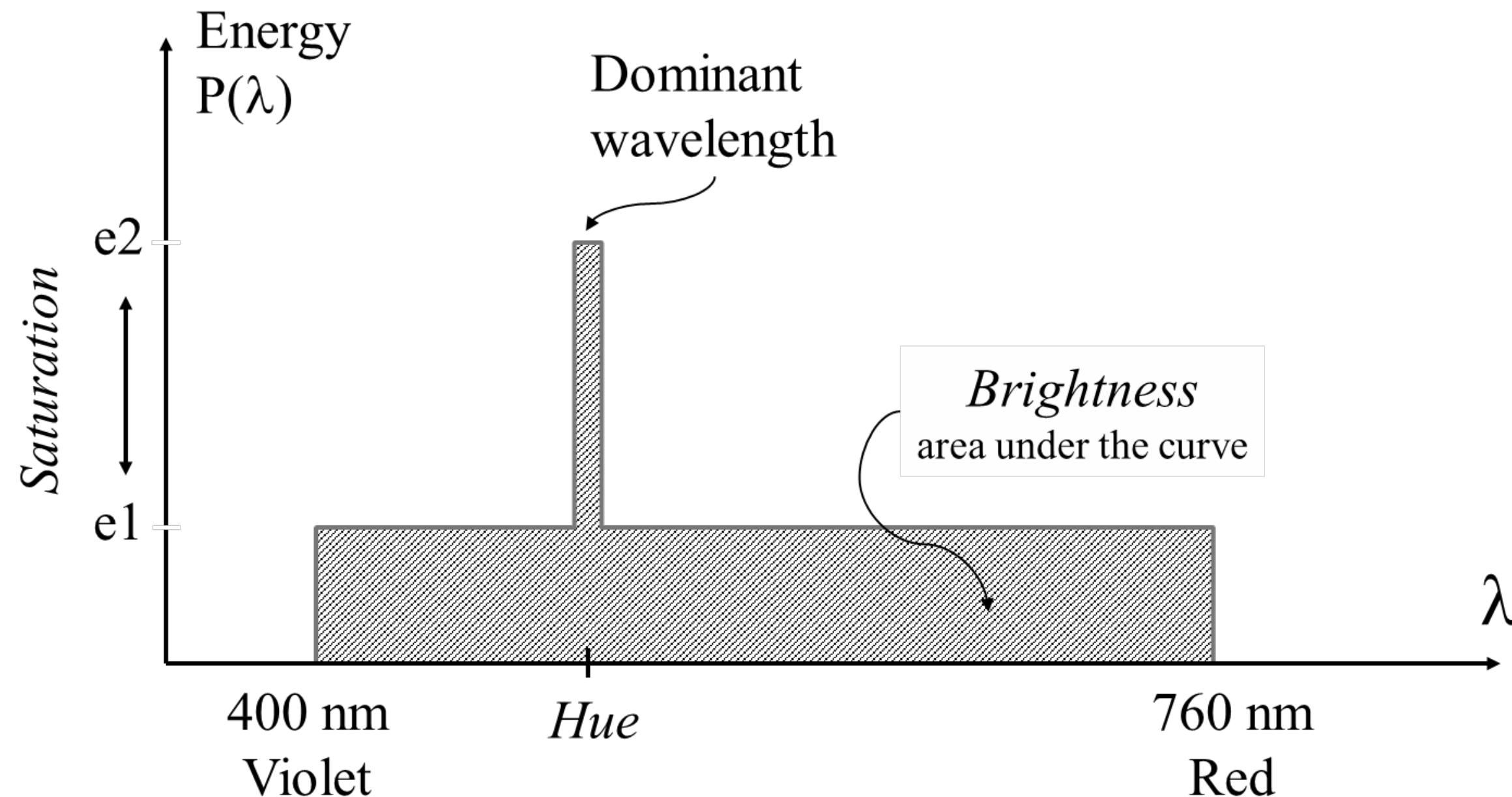
- What is color?
 - Physical
 - Spectra of wavelengths
 - Psychological
 - Stimulus sent from the optic system to the brain
 - Sensors on the retina of the eye: rods and cones
 - Computer graphics
 - Different sets of bases and coordinates
 - Depending on the type of display and application

Color

- Perceptual terms
 - Hue
 - The color seen (e.g. red, blue, ...) - dominant wavelength
 - Saturation Intensität der Farbe
 - How far is the color from a grey of equal intensity (how intense is the hue?)
 - Brightness
 - Total light energy - quantified as luminance
 - Perceived intensity of a self-luminous object - emitted light
 - Lightness: refers to intensity from a reflecting object

Color

- Hue, saturation and brightness



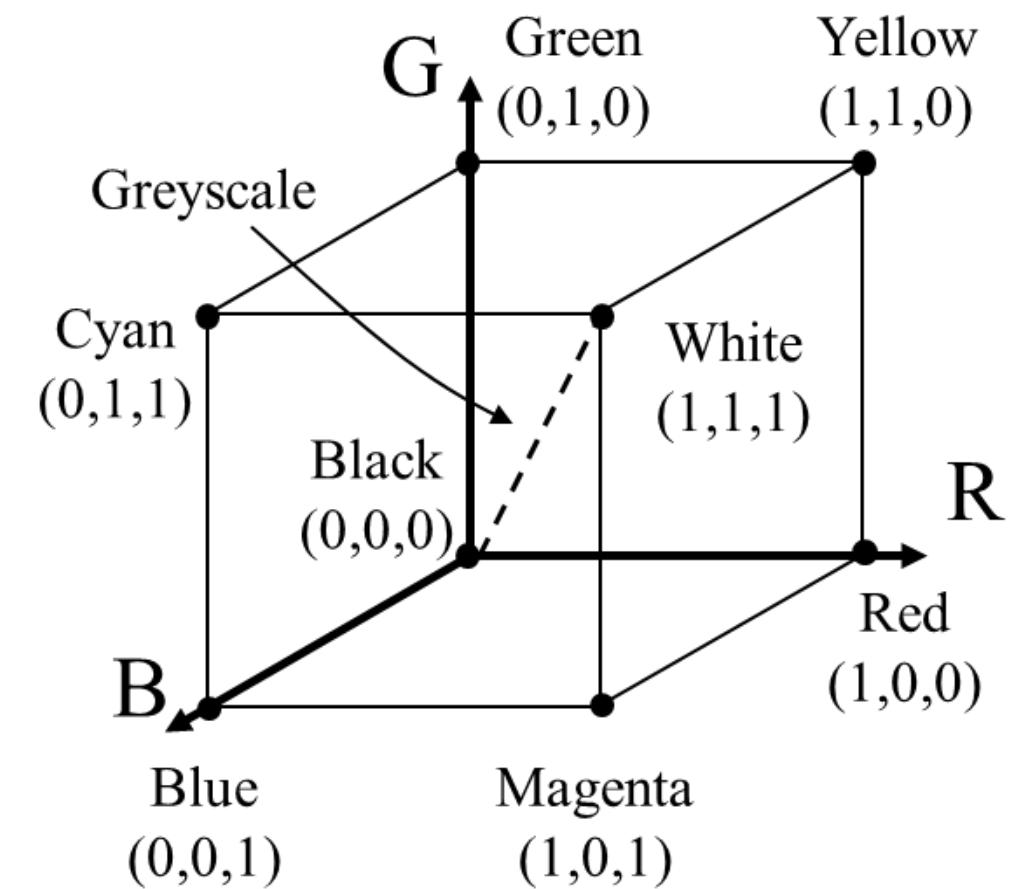
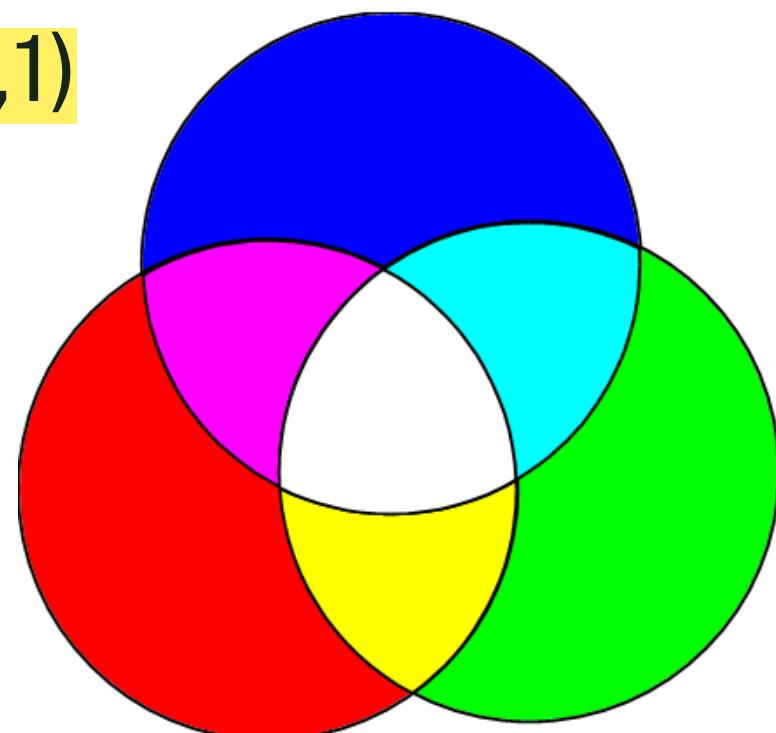
Color

- Complementary colors
 - Mixing produces white light
 - E.g.: red + cyan, green + magenta, blue + yellow
- Primary colors
 - Base colors of color model (three colors sufficient!)
 - Other colors mixed out of primary colors
 - No finite set can produce all possible visible colors!
- Color gamut
 - Set of all colors produced from primary colors

Color

RGB color model

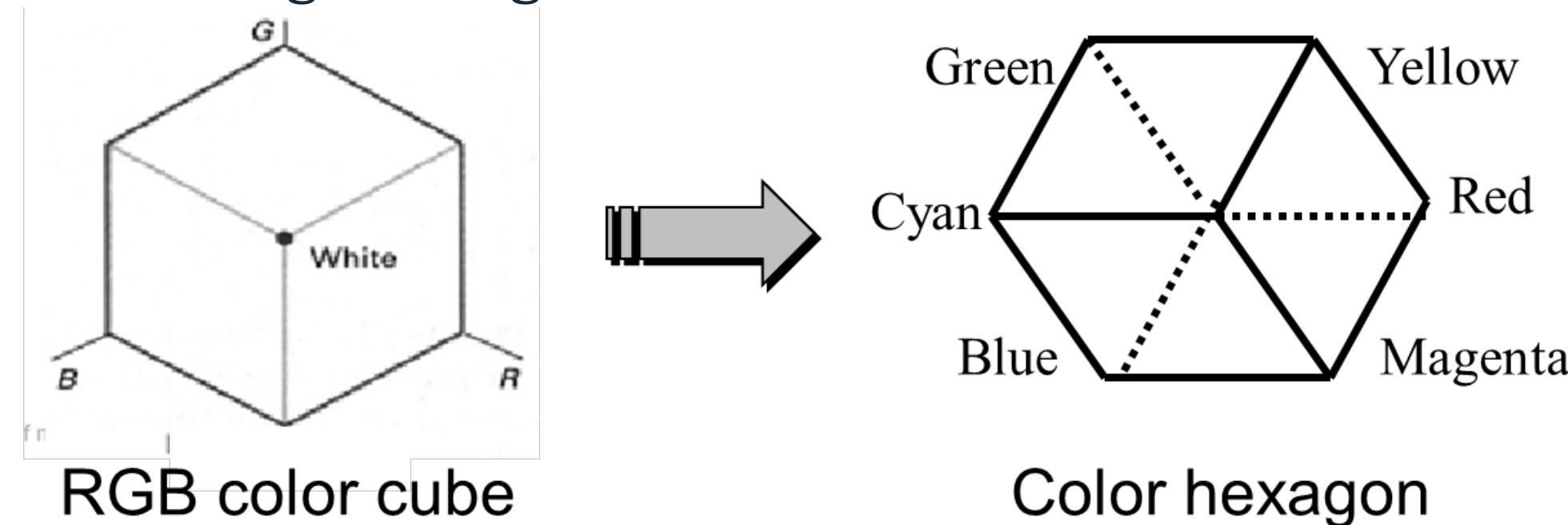
- Red, green and blue primaries
- Used (internally) in every monitor
- Additive (colors added to a black background)
- Black = $(0,0,0)$, White = $(1,1,1)$



Color

HSV color model

- More intuitive color specification (hue, saturation, value = brightness)
- Derived from RGB
 - Flatten RGB color cube along the diagonal from white to black



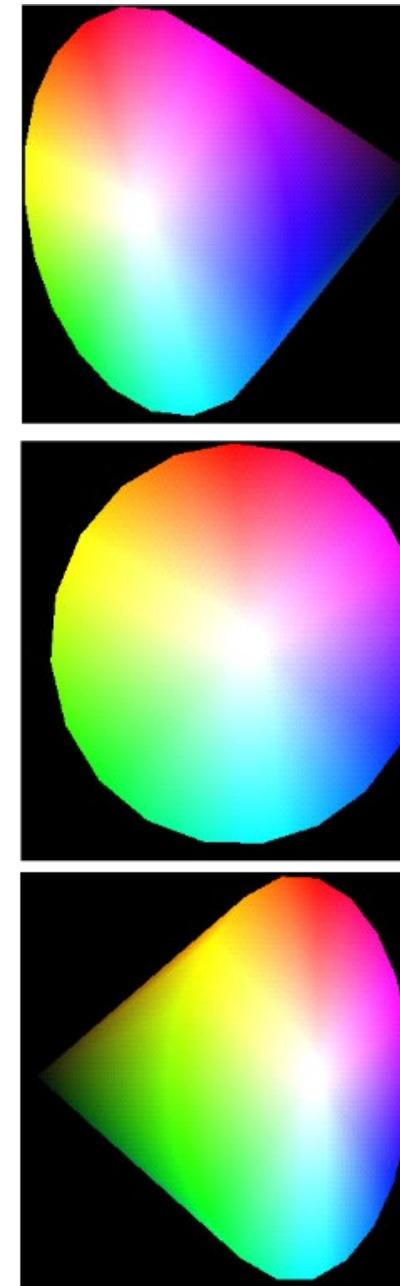
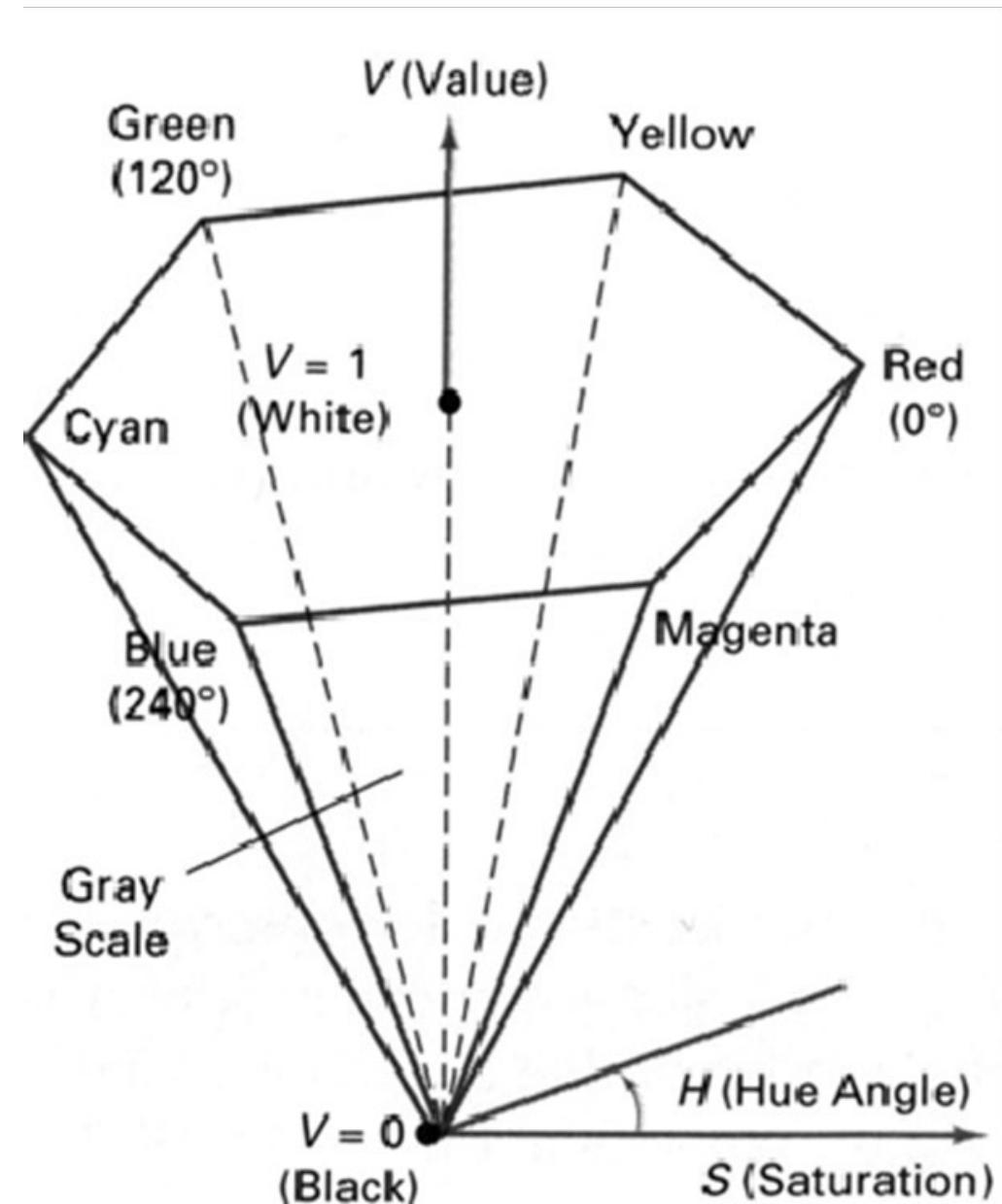
- Hue, saturation and value primaries

Color

HSV color model

- Components
 - Hue (H)
range $[0^\circ, 360^\circ]$
 - Saturation (S)
range $[0, 1]$
 - Value (V)
range $[0, 1]$

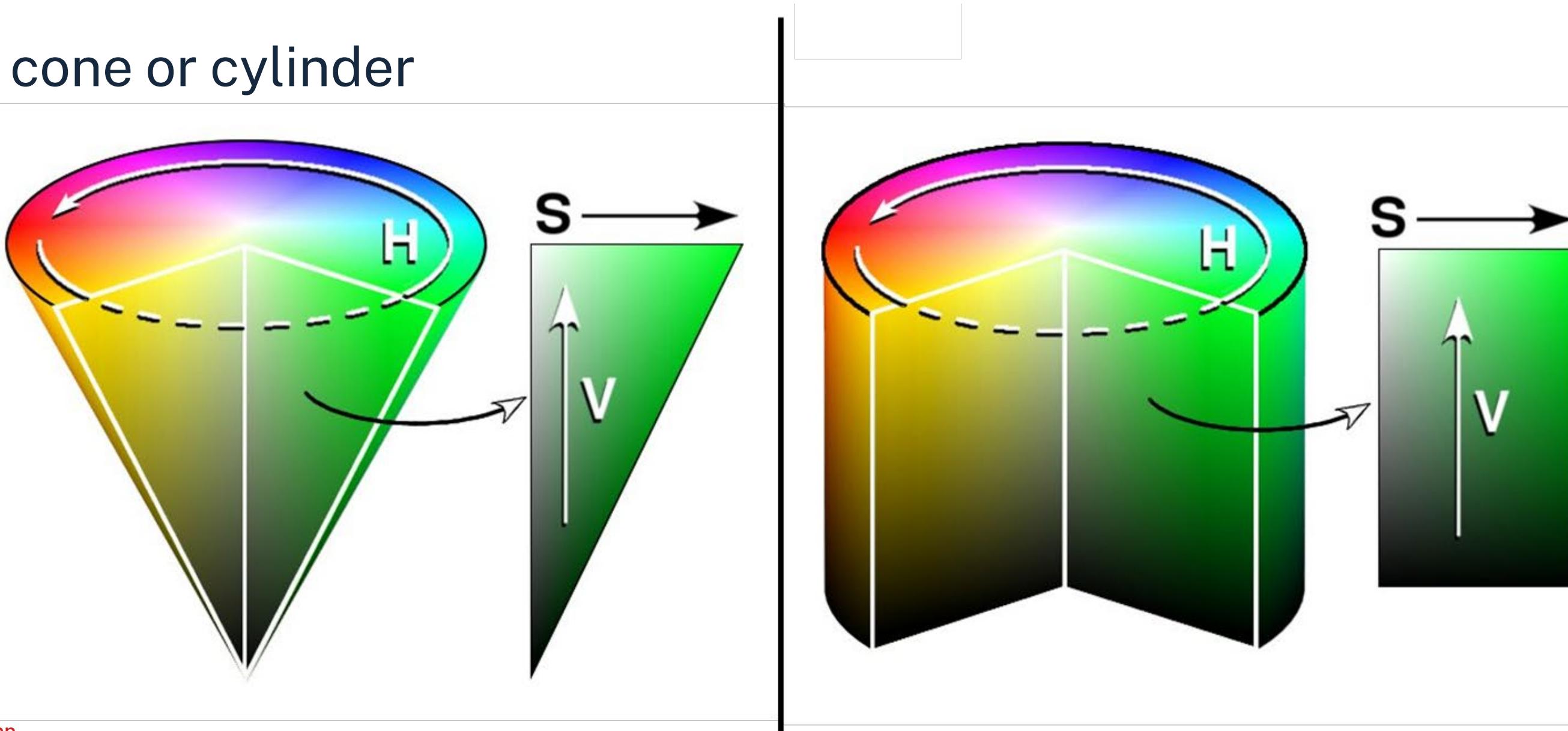
Hue ist Winkel
Value ist y-Achse
Saturation ist x-Achse



Color

Geometric model for HSV

- Circular cone or cylinder



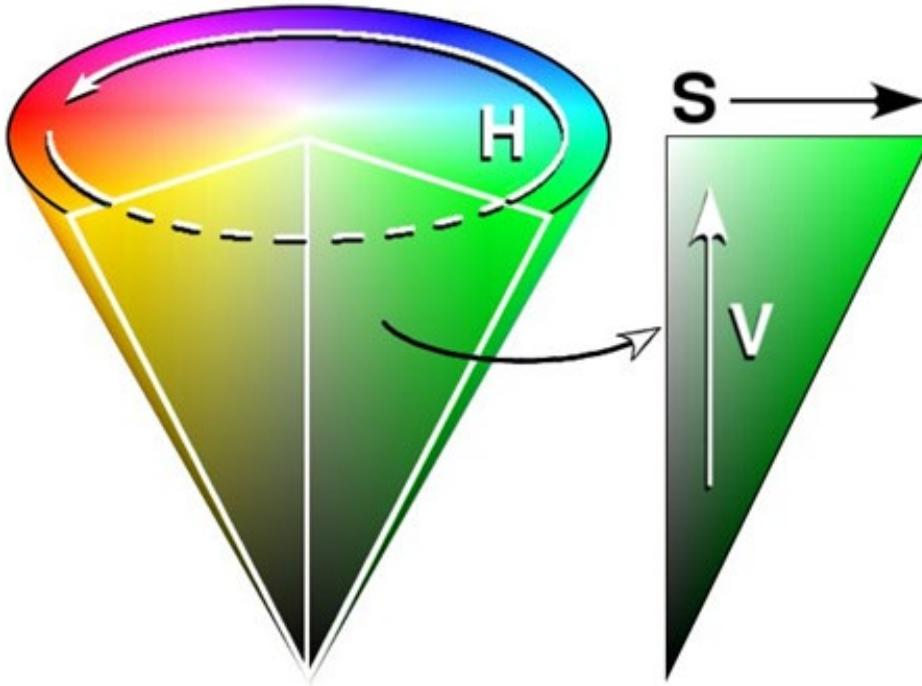
Color

- RGB to HSV (cone model)

$$V = \max(R, B, G) \quad (= \text{max}),$$

$$S = \frac{\max - \min}{\max},$$

$$H = \begin{cases} 60 \cdot \frac{G-B}{\max - \min} & \text{if } \max \{R, G, B\} = R \\ 60 \cdot \frac{B-R}{\max - \min} + 120 & \text{if } \max \{R, G, B\} = G \\ 60 \cdot \frac{R-G}{\max - \min} + 240 & \text{if } \max \{R, G, B\} = B \end{cases}$$



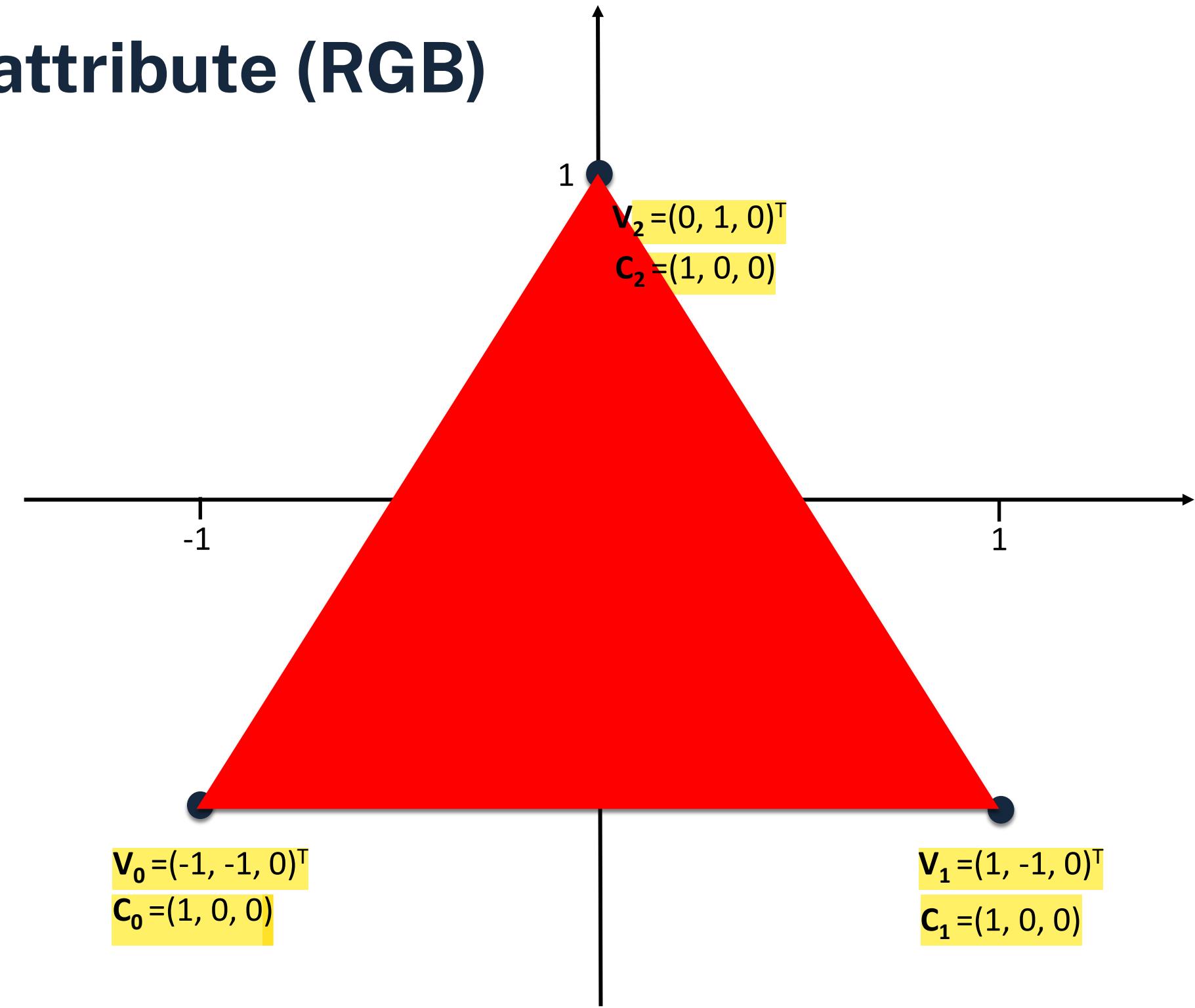
Color

Other color models

- CMY
 - Cyan, magenta and yellow primaries - used usually on hardcopy devices (printers)
 - Subtractive (white background!)
 - Complimentary to RGB ($C=1-R$, $M=1-G$, $Y=1-B$)
- CMYK
 - Addition of K-channel for black (saving ink)
 - Non-unique color presentation, e.g. $(1,1,1,0) == (0,0,0,1)$
- All of these are device dependent!
- Absolute model - device independent: CIE-XYZ
 - Used for calibration and data exchange
 - Based on human perception of colors

Color

Color as vertex attribute (RGB)



2.4 Lighting and Shading

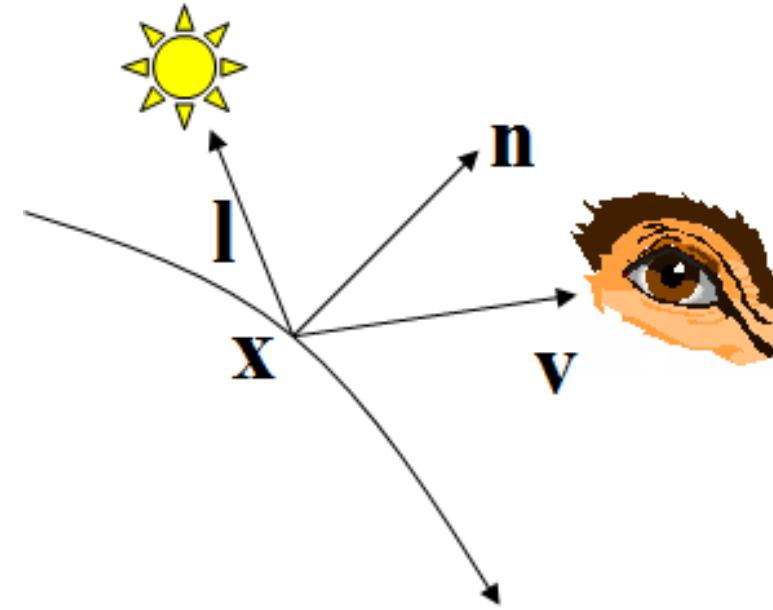
Lighting

- So far
 - Given a triangle and a viewpoint in 3D, the right pixels can be set
 - Given a constant solid color, the pixels can be colorized accordingly
- Question
 - Can we create better images?
- Attempt to create a realistic image
 - Simulate lighting of the surfaces in the scene
 - Fundamentally: simulation of physics and optics
- Approach in CG
 - Use approximations
 - Physically correct calculations are too expensive for real-time graphics!

Lighting

Local illumination model

- Ambient light
 - Indirect light from surrounding
 - Modeled by constant color
- Diffuse light
 - Reflection from rough surfaces
 - Uniform in all viewing directions
- Specular light
 - Reflection from "glossy" but not perfectly mirroring surfaces
 - somewhere in "between" diffuse and mirroring



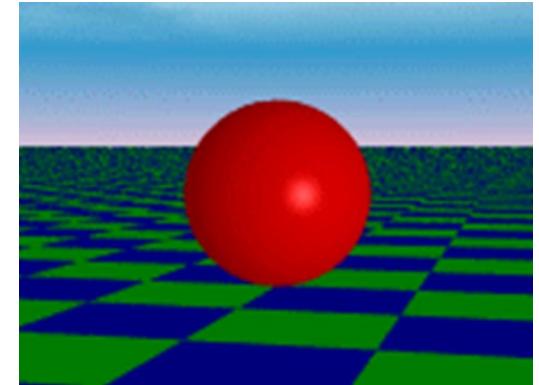
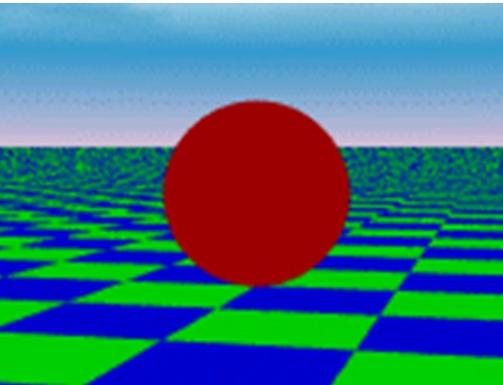
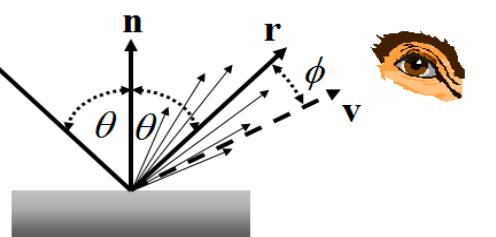
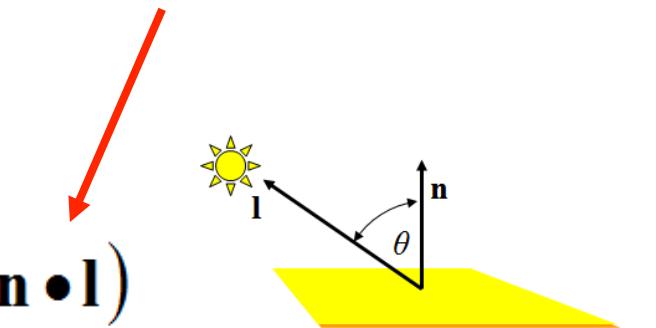
$$L_{total} = L_{ambient} + L_{diffuse} + L_{specular}$$

Lighting

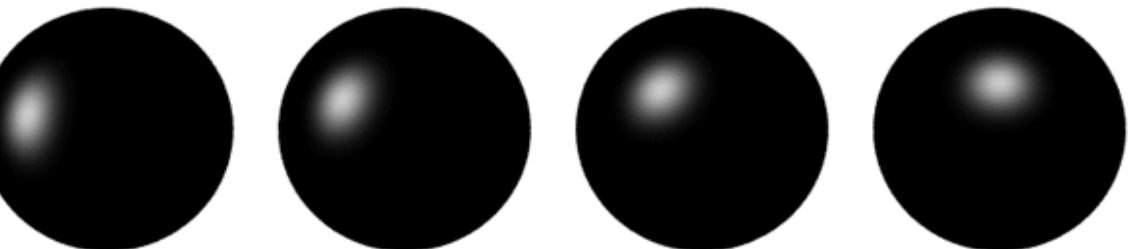
Local illumination model

- Ambient light
 - $L_{amb} = k_{amb}I_{amb}$
- Diffuse light
 - $L_{diff} = k_{diff}I_{in} \cos \theta$ or $L_{diff} = k_{diff}I_{in}(\mathbf{n} \bullet \mathbf{l})$
- Specular light (Phong Lighting)
 - $L_{spec} = k_{spec}I_{in}(\cos \phi)^{n_{shiny}}$
 - $L_{spec} = k_{spec}I_{in}(\mathbf{v} \bullet \mathbf{r})^{n_{shiny}}$
 - with $\mathbf{r} = (2(\mathbf{n} \bullet \mathbf{l}))\mathbf{n} - \mathbf{l}$

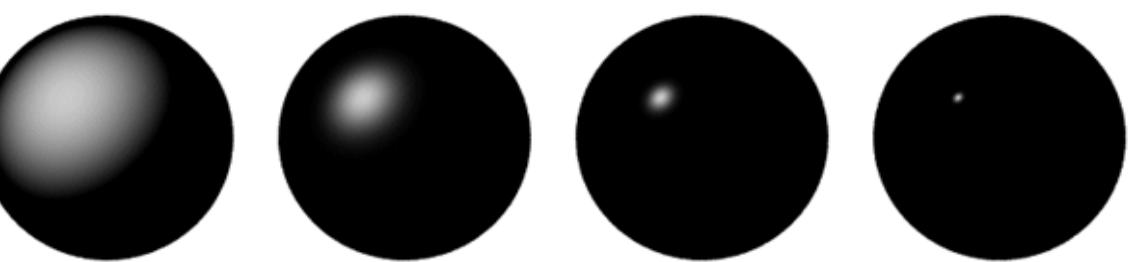
For unit-length
normal vectors



Varying \vec{l}



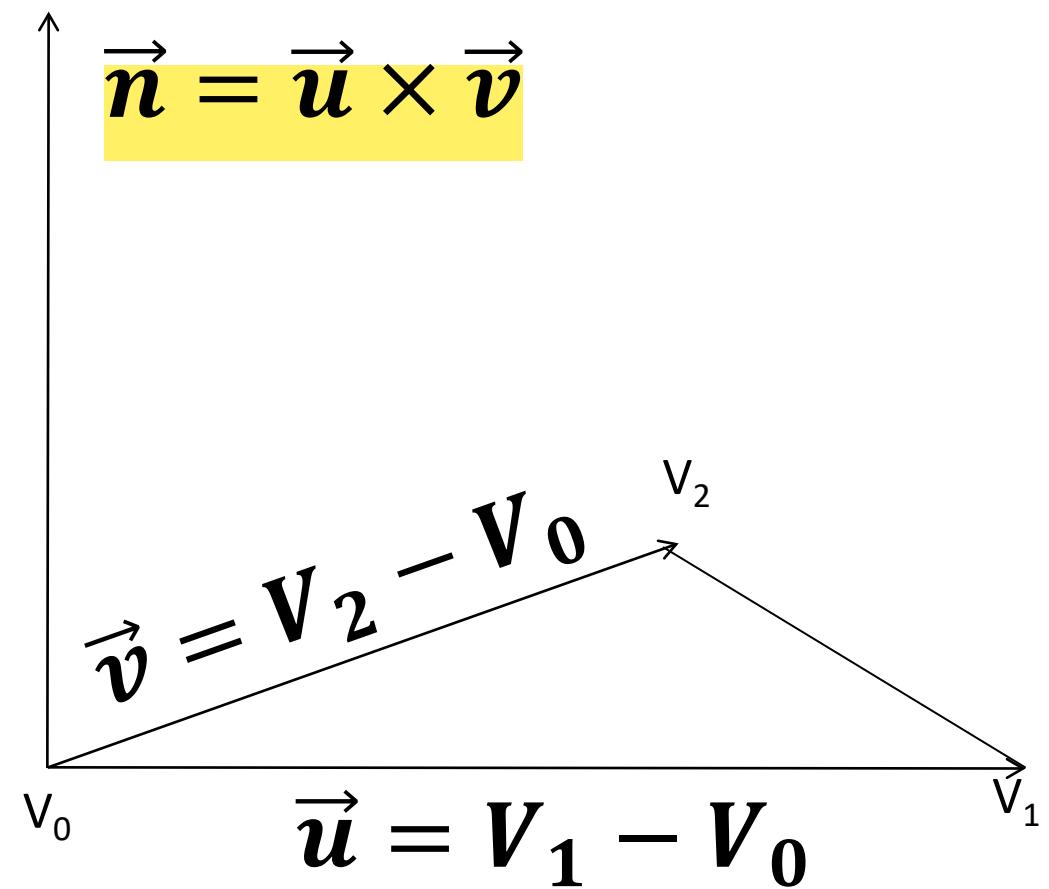
Varying n_{shi}



Lighting

Calculation of normal vectors

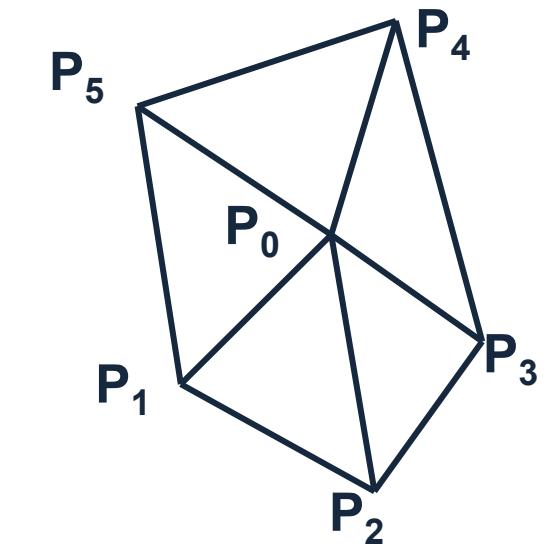
- Note: direction of normal depends on vertex orientation



Lighting

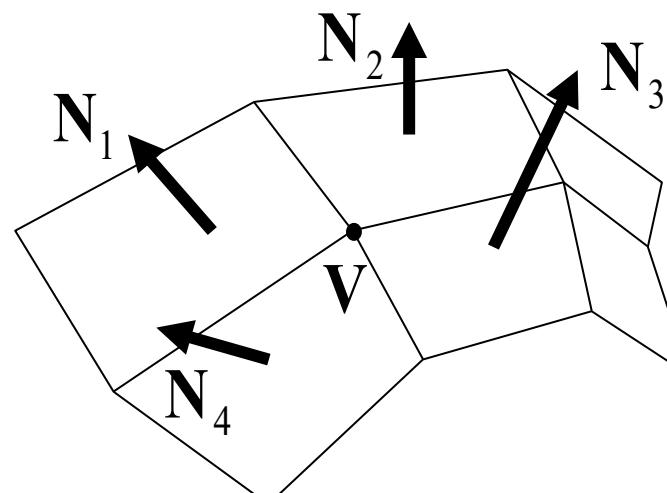
Surface / mesh normals

- Newell's method for polygonal meshes
 - More robust than simple cross product
 - Works also for general polygons
- Average normals of faces to get surface point normal



$$\mathbf{N}_0 = \sum_{i=1}^n \mathbf{P}_i \times \mathbf{P}_{i+1}, \quad \mathbf{P}_{n+1} = \mathbf{P}_1$$

Mitteln der Oberflächen Normalen, um Kanten runder ausschauen zu lassen

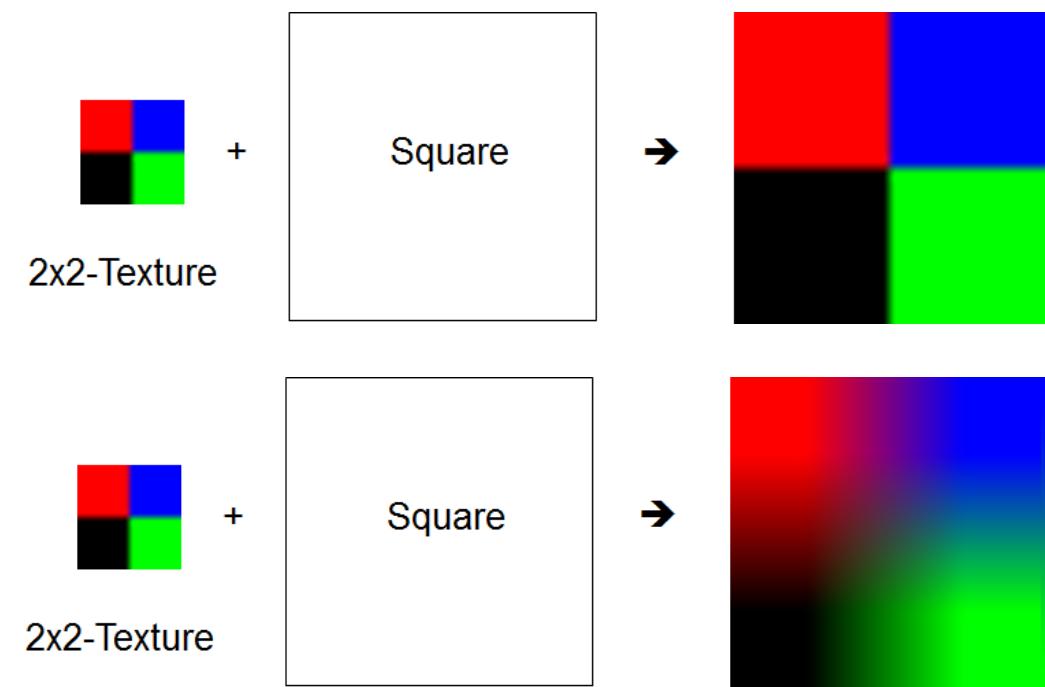
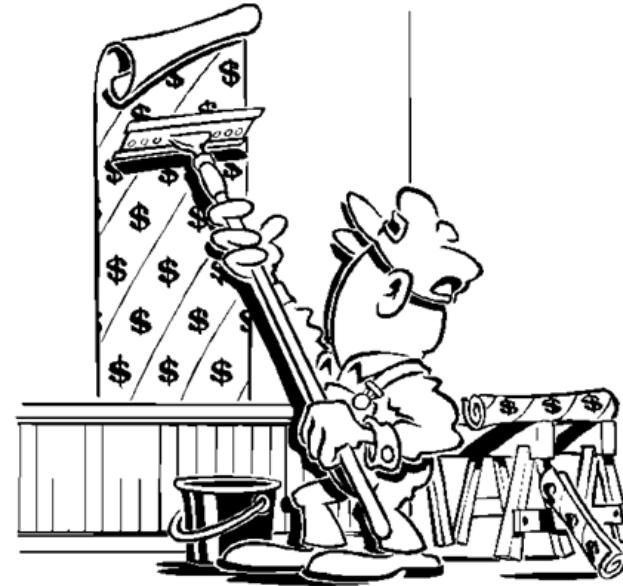


$$\mathbf{N}_V = \frac{\sum_{k=1}^n \mathbf{N}_k}{\left| \sum_{k=1}^n \mathbf{N}_k \right|}$$

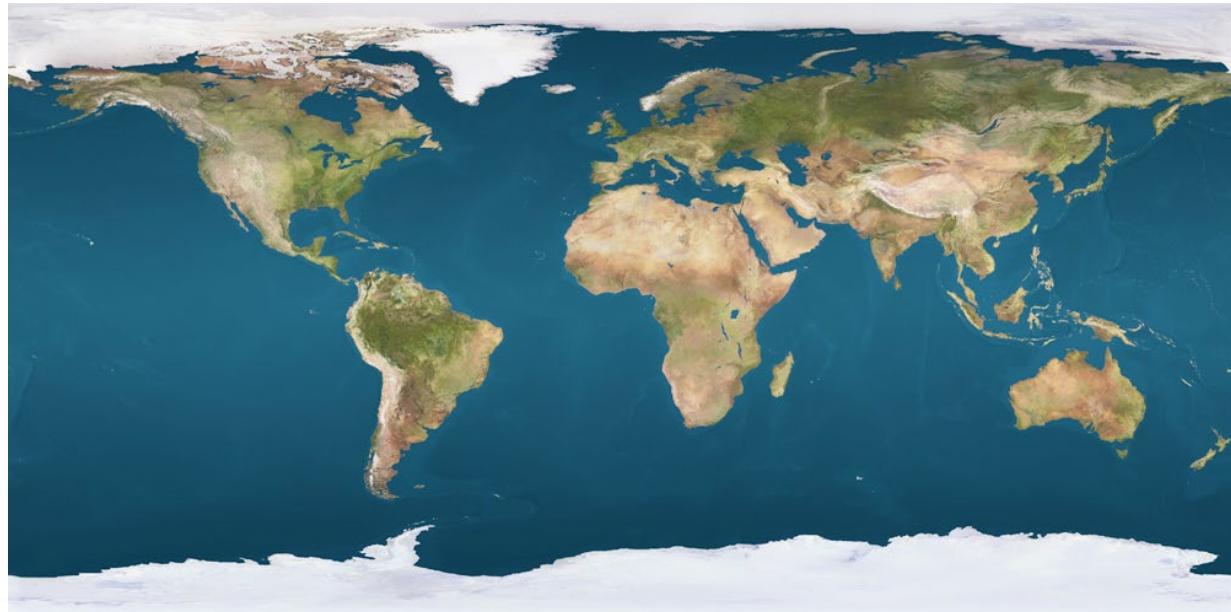
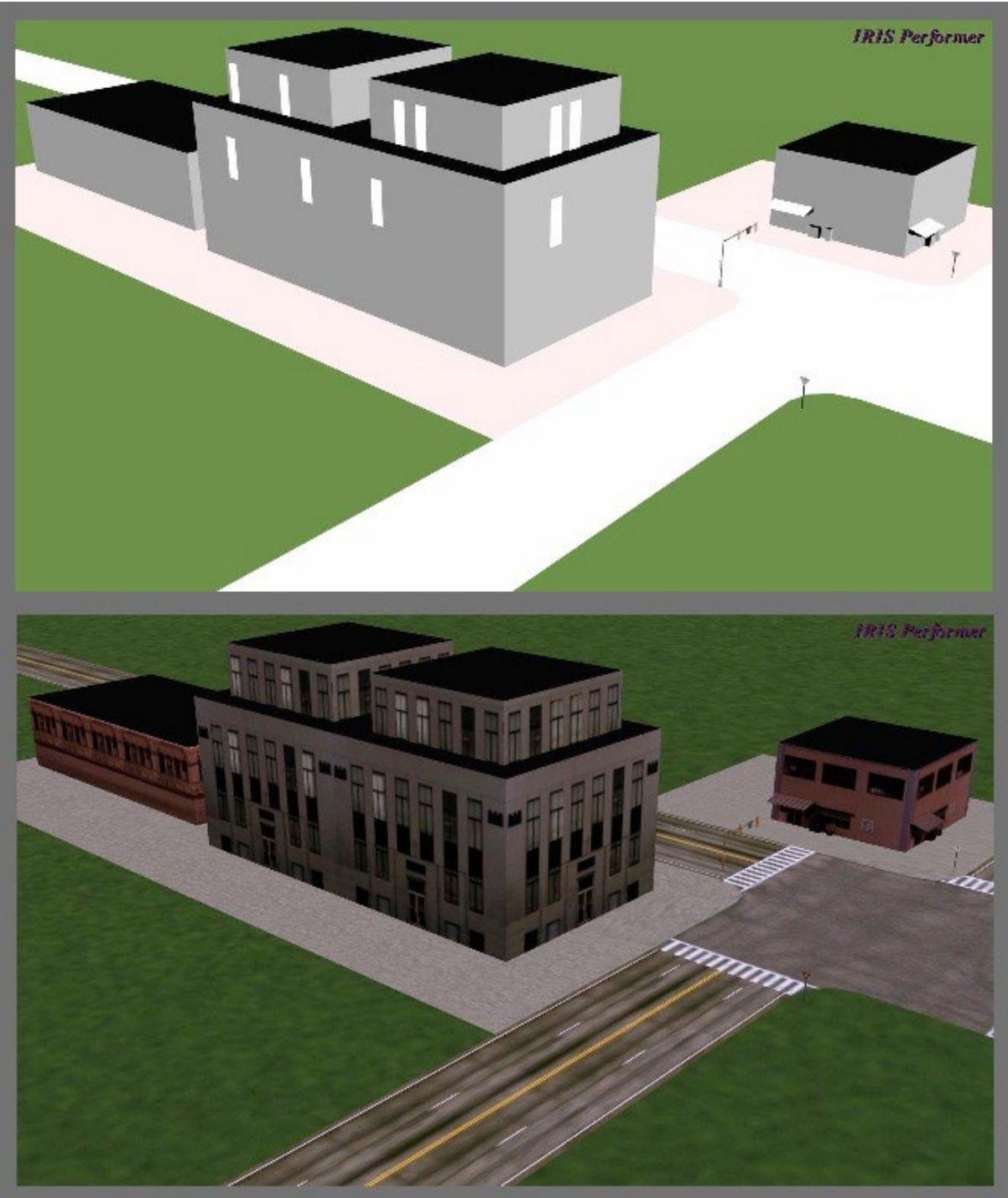
2.5 Texture Mapping

Texture Mapping

- Textures
 - Functions or pixel-based images containing reflectance information
 - Can be 1D, 2D or 3D (esp. in Vis)
 - Attach "images" to polygons to achieve highly detailed surface at low cost
 - Can be used together with lighting



Texture Mapping



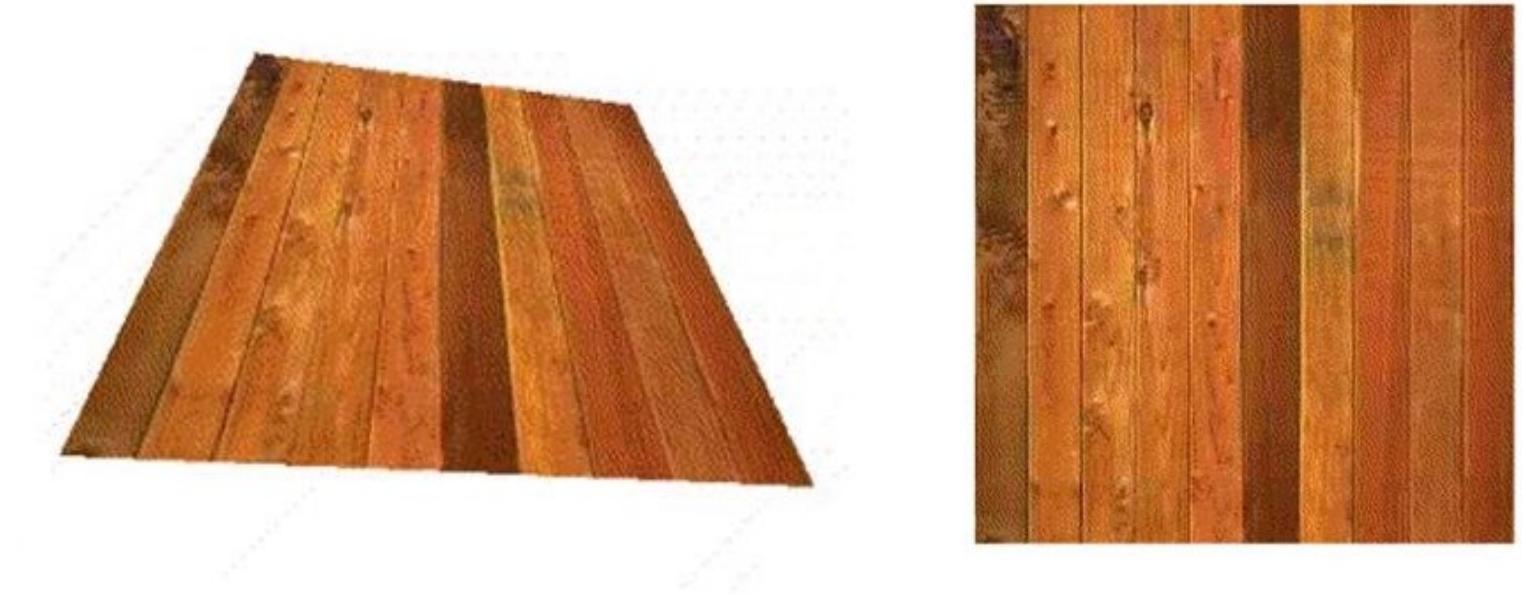
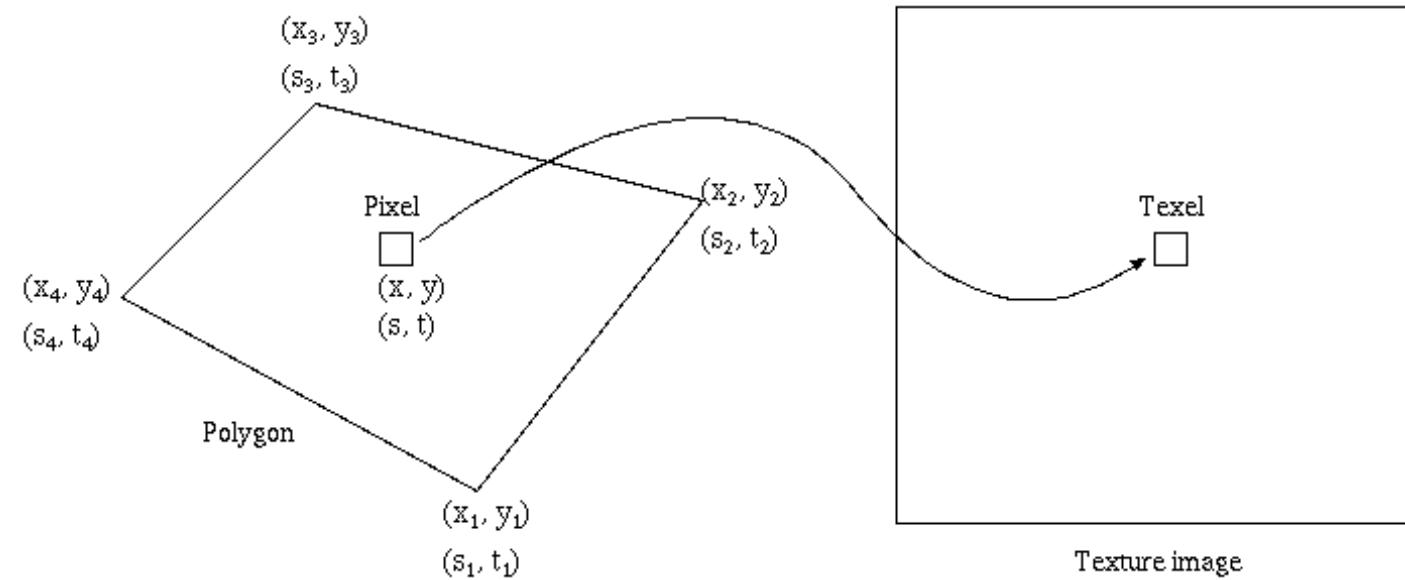
Texture Mapping

- Assign reflectance information to pixels by lookup
 - Texture: typically 2D pixel image (may be 1D or 3D)
 - Determines appearance of a surface
 - Texel
 - Pixel in a texture
 - Requires procedure to map the texture onto the surface (mapping)
 - Easy for single triangle
 - Complex for arbitrary surface in 3D

Texture Mapping

- Mapping in 2D
 - Domain of texture = $[0,1]^2$
 - Image of size $n_x \times n_y$ mapped to $[0,1]^2$
 - Usually (but not necessarily) constrained to powers of two
 - Texture coordinates u and v as vertex attributes
 - Assign to each (x,y,z) one (u,v)
 - Usually a two-step process
 - Assignment for every vertex of a mesh
 - Interpolation (later) of interior points
 - Different possibilities for mapping to $[0,1]^2$ space
 - Clamping, wrapping, periodic extension, ...
 - Texture information is read by some *interpolation* function

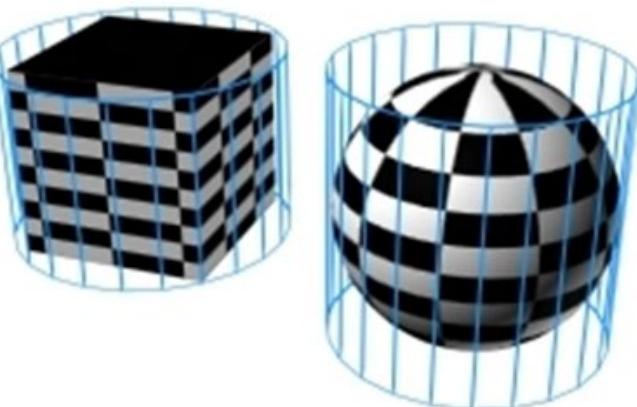
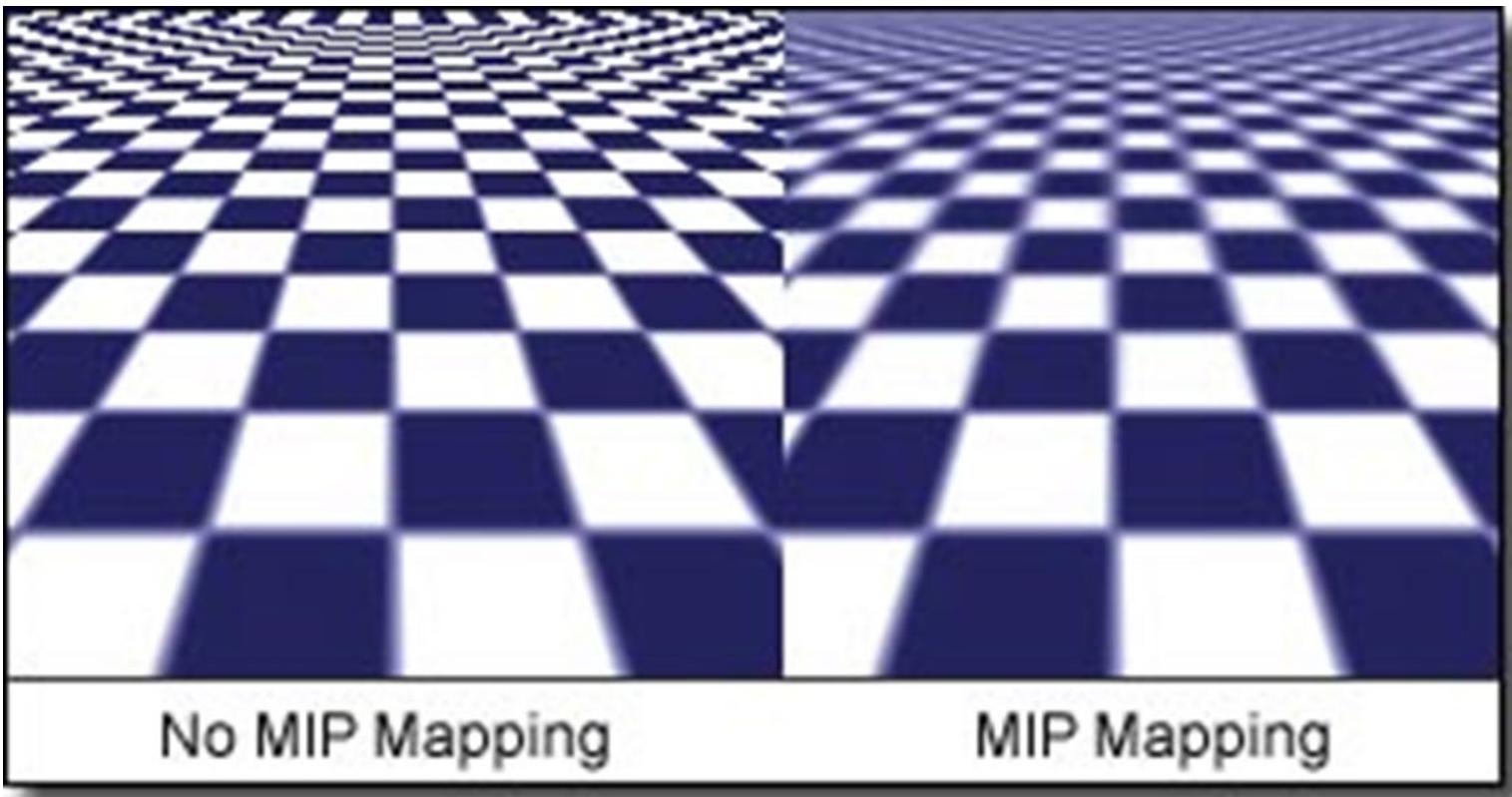
Texture Mapping



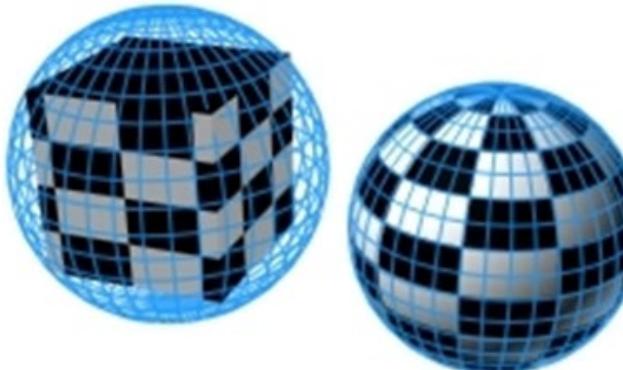
Source: <http://idav.ucdavis.edu/~okreylos/PhDStudies/Winter2000/TextureMapping.html>,
http://www.flipcode.com/archives/Light_Mapping_Theory_and_Implementation.shtml

Texture Mapping

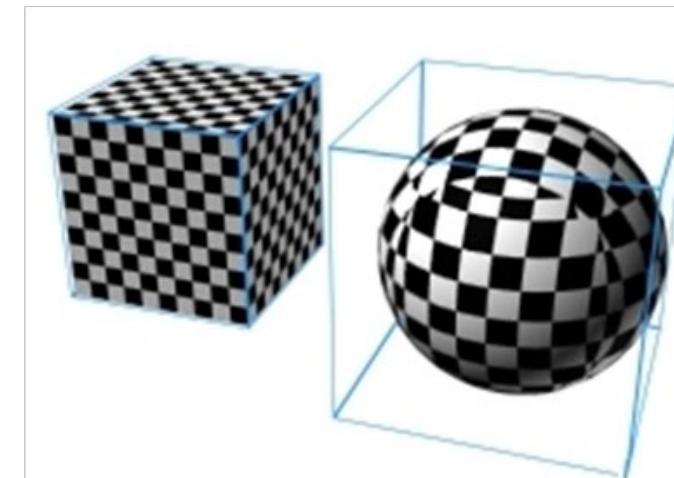
Interpolation and projection



Cylindrical Projection



Spherical Projection

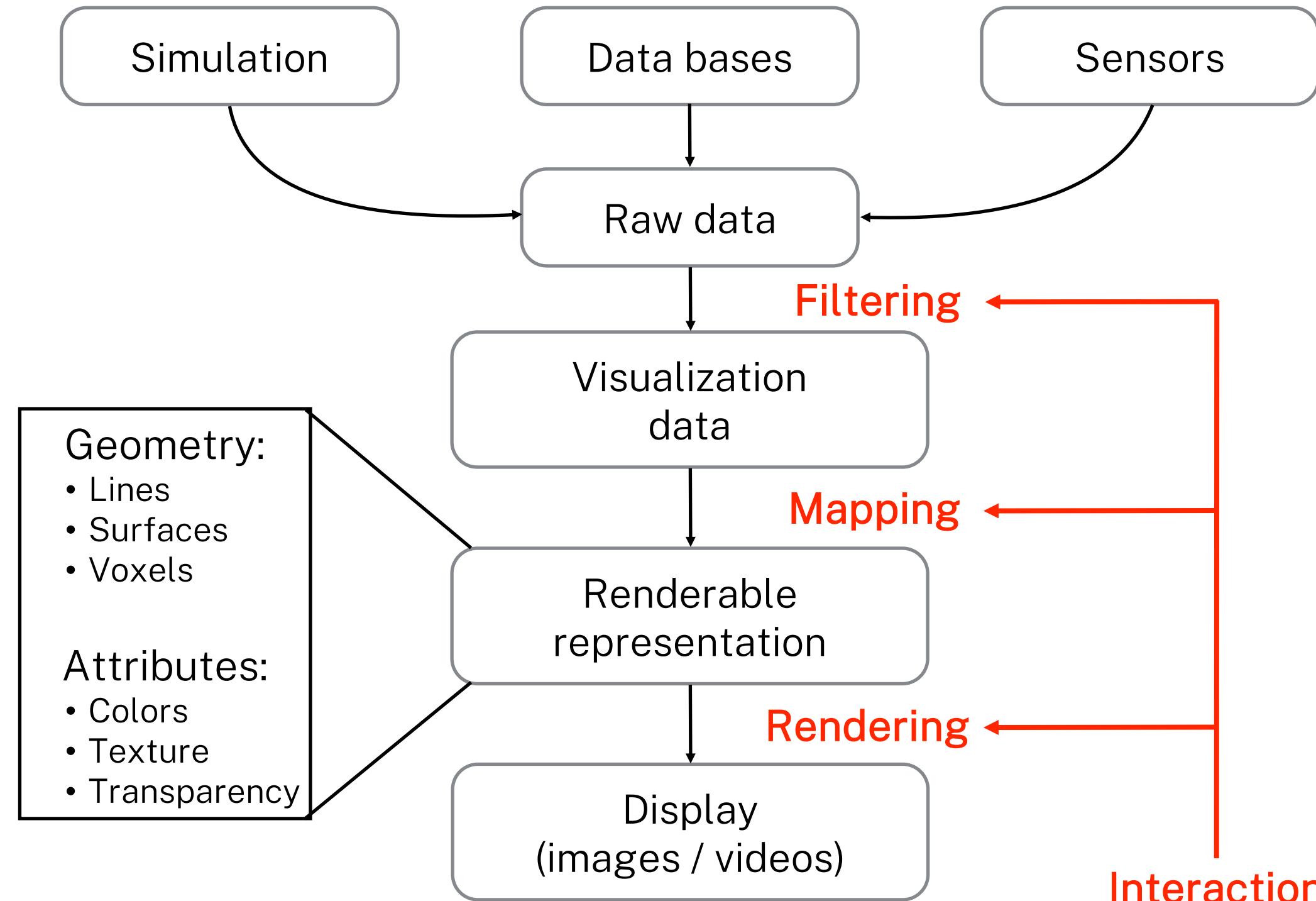


Box Projection

3. Fundamental Concepts

3.1 Visualization Pipeline

Visualization Pipeline

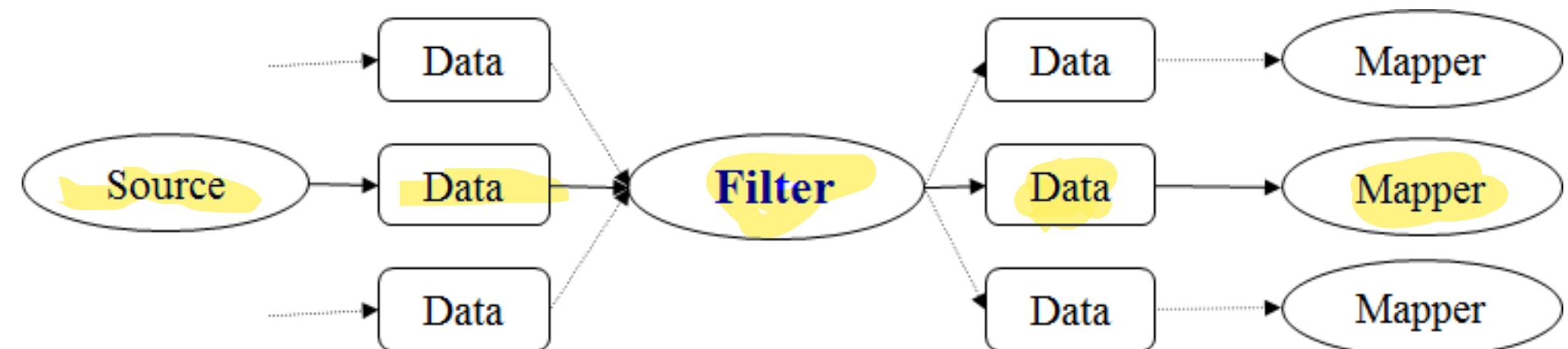


Data sources

- **Real-world**
 - Measurements and observation
- **Theoretical world**
 - Mathematical and technical models
- **Artificial world**
 - Designed data
- Traditional presentation techniques
 - Insufficient for increasing amount of data
 - Data from any source with almost arbitrary size
 - New developments required for efficient visualization of large-scale data sets and new data types

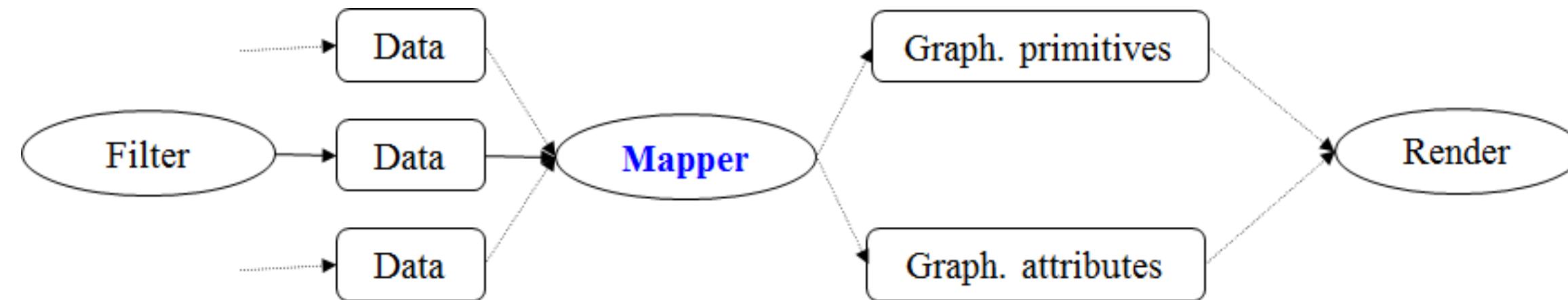
Visualization Pipeline

- **Filter:** transform data into data
 - Data format conversion
 - Clipping, cropping, de-noising
 - Slicing, resampling
 - Interpolation, approximation
 - Classification, segmentation



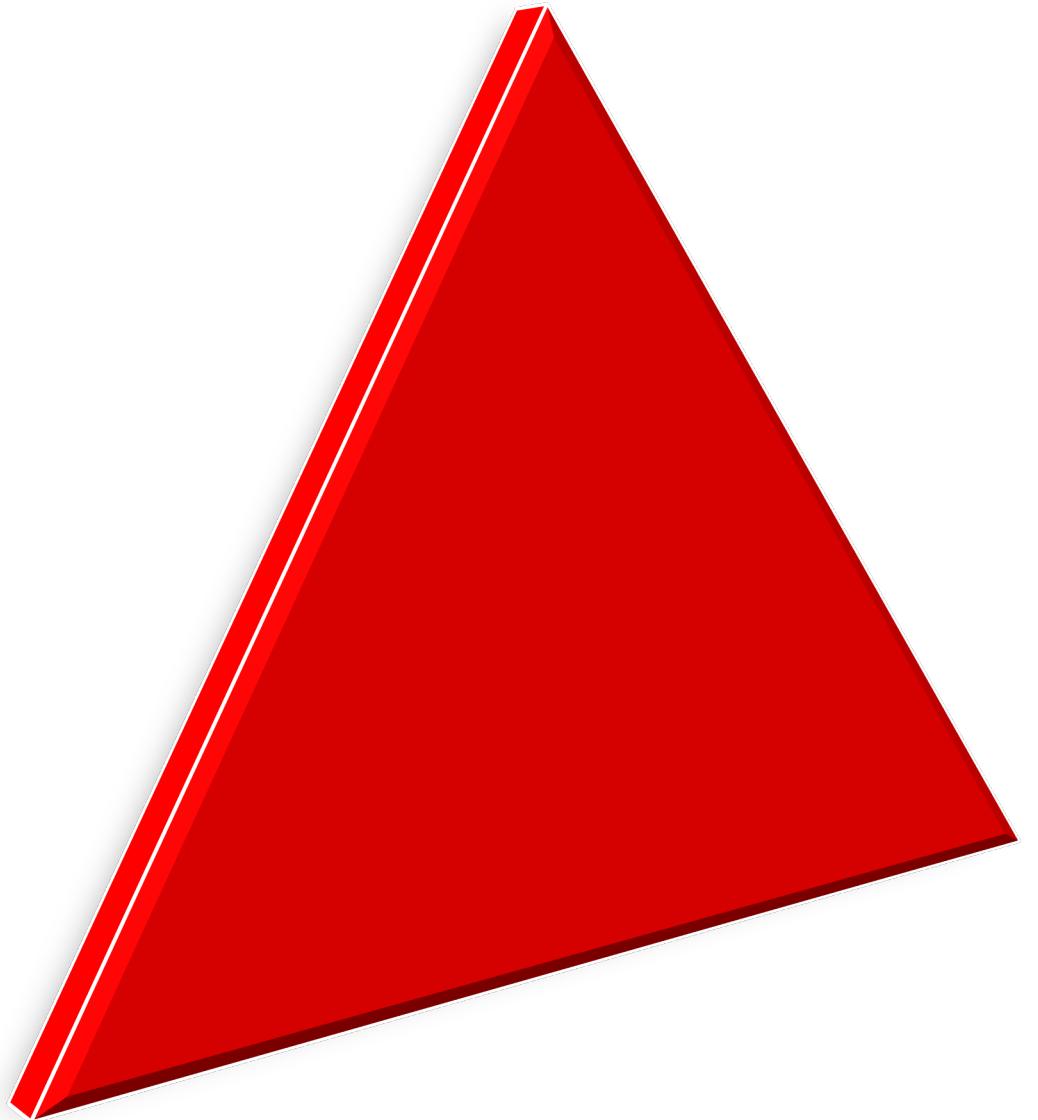
Visualization Pipeline

- **Mapper:** transform data into graphical primitives
 - Scalar field to iso-surface
 - Vector field to glyphs
 - ...

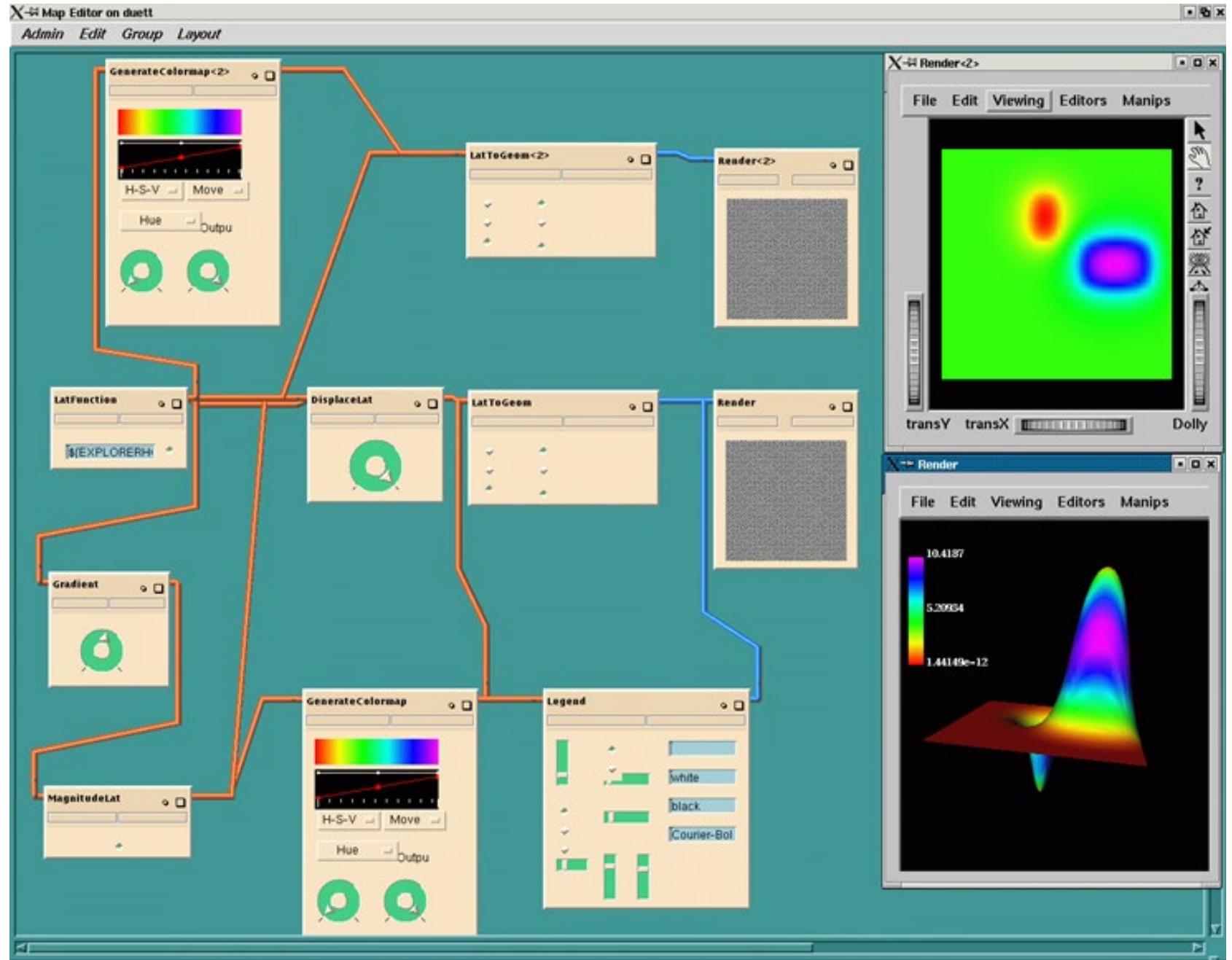


Visualization Pipeline

- **Rendering**
 - Geometry / images / volumes
 - Images / Video
 - “Realism” (e.g. shadows, ...)
 - Lighting, shading
 - Texturing

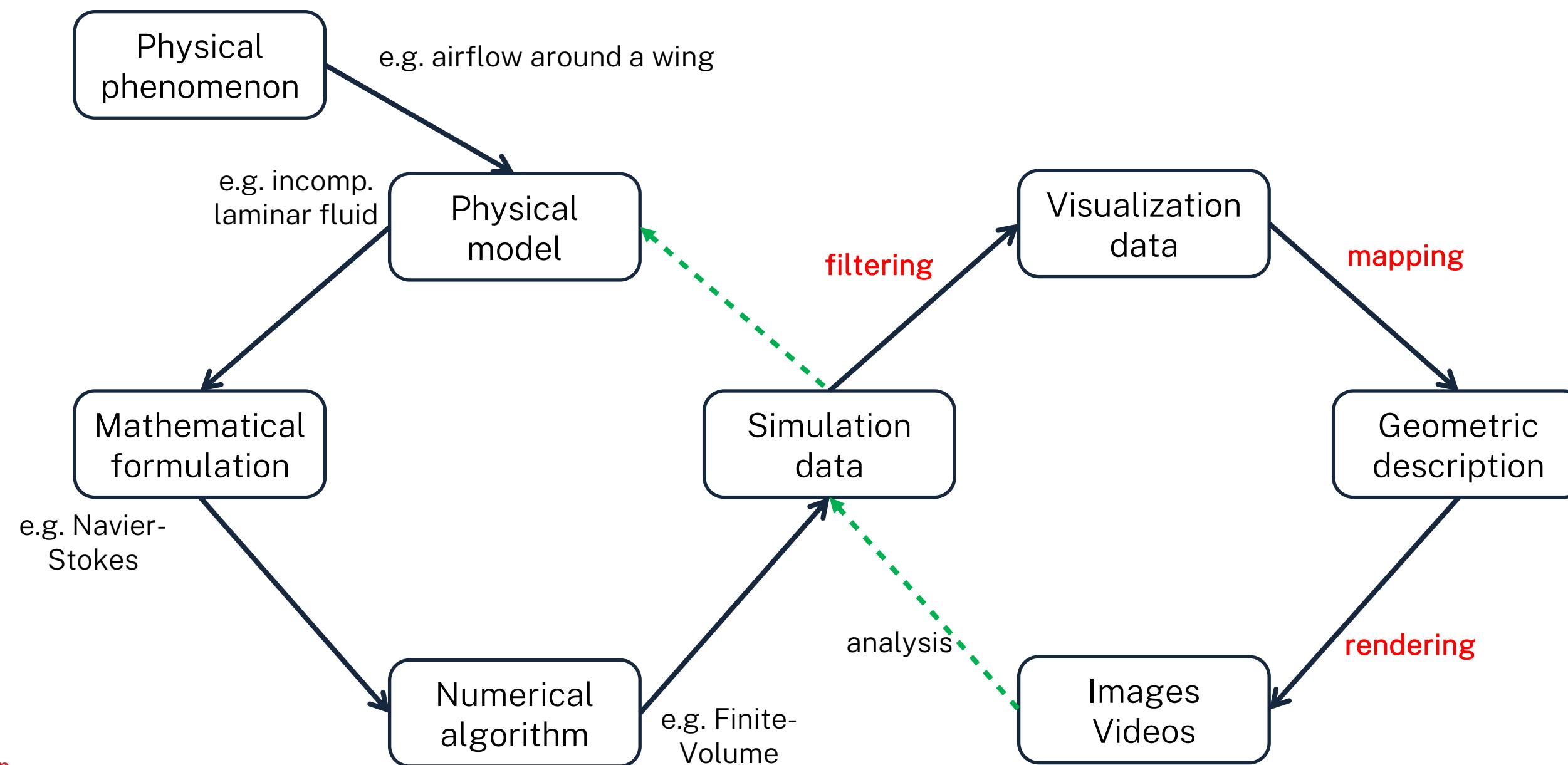


Visualization Pipeline



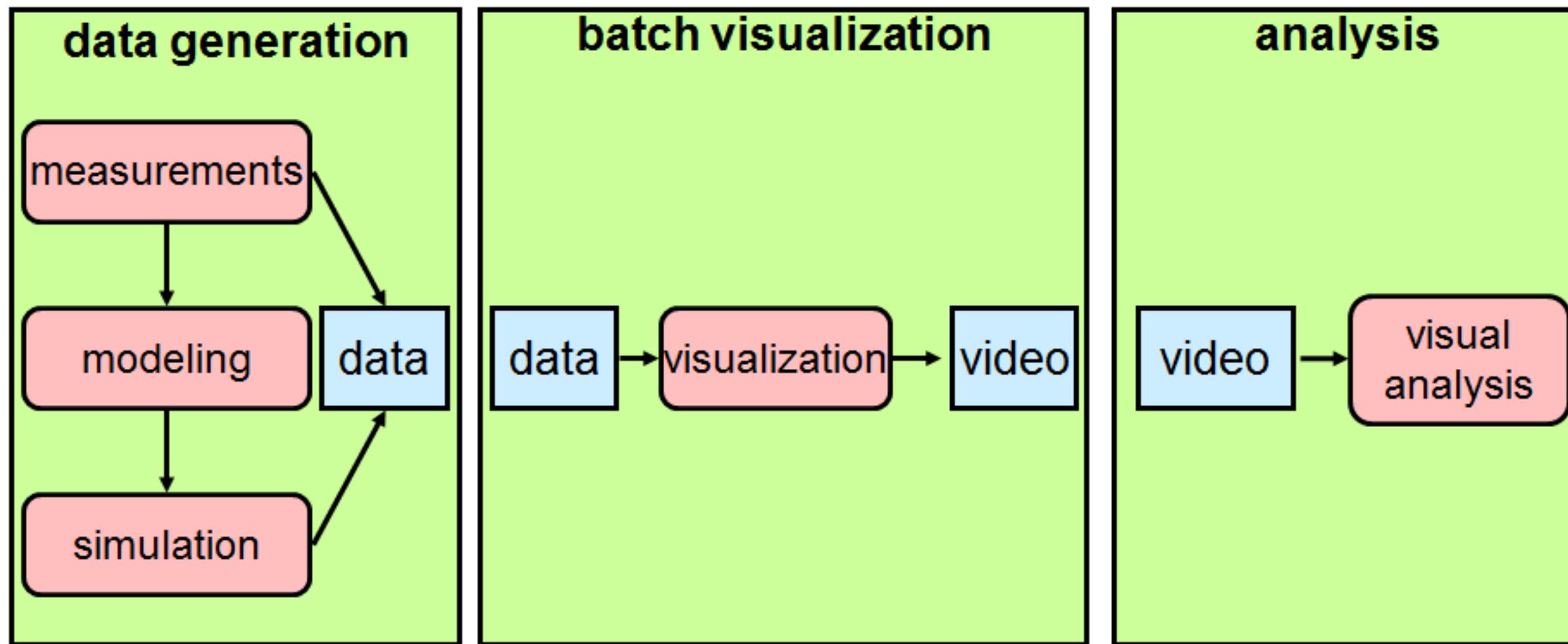
Visualization Pipeline

Visualization - Simulation cycle (example)



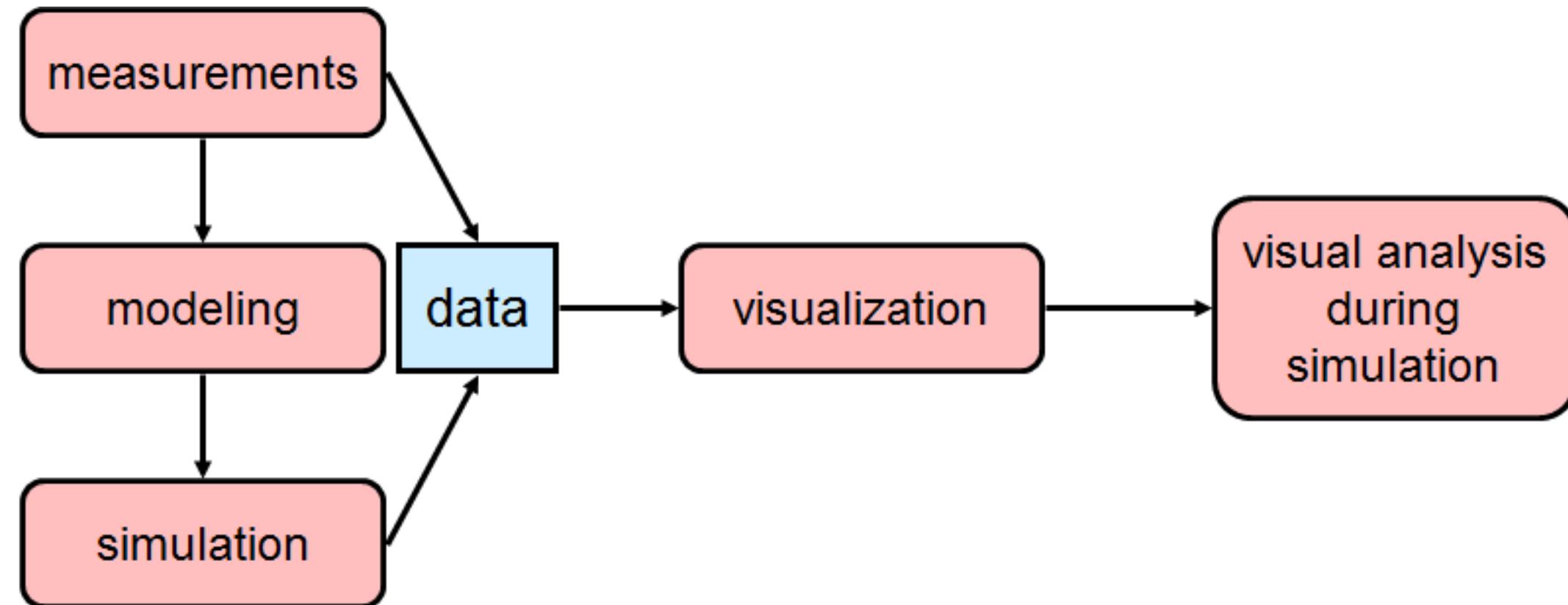
Visualization Pipeline

Example scenarios: Video/movie mode – *offline*, no interaction



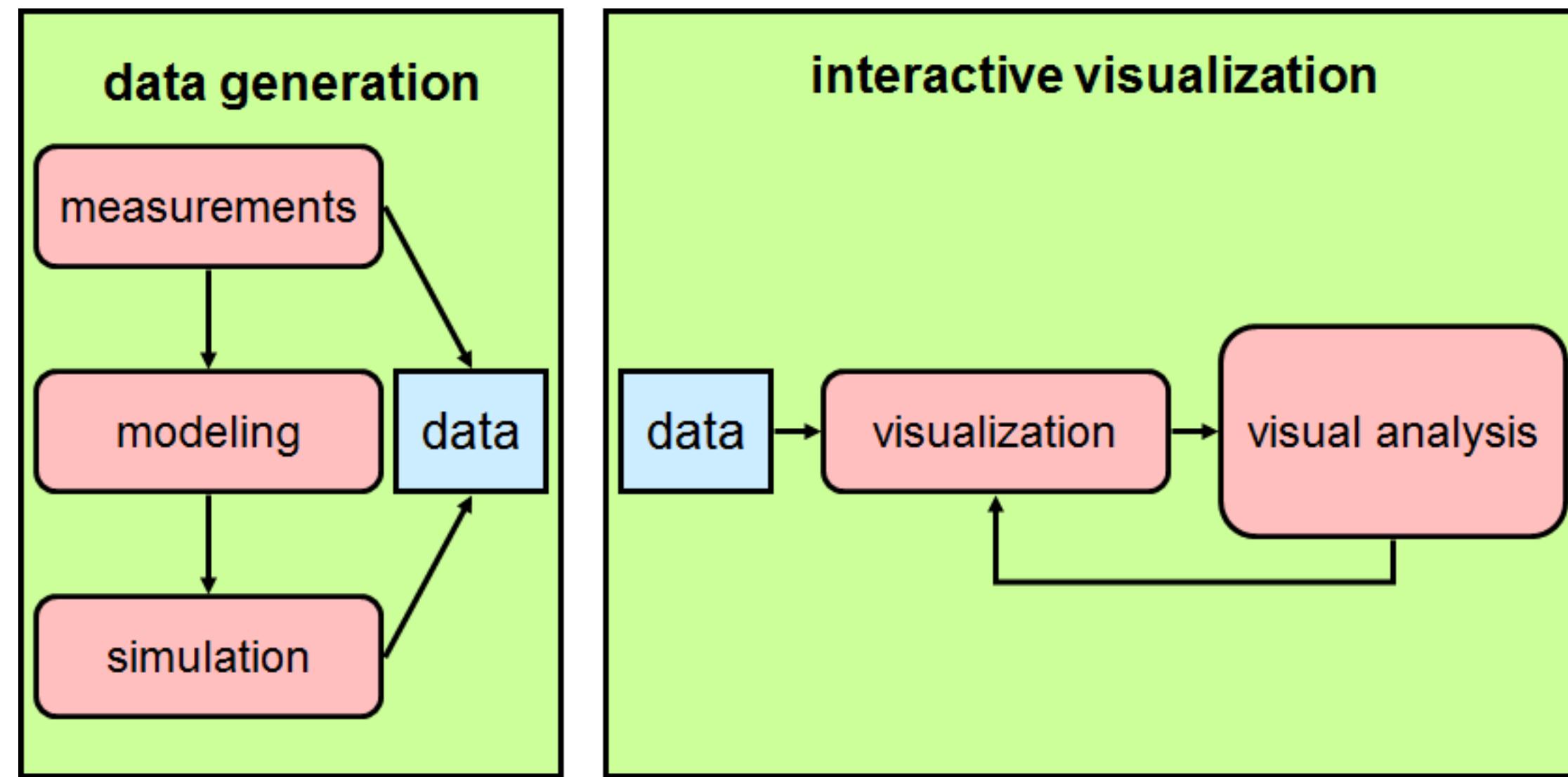
Visualization Pipeline

Example scenarios: Tracking – *online*, no interaction



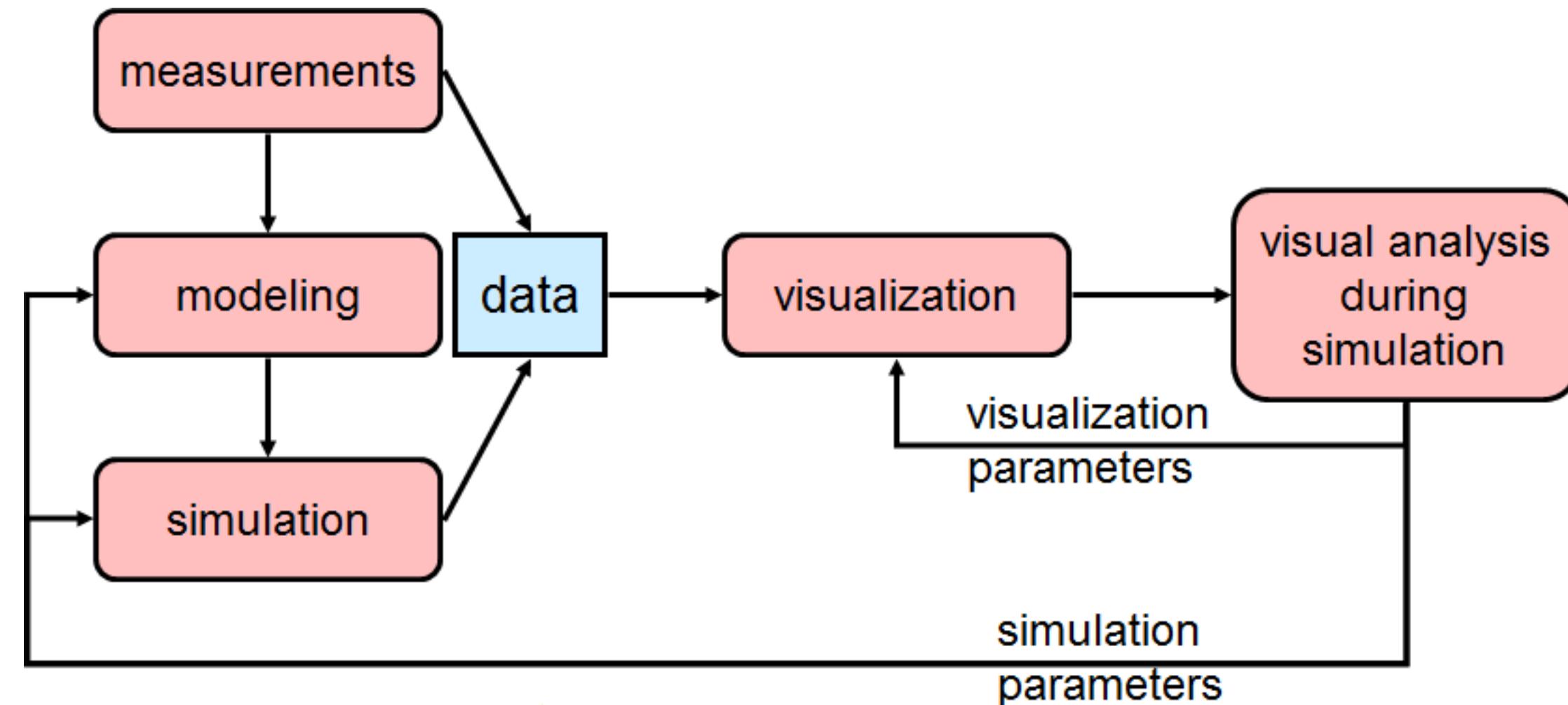
Visualization Pipeline

Example scenarios: Interactive post processing – offline



Visualization Pipeline

Example scenarios: Interactive steering / control



3.2 Sources of Error

Sources of Error

- Data acquisition
 - Sampling
 - Sufficient (spatial) sampling of the data to get what we need?
 - Quantization
 - Conversion of "real" data to representation with enough precision to discriminate the relevant features?
 - Filtering
 - Are we retaining / removing the "important / non-relevant" structures?
 - Frequency / spatial domain filtering: noise, clipping and cropping
 - Selecting the "right" variable
 - Does this variable reflect the interesting features?
 - Does this variable allow for a "critical point" analysis?

Sources of Error

- Functional model for resampling
 - Introduced information by interpolation and approximation?
- Mapping
 - Appropriate choice of graphical primitives?
 - Think of some real-world analogue (metaphor)
- Rendering
 - Need for interactive rendering
 - Often determines the chosen abstraction level
 - Consider limitations of the underlying display technology
 - Data, color, quantization
 - Carefully add "realism"
 - The most realistic image is not necessarily the most informative one

3.3 Data Domain

Data Domain

- Discrete representations
 - Target objects: **continuous**
 - Given data: **discrete** in space and/or time
 - Discrete structures
 - Consist of **samples** → generate **grids/meshes** consisting of **cells**

Data Domain

- Primitives in multi-dimensions

Dimension	Cell	Mesh
1D	Line (edge)	Polyline (Polygon)
2D	Triangle, quadrilateral (e.g. rectangle)	2D mesh
3D	Tetrahedron, hexahedron (e.g. cube), prism, pyramid,...	3D mesh

Data Domain

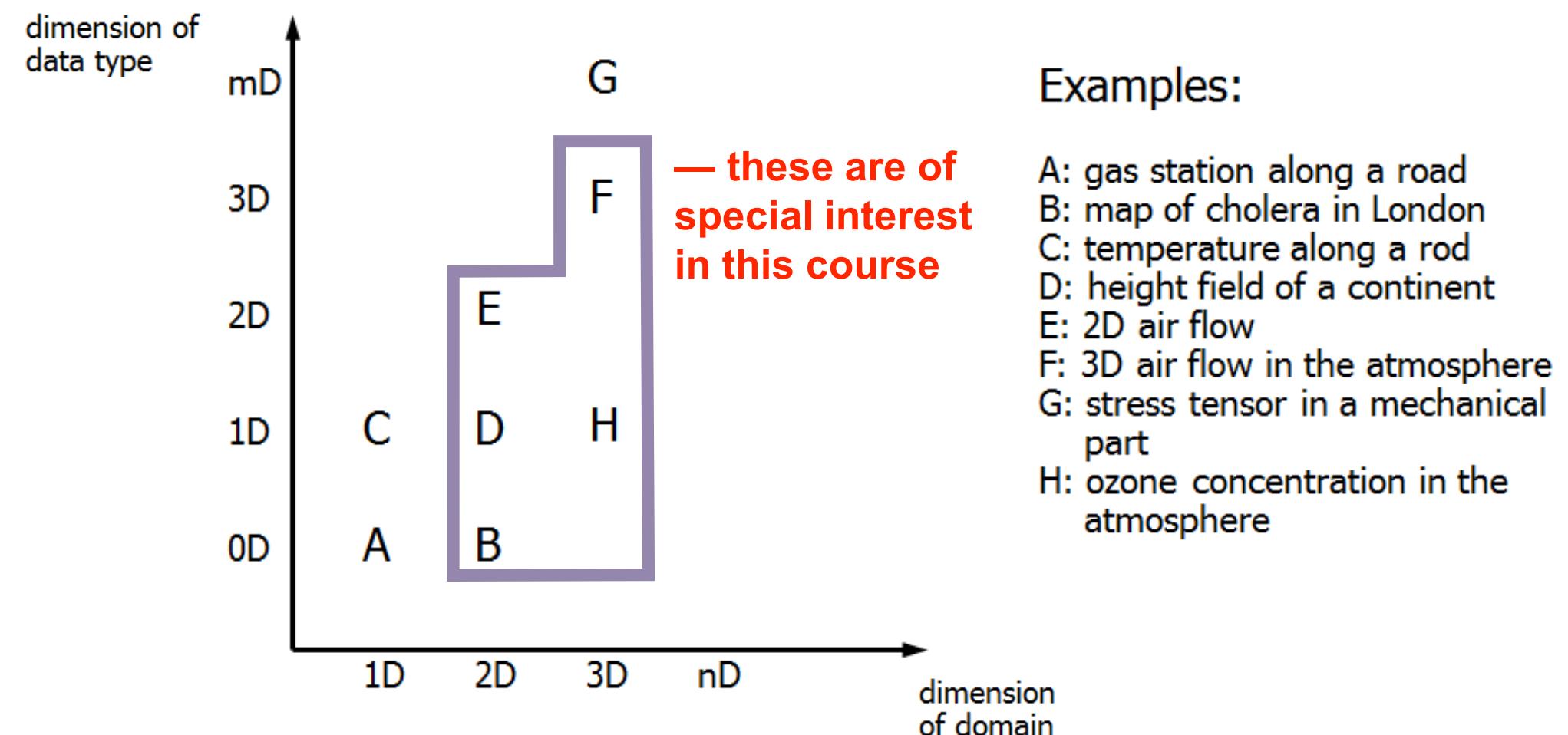
Classification of visualization techniques according to

- **Dimension of the domain**
 - 0D (means unstructured points)
 - 1D, 2D, 3D, nD
- **Type of the data**
 - Scalar, vector, tensor, multi modal
- **Grid type**
 - Uniform cartesian, structured, curvilinear
 - Unstructured, point sets (scattered data)

Data Domain

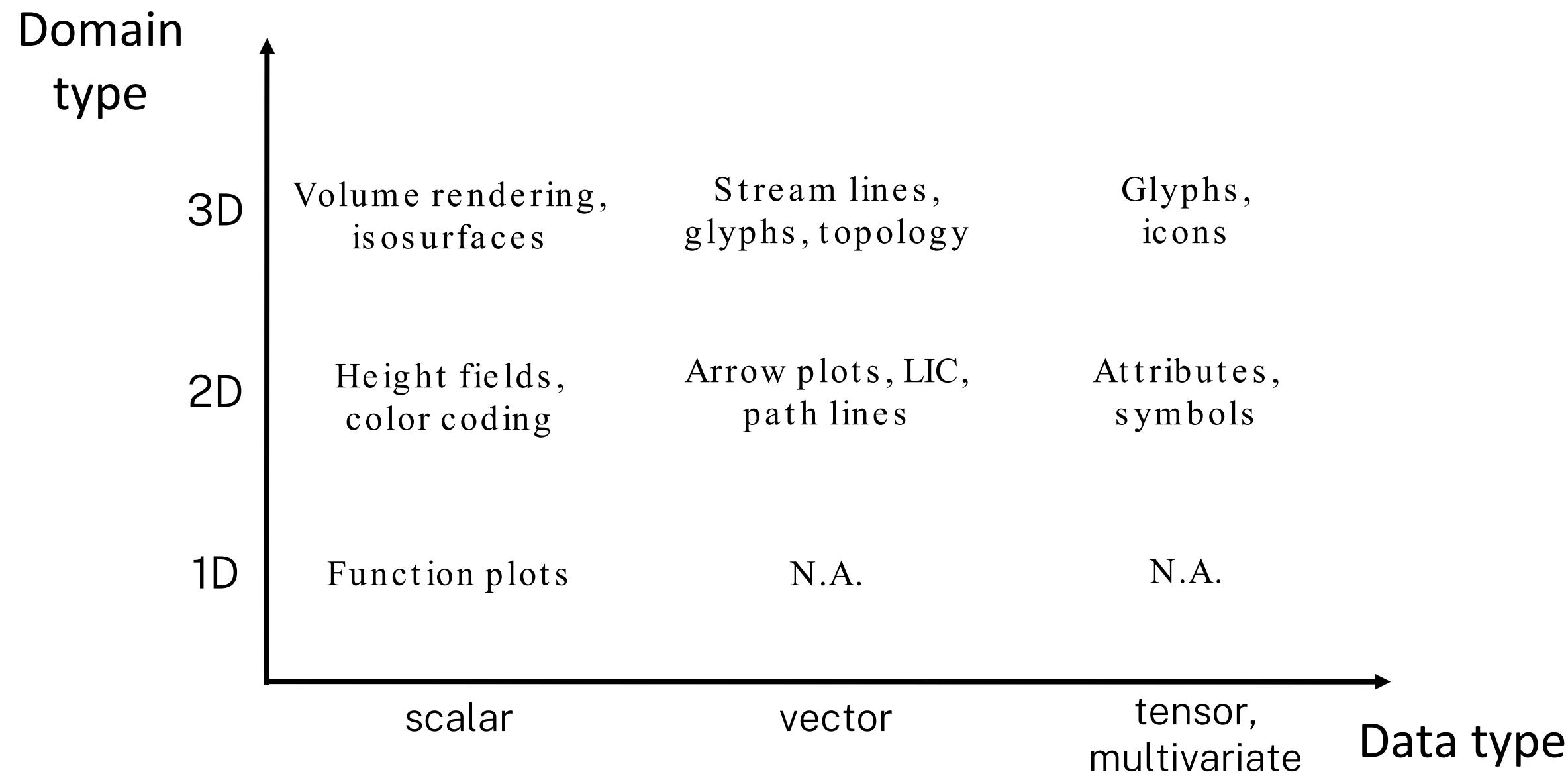
Classification of visualization based on

- Dimension of the problem domain (independent parameters)
- Dimension of the data type (dependent parameters)



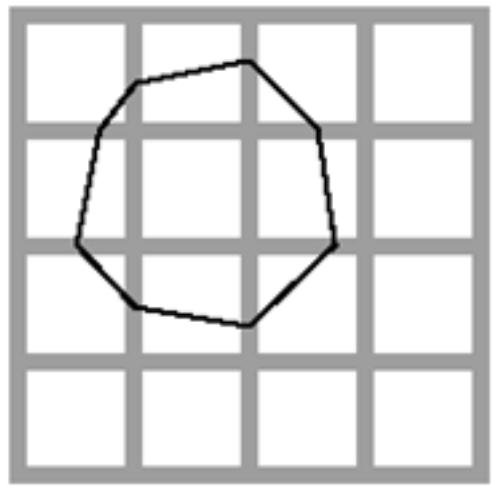
Data Domain

Classification of visualization based on mapping



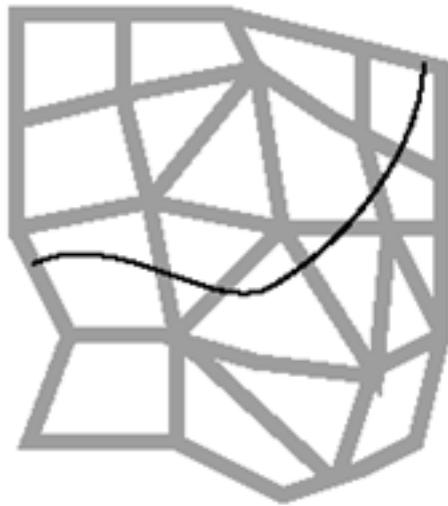
Data Domain

Different data structures → different algorithms



Scalar fields
cartesian

↓
Medical / Volume
Visualization



Vector fields
unstructured

↓
Flow
Visualization



Trees, graphs
Tables

↓
Information
Visualization

Data Domain Composition

- (Geometric) shape of the domain
 - Determined by the positions of sample points
- Domain characterized by
 - Dimension
 - Influence
 - Structure wie hängen die Punkte zusammen
- Influence of data points
 - *Values at sample points influence the data distribution in a certain region around these samples*
 - To reconstruct the data at arbitrary points within the domain, the distribution of all samples must be calculated (interpolation)

Data Domain

Influence types

- **Point influence**
 - Only influence on the point itself
- **Local influence**
 - Only influence within a certain region around the point
 - Voronoi-diagram
 - Cell-wise interpolation
- **Global influence**
 - Each sample might influence any other point within the domain
 - Material properties for whole object
 - Scattered data interpolation

3.4 Coordinate Systems

Coordinate Systems

2D coordinate systems

- Cartesian coordinates

$$P = (x, y)$$

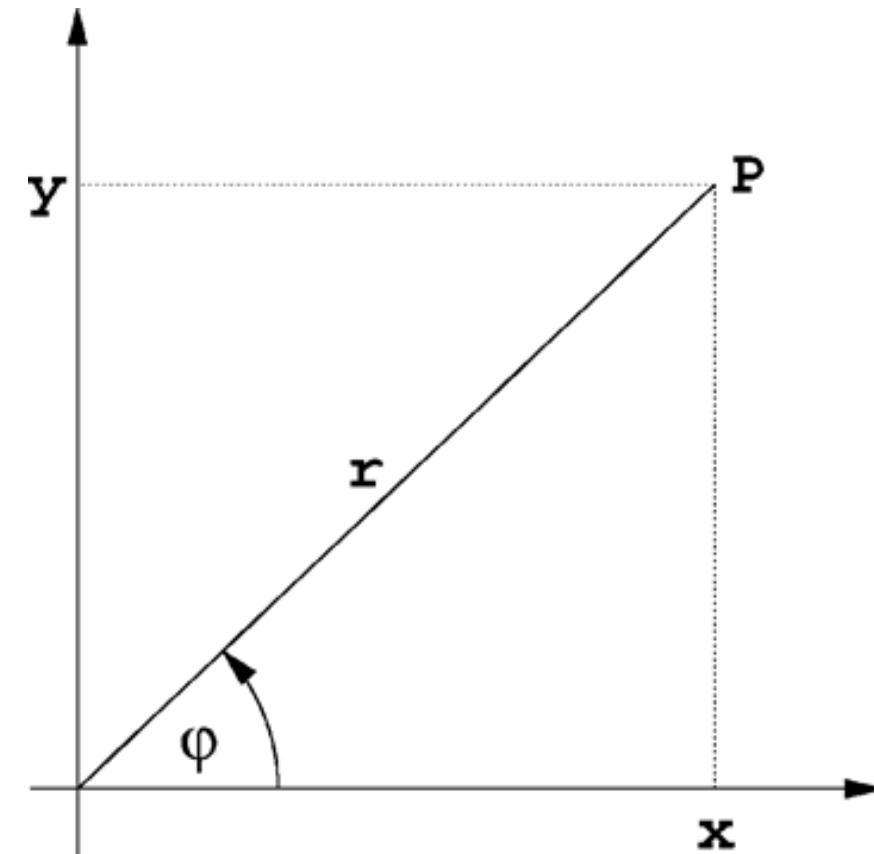
$$P = (r, \varphi)$$

- Polar coordinates

$$x = r \cos \varphi \quad y = r \sin \varphi$$

$$r = \sqrt{x^2 + y^2}$$

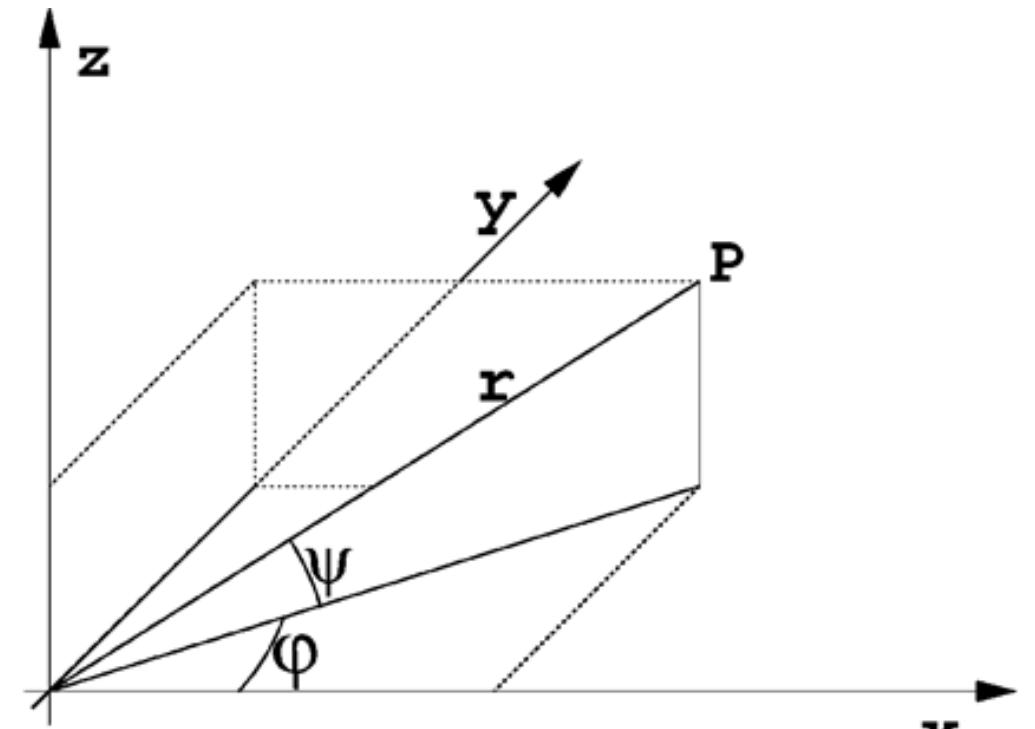
$$\varphi = \arctan \frac{y}{x}$$



Coordinate Systems

3D coordinate systems

- Cartesian coordinates $P = (x, y, z)$
- Cylindrical coordinates $P = (r, \varphi, z)$
 - This is like polar coordinates in 2D
- Spherical coordinates $P = (r, \varphi, \psi)$
 - where $x = r \cos \varphi \cos \psi$
 - $y = r \sin \varphi \cos \psi$
 - $z = r \sin \psi$



3D cartesian and spherical coordinates

Coordinate Systems

P-space \leftrightarrow C-space

- Transformation, where
 - P-space: physical space
 - C-space: corresponding uniform computer representation

3.5 Data Structures

Data Structures

- Requirements
 - Convenience of access
 - Space efficiency
 - Lossless vs. lossy
 - Portability
 - Binary - less portable, more space/time efficient
 - Text - human readable, portable, less space/time efficient
- Definitions
 - Scattered data
 - Arbitrarily distributed points with no connectivity in between
 - Otherwise
 - Data is composed of cells bounded by grid lines

Data Structures

Topology vs. Geometry

- **Geometry**
 - Specifies the position of the data
- **Topology**
 - Specifies the structure (connectivity) of the data
 - Main concern: qualitative questions about geometric structure
 - Are there holes?
 - Is everything connected?
 - Can it be split into individual parts?

Data Structures

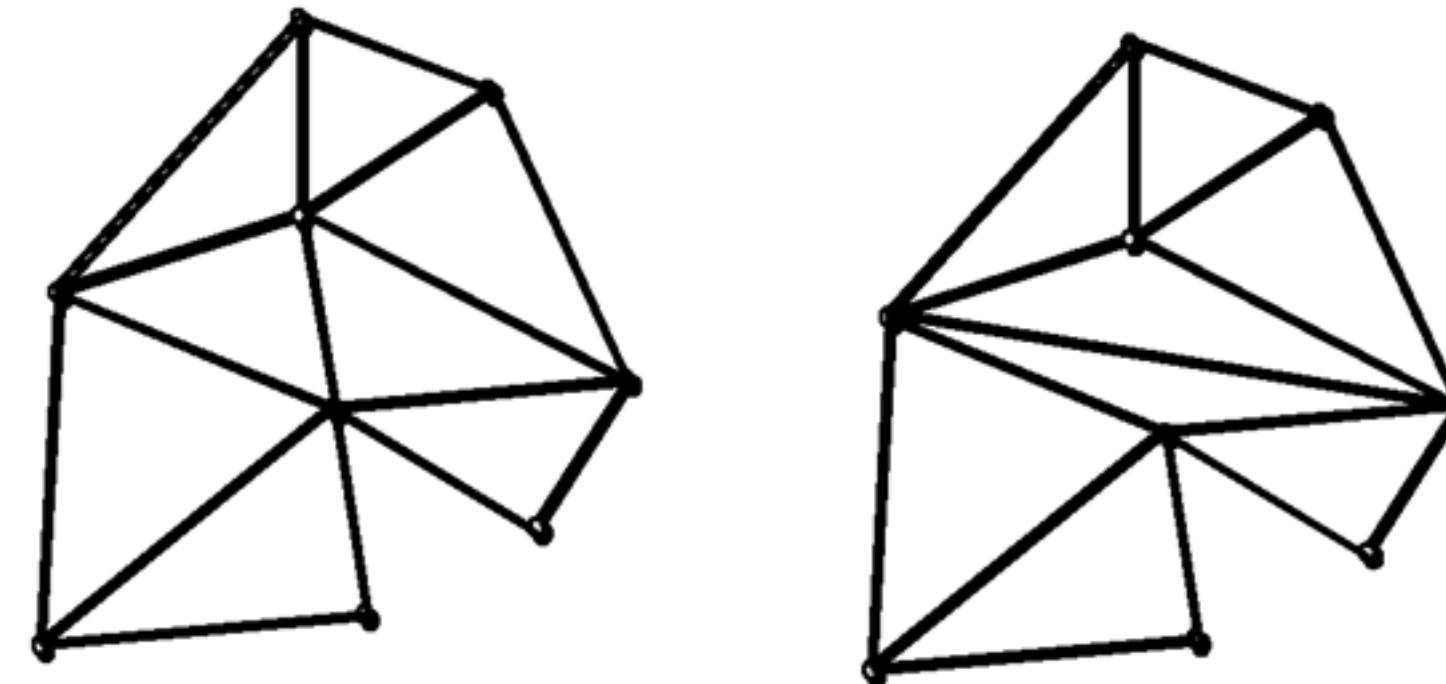
- Example: topological map of underground
 - Does not tell how far one station is from the other, but rather how the lines are connected



Data Structures

Topology

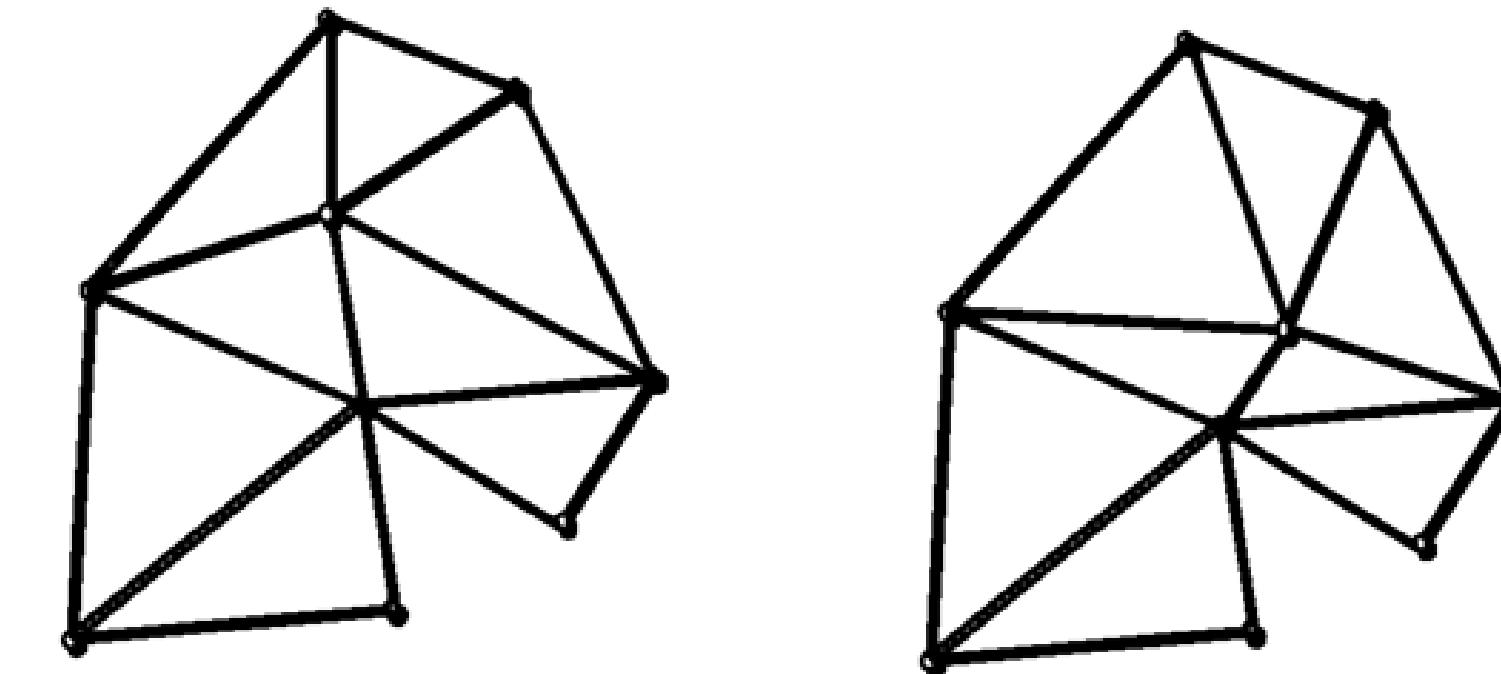
- Properties of geometric shapes that remain unchanged even under distortion



Same geometry (vertex positions), different topology (connectivity)

Data Structures

- Shapes that can be transformed into each other without tearing or introducing new connections are *topologically equivalent*



Topologically equivalent

Data Structures

Discrete representation of data: meshes / grids

- In general
 - List of vertices (explicit or implicit)
 - Global vertex index (explicit or given by order)
 - List of cells (explicit or implicit)
 - Global cell index (explicit or given by order)

Data Structures

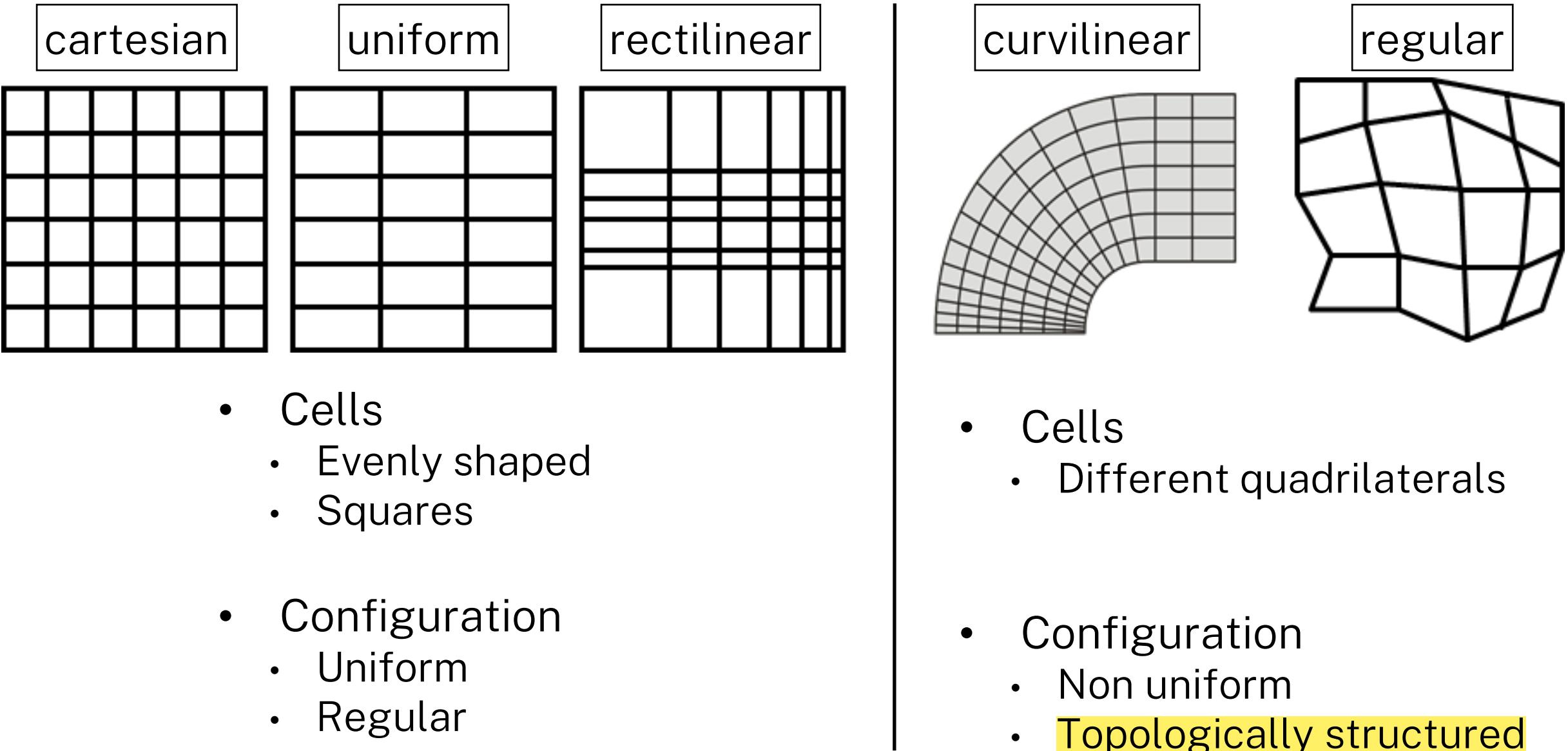
Discrete representation of data: meshes / grids

- Optional
 - List of edges, list of faces (3D)
 - Type flags
 - Edge: interior/boundary/complex/feature edge
 - Vertex: interior/boundary/complex/feature vertex (e.g. "corner")
 - Adjacencies/incidence (neighborhood relation)
 - cell → vertices, cell → faces, cell → edges, face → edges, ...

Data Structures

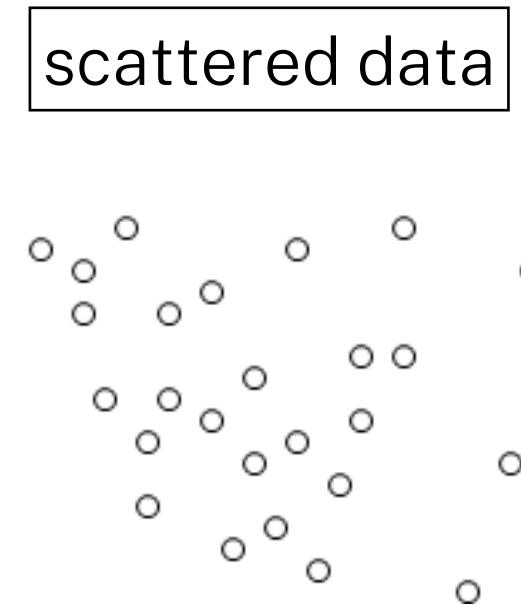
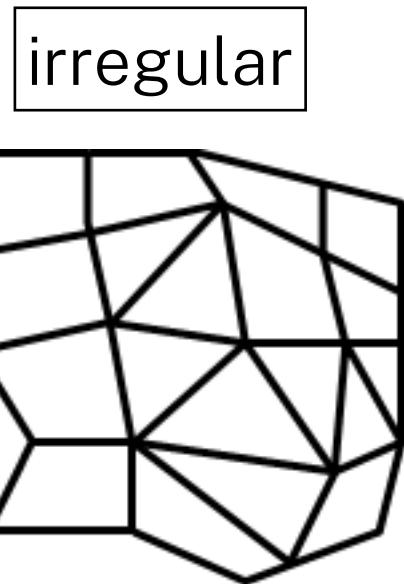
Structured meshes / grids

- Regular topology,
regular / irregular
geometry



Data Structures

Unstructured meshes / grids

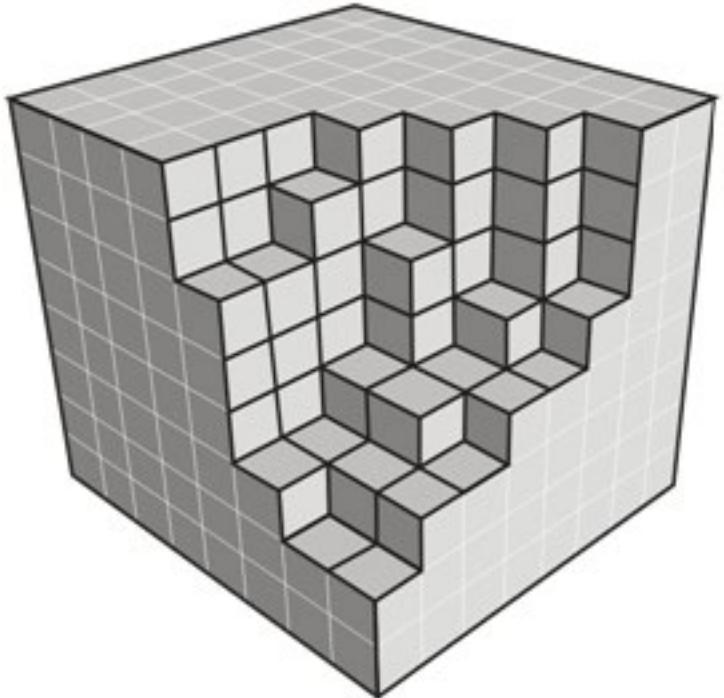


- Cells
 - Triangles (rarely rectangles or polygons)
- Configuration
 - Unstructured
 - Irregular topology and geometry

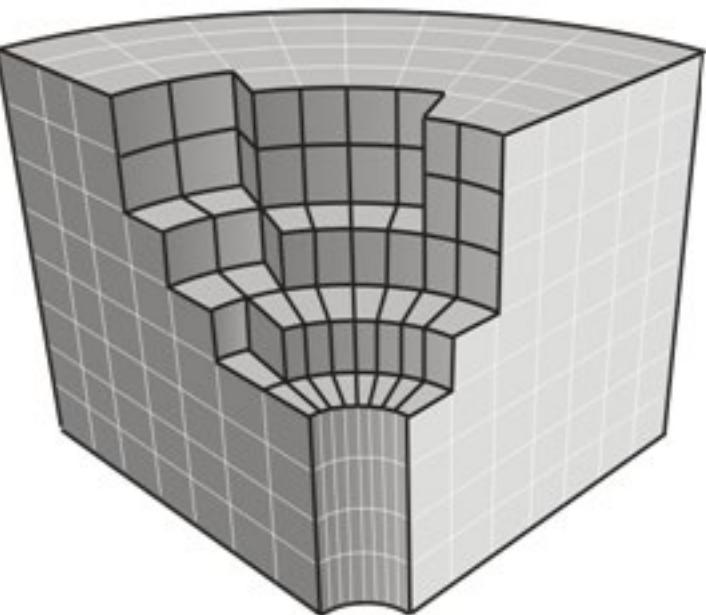
Data Structures

Meshes in 3D

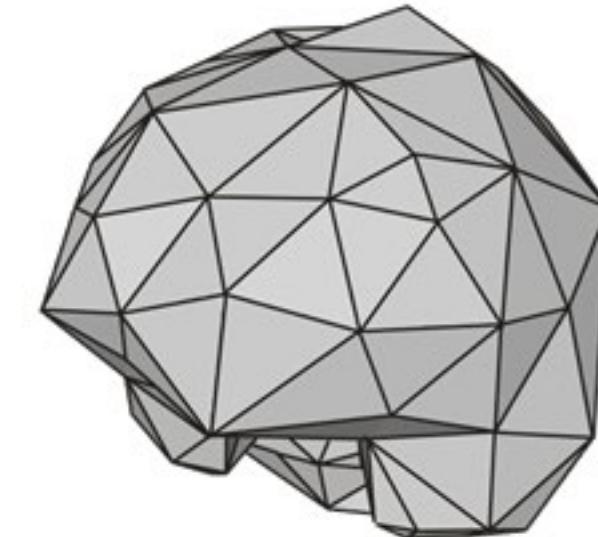
uniform rectilinear



curvilinear



unstructured



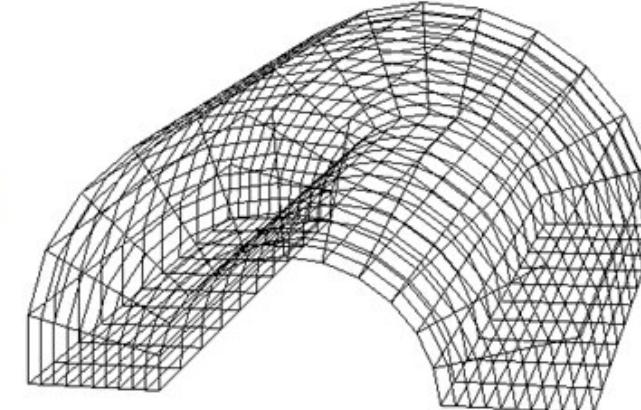
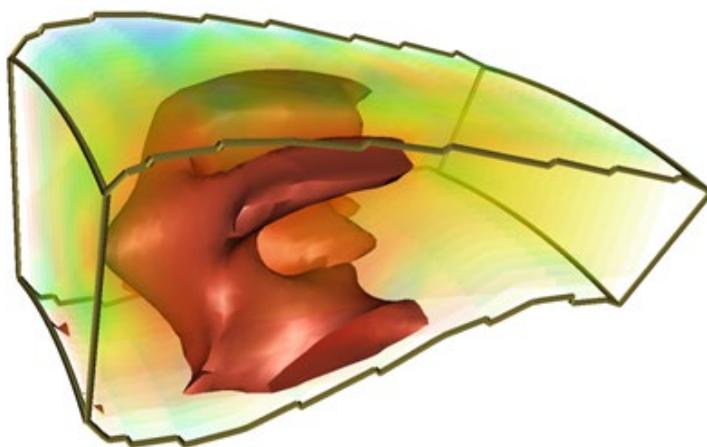
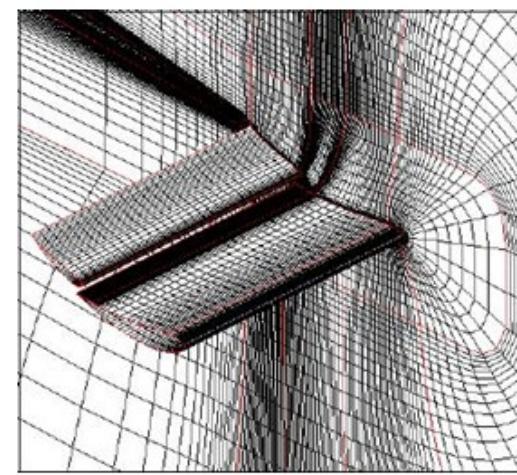
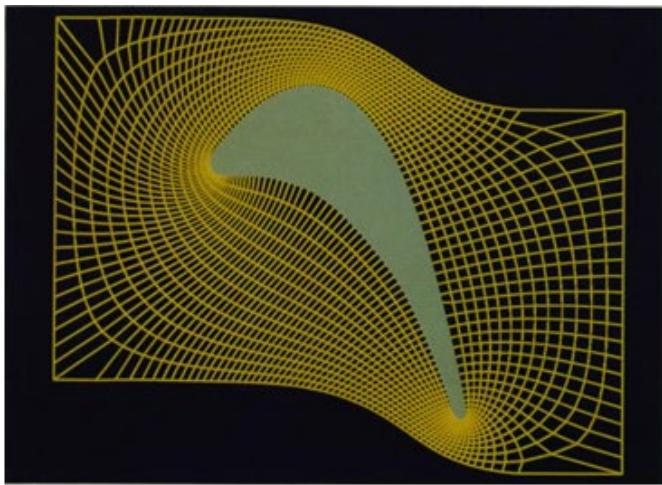
- Cells
 - Uniform hexahedra
- Configuration
 - Uniform
 - Regular

- Cells
 - Different shaped hexahedra
- Configuration
 - Non-uniform
 - Structured

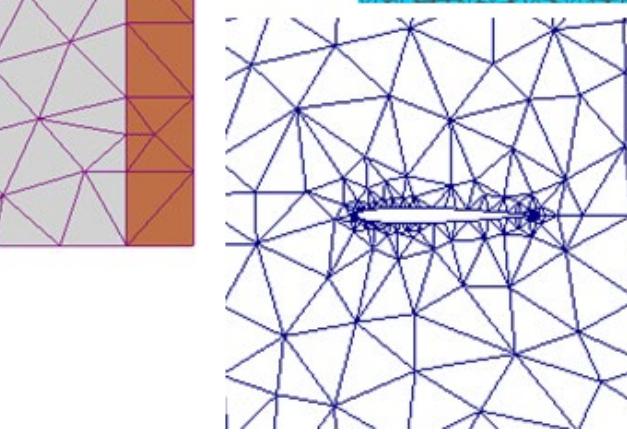
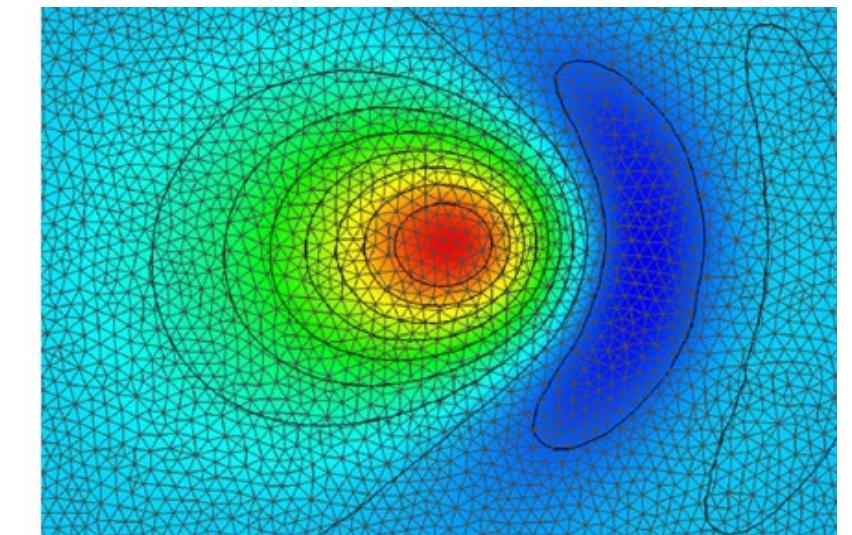
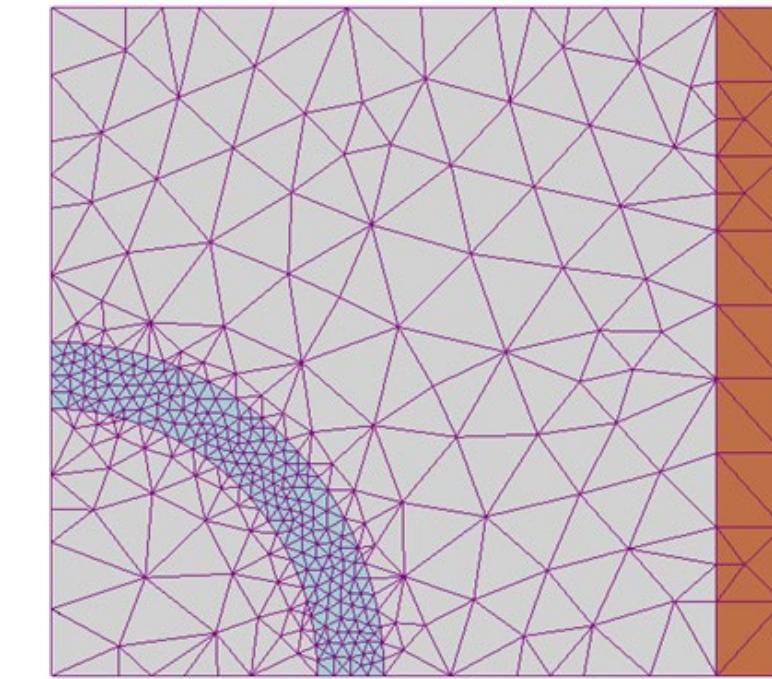
- Cells
 - Tetrahedra, pyramids, hexahedra, prisms
- Configuration
 - Unstructured

Data Structures

Examples of grids



structured



unstructured

3.5.1 Structured Grids

Structured Grids

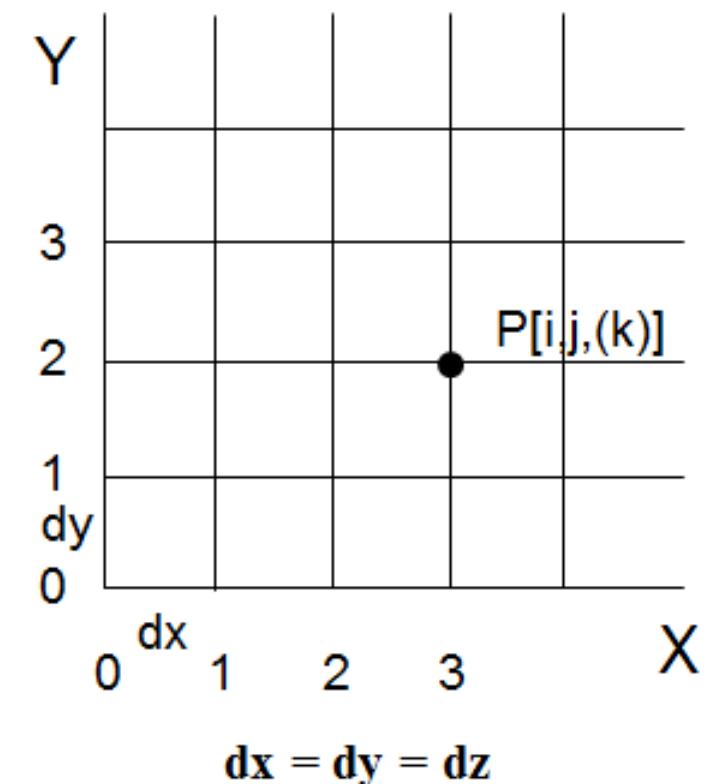
Characteristics of structured grids

- Simple computation
- Typical structure
 - Often parallelograms (hexahedra)
 - Cells being equal or non-linearly distorted
- May require more elements or badly shaped elements to cover the underlying domain
- Topology
 - Implicitly given by an n-vector of dimensions
- Geometry
 - Explicitly given by an array of points
 - Every interior point has the same number of neighbors

Structured Grids

Cartesian or equidistant grids

- Structured grid
- Sequential numbering of cells and points
 - w.r.t increasing X, then Y, then Z
 - or vice versa
- Number of points
 - $N_X \cdot N_Y \cdot N_Z$
- Number of cells
 - $(N_X - 1) \cdot (N_Y - 1) \cdot (N_Z - 1)$



Structured Grids

Cartesian or equidistant grids

- Vertex positions are given implicitly from $[i, j, k]$
 - $P[i, j, k].x = \text{origin} + i*dx$
 - $P[i, j, k].y = \text{origin} + j*dy$
 - $P[i, j, k].z = \text{origin} + k*dz$
- Global vertex index: $I[i, j, k] = k*NY*NX + j*NX + i$
 - $k = I / (NY*NX)$
 - $j = (I \% (NY*NX)) / NX$
 - $i = (I \% (NY*NX) \% NX) = I \% NX$
- Global index allows for linear storage scheme
 - Wrong access patterns may destroy cache coherence

Data Structures

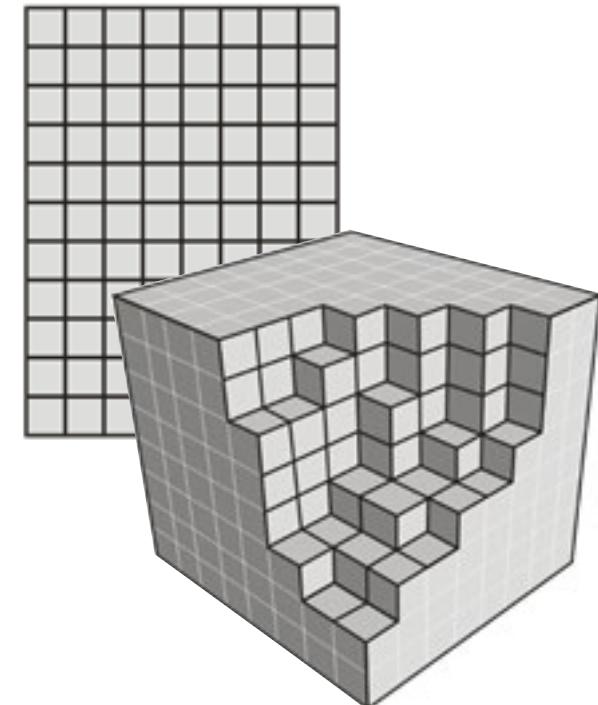
Uniform grids

- Like cartesian grids
- Equal cells, but with **different resolution** in at least one dimension ($dx \neq dy \neq dz$)
 - Constant spacing in each dimension → same indexing scheme as for cart. grids
- Applications: data generated by 3D imaging devices w. different sampling rates for each dimension, e.g.:
 - Medical volume data consisting of slice images
 - Slice images with square pixels ($dx == dy$)
 - Larger (or smaller) slice distance ($dz > (dx == dy) \text{ || } dz < (dx == dy)$)

Structured Grids

Typical grid type in medical imaging: 2D/3D uniform grid

- Position of cells / vertices is given implicitly
 - Dimensions of Grid N_X, N_Y, N_Z
 - Total number of cells: $(N_X - 1) \times (N_Y - 1) \times (N_Z - 1)$
 - Cell size $\Delta x, \Delta y, \Delta z$ (Note: data is in the cells!)
 - Distance of sampling points in x-, y- and z-direction
 - Spatial resolution
 - Pixel: $\Delta x \times \Delta y$, Voxel: $\Delta x \times \Delta y \times \Delta z$
 - Uniform grids (usually anisotropic): $\Delta x = \Delta y \neq \Delta z$
 - Dimensions in continuous space: $X = \Delta x \times N_X, Y = \Delta y \times N_Y, Z = \Delta z \times N_Z,$



Structured Grids

Representation of uniform grids

- Data stored as 1D-array with index i
 - $i = N_X \times N_Y \times z + N_X \times y + x$
 - *Origin*: X_0, Y_0, Z_0 (usually $X_0 = 0, Y_0 = 0, Z_0 = 0$)
- Implementation (C++): 1D- or multi-dimensional array

```
(DataType *data = new DataType[NX * NY * NZ];  
  
value = data[k * NX * NY + j * NX + i];
```

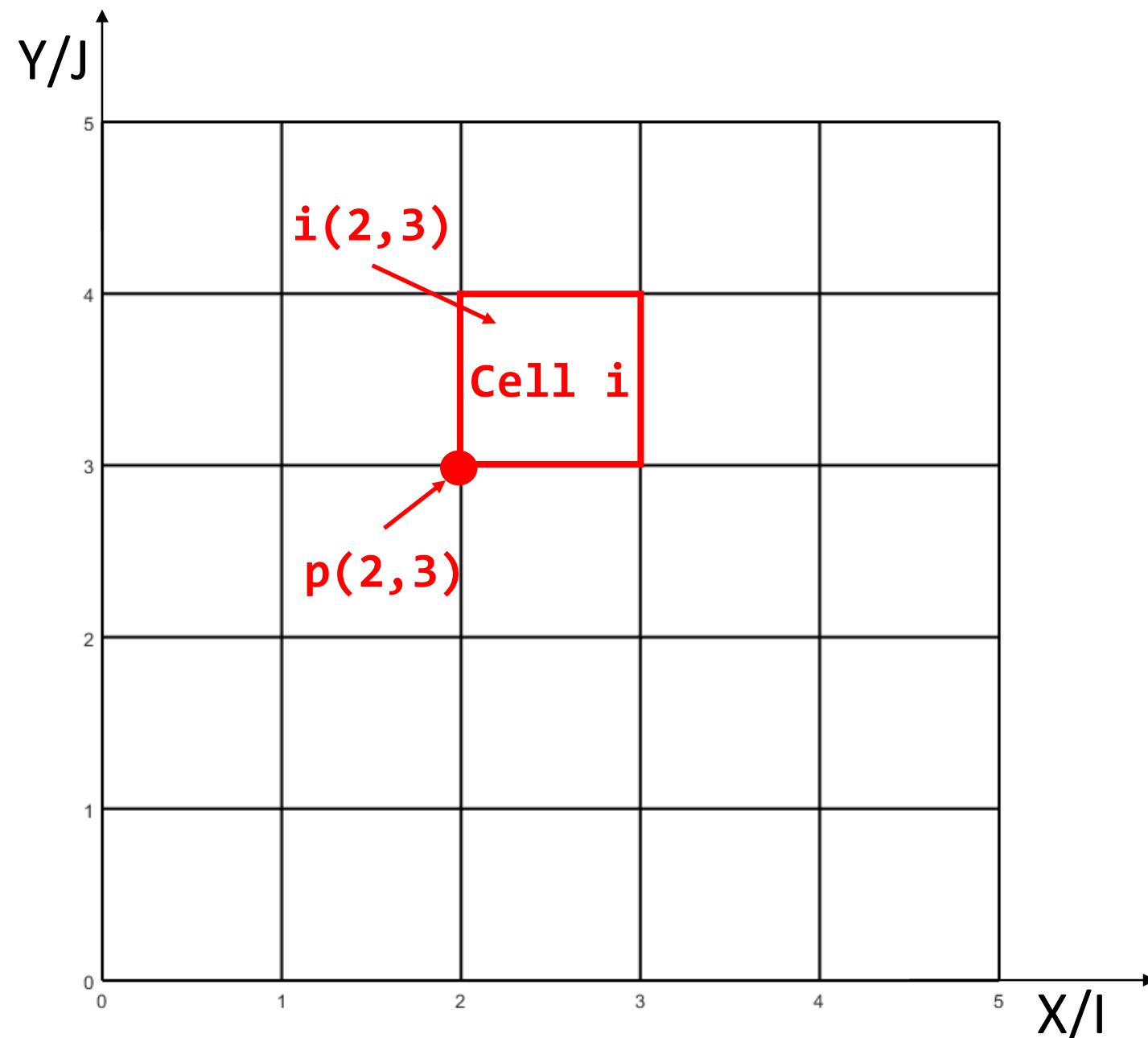
```
DataType ***data;  
data = new DataType**[NX];  
for (int i = 0; i < NX; ++i) {  
    data[i] = new DataType*[NY];  
    for (int j = 0; j < NY; ++j) {  
        data[i][j] = new DataType[k];  
    }  
}  
value = data[i][j][k];
```

- Note: often, i, j, k is used instead of x, y, z to distinguish between index and voxel/world coordinates

Structured Grids

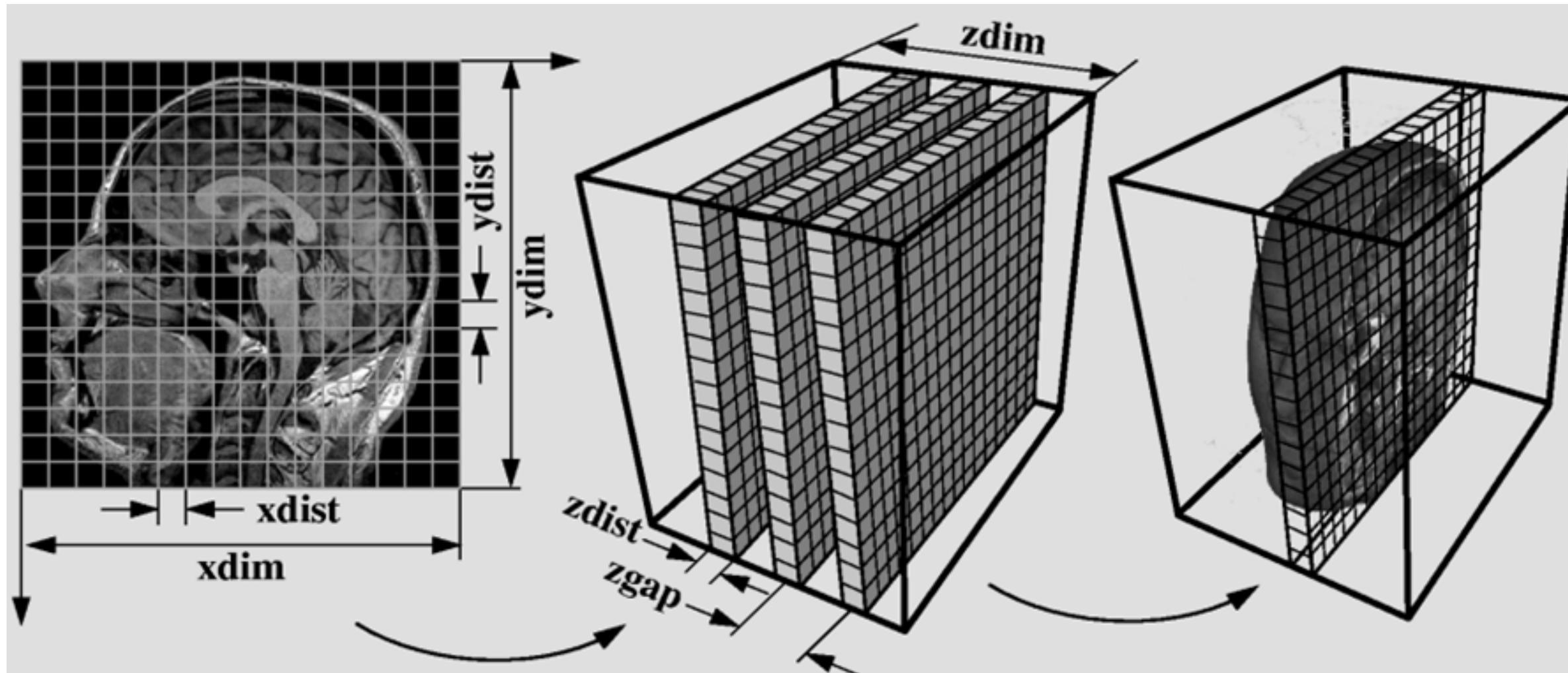
Example: 2D/3D uniform grid

- Example in 2D (voxel → pixel)
 - 6×6 grid → 36 points, 25 cells
- Coordinate and data index
 - $p(2,3) = 6 * 3 + 2 = 20$
 - $i(2,3) = 5 * 3 + 2 = 17$
 - Cell contains the data value (e.g. image sample)
- Note
 - Discrete index coordinates
 - i, j and k are integer values
 - Independent of spacing!



Data Structures

Uniform grids in medical imaging



Slice image

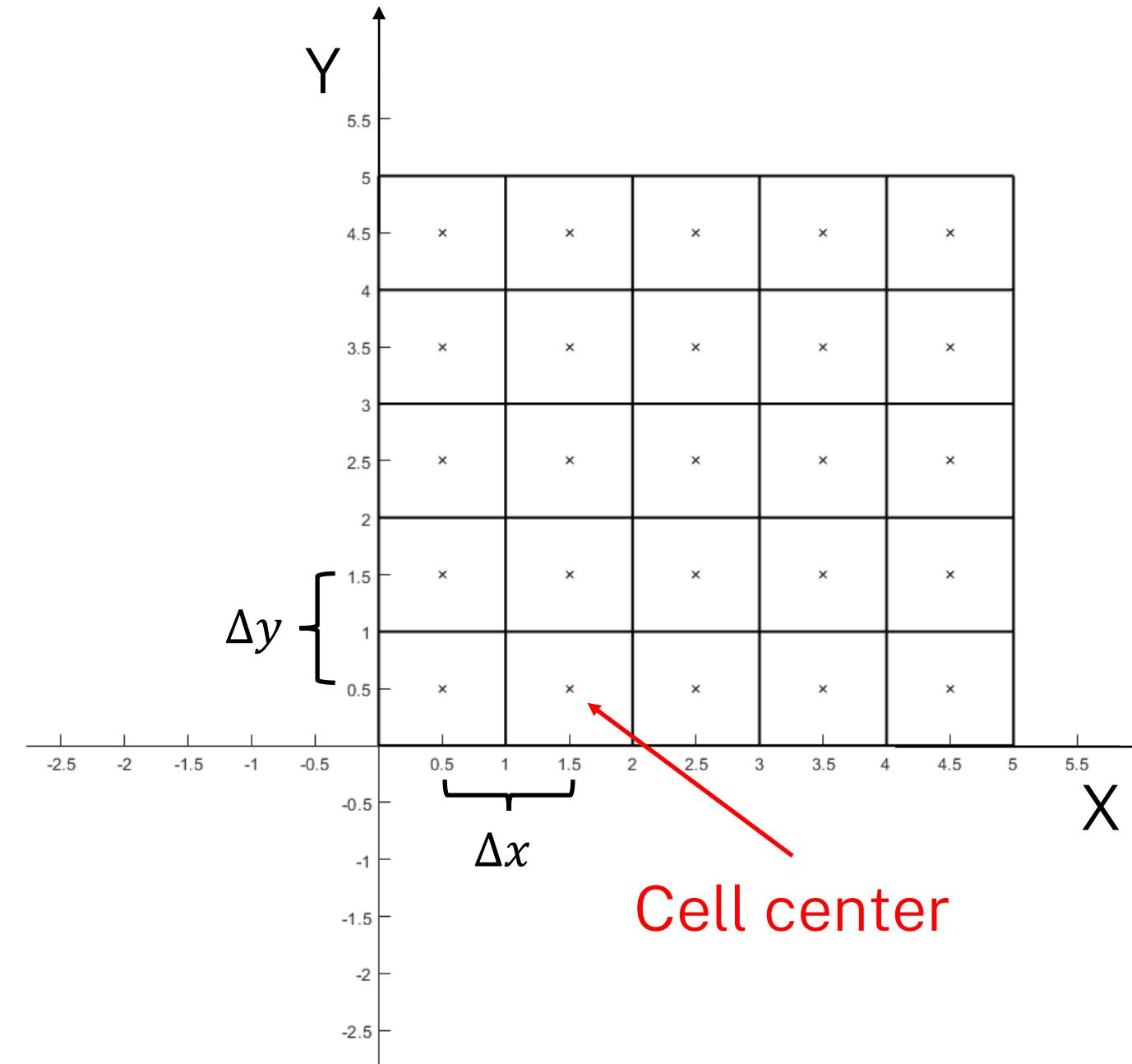
Stack of slice images

Volume dataset

Data Structures

Impact of cell spacing

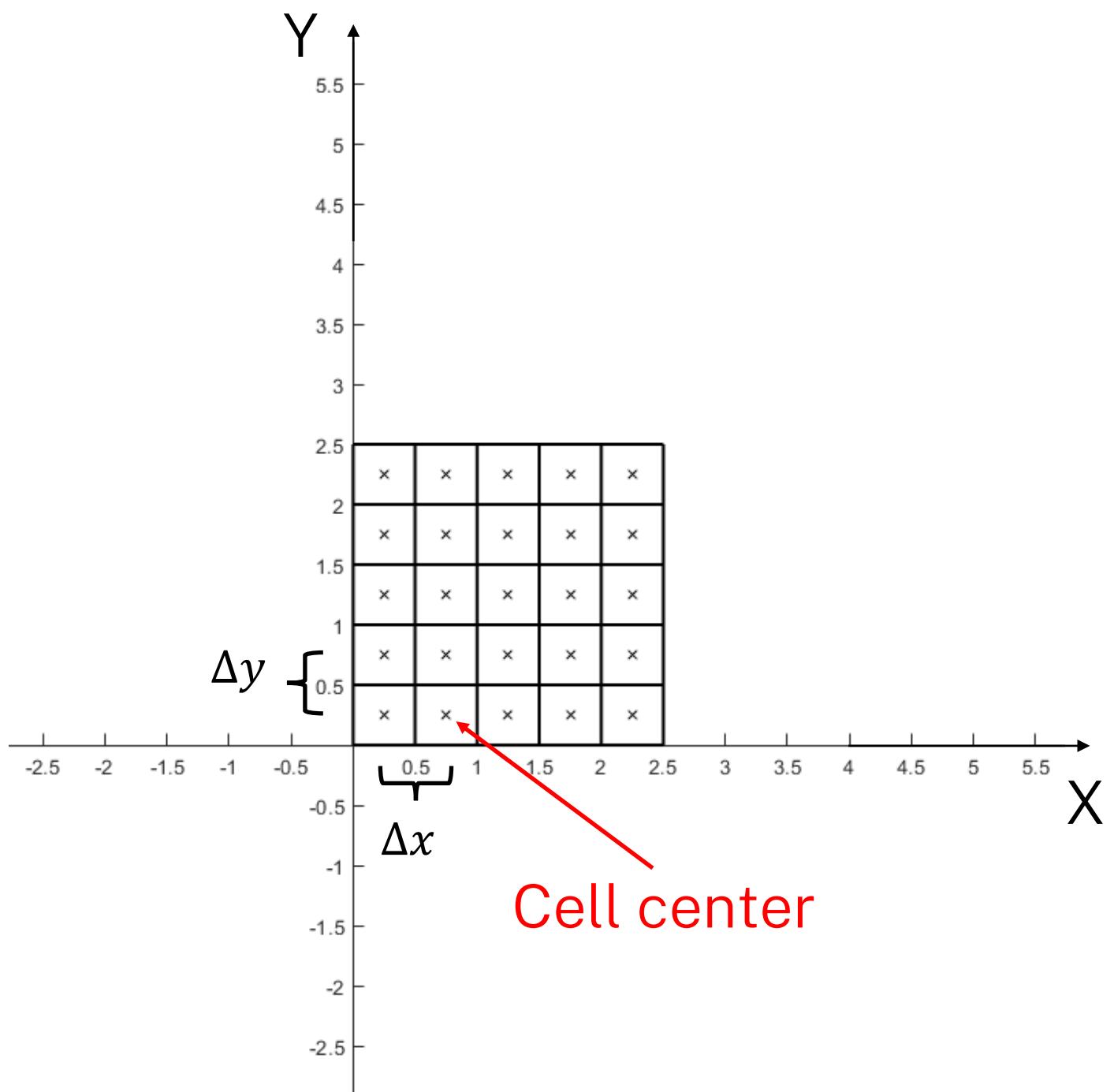
- Data continuous on $[0..X, 0..Y, 0..Z]$
- Relative to data set
- Dependent on spacing
- Often anisotropic, sometimes non-orthogonal
- Data center (usually) at 0.5 (w.r.t. cell)
- Directly related to data in memory



Data Structures

Impact of cell spacing

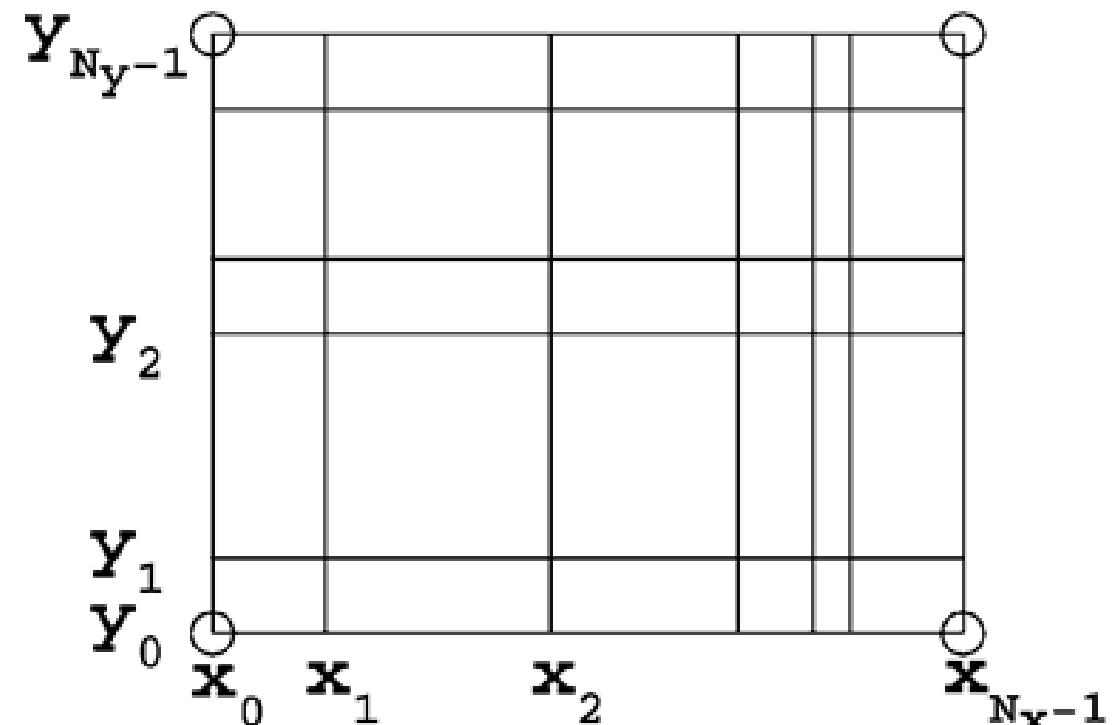
- Same grid, same data
- Assume $\Delta x = \Delta y = 0.5$
- Data center still at 0.5 w.r.t. cell
 - Now 0.25 in world coordinates
- Affects almost all calculations, algorithms and visualization aspects
 - E.g. interpolation, differentiation...
- Neglecting cell spacing is a common implementation error



Data Structures

Rectilinear grids

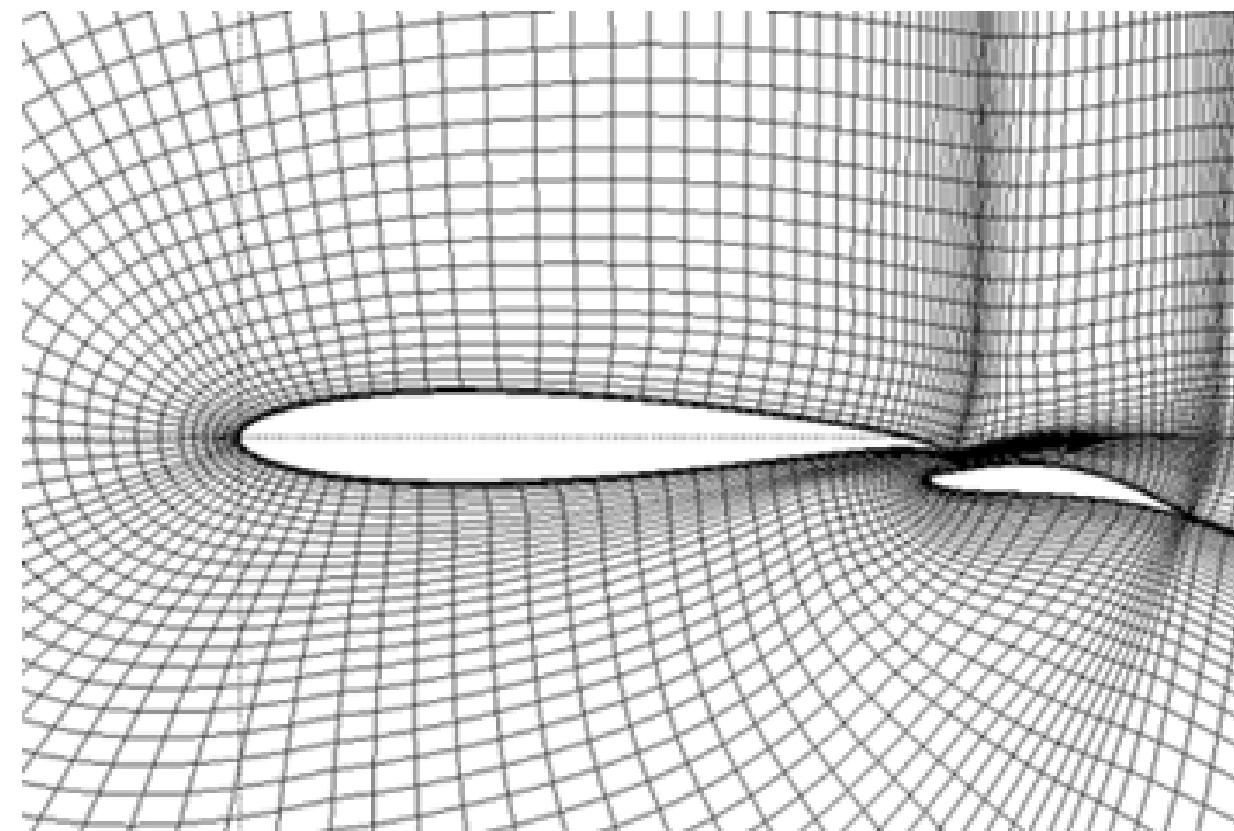
- Topology still regular and implicit
- But: irregular spacing between grid points (Geometry)
 - Non-linear spacing of positions along either axis
- Spacing must be stored explicitly
 - `x_coord[NX]`
 - `y_coord[NY]`
 - `z_coord[NZ]`
- and
- `data[r]` with $r = k*NX*NY + j*NX + i$



Data Structures

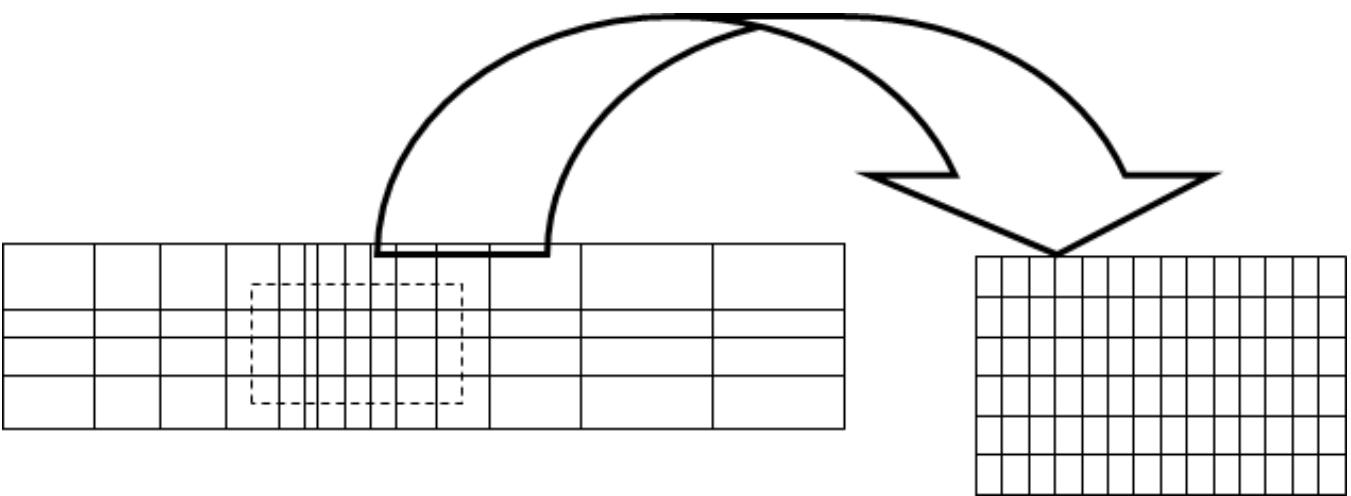
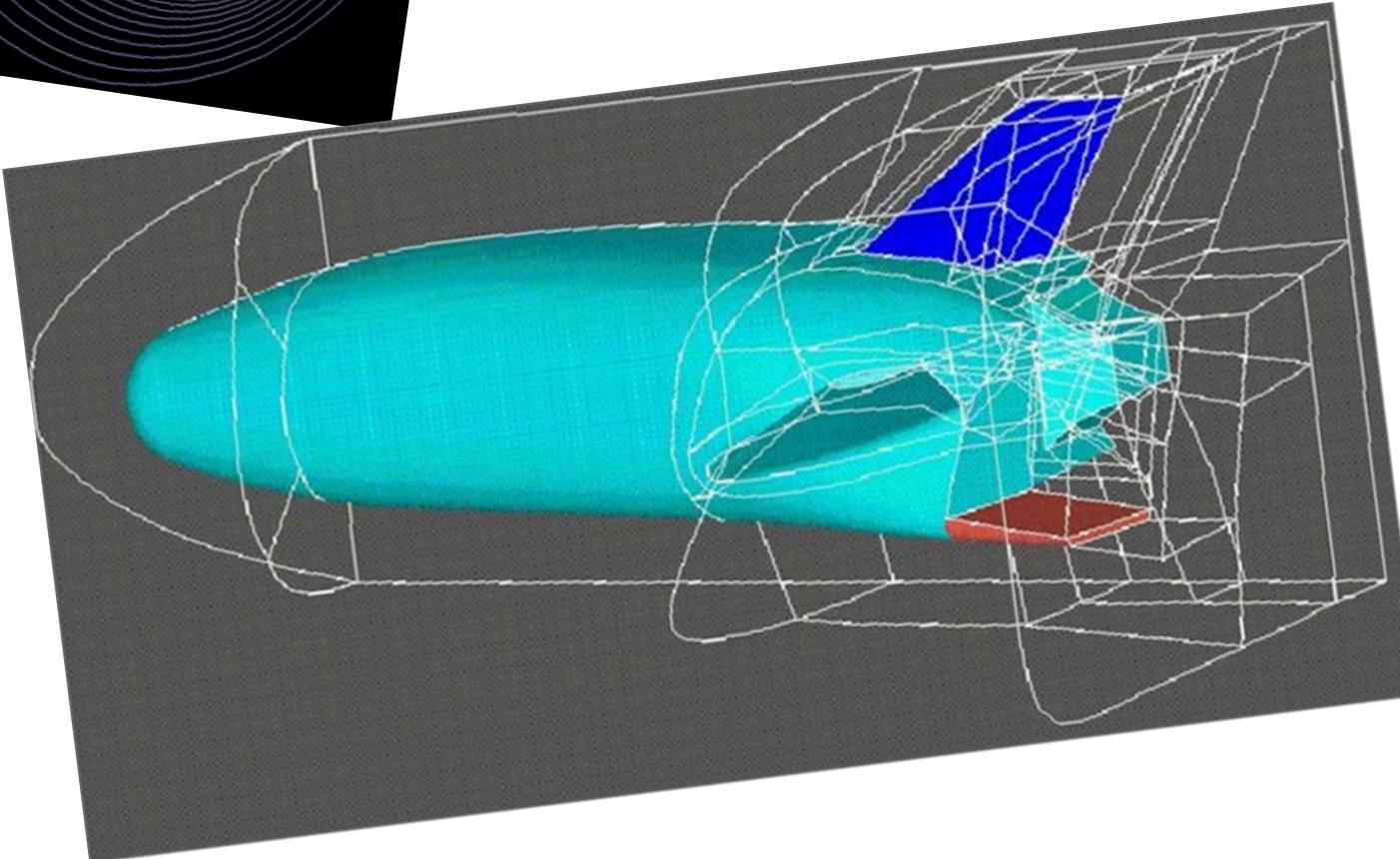
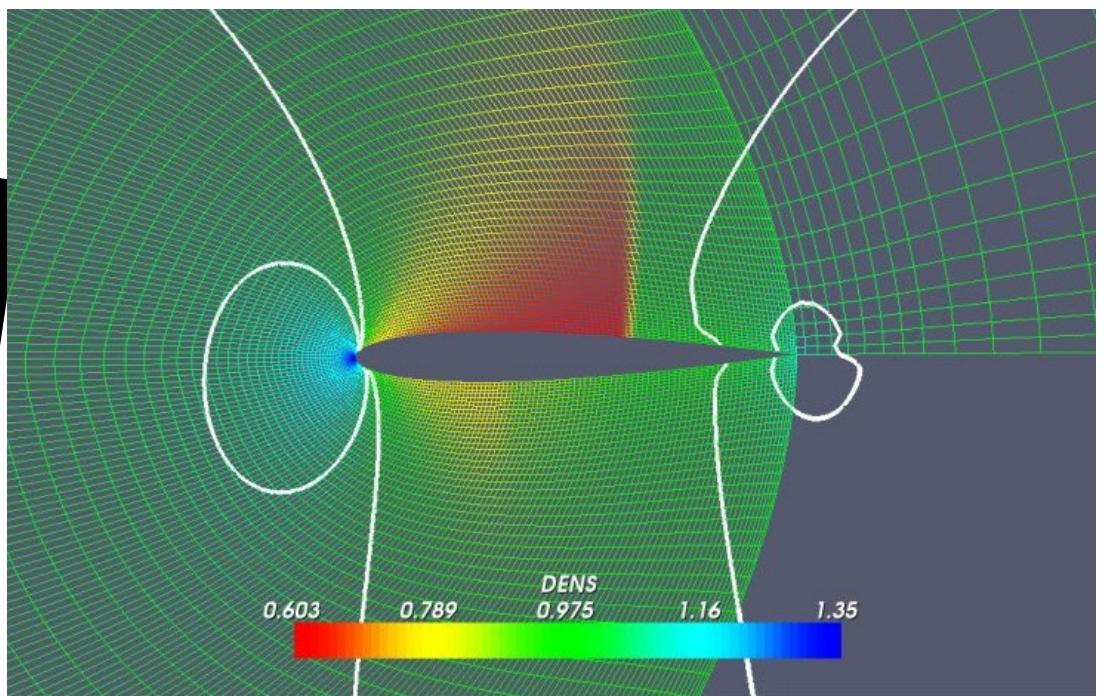
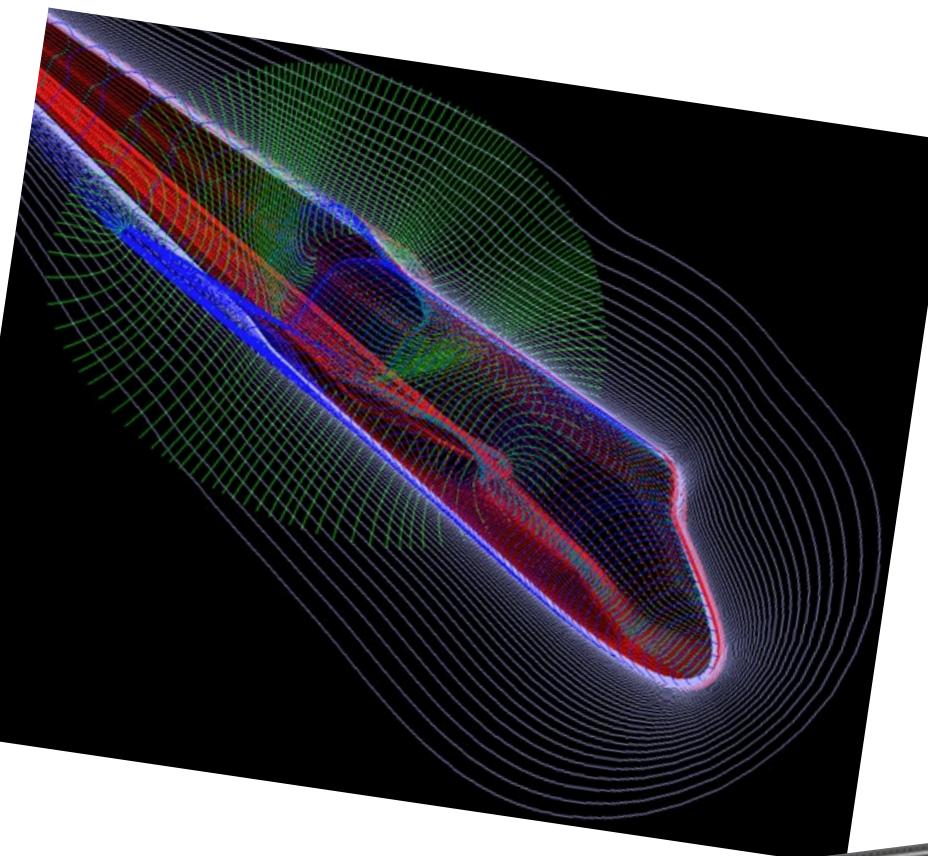
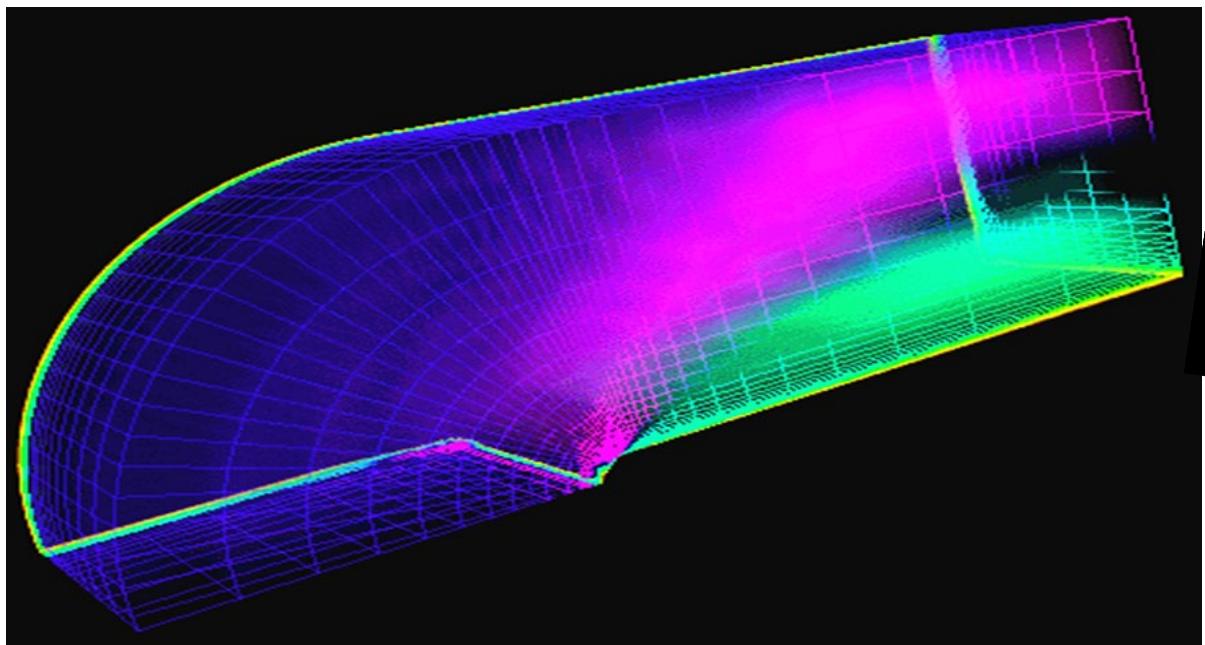
Generally structured or curvilinear grids

- Topology
 - Still regular, but irregular spacing between grid points
 - Positions are non-linearly transformed
- Geometry is explicitly stored
 - `x_coord[NX, NY, NZ]`
 - `y_coord[NX, NY, NZ]`
 - `z_coord[NX, NY, NZ]`
- and
- `data[r]`
- Geometric structure might result in **concave** grids



Data Structures

Examples



Data Structures

Summary

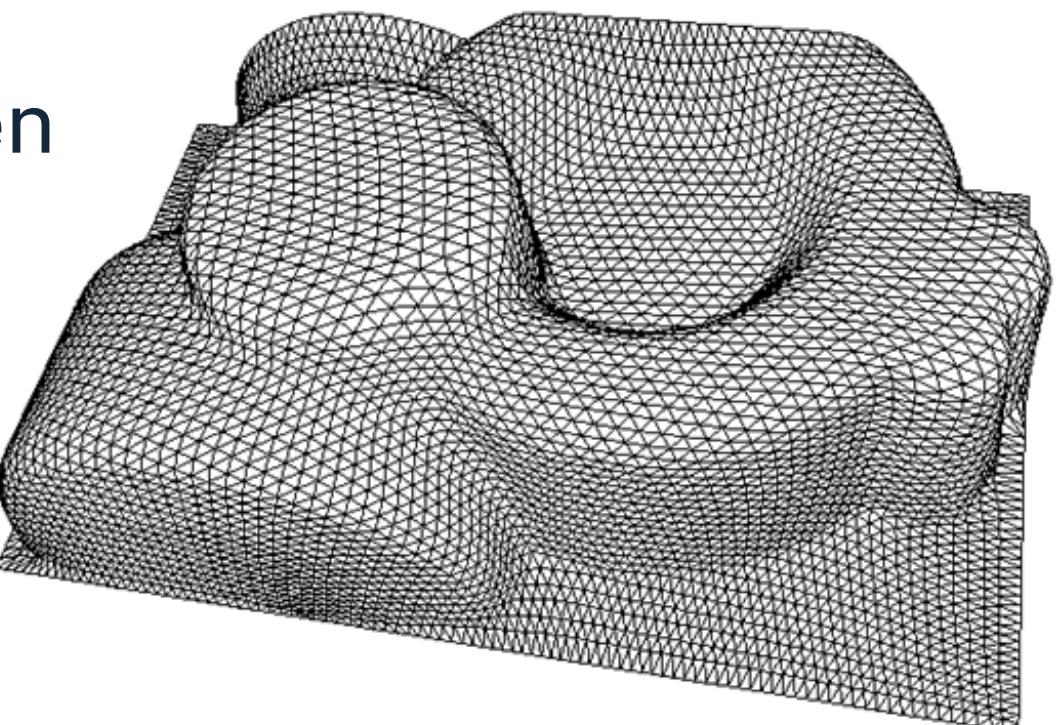
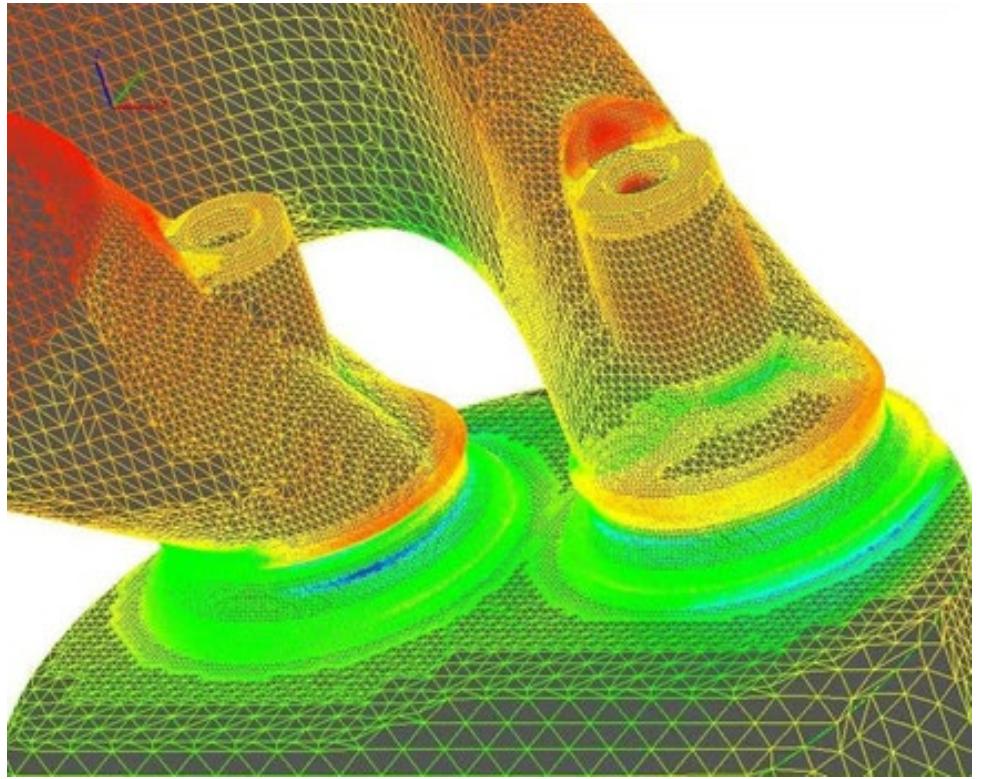
- Structured grids stored in 3D array
- Accessed by indexing
- Topology information implicitly given
 - Direct access to adjacent elements
- Cartesian, uniform, rectilinear grids
 - Necessarily convex
- Visibility ordering of elements implicitly given
 - With respect to viewing direction
- Rigid layout: inflexible for local features
- Curvilinear grids
 - More flexible, but complex sorting of grid elements

3.5.2 Unstructured Grids

Unstructured Grids

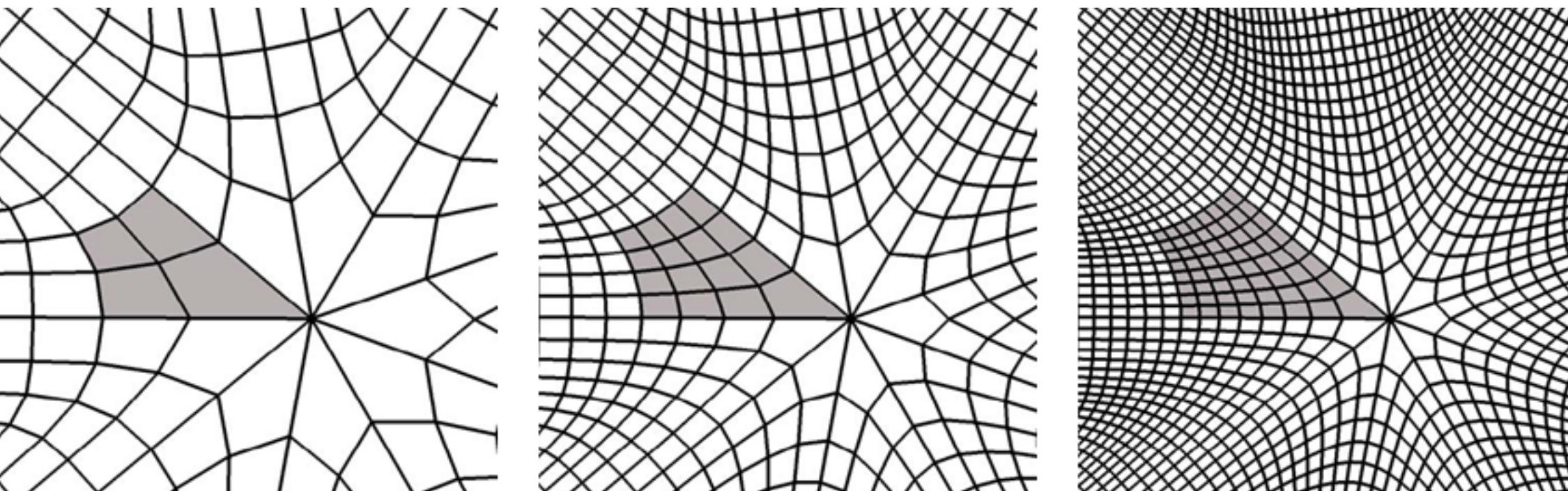
Characteristics of unstructured grids

- Significantly more difficult and complex
 - But much more flexible as there are no constraints
- No implicit topological (connectivity) information given
 - Grid points + connectivity must be explicitly stored
- Dedicated data structures needed
 - Efficient traversal and data retrieval
- Often composed of triangles or tetrahedra
 - Requires triangulation or tetrahedrization
- Less elements are needed to cover the domain



Unstructured Grids

- Dimension
- Number of vertices
- Number of cells
- Vertex list: $(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2), \dots$
- Cell list: $(\text{index}_{v1}, \text{index}_{v2}, \text{index}_{v3}, \text{index}_{v4}), \dots$



Unstructured Grids

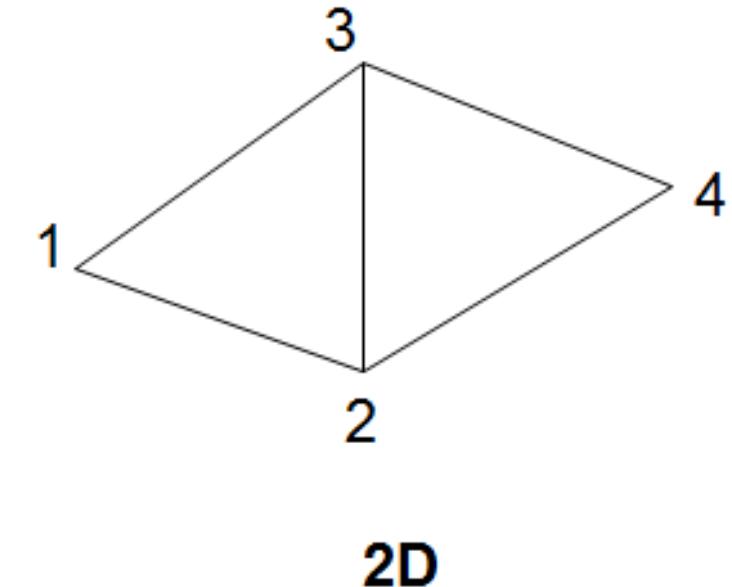
Representation in direct form

x_1, y_1
 x_2, y_2
 x_3, y_3

} face 1

x_2, y_2
 x_3, y_3
 x_4, y_4

} face 2



Struct face {
 float verts [3][2];
 DataType value;
}

Struct face {
 float verts [3][3];
 DataType value;
}

}

- Additionally, store the data values

}

- Problems: storage space, redundancy of edges

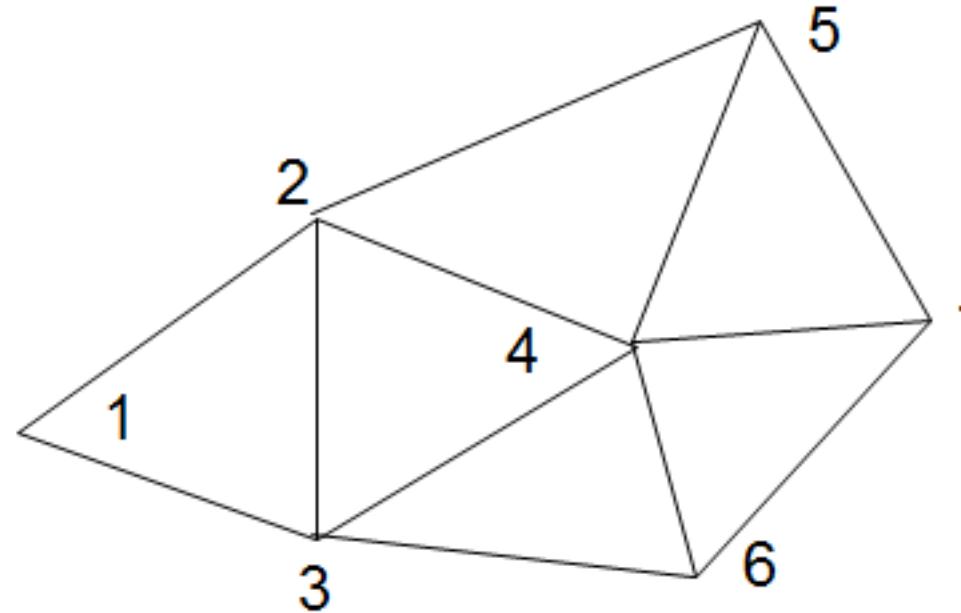
Unstructured Grids

Representation in indirect form ("indexed face set")

Vertex list Face list

x1, y1	1,3,2
x2, y2	3,4,2
x3, y3	3,6,4
....	2,4,5
....

} Uniform ordering
Counter clockwise

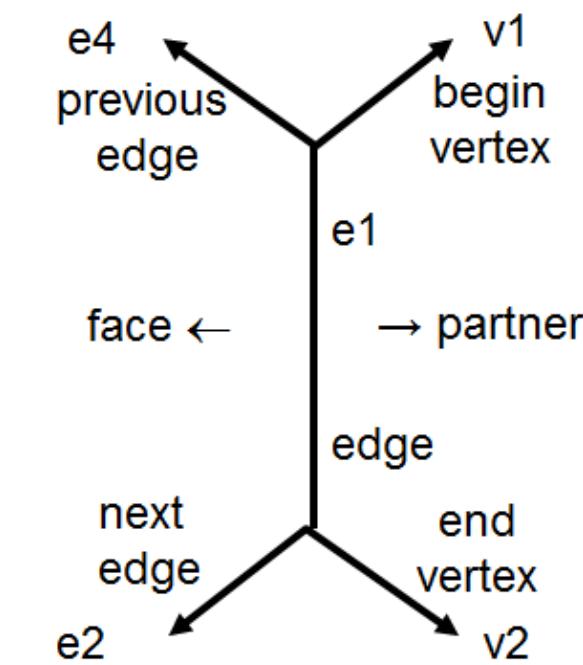


- More efficient than direct approach in terms of memory requirements
- But, still have to do global search to find local information
 - e.g., which faces share an edge?

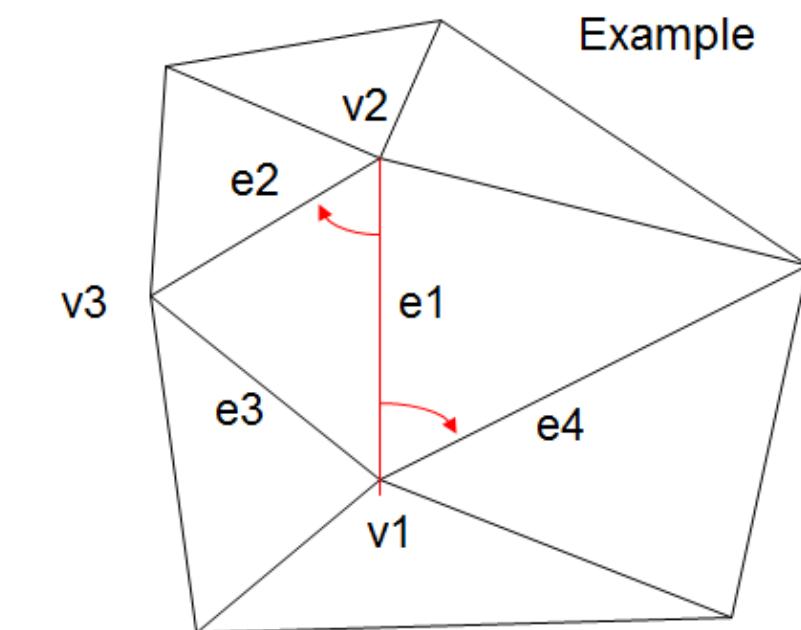
Unstructured Grids

Edge based approach ("winged edge")

- Name from underlying structure



- For every vertex (vertex table)
 - Store a pointer to an arbitrary edge that is adjacent
- For every face (face table)
 - Store a pointer to an edge on its boundary



Unstructured Grids

Edge based approach ("winged edge")

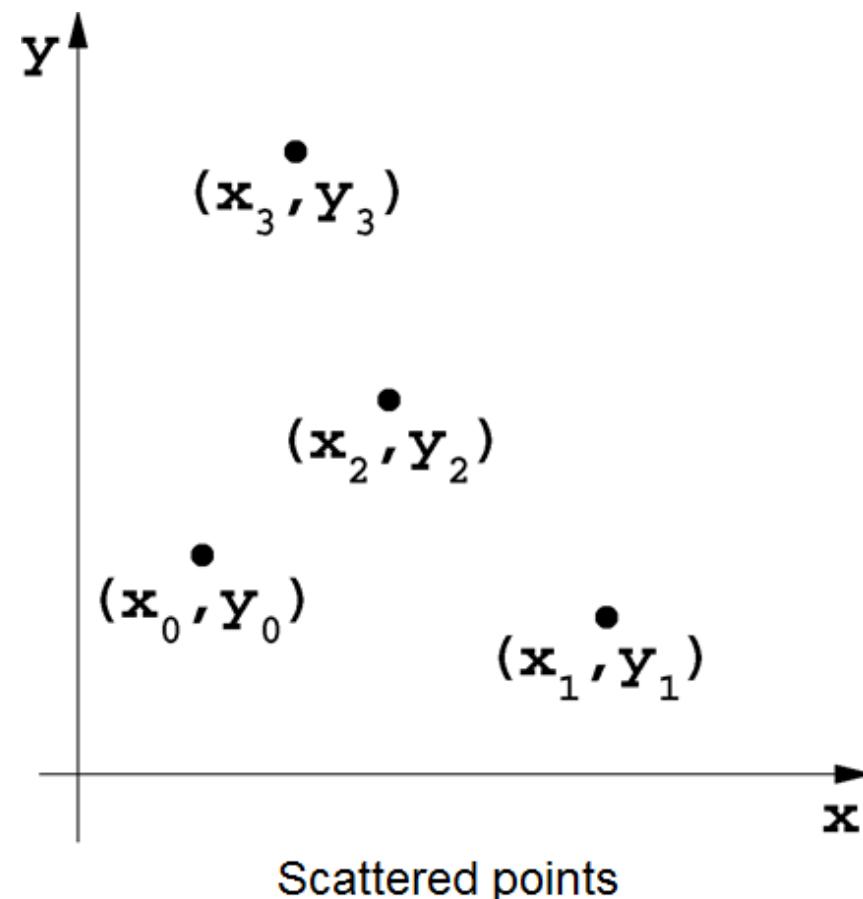
- Answers the following queries
 - Faces sharing an edge
 - Faces sharing a vertex
 - Walk around: faces, vertices (like a fan)
- Implicit assumption
 - Every edge has at most 2 faces which meet at the edge
- Advantages
 - Memory efficient and fast traversal

3.5.3 Scattered Data and Triangulation

Scattered Data

Scattered points

- Problem: create grid from scattered points
- Given
 - Number of vertices
 - Vertex list:
 - $x_0, y_0, z_0;$
 - $x_1, y_1, z_1;$
 - ...



Triangulation

Problem setting

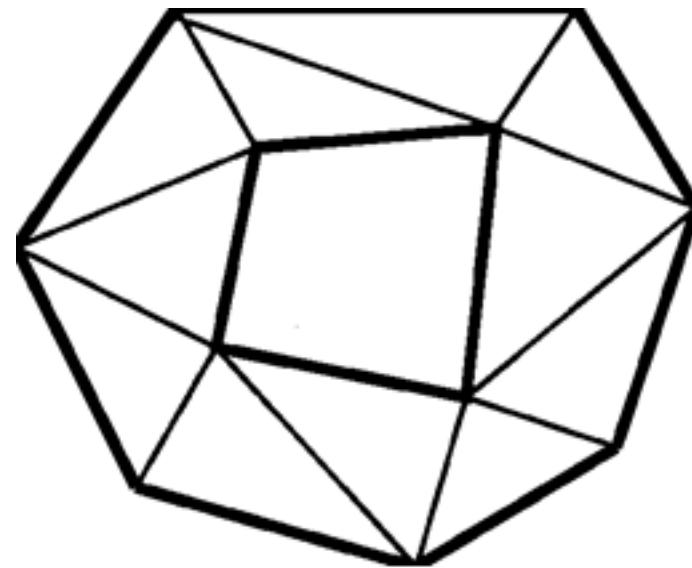
- Given information
 - Data points: $(x_0, y_0), (x_1, y_1), \dots, (x_{N-1}, y_{N-1})$
 - Data values: f_0, f_1, \dots, f_{N-1}
- "Neighborhood information" is required (topology)
 - Task: Find triangular mesh with given scattered data points as vertices
- As measure for "good" triangulation of the data points, consider the "roundness" of triangles
 - radius incircle / radius out-circle
 - Maximal (or minimal) angle

Triangulation

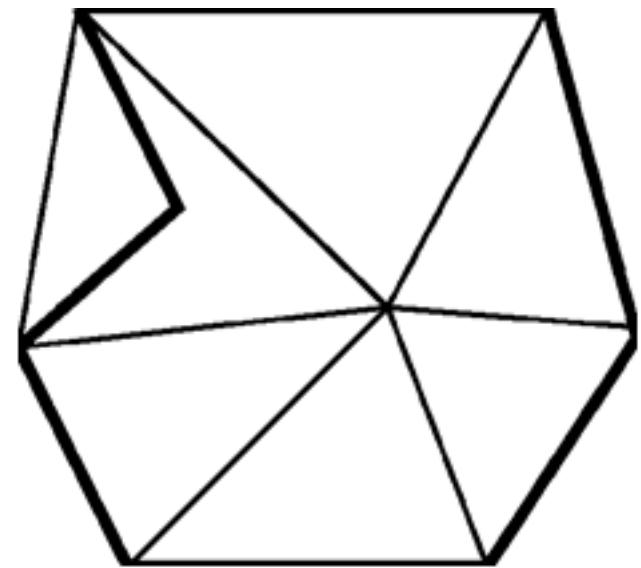
- A triangulation of data points $S = \{s_1, \dots, s_n\}$ in \mathbb{R}^2 consists of:
 - Vertices \rightarrow 0D cells = S
 - Edges \rightarrow 1D cells connecting 2 vertices
 - Faces \rightarrow 2D cells connecting 3 vertices
 - Tetrahedra \rightarrow 3D cells connecting 4 vertices
- Conditions to satisfy
 - \cup faces... = $\text{conv}(S)$,
 - i.e. the union of all faces (including the boundaries, which are the edges and vertices) is the convex hull of all vertices.
- Intersection of two triangles is
 - either empty
 - or a common vertex / edge / face

Triangulation

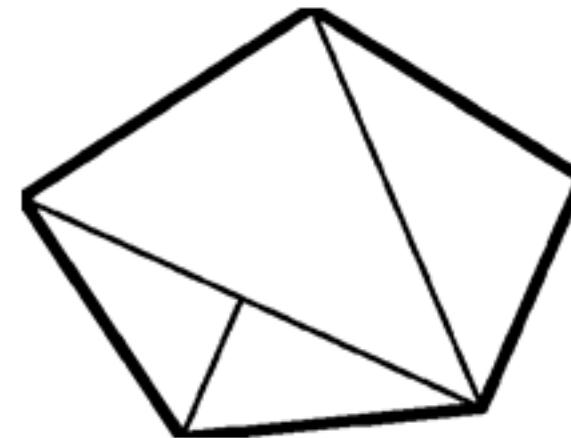
Non-valid triangulations



Hole



Faces overlap

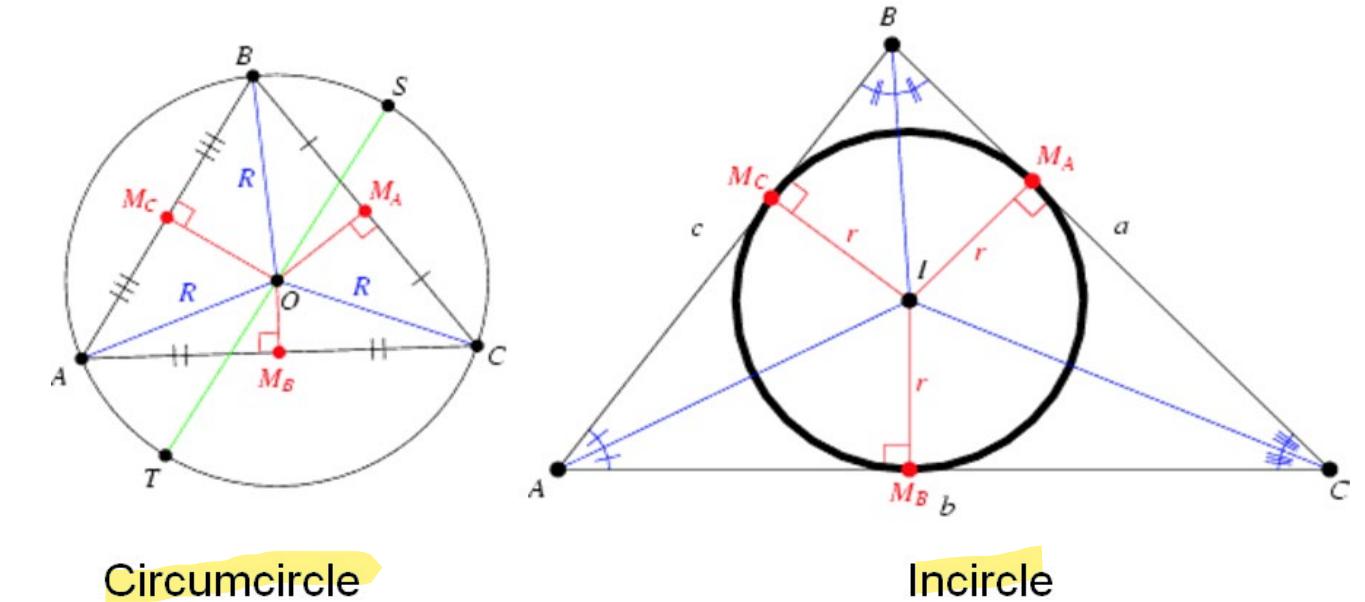


T-Vertex

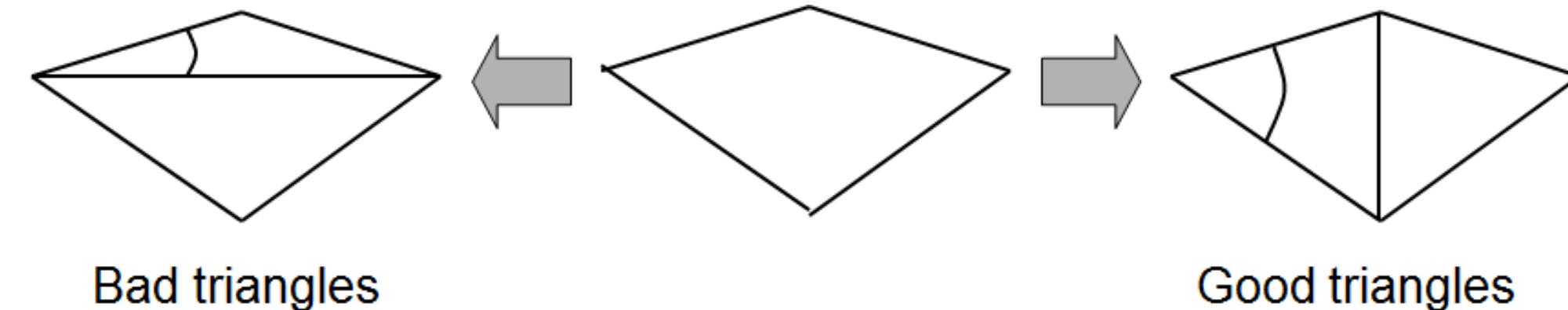
Keine Lücken, keine Überlappungen, keine Verwendung von gleichen Kanten

Triangulation

How to get "good" triangulations?



- For one set of points, there are different triangulations
 - Good ones and less optimal ones
- Good triangulations avoid long, thin triangles

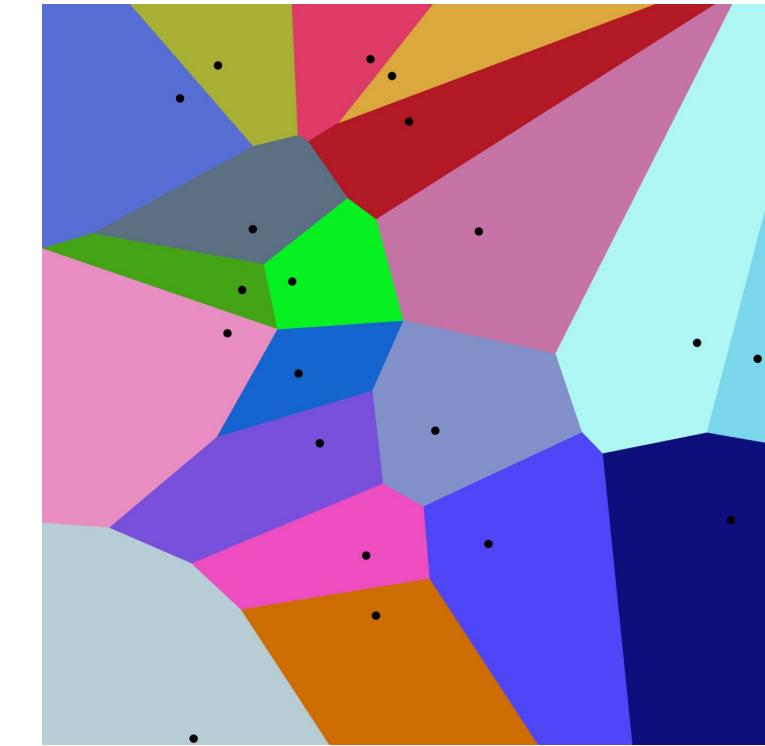
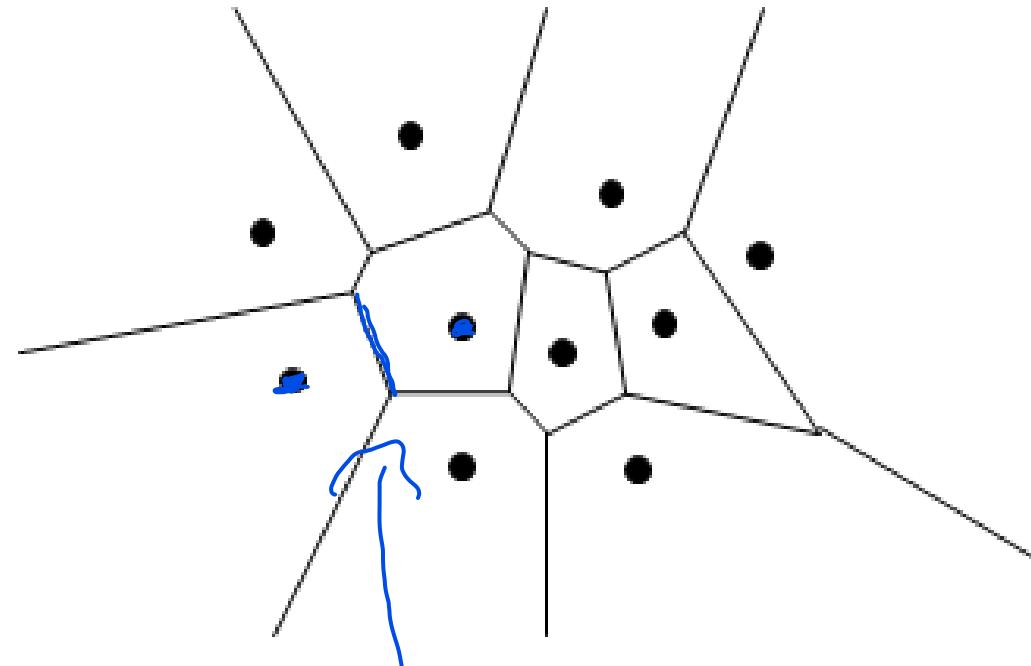


- Measure of quality: aspect ratio of triangles $\rho = \frac{r_{\text{incircle}}}{R_{\text{circumcircle}}} \rightarrow \max$

Triangulation

Voronoi diagram

- Around each sample point construct a region that is closer to that sample than to every other sample



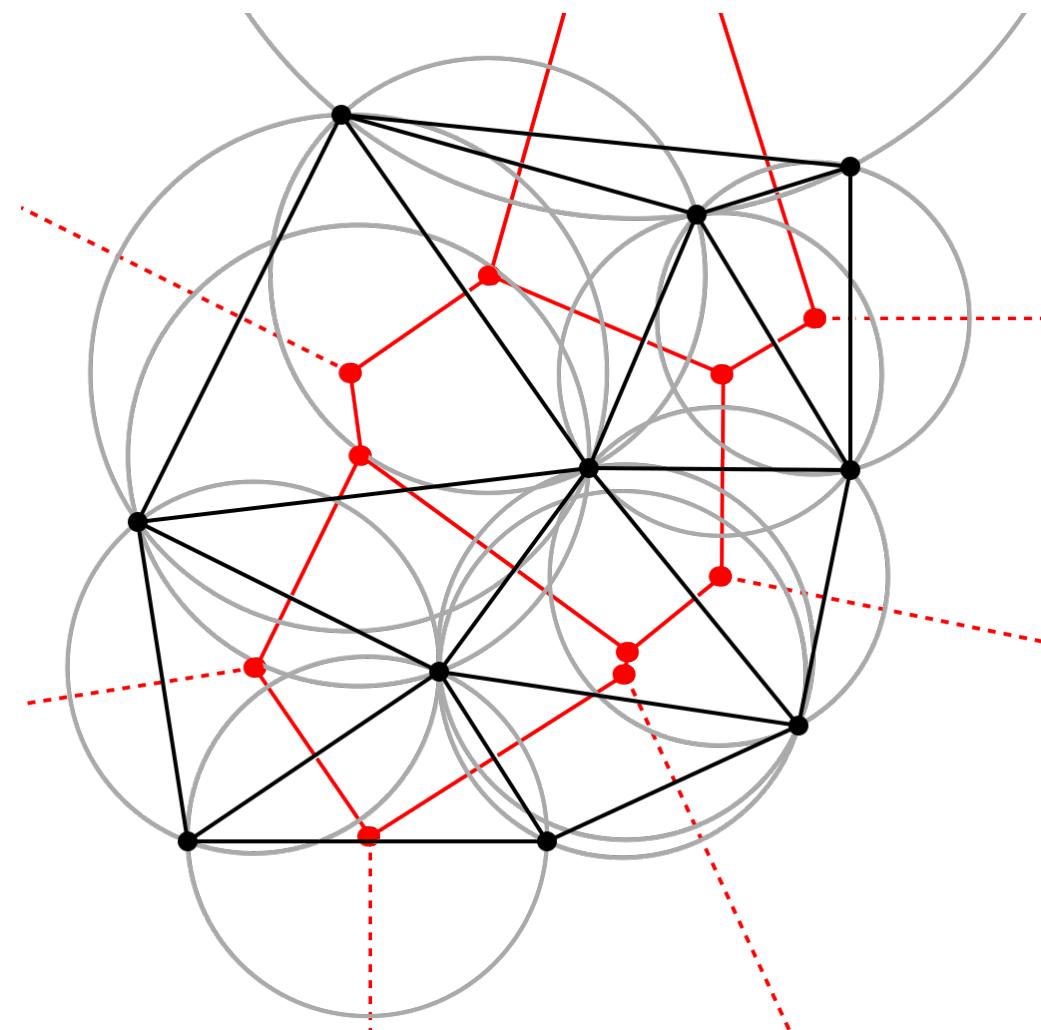
Alle Punkte auf einer Kante sind zu Sample Punkten gleich weit entfernt

Source: https://en.wikipedia.org/wiki/Voronoi_diagram

Triangulation

Delaunay triangulation

- The vertices of the Voronoi diagram as circumcenters of triangles



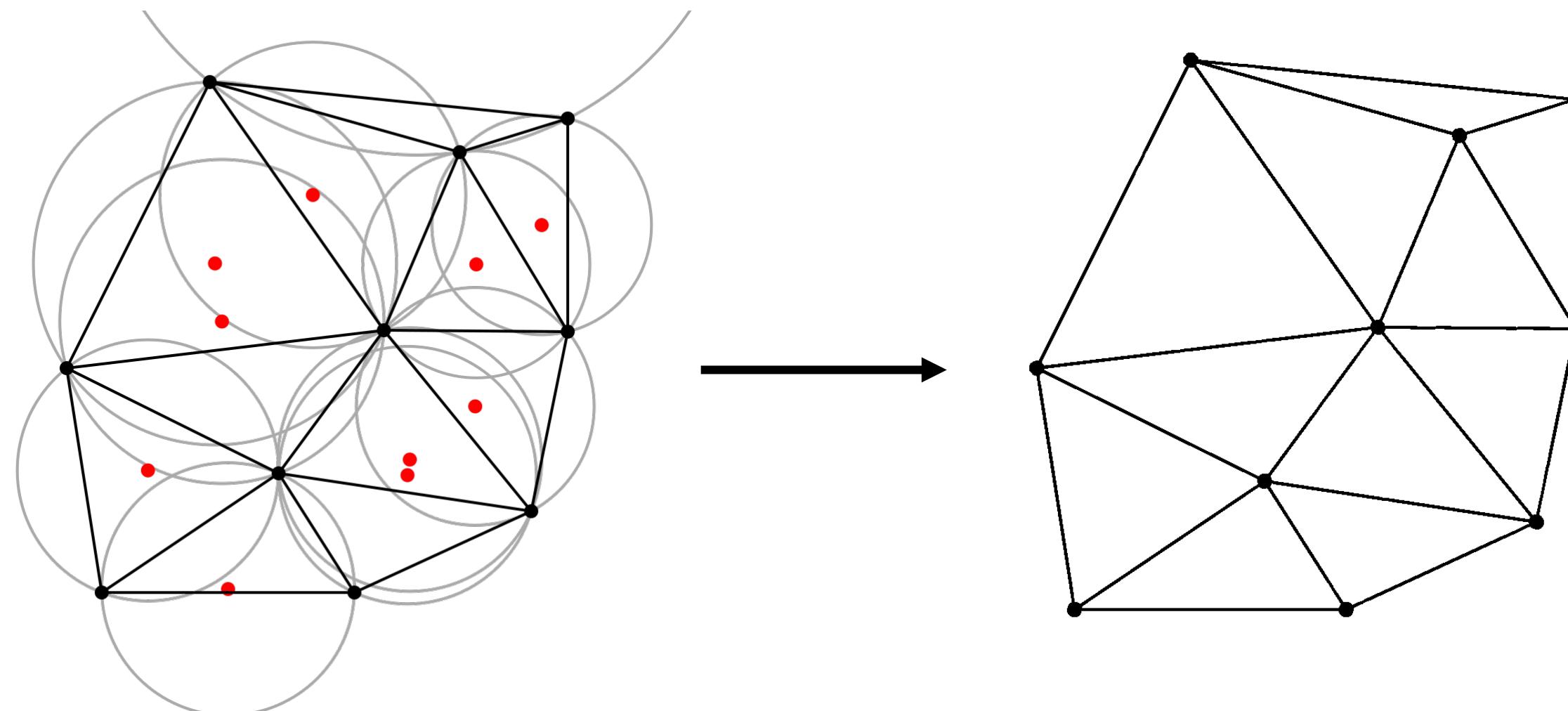
Source: https://en.wikipedia.org/wiki/Delaunay_triangulation

Prof. Dr. Matthias Teßmann

Triangulation

Delaunay triangulation

- The vertices of the Voronoi diagram as circumcenters of triangles



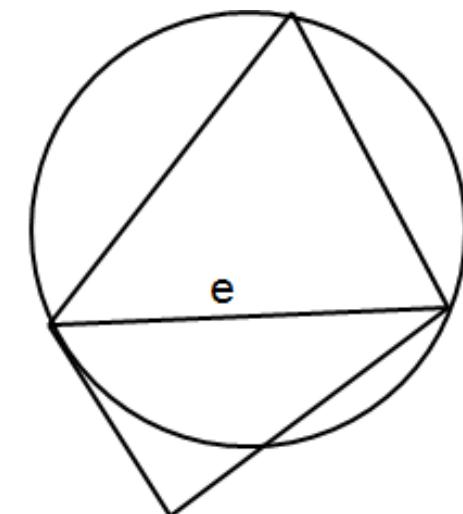
Source: https://en.wikipedia.org/wiki/Delaunay_triangulation

Prof. Dr. Matthias Teßmann

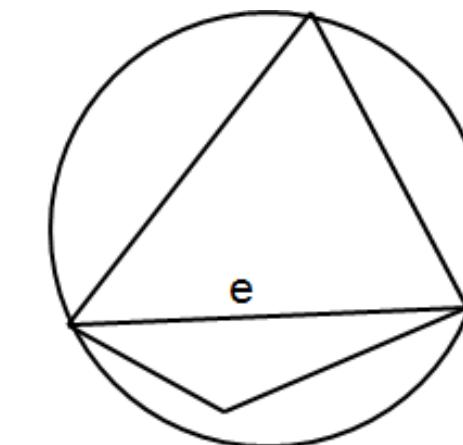
Schnittpunkte der Kreise verbinden

Triangulation

- Delaunay triangulation is the geometric dual of the Voronoi diagram
 - Maximizes the smallest angle and the aspect ratio of triangles
 - Triangles fulfill the "local Delaunay property"
 - For each edge, the perimeter of the adjacent triangle does not contain the 'other' vertex



Local Delaunay property



No local Delaunay property

Punkt liegt im
Umkreis des
anderen Dreiecks

Triangulation

Two-step algorithm: initial triangulation and edge flip

- Initial triangulation

```
- Sort points from left to right  
  
- Construct initial triangle using first three vertices  
  
- For (each point pi in vertex list) {  
    Use last inserted point pi as starting point  
    Take convex polygons of triangulation  
    Find edge of minimal distance  
    Triangulate edge + pi  
}
```

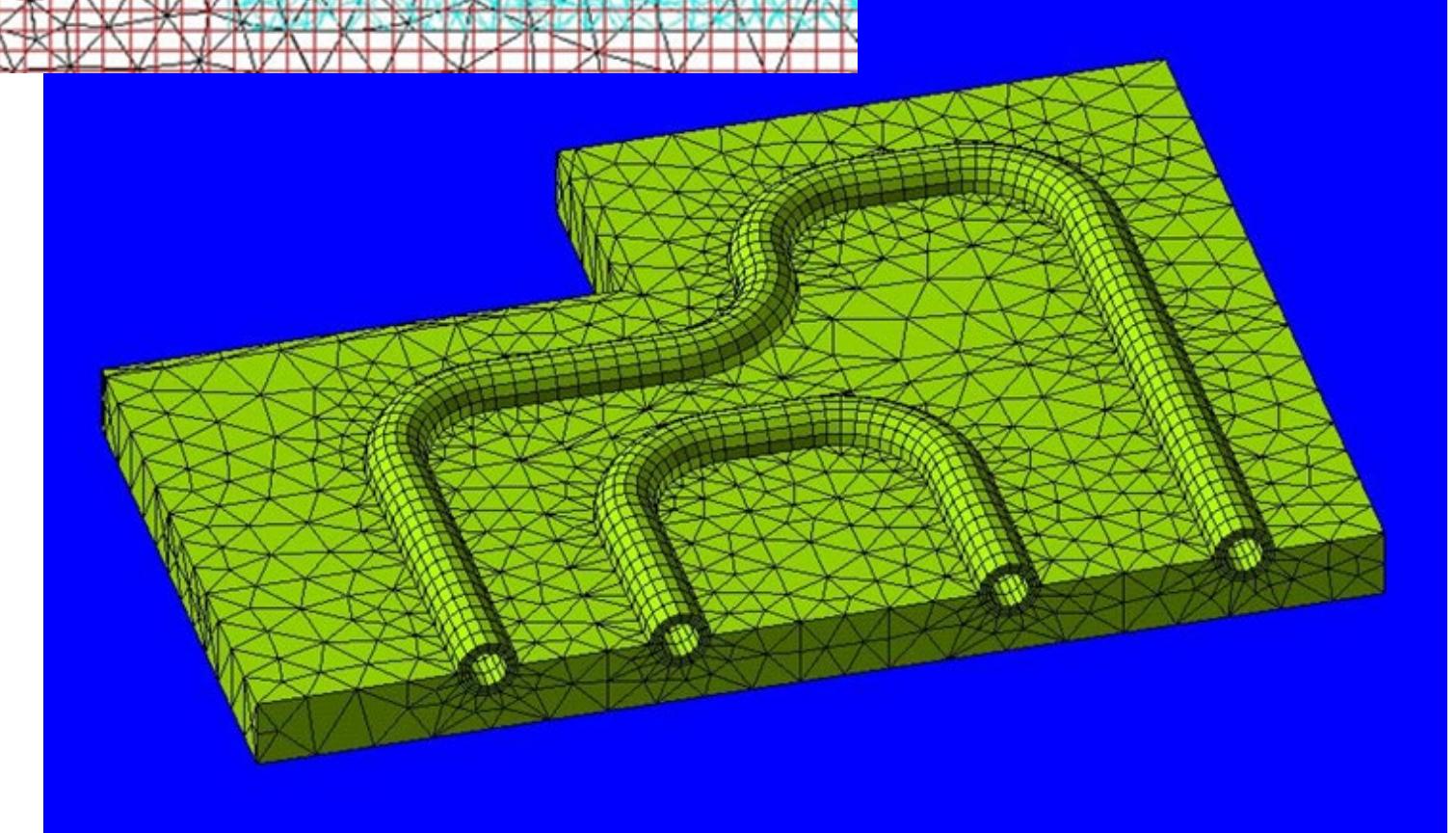
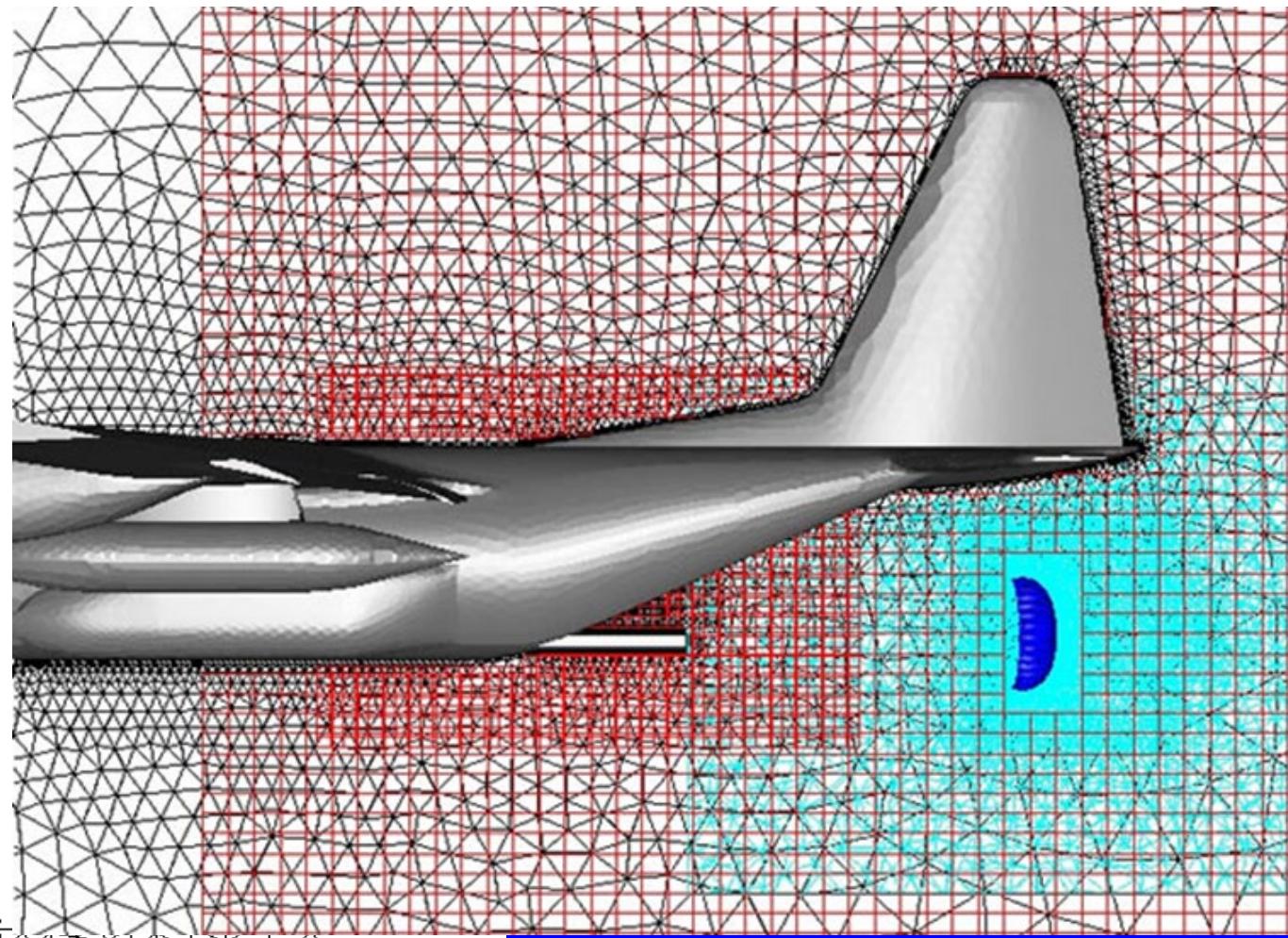
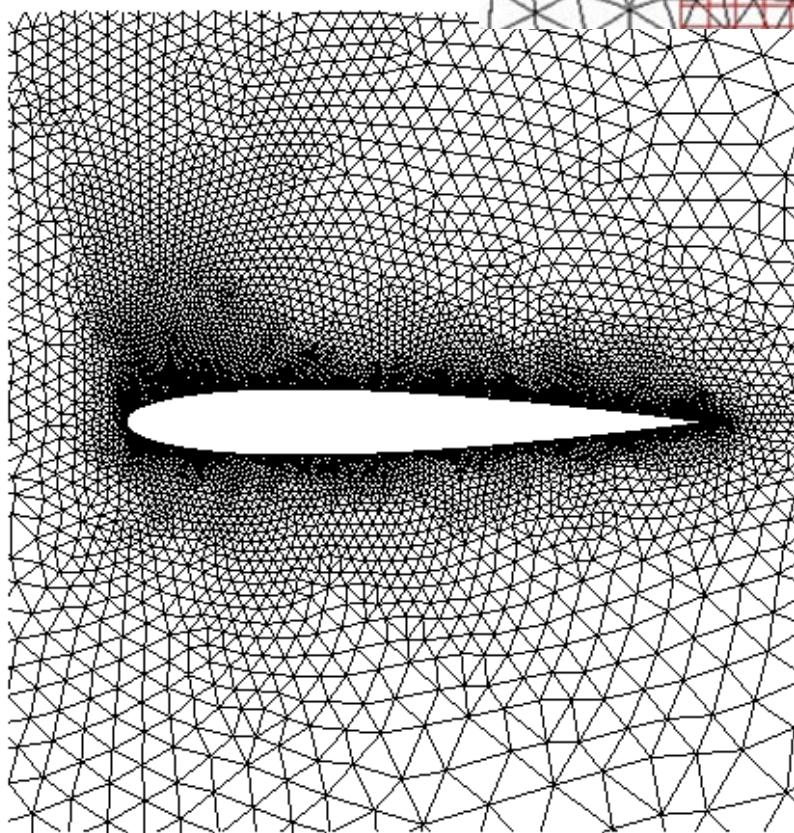
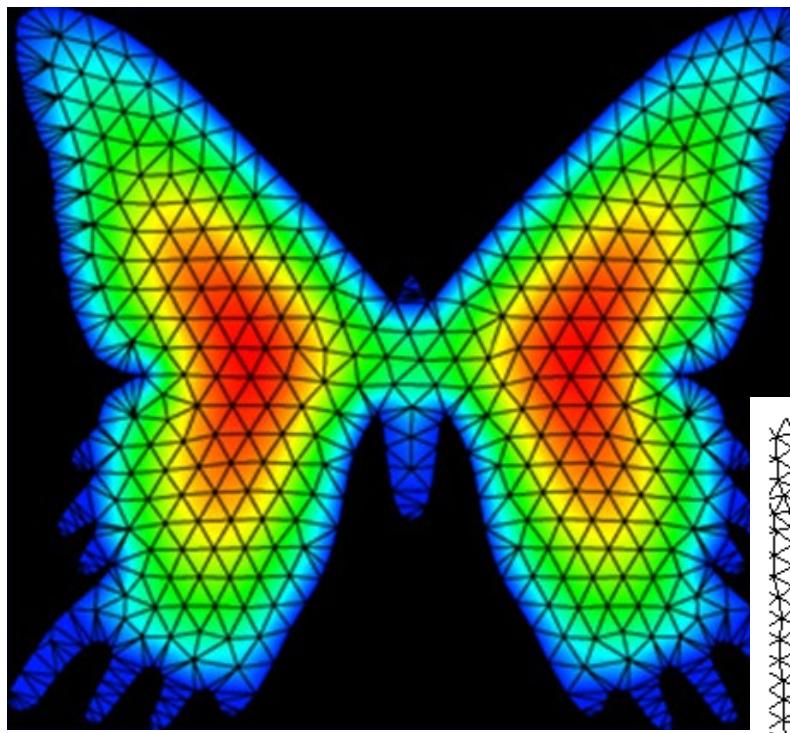
- Typically results in suboptimal triangulation

Triangulation

Two-step algorithm: initial triangulation and edge flip

- Edge flip
 - Find initial (valid) triangulation
 - Find all edges where local Delaunay property does not hold
 - Mark these edges + push them onto stack
 - While (stack not empty) {
 - Pop edge from stack
 - if (edge does no satisfy Delaunay property) {
 - Flip edge
 - Flip all neighboring edges that do not satisfy the local Delaunay property any more
 - }
 - }
- Note: this algorithm terminates

Triangulation Examples



3.5.4 Differentiation on Grids

Differentiation on Grids

Problem

- Typically, discrete measured data are given
- We actually want to **visualize continuous data**
- Mathematical description
 - Scalar data: 1D-Scalar: $f : \mathbb{R} \mapsto \mathbb{R}$
 - 2D-Scalar: $g : \mathbb{R}^2 \mapsto \mathbb{R}$
 - 3D-Scalar: $h : \mathbb{R}^3 \mapsto \mathbb{R}$
- Vector data: 2D/3D Vector: $F : \mathbb{R}^2 \mapsto \mathbb{R}^2$ or $G : \mathbb{R}^3 \mapsto \mathbb{R}^3$
- Tensor: $I : \mathbb{R}^3 \mapsto Mat(3, 3) = \mathbb{R}^{3x3}$
- **Points of special interest**
 - Locations where quantities *change rapidly (high variation)*
 - Changes are measured by **derivatives (differentials)**

Differentiation on Grids

- First approach
 - Approximate / interpolate (locally) by differentiable function and differentiate this function
- Second approach
 - Replace differential by "finite differences"

Ableitung

$$f'(x) = \frac{df}{dx} \rightarrow \frac{\Delta f}{\Delta x}$$

Differentiation on Grids

1D uniform grids with grid size $h = \Delta x$

- Forward difference

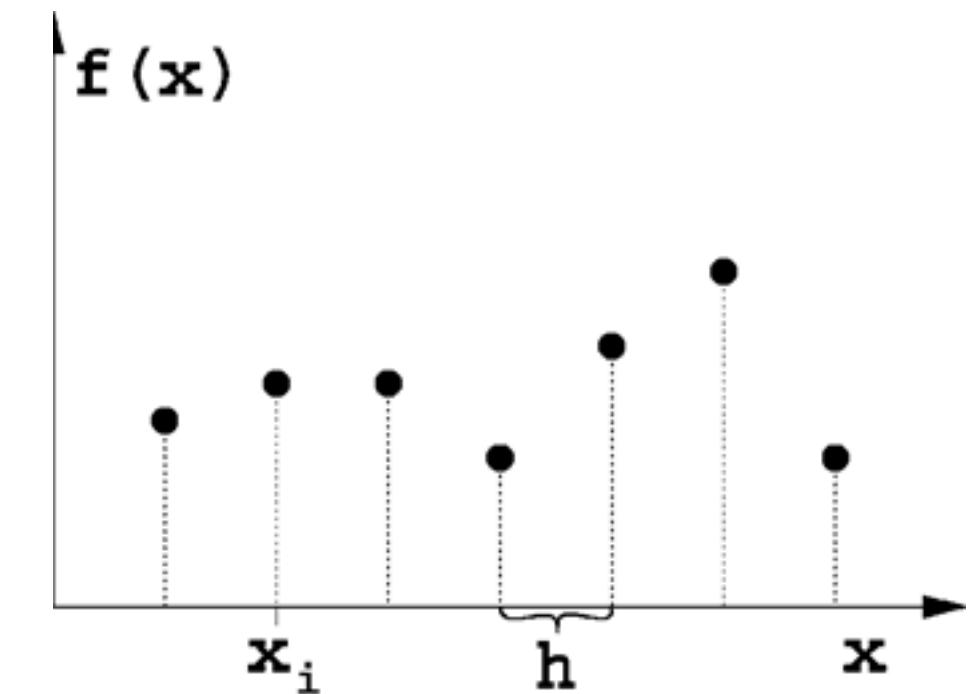
$$f'(x) = \frac{f(x_{i+1}) - f(x_i)}{h}$$

- Backward difference

$$f'(x) = \frac{f(x_i) - f(x_{i-1})}{h}$$

- Central difference

$$f'(x) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$



- The error is $O(h)$ for forward/backward difference and $O(h^2)$ for central difference

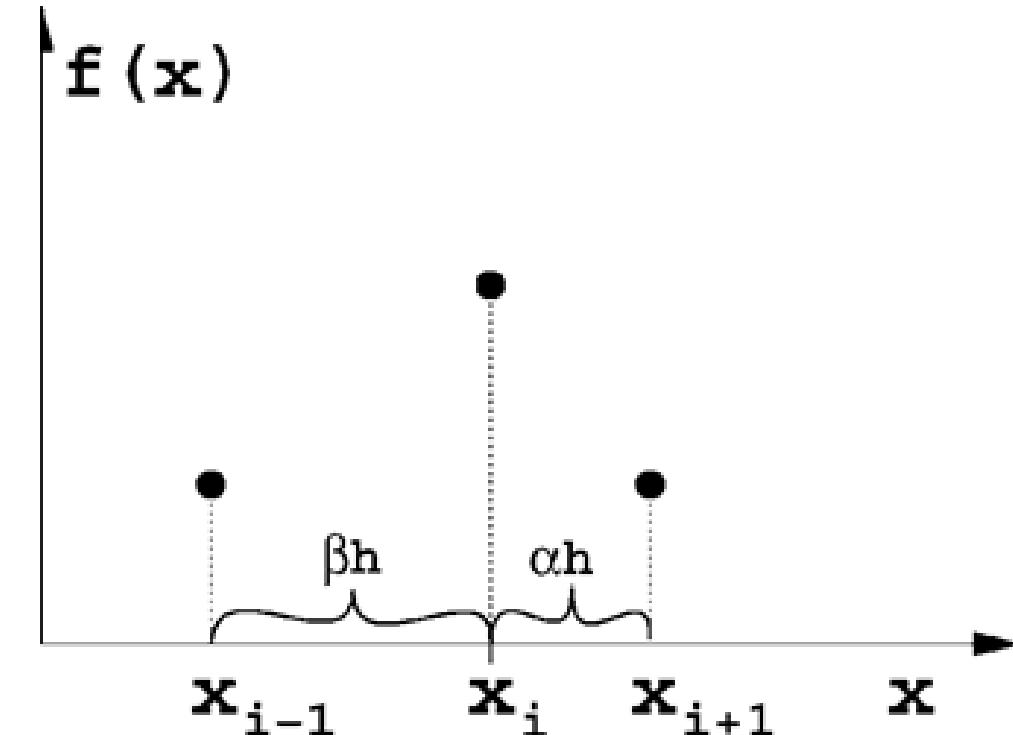
Differentiation on Grids

1D non-uniform grid

- Forward or backward difference is the same as for uniform grids
 - Note: $h = \Delta x$ is different in each direction, as spacing is non-uniform

$$x_{i+1} - x_i = \alpha h$$

$$x_i - x_{i-1} = \beta h$$



Differentiation on Grids

1D non-uniform grid

- Central difference needs to consider the different spacing
 - From Taylor

$$\begin{aligned} f(x_{i+1}) &= f(x_i) + \alpha h f'(x_i) + \frac{(\alpha h)^2}{2} f''(x_i) + \dots \\ f(x_{i-1}) &= f(x_i) + \beta h f'(x_i) + \frac{(\beta h)^2}{2} f''(x_i) + \dots \end{aligned}$$
$$\Rightarrow \frac{1}{\alpha^2} (f(x_{i+1}) - f(x_i)) - \frac{1}{\beta^2} (f(x_{i-1}) - f(x_i)) = \frac{h}{\alpha} f'(x_i) + \frac{h}{\beta} f(x_i) + O(h^3)$$

- Division by α^2 and β^2 eliminates the parts $\frac{h^2}{2} f''(x_i)$

Differentiation on Grids

1D non-uniform grid

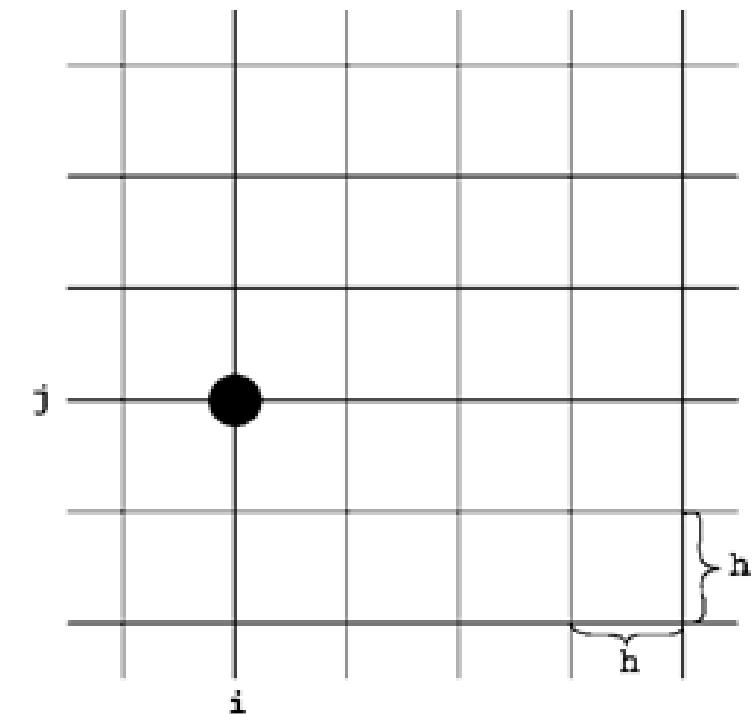
- Then, the final approximation of the derivative is

$$f'(x_i) = \frac{1}{h(\alpha + \beta)} \left(\frac{\beta}{\alpha} f(x_{i+1}) - \frac{\alpha}{\beta} f(x_{i-1}) + \frac{\alpha^2 - \beta^2}{\alpha\beta} f(x_i) \right)$$

Differentiation on Grids

2D / 3D uniform or rectangular grids

- Partial derivatives $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
Partielle Ableitung
- Same as in univariate case along each coordinate axis
- Example
 - Gradient on a 3D uniform grid with size h



$$\text{grad } f = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{f_{i+1,j,k} - f_{i-1,j,k}}{2h} \\ \frac{f_{i,j+1,k} - f_{i,j-1,k}}{2h} \\ \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2h} \end{pmatrix}$$

Differentiation on Grids

Unstructured grids

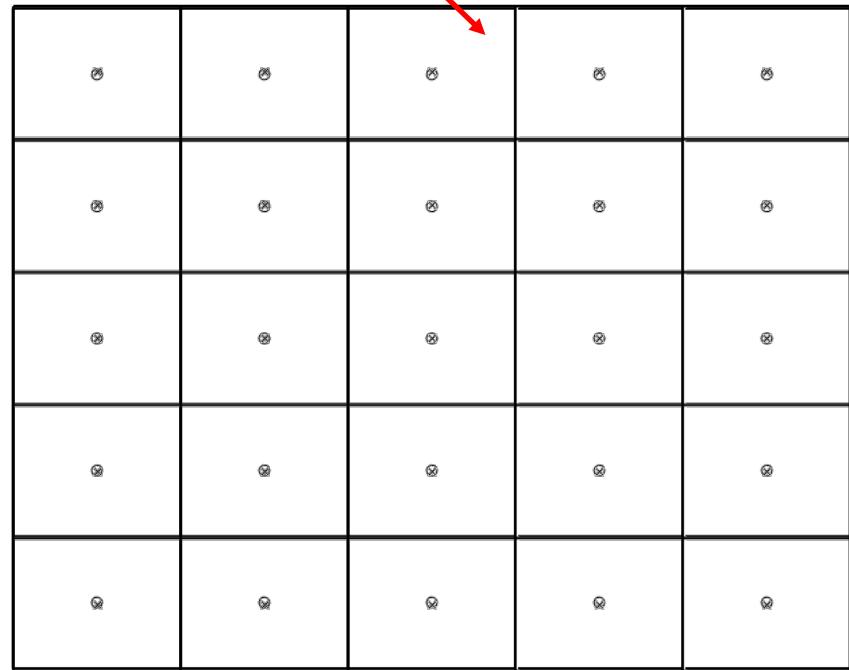
- Dilemma
 - There is no generalization of the difference quotient!
- Solution (1)
 - Interpolation / approximation with function to differentiate
 - Easiest case: linear interpolation in each cell leads to one gradient per cell
- Solution (2)
 - Resampling into structured grid
 - Interpolation introduces additional error

3.6 Interpolation

Interpolation

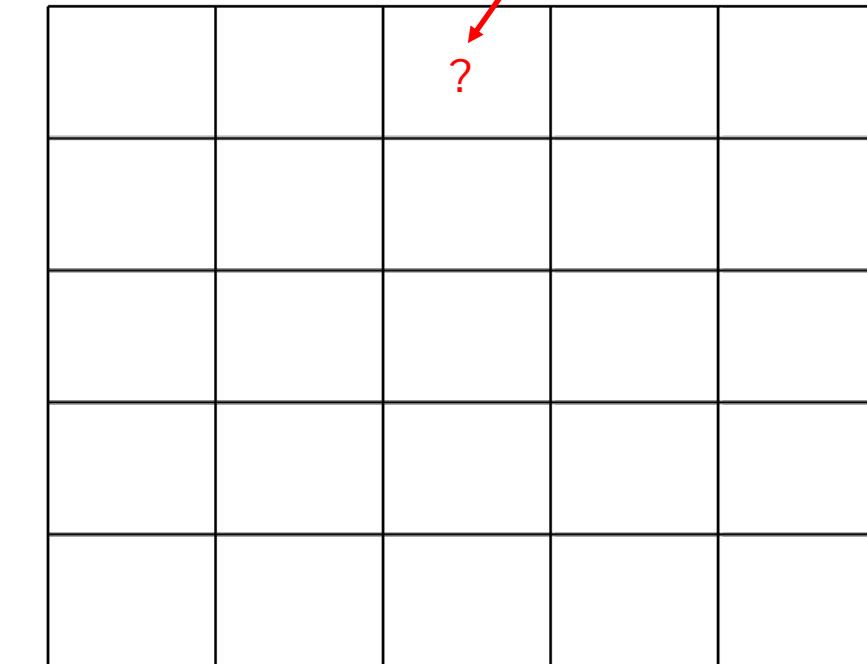
What happens when an image is rotated?

Original sample points move



Input Image

Pixel / data grid stays constant, voxel grid changes.
What should be the center (sampling) value here?

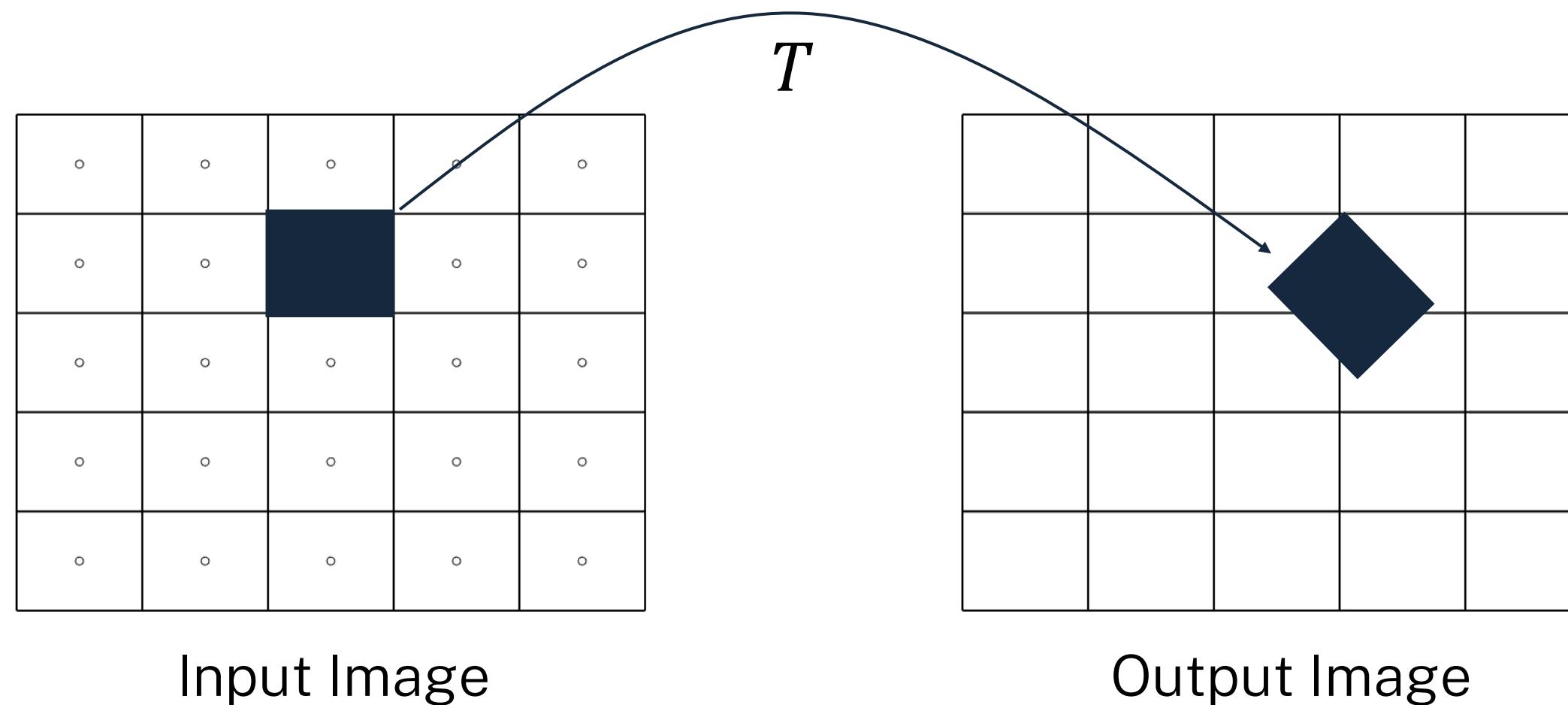


Output Image

Interpolation

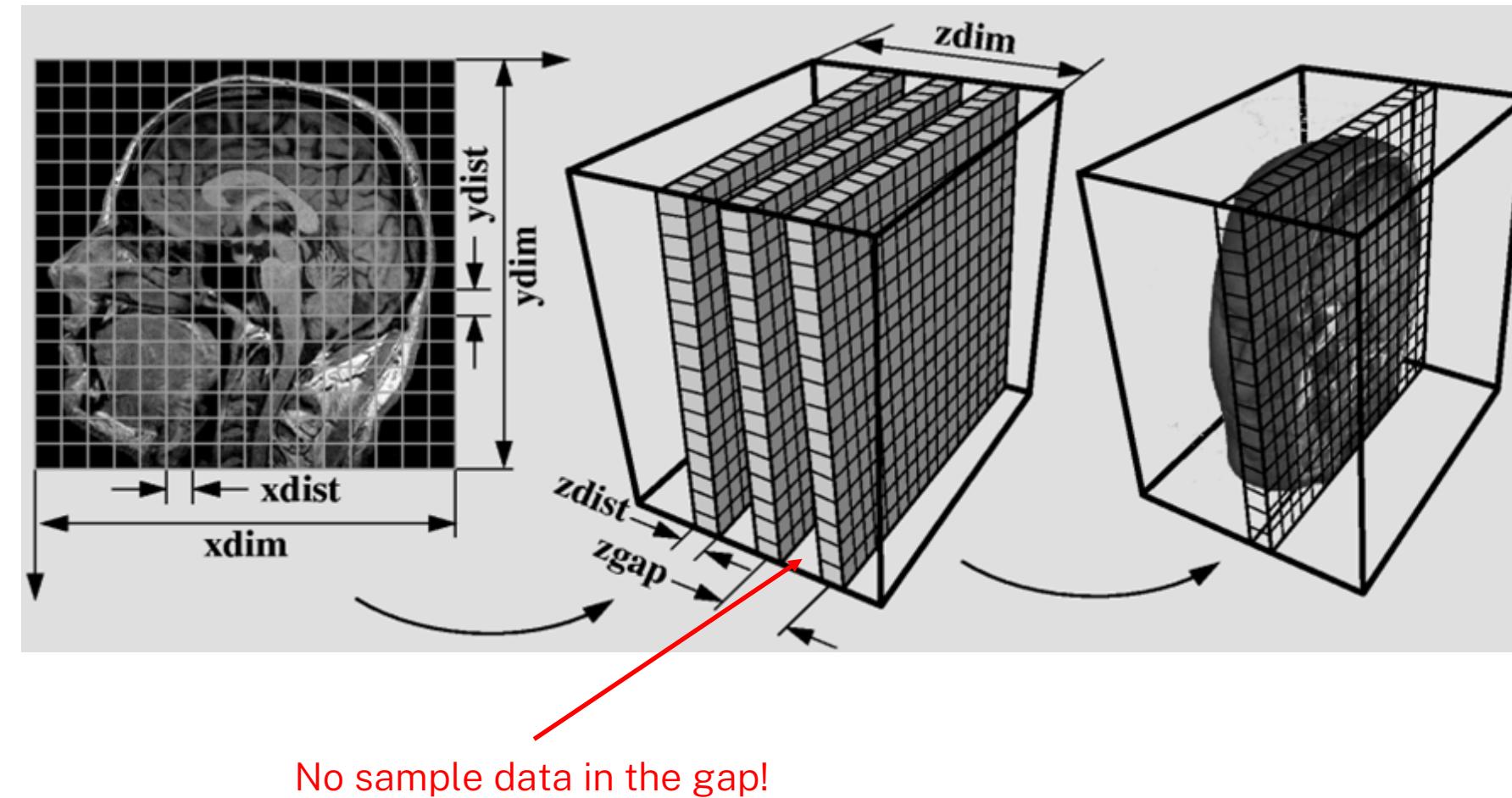
Resulting cells do not match the required sampling grid

- Resampling is required
 - Determine cells (and data values) on the output grid



Interpolation

- When constructing orthogonal or arbitrary orientations from a dataset there might be no data at all
- This is also true when, e.g. enlarging an image or at the corners after rotation

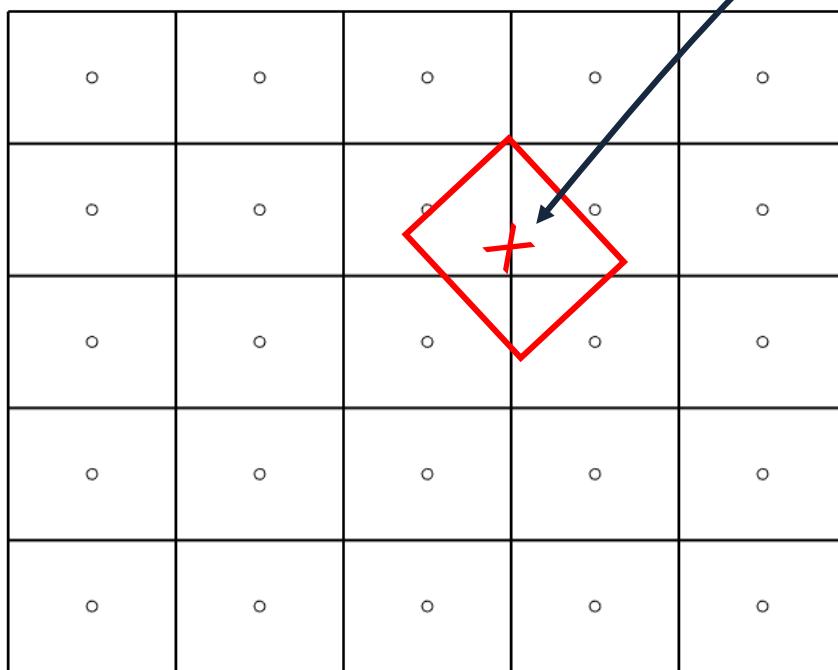


Interpolation

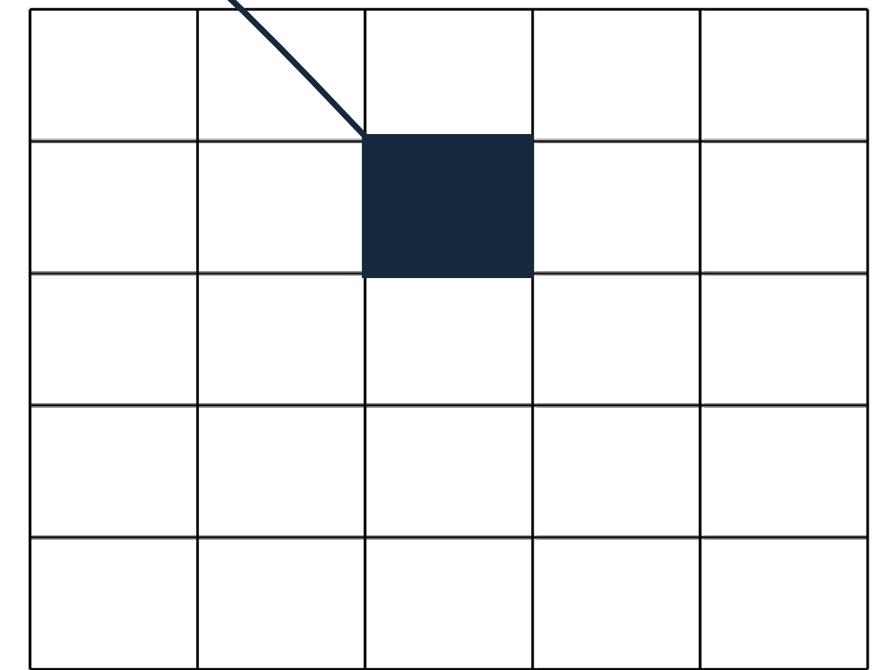
Resampling

- Traverse **output grid** and calculate cell location in original image by transforming with T^{-1}
 - Calculate cell center value
 - Apply as data value

How is the output value calculated?



$$T^{-1}$$

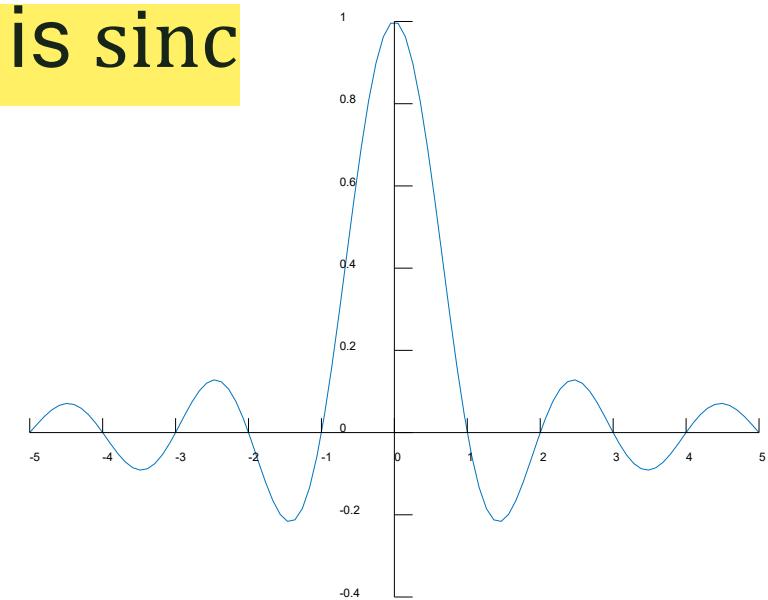


Interpolation

- Problem
 - (Image) function known only at discrete sampling points
 - Function values required in-between
 - Analytical description unknown / impossible to construct
- Interpolation rekonstruieren der Funktion aus diskreten sampling points
 - Estimate values of unknown function in-between given sampling points
 - The ideal interpolation function for a continuous signal is sinc

$$\text{sinc } x = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-j\omega x} d\omega = \begin{cases} \frac{\sin \pi x}{\pi x} & , x \neq 0 \\ 1 & , x = 0 \end{cases}$$

- Problem: sinc requires infinitely many samples for perfect interpolation



Interpolation

Classification

- Local interpolation methods (cell-wise)
 - Use information stored in the neighbor nodes (vertices)!

Nachbar, Ecken, Gitter, ... betrachten

- Global interpolation methods
 - Use information of all vertices or a *broader* neighborhood

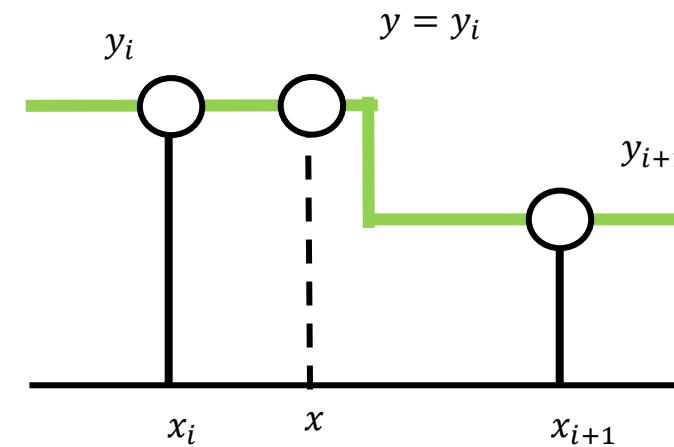
3.6.1 Local Interpolation

Local Interpolation

Simple approximations

Abstand zum nahestehen Nachbar

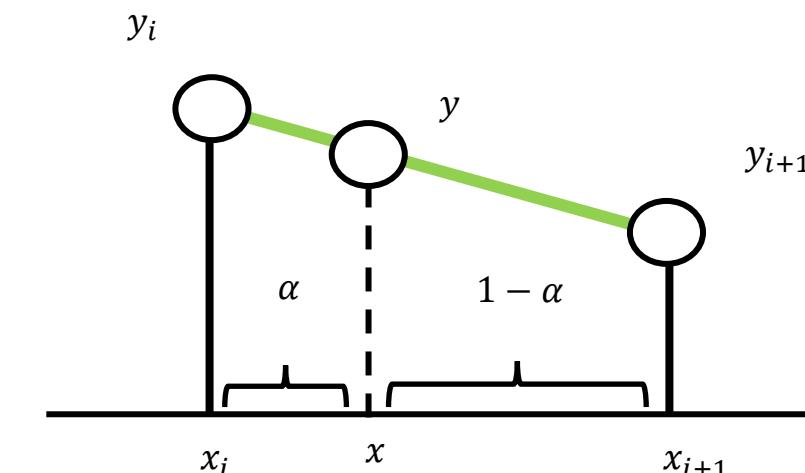
- **Nearest-neighbour**
 - Very coarse approximation
 - Low quality interpolant
 - Fast calculation



$$f(x) = y = \begin{cases} y_i, & \frac{x - x_i}{x_{i+1} - x_i} < \frac{1}{2} \\ y_{i+1}, & \frac{x - x_i}{x_{i+1} - x_i} \geq \frac{1}{2} \end{cases}$$

- **Linear interpolation**

- Assume linear relationship between samples
- Quality ok, esp. on dense sampling grids
- Fast calculation
- Also known as linear blending
 - "Default" on graphics hardware
 - Shaders allow more sophisticated interpolation kernels



$$f(x) = y = \alpha y_{i+1} + (1 - \alpha) y_i$$

$$\text{with } \alpha = \frac{x - x_i}{x_{i+1} - x_i}$$

$\alpha :=$ Abstand

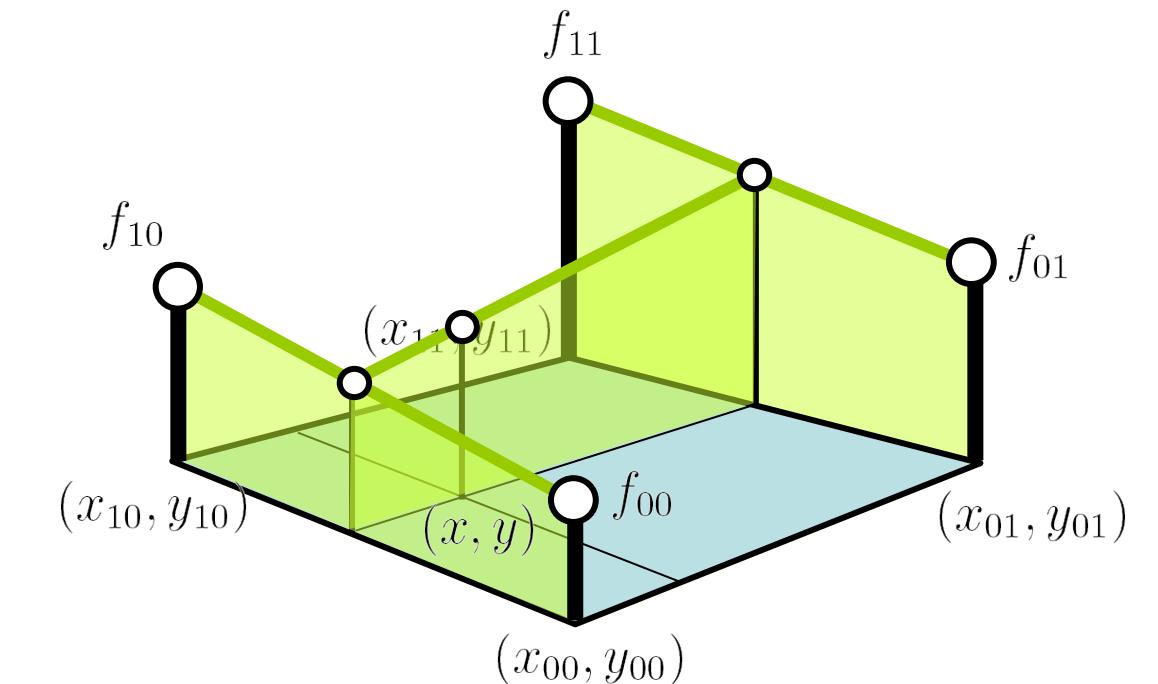
Local Interpolation

Bi-linear interpolation (2D)

Zwei mal lineare Interpolation von x und y

- Most common interpolation function on 2D image grids
- Naive implementation requires 8 multiplications, optimized variant requires 3
- Fast data access with pointer arithmetic

```
dp = &data[y * NX + x];
f00 = dp[0];
f10 = dp[1];
dp += NX;
f01 = dp[0];
f11 = dp[1];
```



$$f_0 = \alpha f_{10} + (1 - \alpha) f_{00} \quad \alpha = \frac{x - x_{00}}{x_{10} - x_{00}}$$

$$f_1 = \alpha f_{11} + (1 - \alpha) f_{01}$$

$$f(x, y) = \beta f_1 + (1 - \beta) f_0 \quad \beta = \frac{y - y_{00}}{y_{10} - y_{00}}$$

- Note: bi-linear interpolation is not a linear operation anymore!

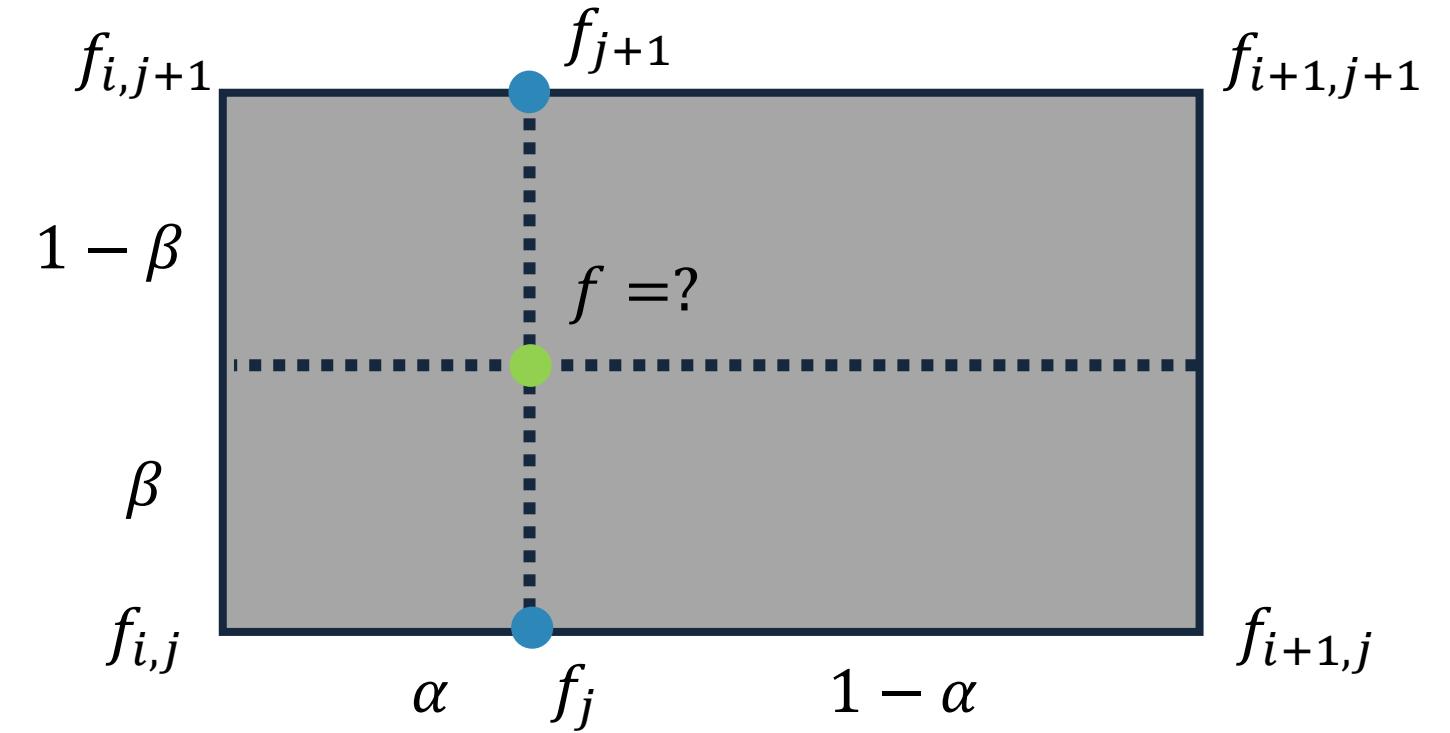
Local Interpolation

Bilinear interpolation on a rectangle

$$f_j = (1 - \alpha)f_{i,j} + \alpha f_{i+1,j}$$

$$f_{j+1} = (1 - \alpha)f_{i,j+1} + \alpha f_{i+1,j+1}$$

$$f = (1 - \beta)f_j + \beta f_{j+1}$$



$$f(x, y) = (1 - \beta)[(1 - \alpha)f_{i,j} + \alpha f_{i+1,j}] + \beta[(1 - \alpha)f_{i,j+1} + \alpha f_{i+1,j+1}]$$

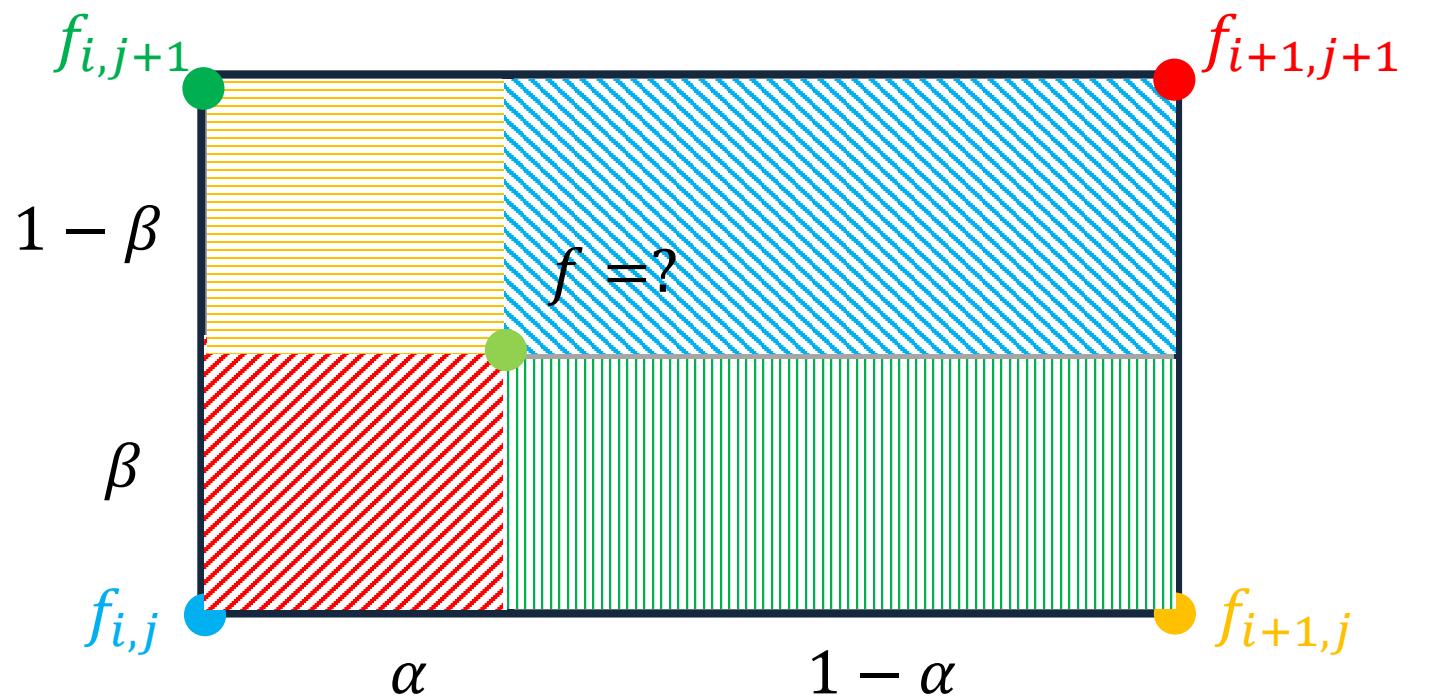
$$\alpha = \frac{x - x_i}{x_{i+1} - x_i}, \quad \beta = \frac{y - y_i}{y_{i+1} - y_i}, \quad \alpha, \beta \in [0, 1]$$

Local Interpolation

Bilinear interpolation on a rectangle

Gewichtete Summen vom gegenüberliegenden Punkt

$$\begin{aligned} f(x, y) = & (1 - \alpha)(1 - \beta)f_{i,j} \\ & + \alpha(1 - \beta)f_{i+1,j} \\ & + (1 - \alpha)\beta f_{i,j+1} \\ & + \alpha\beta f_{i+1,j+1} \end{aligned}$$



- Weighted by local areas of the opposite point
- Bilinear interpolation is not linear (but quadratic)!
- Cannot be inverted easily!

Local Interpolation

Tri-linear interpolation (3D)

- Voxel-grids, non-linear operation
- 27 vs. 7 multiplications
- Computationally intensive on large voxel datasets

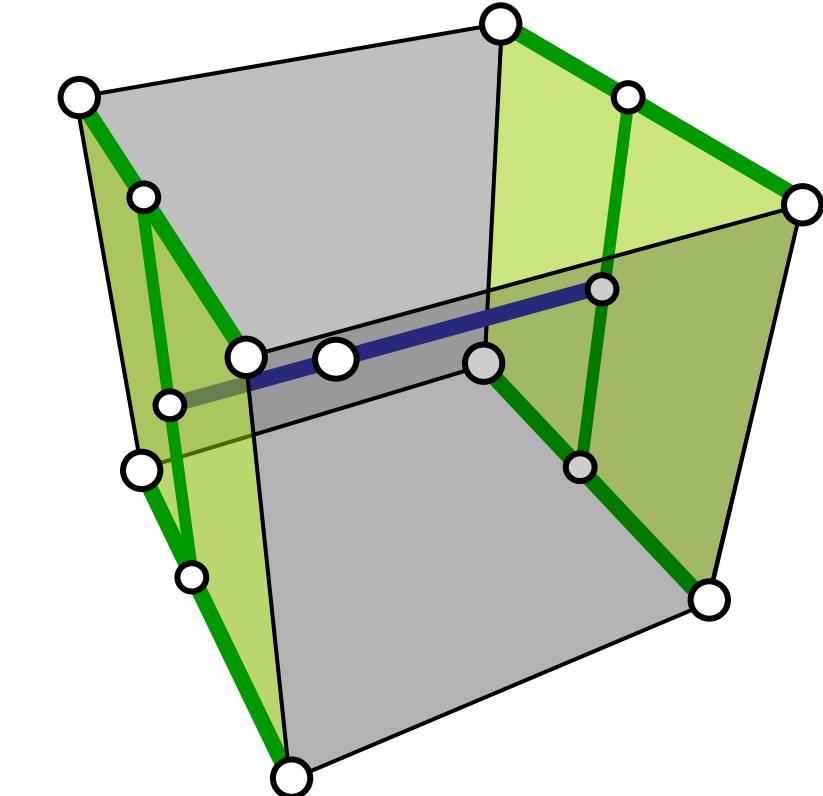
$$\begin{aligned} f_{00} &= \alpha f_{001} + (1 - \alpha) f_{000} \\ f_{01} &= \alpha f_{011} + (1 - \alpha) f_{010} \\ f_{10} &= \alpha f_{101} + (1 - \alpha) f_{100} \\ f_{11} &= \alpha f_{111} + (1 - \alpha) f_{110} \end{aligned}$$

4 linear interpolations

2 bi-linear
interpolations

1 tri-linear
interpolation

$$\begin{aligned} f_0 &= \beta f_{10} + (1 - \beta) f_{00} \\ f_1 &= \beta f_{11} + (1 - \beta) f_{01} \\ f(x, y, z) &= \gamma f_1 + (1 - \gamma) f_0 \end{aligned}$$



8 Nachbarpunkte

Local Interpolation

3D-case on a 3D uniform grid

- Trilinear interpolation of points (x,y,z)
 - Straightforward extension of bilinear interpolation
 - Three local coordinates α, β, γ
- Trilinear interpolation is not linear!
- Efficient evaluation (Horner):
$$f(\alpha, \beta, \gamma) = \alpha + \alpha(b + \beta(e + h\lambda)) + \beta(c + f\gamma) + \gamma(d + g\alpha)$$
 - Coefficients a, b, c, d, e, f, g from data at corner vertices

Local Interpolation

Linear interpolation on a triangle

- Point P can be expressed as

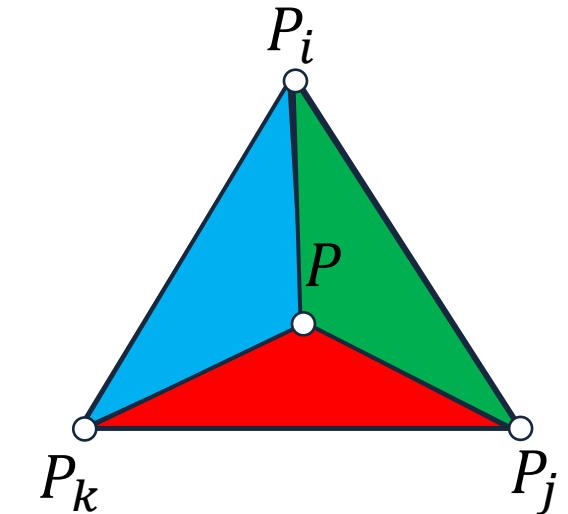
$$P = a_i P_i + a_j P_j + a_k P_k$$

- with $a_i + a_j + a_k = 1$ and

$$a_i = \frac{\text{Area}(\Delta PP_j P_k)}{\text{Area}(\Delta P_i P_j P_k)}$$

$$a_j = \frac{\text{Area}(\Delta P_i PP_k)}{\text{Area}(\Delta P_i P_j P_k)}$$

$$a_i = \frac{\text{Area}(\Delta PP_j P_k)}{\text{Area}(\Delta P_i P_j P_k)}$$



If $a_i = a_j = a_k = \frac{1}{3}$
 $\Rightarrow P$ is **barycenter** of the triangle

a_i, a_j, a_k are **barycentric** coordinates
(Baryzentrische Koordinaten)

Interpolation

Barycentric interpolation

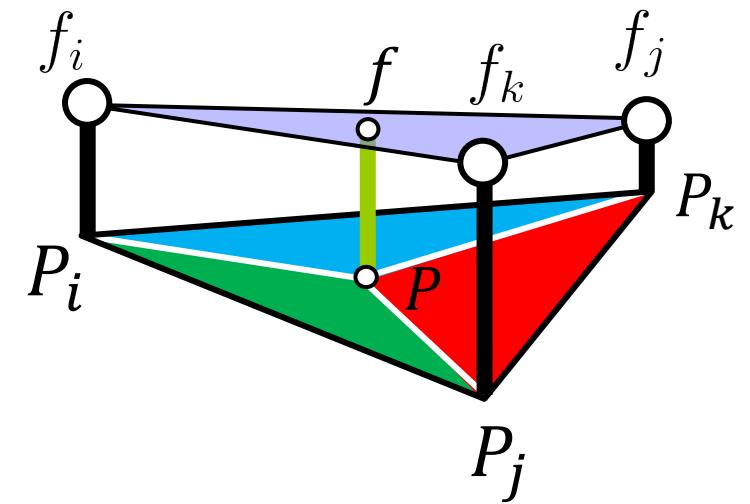
- Determine barycentric coordinates of point P

$$P = a_i P_i + a_j P_j + a_k P_k$$

- Then determine function f value using the same weights

$$f = a_i f_i + a_j f_j + a_k f_k$$

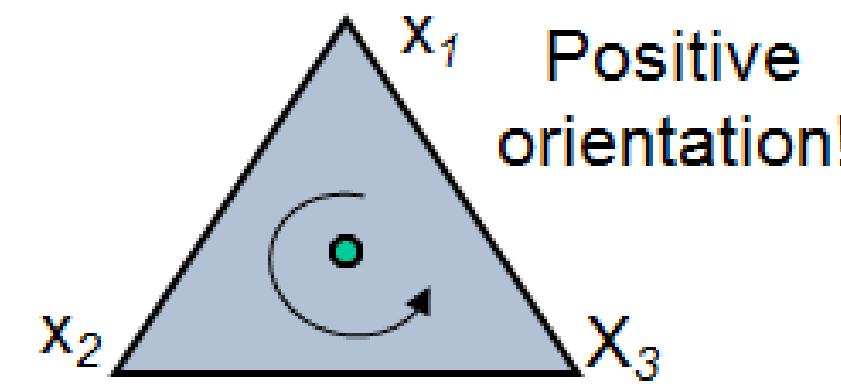
- Note
 - Points outside of the triangle can be expressed with barycentric coordinates
 - Then some of the weights are negative
 - But still: $a_i + a_j + a_k = 1$



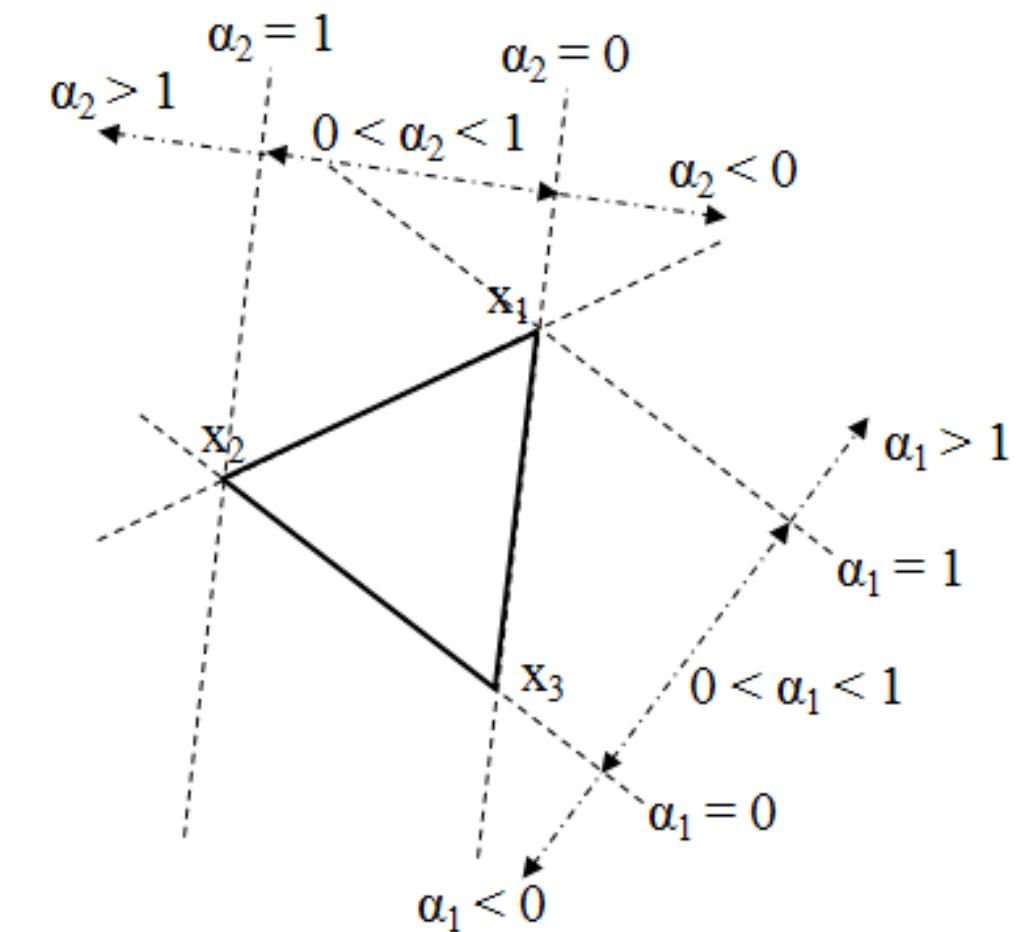
Local Interpolation

Localization with barycentric coordinates

- Use right-hand-rule for orientation



- A point is inside the triangle if and only if for all weights: $0 < \alpha_i < 1$
- Use for cell search



Local Interpolation

Barycentric coordinates in multiple dimensions

- Simplex in $\mathbb{R}^d \rightarrow$ defined by its vertices

- $d+1$ affinely independent points

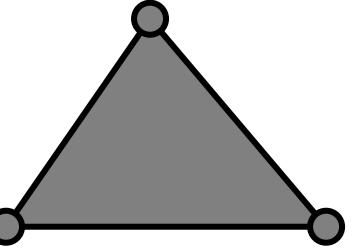
- 0D: point



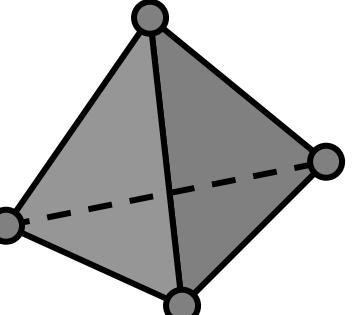
- 1D: line



- 2D: triangle



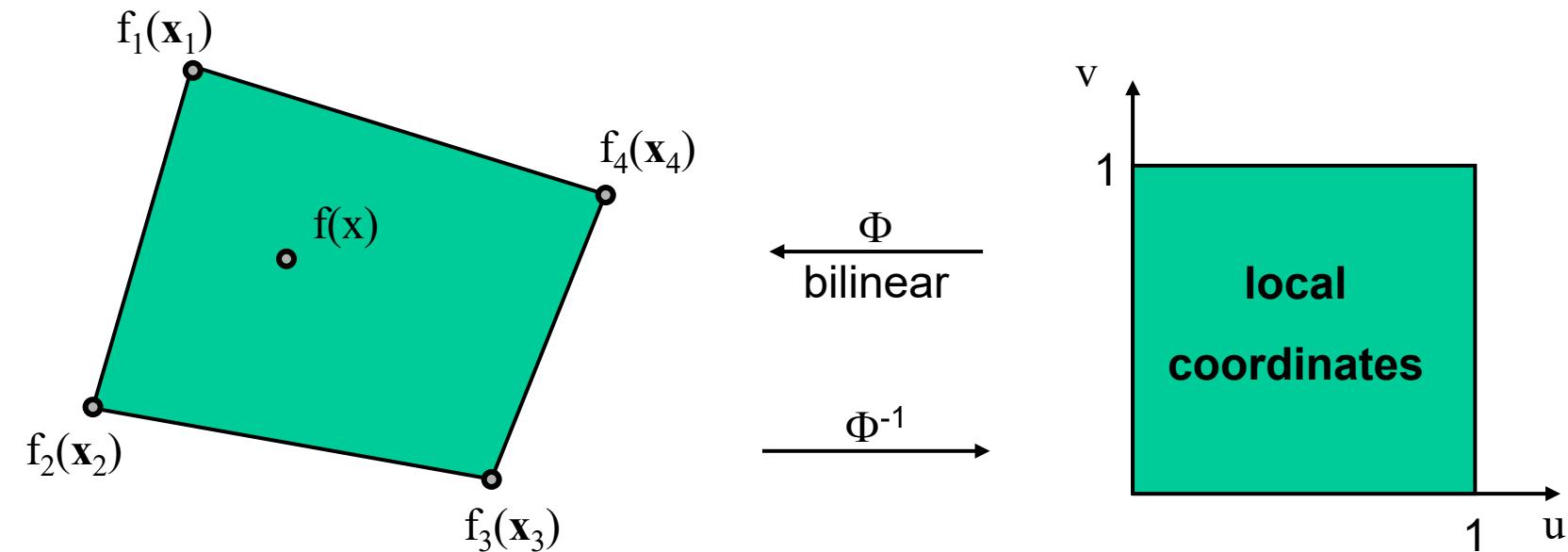
- 3D: tetrahedron



Local Interpolation

Interpolation in a generic quadrilateral

- Main application in curvilinear grids
 - Find a parameterization of arbitrary quadrilaterals
 - Use local coordinates for interpolation

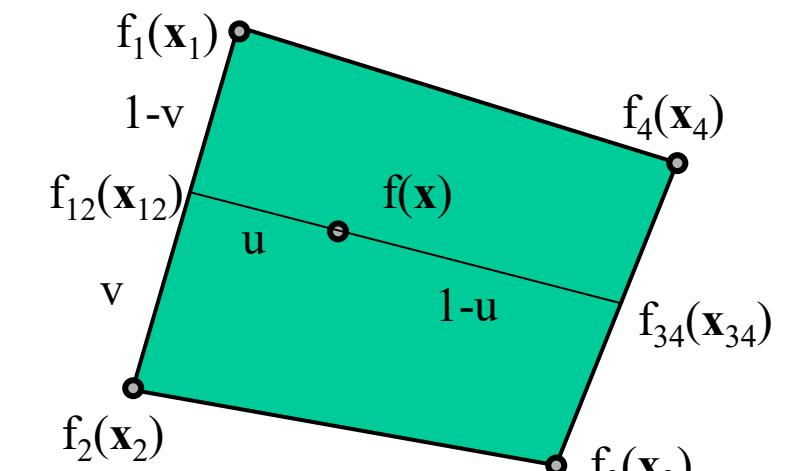


Local Interpolation

- We know $\Phi(u, v) = (x, y)^T$
- Mapping from rectangular to quadratic domain
 - Bilinear interpolation on rectangle

$$\begin{aligned}\mathbf{x}_{12} &= v\mathbf{x}_1 + (1 - v)\mathbf{x}_2 \\ \mathbf{x}_{34} &= v\mathbf{x}_4 + (1 - v)\mathbf{x}_3\end{aligned}$$

$$\Phi(u, v) = \begin{pmatrix} x \\ y \end{pmatrix} = u\mathbf{x}_{34} + (1 - u)\mathbf{x}_{12} \quad \text{with} \quad u, v \in [0, 1]$$



- Final value $f = uvf_4 + u(1 - v)f_3 + (1 - u)vf_1 + (1 - u)(1 - v)f_2$

Local Interpolation

Computing the inverse $\Phi^{-1}(x, y) = (u, v)^T$ is more complicated

- Analytically, solve quadratic system for u, v
 - Use numerical solution by Newton iteration instead
- Stencil-walk algorithm [Bunnig '89]
 - In context of flow visualization
- Start with seed points $\Phi(u_0, v_0) = (x_0, y_0)^T$
 - Iteratively, move towards solution (x, y) by means of Jacobian $J_\Phi(u, v)$

Local Interpolation

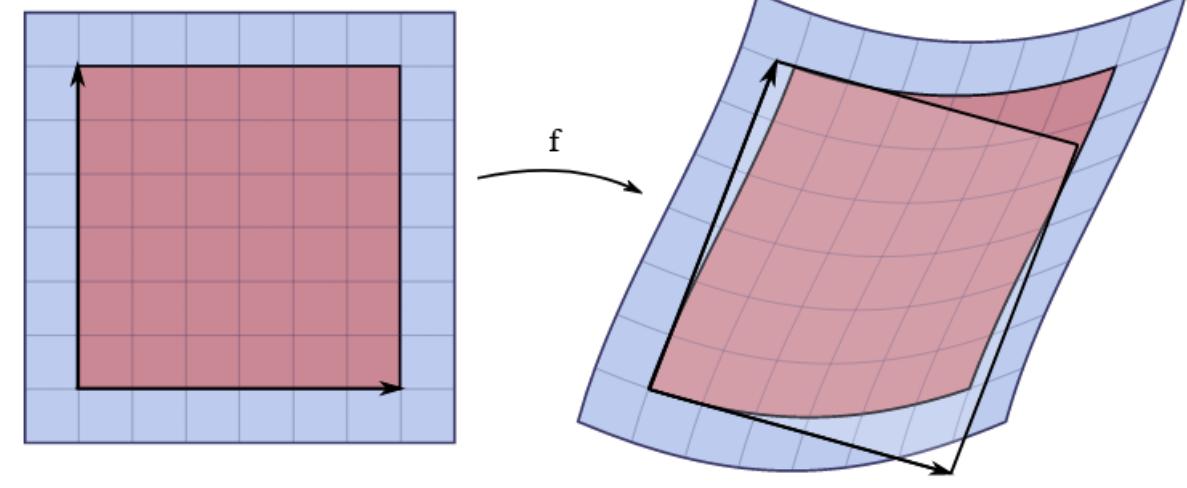
Computing the inverse $\Phi^{-1}(x, y) = (u, v)^T$ is more complicated

- Determine start value
 - $(u_0, v_0) \in \{(0,0), (1,0), (0,1), (1,1)\}$ depending on closest vertex (x, y)
 - Alternatively, start from center: $u_0 = v_0 = 1/2$
- Jacobi matrix $J_\Phi(u, v)$
 - $J_\Phi(a)_{i,j} = \frac{\partial \Phi_i}{\partial x_j}$, with $x = (u, v)$
 - The Jacobian describes direction and speed of position changes of Φ when the parameters are varied

Local Interpolation

- From Taylor:

$$\begin{pmatrix} x \\ y \end{pmatrix} - \Phi(u_0, v_0) \approx J_\Phi(u_0, v_0) \begin{pmatrix} u_1 - u_0 \\ v_1 - v_0 \end{pmatrix}$$



solange Ableiten bis keine Änderung mehr

- Determine local coordinates (u_1, v_1) as follows:

- Solve $J_\Phi(u_0, v_0) \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \Phi(u_0, v_0)$ and set $(u_1, v_1) = (u_0 + \Delta u, v_0 + \Delta v)$

- Newton iteration

```

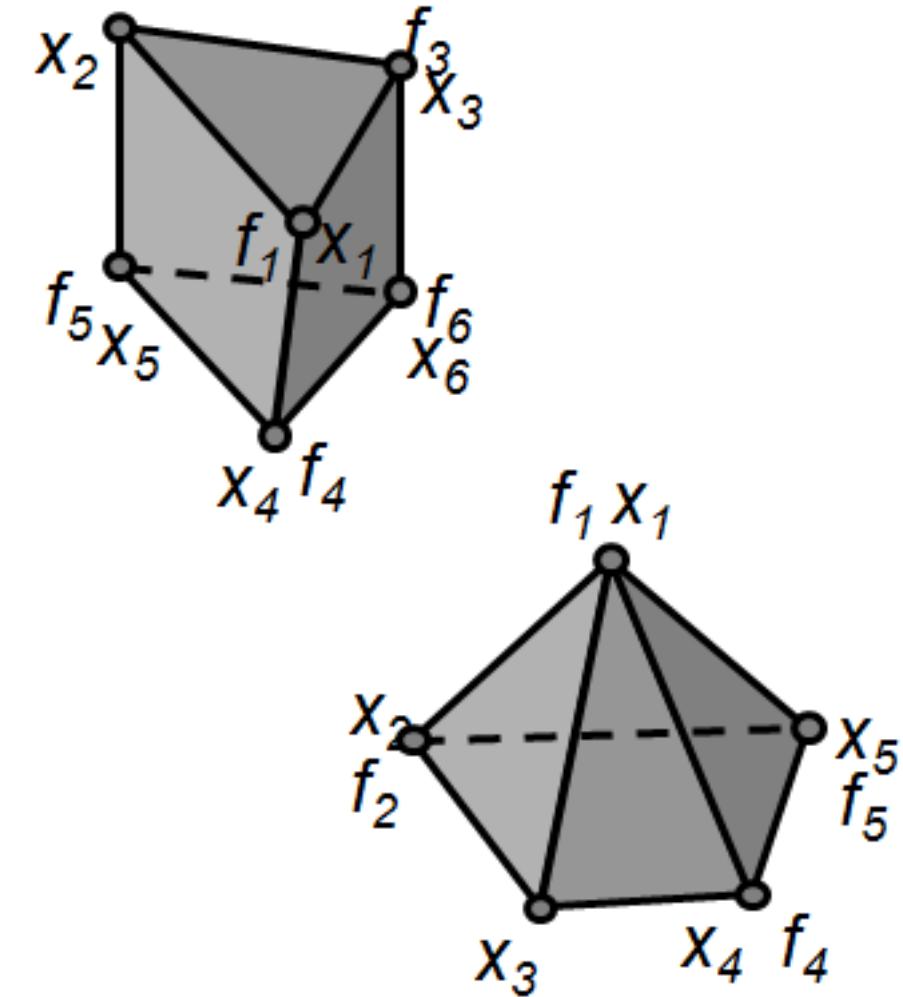
start with start configuration as seed points
while (||x - Φ(u, v)|| > ε)
    compute J(Φ(u, v))
    transform x in coordinate system J(Φ):
        (Δu, Δv) = J(Φ(u, v))⁻¹ · (x - Φ(u, v))
    update (ui+1, vi+1) = (ui, vi) + (Δu, Δv)

```

Local Interpolation

More 3D cases

- Tetrahedron → Barycentric
- Hexahedron (Box) → Trilinear
- Prism (over triangle)
 - Twice barycentric → once linear
- Pyramid (over quadrangle)
 - Bilinear on base face → then linear



3.6.2 Global Interpolation

Global Interpolation

More advanced interpolation functions

- Interpolation aims to estimate function values based on samples
 - Linear interpolation: very simple + acceptable quality
 - Considers only local neighborhood
- To improve quality find a better approximation function
 - Increases calculation cost (usually)
 - The right choice is often application dependent
- Might consider bigger neighborhood or all samples → global interpolation

Global Interpolation

Lanczos Filter

- sinc on a sliding window (neighborhood size)
 - Choose window size a (number of samples), then

$$L(x) = \begin{cases} 1, & x = 0 \\ \frac{a \sin(\pi x) \sin(\pi x/a)}{\pi^2 x^2}, & -a \leq x < a \wedge x \neq 0 \\ 0, & sonst \end{cases}$$

- and the interpolation function is $f(x) = \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} x_i L(x - i)$

Global Interpolation

Splines

- In general, given $n + 1$ samples $\{x_0, x_1, \dots, x_n\}$ and corresponding values $\{y_0, y_1, \dots, y_n\}$, where $x_i \neq x_k$, if $i \neq k$ and $0 \leq i, k \leq n$ the set of polynomials

$L_i(x) = \prod_{\substack{0 \leq k \leq n \\ i \neq k}} \frac{x - x_k}{x_i - x_k}$ of degree form a basis of a space where we can construct a

function $f(x) = \sum_{i=0}^n y_i L_i(x)$, that interpolates the given samples (i.e. $f(x_i) = y_i$)

- The polynomials are called **Lagrange-Polynomials** and there is a **unique** polynomial for any given set of points

Global Interpolation

Splines

- Problems
 - For $n+1$ data points we get a polynomial of degree n
 - High degree polynomials lead to oscillation at the edges of intervals (Runge's phenomenon) – hence they are infeasible for interpolation of large data
- Solution approach
 - Split the interval $x_0 \leq x_n$ into subintervals $[t_0, t_1], \dots, [t_i, t_{i+1}], \dots [t_{n-1}, t_n]$
 - Interpolate each subinterval with appropriate polynomial
 - Connect the individual segments → **Spline**

Global Interpolation

Splines

- Spline polynomials form a base for $f: f(x) = \sum_{i=0}^n B_i^n(x)b_i$ on the intervals
 - B_i^n are the basis functions, b_i the control points, forming a control polygon
 - b_i is usually not equal to x_i (depending on the spline)
 - b_i can be chosen such that $f(x_i) = y_i$
 - Continuity constraints at the interval borders depend on spline type
- There exist different Spline types with different properties
 - Cardinal Splines, Catmull-Rom Splines, Bézier-Splines, B-Splines, ...

Global Interpolation

Catmull-Rom Interpolation

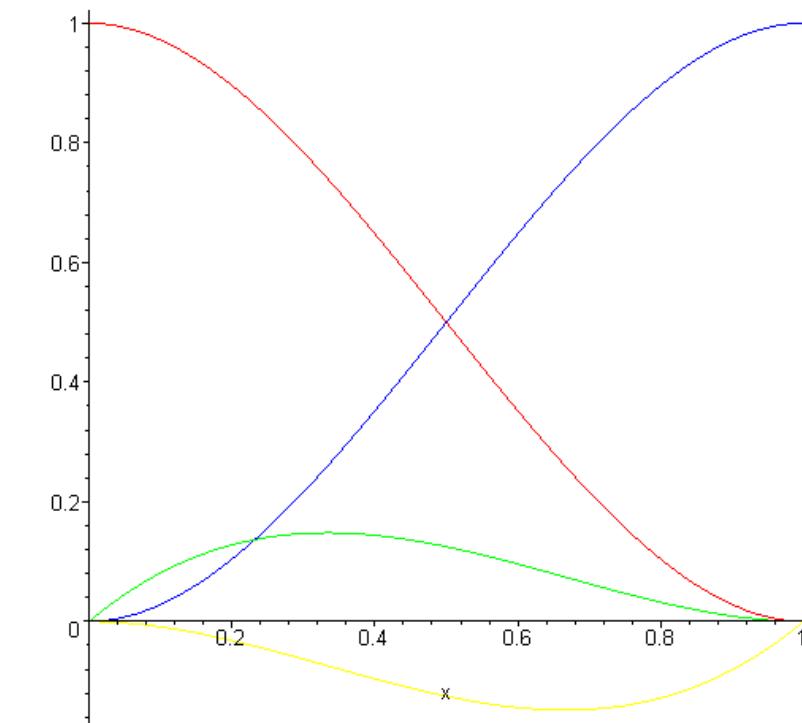
- Compromise between "local linear" and "global B-Spline"
- Properties
 - Spline passes through all control points (i.e. $b_i = x_i$!)
 - Piecewise cubic, C^1 -continuous
 - There are *no discontinuities* in tangent direction and magnitude
 - Not C^2 -continuous
 - Second derivative is *linearly interpolated* within each segment
 - Error: $O(h^3)$

Global Interpolation

Hermite-Interpolation

- Given P_0, P'_0, P_1, P'_1 , then there exists a unique cubic polynomial f with
 - $f(t_0) = P_0, f'(t_0) = P'_0, f(t_1) = P_1, f'(t_1) = P'_1$

Hermite polynomials	$f(0)$	$f'(0)$	$f'(1)$	$f(1)$
$H_0(t) = (1-t)^2(2t+1)$	1	0	0	0
$H_1(t) = t(1-t)^2$	0	1	0	0
$H_2(t) = -t^2(1-t)$	0	0	1	0
$H_3(t) = t^2(3-2t)$	0	0	0	1

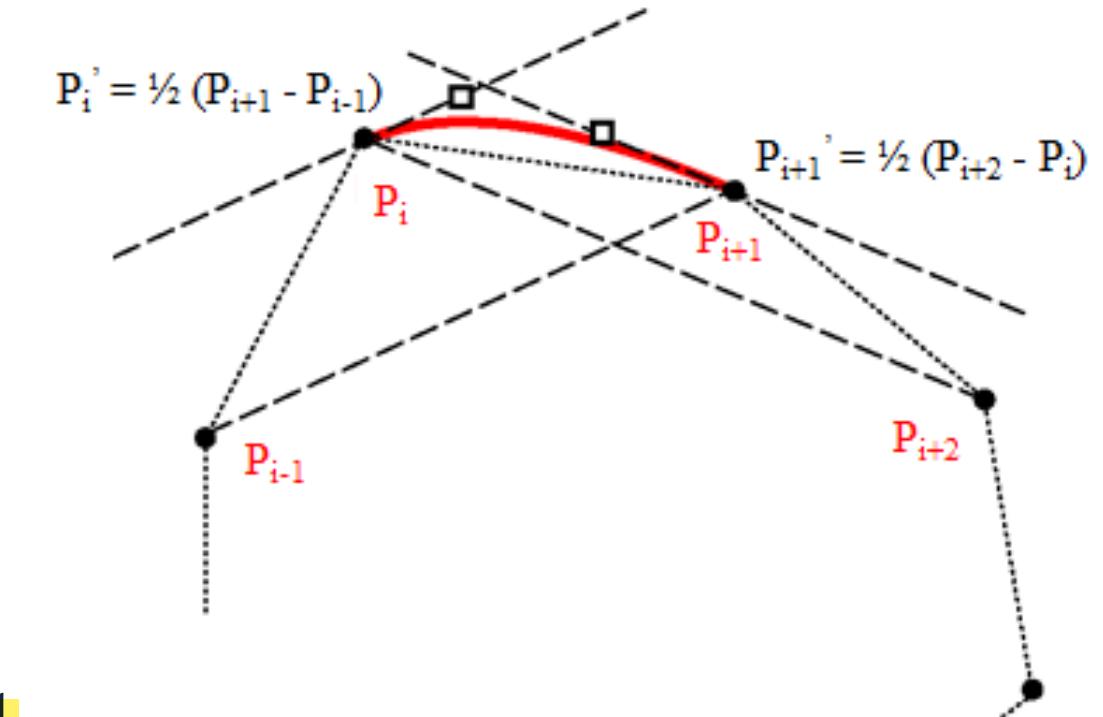


- Then $f(t) = P_0H_0(t) + P'_0H_1(t) + P'_1H_2(t) + P_1H_3(t)$ interpolates the positions P_0 and P_1 and the derivatives P'_0 and P'_1 !

Global Interpolation

Cardinal splines

- Subset of Hermite curves
 - Calculate tangent points from control points: $P'_i = \frac{1}{2}(1 - c)(P_{i+1} - P_{i-1})$
 - c is a *tension parameter*
- **Catmull-Rom Splines**
 - Subset of cardinal splines where $c = 0$
 - Estimate derivatives by symmetric differences
 - Derivatives at a point P_i are parallel to the line $[P_{i-1}; P_{i+1}]$
 - We have the local property, i.e. if $t_i < t < t_{i+1}$, then $f(t)$ depends only on $P_{i-1}, P_i, P_{i+1}, P_{i-2}$

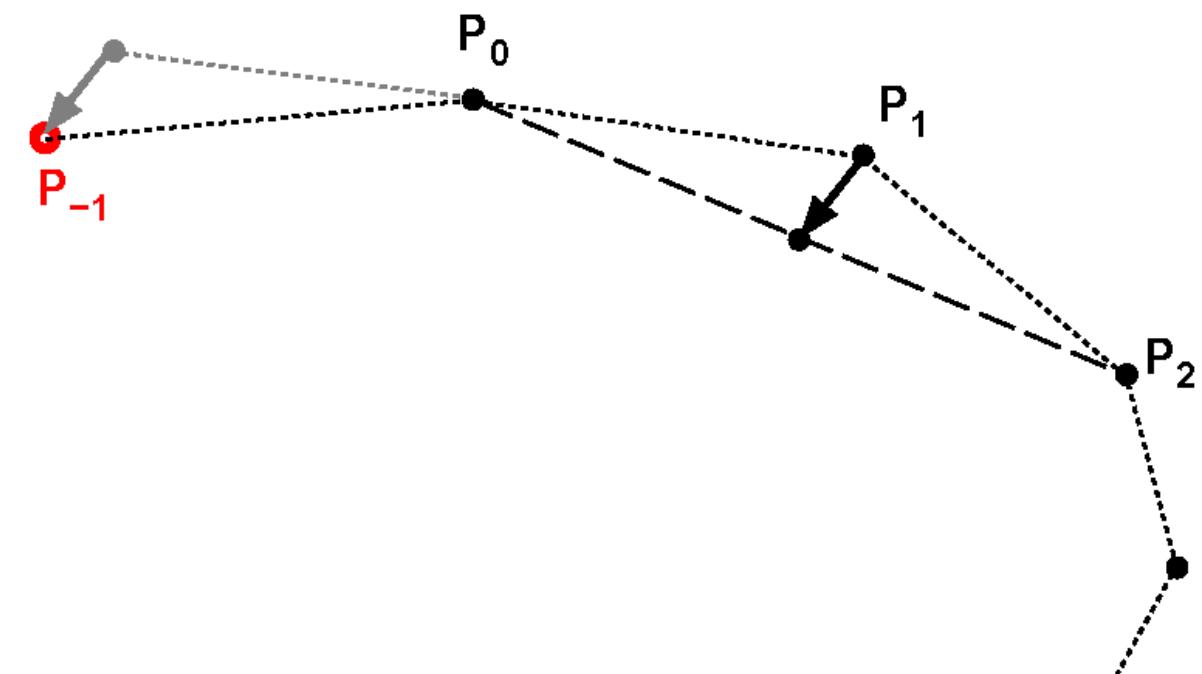


Ableitung am Punkt P_i

Global Interpolation

Estimate of derivatives in border points

- For this determine P_{-1} resp. P_{n+1}
 - i.e. estimate tangent along the lines $[P_{-1}; P_0]$, and $[P_n; P_{n+1}]$
 - This leads to the natural boundary condition $f''(t_0) = f''(t_n) = 0$



Global Interpolation

Further Reading and advanced topics

- Curves and Surfaces (Geometric Modelling)
 - Gerald Farin, Curves and Surfaces for CAGD, 5th Ed., Elsevier
 - Banchoff & Lovett, Differential Geometry of Curves and Surfaces, 3rd Ed., Chapman and Hall/CRC
 - Abate & Tovena, Curves and Surfaces, Springer
- Advanced Topics: Scattered Data Interpolation using Radial Basis Functions
 - e.g. Iske, Scattered Data Modelling Using Radial Basis Functions, Springer

4. 1D and 2D Scalar Fields

Introduction

Basic mapping techniques

- Mapping
 - from (filtered) data to renderable representation (visualization pipeline)
- Most important part of visualization
- Possible visual representations
 - Position
 - Size
 - Orientation
 - Shape
 - Brightness
 - Color (hue, saturation)
 - ...

Introduction

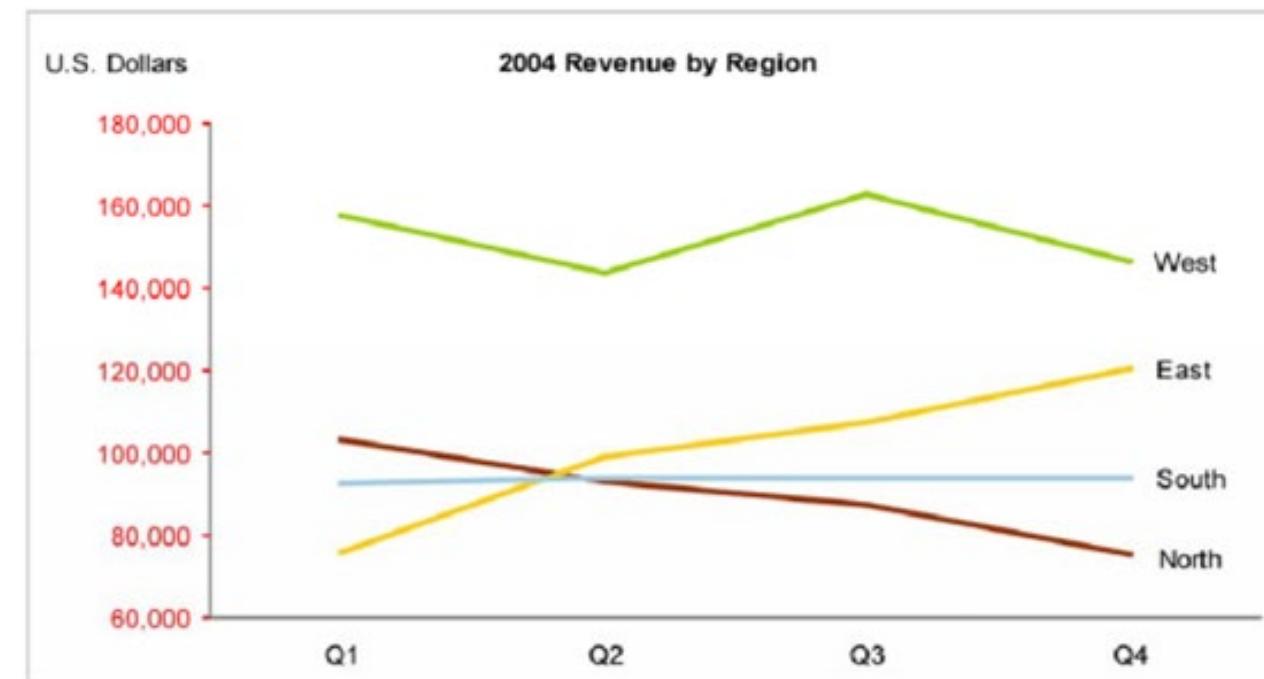
1D Functions - Typical strategy when creating a graph

- Stepping through a series of choices, including
 - Different types of graphs
 - Several aspects of appearance
- Choices are often made as if one is sleepwalking
 - Neglected issue: Why one choice is better than another?
- What we have to know
 - Nature of the data
 - Graphing conventions
 - Psychological investigations
 - A bit about visual perception
 - Evaluate appropriateness of mapping approaches

Data and Scales

Quantitative information consists of both

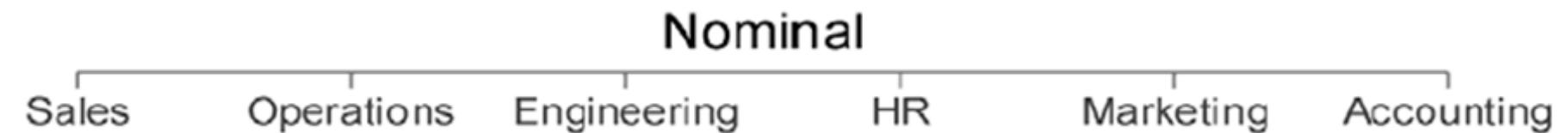
- Quantitative data: Numbers
- Categorical data: Labels what numbers measure
- Example: Categorical (horiz. axis) and quantitative data (vert. axis)



Data and Scales

Three types of categorical scales

- **Nominal scales**
 - Consists of discrete items
 - Belong to a common category
 - Do not relate to one another in any particular way
 - Differ in name only (nominally!)
 - Have no particular order and don't represent quantitative values

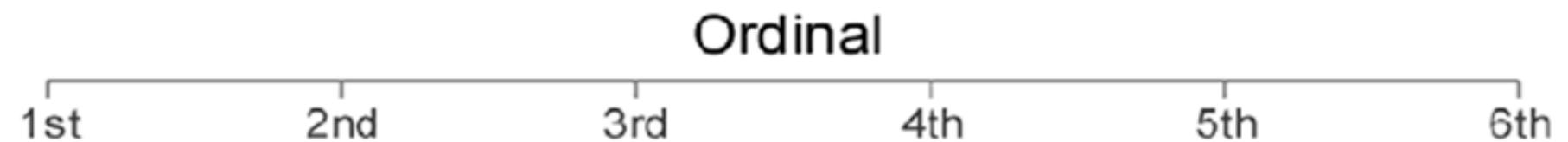


- Further examples
 - Continents: the Americas, Asia, Europe, ...
 - Departments (e.g. sales, marketing and finance)

Data and Scales

Three types of categorical scales

- **Ordinal scales**
 - Consists of items with intrinsic *order*
 - Items do not represent quantitative values

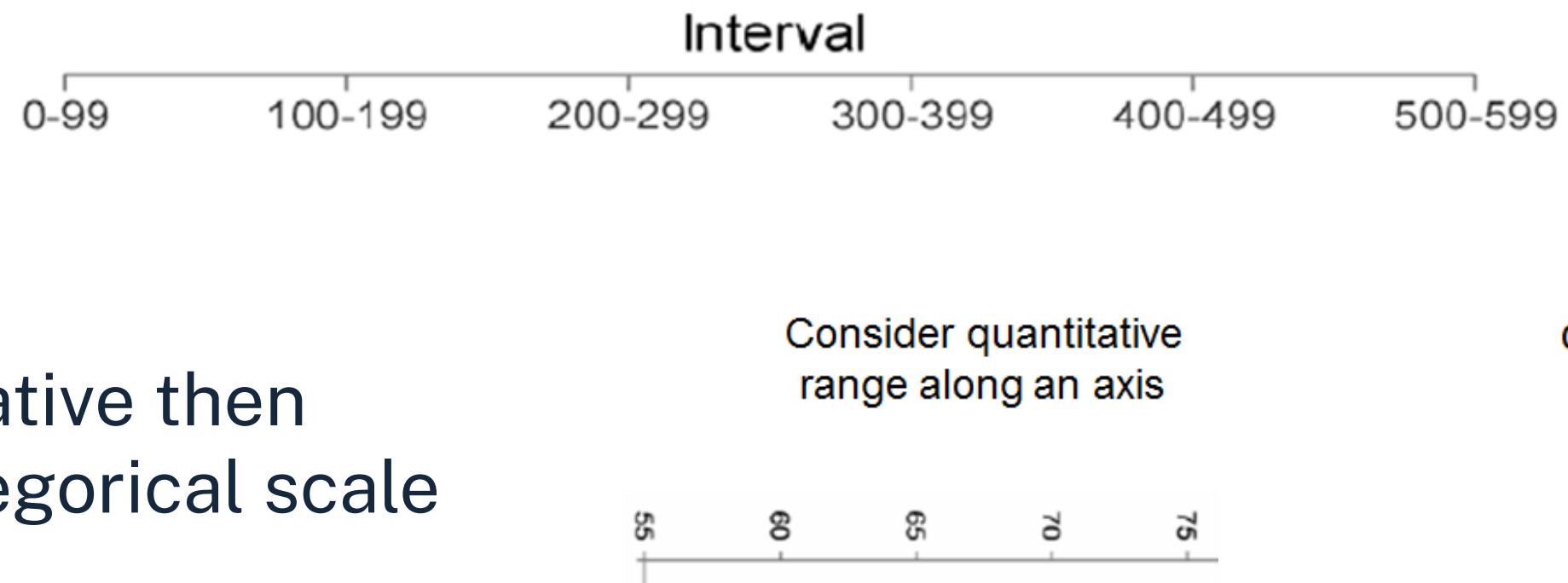


- Further examples involve rankings
 - "A, B and C"
 - "small, medium and large"
 - "poor, below average, average, above average and excellent"

Data and Scales

Three types of categorical scales

- **Interval scales**
 - Consists of items with *intrinsic order*
 - Represent quantitative values as well



- Example
 - Start as quantitative then convert into categorical scale

- Conversion into a categorical scale according to
1. $> 55 \text{ and } \leq 60$
 2. $> 60 \text{ and } \leq 65$
 3. $> 65 \text{ and } \leq 70$
 4. $> 70 \text{ and } \leq 75$
 5. $> 75 \text{ and } \leq 80$

4.1 Diagram Techniques

Diagram Techniques

- Discrete range of definition
 - 1D drawing / 2D drawing
- Example
 - Distance from reference location (e.g. Erlangen)

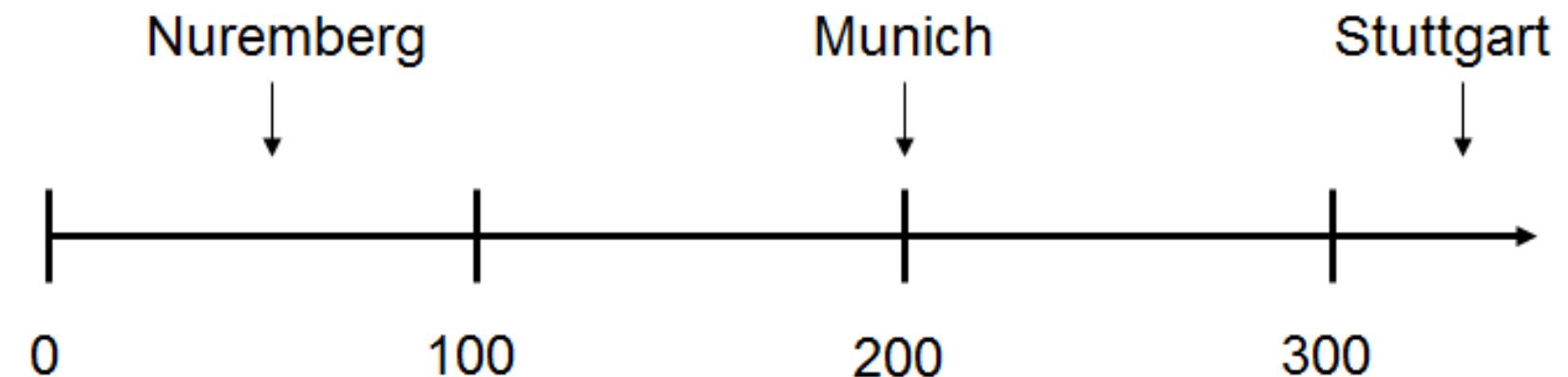
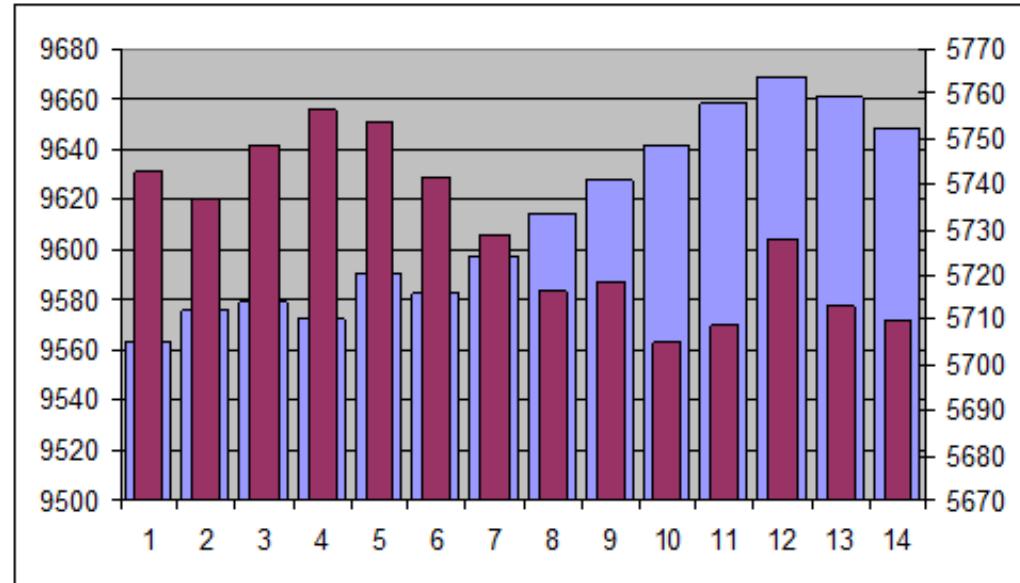


Diagram Techniques

Bar charts

- Appropriate for **nominal and ordinal scales**
 - Individual items *not related closely enough* for link with lines



- Example
 - Transition from one sales region to the next
 - Use bars - inappropriate to display with lines

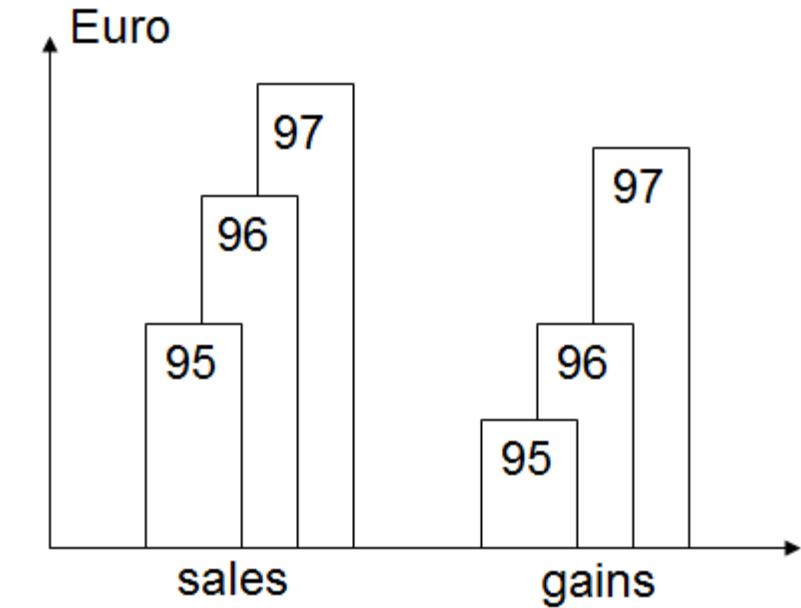


Diagram Techniques

Pie charts

- Quantitative data that adds up to a (fixed?) number

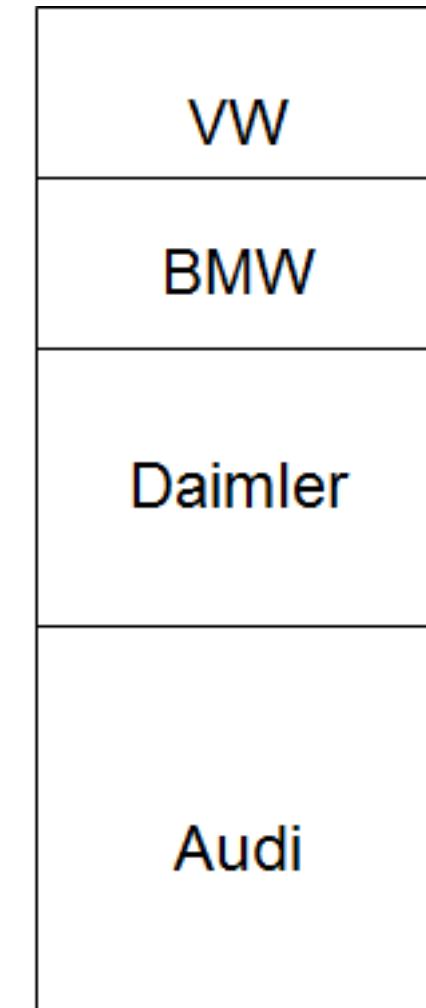
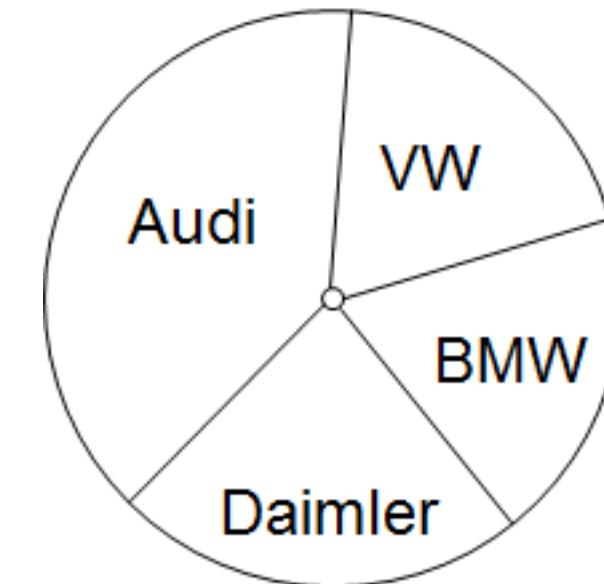
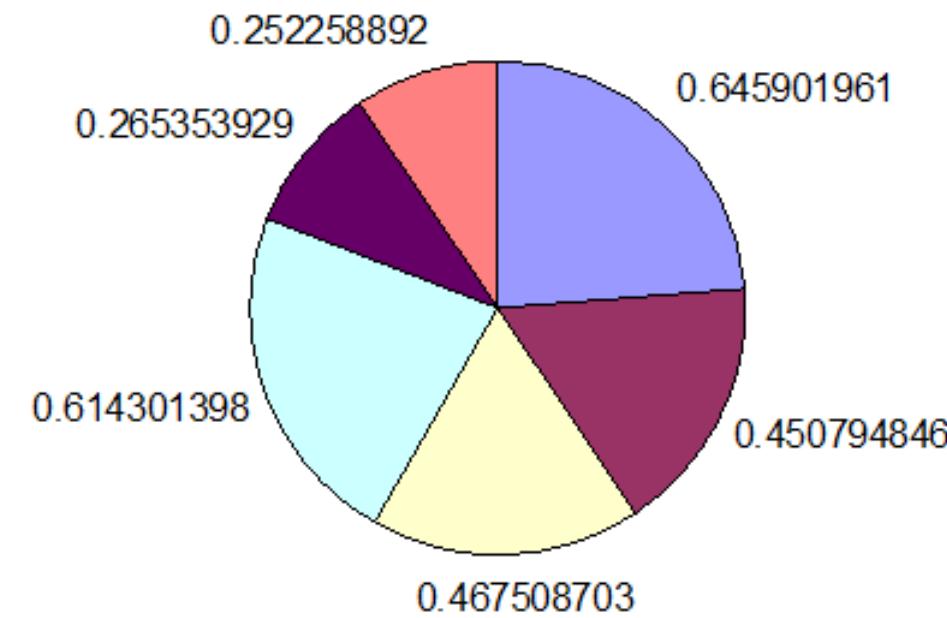


Diagram Techniques

Histogram

- Measure the frequency (number) of items
 - Very important in scientific analysis
- Example
 - Grey values in an 8-bit image

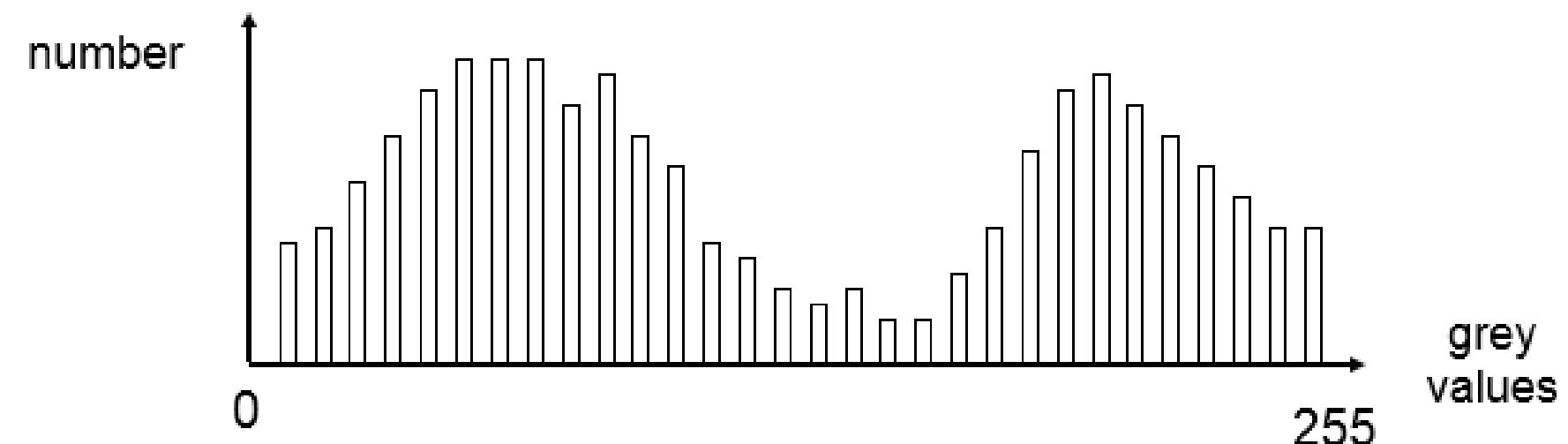
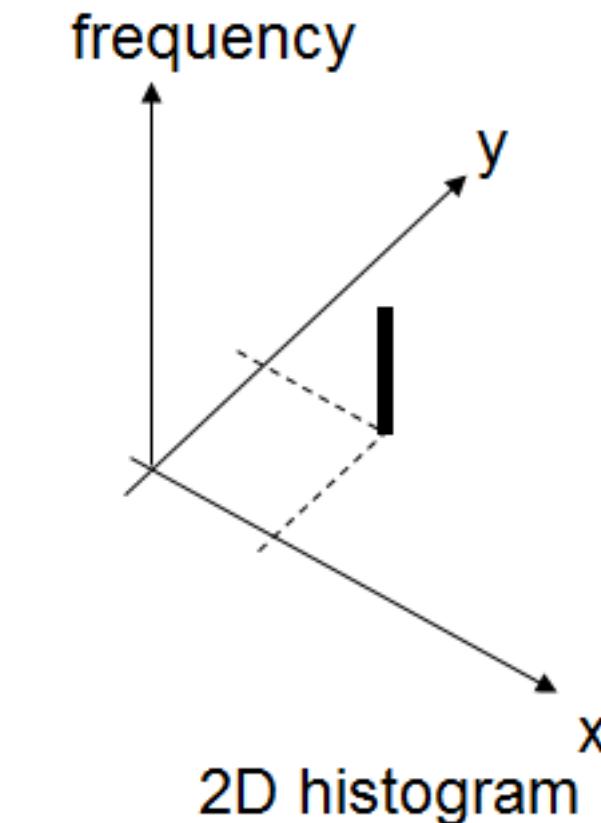
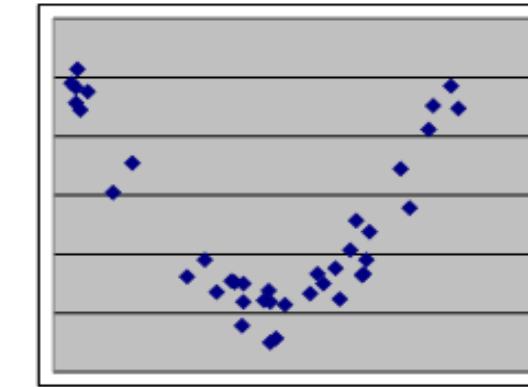
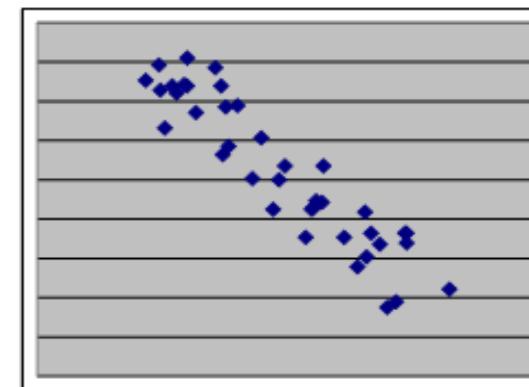
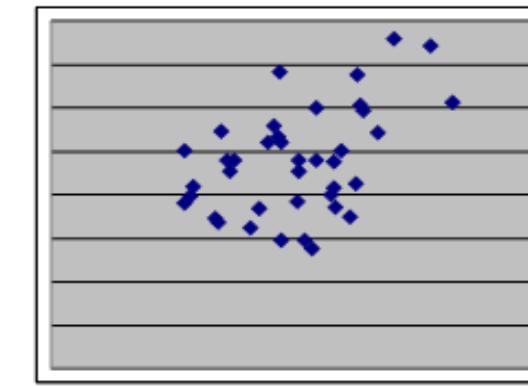
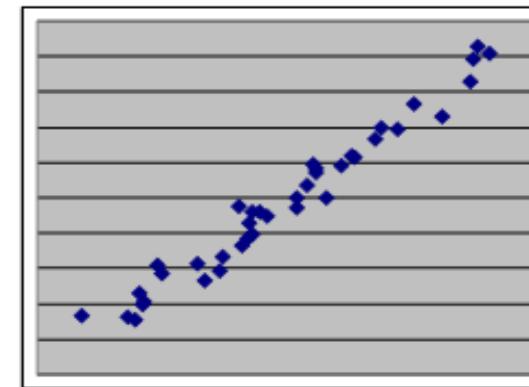


Diagram Techniques

Scatter plots

- Quantitative scales along both axes
- Visual recognition of correlations



2D histogram

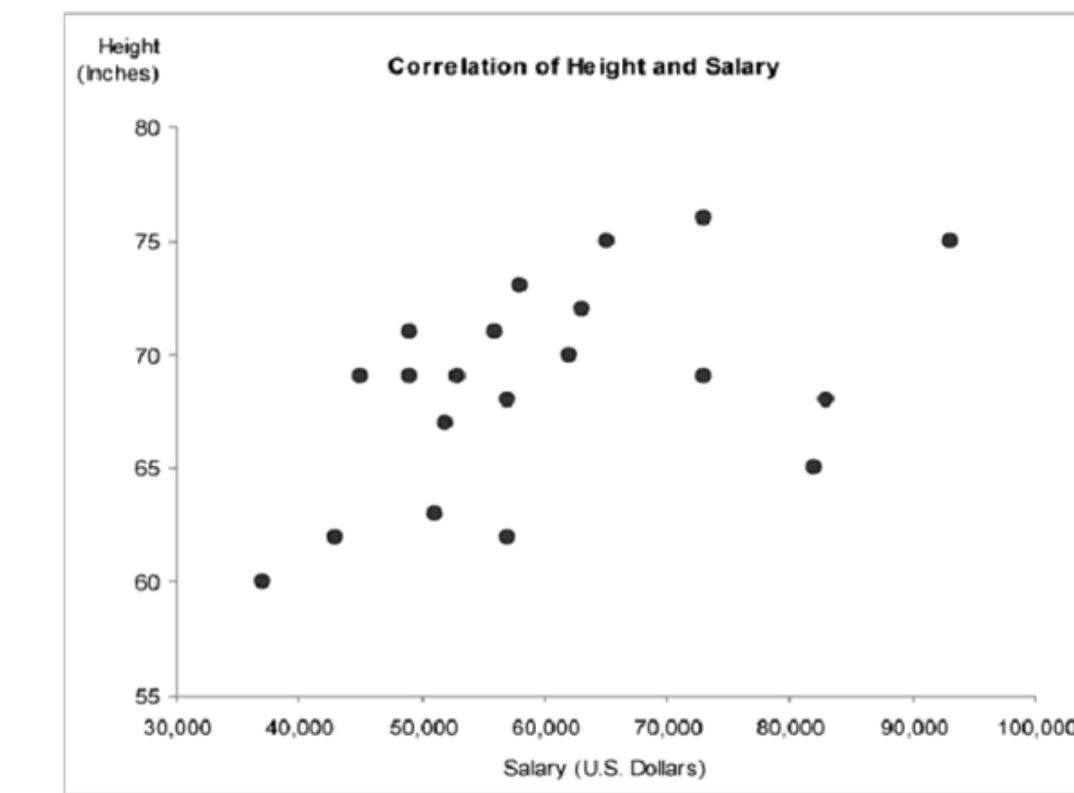


Diagram Techniques

Line graphs

- Connection between points (*interpolation!*)
- Appropriate for interval scale
 - Shows trend of patterns in the data
- Example
 - Change from one day to the next or from one price to the next
 - Appropriate to use lines for display

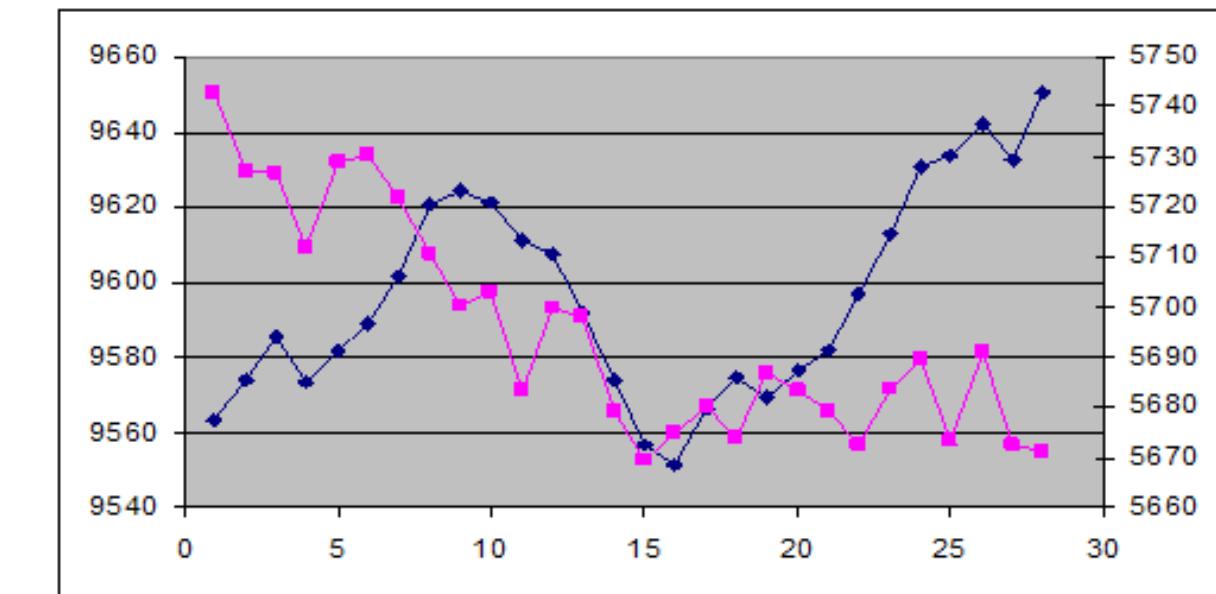
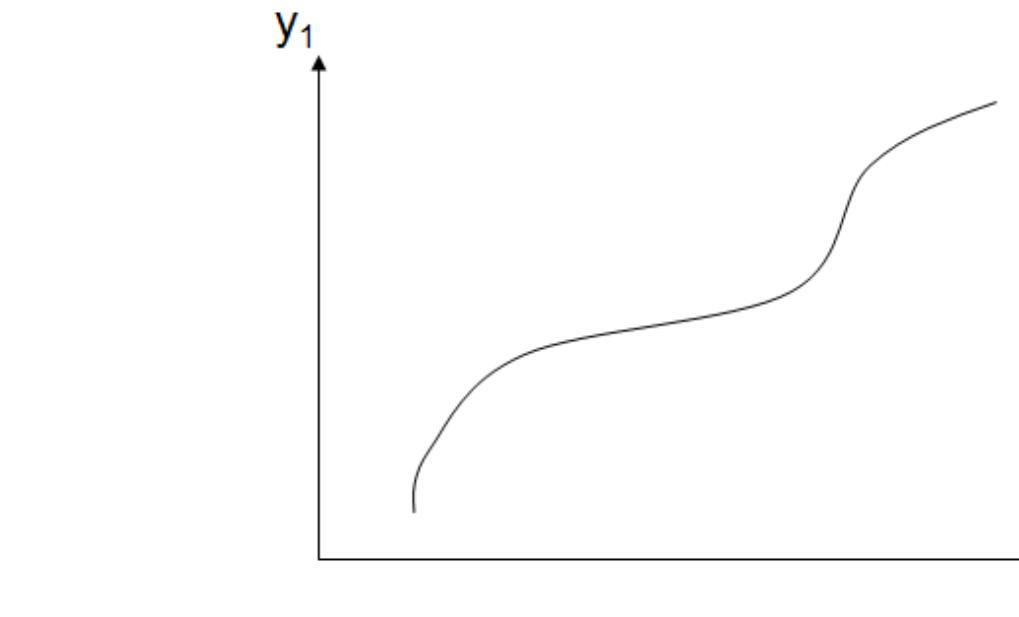


Diagram Techniques

Line graphs

- Inappropriate and appropriate use of lines

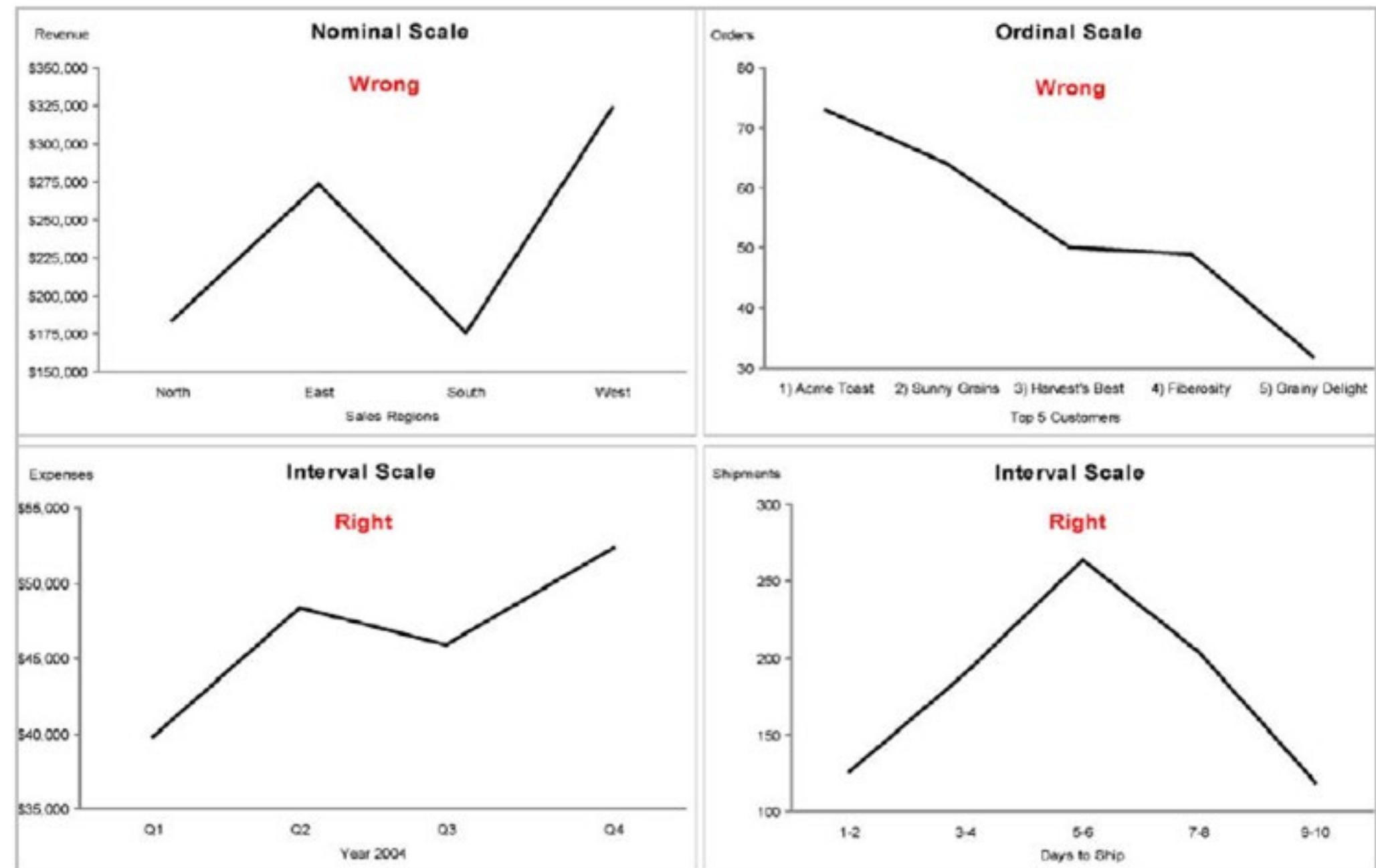
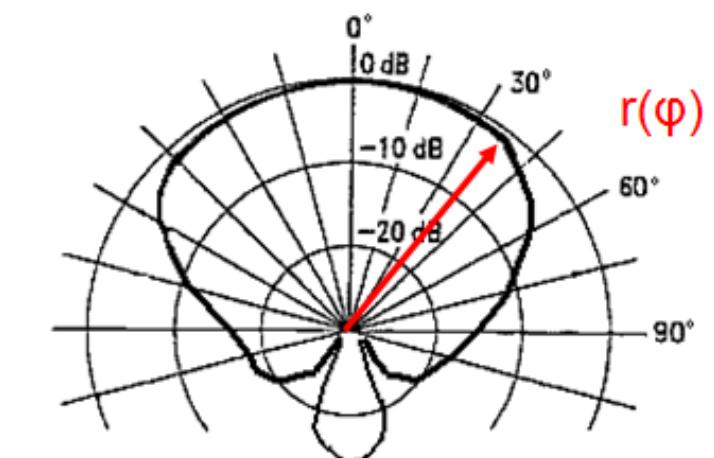
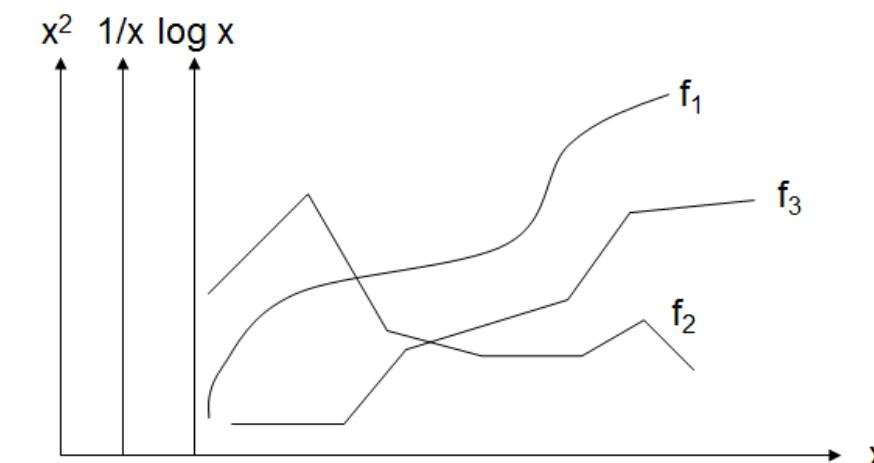
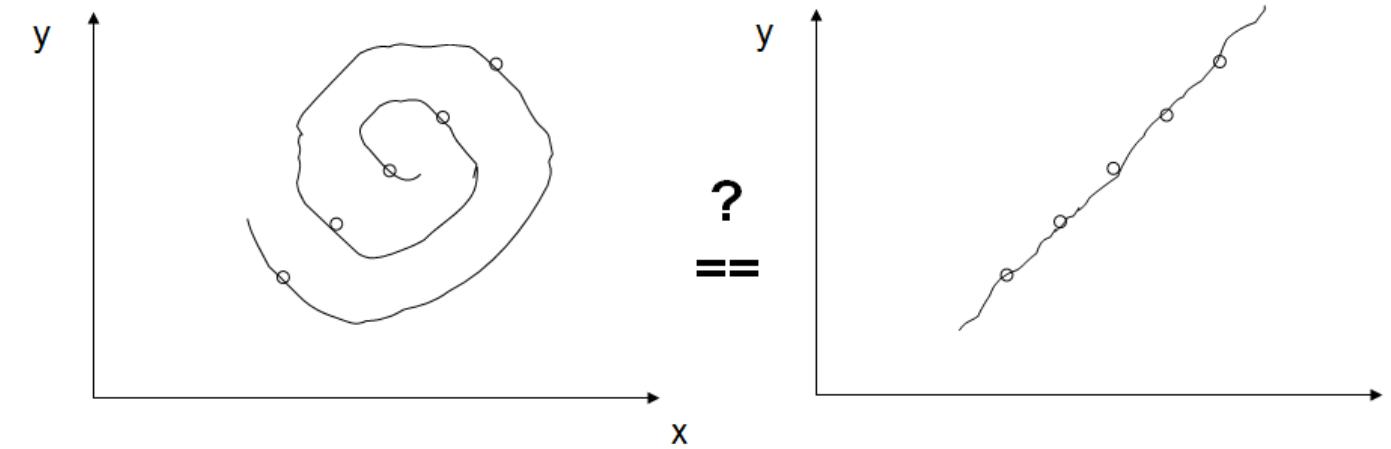
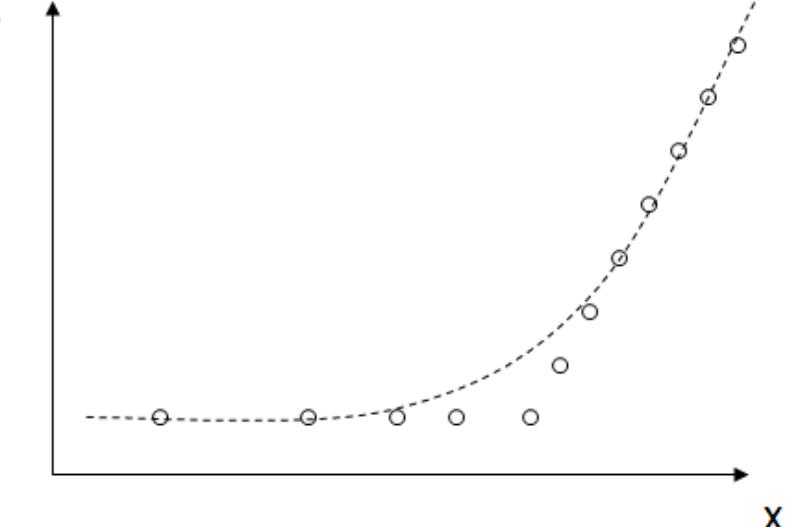
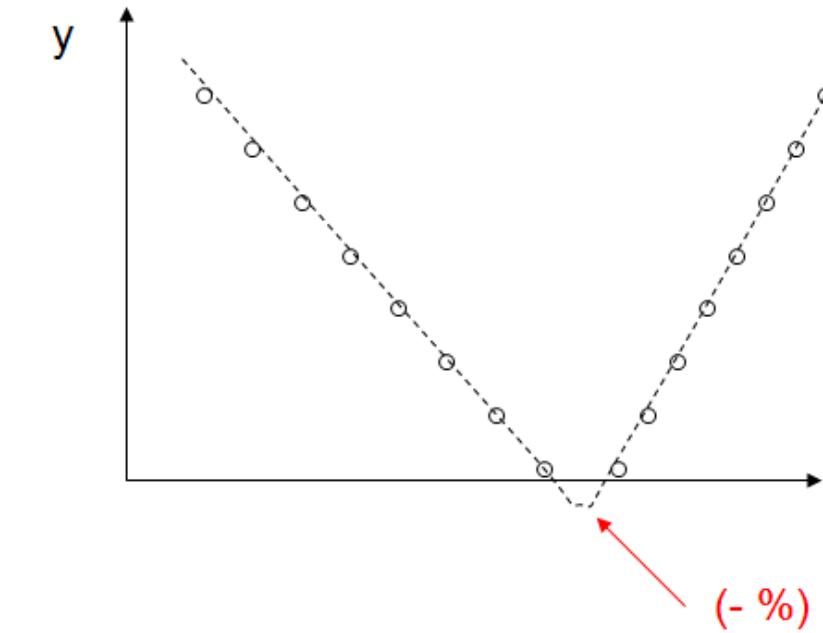


Diagram Techniques

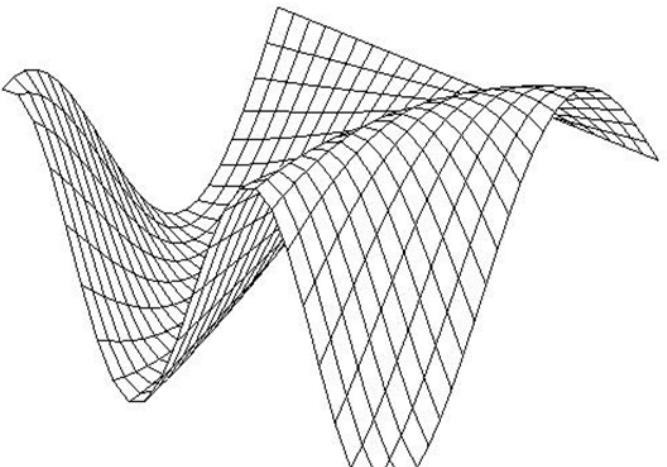
Line graphs

- Avoid "blind" interpolation
- Show all data points
- Set an order
- Caption of axes
 - Several ordinates / y-axes
- Additional annotations
 - e.g. Labels, error bars
- Line type
 - Dashed, dotted, color, ...
- Continuous range-parametric representation

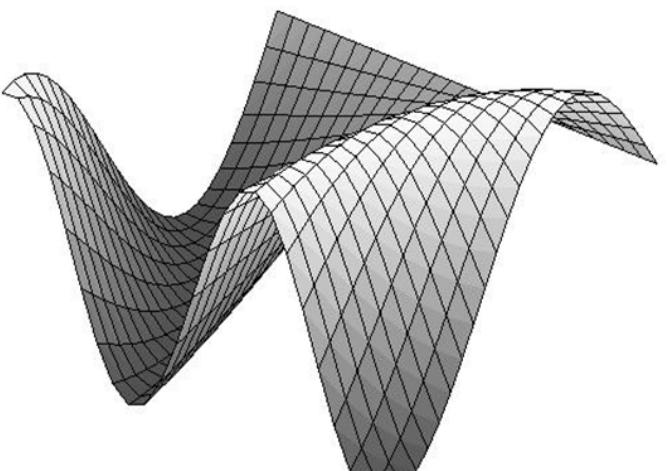
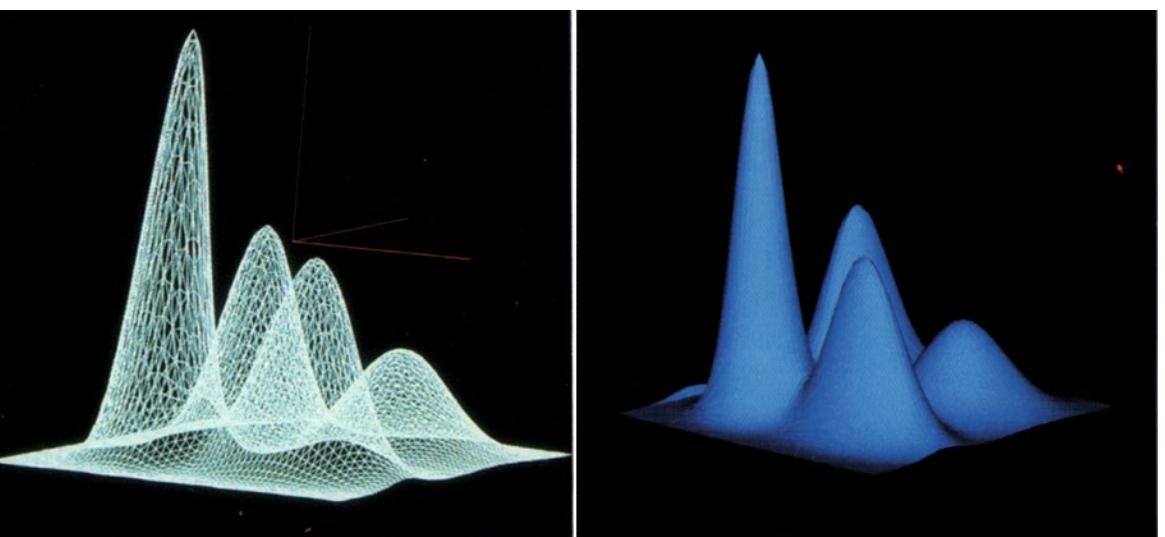
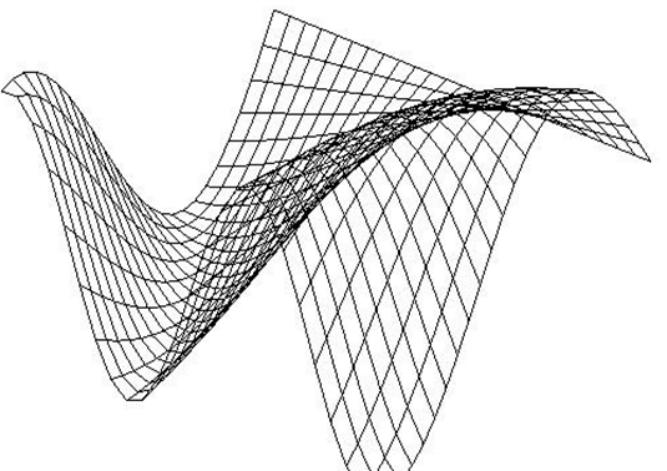
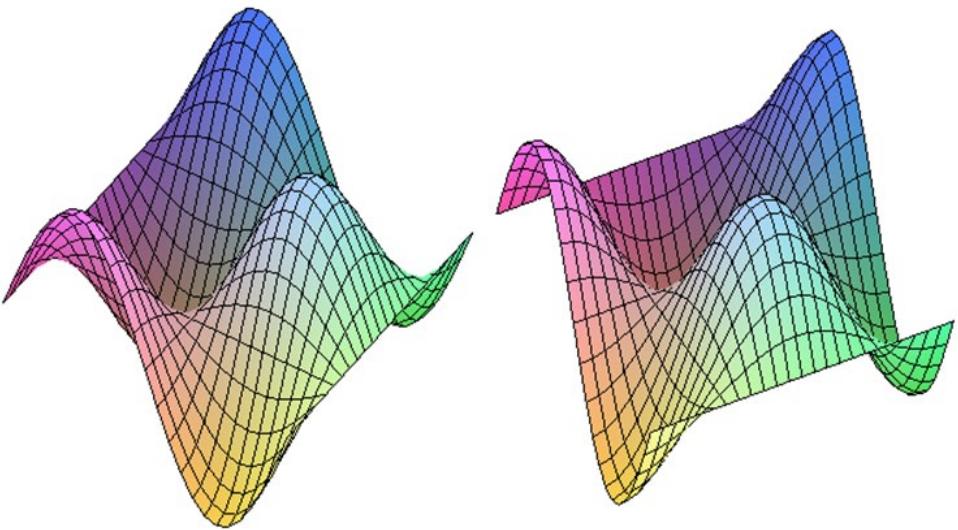


Height Fields

Function plot for a 2D scalar field



- 2D manifold: surface
- Plot styles
 - Wireframe
 - Hidden lines
 - Shaded surface



4.2 Isolines

Isolines

Visualization of 2D scalar fields

- Given a scalar function $f: \Omega \rightarrow \mathbb{R}$ and a scalar value $c \in \mathbb{R}$
- The isoline / isocontour or contour line consist of points $\{(x, y) \mid f(x, y) = c\}$

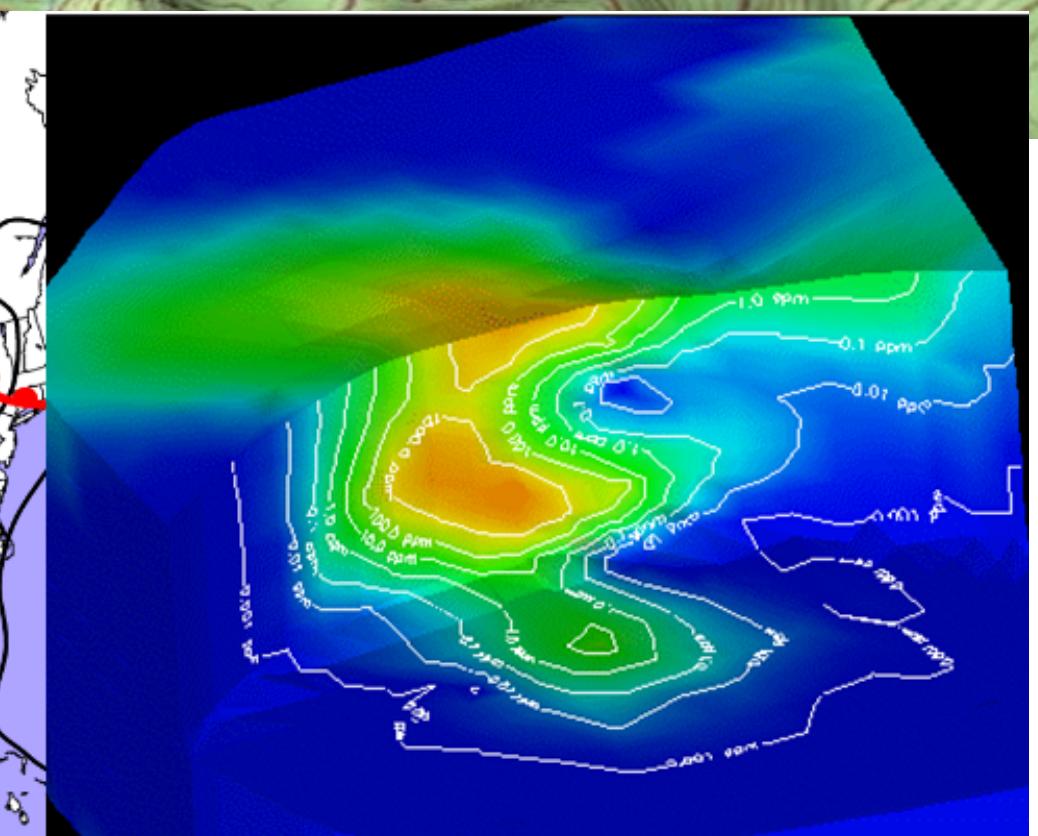
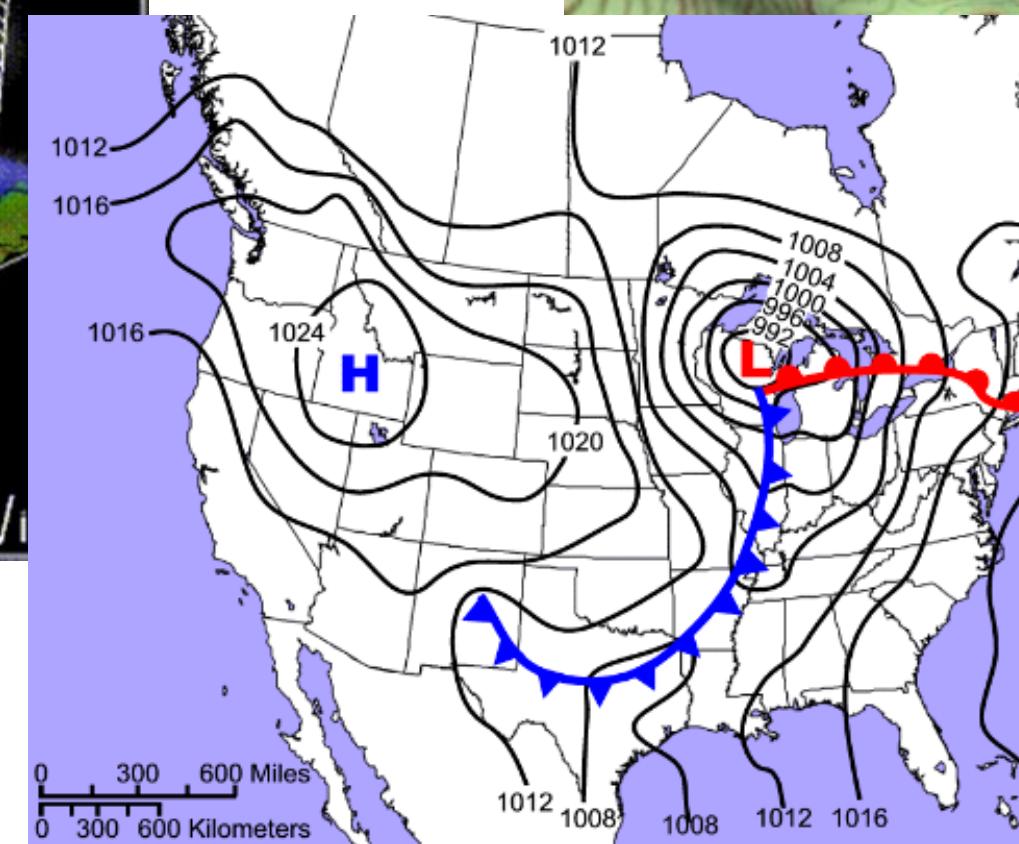
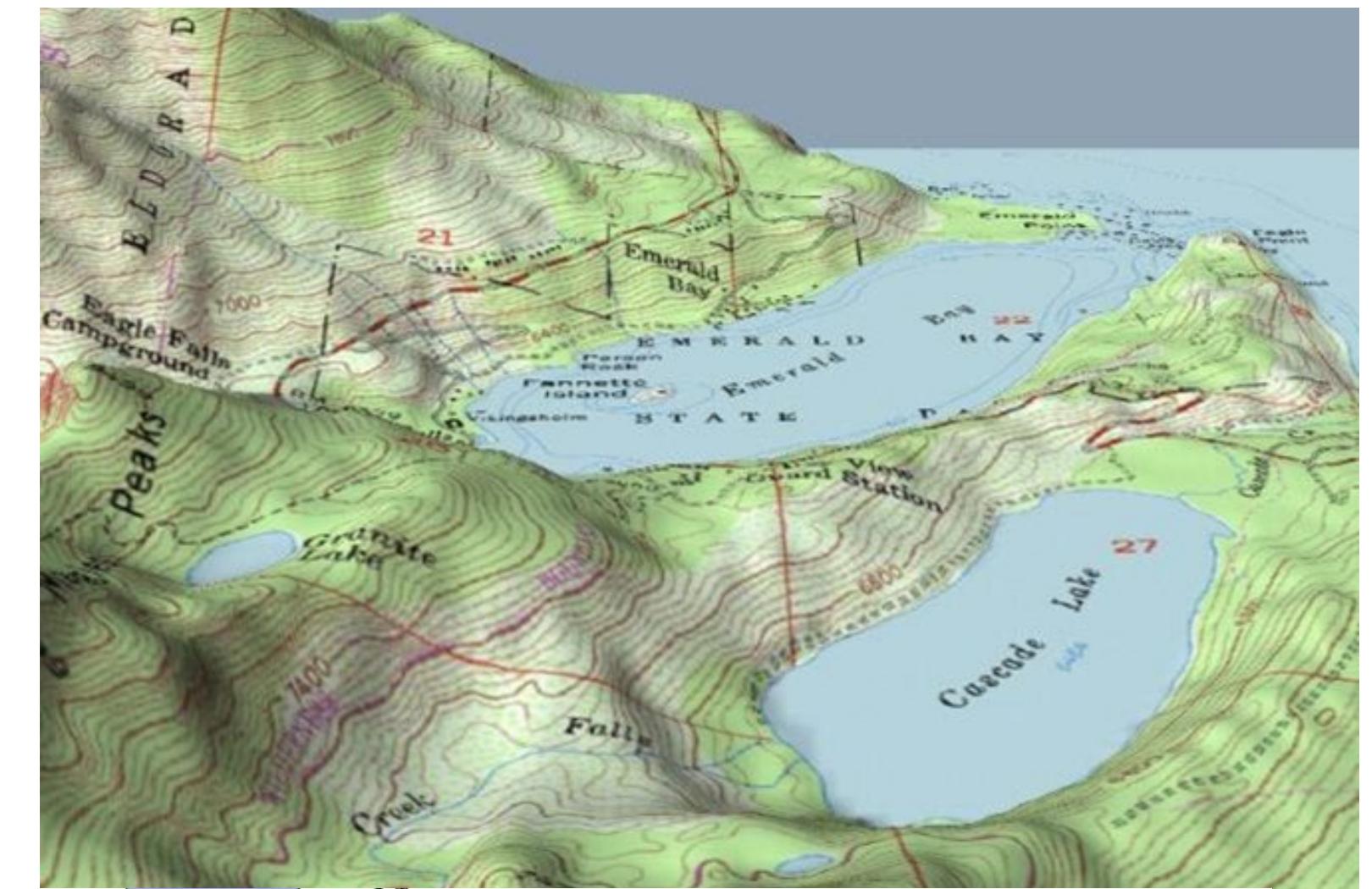
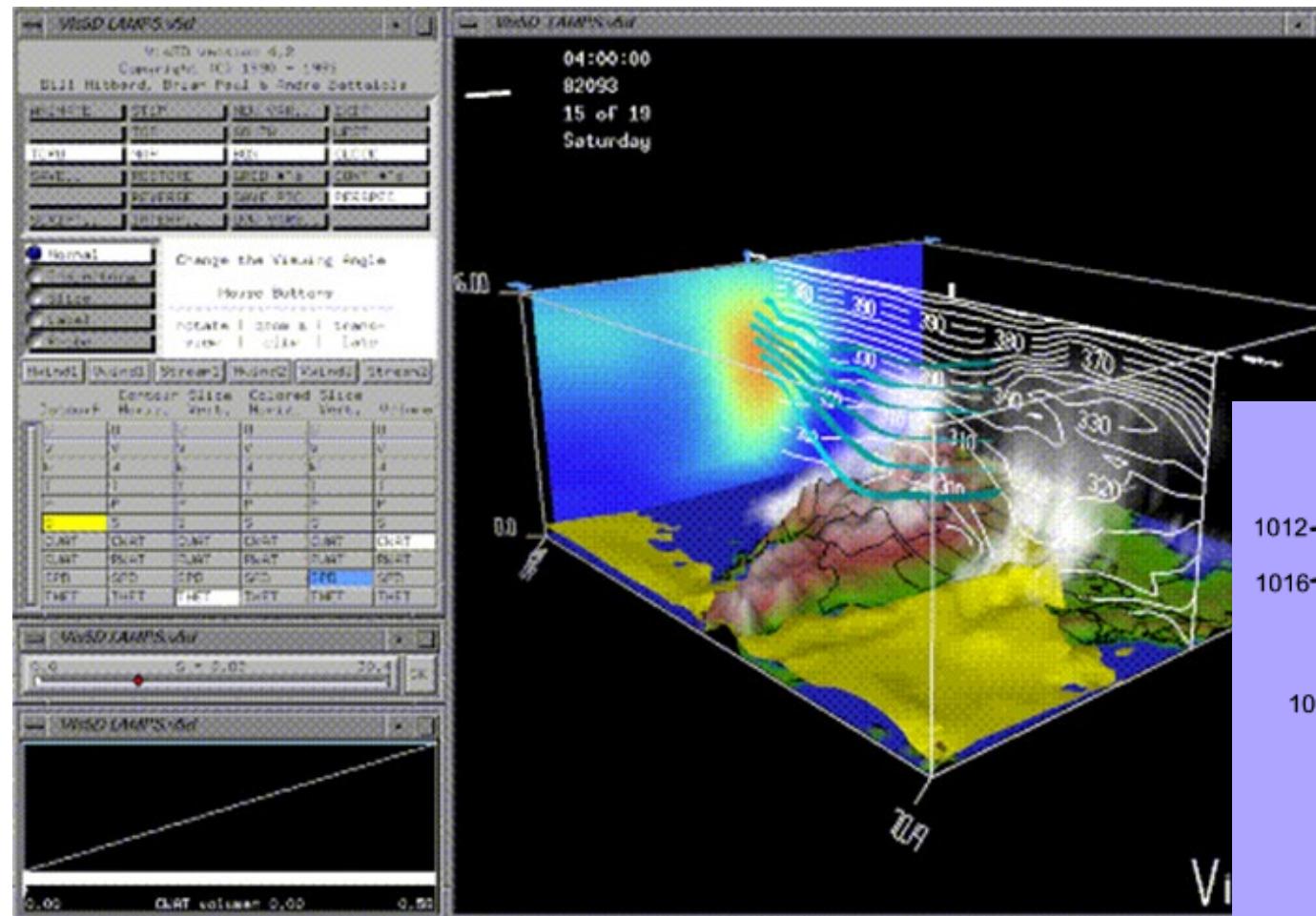
Punkt -> Konstante bei Funktion für Punkt

- If f is differentiable and $\text{grad}(f) \neq 0$, then isolines are curves
 - if $\text{grad}(f) = 0$ in some points, "isolines" may be single points or whole areas of the domain

- Remark
 - Isolines are closed curves, or they get in and out of the domain. They **do not terminate** in the domain!

Isolines

Examples



Isolines

Pixel by pixel contouring

- Straightforward approach
 - Scanning all pixels for equivalence with iso-value (this is an image space method)
- Input
 - $f : (1, \dots, x_{\max}) \times (1, \dots, y_{\max}) \rightarrow \mathbb{R}$
 - Iso-values I_1, \dots, I_n and iso-colors c_1, \dots, c_n
- Algorithm

```
for all (x,y) ∈ (1, …, xmax) × (1, …, ymax) do
    for all k ∈ { 1, …, n } do
        if |f(x,y) - I_k| < ε then
            draw(x, y, c_k)
```
- Problems
 - Isoline missed if gradient of f is too large, produces no additional data, only visualization

Isolines

Isolines on triangle meshes

- Cell search over all cells (isovalue f_c)

- No intersection

```
if ( $f_i > f_c$  or  $f_i < f_c$ )
for i = 1,2,3
```

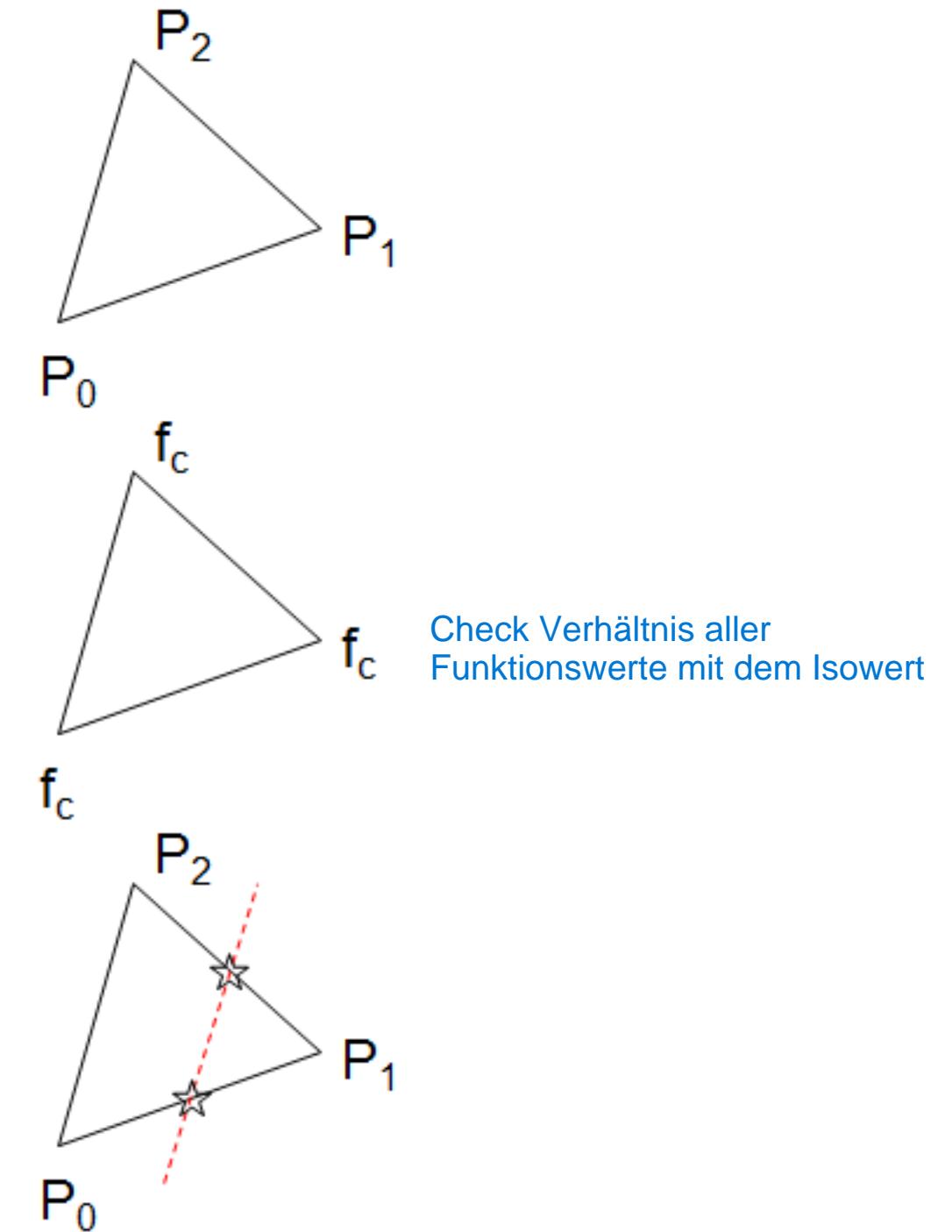
- Trivial reject

```
if ( $f_i == f_c$ )
for i = 1,2,3
```

- Normal case: find 2 intersection points

```
if ( $f_0 < f_c < f_1$  or  $f_1 < f_c < f_0$ )
then calculate intersection point
```

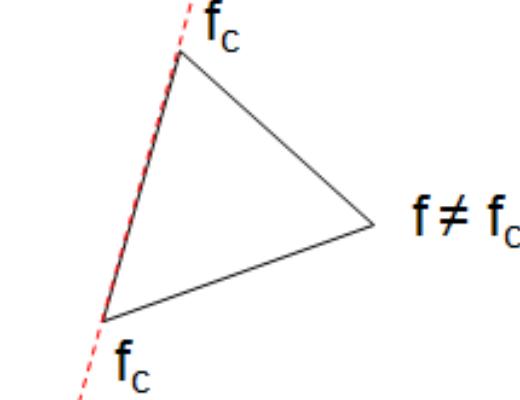
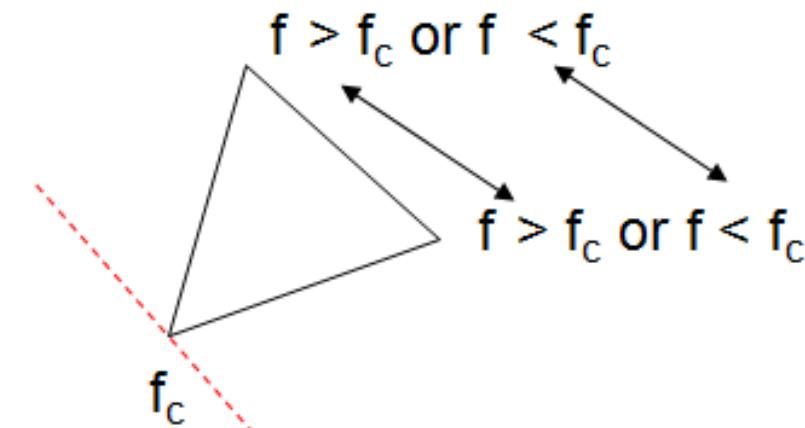
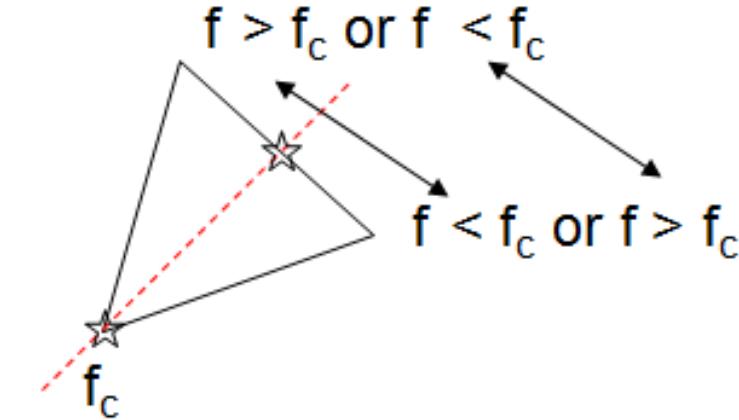
do the same for P_1P_2 and P_2P_0



Isolines

Special cases

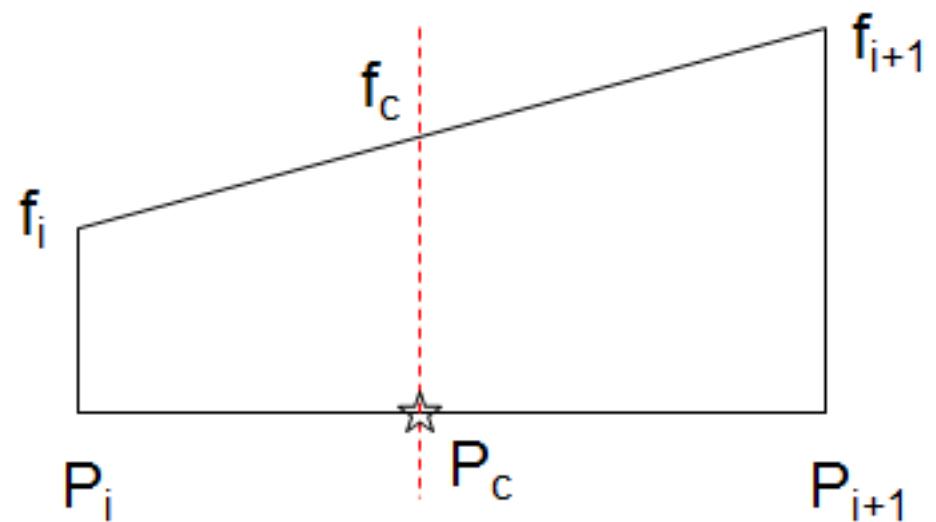
- Only one "normal" intersection point
- No "normal" intersections
 - Draw no lines
 - Consider in other triangle
 - Draw line along edge
 - Draw twice
 - For this and the neighboring triangle



Isolines

Calculate intersection points

- Interpolate point from data value

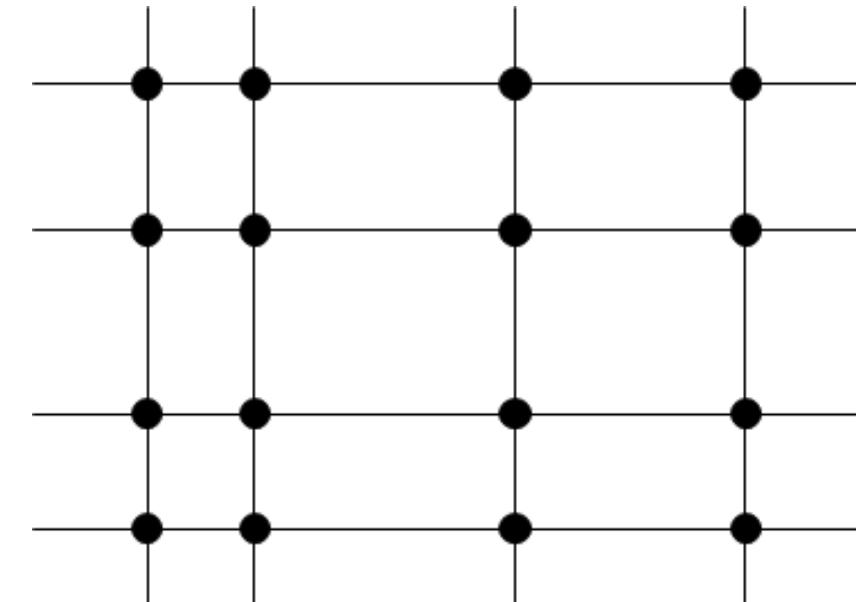


$$P_c = \frac{1}{f_{i+1} - f_i} [P_i(f_{i+1} - f_c) + P_{i+1}(f_c - f_i)]$$

Isolines

Rectilinear meshes

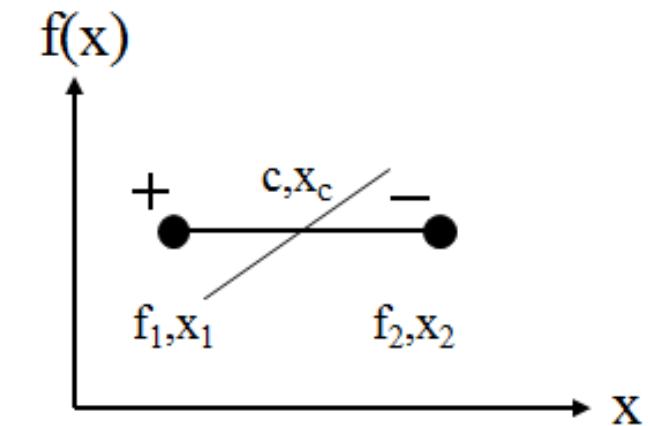
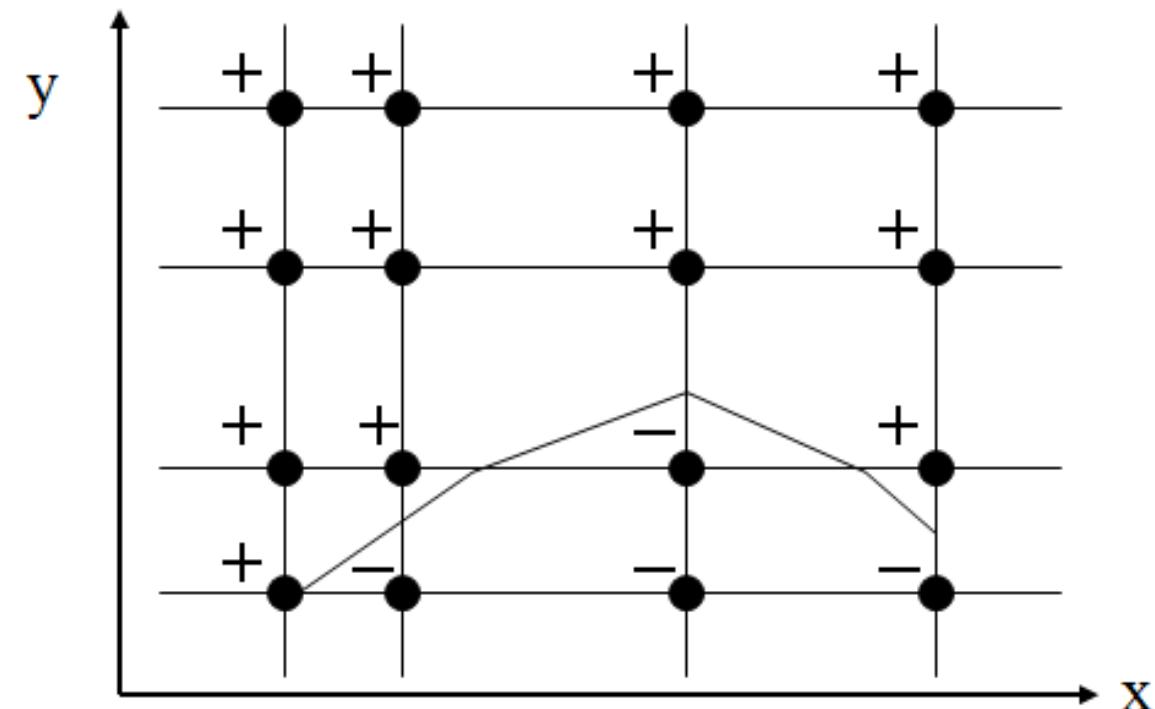
- Representation of scalar function on a rectilinear grid
 - Scalar values are given at each vertex $f \leftrightarrow f_{ij}$
 - Consider interpolation within cells (bilinear!)
 - *Isolines cannot be missed*
- Divide and conquer - "**Marching Squares**"
 - Consider cells *independently* of each other



Isolines

Marching Squares

- Which cells will be intersected?
 - Initially mark all vertices with "+" or "-", depending on the condition $f_{ij} \geq c$, $f_{ij} < c$
 - Isoline *does not pass through* cells which have the *same sign* at all four vertices
 - Only determine edges with different signs and *find intersection by linear interpolation*

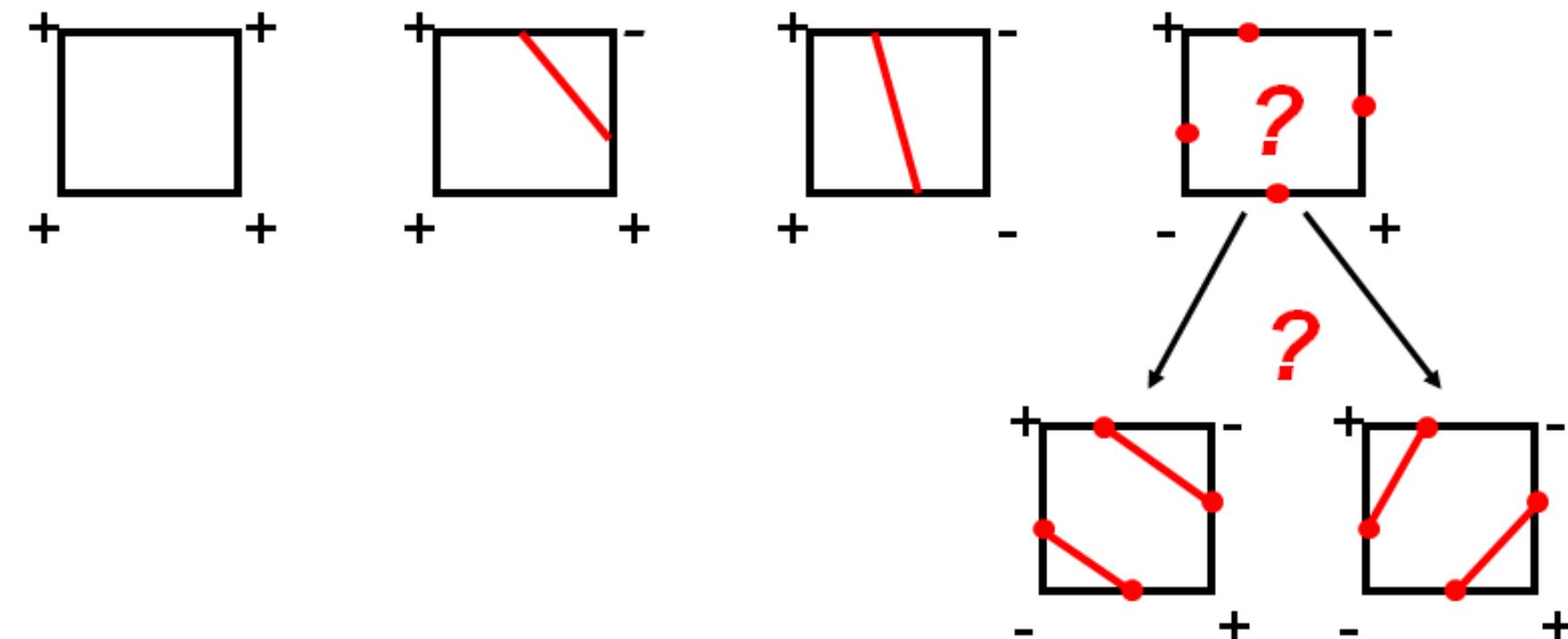


$$x_c = \frac{1}{f_2 - f_1} [(f_2 - c)x_1 + (c - f_1)x_2]$$

Isolines

Marching Squares

- There are $2^4 = 16$ different intersection possibilities (combinations of signs)
 - Using symmetries, only 4 different cases remain (rotation, reflection)
 - Compute intersections between isoline and cell edge (linear interpolation)



- Distinguish ambiguous case by a decider

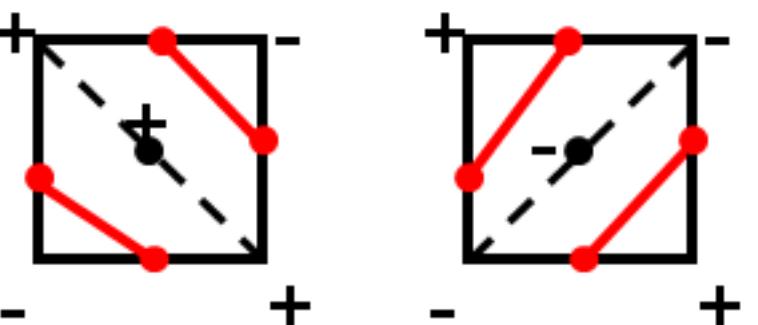
Isolines

Mid-point decider

- Interpolate the function value at the center

$$f_{\text{center}} = \frac{1}{4}(f_{i,j} + f_{i+1,j} + f_{i,j+1} + f_{i+1,j+1})$$

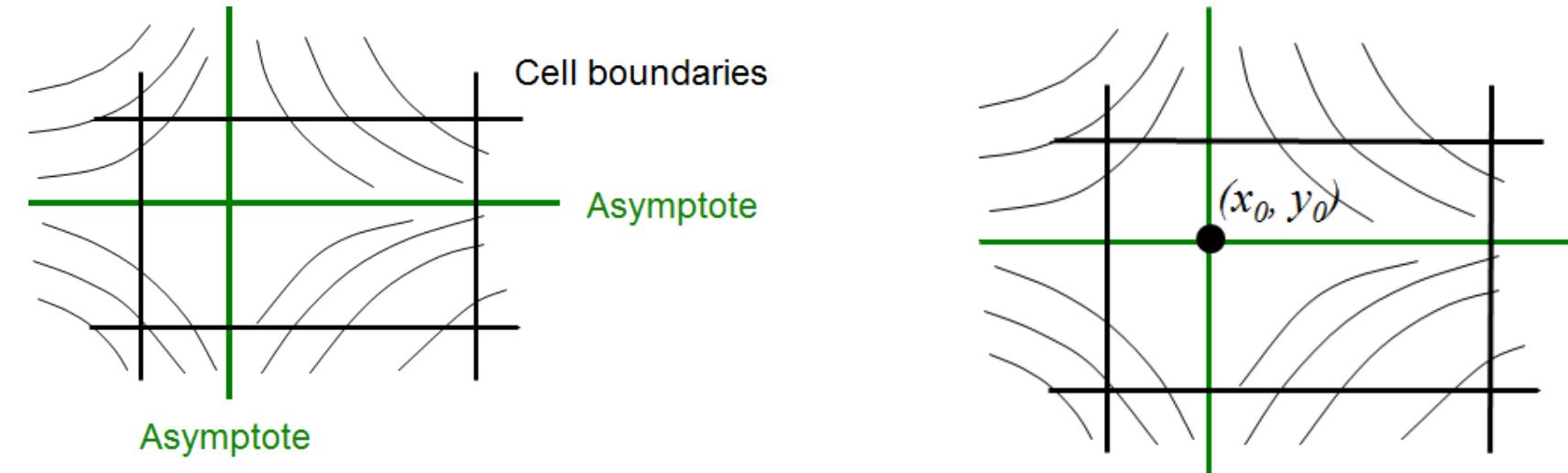
- If $f_{\text{center}} < c$ choose right case, otherwise left case



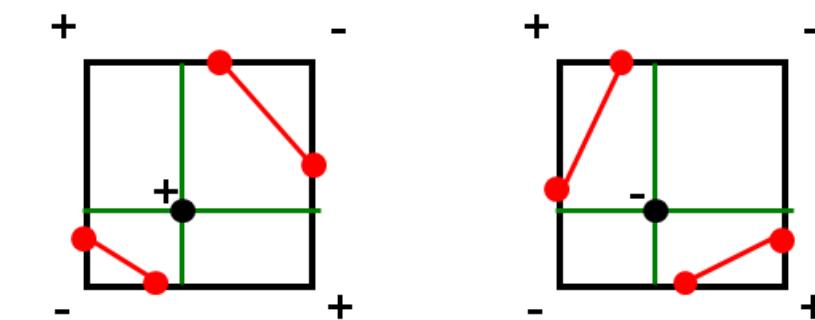
- This is not always the correct solution!

Isolines

Asymptotic decider



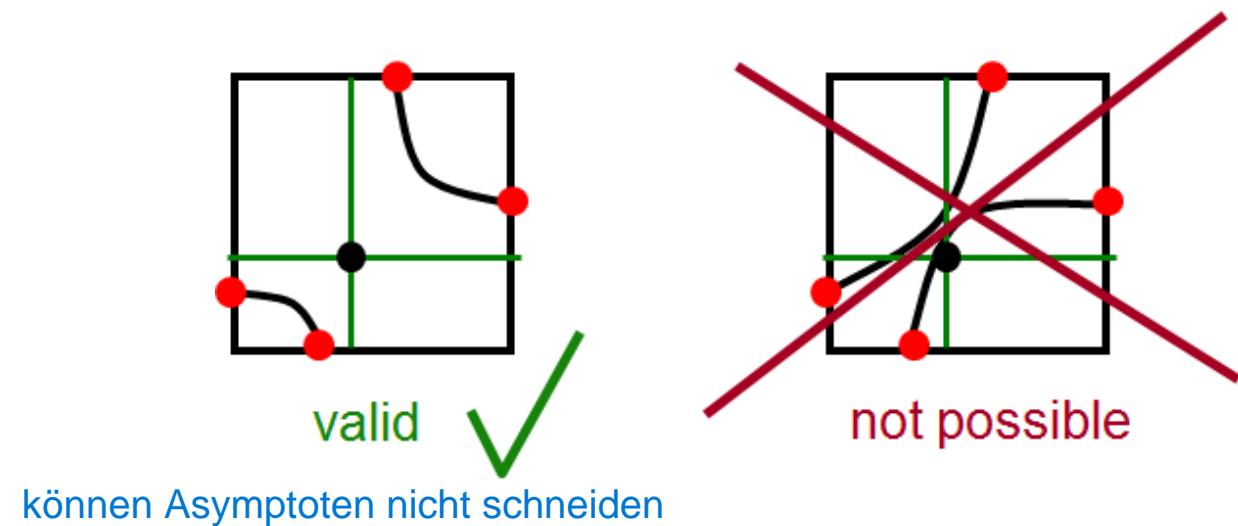
- We know that the true isolines within a cell are hyperbolas
 - Why?-Consider the bilinear interpolant within a cell
 - i.e. find stationary point where $\text{grad } f = 0$
- Interpolate function bilinearly $f(x, y) = f_{i,j}(1-x)(1-y) + f_{i+1,j}x(1-y) + f_{i,j+1}(1-x)y + f_{i+1,j+1}xy$
- Transform to normal form by comparison of coefficients $f(x, y) = \eta(x - x_0)(y - y_0) + \gamma$
 - γ is the function value in the intersection point of the asymptotes
 - If $\gamma \leq c$ choose right case, otherwise left case



Isolines

Explicit transformation of f can be avoided

- Consider order of intersection points either along x or y axis
 - Build pairs of first two and last two intersections

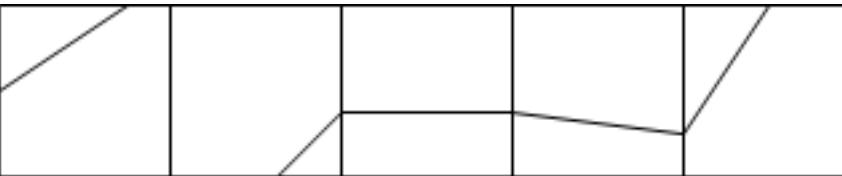


- Isolines cannot intersect asymptotes!

Isolines

Algorithms: Cell order approach for marching squares

- Apply marching squares to each individual cell

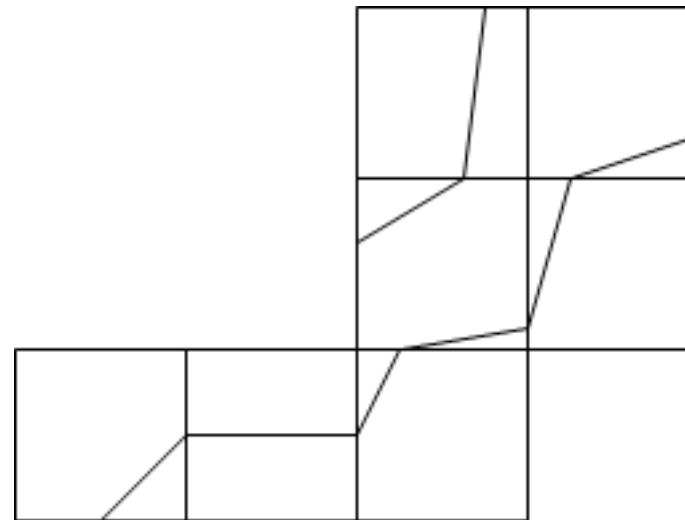


- Advantage
 - Simple approach
- Disadvantage
 - Process every vertex and every edge twice
 - Output is just a collection of pieces of isolines which have to be post-processed to get (closed) isoline

Isolines

Algorithms: Contour tracing approach

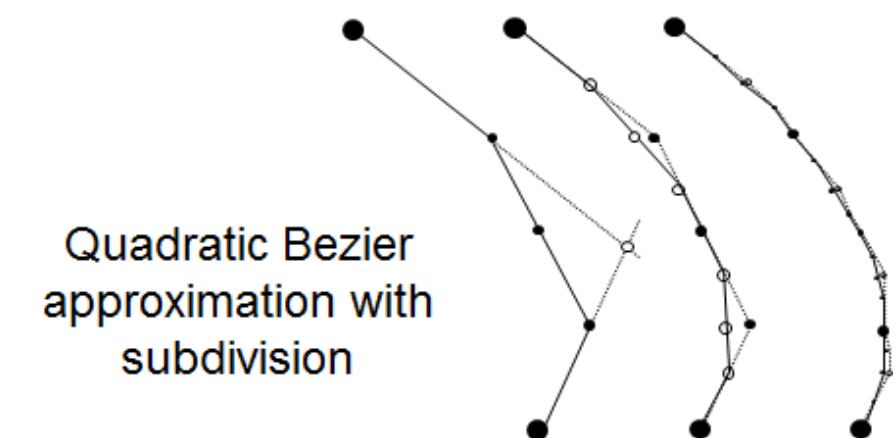
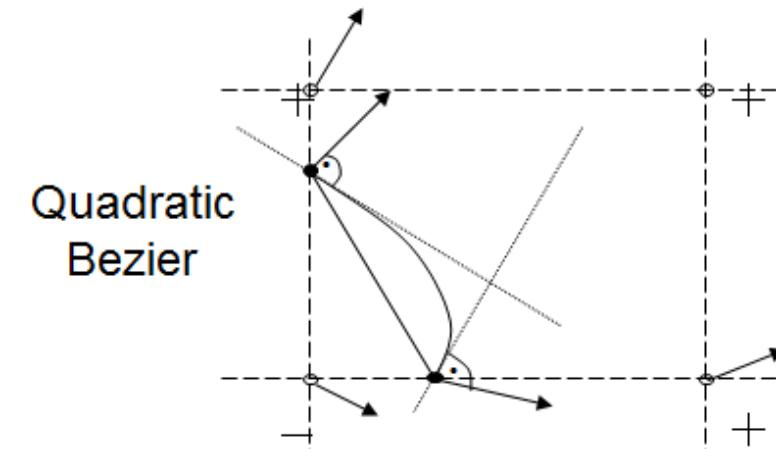
- Start at a seed point of the isoline
 - Move to neighboring cell (into which the isoline enters)
 - Trace isoline until bounds of domain are reached or isoline is closed
- Advantage
 - Polygons
 - Information about inclination
- Disadvantage
 - Marking of processed cells (list of started contours)
 - Problem: how to find seed points efficiently?
 - Remove marker from traversed cells (unless 4 intersections!)



Isolines

Smoothing isolines

- Evaluate gradient at vertices by central differences
- Estimate tangents at intersection points by linear interpolation - Gradient is perpendicular to isoline!
- Draw parabolic arc which is tangential to the estimated tangents at the intersections
 - Quadratic Bezier curve
(2-3 subdivisions per step is sufficient!)



Isolines

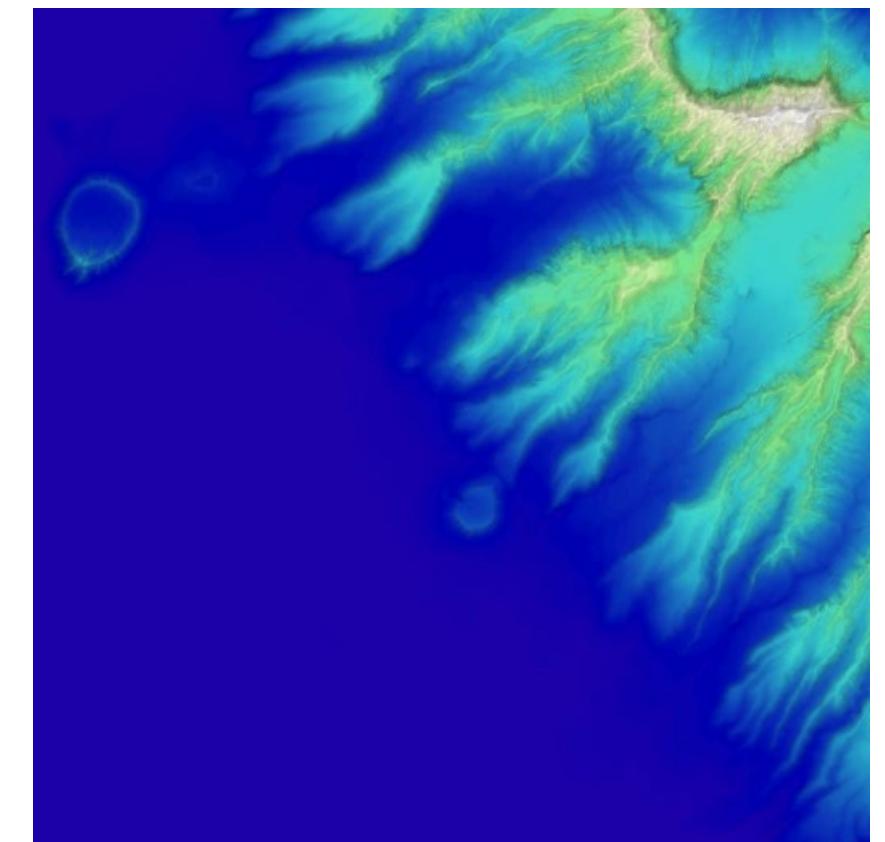
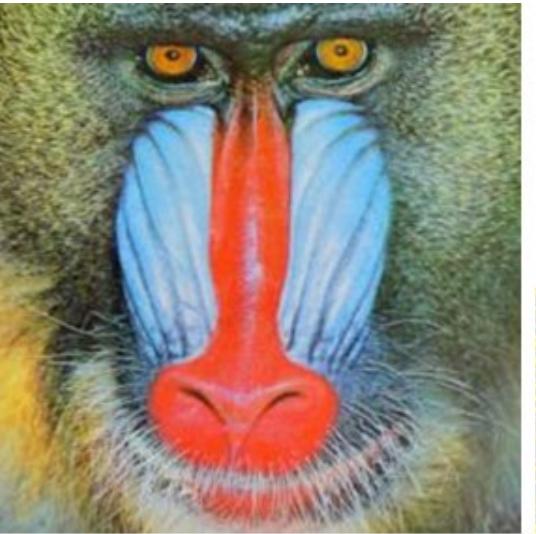
Summary

- Smoothing by interpolation falsifies the contour
- Multiple isolines
 - Line type (dashed, dotted, ...), annotation, color
- Advantages
 - Line plots in black/white possible
 - Annotation within image
 - Shows gradient within image
 - Simple extension for complex grids
- Disadvantages
 - Max/min not directly visible
 - Strongly depends on choice of isovalue
 - No use of color and rasterization features

4.3 Color Coding

Color Coding

- Questions
 - What kind of data can be color-coded?
 - What kind of information can be efficiently visualized?
- Areas of application
 - Provide information coding
 - Designate / emphasize target in crowded display
 - Provide sense of realism (or virtual realism)
 - Warning signals or signify low probability events
 - Group, categorize and chunk information
 - Convey emotional content
 - Provide an aesthetically pleasing display
- Possible problems
 - Distract the user when inadequately used
 - Dependent on viewing and stimulus conditions
 - Ineffective for color deficient individuals
 - Results in information overload
 - Unintentionally conflict with cultural conventions
 - Cause unintended visual effects and discomfort



Reds and Blues are on
opposite ends of the

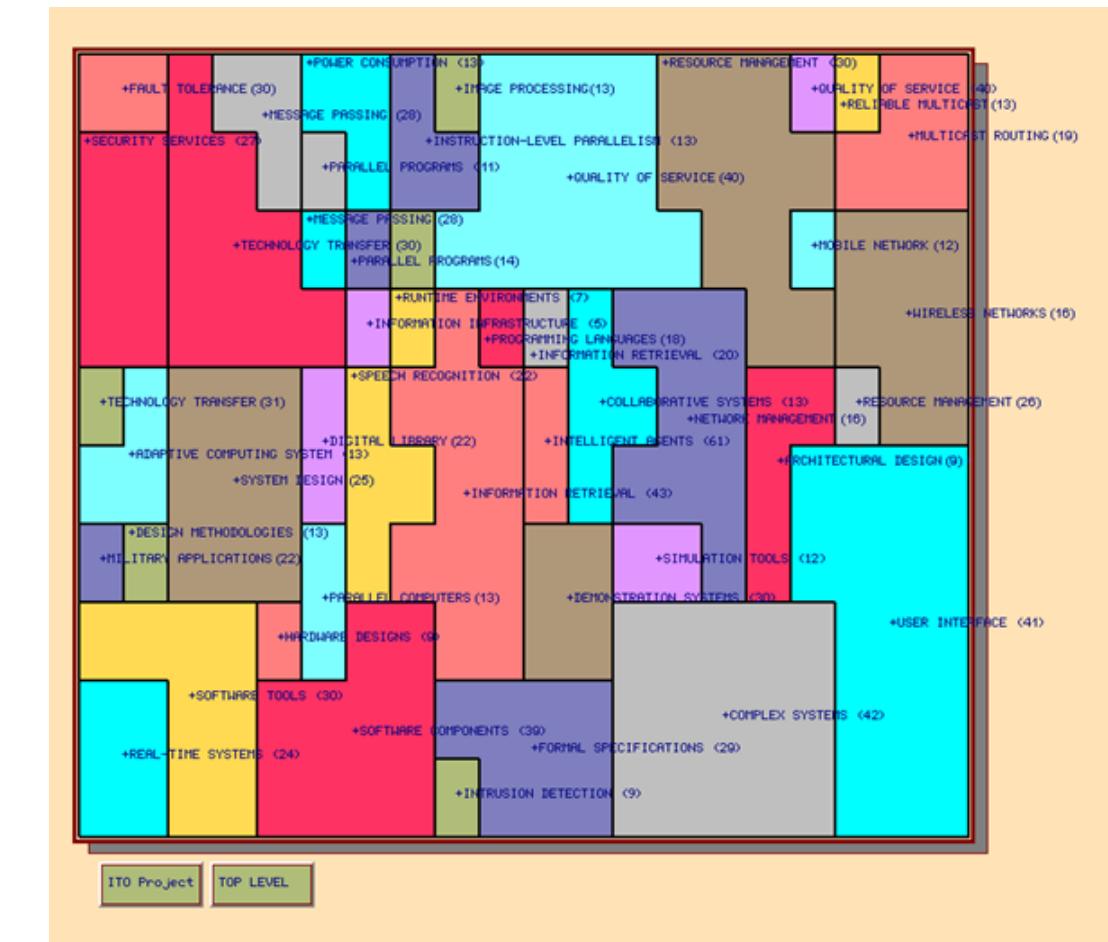
Chromatic aberration =
different refraction depending on wavelength
(red "nearer", blue "farther" - different foci!)

focus on both.

Color Coding

Nominal data

- Colors need to be distinguished
 - Localization of data
 - Around 8 different basis colors

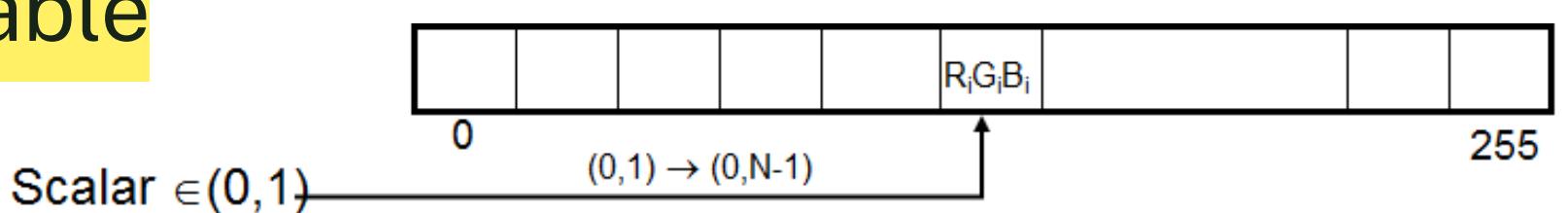


co-citation analysis

Color Coding

Ordinal and interval data

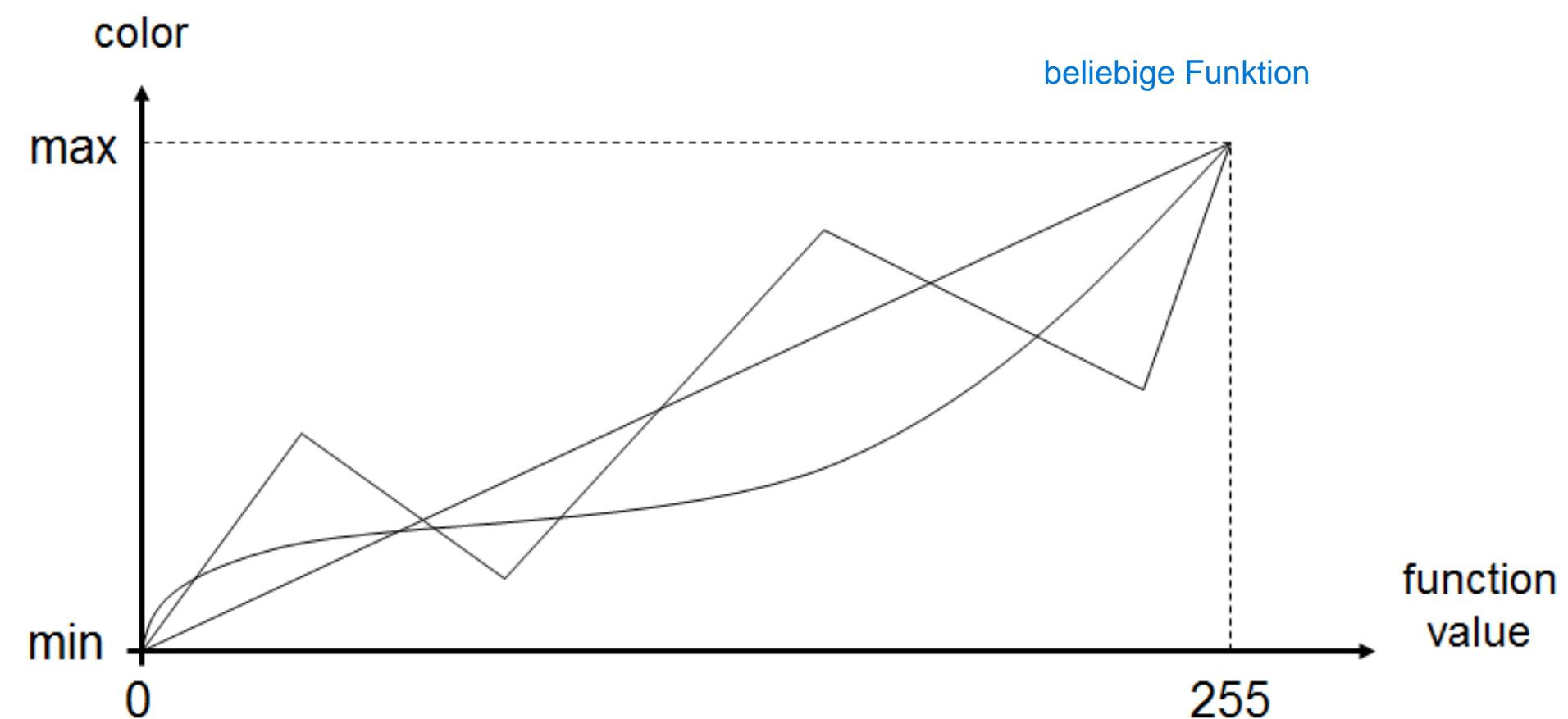
- Represent ordering of data by ordering of colors
 - Psychological aspects
 - $x_1 < x_2 < \dots < x_n \rightarrow E(c_1) < E(c_2) < \dots < E(c_n)$
- Color coding for scalar data
 - Assign to each scalar value a different color value
 - Assignment via transfer function T : scalar value \rightarrow color value (RGB or HSV)
 - Code color values into a color lookup table



Color Coding

Assignment via transfer function T

- T : scalar value \rightarrow color value (RGB, HSV, ...)



Color Coding

Linear transfer function for color coding

- Specify color for f_{min} and f_{max}
 - $(R_{min}, G_{min}, B_{min})$ and $(R_{max}, G_{max}, B_{max})$
 - Linearly interpolate between them

$$f \mapsto \frac{f - f_{min}}{f_{max} - f_{min}} (R_{min}, G_{min}, B_{min}) + \frac{f_{max} - f}{f_{max} - f_{min}} (R_{max}, G_{max}, B_{max})$$

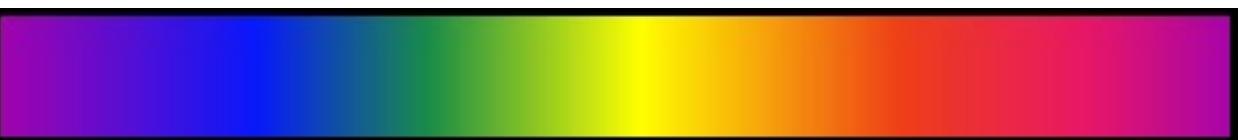
lineare Interpolation

- Different color spaces
 - Lead to different interpolation functions
 - In order to visualize (enhance / suppress) specific details, non-linear color lookup tables are needed

Color Coding

Examples of different color tables

- Gray scale
 - Intuitive ordering
- Rainbow
 - Less intuitive
- Temperature



Color Coding

Bivariate and trivariate color tables

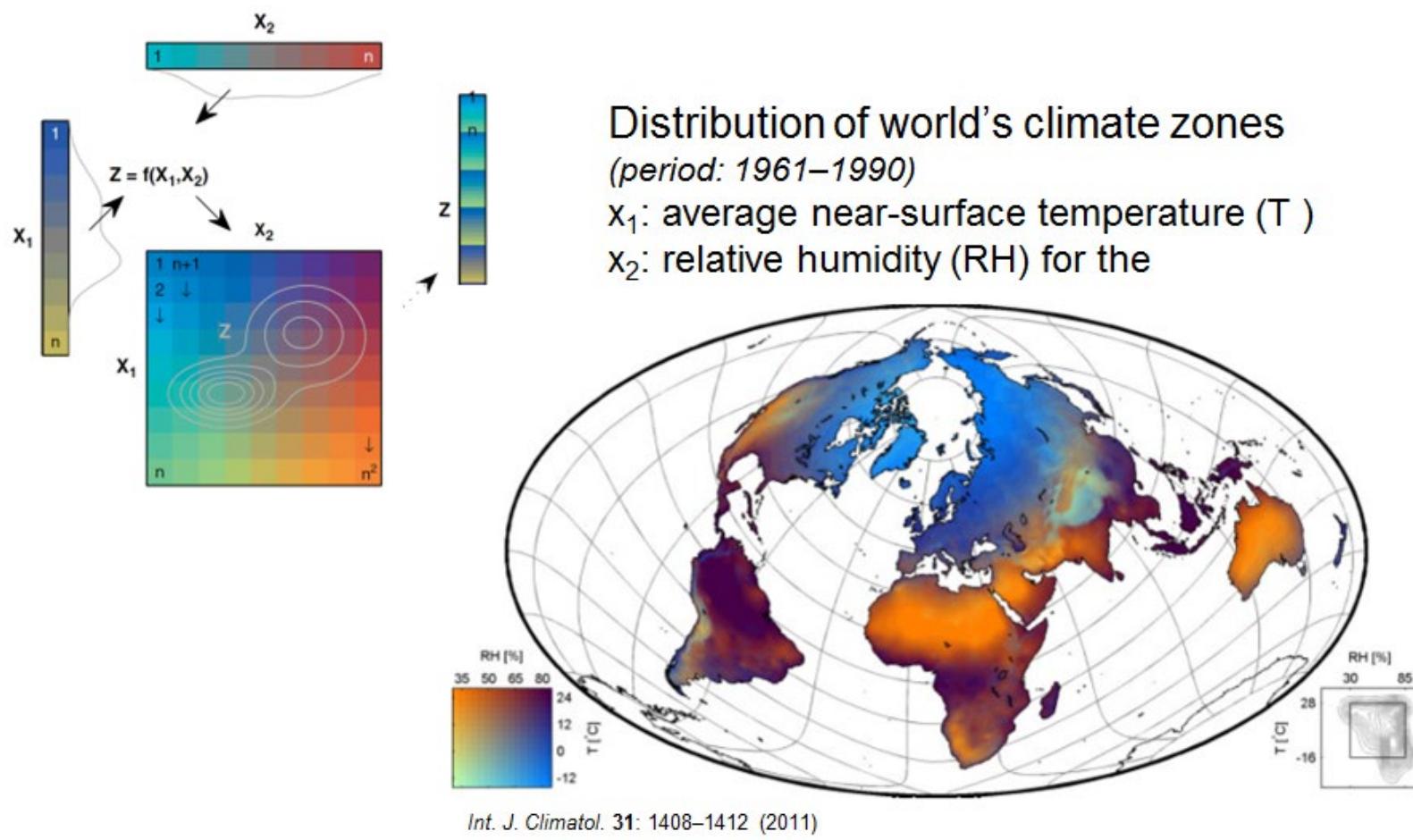
2-dim

3-dim

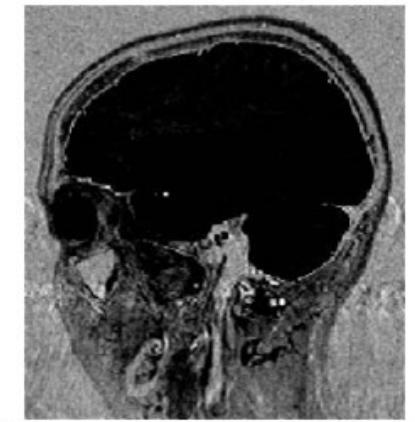
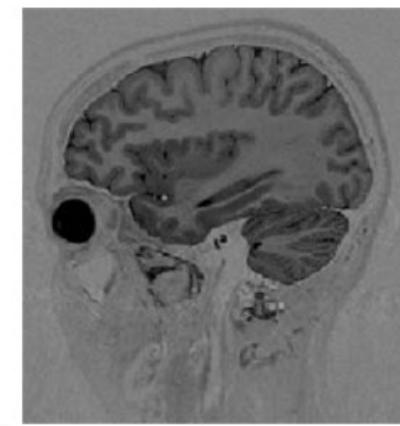
- Not very useful as there is no intuitive ordering
 - Colors hard to distinguish
- Many more color tables
 - For specific applications
 - Design of good color tables depends on psychological / perceptual issues
- Important requirement
 - Often, interactive specification of transfer function is needed to extract important features (which leads to nonlinear transfer functions), e.g. medical data

Color Coding

Examples



Magnetic
Resonance
Imaging
(MRI)



Computed
Tomography
(CT)



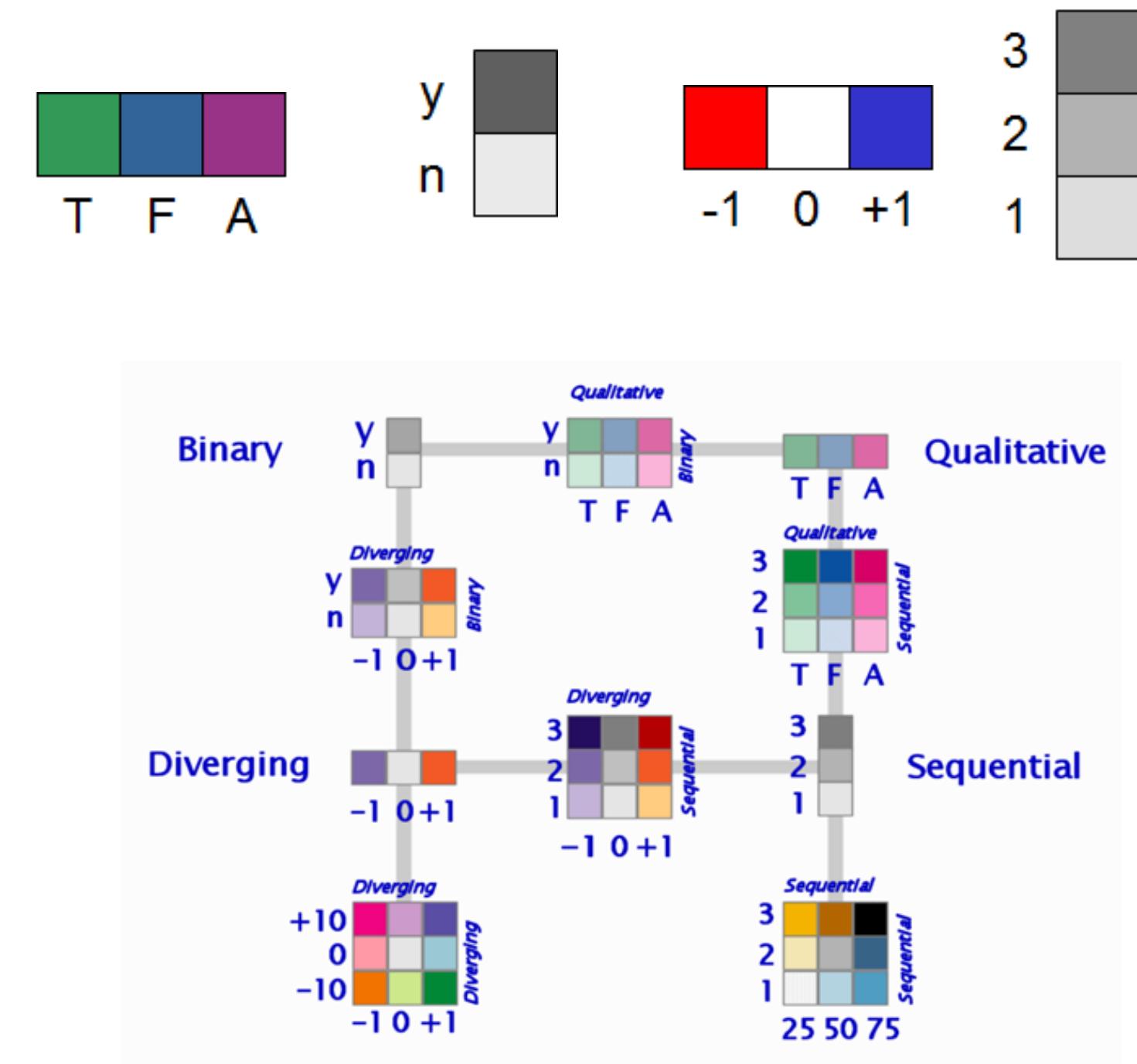
Linear mapping of
all data values

Special transfer function
for brain tissue

Special transfer function
for bone structure

Color Coding

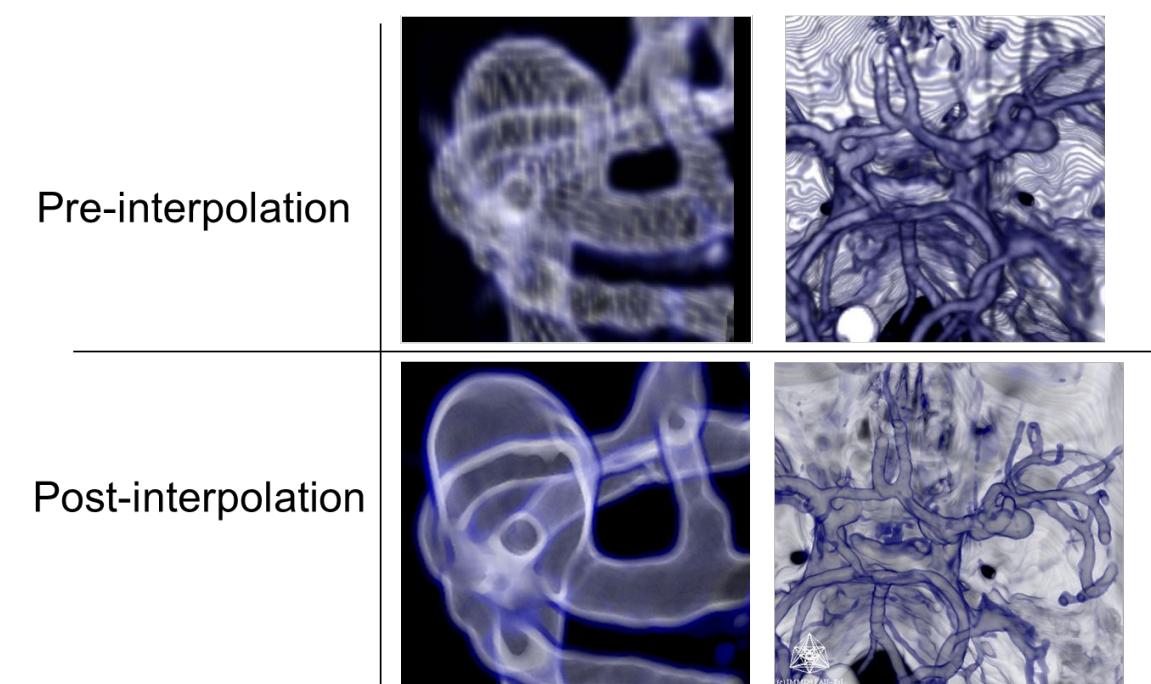
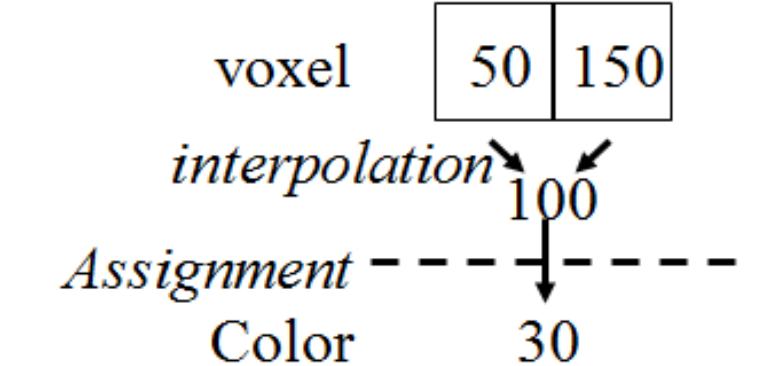
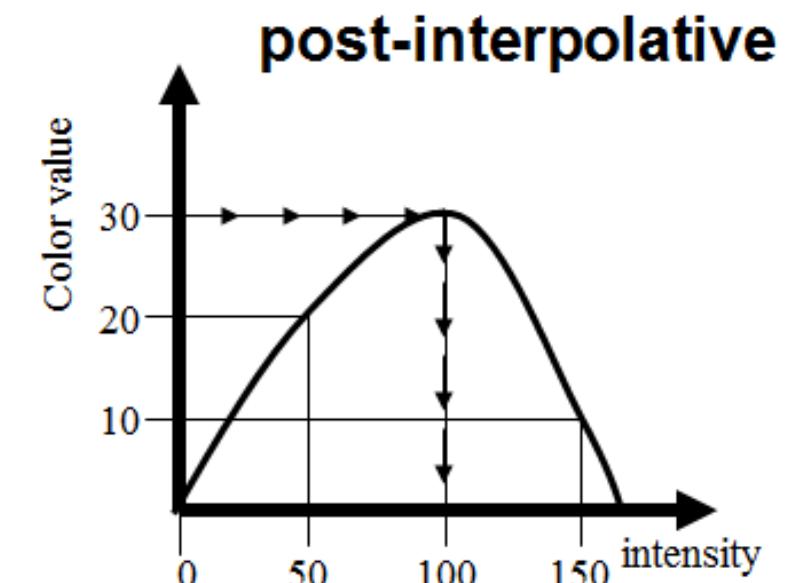
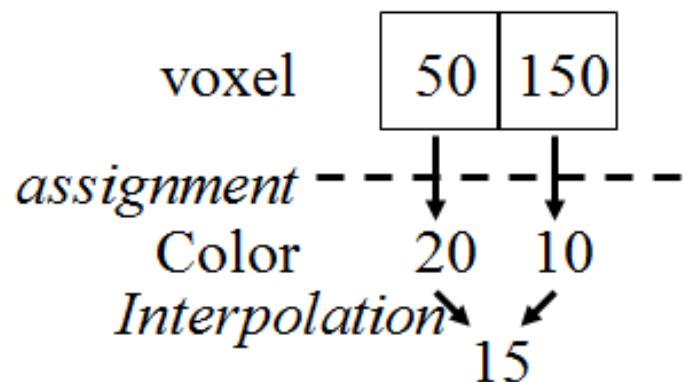
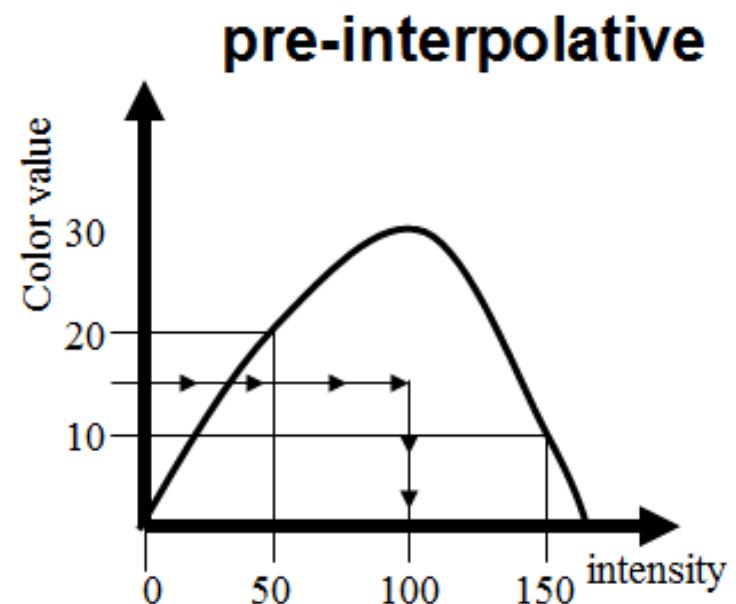
- Color guidelines
 - [C.A. Brewer, 1994, "Color Use Guidelines for Mapping and Visualization", Chapter 7 (pp. 123-147) in Visualization in Modern Cartography]
- 4 different types of color schemes
 - Qualitative - data of different categories
 - Represented by differences in hue
 - E.g. blue - rivers, brown - mountains, red - road
 - Binary - only two categories
 - Use difference in hue or lightness
 - Diverging - data showing different trends
 - Use colors with very different hues in extremes
 - E.g. red - blue, red - green (yellow - blue for color blind)
 - Sequential - ordered data in sequential form
 - Use variations in lightness or saturation of given hue
- Bivariate color schemes. www.infovis.net (2006, No. 184)



Color Coding

Pre-shading vs. post-shading

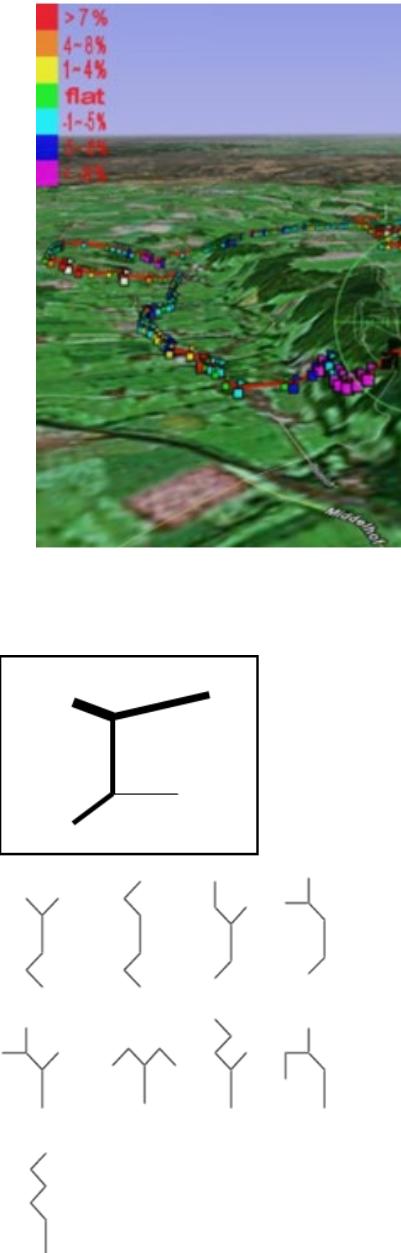
- Pre-shading (pre-interpolation)
 - Assign color values to original function values (i.e. vertices of a cell)
 - Interpolate between color values (within a cell)
- Post-shading (post-interpolation)
 - Interpolate between scalar values (within a cell)
 - Assign color values to interpolated scalar values



4.4 Glyphs, Icons and others...

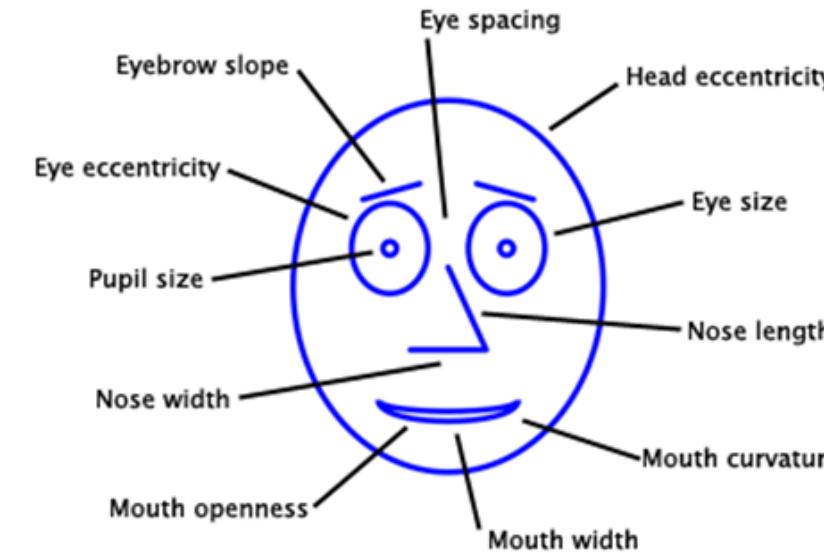
Glyphs and Icons

- Features
 - Should be easy to distinguish and combine
- Icons
 - Should be separated from each other
- Mainly used for multivariate data
 - More than one data value at each coordinate
- Stick-figure icon [Picket & Grinstein 88]
 - 2D figure with 4 limbs
 - Coding of data via
 - Length, thickness, angle with vertical axis
 - 12 attributes
 - Exploits human capability to recognize patterns/textures

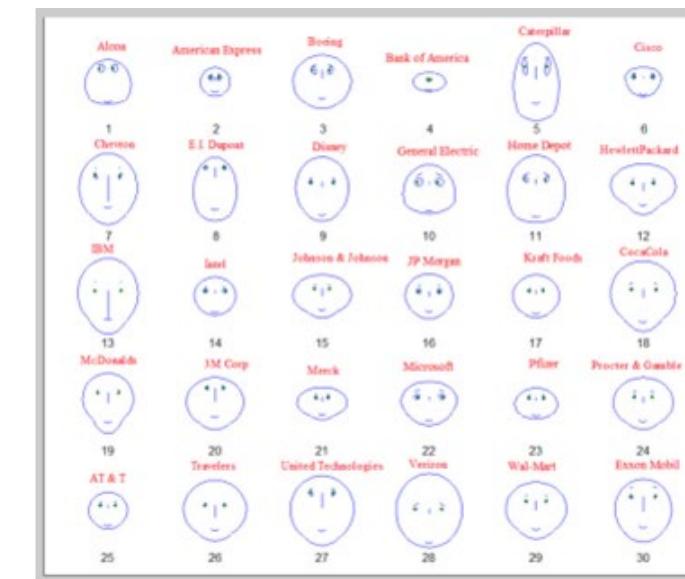
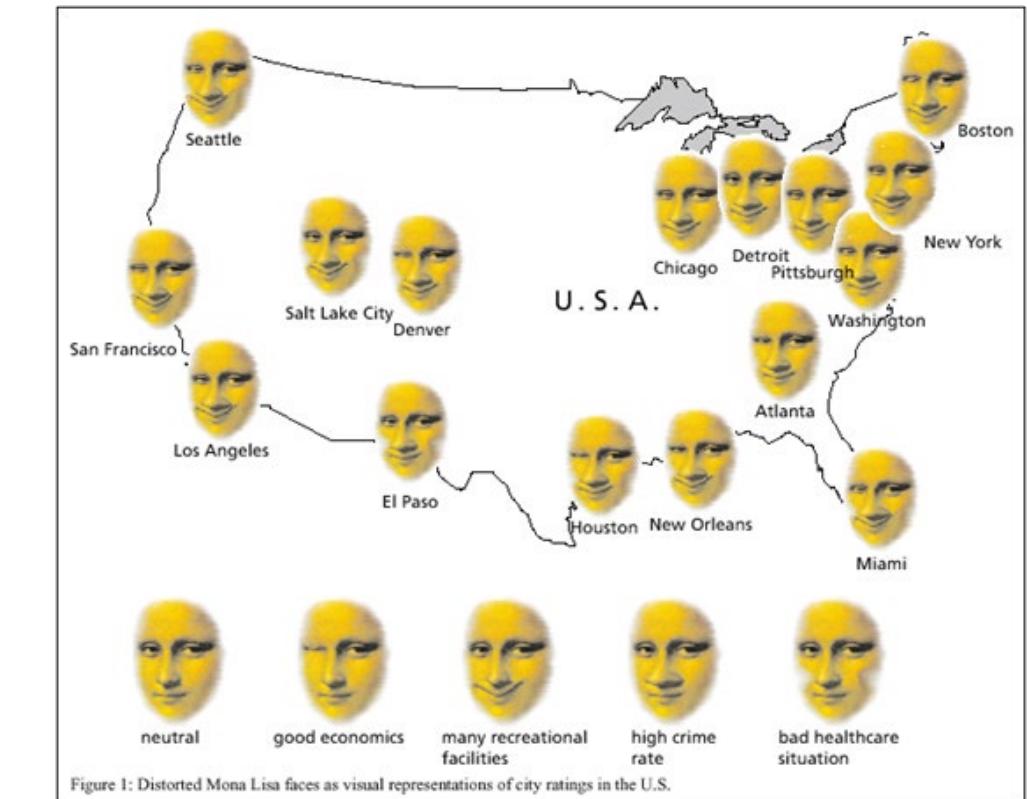


Glyphs and Icons

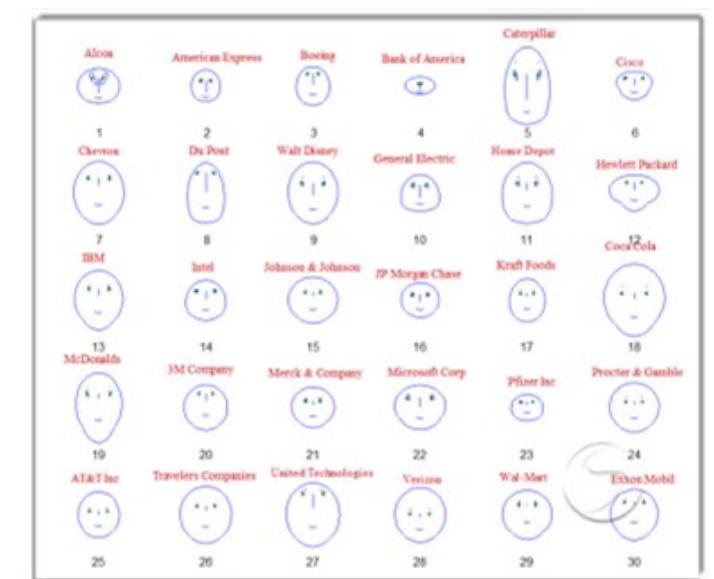
- Chernoff faces/icons
 [H. Chernoff (1973). The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, 68 :361-368]
 - Each facial feature represents one variable
 - Useful for qualitative and multivariate data
 - Human ability to distinguish small features in faces
 - Useful for sparse data representations
 - Possible assignment in decreasing order of importance
 Area of face, shape of face, length of nose, location of mouth
 - Curve of smile, width of mouth, location, separation, angle, shape and width of eyes, location of pupil
 - Location, angle and width of eyebrows
 - Additional variables
 - Could be encoded by making faces asymmetric



Codierung von Daten
in einem Icon (=Gesicht)



31 January 2011



15 December 2010

A	Column	Attribute	B	ValuEngine Model Variable
1	Size of face	Valuation (+/- %)		
2	Forehead/jaw relative arc length	Average Volume		
3	Shape of forehead	Last 12-m Return %		
4	Shape of jaw	Sharpe Ratio		
5	Width between eyes	5-Year Return %		
6	Vertical position of eyes	1-Month Forecast Return %		
7	Height of eyes	Beta		
8	Width of eyes (this also affects eyebrow width)	Volatility %		
9	Angle of eyes (this also affects eyebrow angle)	Expected EPS		
10	Vertical position of eyebrows	Market Capitalization		
11	Width of eyebrows (relative to eyes)	P/E Ratio		
12	Angle of eyebrows (relative to eyes)	M/B Ratio		
13	Direction of pupils	P/S Ratio		
14	Length of nose	Actual EPS		
15	Vertical position of mouth	Forecast EPS		

5. Vector Fields

Contents

- Introduction
- Arrows and Glyphs
- Characteristic lines in vector fields
- Particle tracing
- Line integral convolution

Introduction

- Problem: visualize a function like
 - $F: \Omega \rightarrow \mathbb{R}^2$, with F given only at certain vertices

$$F \leftrightarrow F_{ij} = \begin{pmatrix} F_{ij}^x \\ F_{ij}^y \end{pmatrix}$$

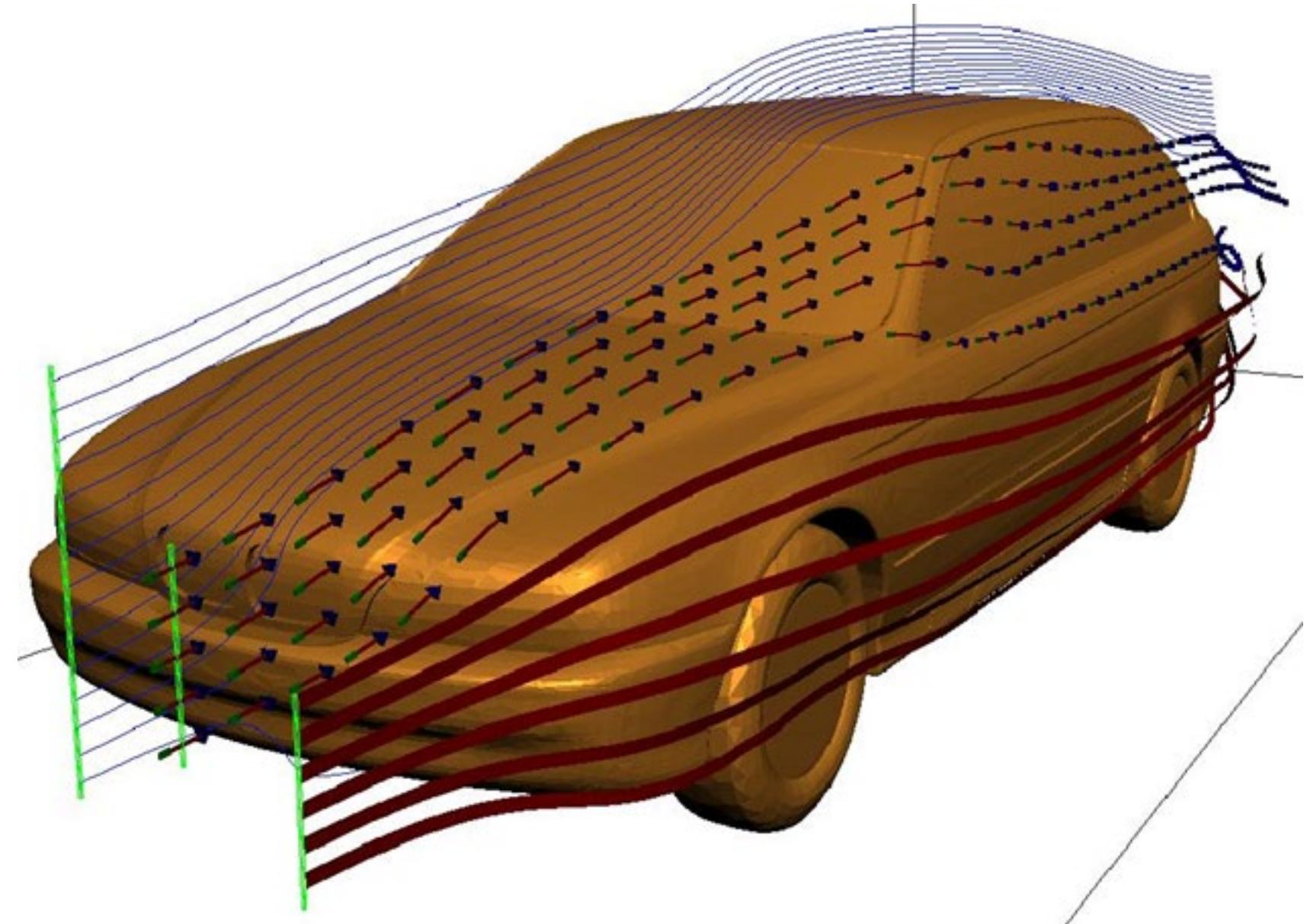
- Ideas
 - Visualize the two scalar fields F_x and F_y
 - But: components are usually not independent \rightarrow no insight!
 - If the data represents some form of velocity
 - Direction of moving particles
 - Visualize the "flow"

Introduction

- Main application area: **flow visualization**
 - Gas (car industry, airplanes, ...)
 - Fluids (water, reactors, blood vessels, ...)
- Important entities
 - Geometric boundary conditions
 - Velocity (flow) field $v(x, t)$
 - Pressure p , temperature T , density ρ
 - Vorticity: $\nabla \times \vec{v}$
 - Amount of circulation (total angular rate of rotation)
 - Navier-Stokes equations
 - CFD (Computational Fluid Dynamics)

Introduction

Flow visualization based on CFD data



Introduction

Classification of flow visualization

- Dimension (2D or 3D)
- Time-dependency
 - stationary (steady) vs. in-stationary (unsteady)
- Grid type
- Compressible vs. incompressible fluids
- In most cases of flow visualization
 - Numerical methods required

5.1 Arrows and Glyphs

Arrows and Glyphs

Glyphen => "Pfeile in 3D"

- Natural "vector" display
 - Arrows
 - Visualizes local features of a vector field
 - Vector itself
 - Vorticity ($\nabla \times \vec{v}$)
 - External data: pressure, temperature, etc.
- Important elements of a vector
 - Direction
 - Magnitude
 - Not: individual components

Arrows and Glyphs

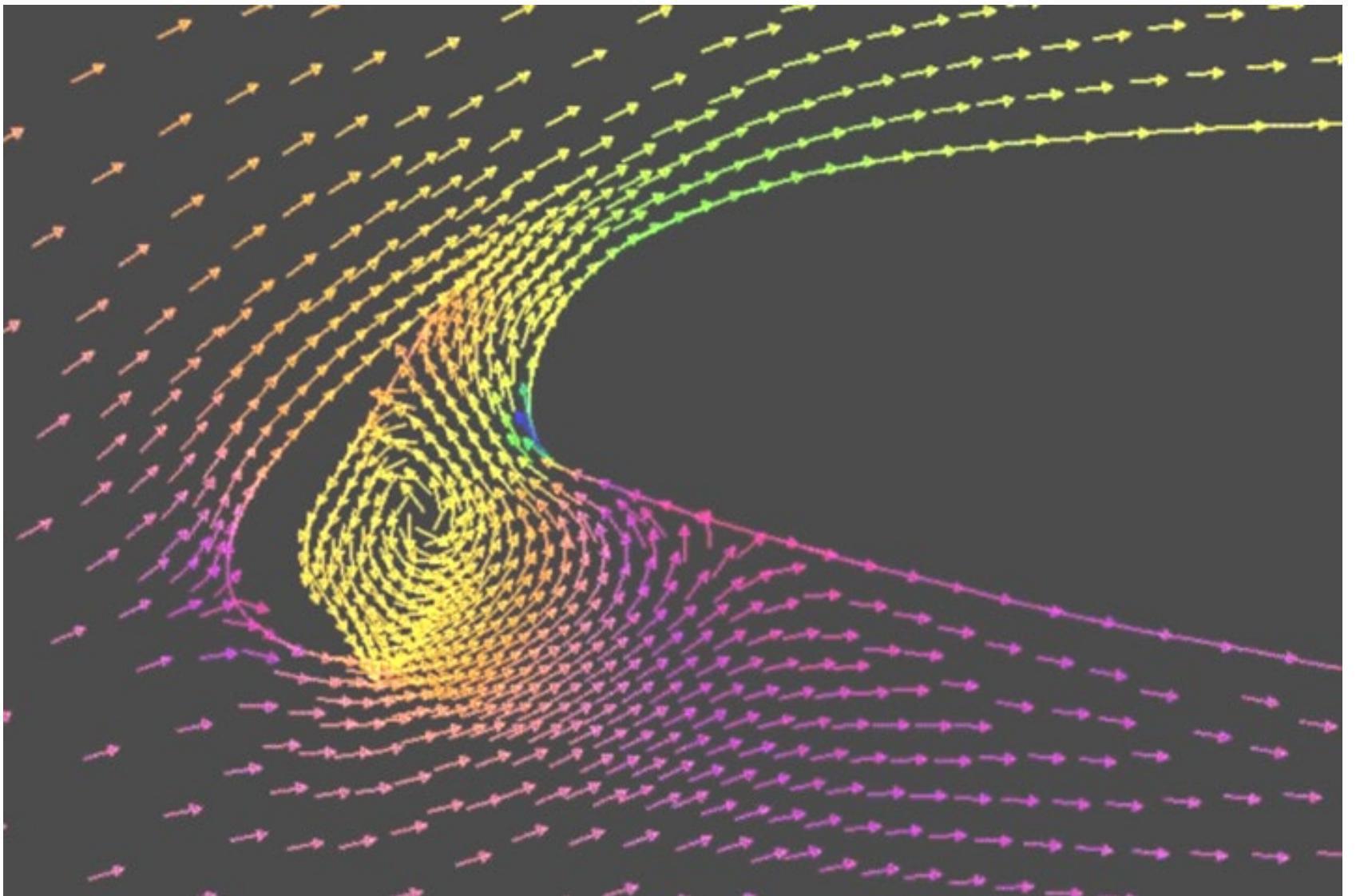
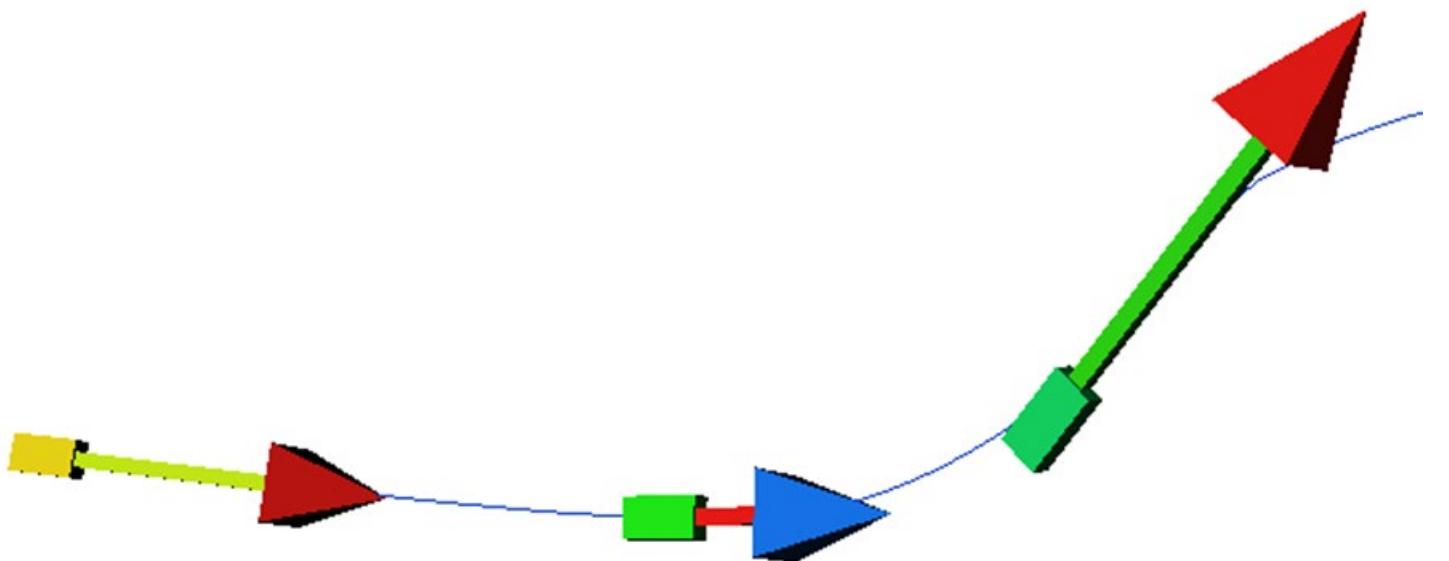
Arrows

- Place arrow geometry at each (nth) grid point
 - Geometry: 
- Length
 - Represents magnitude of the vector field
- Direction
 - Coincides with direction of vector field at that vertex
- In case of very fast changing velocity
 - Results are typically not good
 - Better use arrows of constant length
 - Indicate magnitude separately (e.g. color coding)

ohm

Arrows and Glyphs

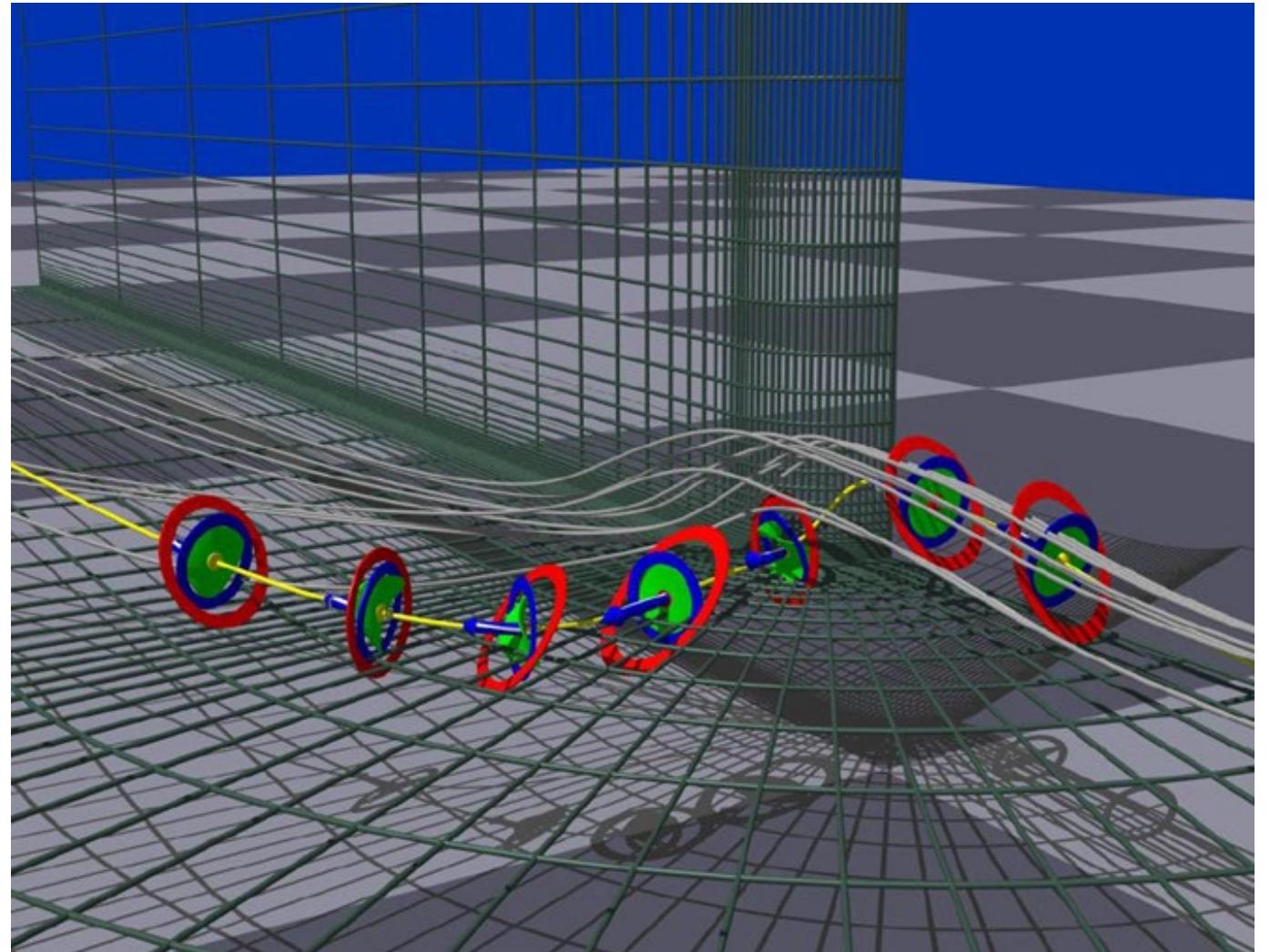
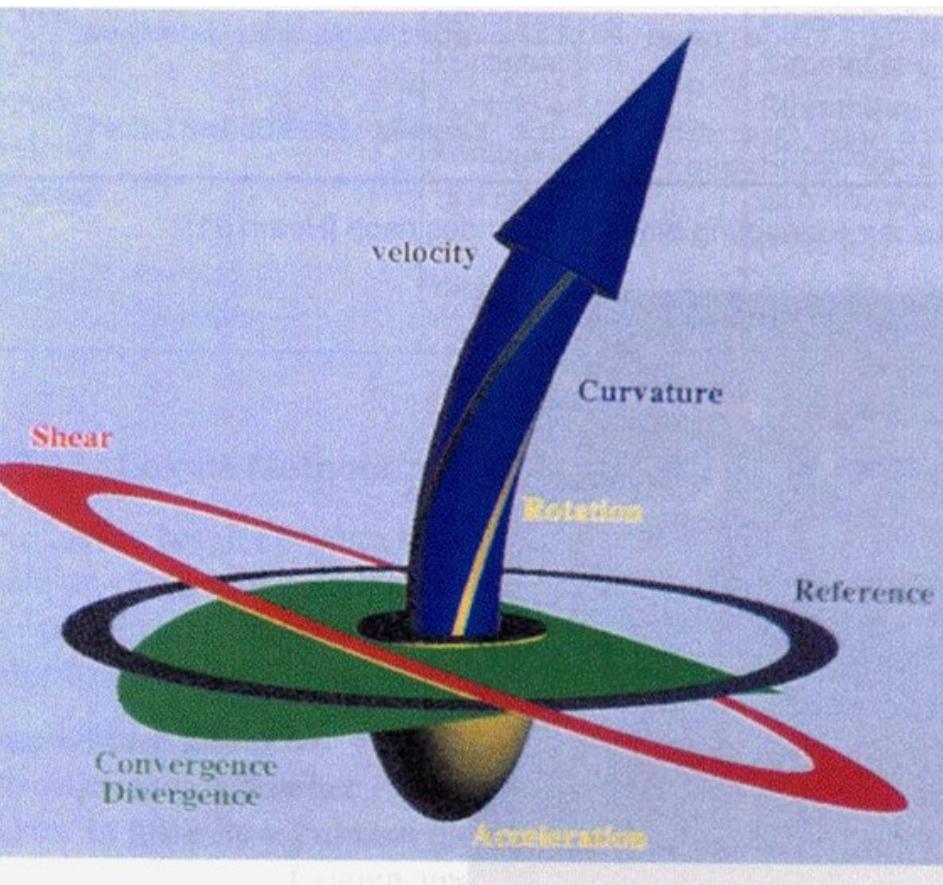
Example with arrows



Arrows and Glyphs

Glyphs

- More complex information, usually used in 3D
- Often more difficult interpretation



Arrows and Glyphs

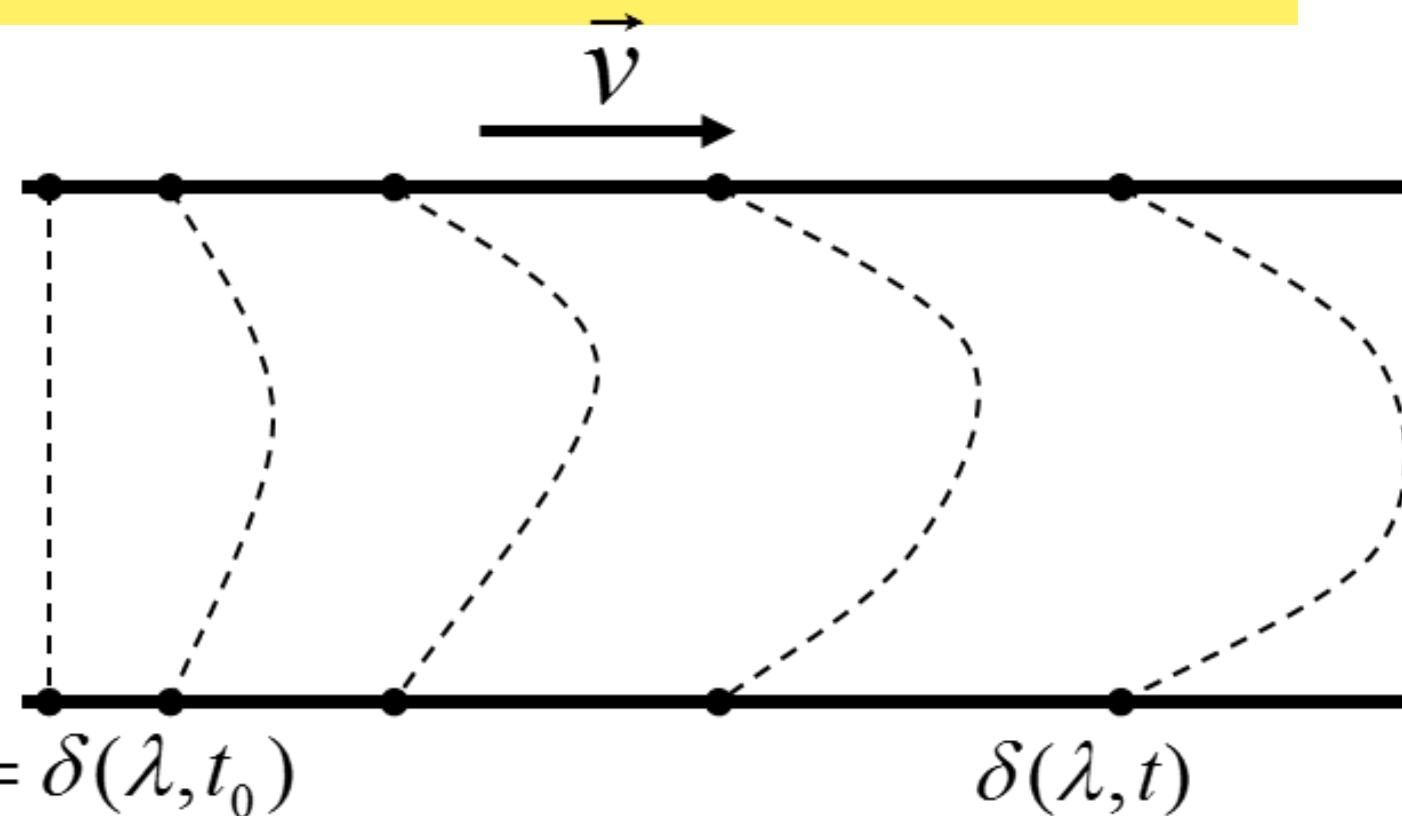
- Advantages
 - Simple
 - Absolute value directly visible
 - 2D fields without complex small structures
- Disadvantages
 - Inherent occlusion effects
 - Very small and overlapping arrows
 - Difficult for 3D and time-dependent fields
 - Poor results if magnitude of velocity changes rapidly
 - Use arrows of constant length and color code magnitude

5.2 Characteristic Lines in Vector Fields

Characteristic Lines

Time lines / time surfaces

- Line/surface of massless elements moving with flow
- Points of identical time
- Connect particles that were released simultaneously

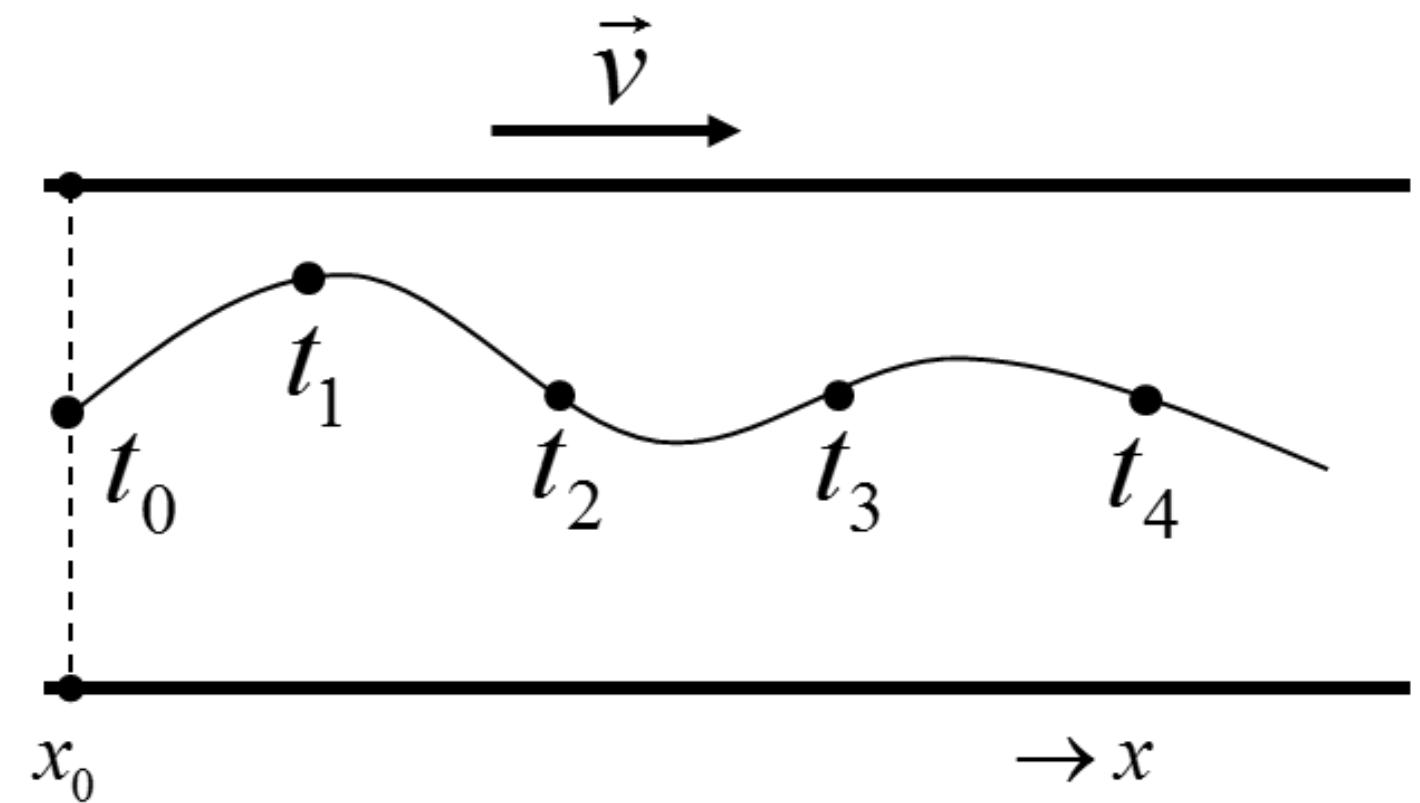


Linienbewegung durch Vektorfeld

Characteristic Lines

Path lines

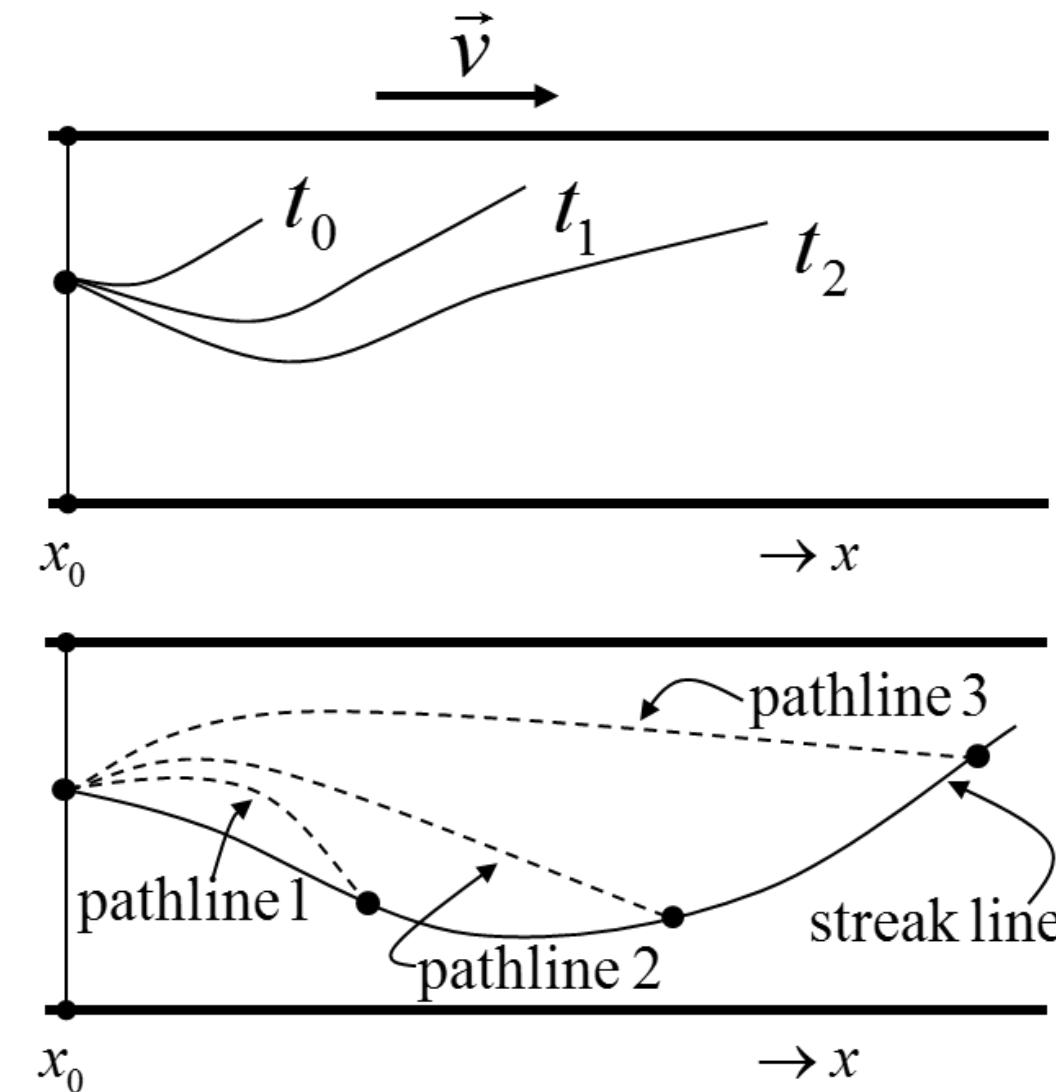
- Path of a single, massless particle
 - Inject particle at point $\mathbf{x}(0) = \mathbf{x}_0$
 - Long time exposure photo
- Solve $\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t)$ with $\mathbf{x}(0) = \mathbf{x}_0$



Characteristic Lines

Streak lines

- Trace of dye continuously released at x_0
- Measurement
 - No particle path, but velocity direction (on line described by dye)
- Representation
 - Connecting line of all particles starting within $0 < t_i < t_0$ at x_0
- Calculation
 - Path lines for $(x_0, x_0 + t)$

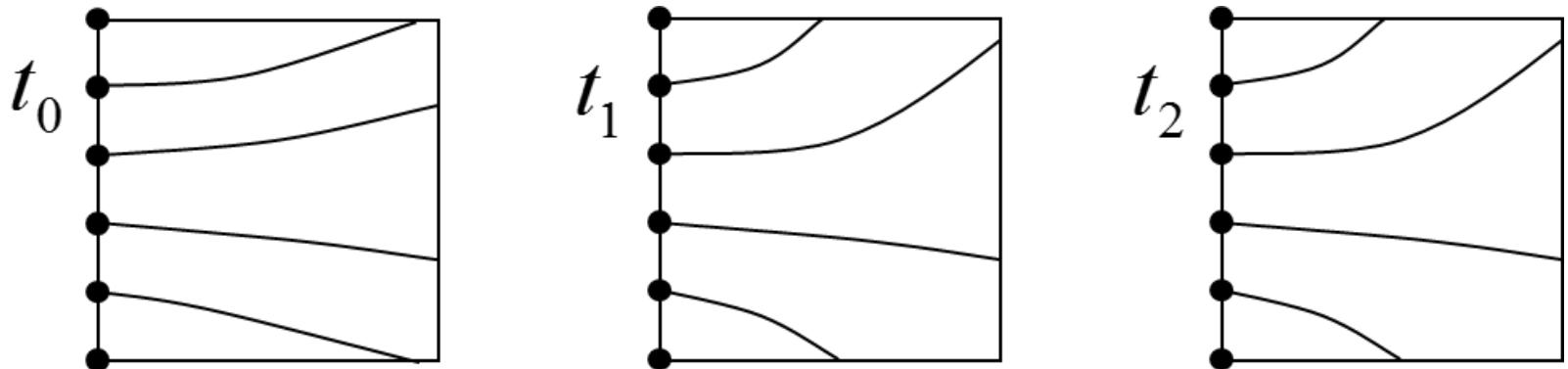


Characteristic Lines

Stream lines (or streamlines)

- Tangential to the vector field - path lines for each t_i
 - Vector field at arbitrary, yet fixed time t

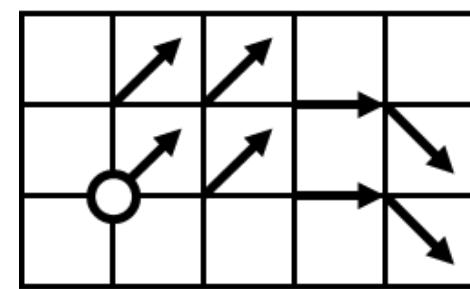
- Solve $\frac{d\mathbf{x}(s)}{ds} = \mathbf{v}(\mathbf{x}(s), t)$ with $\mathbf{x}(0) = \mathbf{x}_0$



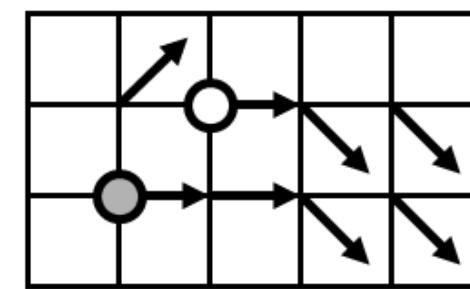
- Photo with short time exposure of vector field v at t_i
 - Freeze the velocity field and compute path lines
 - Information about whole vector field

Characteristic Lines

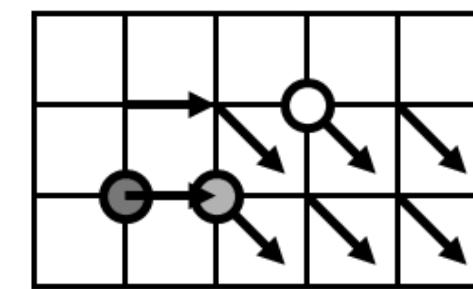
Comparison



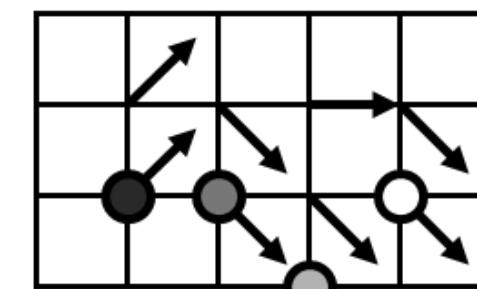
t_0



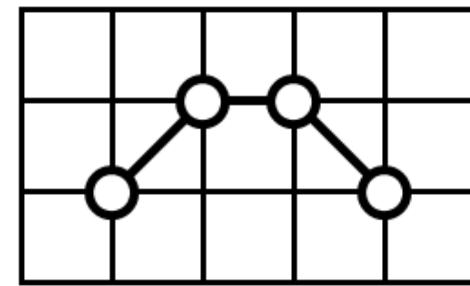
t_1



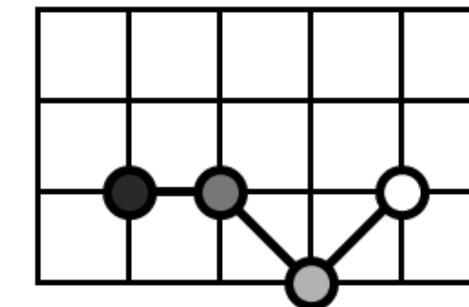
t_2



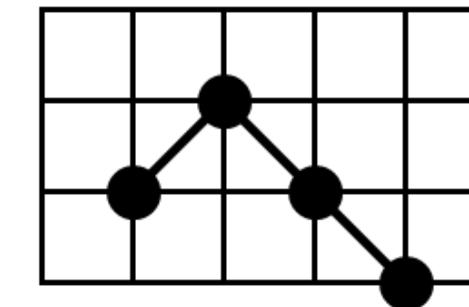
t_3



path line



streak line



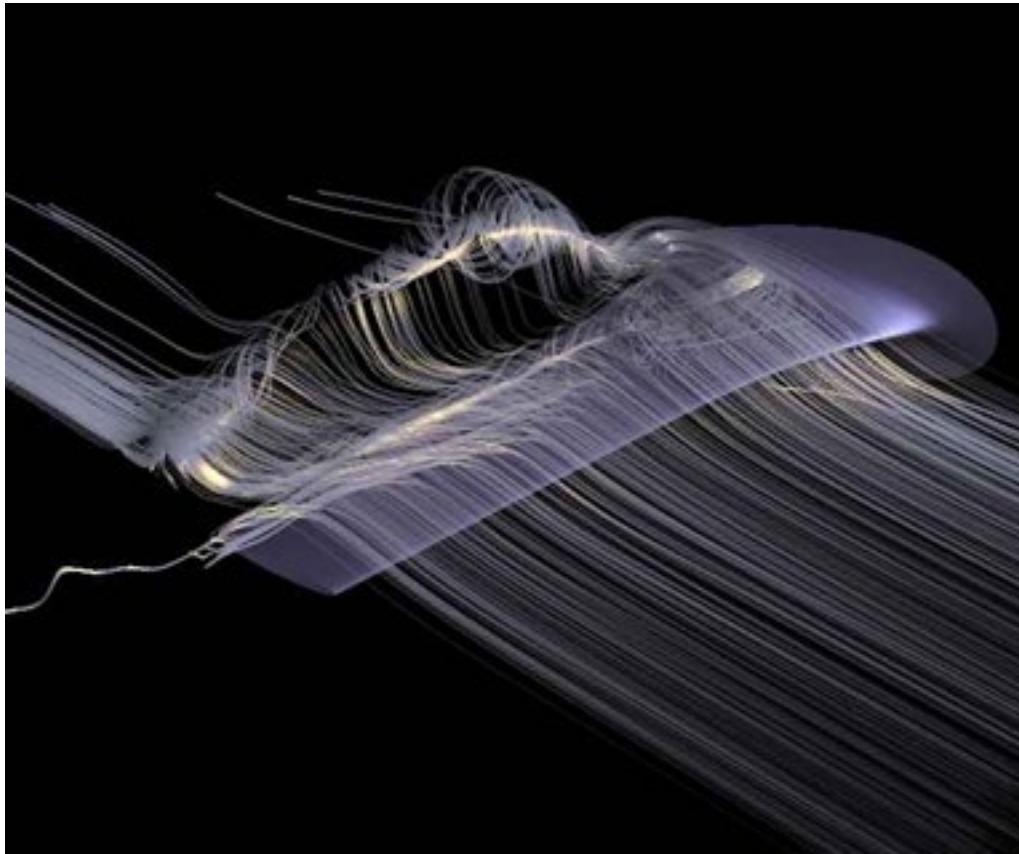
stream line for t_3

For steady flow, path lines, streak lines
and stream lines are identical!

5.3 Particle Tracing

Particle Tracing

- Vector fields usually combined with "transport"
 - Basic idea: trace particles within the field
 - Characteristic lines
 - Mapping approach
 - Lines, surfaces
 - Sometimes animated



Particle Tracing

- Given
 - Velocity field of particles / fluid $\vec{v}(\vec{r}, t)$, $\vec{r} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$
- Task
 - Find the path $\vec{r}(x(t), y(t))$ of one (or many) massless particles in this field
 - Starting point requirement $\vec{r}(t_0) = \vec{r}_0$

• Solve $\frac{d\vec{r}}{dt} = \vec{v}(x(t), y(t)) = \vec{v}(\vec{r}, t) \rightarrow$ thereby $\vec{r}(t_1), \vec{r}(t_2), \dots$

Integration

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \int_t^{t+\Delta t} \vec{v}(\vec{r}(t), t) dt$$

Particle Tracing

Integration

- This requires solving ordinary differential equations (ODE)
 - Usually of first order initial value problems
- Various methods
 - Single step: Euler
 - Multi step: Adams Bashforth
 - Multi stage: Runge-Kutta, Heun
- Literature
 - Schwarz, "Numerische Mathematik" (Kapitel 9)
 - Deuflhard, Bornemann, "Numerische Mathematik II"
 - Brauer, "Numerische Behandlung gewöhnlicher Differentialgleichungen"
 - and many more ...

Particle Tracing

Ordinary differential equation

- Initial value problem $\frac{d\vec{r}}{dt} = \vec{v}(\vec{r}, t)$, where $\vec{r}(t_0) = \vec{r}_0$
- Numerical solution
 - Discretize the "time variable": $\{t_1, t_2, \dots\}$
 - Determine r_j which approximates the solution $r(t_j)$
 - Usually, we use constant step size: $t_{j+1} = t_j + \Delta t$

konstant

Particle Tracing

Classification of methods

		neue Position aus alter	mehrere Positionen werden betrachtet
		Single step	Multi step
Explicit	Single step	$\vec{r}_{i+1} = \phi_i(\Delta t, t_i, \vec{r}_i)$	$\vec{r}_{i+1} = \phi_i(\Delta t, t_i, \vec{r}_i, \vec{r}_{i-1}, \vec{r}_{i-2}, \dots)$
	Multi step		$\psi_i(\Delta t, t_i, \vec{r}_i, \vec{r}_{i+1}) = 0$

Particle Tracing

- Single step methods (e.g. Euler)
 - Position r_{i+1} depends only on r_i
 - Good for adaptive strategy since no recalculation is required
- Multi step methods (e.g. Adams Bashforth)
 - Uses results of a fixed number of previous time steps
 - Cost intensive for adaptivity
- Multi stage (e.g. Heun, Runge-Kutta)
 - Single step approach with multiple evaluations of the function per time step
 - Due to cost effectiveness applied for refinement

Particle Tracing

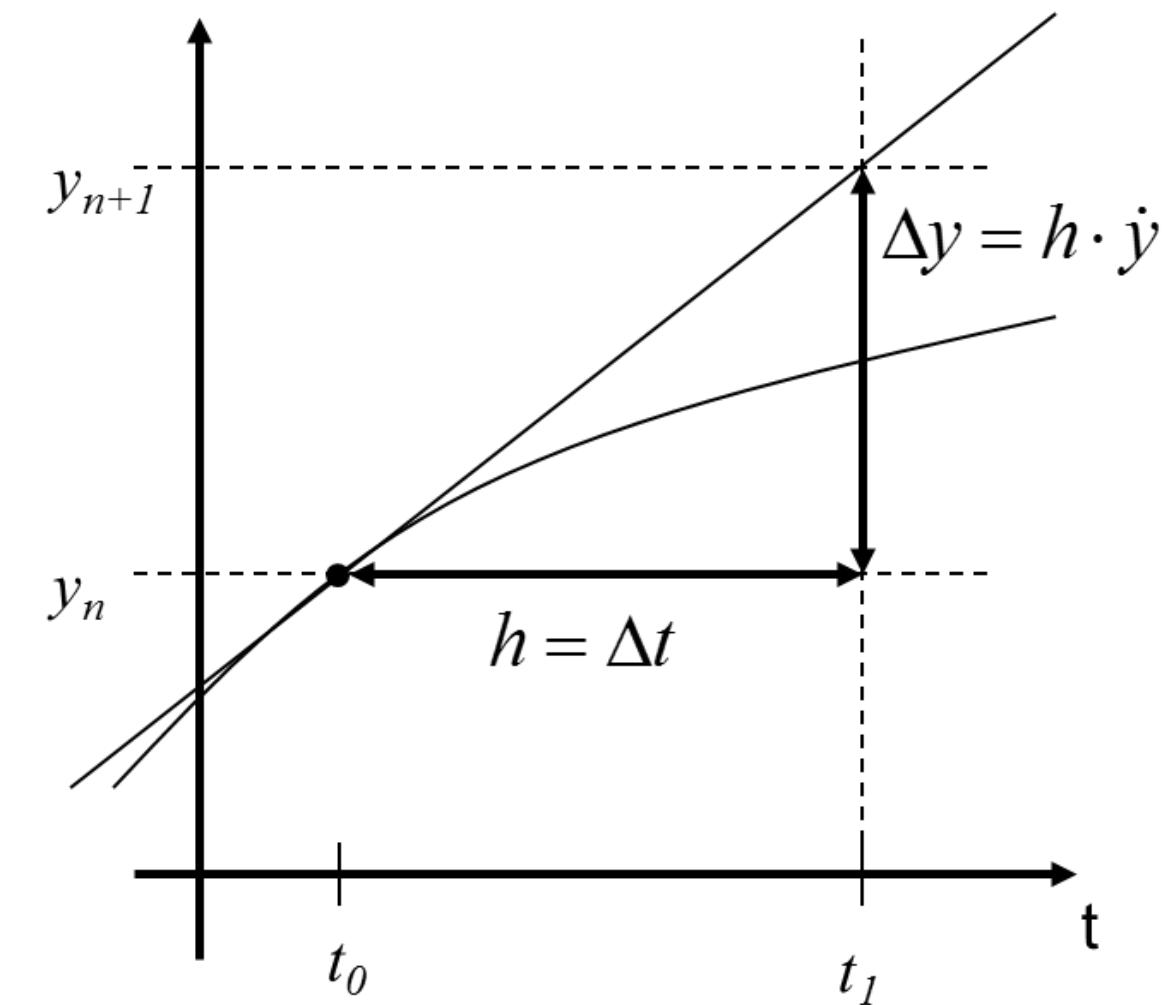
Euler method (Runge-Kutta 1st order - RK1)

- Explicit $\vec{r}_{j+1} = \vec{r}_j + \Delta t \vec{v}(\vec{r}_j, t_j) \quad t_{j+1} = t_j + \Delta t$
- Implicit $\vec{r}_{j+1} = \vec{r}_j + \Delta t \vec{v}(t_j + \Delta t, \vec{r}_{j+1})$
- Local discretization error $O(\Delta t^2)$

Particle Tracing

Explanation in 1D

- Explicit $y_{n+1} \cong y_n + h f(y(n), t)$
 $\cong y_n + h \dot{y}(y(n), t)$
 $\cong y_n + h \dot{y}_n$
- Implicit $y_{n+1} \cong y_n + h \dot{y}_{n+1}$



Particle Tracing

Example with Euler

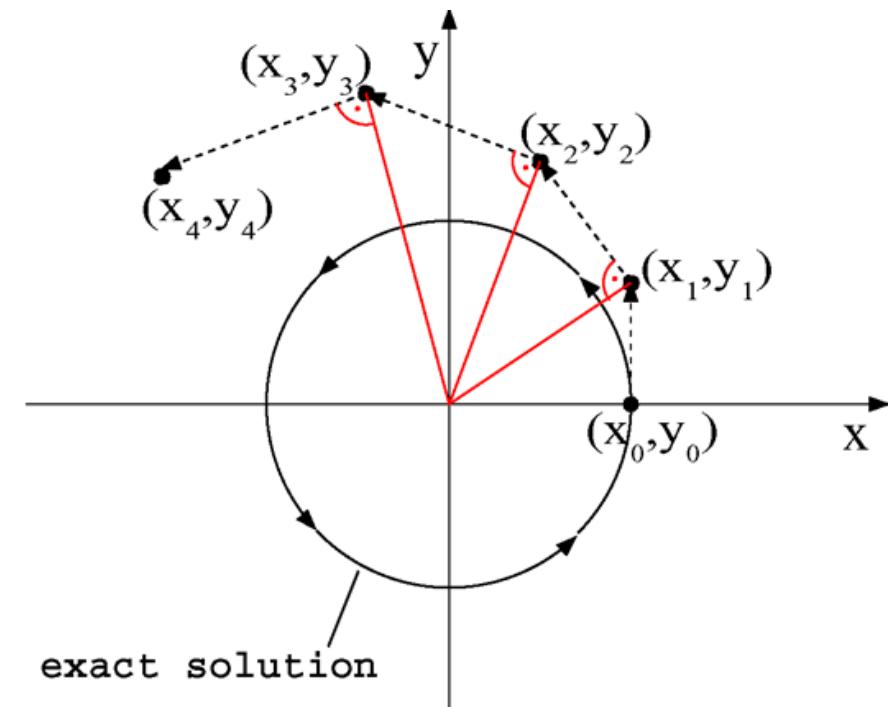
- Curl field $\vec{v}(t, x, y) = \vec{v}(x, y) = \begin{pmatrix} y \\ -x \end{pmatrix}$
- Exact solution is $\begin{pmatrix} y \\ x \end{pmatrix} = C \begin{pmatrix} \sin t \\ \cos t \end{pmatrix}$
- This is a circle centered at the origin (radius depends on the starting point)

Particle Tracing

Explicit Euler

- Obtain velocity at current (x_i, y_i) with $v(x_i, y_i)$
 - Consider tangent at "starting point" (x_i, y_i)
- Assuming velocity is constant over the next time step leads to →
 - Form of a spiral due to discretization error!

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) + \Delta t \vec{v}(x_i, y_i)$$

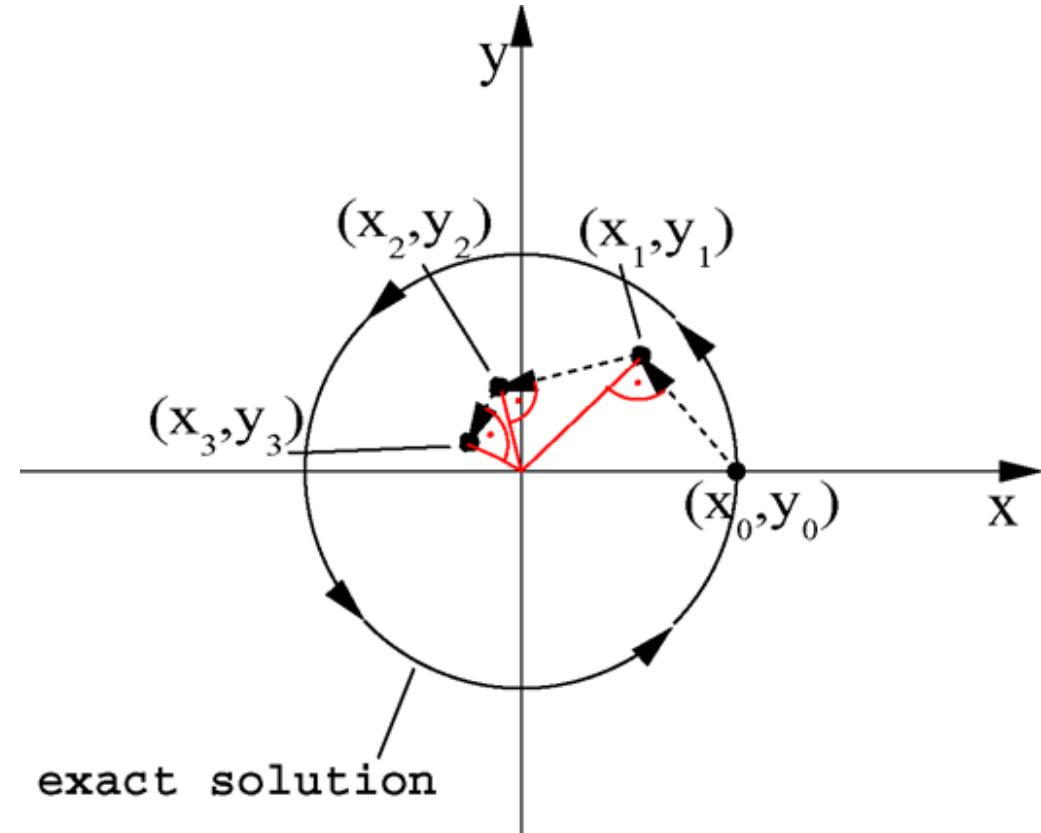


Particle Tracing

Implicit Euler

- Search (x_{i+1}, y_{i+1}) with velocity $v(x_{i+1}, y_{i+1})$
 - Consider tangent at "end-point" (x_{i+1}, y_{i+1})
 - This leads to →
 - Also form of a spiral
- 'due to discretization error!'

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) + \Delta t \vec{v}(x_{i+1}, y_{i+1})$$



Particle Tracing

Euler summary

- Explicit
 - Local error $O(\Delta t^2)$
 - Global error $O(\Delta t)$
 - We need $1/\Delta t$ steps
- Implicit
 - Local error $O(\Delta t_2)$, global error $O(\Delta t)$
 - More stable than explicit
 - If v is known analytically, we have to solve for x_{i+1}, y_{i+1}
 - Otherwise: iteration is required
 - Predictor-corrector variant (not covered here)
- Bottom line: avoid explicit Euler methods!

Particle Tracing

Heun method (Runge-Kutta 2nd order - RK2)

- Explicit

$$\vec{r}_{i+1} = \vec{r}_i + \frac{\Delta t}{2} \left(\vec{k}_1 + \vec{k}_2 \right) \quad \begin{aligned} \vec{k}_1 &= \vec{v}(t_j, y_j) \\ \vec{k}_2 &= \vec{v}(t_j + \Delta t, \vec{r}_j + \Delta t \vec{k}_1) \end{aligned}$$

- Some kind of "average" between implicit and explicit Euler
- Implicit

$$\vec{r}_{j+1} = \vec{r}_j + \frac{\Delta t}{2} \left(\vec{v}(t_j, \vec{r}_j) + \vec{v}(t_j + \Delta t, \vec{r}_{j+1}) \right)$$

- Local discretization error $O(\Delta t^3)$

Particle Tracing

Classical Runge-Kutta (RK4)

- Application of Simpson's rule

$$\vec{r}_{j+1} = \vec{r}_j + \Delta t / 6 \left(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4 \right)$$

where

$$\vec{k}_1 = \vec{v}(t_j, \vec{r}_j)$$

$$\vec{k}_2 = \vec{v}\left(t_j + \frac{\Delta t}{2}, \vec{r}_j + \frac{\Delta t}{2} \vec{k}_1\right)$$

$$\vec{k}_3 = \vec{v}\left(t_j + \frac{\Delta t}{2}, \vec{r}_j + \frac{\Delta t}{2} \vec{k}_2\right)$$

$$\vec{k}_4 = \vec{v}(t_j + \Delta t, \vec{r}_j + \Delta t \vec{k}_3)$$

NOTE:

- There is also implicit RK4
- Note that stability is achieved either by multiple function evaluations or with implicit techniques.
- Here: explicit RK4 has multiple function evaluations \Rightarrow ok!

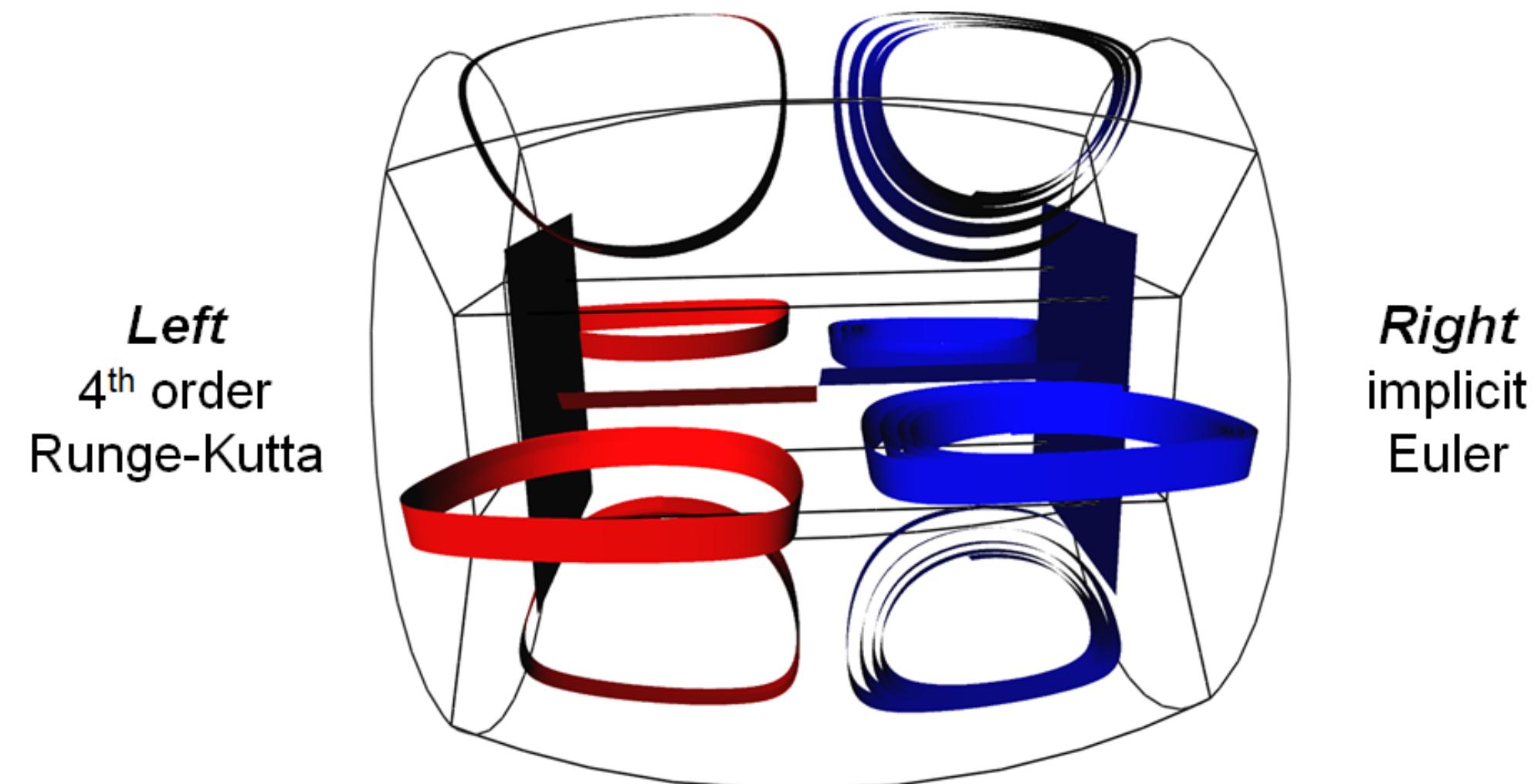
- Local discretization error $O(\Delta t^5)$

relativ kleiner Diskretisierungsfehler

Particle Tracing

The right integration technique can make a difference

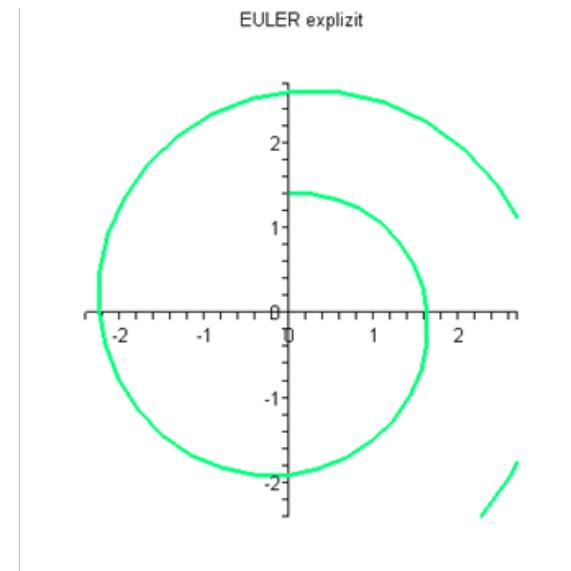
- Example: flow in a floating zone furnace for crystal growing



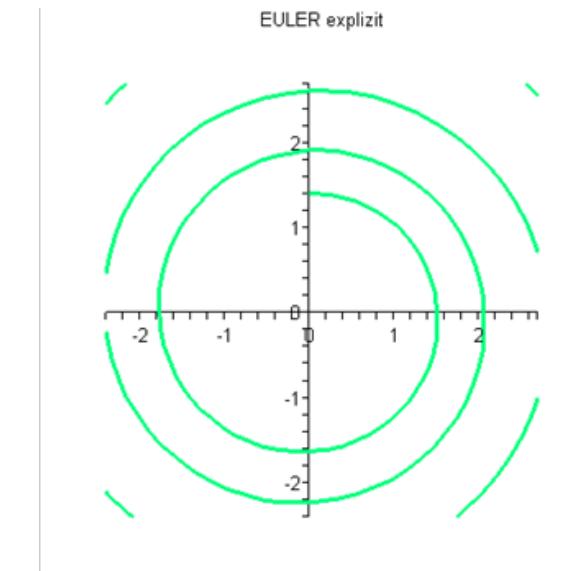
Particle Tracing

Example: circular flow with Euler

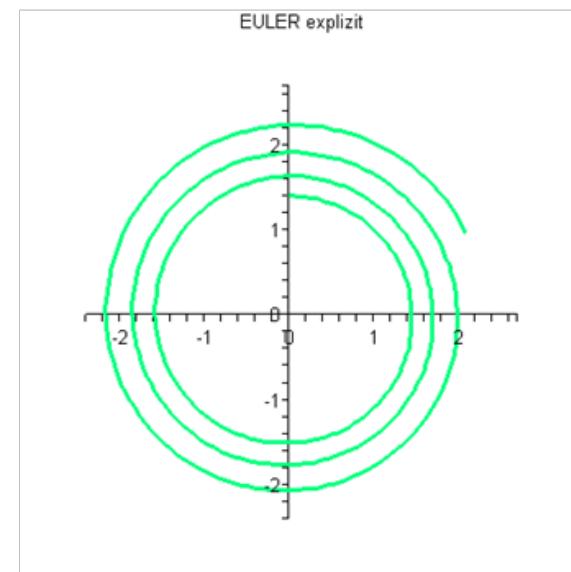
$\Delta t = 0.200$



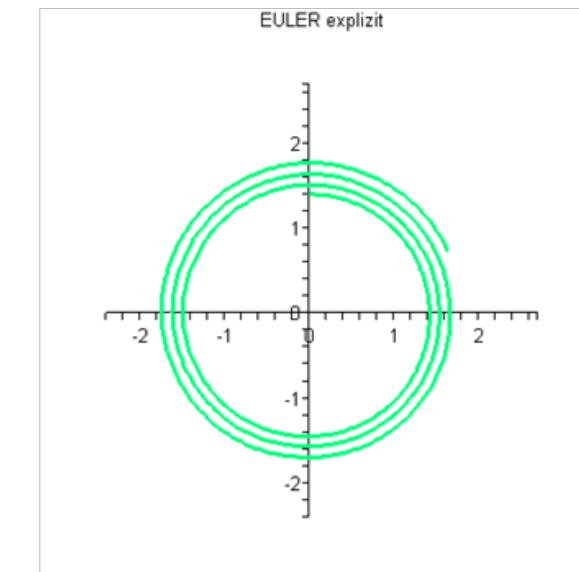
$\Delta t = 0.100$



$\Delta t = 0.050$



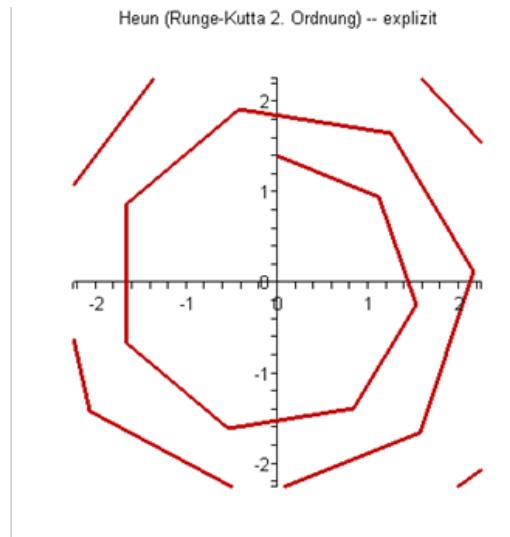
$\Delta t = 0.025$



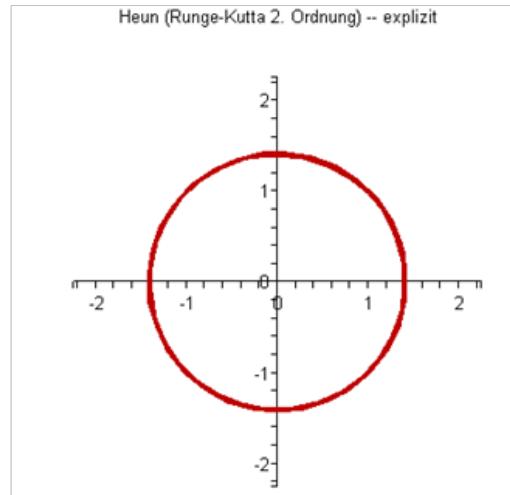
Particle Tracing

Example: circular flow with Heun

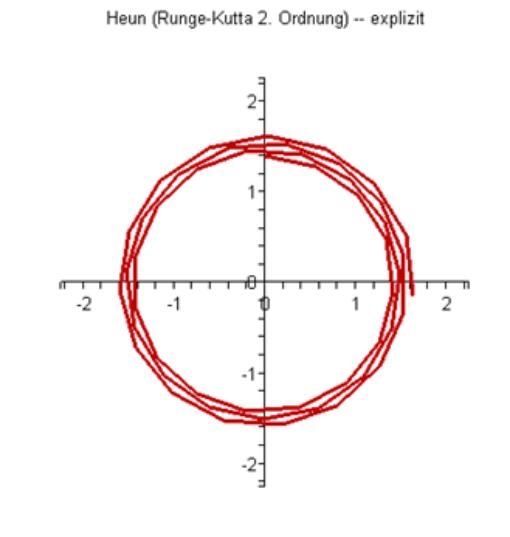
$\Delta t = 0.800$



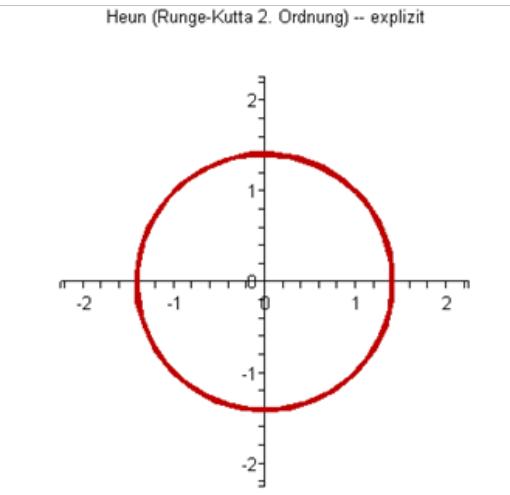
$\Delta t = 0.200$



$\Delta t = 0.400$

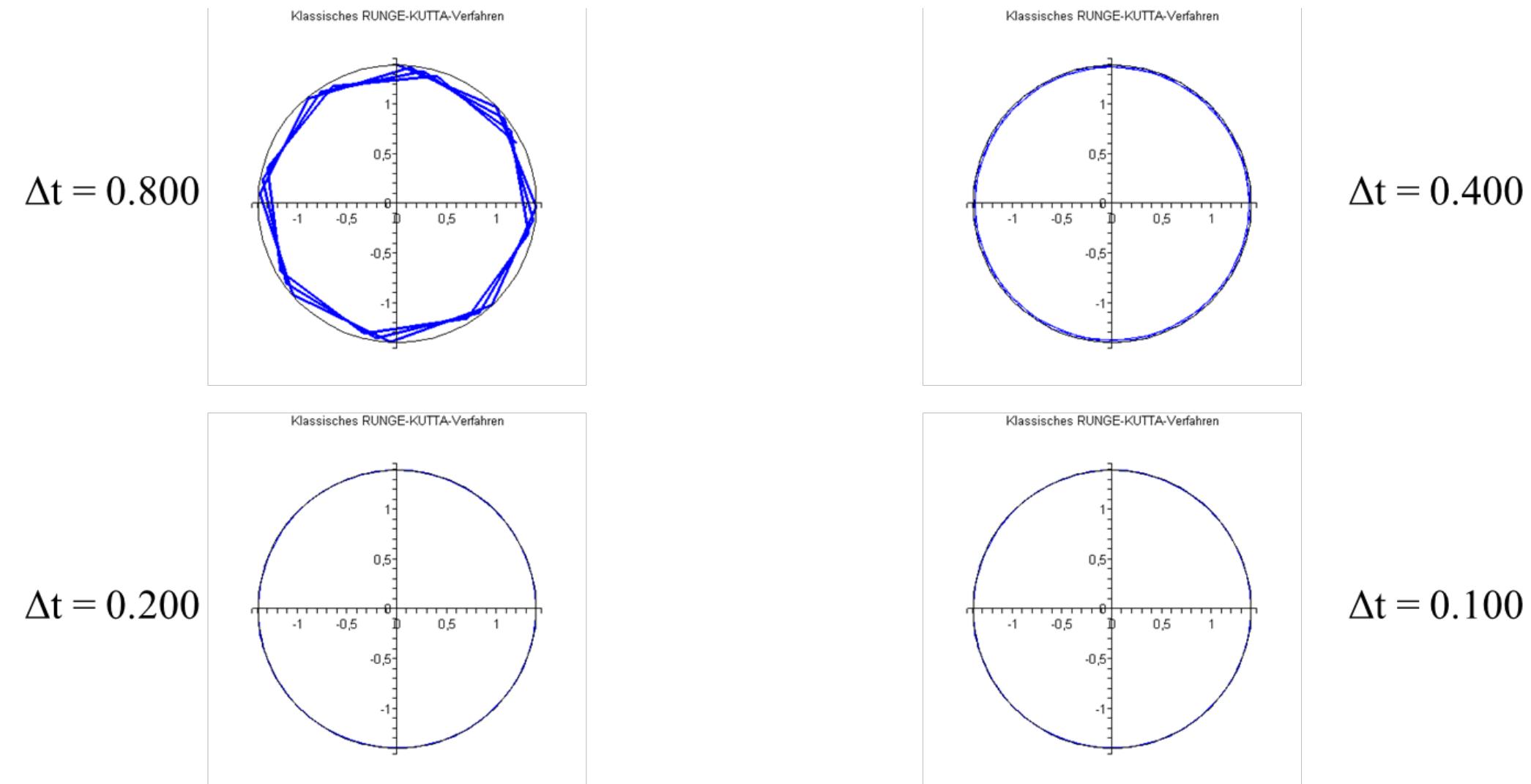


$\Delta t = 0.100$



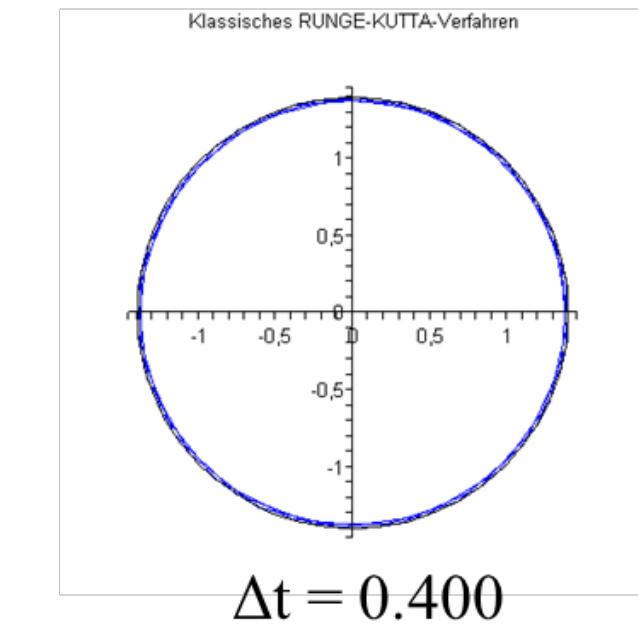
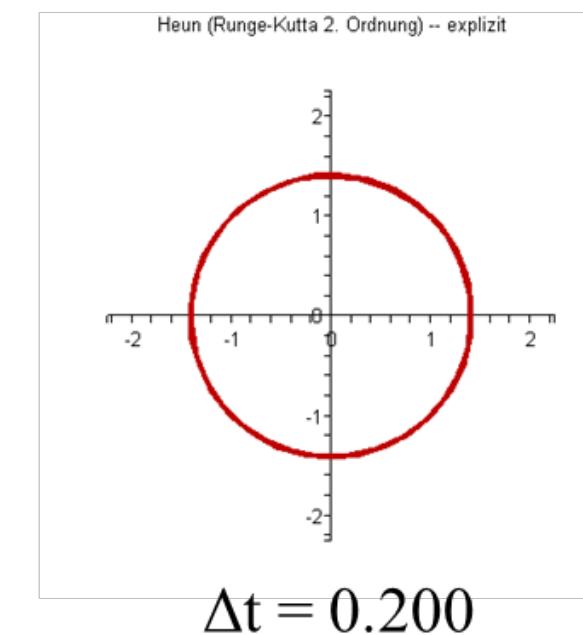
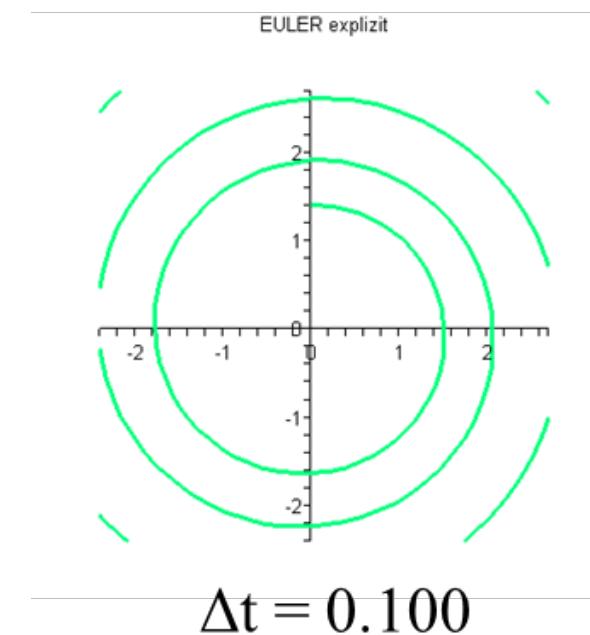
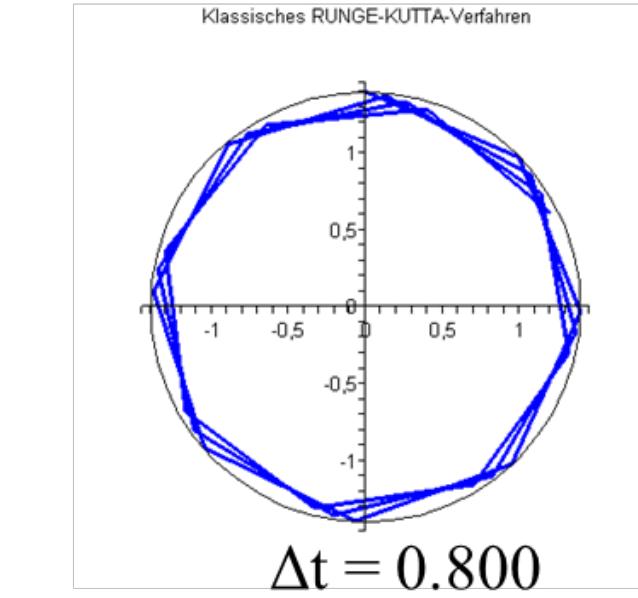
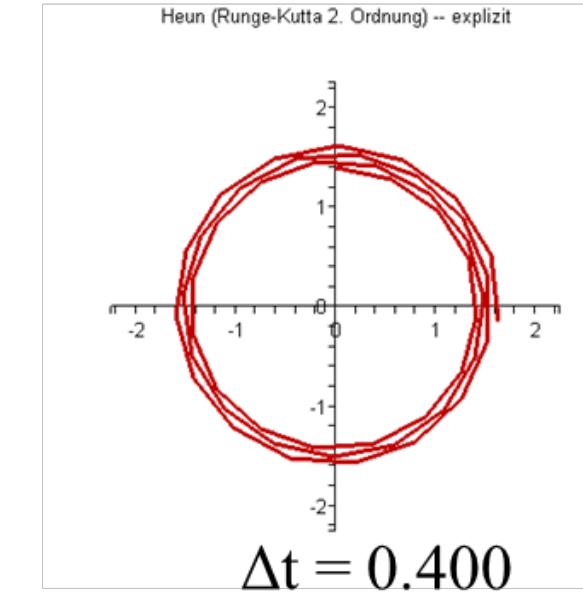
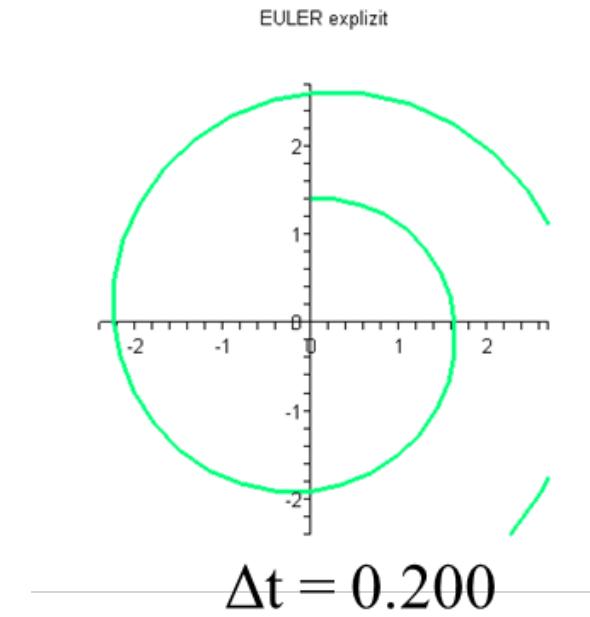
Particle Tracing

Example: circular flow with Runge-Kutta



Particle Tracing

Comparision & dependency on Δt

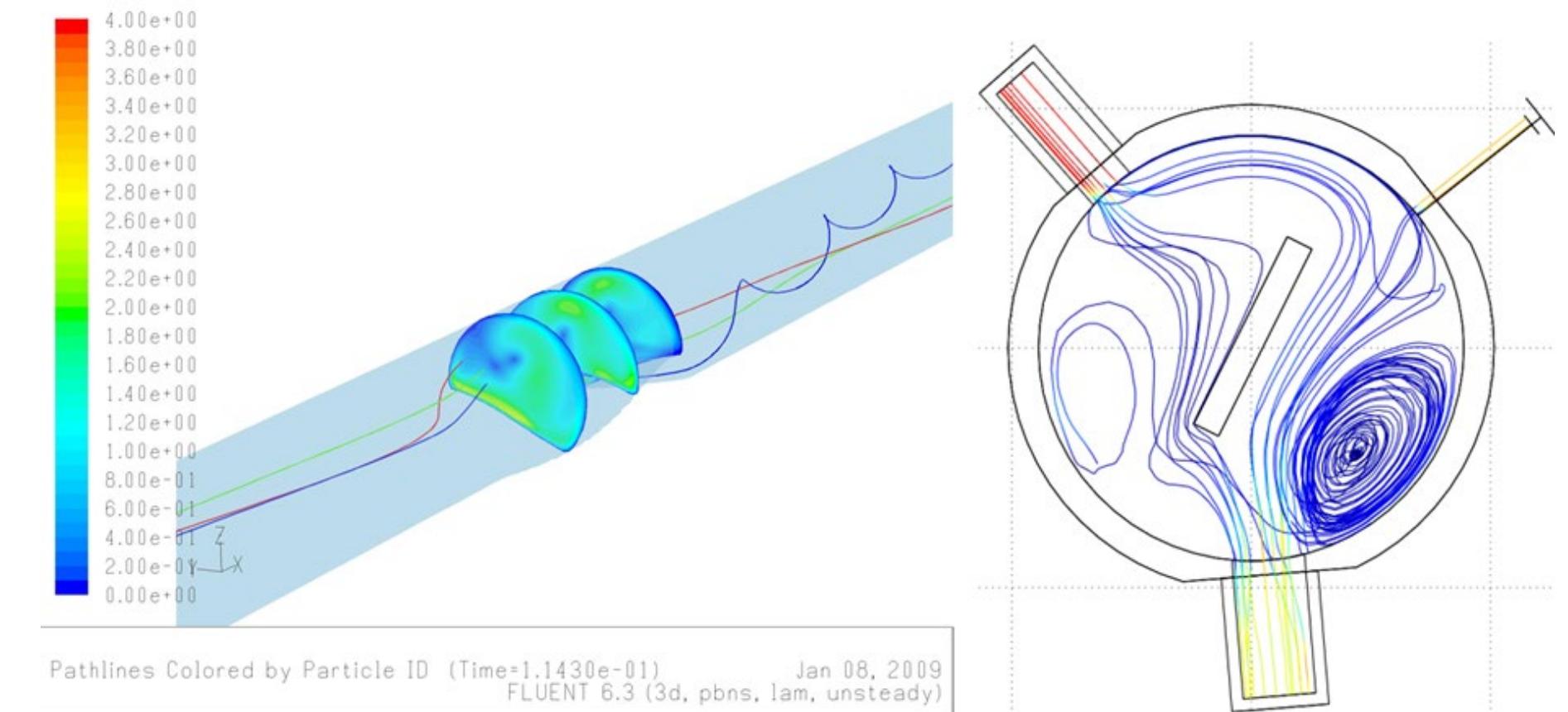


Particle Tracing

- Issues with numerical integration
 - Stability
 - Order of error
 - Step size control
 - Computational effort

Particle Tracing

- Fundamental algorithm
 - Pick a point (= particle) in the data region
 - Determine its path (= integral curve) emanating at this point
- Interpretation
 - Trace the particle without mass in the vector field flow

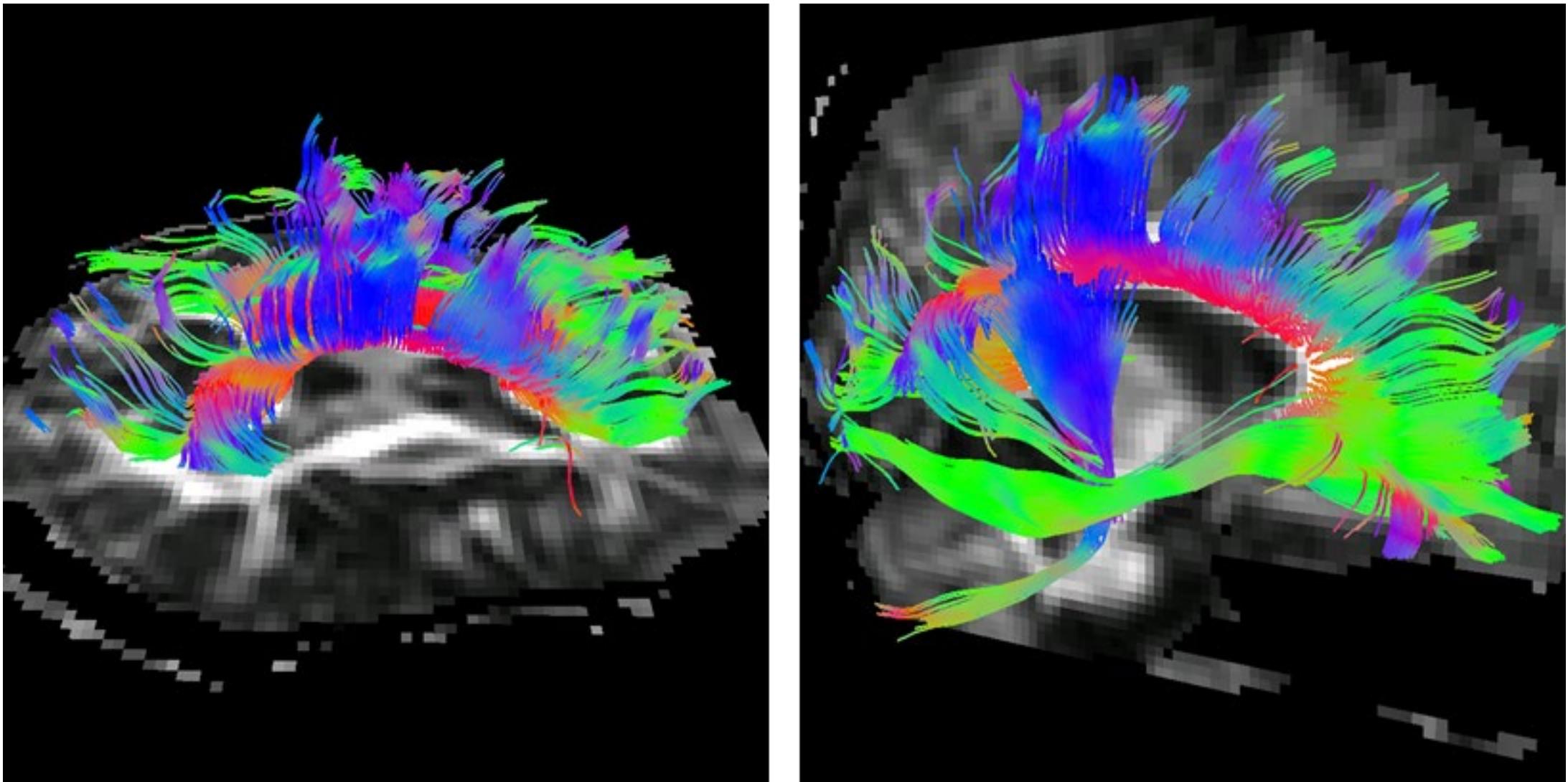


Particle Tracing

- Assumptions
 - Consider only stationary fields $v = v(r)$
 - Time-dependent fields not covered in this lecture
 - The vector field
 - Is not given as an analytic function
 - It is defined on a discrete mesh (rectilinear, structured, unstructured, ...)
 - Values "in-between" have to be interpolated!

Particle Tracing

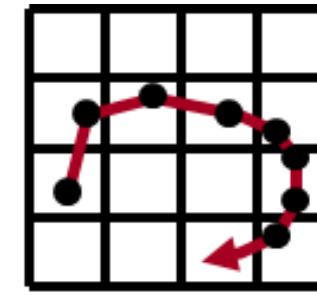
Example: Fiber tracking in a human brain



Particle Tracing

Fundamental algorithm

```
Select start point  $r_{old} = r_0$ 
Find cell that contains  $r_{old}$            // point location
Set step size:  $\Delta t = \dots \leq$           //  $\frac{1}{2}$ (average cell size)
while (particle in domain) do
    Determine vector field at  $r_{old}$         // interpolation
    Integrate to new position  $r_{new}$        // integration
    Perform step size control             // optional
        (i.e.:  $\Delta t = \Delta t/2$  or  $\Delta t = 2\Delta t$ )
    Draw line segment  $[r_{old}, r_{new}]$ 
     $r_{old} = r_{new}$ 
    Find cell that contains  $r_{old}$            // point location
endwhile
```



Particle Tracing

- How to do cell search?
 - Depends on the type of mesh!
 - Global search only at the very beginning
 - Later on, only local search
 - Because we know the cell of a point that is close
 - Overview
 - Rectilinear meshes
 - No problem
 - Unstructured triangular meshes
 - Barycentric coordinates
 - Curvilinear grids
 - P-Space / C-Space algorithm

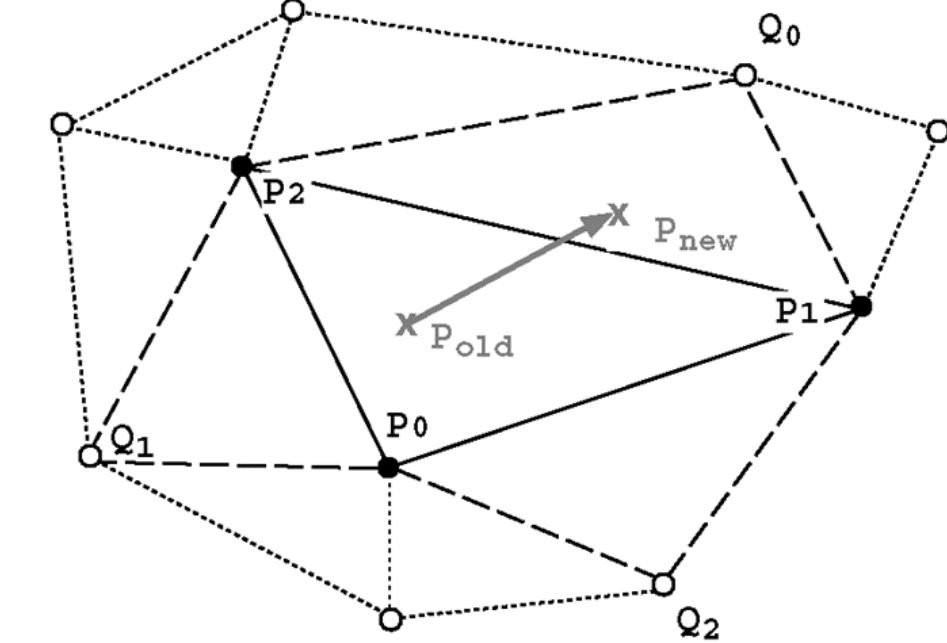
Particle Tracing

- Cell search in rectilinear meshes
 - Index of cell directly from position (x,y,z)

• e.g. $\Delta x = \frac{x_{\max} - x_{\min}}{n}$ $i = \left\lfloor \frac{x_p}{\Delta x} \right\rfloor$

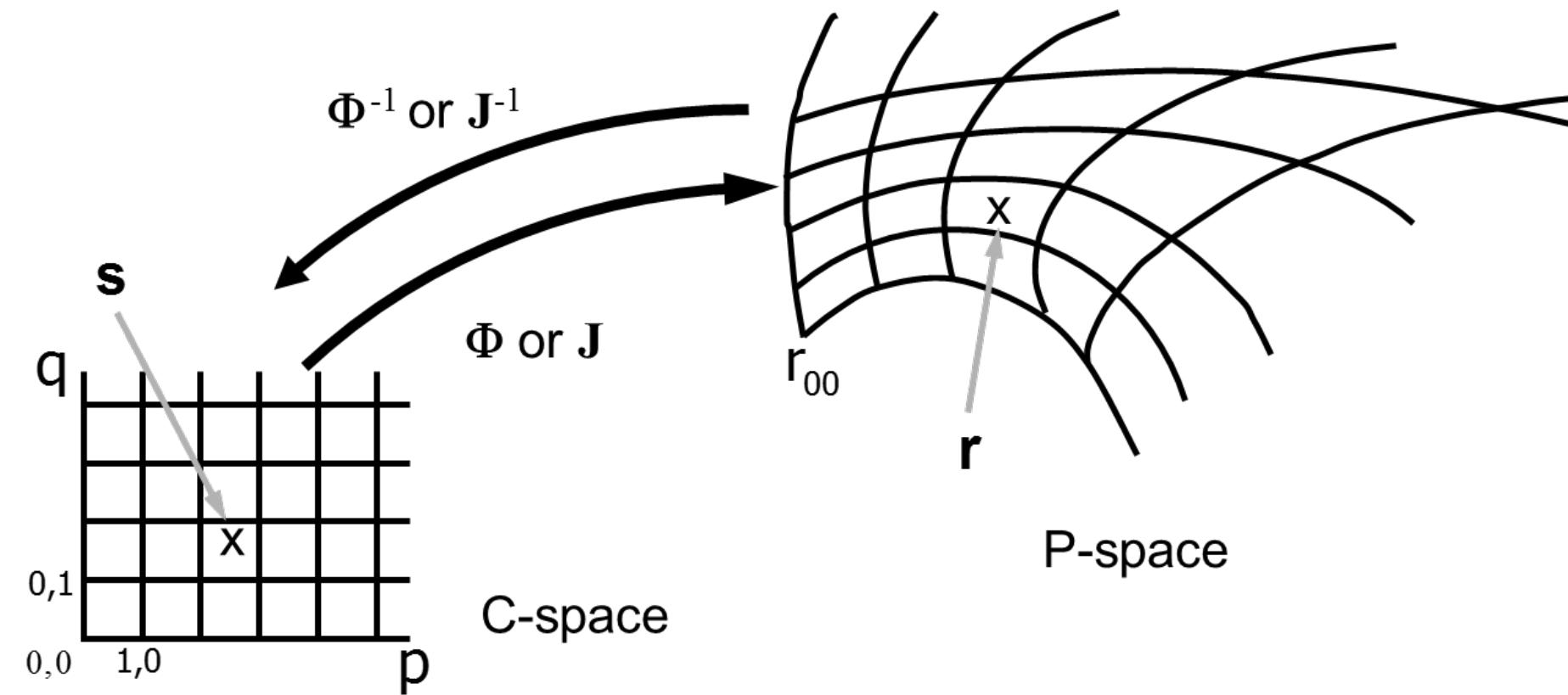
- Unstructured triangular meshes
 - Barycentric coordinates of P_{new} : d_0, d_1, d_2

$d_0 < 0$	$d_1 < 0$	$d_2 < 0$	
0	0	0	$P_{\text{new}} \in \Delta(P_0, P_1, P_2)$
0	0	1	check $P_{\text{new}} \in \Delta(Q_2, P_1, P_0)$ recursively
0	1	0	check $P_{\text{new}} \in \Delta(Q_1, P_0, P_2)$ recursively
1	0	0	check $P_{\text{new}} \in \Delta(Q_0, P_2, P_1)$ recursively
0	1	1	process fan centered at P_0
1	0	1	process fan centered at P_1
1	1	0	process fan centered at P_2
1	1	1	error



Particle Tracing

- Curvilinear grids

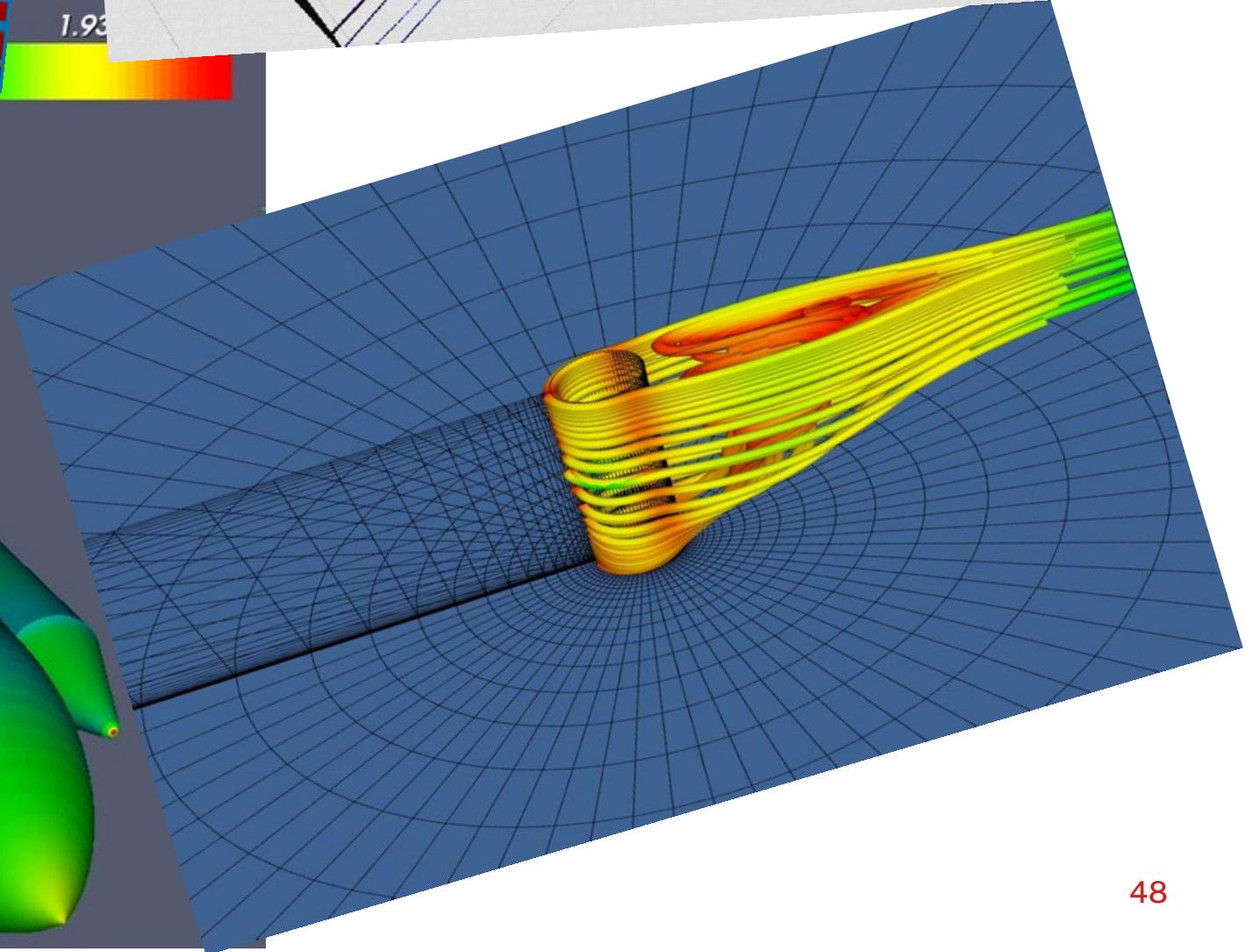
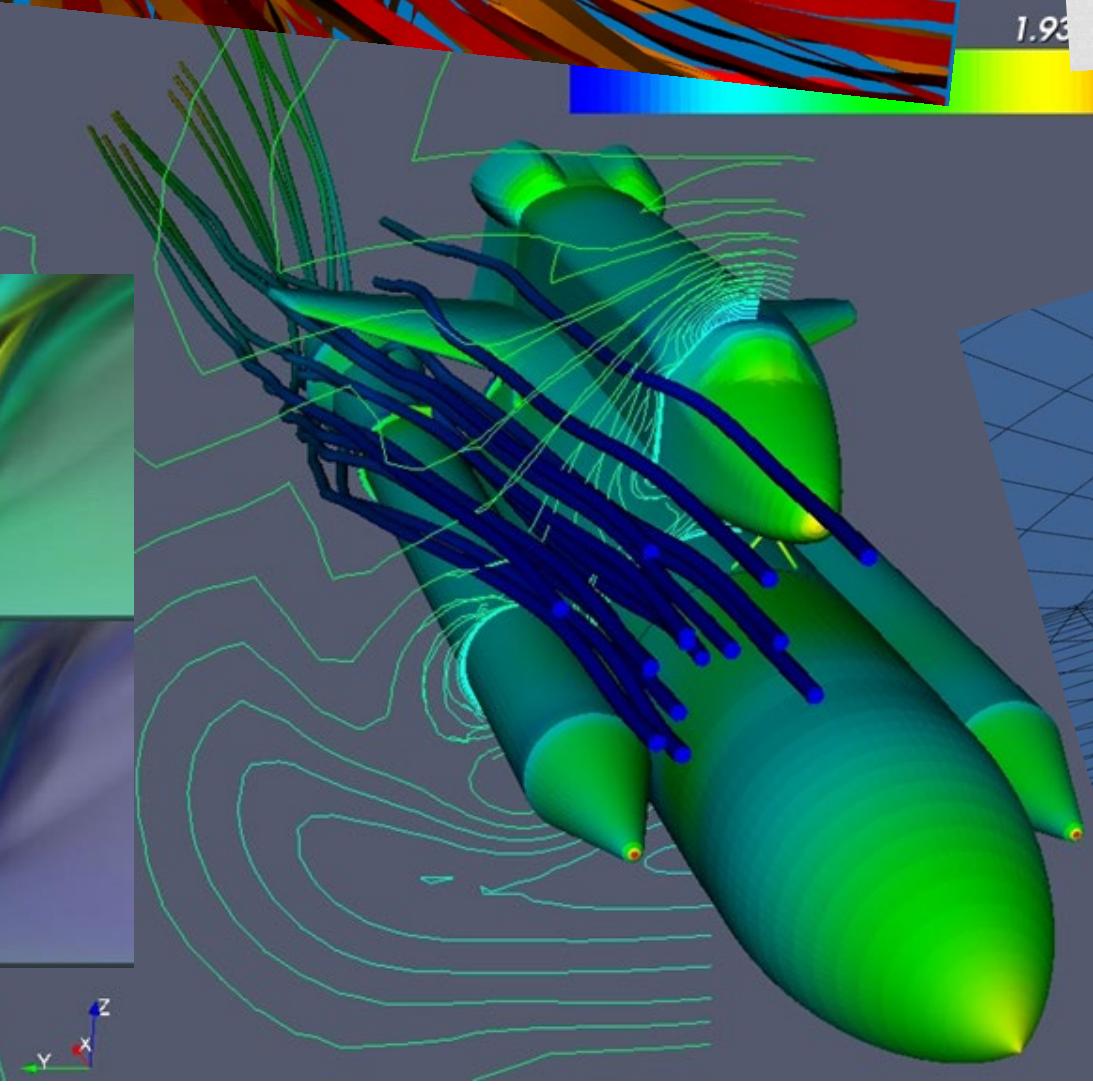
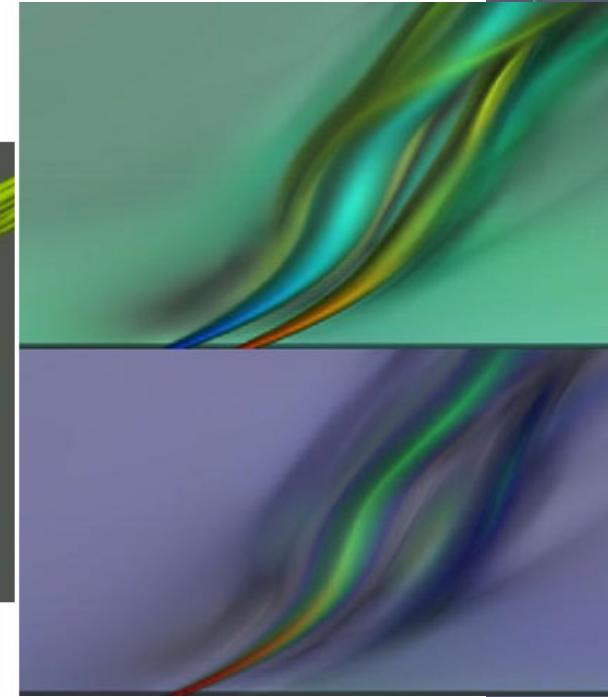
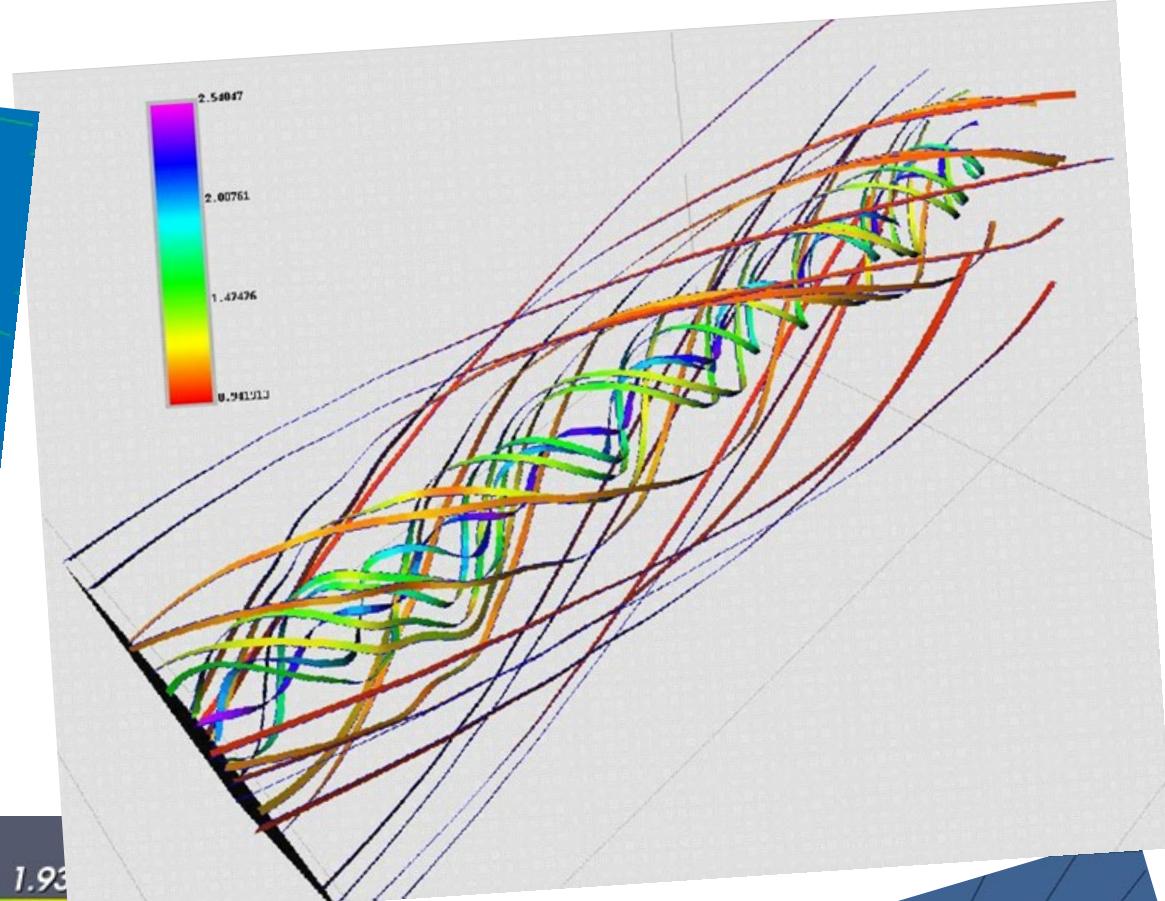
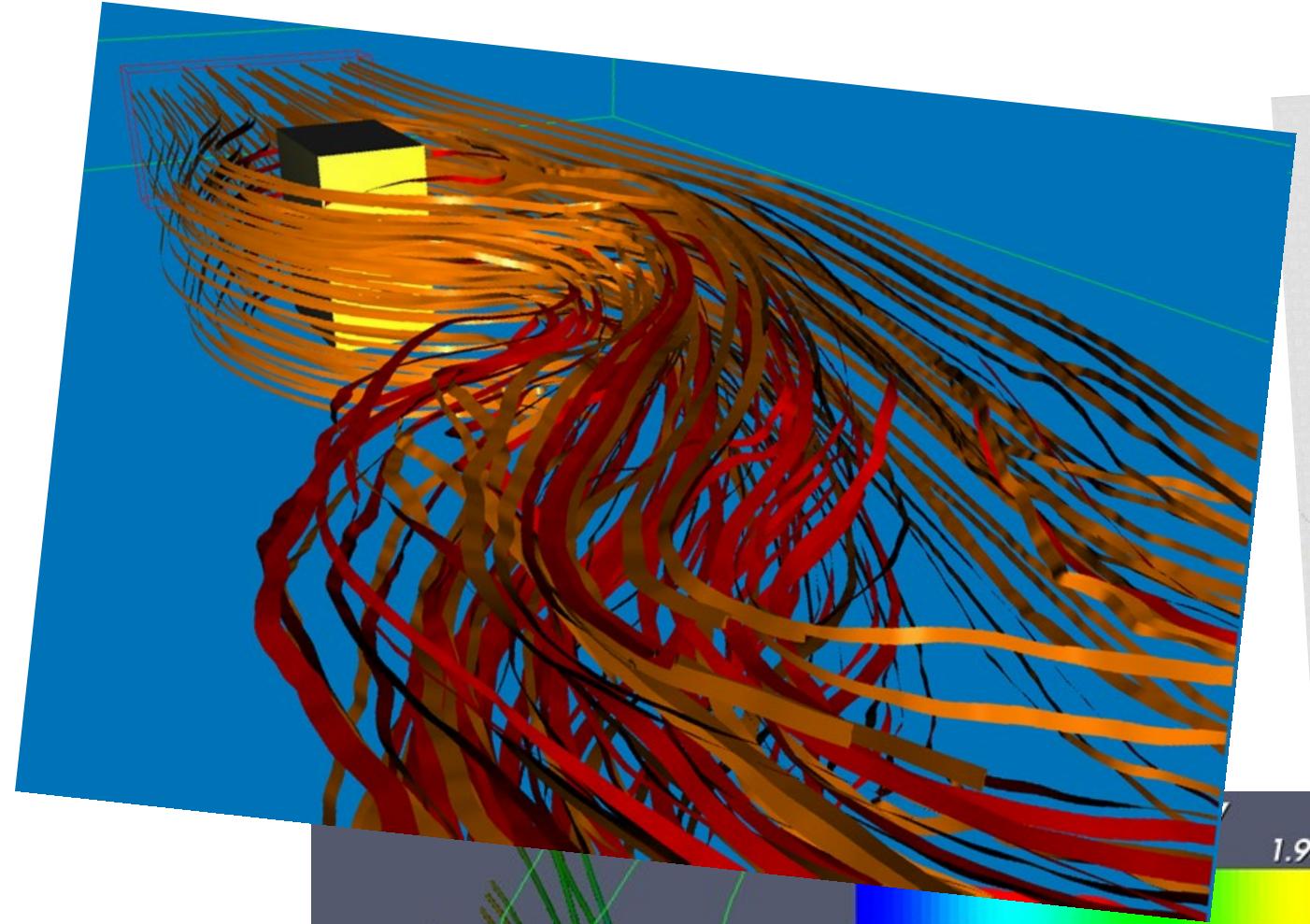
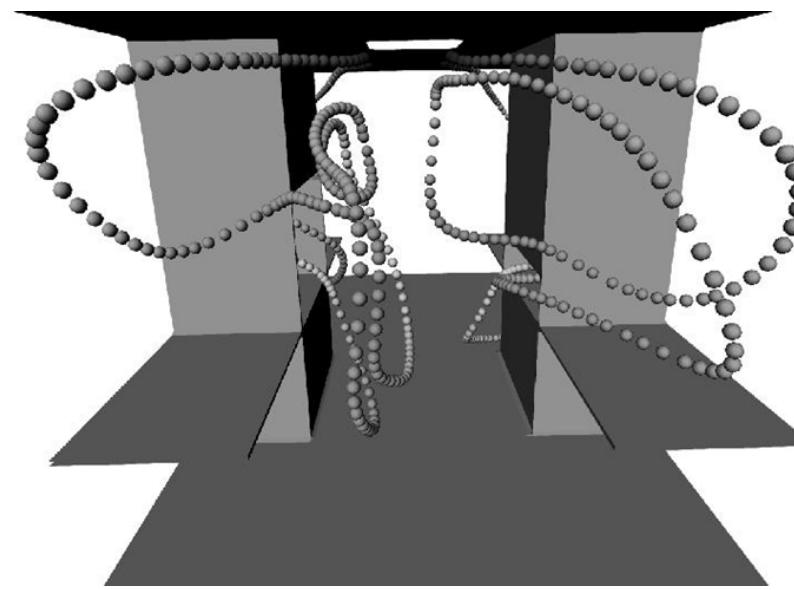


- Transformation of spaces
- This is considerably more involved than processing structured grids!

Particle Tracing

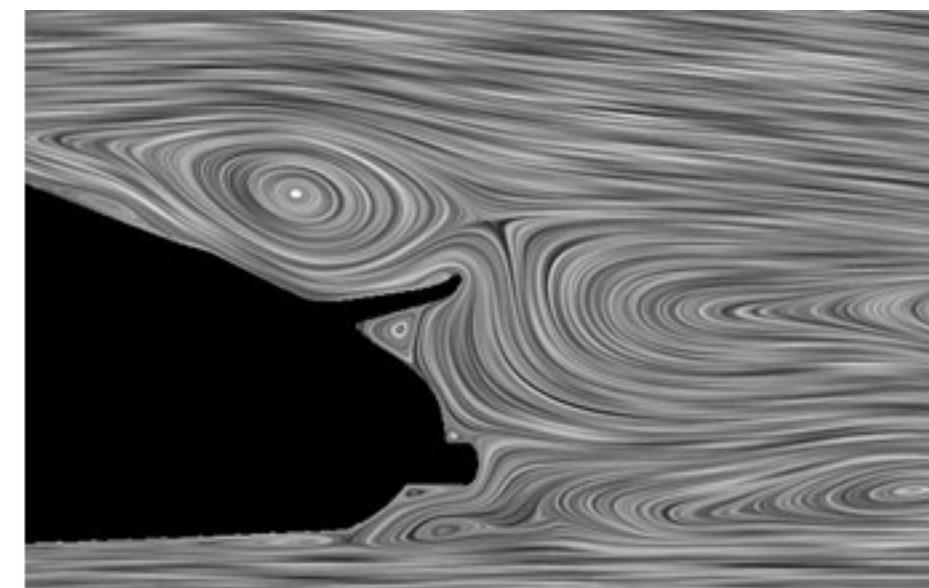
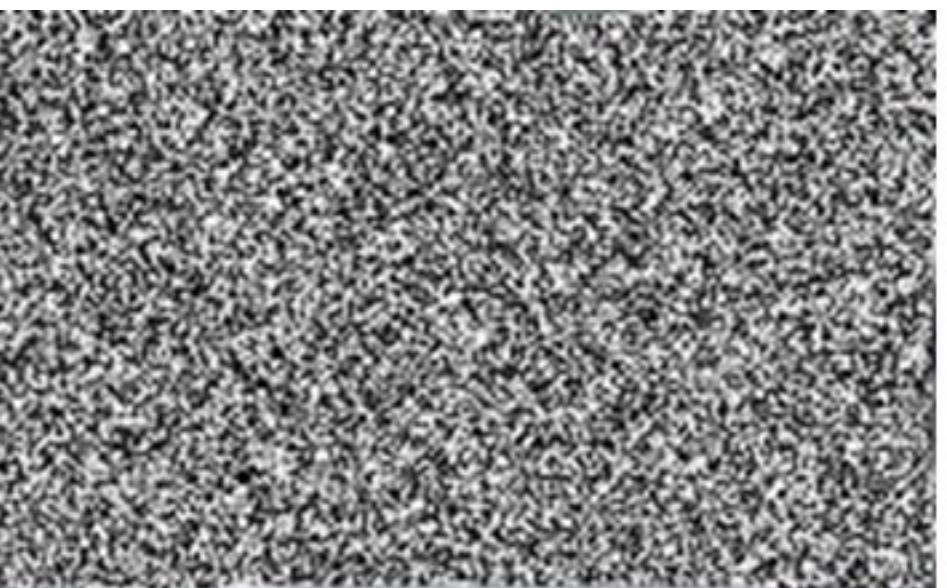
- Based on the fundamental particle tracing algorithm
 - Path lines / surfaces
 - Time lines / surfaces
 - Stream lines / surfaces
 - Ribbons
 - Tubes
 - and many more ...

Examples



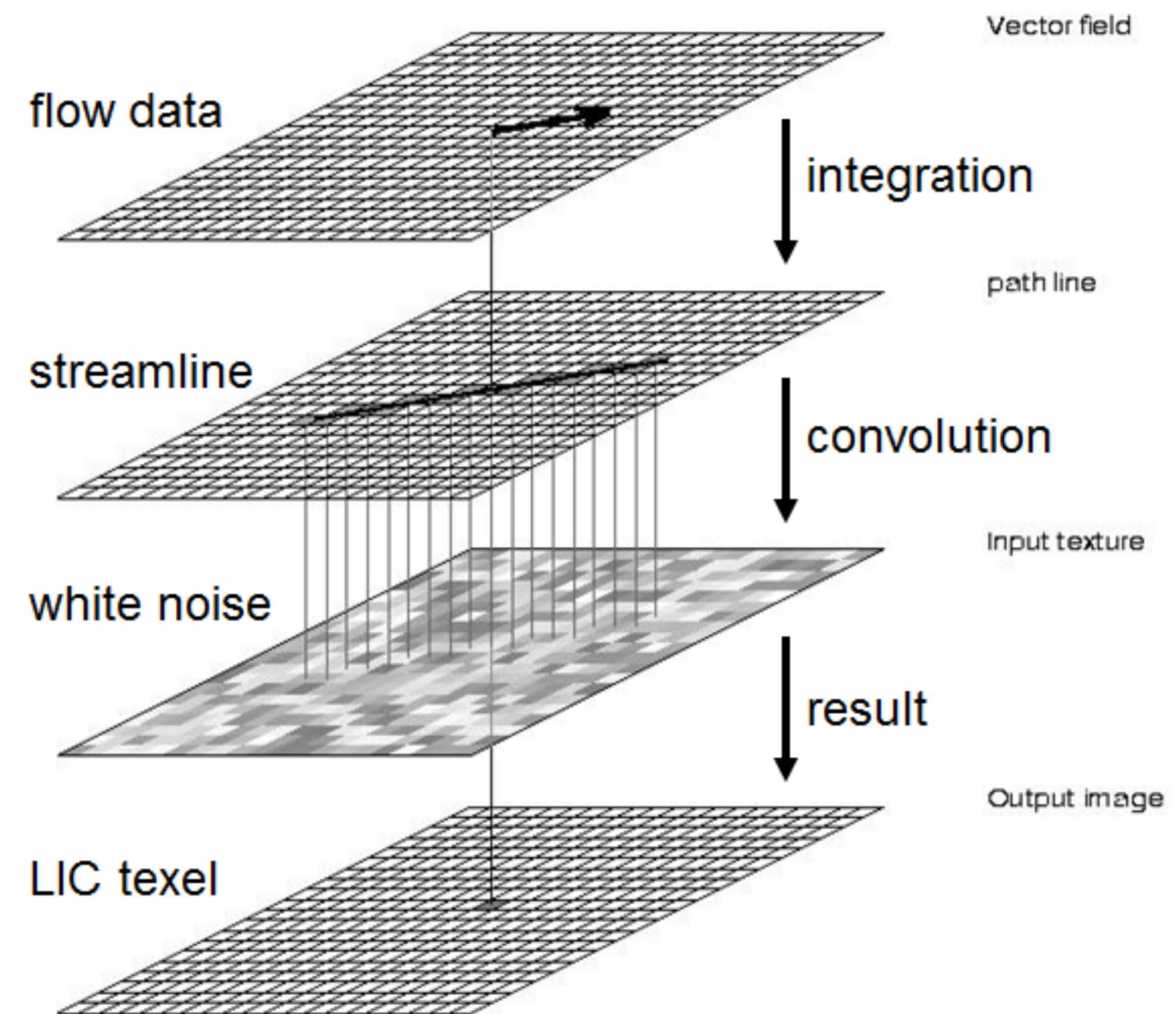
5.4 Line Integral Convolution (LIC)

- Main Idea
 - Traditional approach inappropriate for dense vector fields
 - "Image" integral curves
 - Cover domain with random texture (input texture)
 - Stationary white noise
 - Convolve (blur) input texture along the path lines



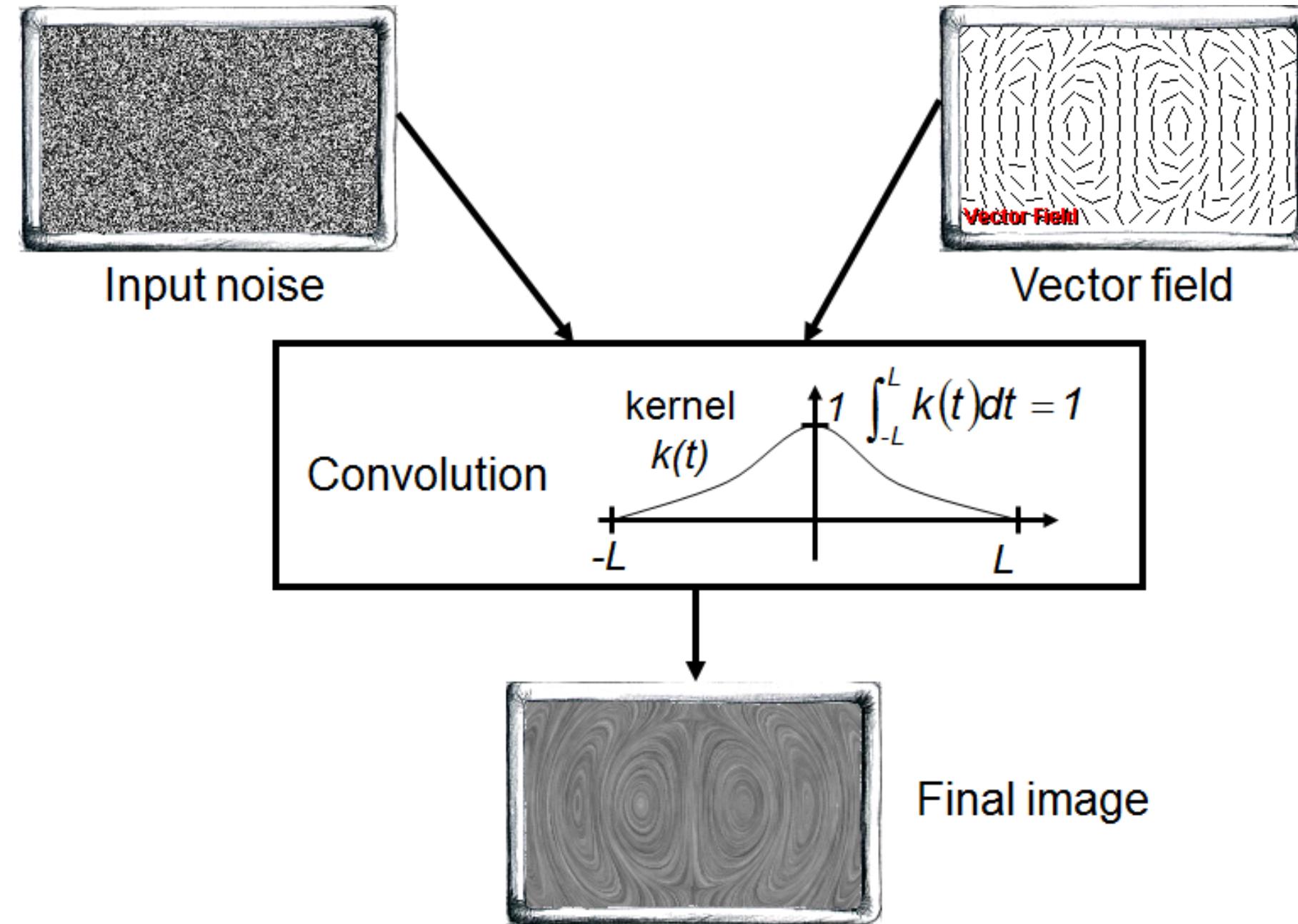
LIC

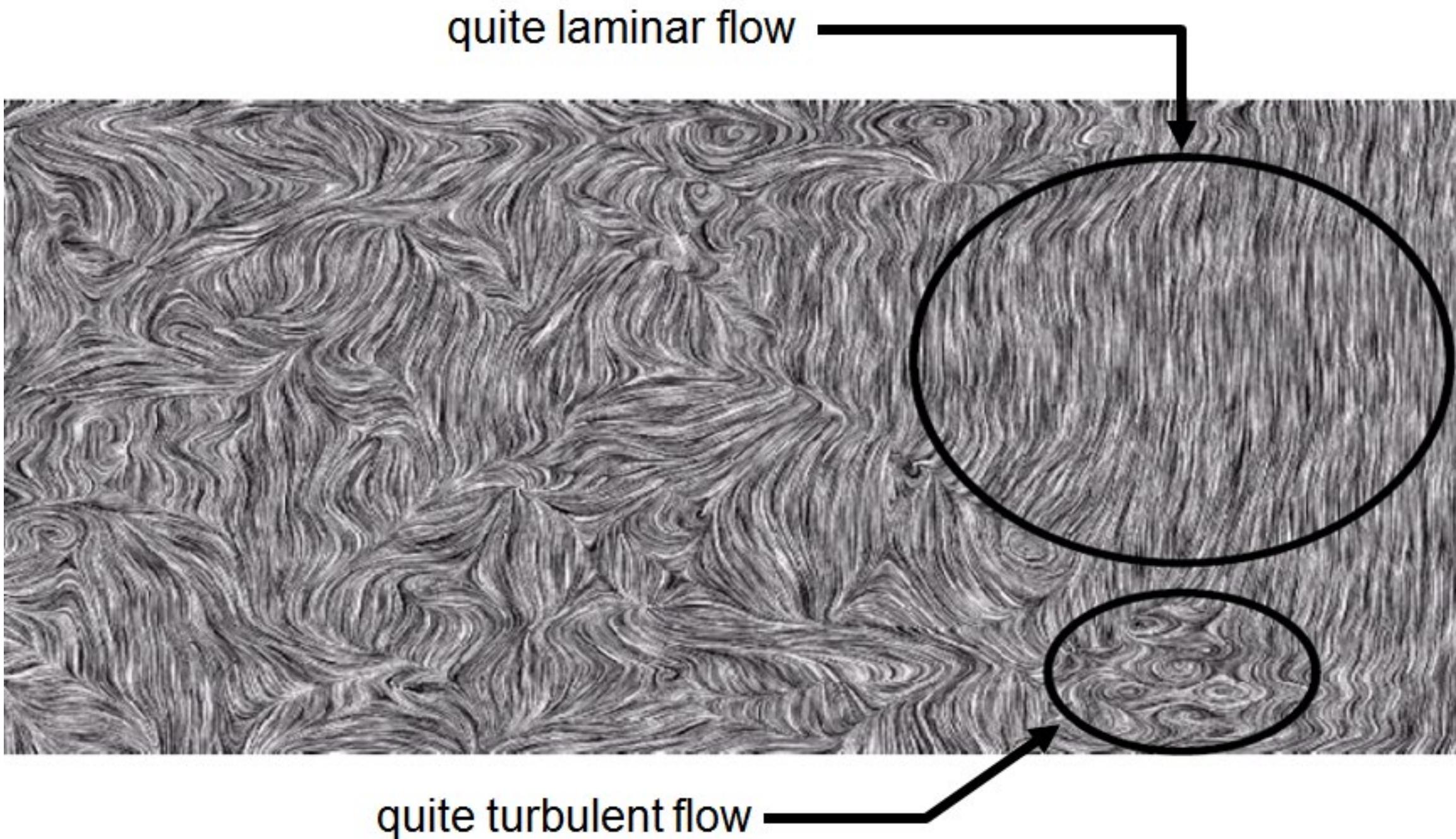
- Algorithm for 2D LIC
 - Convolve random texture along the streamlines



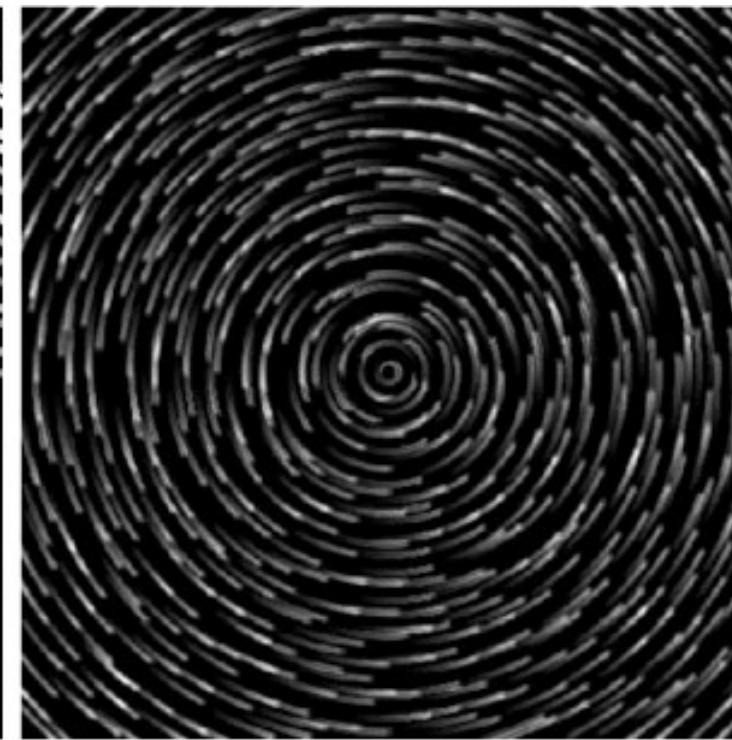
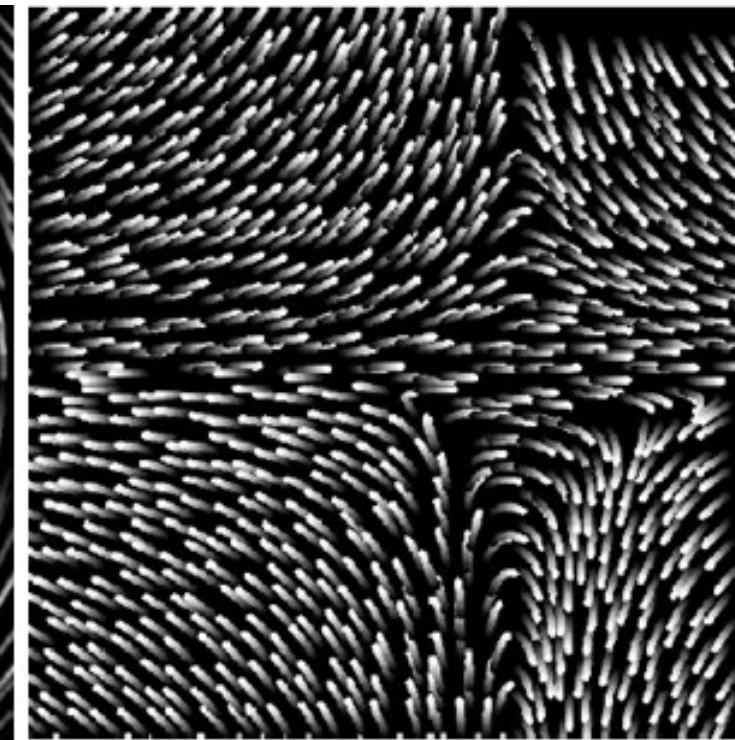
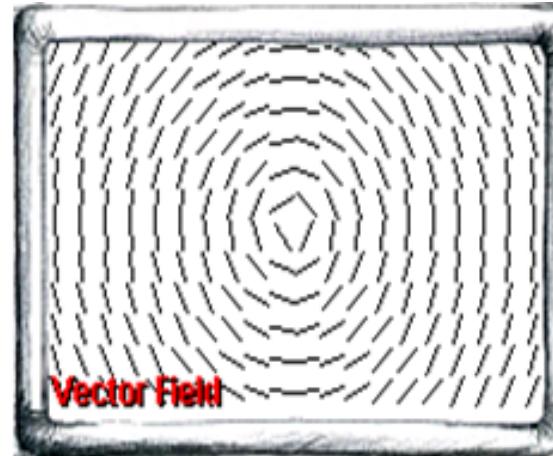
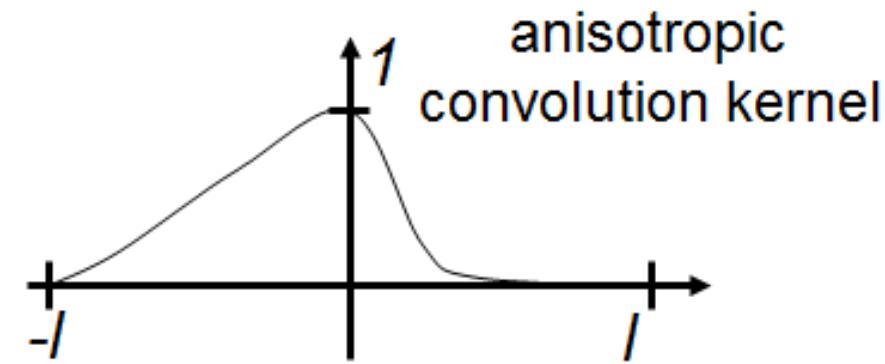
LIC

Idee: Bildfaltung

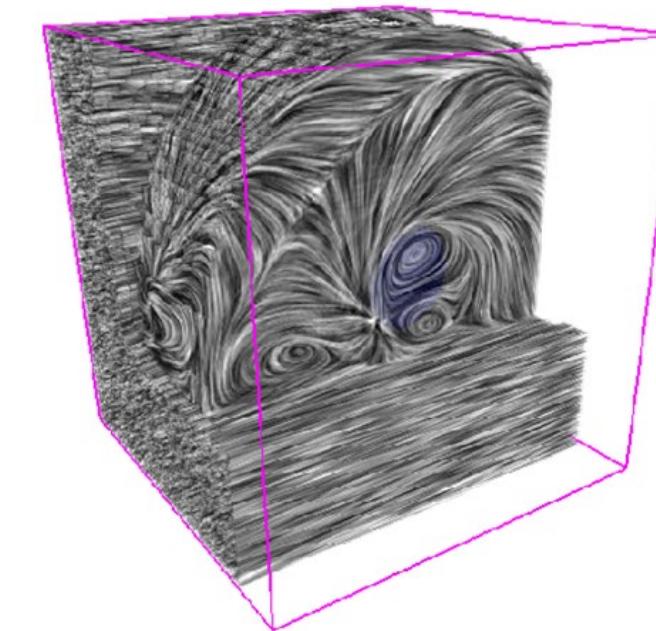
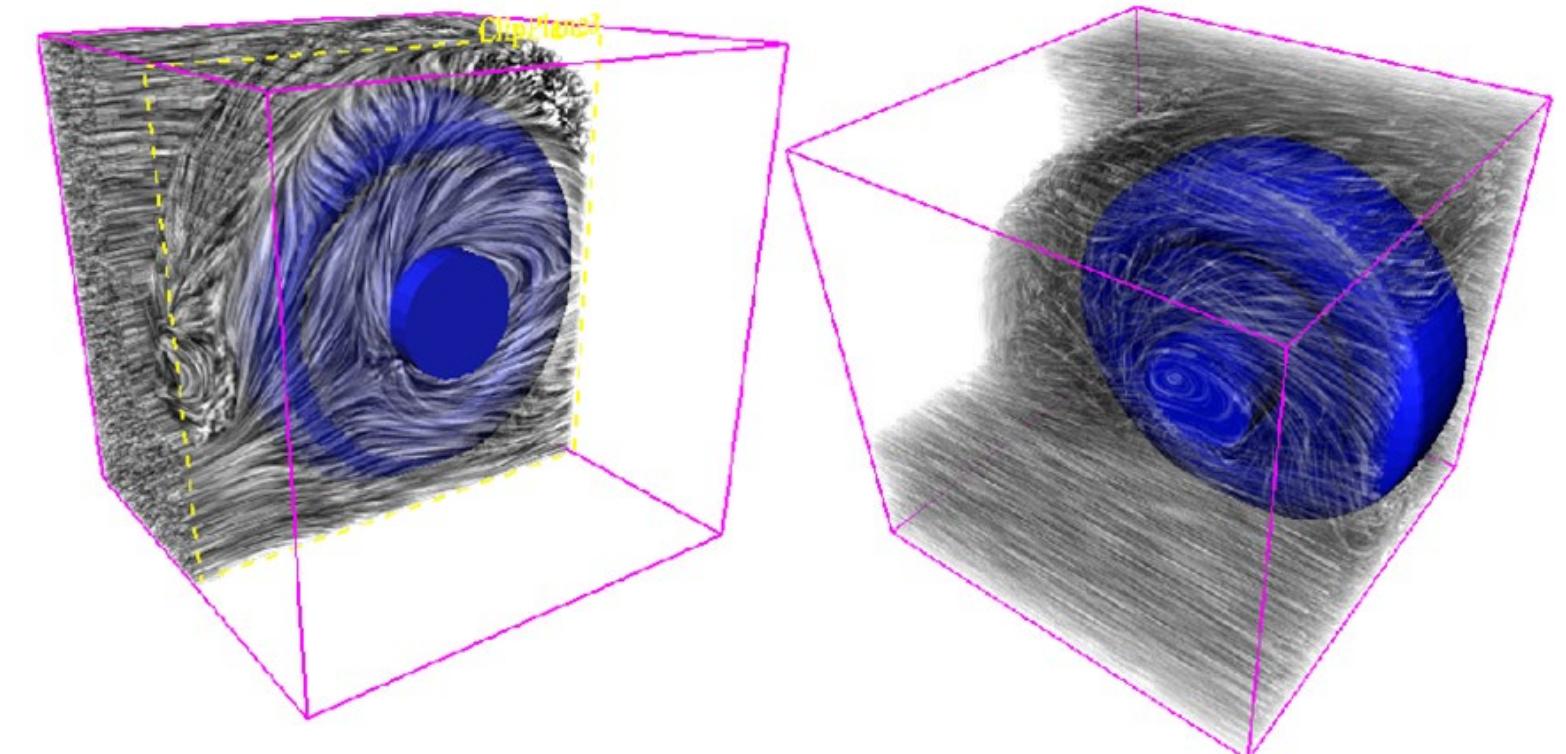
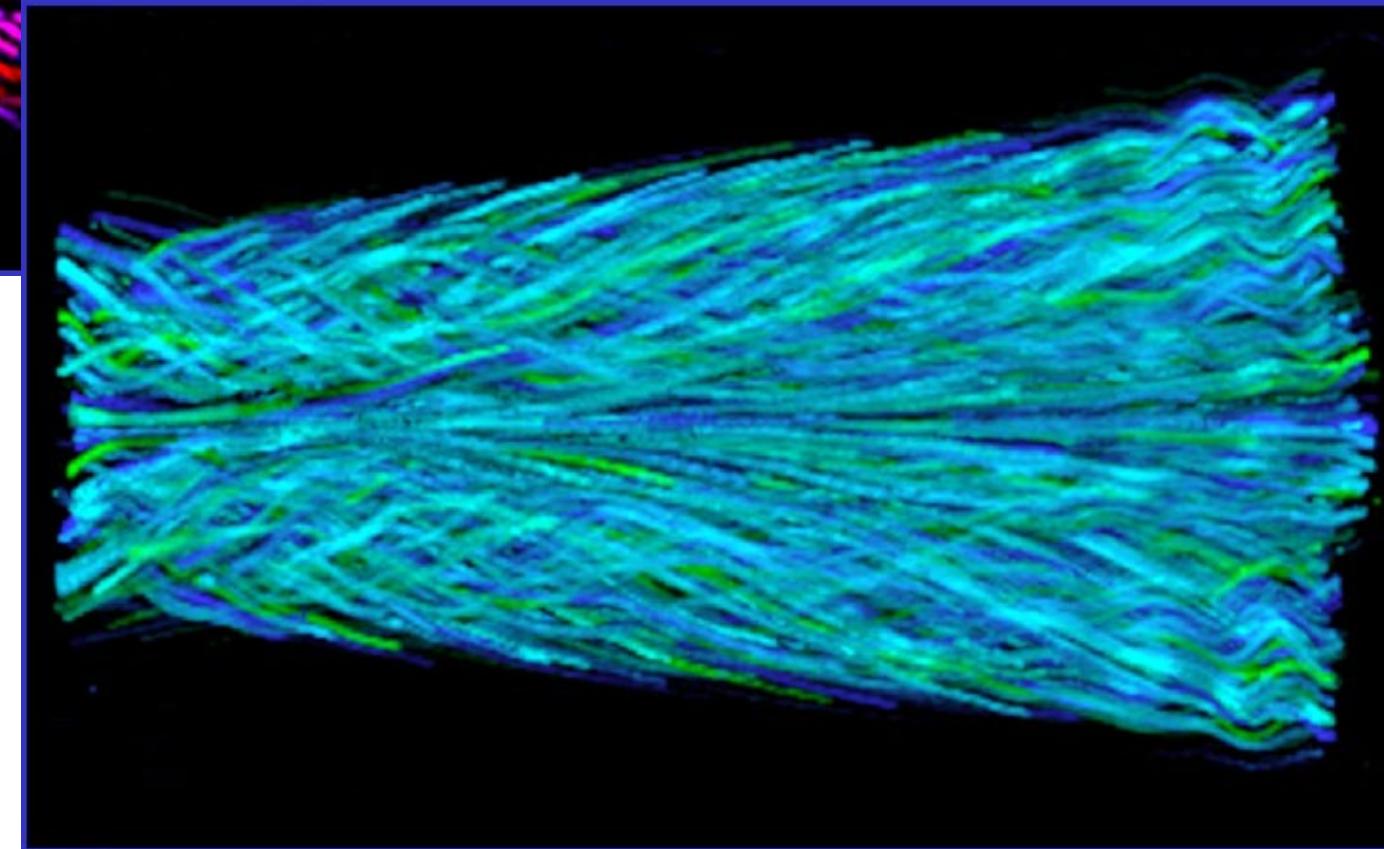
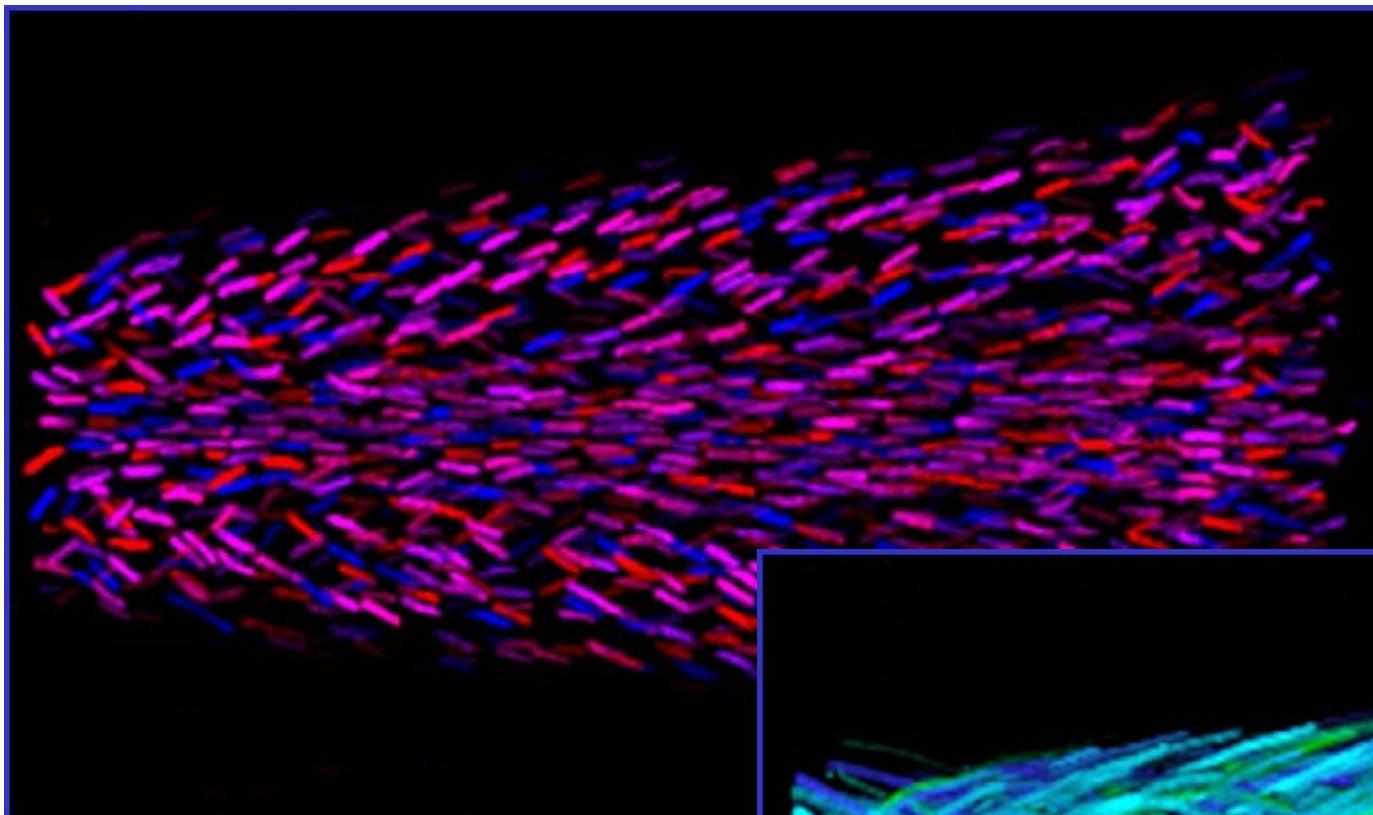




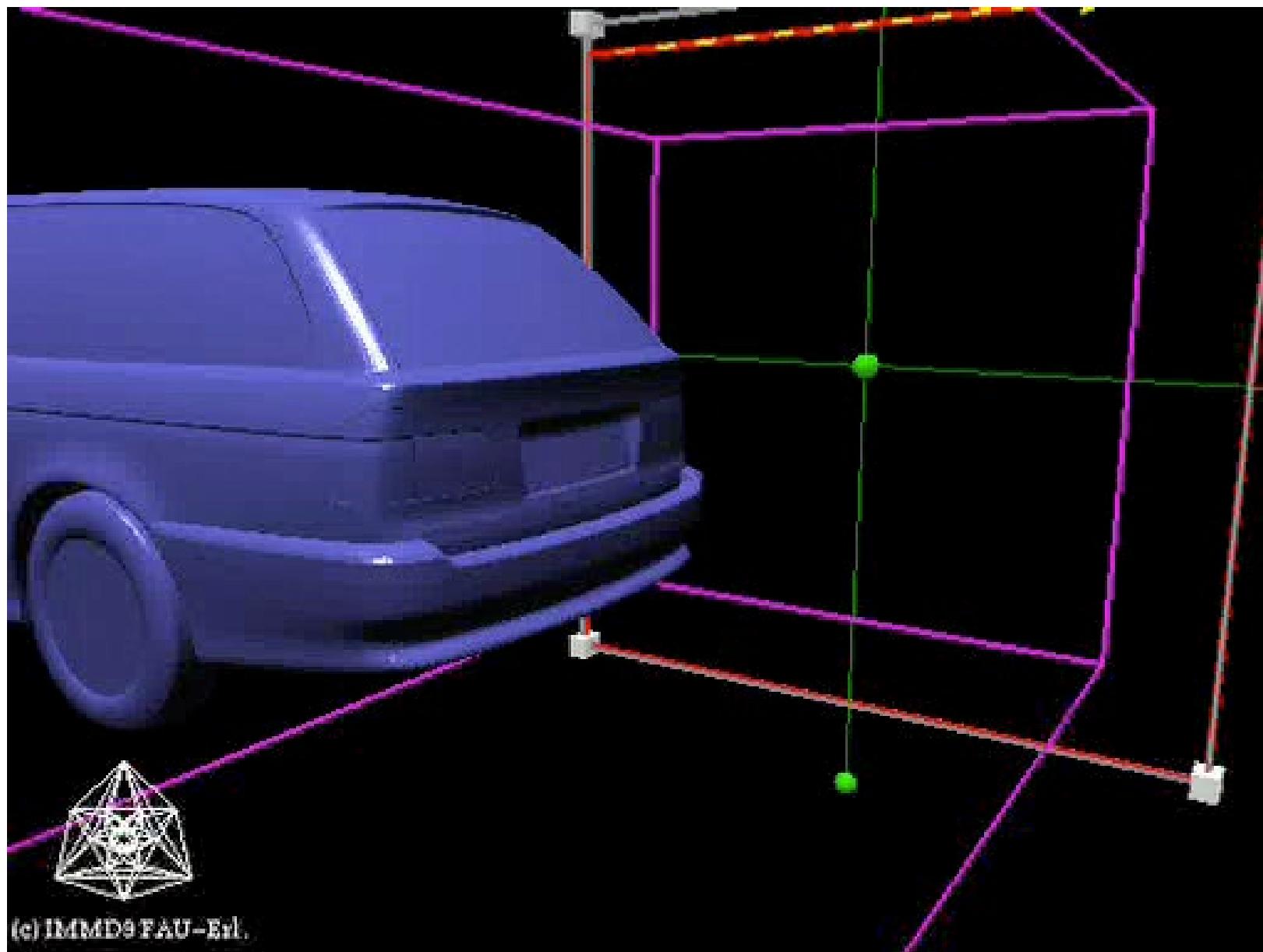
Oriented LIC (OLIC)



3D LIC



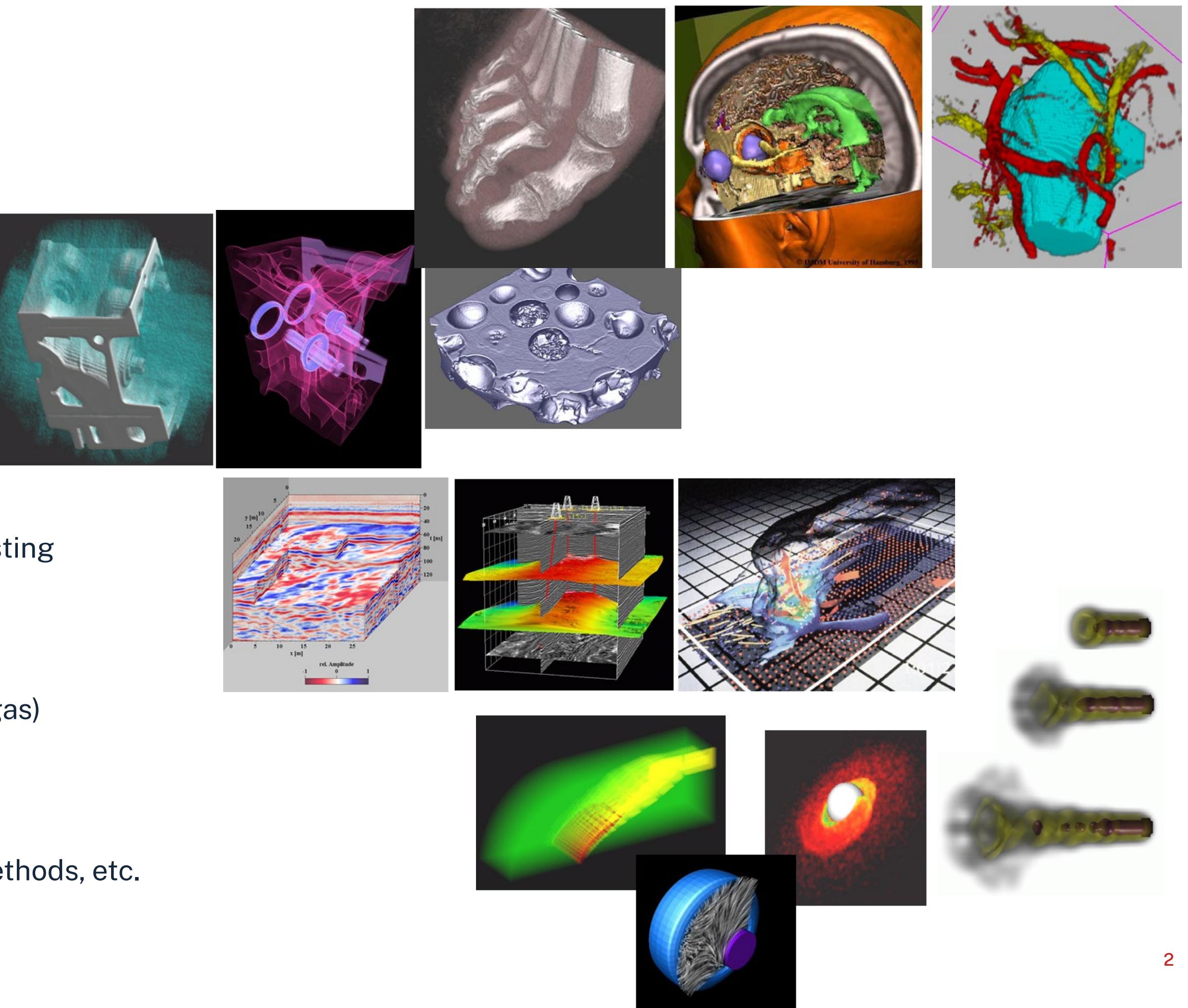
3D LIC



6. 3D Scalar Fields

Introduction

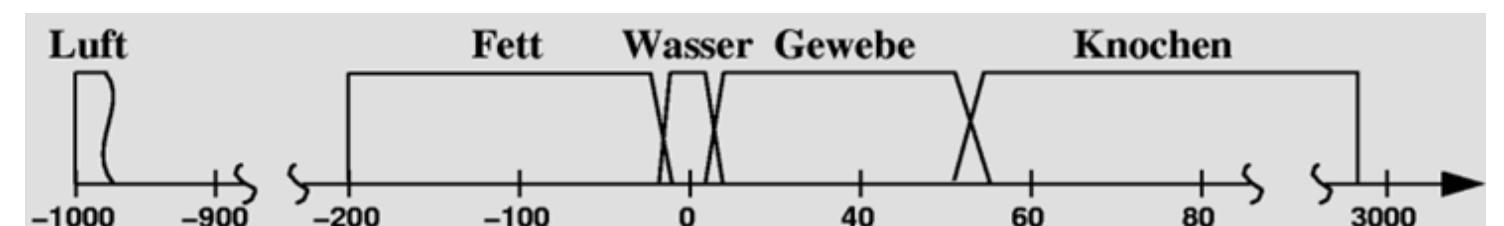
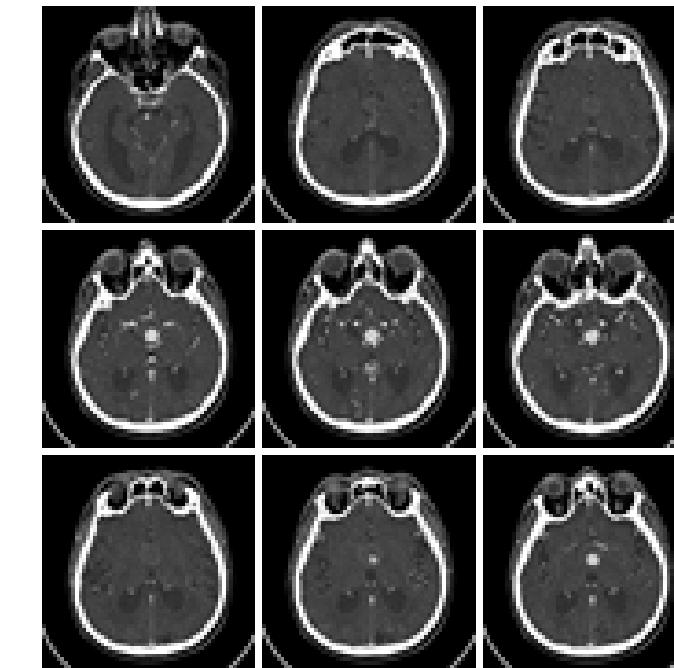
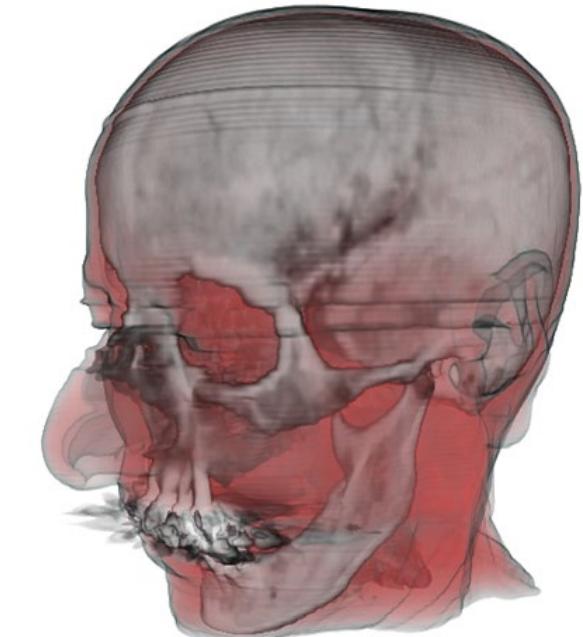
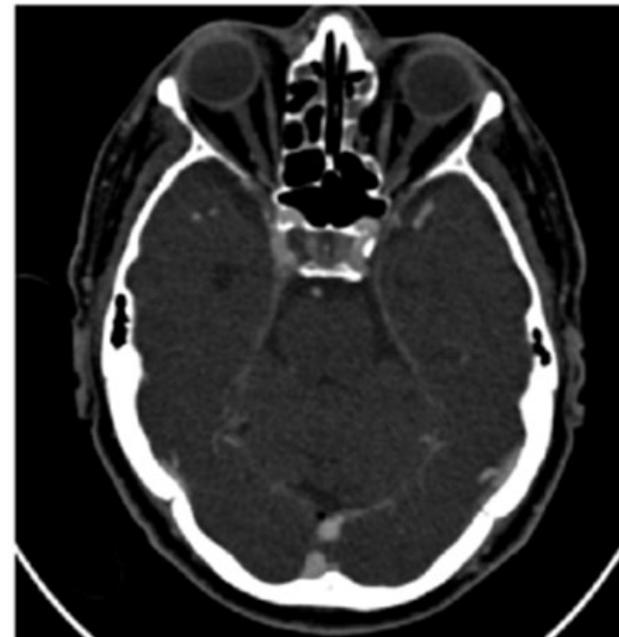
- Medical Applications
 - CT (Computed Tomography)
 - MRI (Magnetic Resonance Imaging)
 - Confocal Microscopy
 - 3D Ultrasound (US is usually 2D)
- Materials sciences
 - Industrial CT
 - Quality control, non-destructive testing
 - E.g. statue, engine block, wheels, etc.
- Geosciences
 - Geoseismic data
 - Exploration of natural resources (oil, gas)
 - Meteorological data
- Numerical simulations
 - Particle simulations
 - Finite element or finite differences methods, etc.



Introduction

Example: typical medical volume data

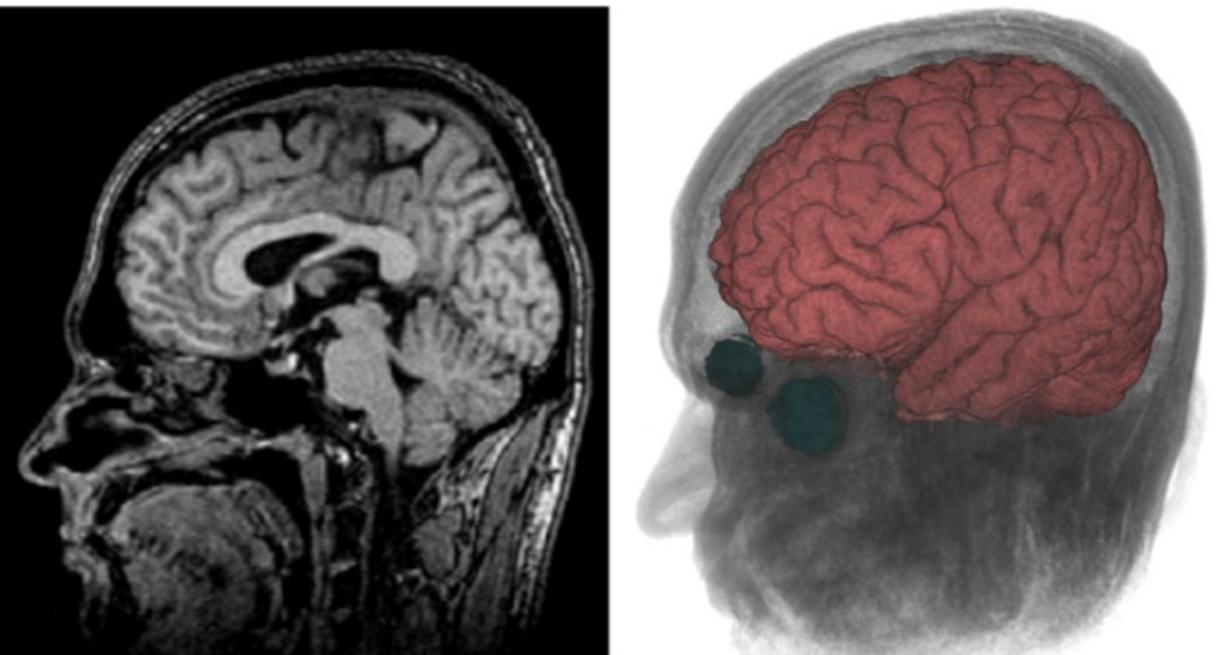
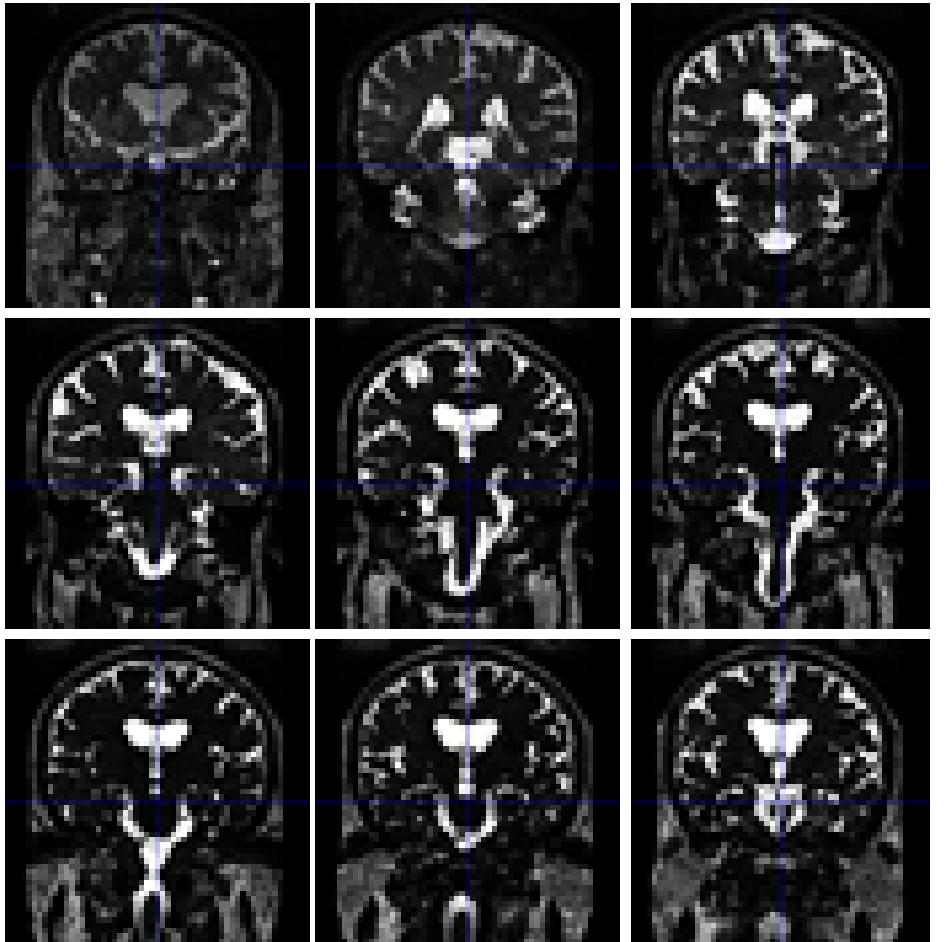
- Computed Tomography (CT)
 - Stack of slice images
 - Number of slice images: 100 - 2000
 - Typical image size: 512 x 512 pixels
 - Data representation: 12 or 16 bits
 - Typical voxel sizes: 0.4x0.4x0.4mm³, 0.7x0.7x1.2mm³
 - Representations (based on X-ray projections)
 - Good contrast between air, fat, muscle, bone
 - Metal objects are "bad" but possible
 - Hounsfield scale: -1000 - 3095 (12 bits)
 - Direct correlation between data values and tissue types



Introduction

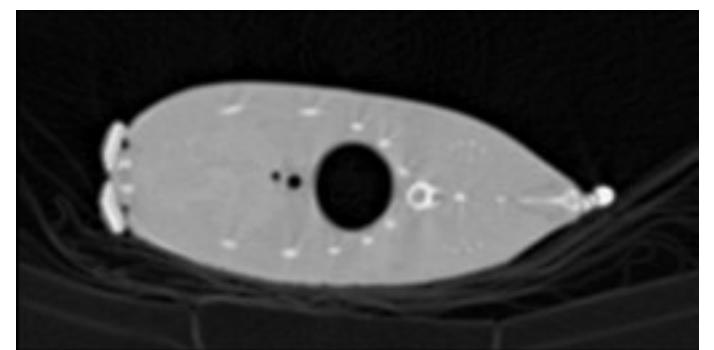
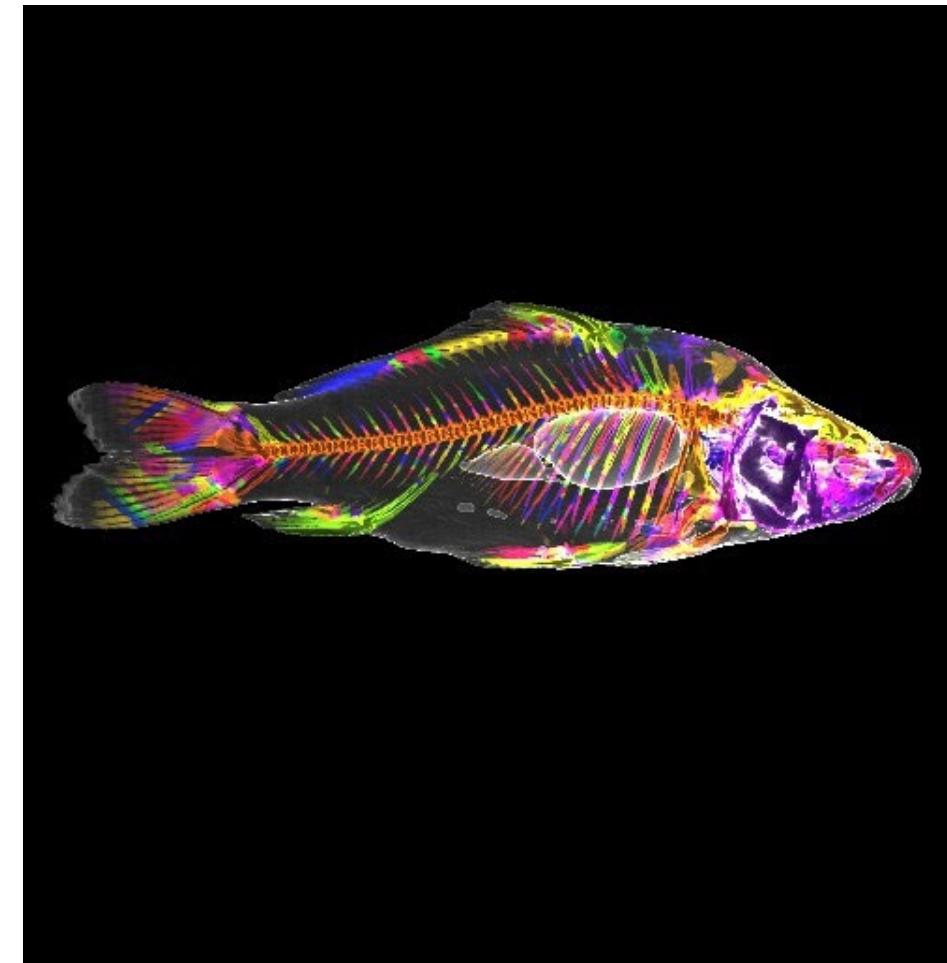
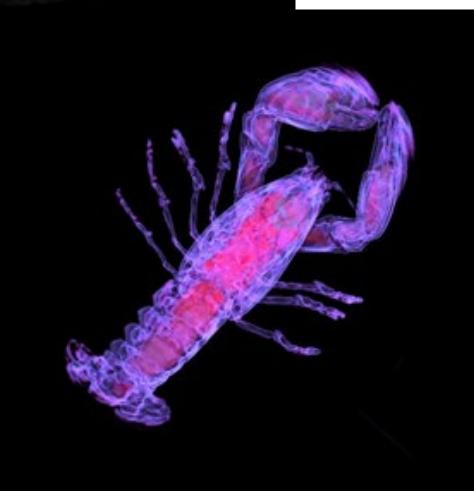
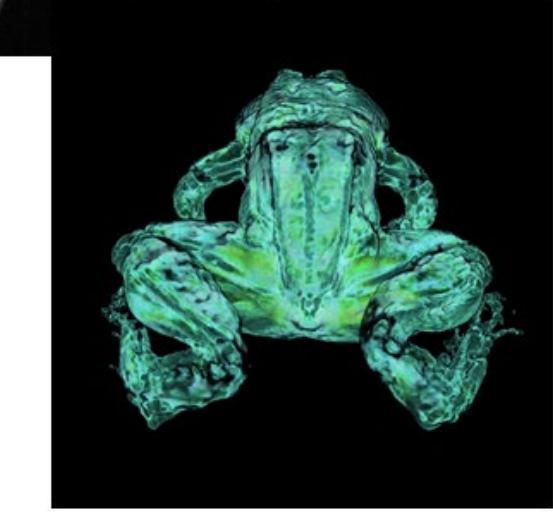
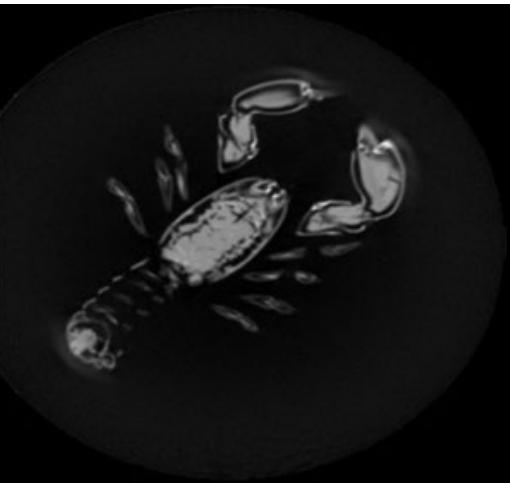
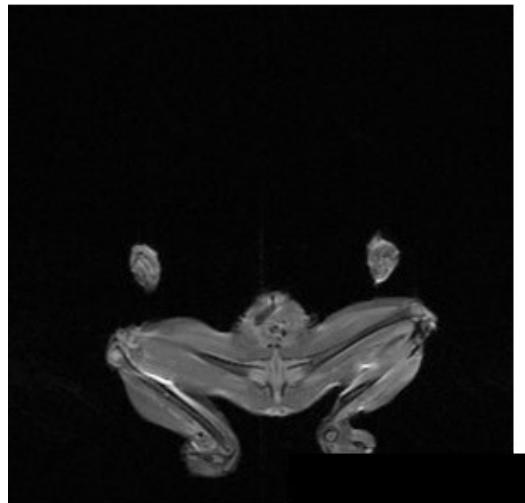
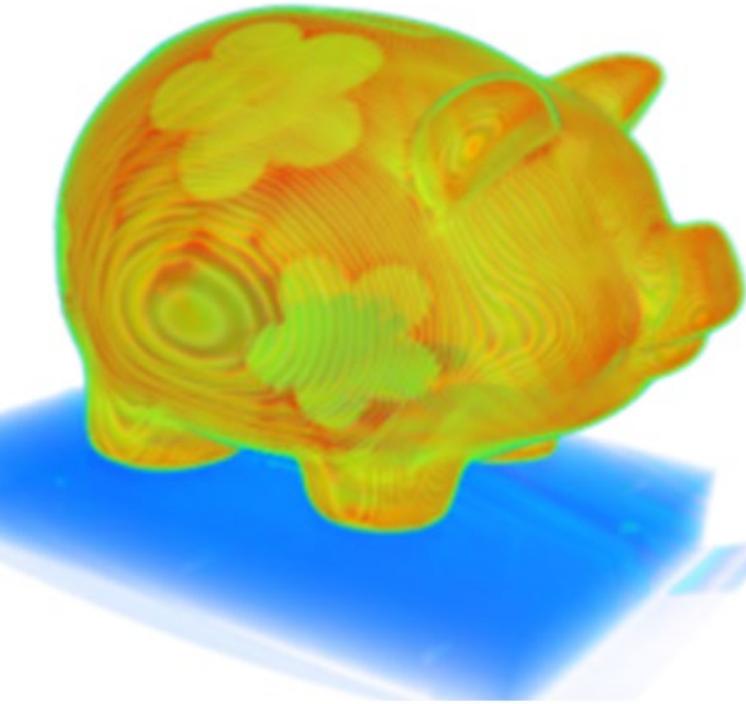
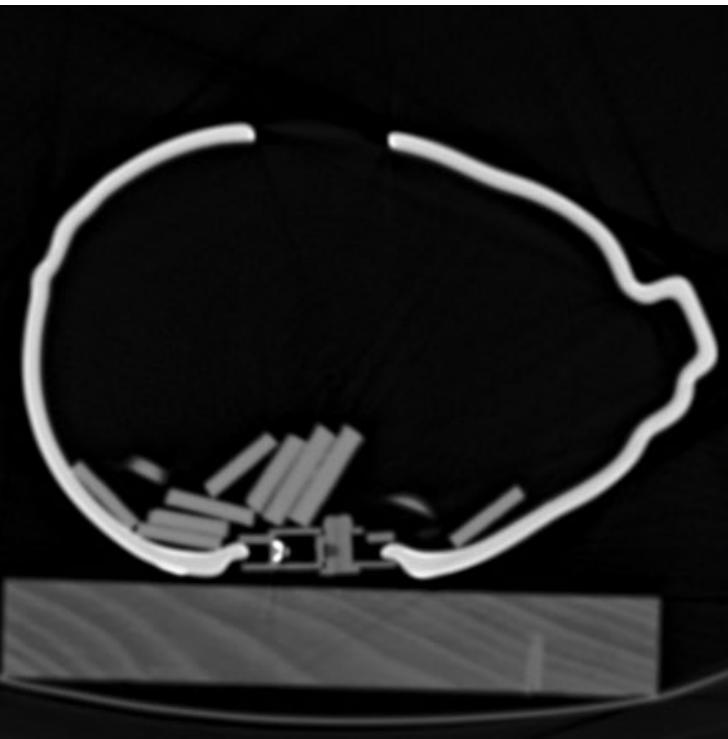
Example: typical medical volume data

- Magnetic Resonance Tomography (MRT)
 - Stack of slice images
 - Number of slice images: 40 - 250
 - Typical image size: 512x512 pixels
 - Data representation: 12 or 16 bits
 - Typical voxel sizes: 0.7x0.7x0.7mm³, 1.0x1.0x1.7mm³
 - Representations (no ionizing radiation!)
 - Better contrast for soft tissue, bone is dark
 - Metal object scans are "not possible"
 - Nothing comparable to Hounsfield scale
 - No direct correlation between data values and tissue types



Introduction

More examples



6.1 Volume Visualization

Volume Visualization

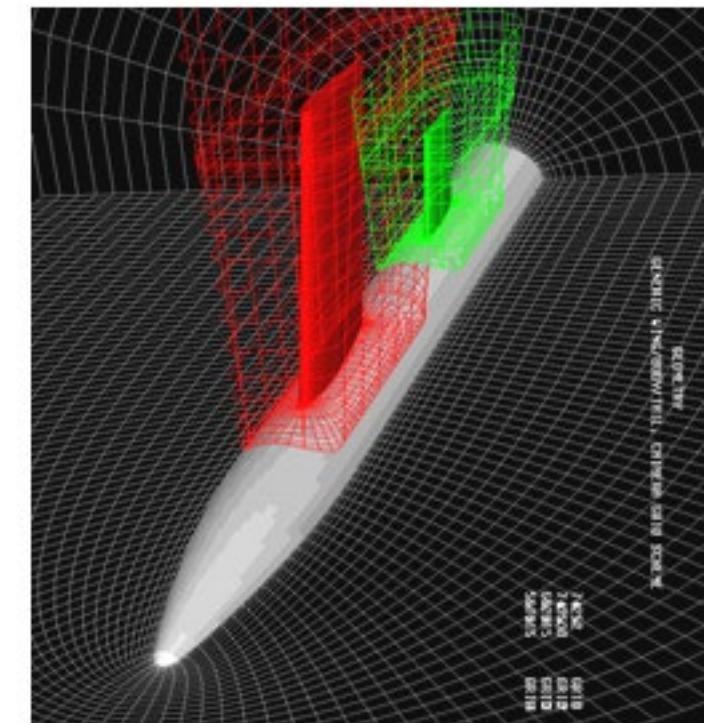
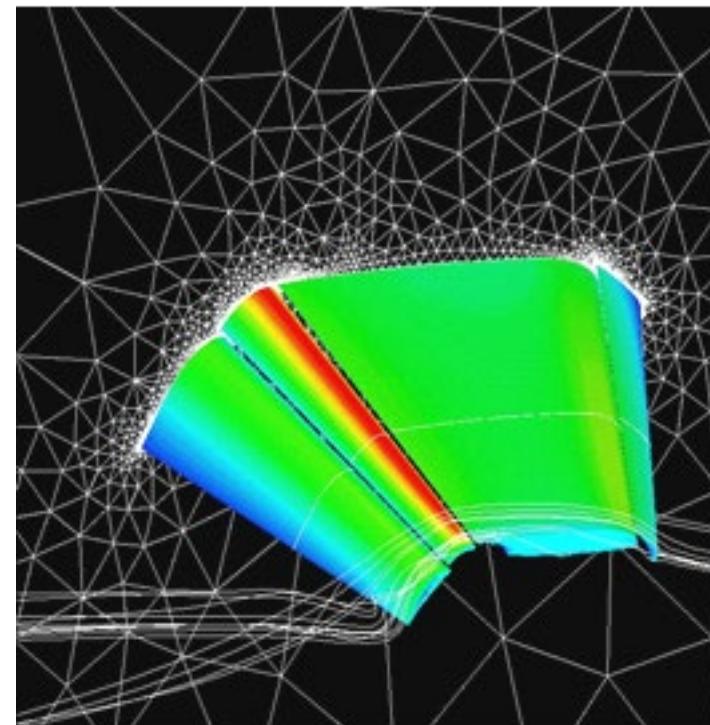
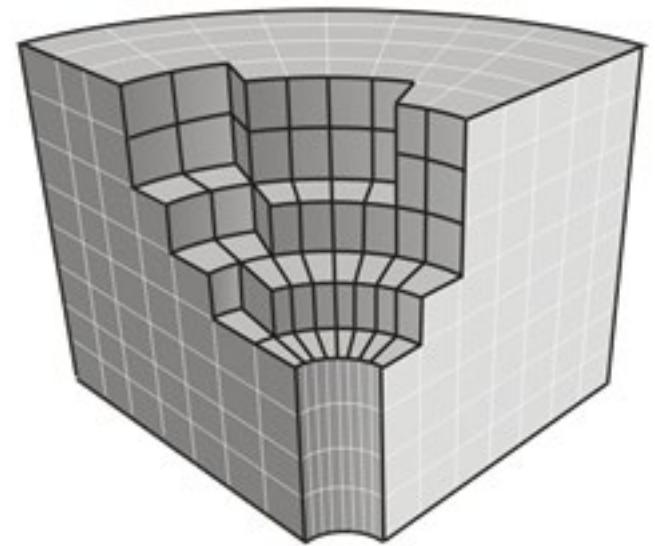
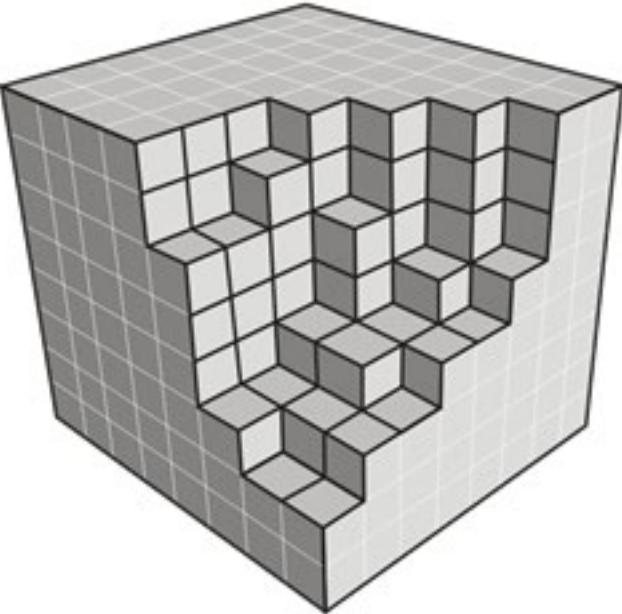
Possible characteristics of volume data

- Essential information in the interior
- Cannot be described by geometric representation
 - Fire, clouds, gaseous phenomena
- Distinguish between shape
 - Given by the geometry of the grid
- and appearance
 - Given by the scalar values
- Even if the data could be described geometrically, there are, in general, too many primitives to be represented

Volume Visualization

Grid structures

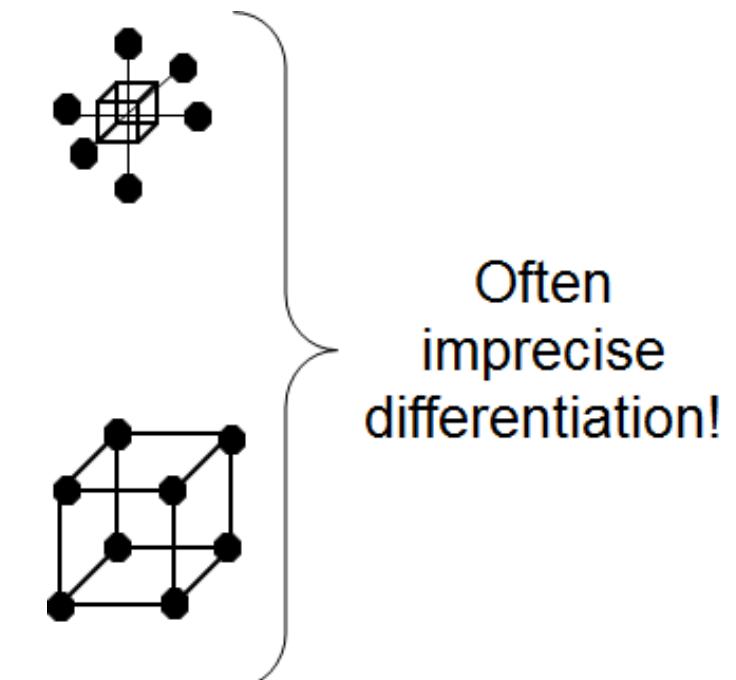
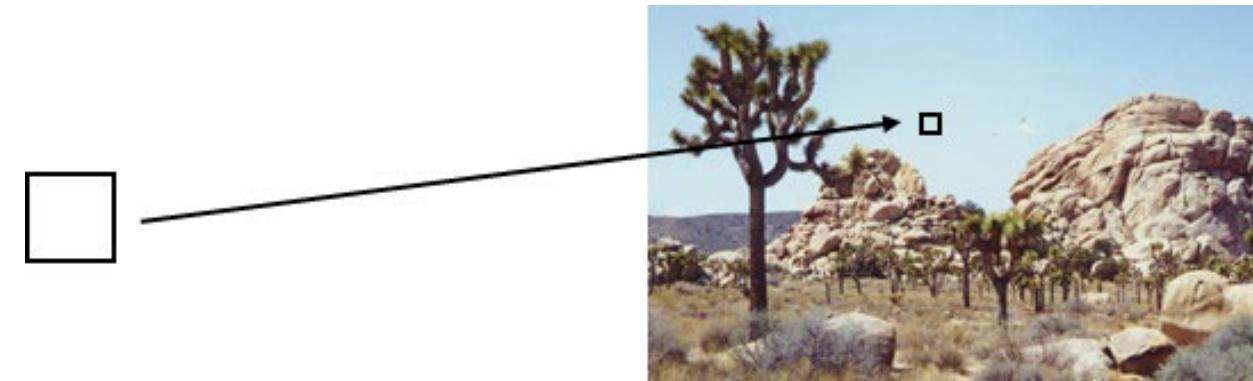
- Structured
 - Uniform
 - Rectilinear
 - Curvilinear
- Unstructured
 - Tetrahedral
 - Mixed elements
 - Hexahedron
 - Tetrahedron
 - Prism
 - Pyramid
 - Scattered data



Volume Visualization

Definitions (most common for structured uniform grids)

- Pixel
 - "Picture element"
- Voxel
 - "Volume element"
 - Values are constant within a region around a grid point
- Cell
 - Values between grid points are resampled by interpolation



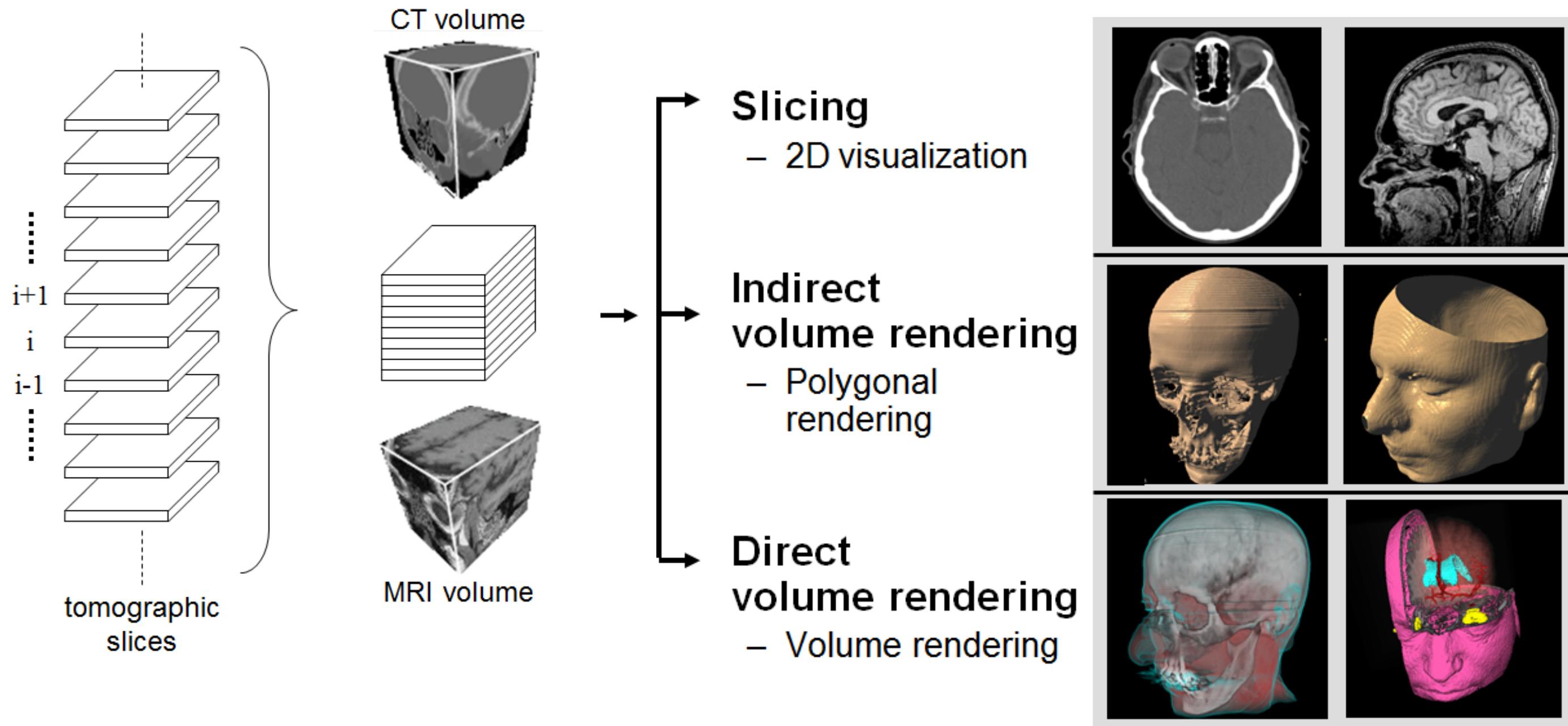
Volume Visualization

Main visualization approaches

- Slicing
 - Standard 2D approach
 - Techniques for 2D scalar fields
- Indirect volume rendering
 - Convert/reduce volume data to surface representation
 - Rendering with traditional techniques
- Direct volume rendering
 - Take volume data as real volumetric object
 - Consider as light-emitting, semi-transparent gel
 - Directly get a 3D representation of the volume data

Stichwort: Isoflächen

Volume Visualization



Volume Visualization

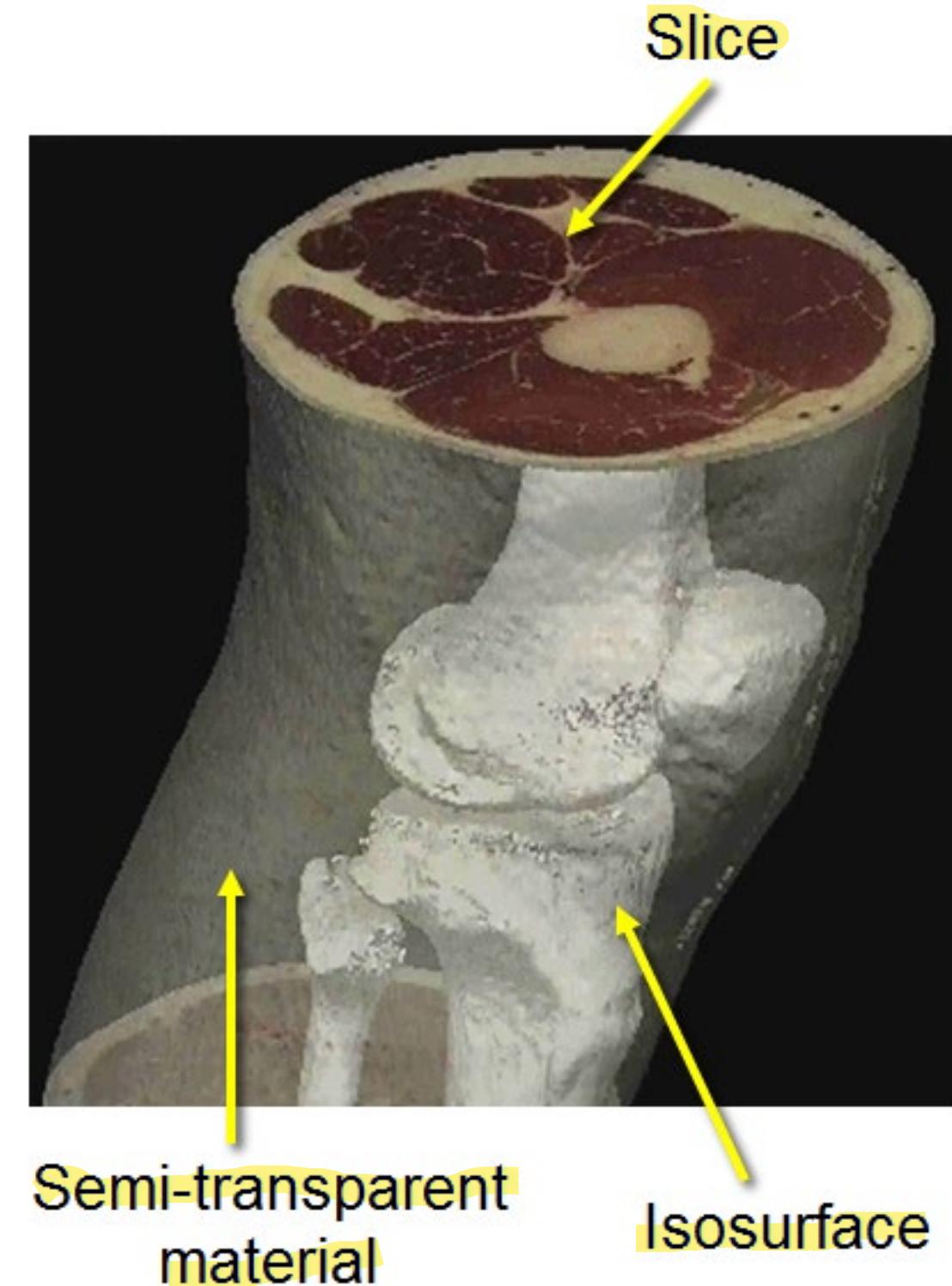
Considerations

- Indirect volume rendering techniques
 - Often result in complex representations
 - Preprocessing of the surface representation might help
 - Standard graphics hardware for interactive display
- Direct volume rendering techniques
 - "Global" representation integrating physical characteristics
 - Interactive display difficult due to numerical complexity
 - Use graphics hardware for acceleration
- Goal
 - Integration of different techniques for optimal display
 - The most correct method in terms of physical realism may not be the most optimal one in terms of understanding the data

Volume Visualization

Example

- Slicing
 - Display the volume data, mapped to colors, on a slice plane
- Iso-surfacing
 - Generate opaque / semi-opaque surfaces
- Transparency effects
 - Volume material attenuates reflected or emitted light

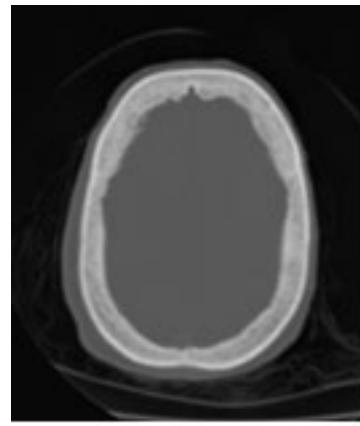


6.2 Slicing of Volume Data

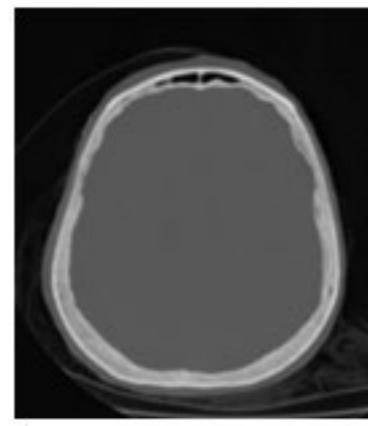
Slicing of Volume Data

Orthogonal slicing

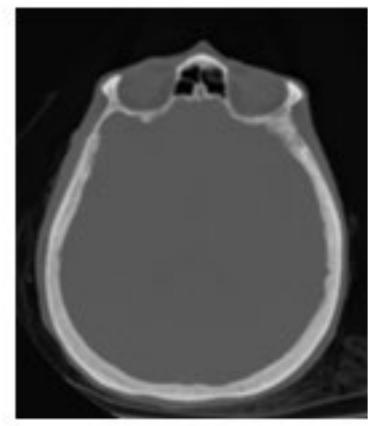
- Resample the volume data on parallel planes perpendicular to the x-, y- or z-axis
 - Example: $z = -25, -24, \dots, 23, 24$
 - Let the user interactively change the z-value



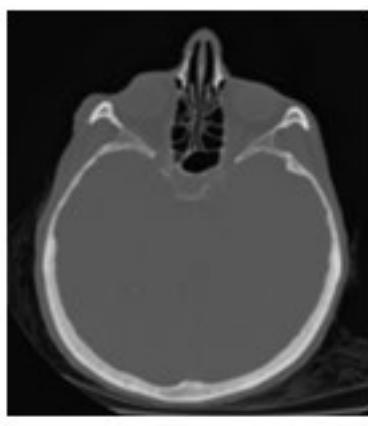
$z = 20$



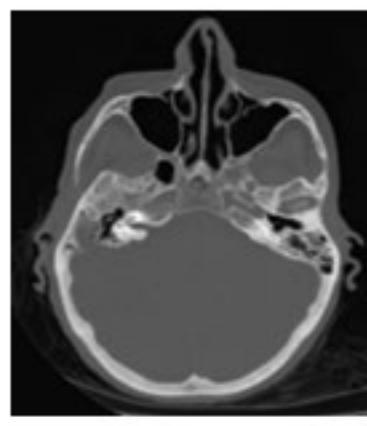
$z = 30$



$z = 40$



$z = 50$

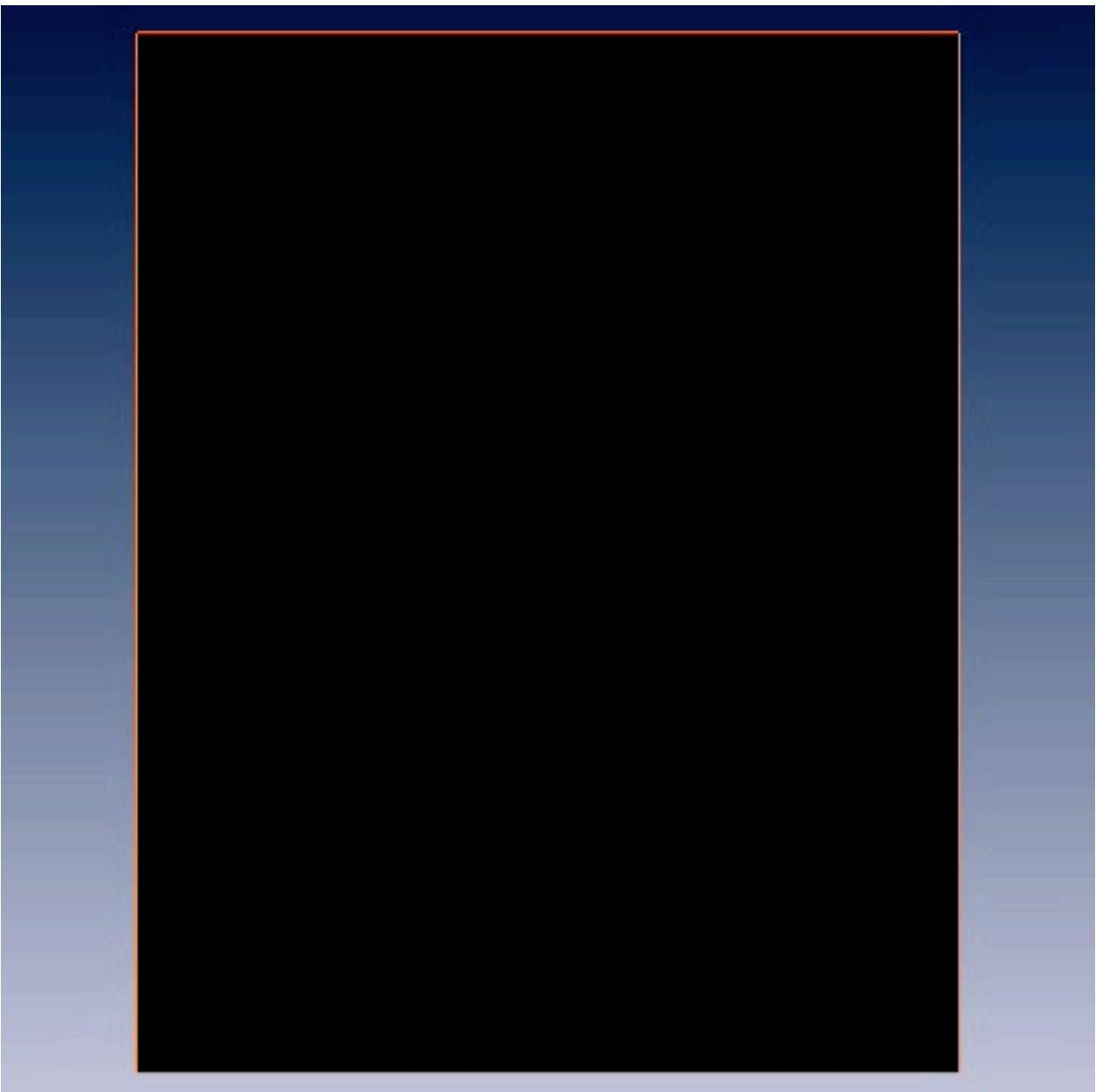


$z = 60$

- Use visualization techniques for 2D scalar fields
 - Color coding, isolines, height fields, ...

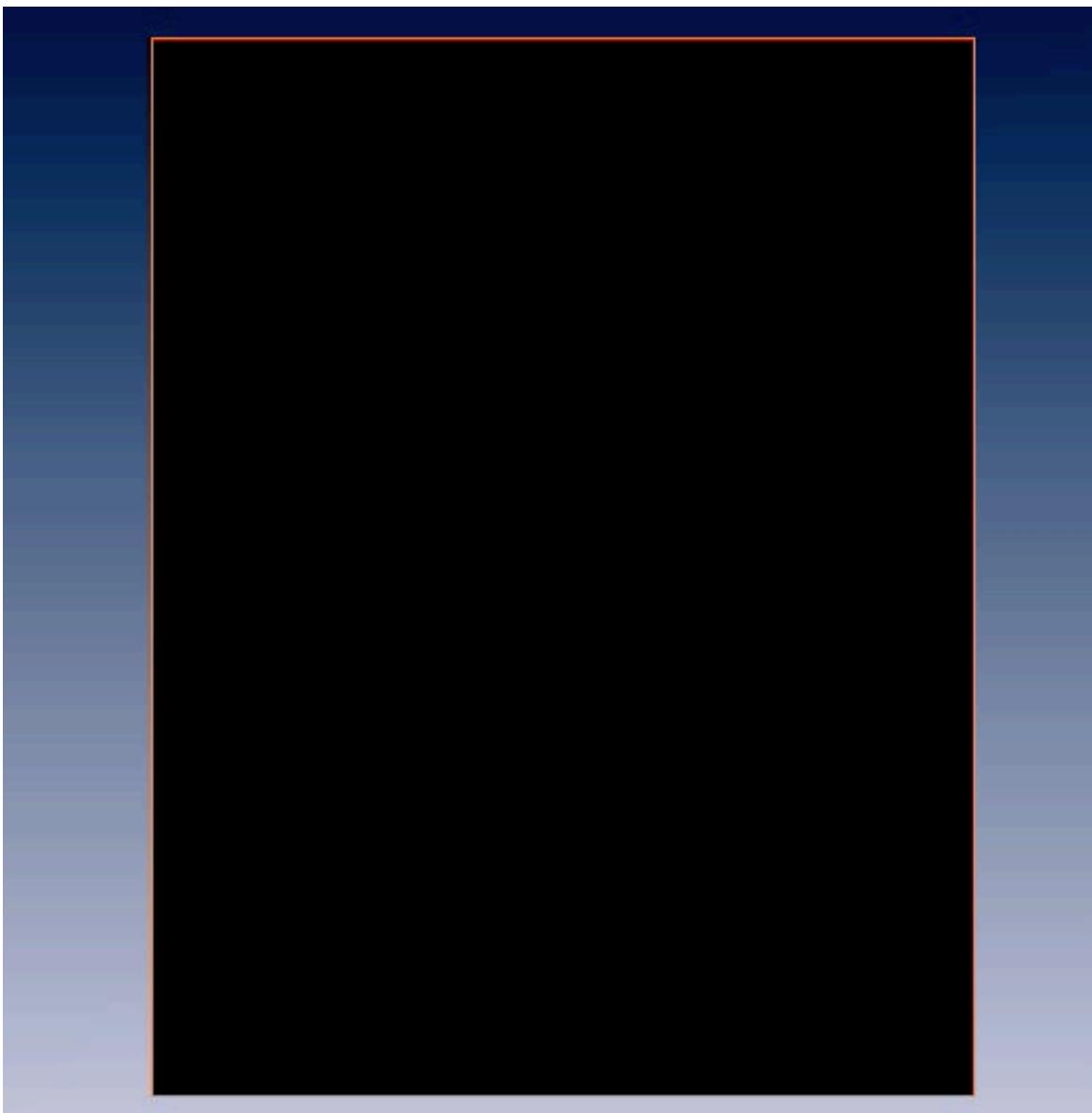
Slicing of Volume Data

- Example: Visible male



Slicing of Volume Data

- Example: Visible male



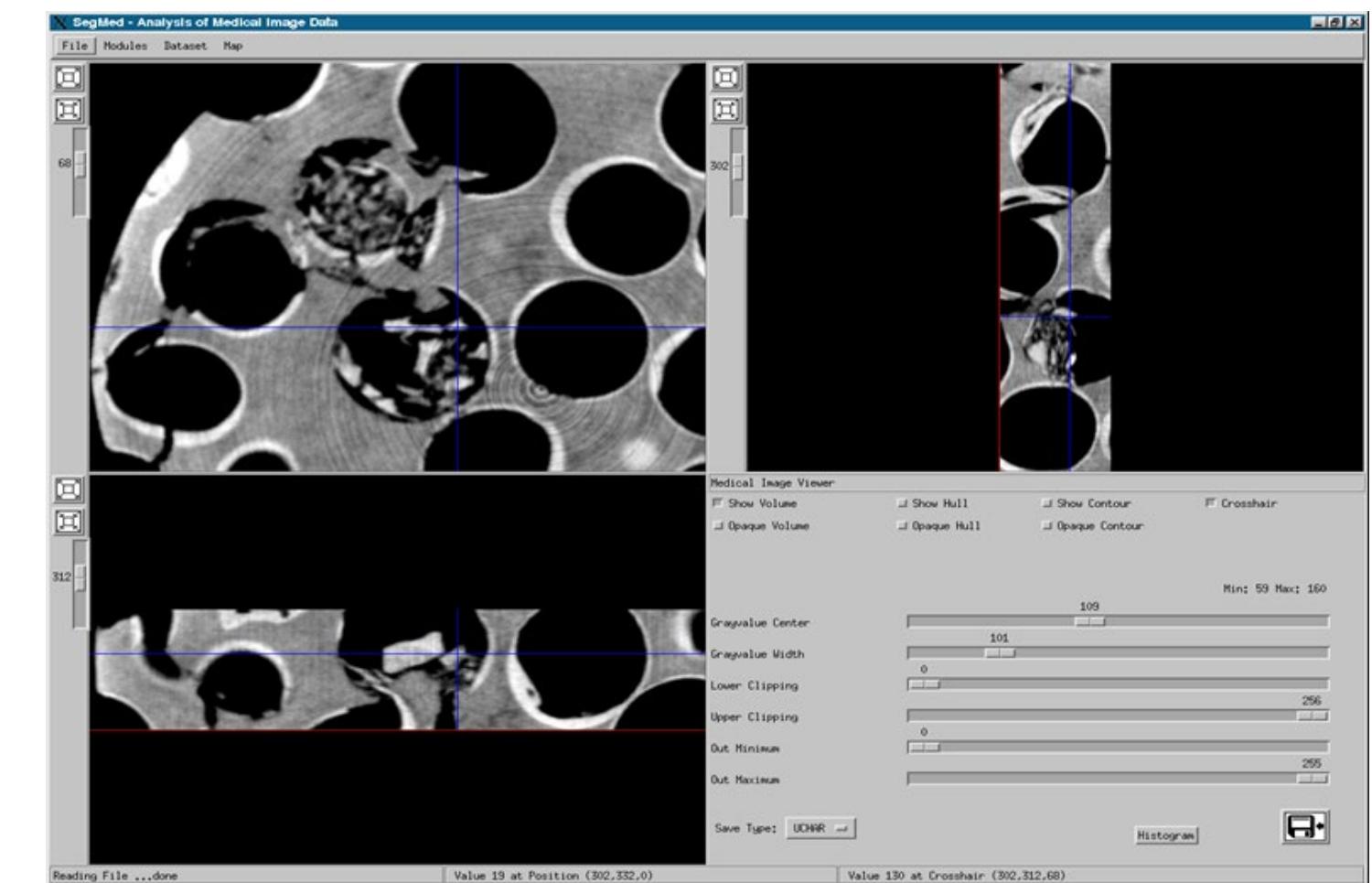
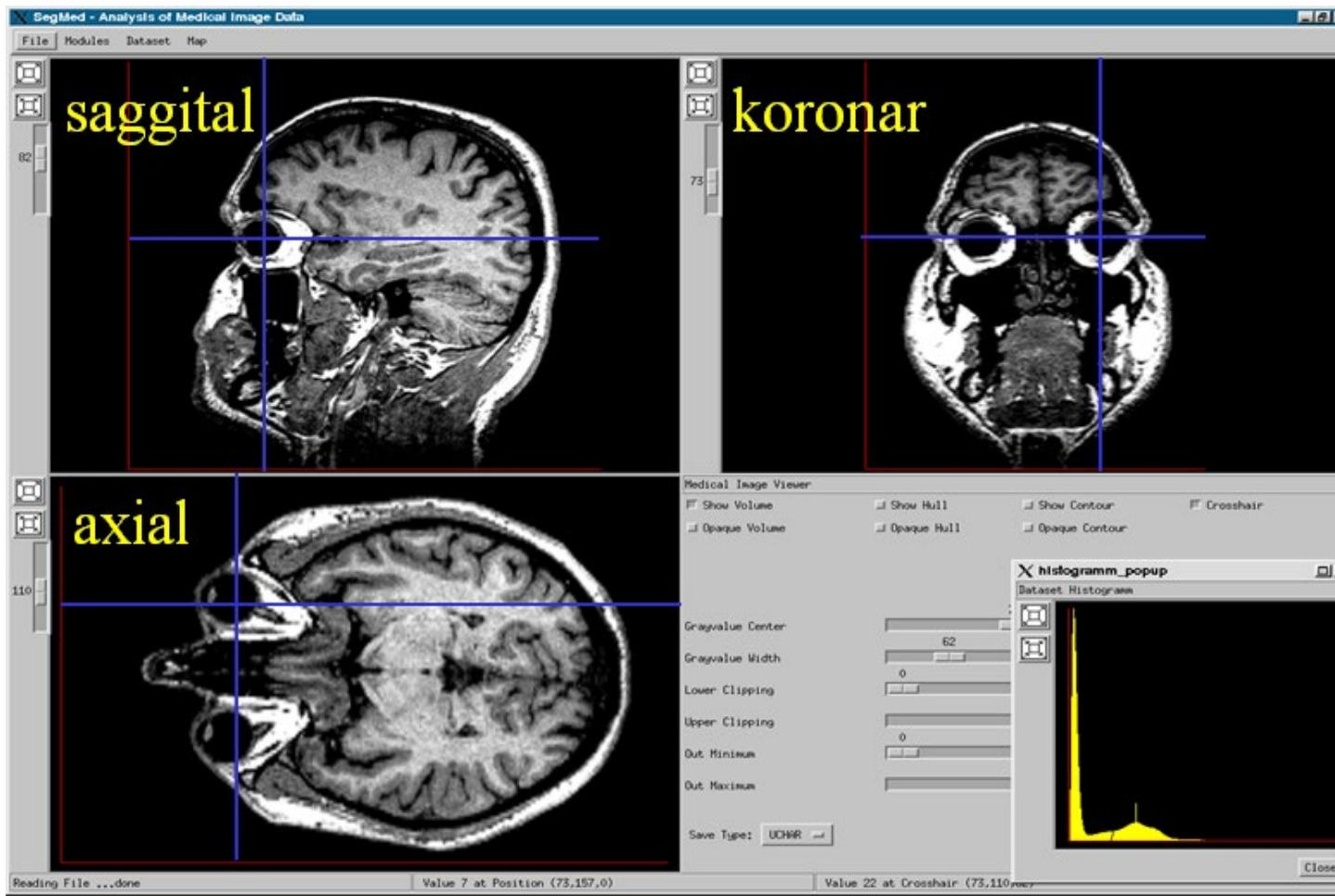
coronar



sagittal

Slicing of Volume Data

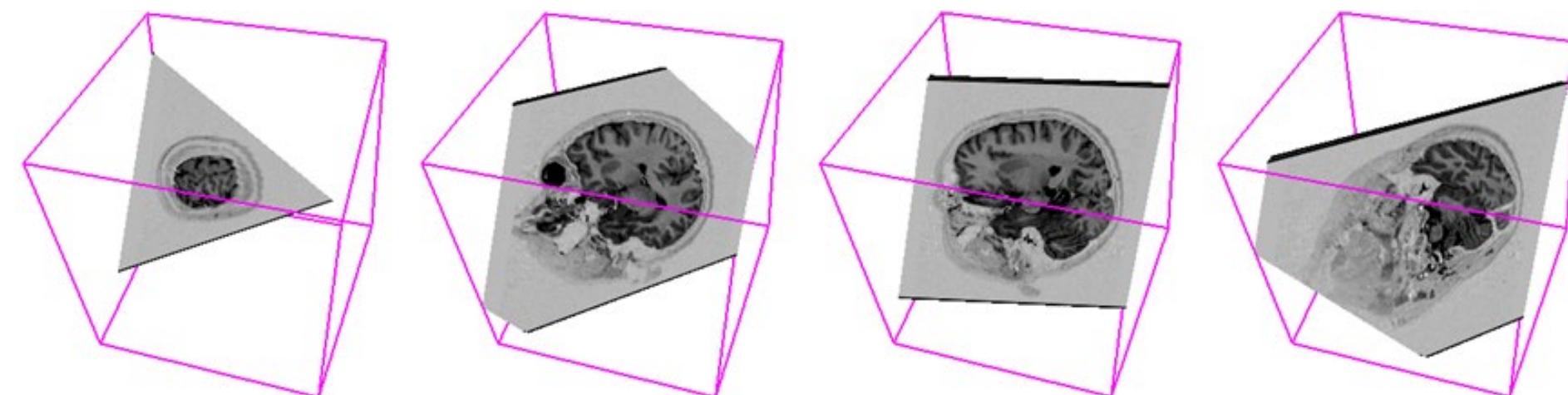
- Simultaneous slicing in x-, y-or z-axis



Slicing of Volume Data

Oblique slicing (MPR - multiplanar reformatting)

- Resample volume data on arbitrarily oriented slices
- Interpolation in software on CPU
- Exploit graphics hardware
 - Store volume data in 3D texture memory
 - Get sectional polygon (plane clipped with bounding box of volume)
 - Render textured polygon

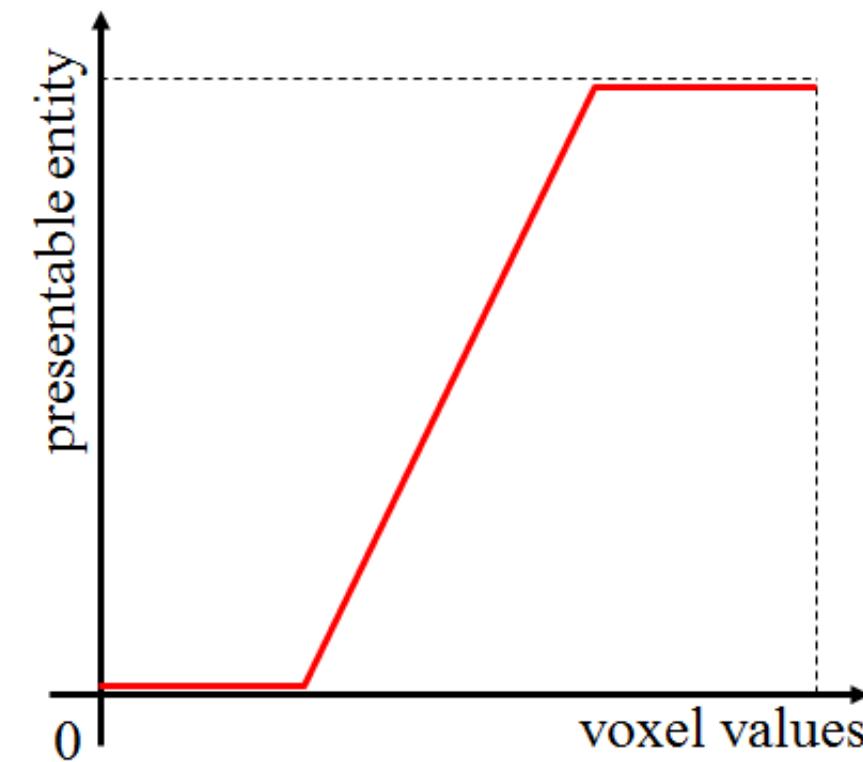
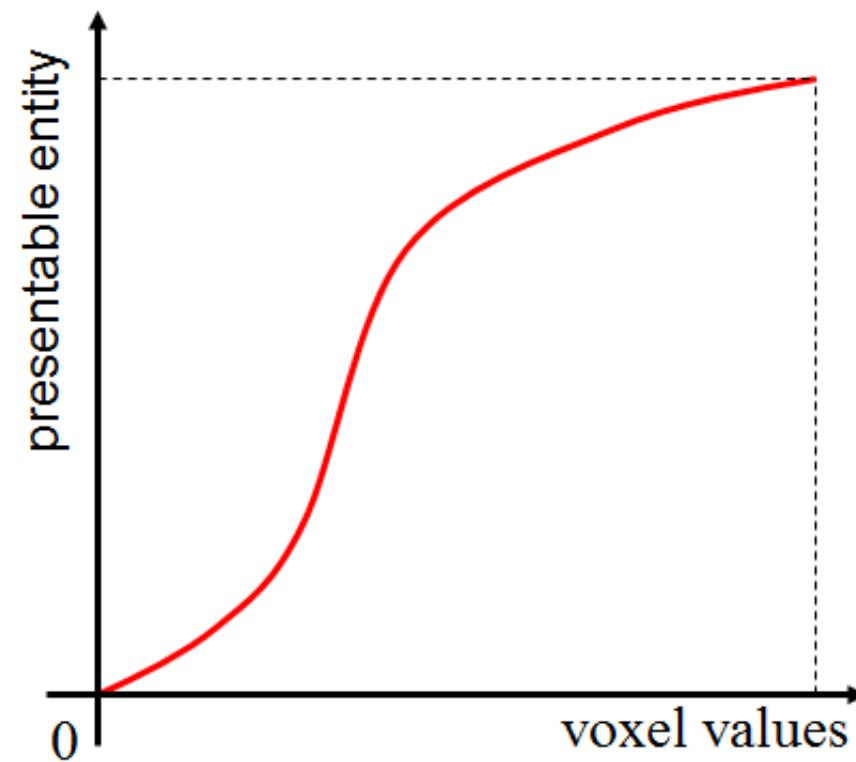


6.3 Classification

Classification

Transfer function

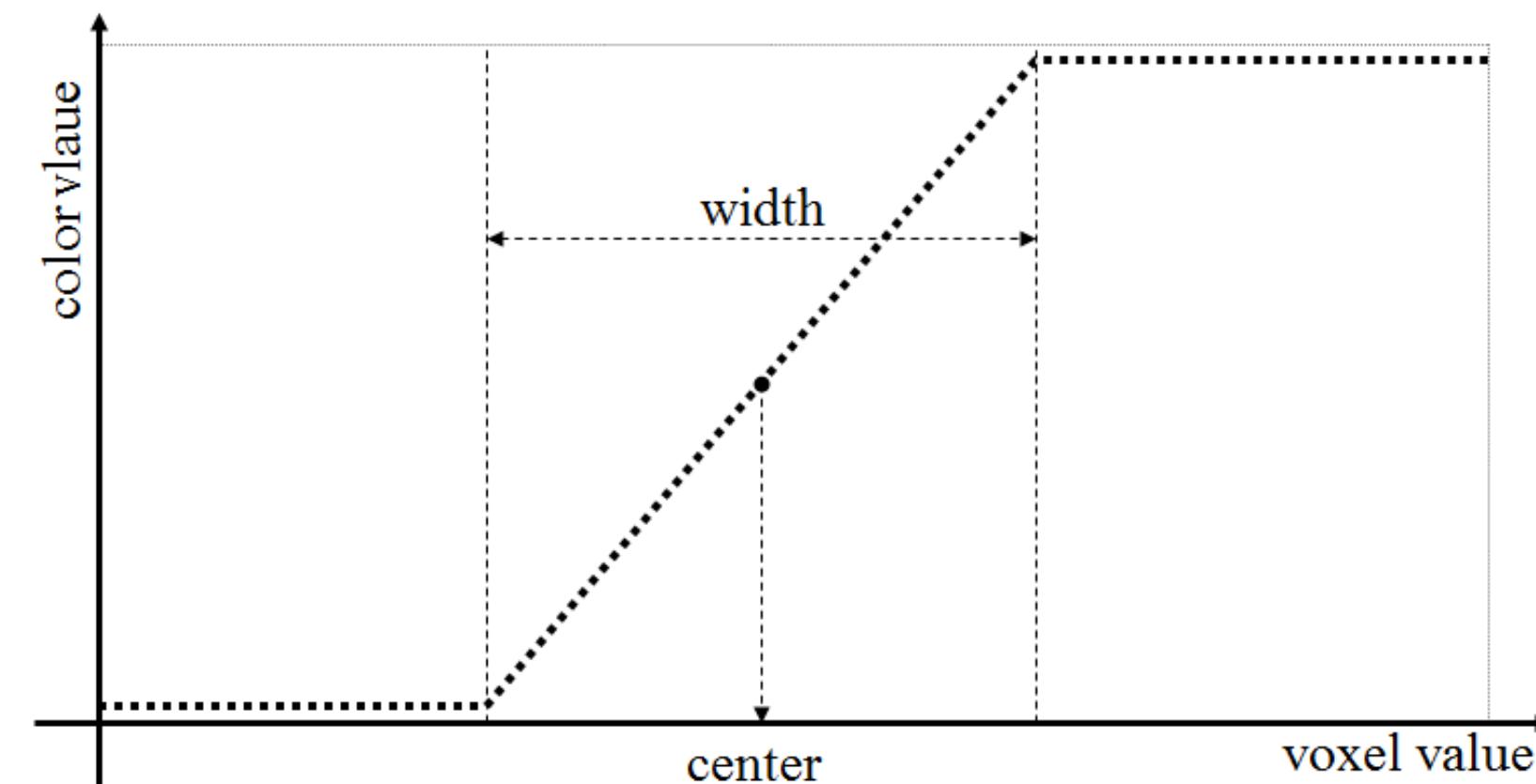
- Map voxel values to representable entities
 - Color, intensity, opacity, etc.



Classification

Transfer function

- Grey value transformation ("windowing")



Adjustment of width and center

Classification

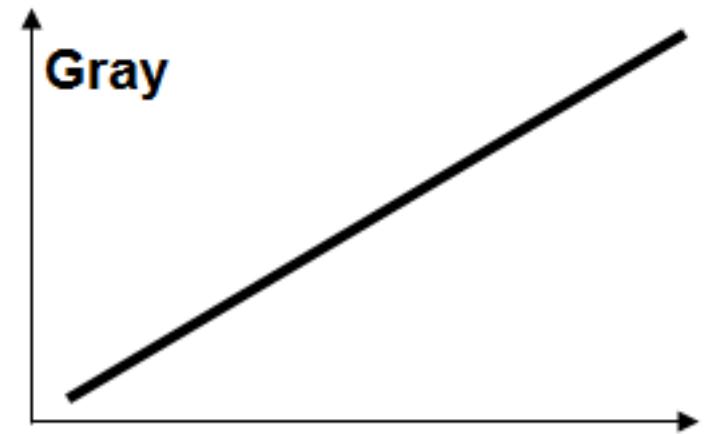
Goals and issues

- Empowers user to select "structures"
- Extract important features of the data set
- Classification is non trivial
- Histogram can be a useful hint
- Often interactive manipulation of transfer functions needed

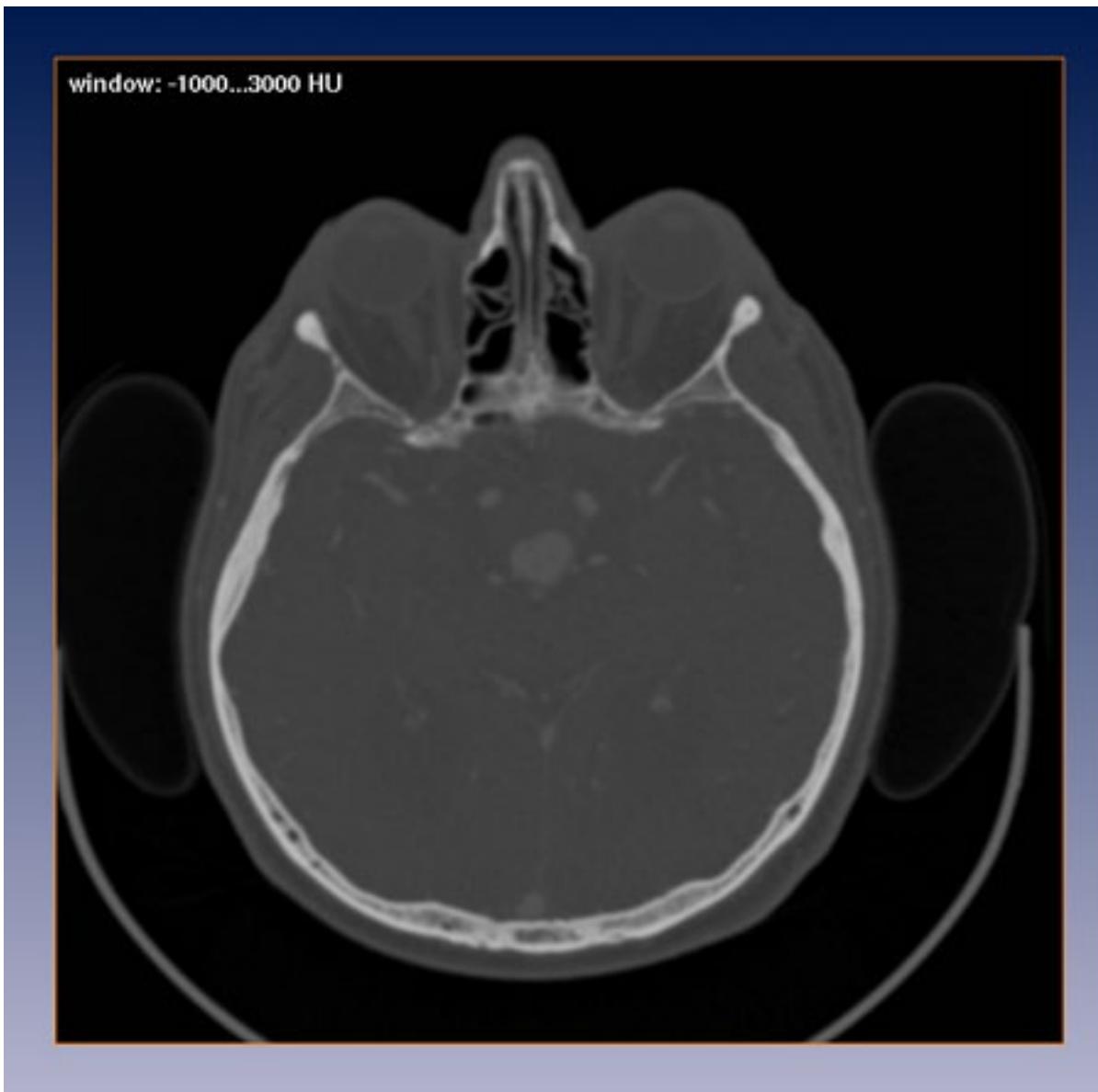
Classification

Example - CT data

- Linear ramp transfer function
- $f_1: [-3000, 1000] \rightarrow [0, 255]$



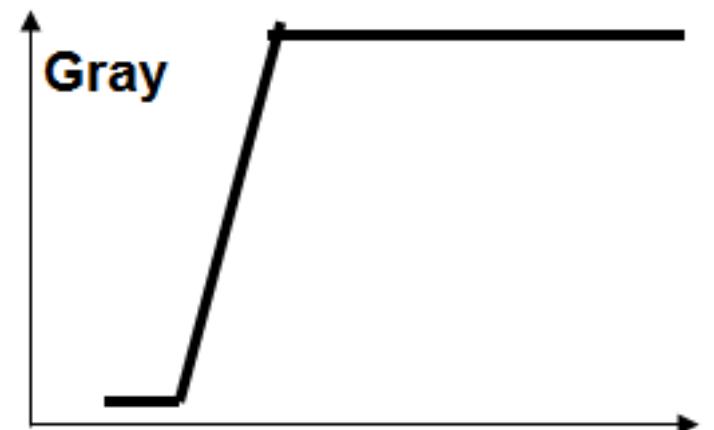
- All data values visible but no details displayed



Classification

Example - CT data

- Center: 0 (water)
Width: 500
- $f_2: [-250, 250] \rightarrow [0, 255]$



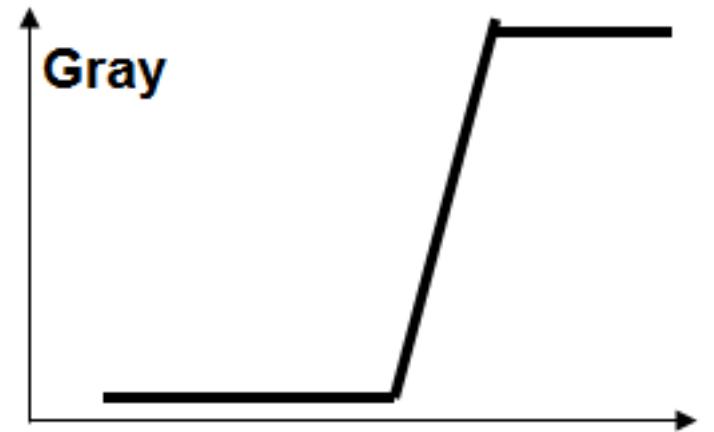
- Blood is highlighted



Classification

Example - CT data

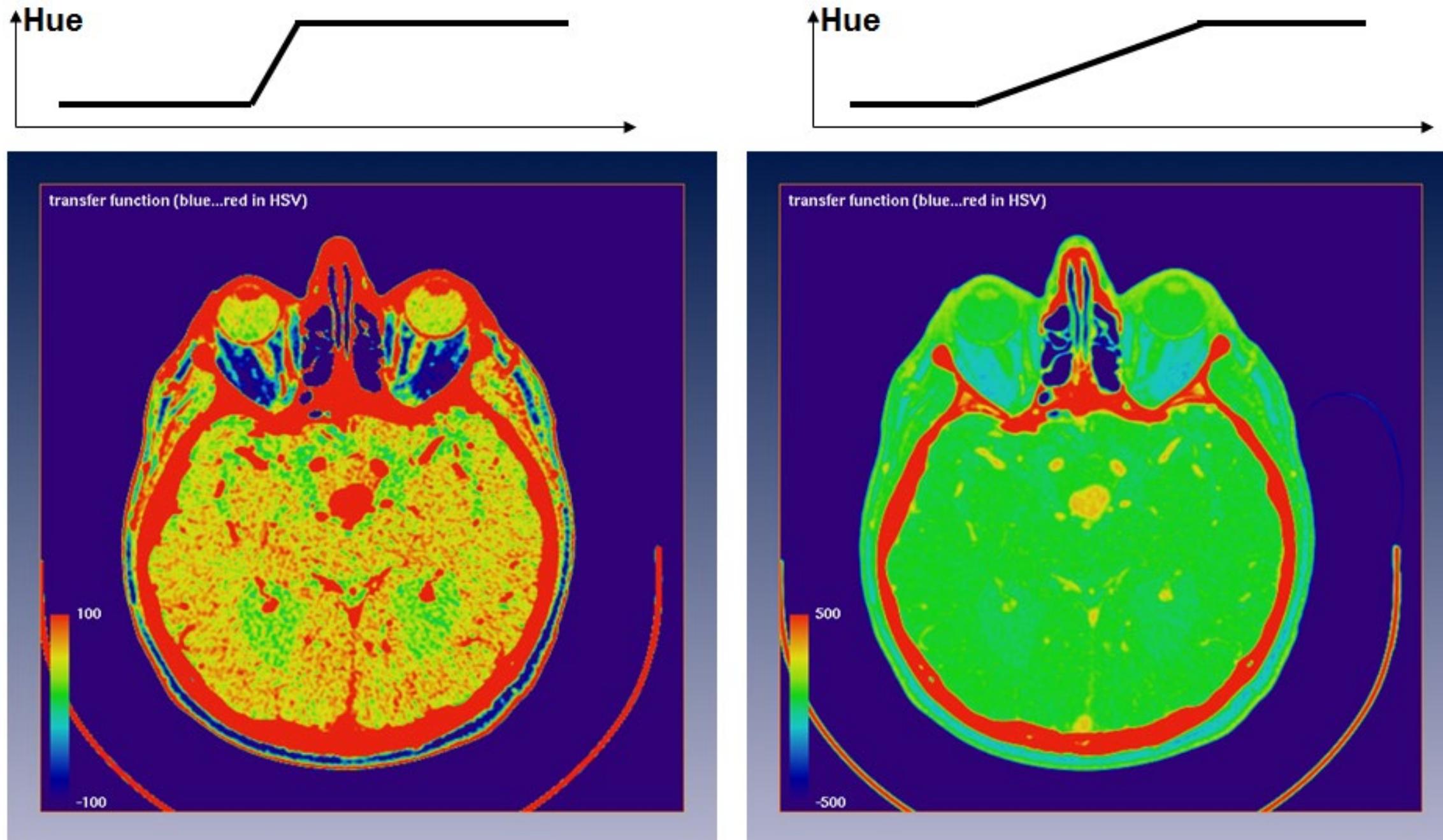
- Center: 1000 (bone)
Width: 500
- $f_2: [750, 1250] \rightarrow [0, 255]$



- Only bone is displayed



Classification



6.4 Indirect Volume Rendering

Indirect volume rendering

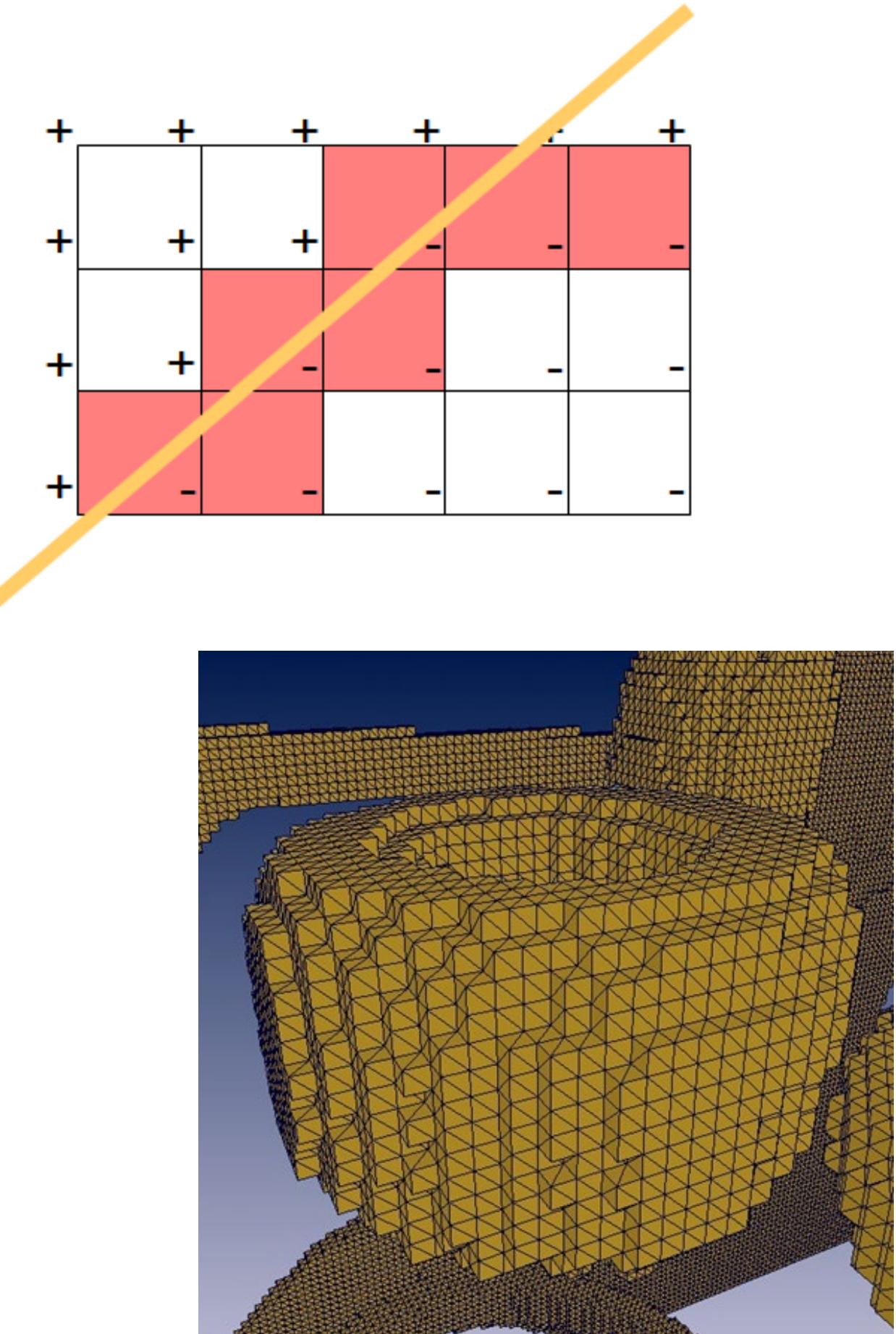
Introduction

- If $f(x,y,z)$ is differentiable in every point, then the level sets $\{(x,y,z) | f(x,y,z) = c\}$ are smooth surfaces (denoted iso-surfaces) to the iso-value c
- Techniques to reconstruct isosurfaces
 - Opaque cube (Cuberille method), Dividing cube
 - Marching cube
 - Marching tetrahedra
 - Surfaces from contours (e.g. NUAGES)
- While opaque cube and marching cube methods are restricted to structured grids, the marching tetrahedra method can deal with any kind of mesh

6.4.1 Opaque Cubes

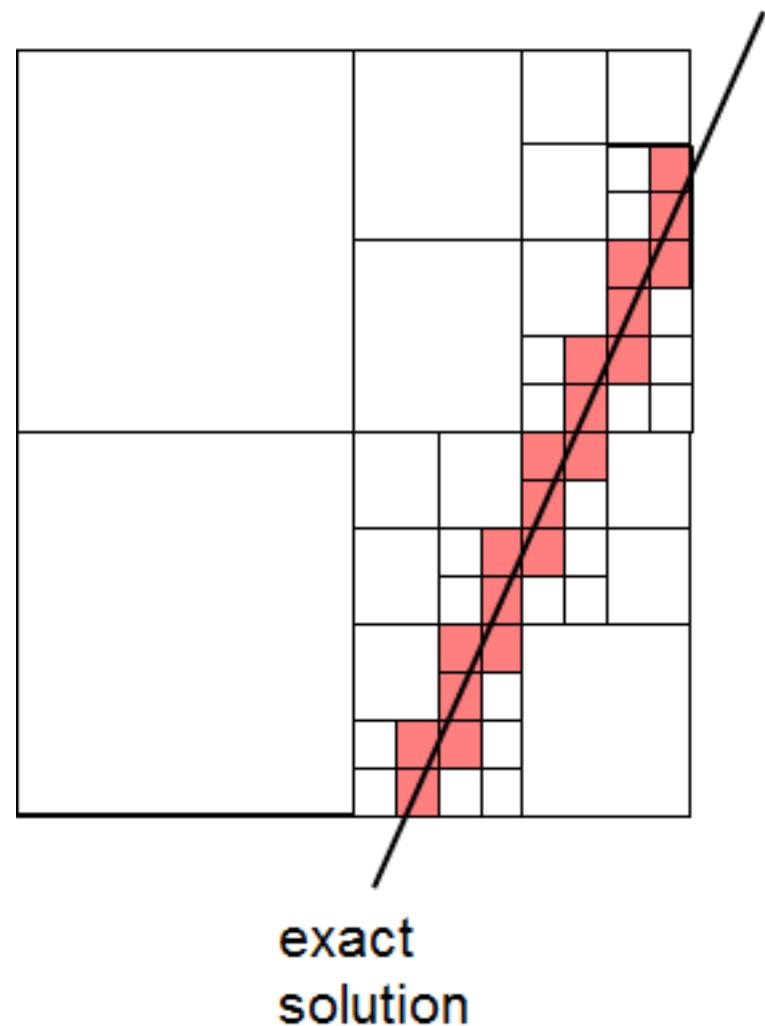
Opaque Cubes

- Approach [Herman, 1979]
 - Volume data on regular grid
 - Detect all cells that intersect the iso-surface $f = c$ by checking vertices
 - If $f_{i,j,k} > c$ mark vertex (i,j,k) with +, otherwise with -
 - Find all boundary front-faces
 - Faces where normal points towards viewpoint ($N \cdot V > 0$) and where normal points outwards the cell
 - Render these faces as shaded polygons
 - "Voxel" point of view: NO interpolation within cells



Opaque Cubes

- Evaluation
 - Method yields blocky surfaces since cell is either opaque or not
 - Improvement through adaptive subdivision
 - Subdivide each marked cube into eight smaller cubes
 - Use trilinear interpolation to determine the function value in the center
 - Repeat the cuberille approach for each new cube until pixel size
 - This is known as "dividing cubes"



6.4.2 Marching Cubes

Marching Cubes

-> Oberflächen

- Approach [Lorensen, Cline, 1987]
 - Better approximation of the "real" iso-surface
 - 3D analog to the isoline construction method (marching squares)
 - Strategy
 - Works on the original data
 - Approximates the surface by a triangle mesh
 - Surface found by linear interpolation along cell edges
 - Uses gradients as the normal vectors of the iso-surface
 - Efficient computation by means of lookup tables
 - THE standard algorithm for geometry-based iso-surface extraction

Marching Cubes

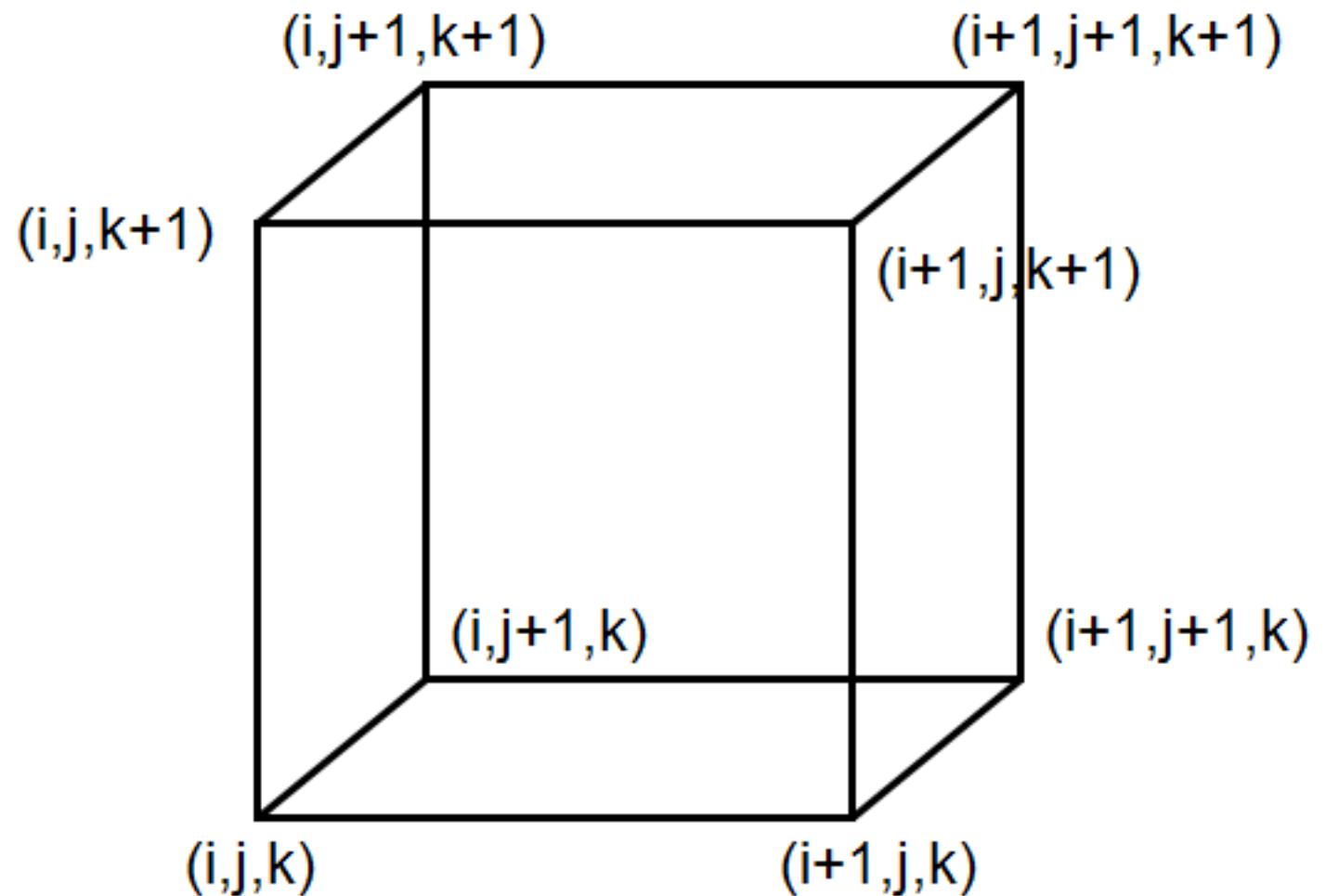
- The core algorithm
 - Cell consists of 4 (8) pixel (voxel) values
 - $(i+[0/1], j+[0/1], k+[0/1])$
- Consider a cell
- Classify each vertex as inside or outside
- Build an index
- Get edge list from table[index]
- Interpolate the edge location
- Compute gradients
- Consider ambiguous cases
- Go to next cell



Marching Cubes

Step 1

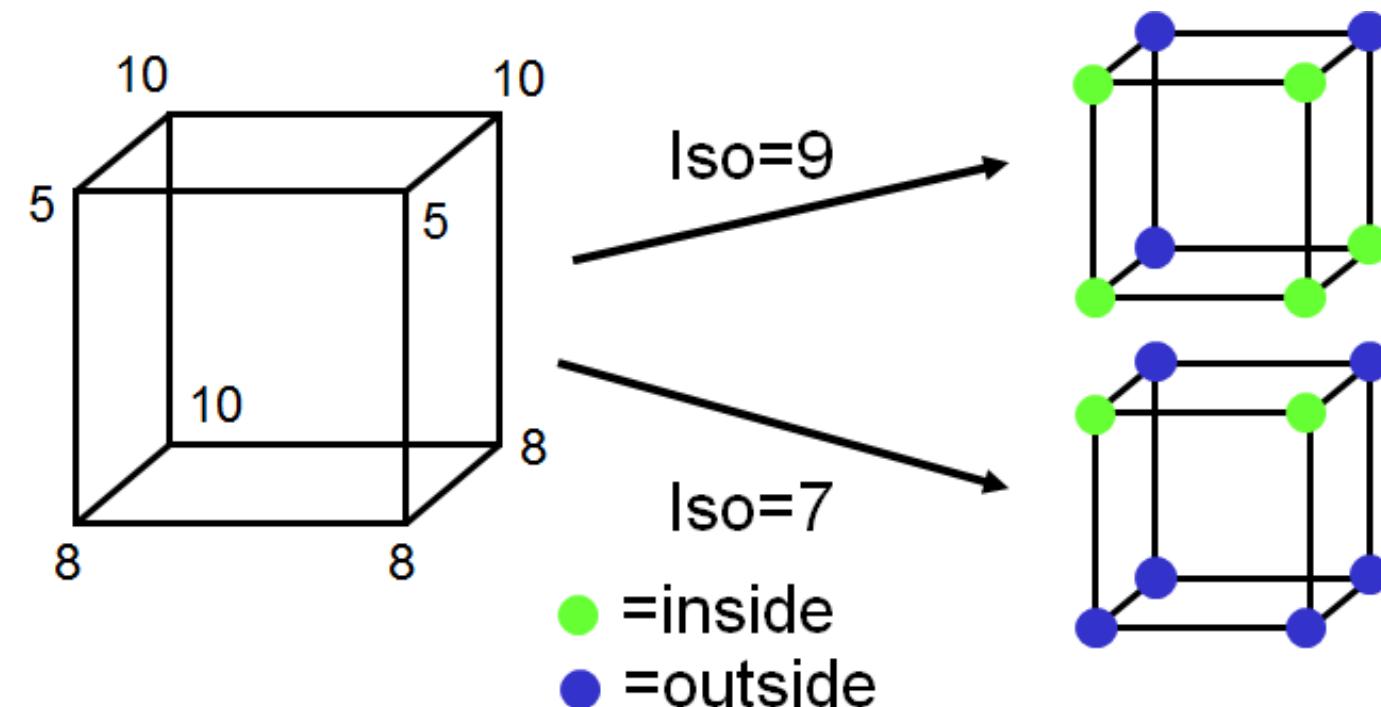
- Consider a cell defined by eight data values



Marching Cubes

Step 2

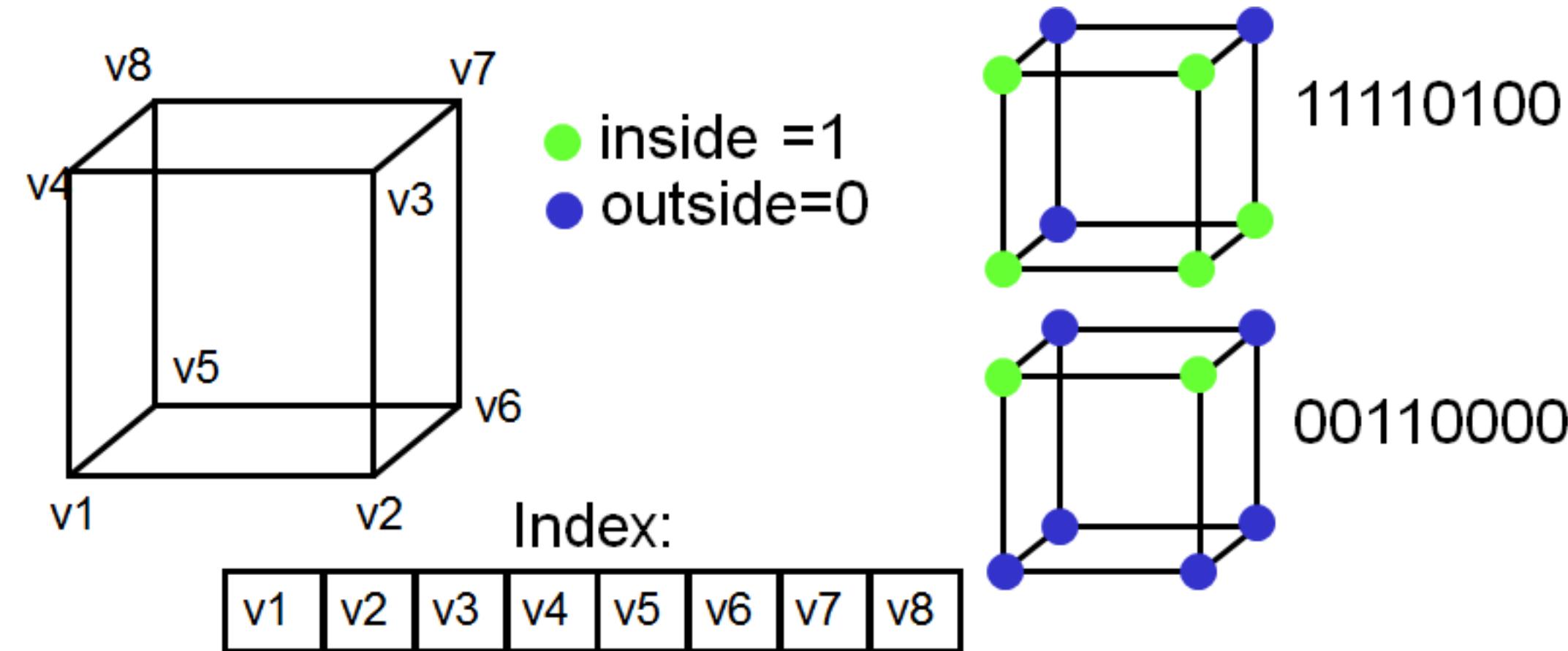
- Classify each voxel according to whether it lies
 - Outside the surface (value > iso-surface value)
 - Inside the surface (value \leq iso-surface value)



Marching Cubes

Step 3

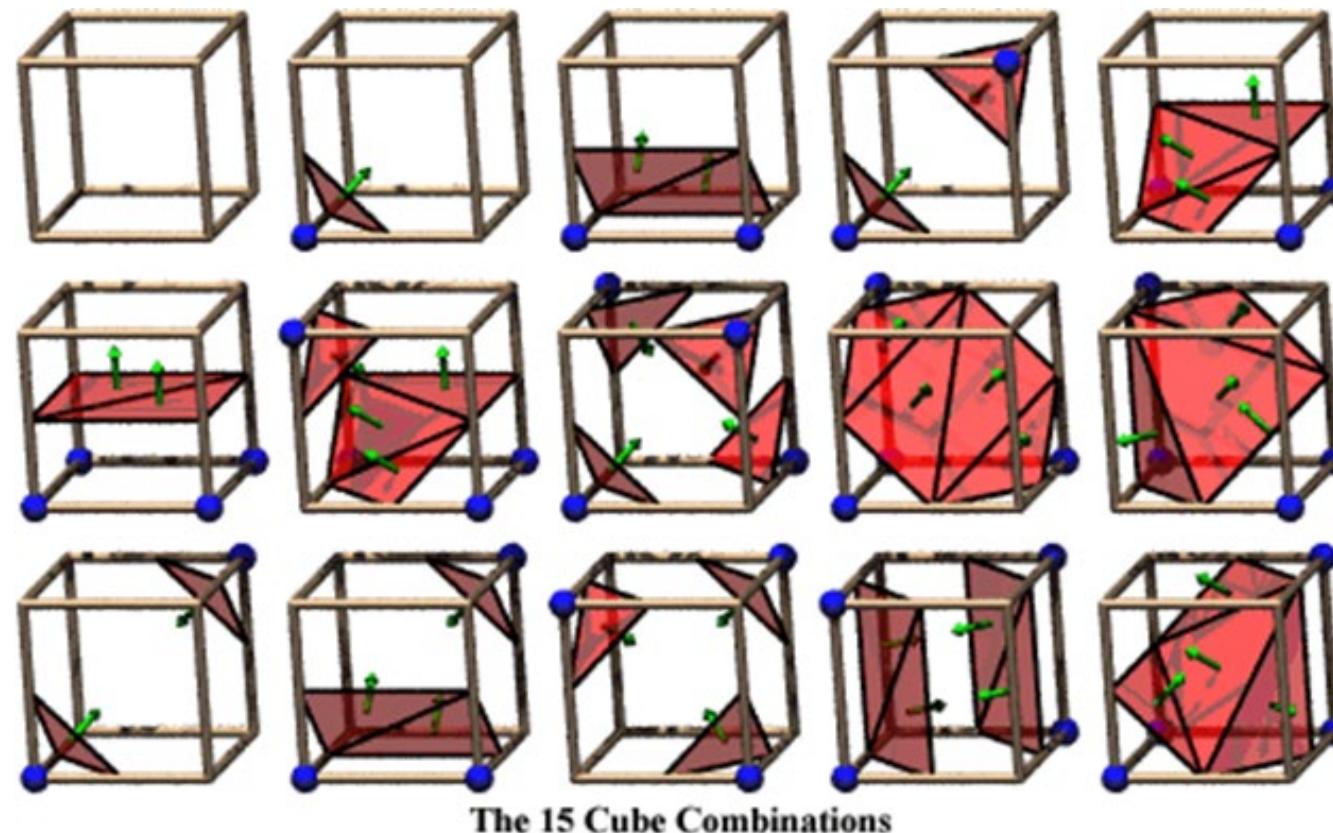
- Use binary labeling of each voxel to create an index



Marching Cubes

Step 4

- For a given index, access an array storing a list of edges
 - All 256 cases can be derived from $1+14 = 15$ base cases due to symmetries



Marching Cubes

Step 4 (cont.)

- Get edge list from table
 - Example for

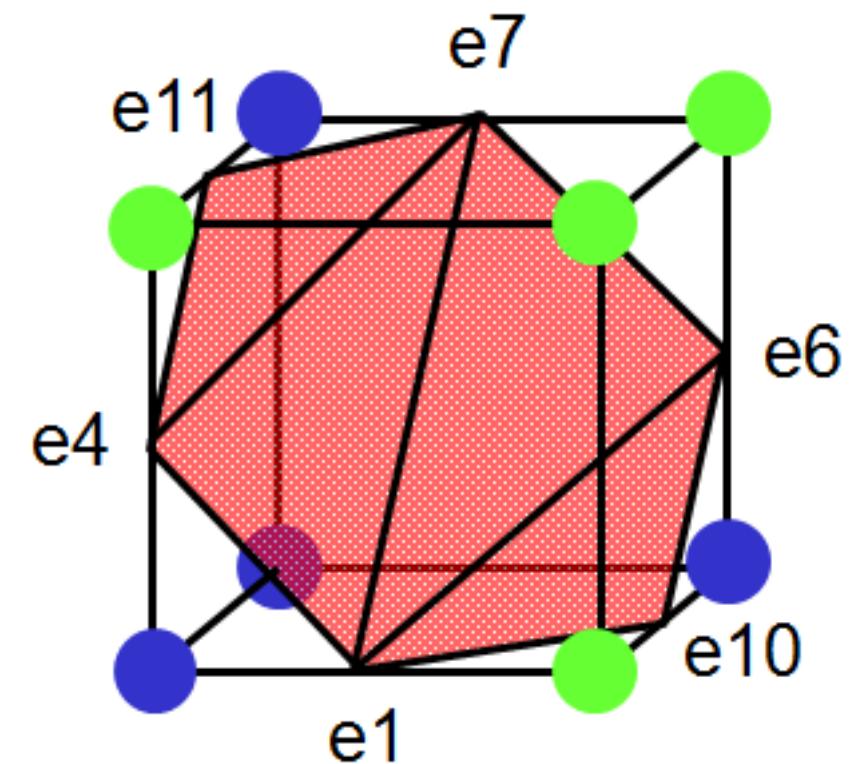
Index = 10110001

tri 1 = e4, e7, e11

tri 2 = e1, e7, e4

tri 3 = e1, e6, e7

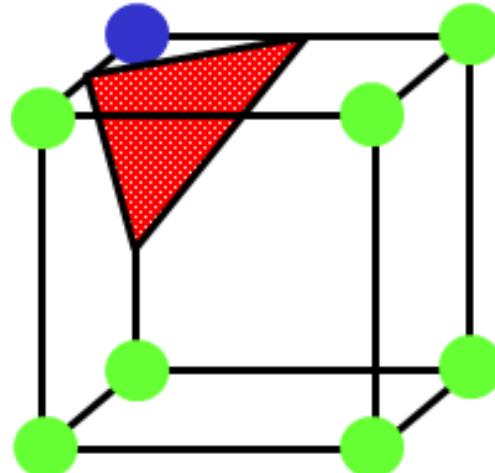
tri 4 = e1, e10, e6



Marching Cubes

Step 5

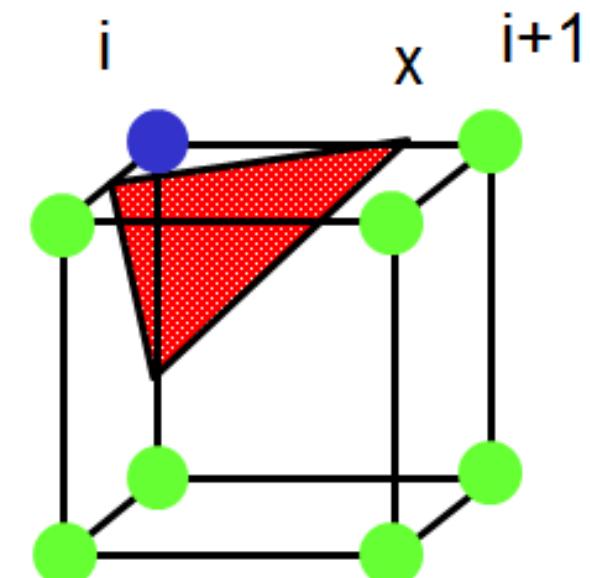
- For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values



T=5

• =10
• =0

$$x = i + \left(\frac{T - v[i]}{v[i+1] - v[i]} \right)$$



T=8

Marching Cubes

Step 6

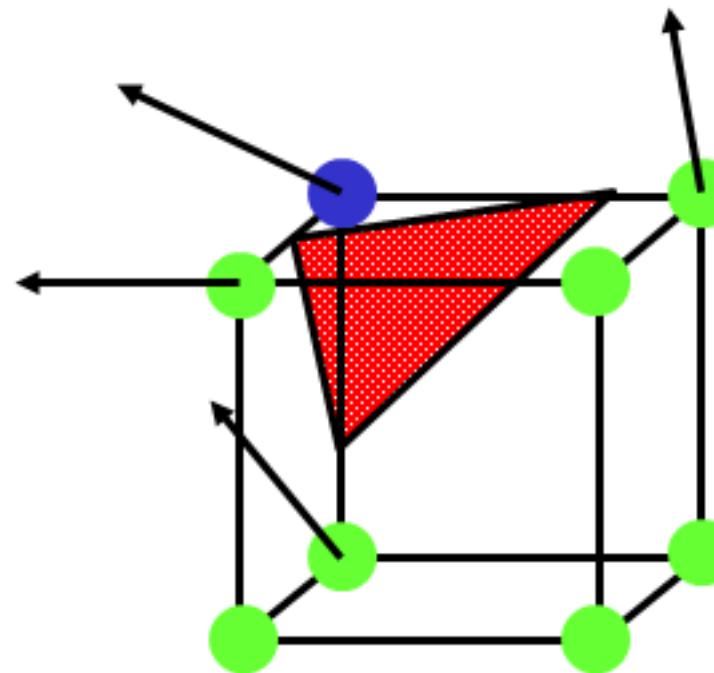
- Calculate normal at each cube vertex (central differences)

$$\bullet G_x = V(x+1,y,z) - V(x-1,y,z)$$

$$G_y = V(x,y+1,z) - V(x,y-1,z)$$

$$G_z = V(x,y,z+1) - V(x,y,z-1)$$

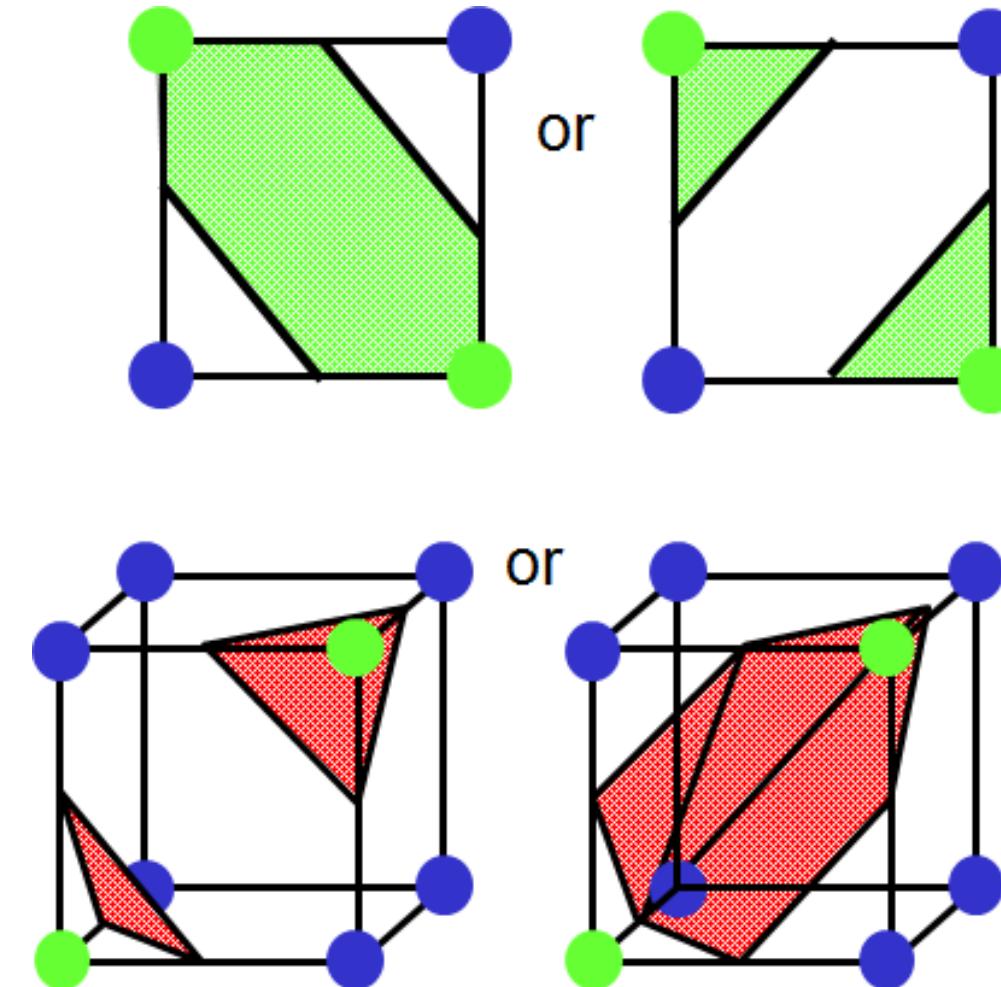
- Use linear interpolation to compute the polygon vertex normal (of the iso-surface)



Marching Cubes

Step 7

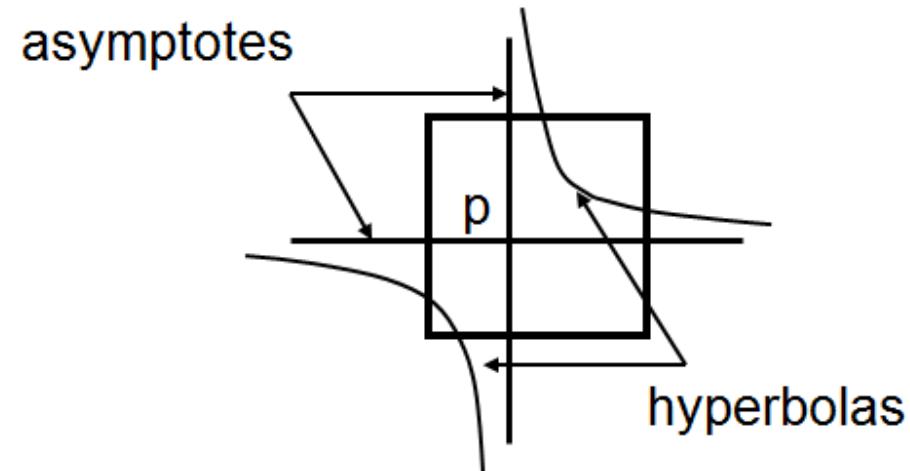
- Consider ambiguous cases
 - Ambiguous cases
 - 3, 6, 7, 10, 12, 13
 - Adjacent vertices:
 - Different states
 - Diagonal vertices:
 - Same state
- Resolution
 - Choose one case
(the right one!)



Marching Cubes

Step 7 (cont.)

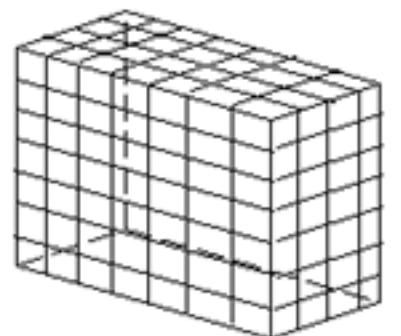
- Consider ambiguous cases
- Asymptotic decider [Nielson, Hamann, 1991] (comp. marching squares!)
 - Assume bilinear interpolation within a face
 - Hence iso-surface is a hyperbola
 - Compute point p where asymptotes meet
 - Sign of $S(p)$ decides on the connectivity
 - This is analog to the 2D case



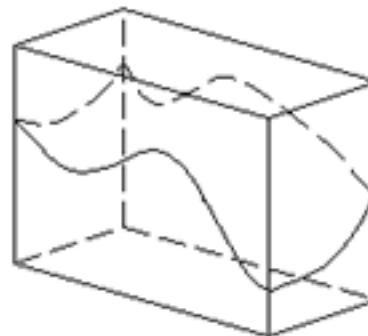
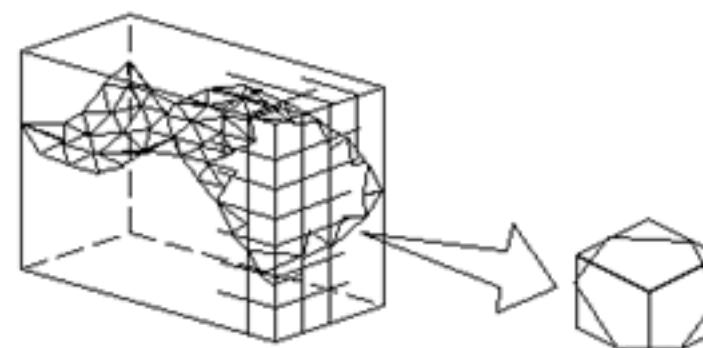
Marching Cubes

Evaluation

- Up to 5 triangles per cube
- Dataset of 512^3 voxels
 - Can result in several millions of triangles (many Mbytes!)
- Both very big and very small triangles
 - Post-processing is necessary to get a "good" triangular mesh
- Many special cases
 - Ambiguity in cases can cause holes if arbitrary choices are made
 - Special cases at the boundaries of the volume



(a) Volume data

(b) Isosurface
 $S = f(x, y, z)$ 

(c) Polygonal Approximation

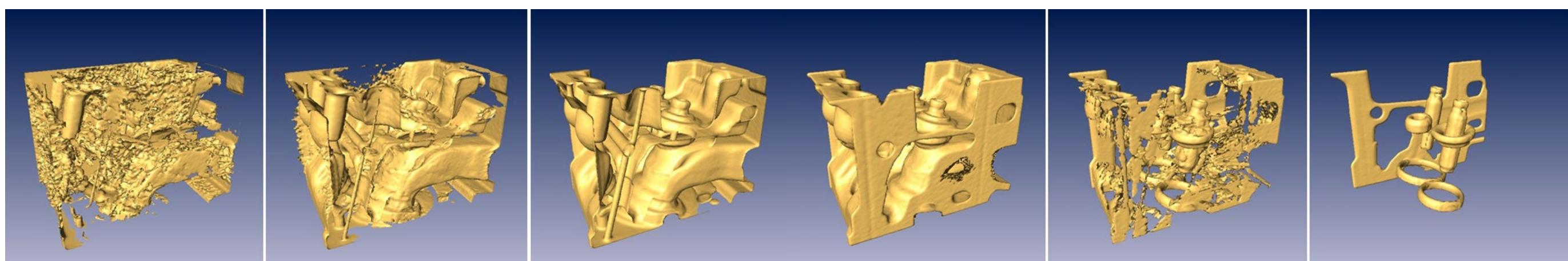
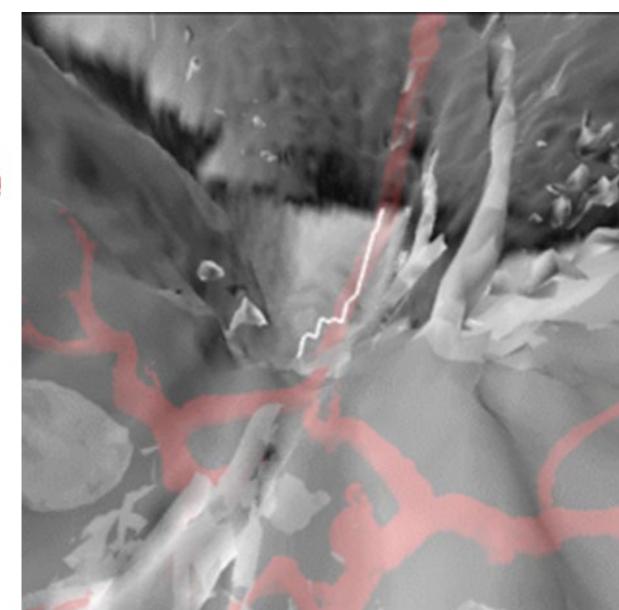
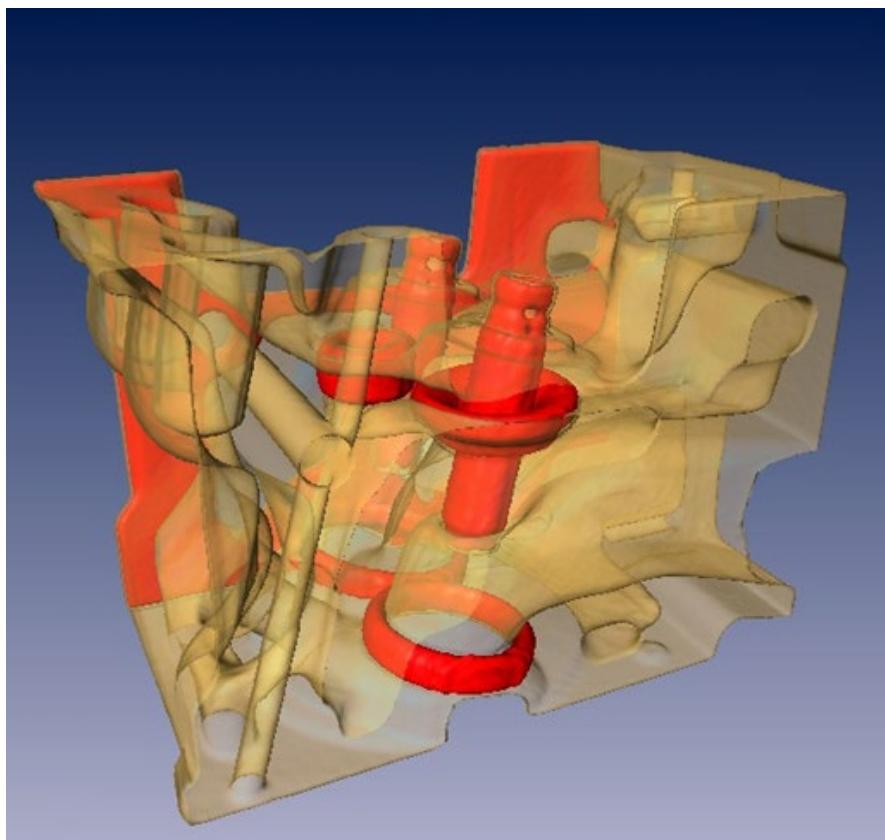
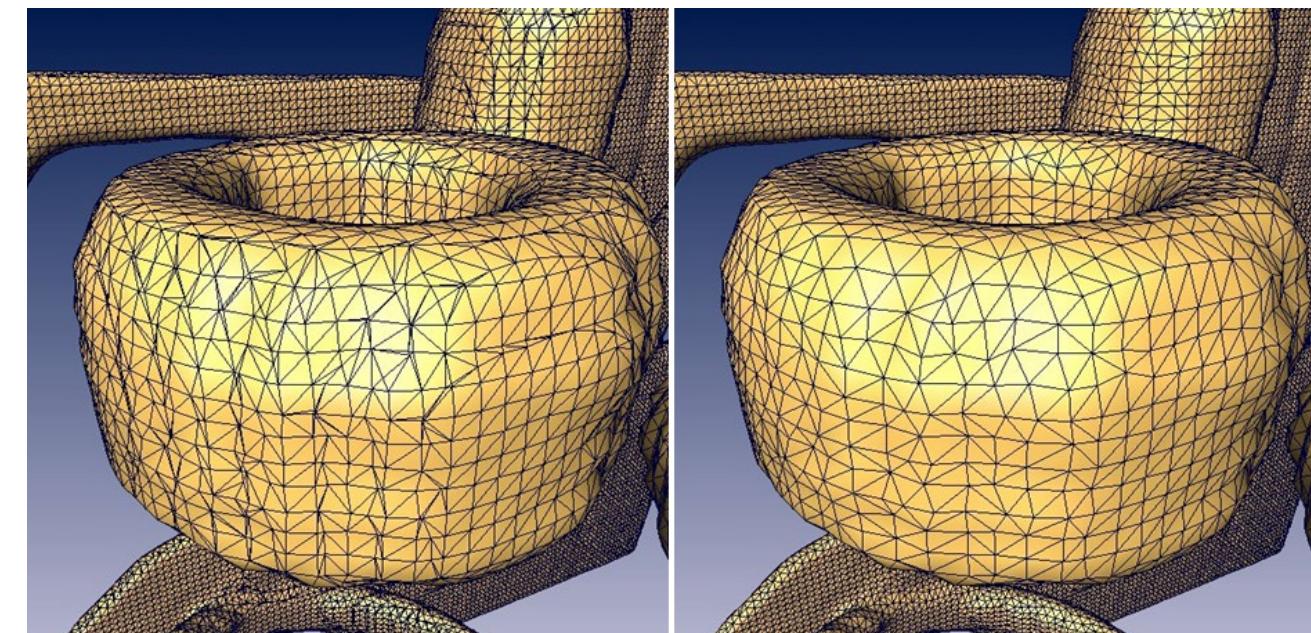
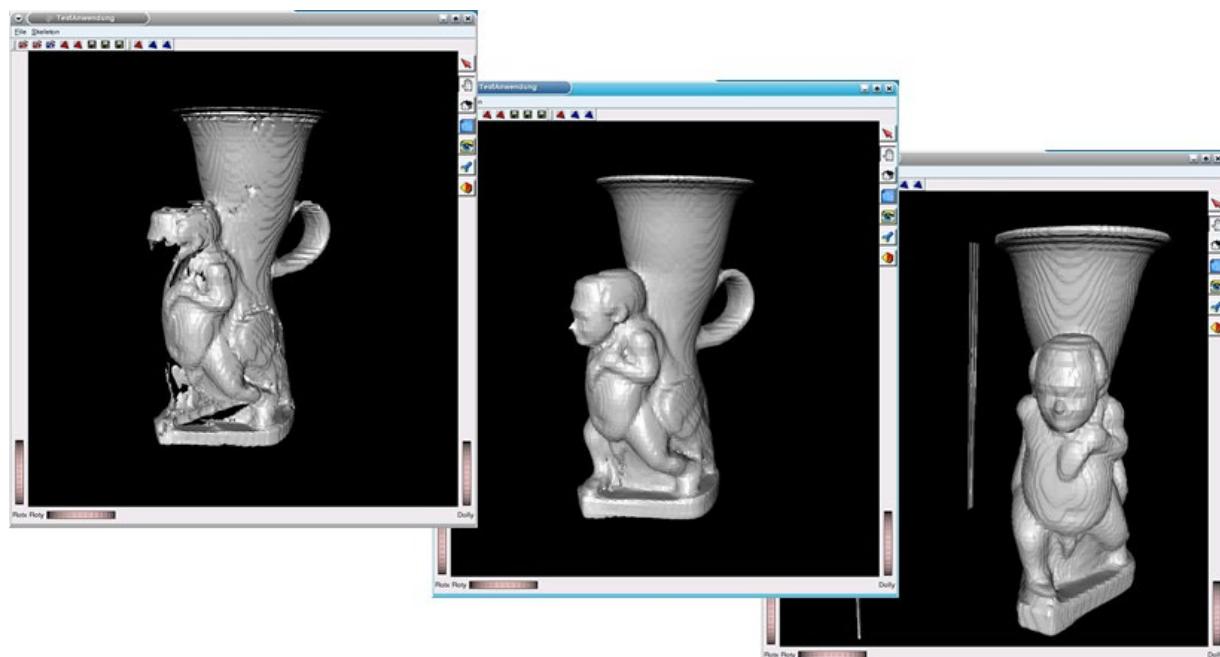
Marching Cubes

Enhancements

- Semi-transparent representation
 - Requires sorting
- Optimization
 - Reuse values from prior calculations
 - Prevent vertex replication
 - Mesh simplification
- Accelerated display
 - Graphics hardware
- High quality display
 - Ray-tracing algorithm

Marching Cubes

Examples

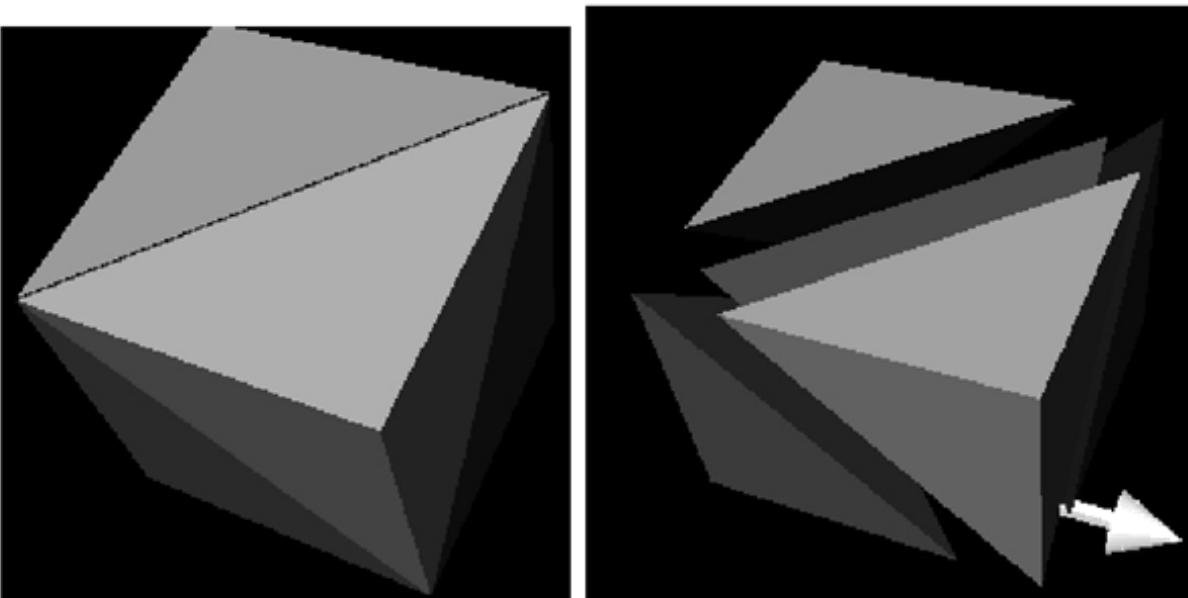


6.4.3 Marching Tetrahedra

Marching Tetrahedra

Alternative to Marching Cubes

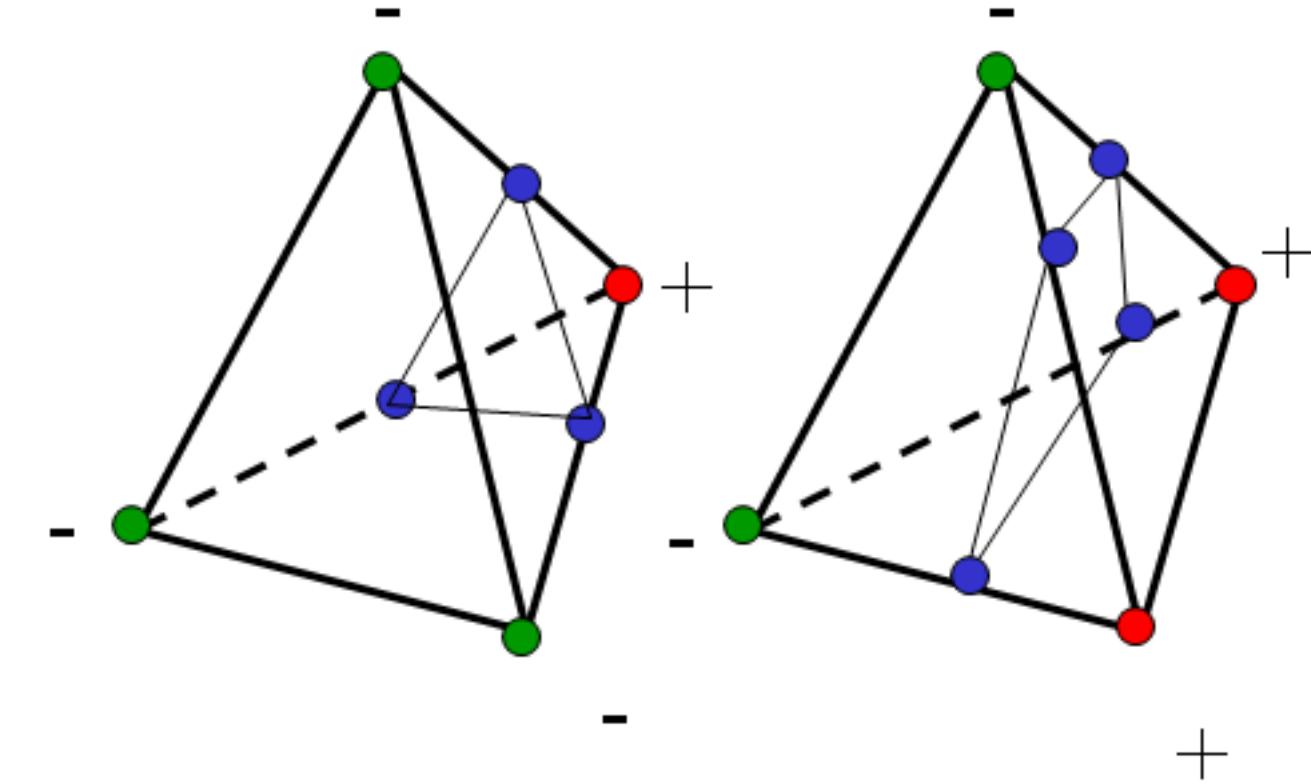
- Split each cube into five or six tetrahedra
 - Depending on the method used for splitting the cube
 - Example for 5 tetrahedra



- Yields more triangles and "rougher" surface than MC

Marching Tetrahedra

- Approach
 - Primarily used for unstructured grids
 - Process each cell similarly to the MC-algorithm
 - Mark vertices with + or -, depending on $f > c$ or not
 - Only two possible non-trivial cases for a tetrahedron
 - One "-" and three "+" (or vice versa)
 - Intersection surface is a triangle
 - Two "-" and two "+"
Intersection surface is a quadrilateral
 - Split into two triangles using the shorter diagonal



Marching Tetrahedra

- Properties
 - Fewer cases, i.e. 3 instead of 15
 - Linear interpolation within cells
 - No problems with consistency between neighboring cells
 - Also many triangles of different size
 - Number of generated triangles might increase considerably compared to the MC algorithm due to splitting into tetrahedra
 - Huge amount of geometric primitives
 - But, several improvements exist
 - Hierarchical surface reconstruction
 - View-dependent surface reconstruction
 - Mesh decimation

6.4.4 Surfaces from Contours

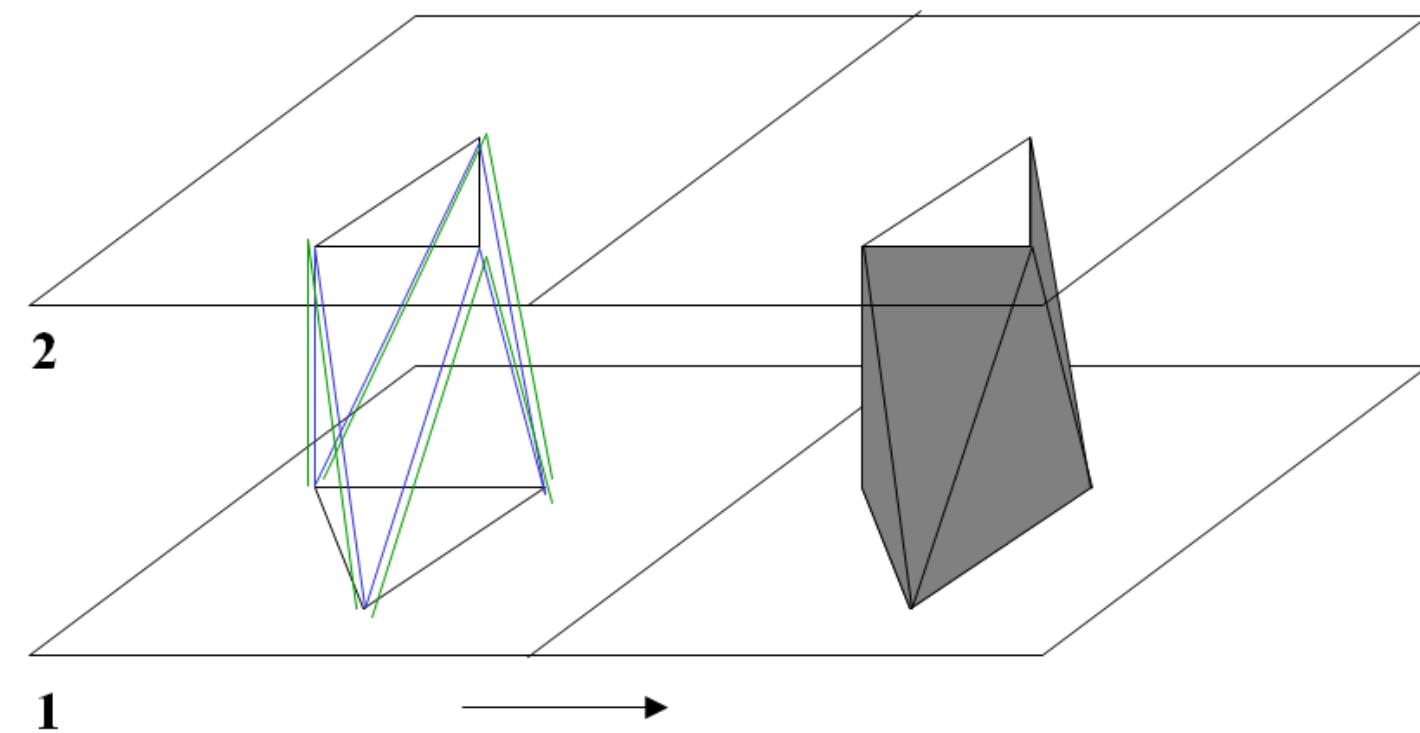
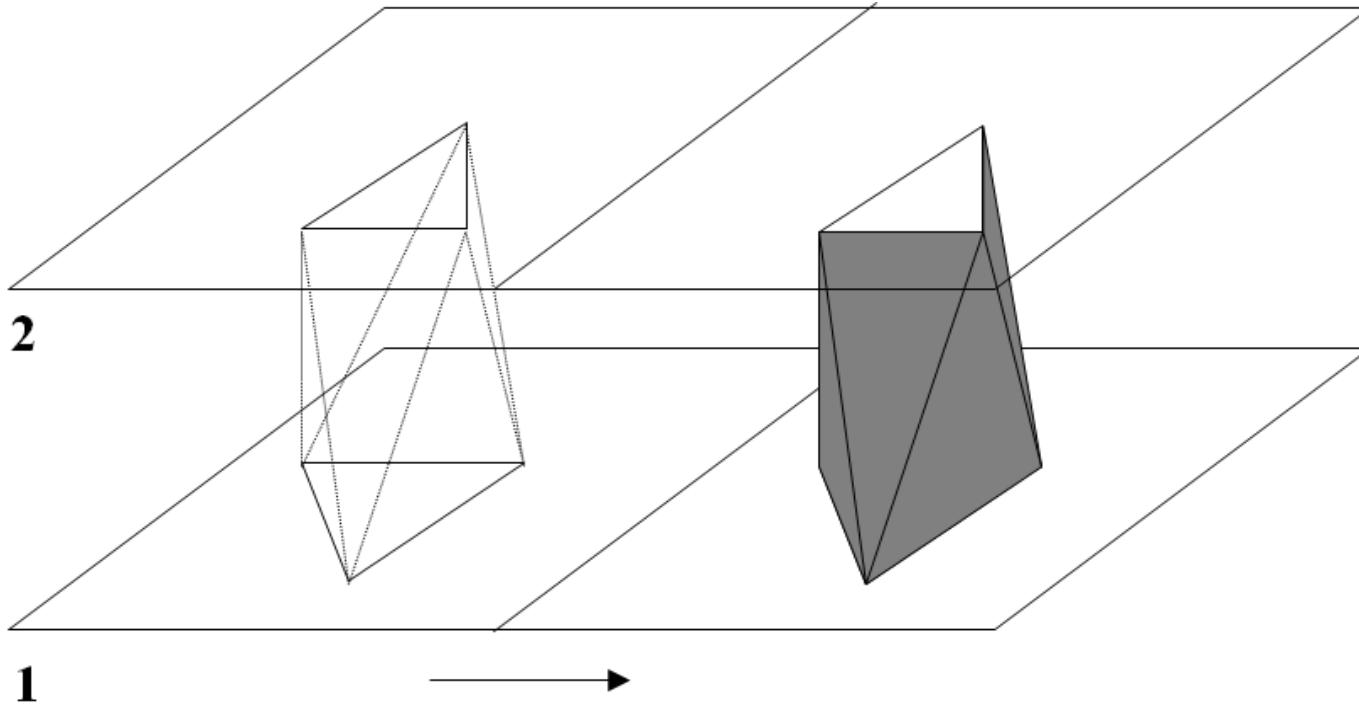
Surfaces from Contours

- Approach
 - Explicit segmentation
 - Find closed contours in successive 2D slices
 - Often semi-automatic procedure with user-interaction!
 - Requires a lot of expertise
 - Represent contours as poly-lines
 - Labeling
 - Identify different structures: e.g. brain, vessels, ...
 - Reconstruction
 - Connect contours representing the same object from neighboring slices and form triangles
 - Rendering
 - Display triangles

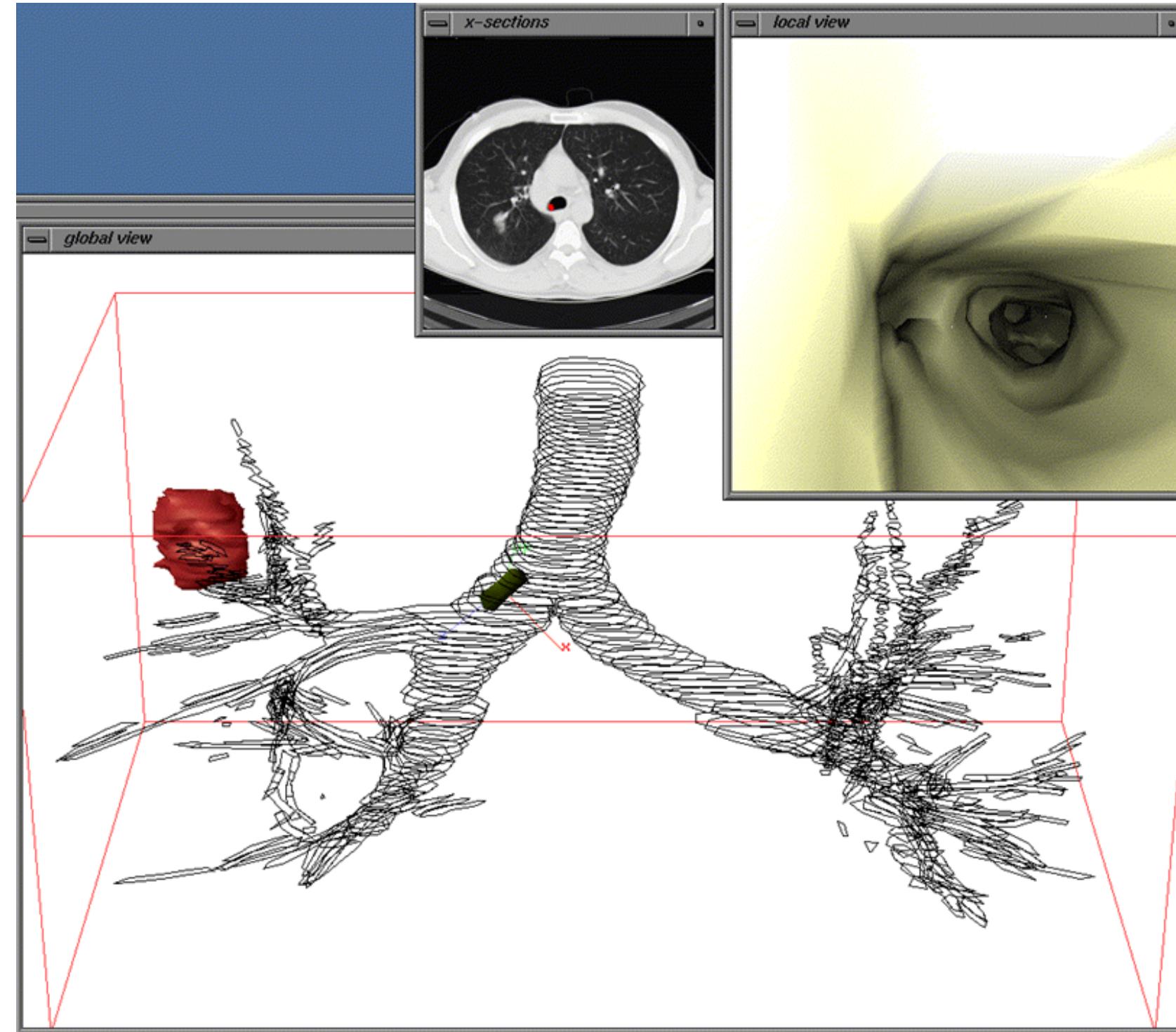
Surfaces from Contours

Example

- Identify for edge on level 2 corresponding vertex on level 1
- Identify for edge on level 1 corresponding vertex on level 2



Surfaces from Contours



Surfaces from Contours

- Problems
 - Many contours in each slice
 - There is a high variation between slices
 - Correspondence between neighboring slices can be difficult
 - Branching has to be handled (not always easy)
- Literature
 - B. Geiger: NUAGES (INRIA France, 1993) - see paper and software
<http://www-sop.inria.fr/prisme/fiches/Medical/index.html.en>
 - F. Cazals, J. Giesen, Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms, INRIA, Tech Report 2004
 - D. Wang, O. Hassan, K. Morgan, N. Weatherill, Efficient surface reconstruction from contours based on two-dimensional Delaunay triangulation, Int. J. Numer. Meth. Engng 2006; 65:734–751

Summary

Summary

- Advantages
 - Unambiguous definition of a surface
 - Clear visual impression (light!)
 - Use standard graphics hardware for rendering
- Disadvantages
 - Explicit segmentation required
 - Difficult and time consuming
 - Exact surface
 - Ill-posed problem in regions of little data value variation
 - Information reduction
 - Too many triangles in case of complex structures

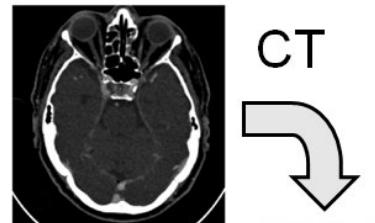
6.5 Direct Volume Rendering

Introduction

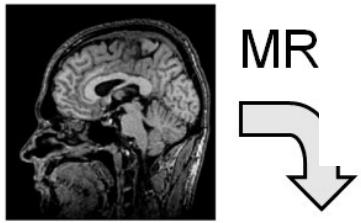
Properties (of direct volume rendering)

- Directly get a 3D representation of the volume data
- The data is considered to represent a semi-transparent light-emitting medium
 - Hence, also gaseous phenomena can be simulated
- Approaches are based on the laws of physics
 - Emission
 - Absorption
 - Scattering
- The volume data is used as a whole
 - Look inside, see all interior structure

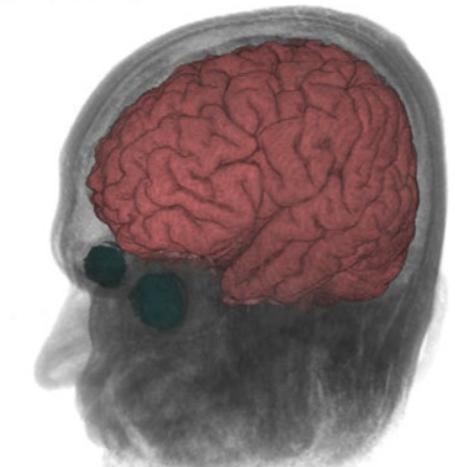
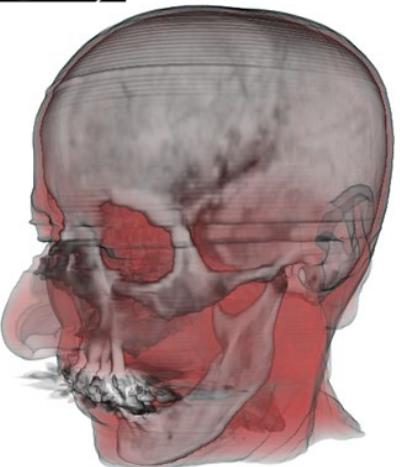
Introduction



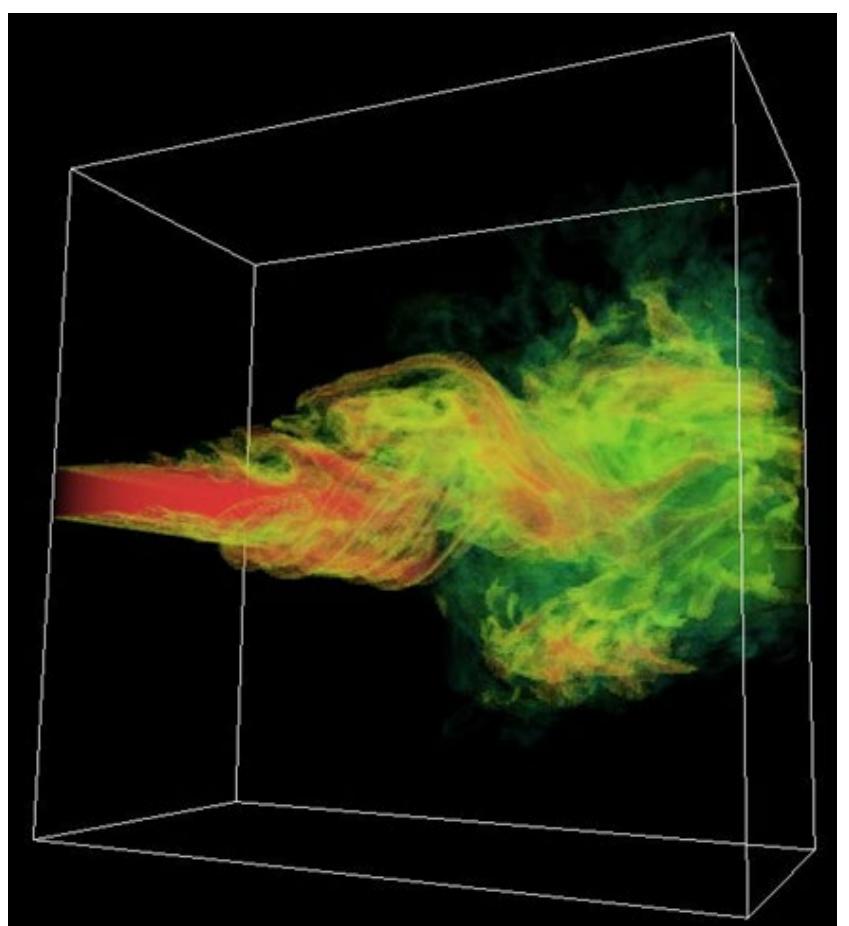
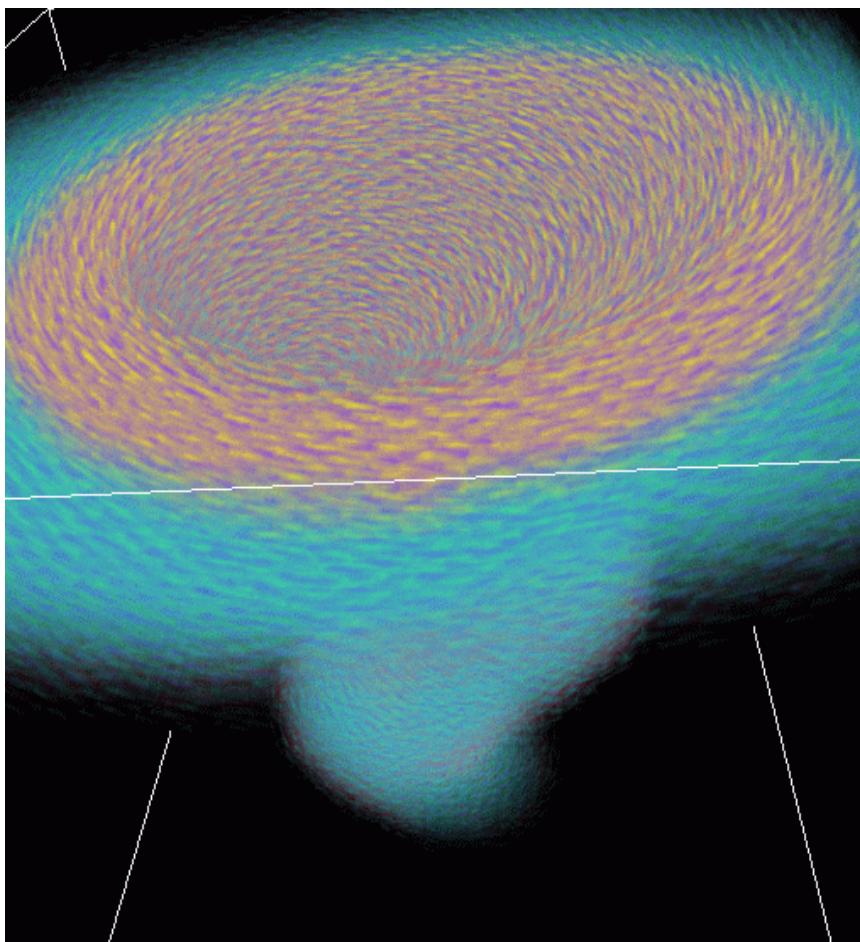
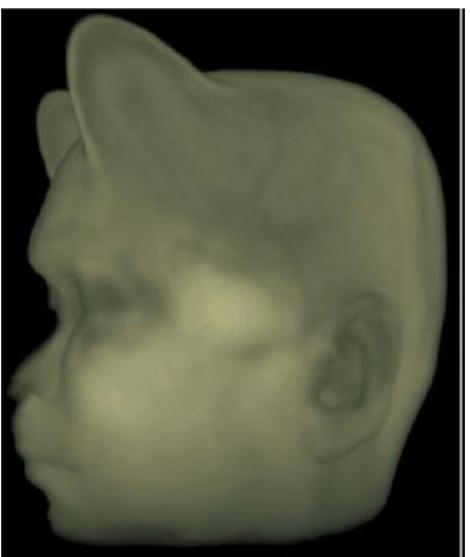
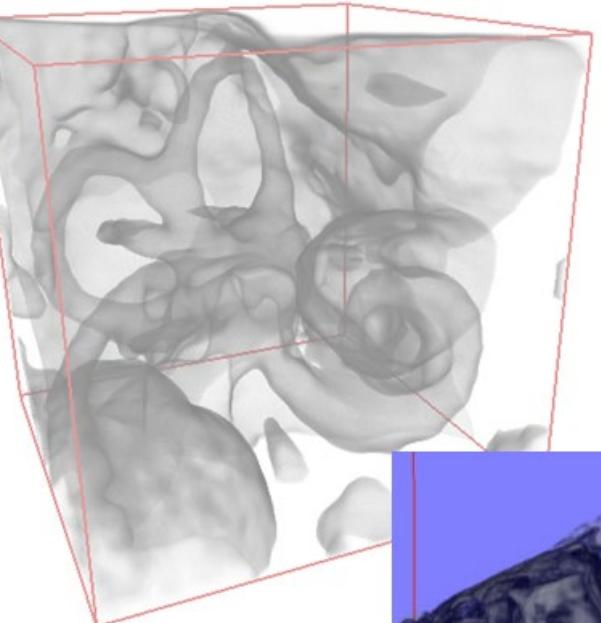
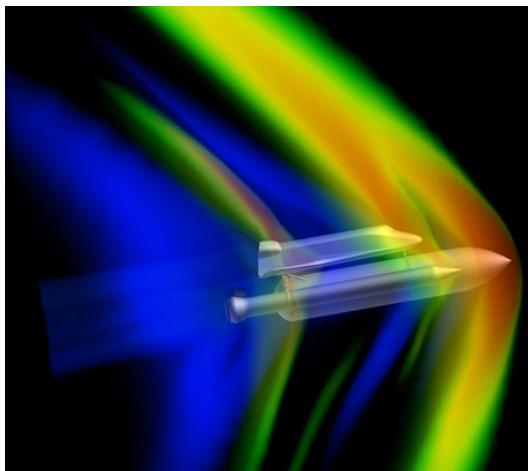
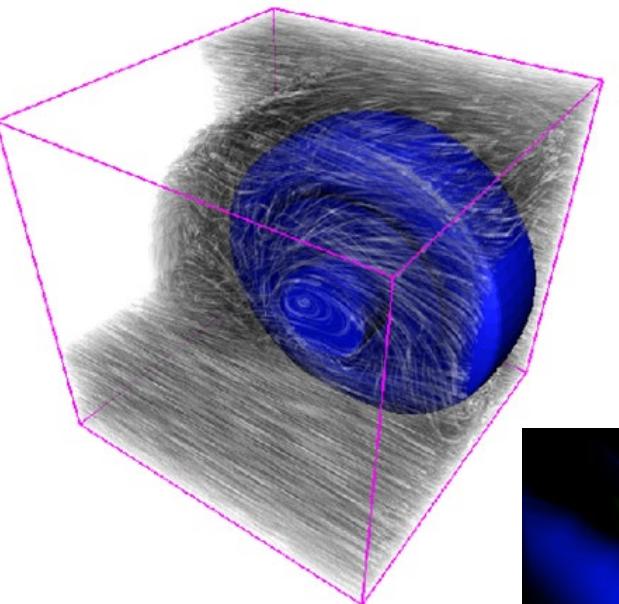
CT



MR



Xue et al, *Fast Dynamic Flow Volume Rendering Using Textured Splats on Modern Graphics Hardware*,
Proceedings SPIE EI 2004



Introduction

Advantages

- Use all information (comprehensive representation)
- Show fuzzy and amorph objects
- This approach does not exclude iso-surfaces (combination)
 - However, often there is no need to use them
- Segmentation Visualisieren der Daten -> Segmentation => zuweisen bestimmter Bereiche mit Farben (Werte)
 - Explicit (voxel labeling)
 - Implicit (use color and opacity based on transfer functions)

Introduction

Disadvantages

- Interactivity
 - Requires special algorithms and special hardware
- Assignment of color and opacity is difficult
- Amount of data
 - MRI
 - $(256^2 \text{ or } 512^2) \text{ px} \times 128 \text{ slices} \times 2 \text{ bytes} = 16\text{-}64 \text{ MB}$
 - CT
 - $512^2 \text{ px} \times (128 \text{ up to } 800) \text{ slices} \times 2 \text{ bytes} = 64\text{-}400 \text{ MB}$
 - Material science
 - $1024^2 \times (128 \text{ up to } 800) \text{ slices} \times 2 \text{ bytes} = 256\text{-}1600 \text{ MB}$
 - If data is time dependent > 1GB

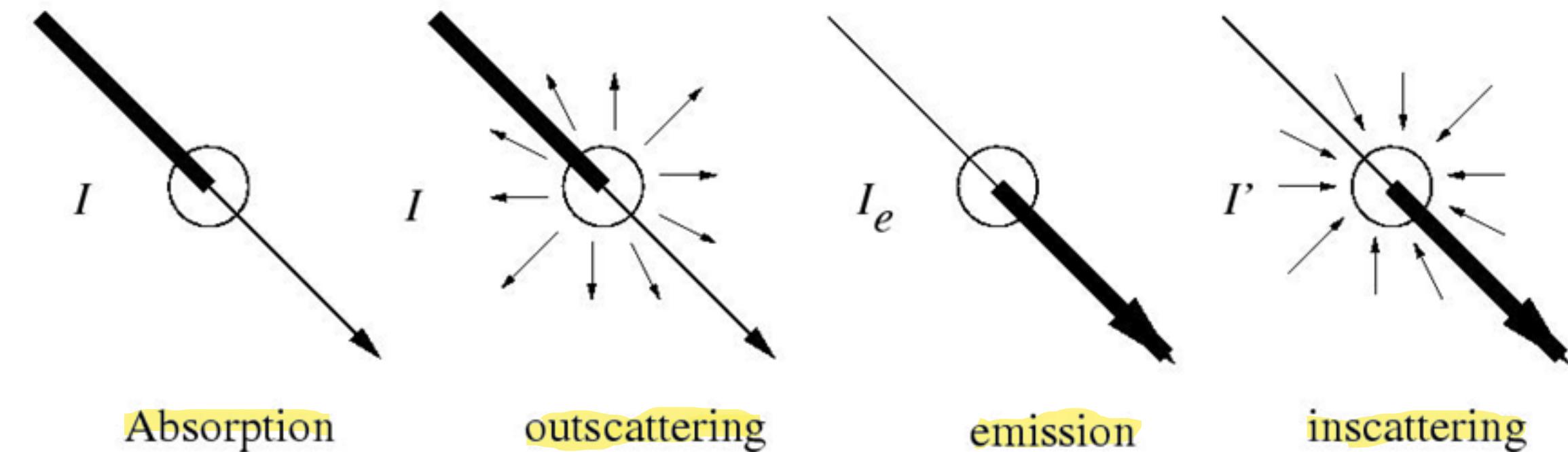
→ Huge amount of interpolations!

6.5.1 Volume Rendering Equation

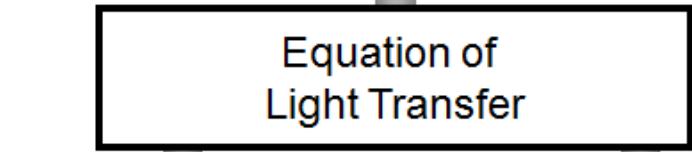
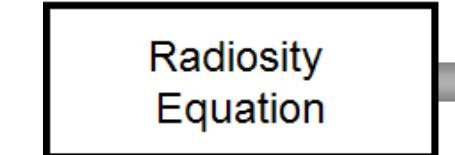
Volume Rendering

Light transport

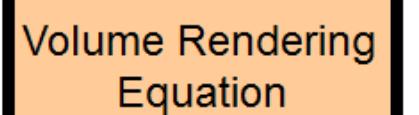
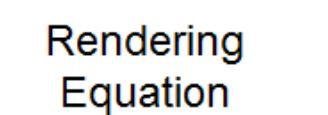
- Contributions to radiation at single position
 - Absorption
 - Emission
 - Scattering



no scattering



No Medium



No Scattering

Volume Rendering

Emission-absorption model

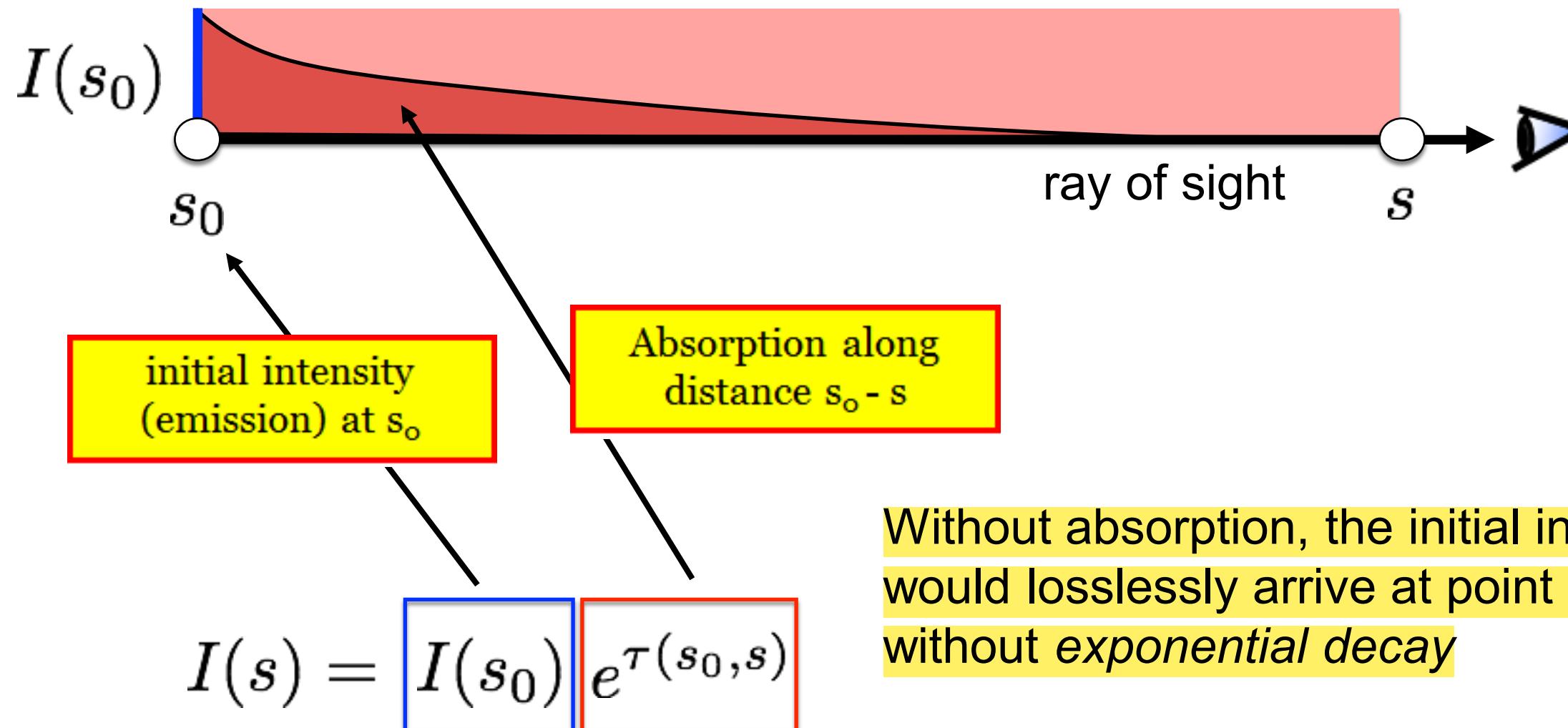
- Special case for most volume rendering
 - Also denoted: density-emitter model [Sabella 1988]
- Idea
 - Volume filled with light-emitting particles but also absorbe
 - Particles described by density function
- Simplifications
 - No scattering
 - Emission coefficient has source term only
 - Absorption coefficient has true absorption only

Volume Rendering

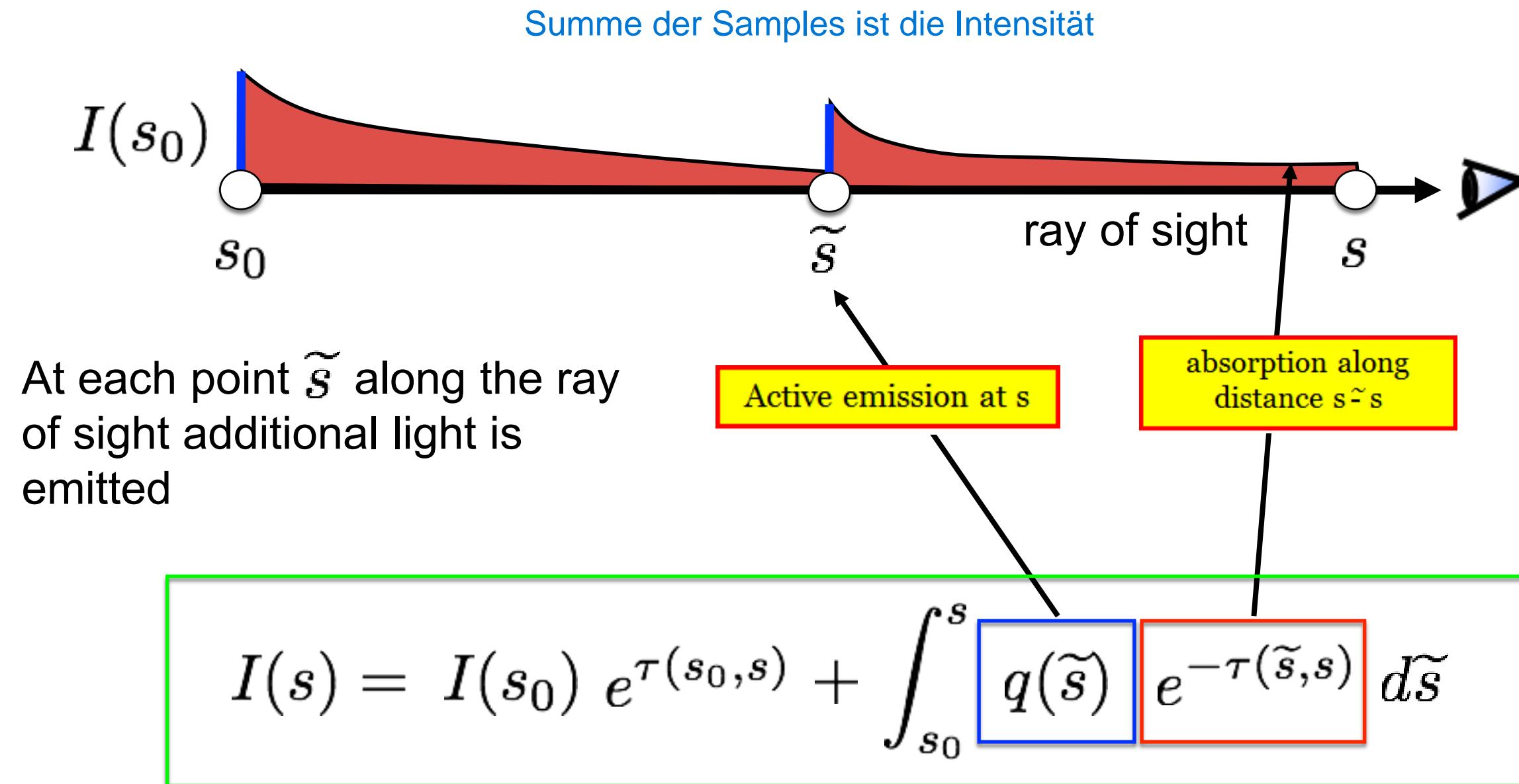
Absorption intensity decays exponentially

optical depth τ
absorption κ

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s) ds.$$



Volume Rendering



Volume Rendering

Discretization

- Define
 - Transparency part

$$\theta_k = e^{-\tau(s_{k-1}, s_k)}$$

- Emission part

$$b_k = \int_{s_{k-1}}^{s_k} q(s) e^{-\tau(s, s_k)} ds$$

Summe über alle Emissionen gewichtet über Absorptionen (Transparenz)

- Discretized volume integral

$$I(s_n) = I(s_{n-1}) \theta_n + b_n = \sum_{k=0}^n \left(b_k \prod_{j=k+1}^n \theta_j \right)$$

Volume Rendering

- Emission Absorption Model



$$I(s) = I(s_0) e^{-\tau(s_0, s)} + \int_{s_0}^s q(\tilde{s}) e^{-\tau(\tilde{s}, s)} d\tilde{s}$$

- Discretization

$$\tilde{C} = \sum_{i=0}^{|T / \Delta t|} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

over operator
with opacity

- Recursive computation

$$C'_i = C_i + (1 - A_i) C'_{i-1}$$

Radiant energy observed at position i
Radiant energy emitted at position i
Absorption at position i
Radiant energy observed at position $i-1$

Volume Rendering

Compositing - Evaluation of the volume rendering equation

- Back to front

$$C'_i = C_i + (1 - A_i)C'_{i-1}$$

- Front to back

$$C'_i = C'_{i-1} + (1 - A'_{i-1})C_i$$

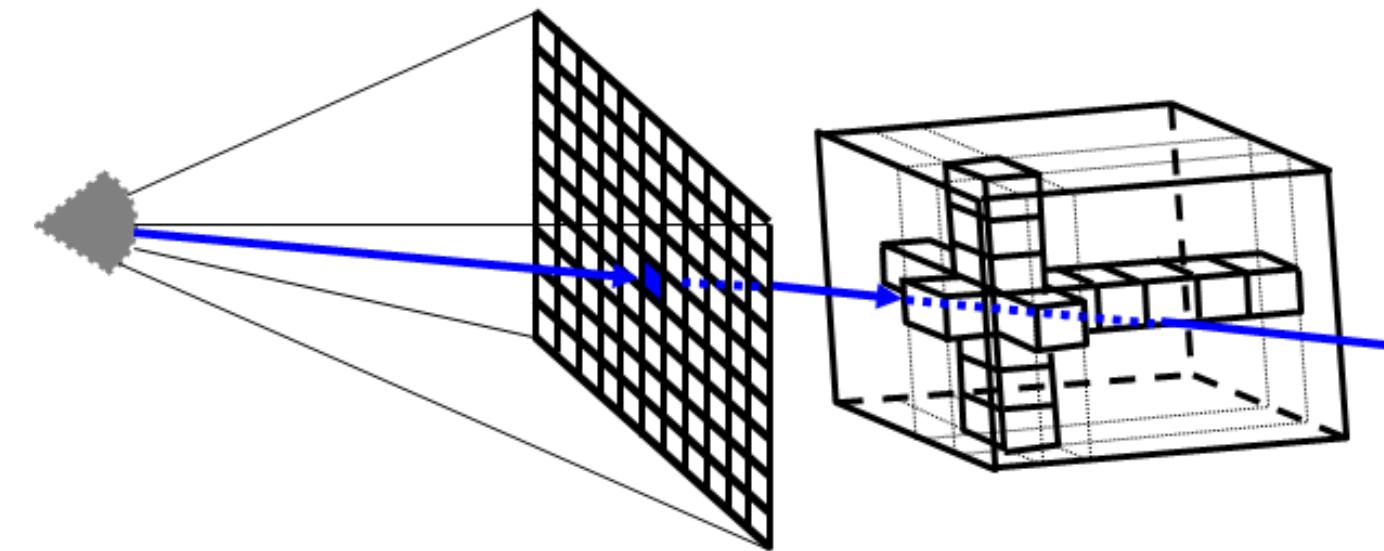
$$A'_i = A'_{i-1} + (1 - A'_{i-1})A_i$$

6.5.2 Direct Volume Rendering

Main Strategies

Image space / image order algorithms

- Performed pixel by pixel of resulting image



```
for (each pixel on image plane)
  for (each sample point on viewing ray)
    compute contribution to pixel (color, opacity)
```

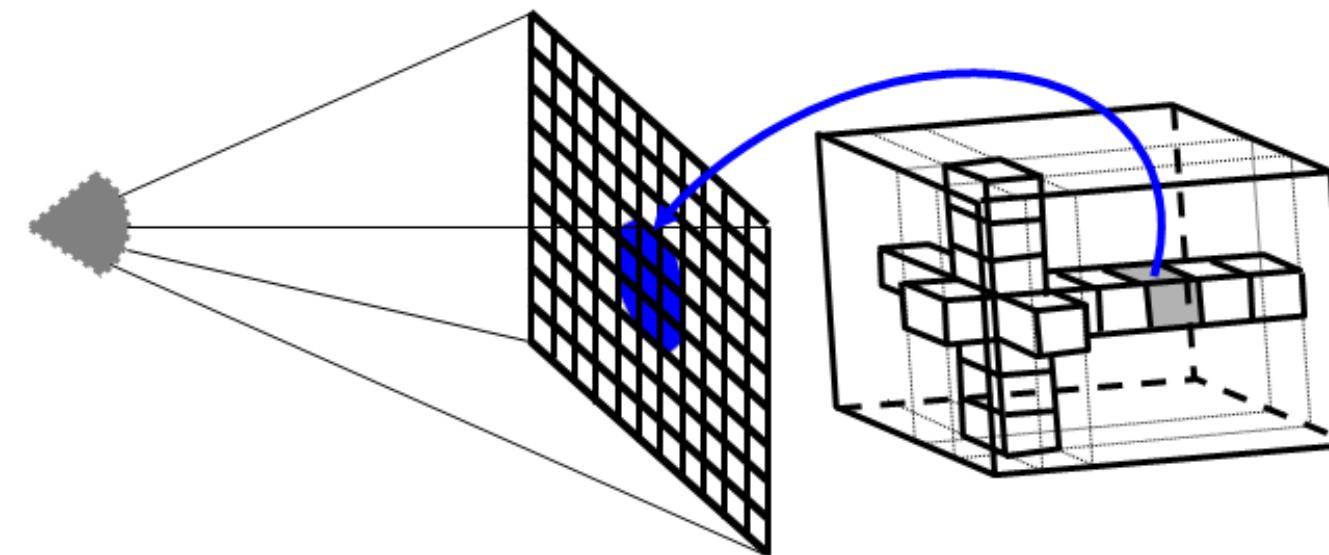
- Example: ray casting

Main Strategies

Object space / object order algorithms

- Performed voxel by voxel of volume data

Projizieren Objekt auf die Bildfläche



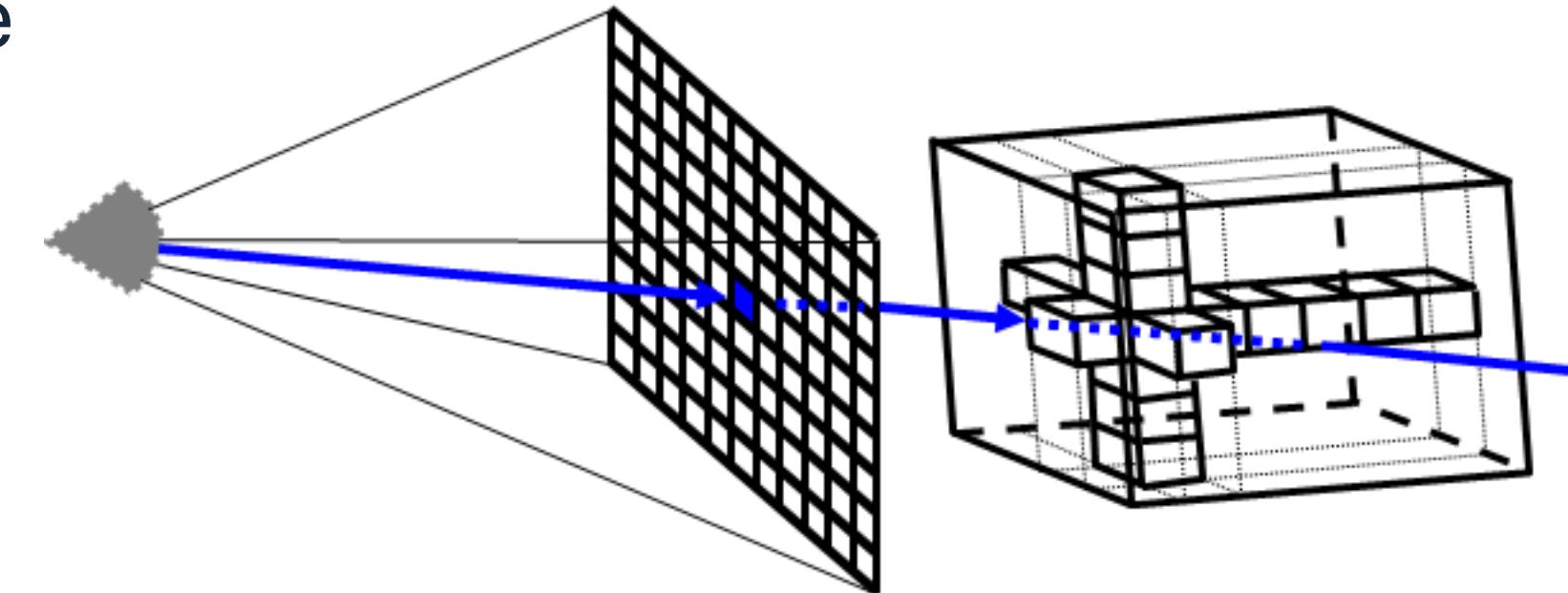
- Example: projection

```
for (each voxel in volume data)
  for (each pixel of image plane projected onto)
    compute contribution to pixel (distribute info!)
```

Raycasting

Raycasting

- Characteristics
 - Similar to ray tracing in surface-based graphics
 - In volume rendering, only primary rays are used (speedup)
 - Hence: ray casting
 - Natural image order technique

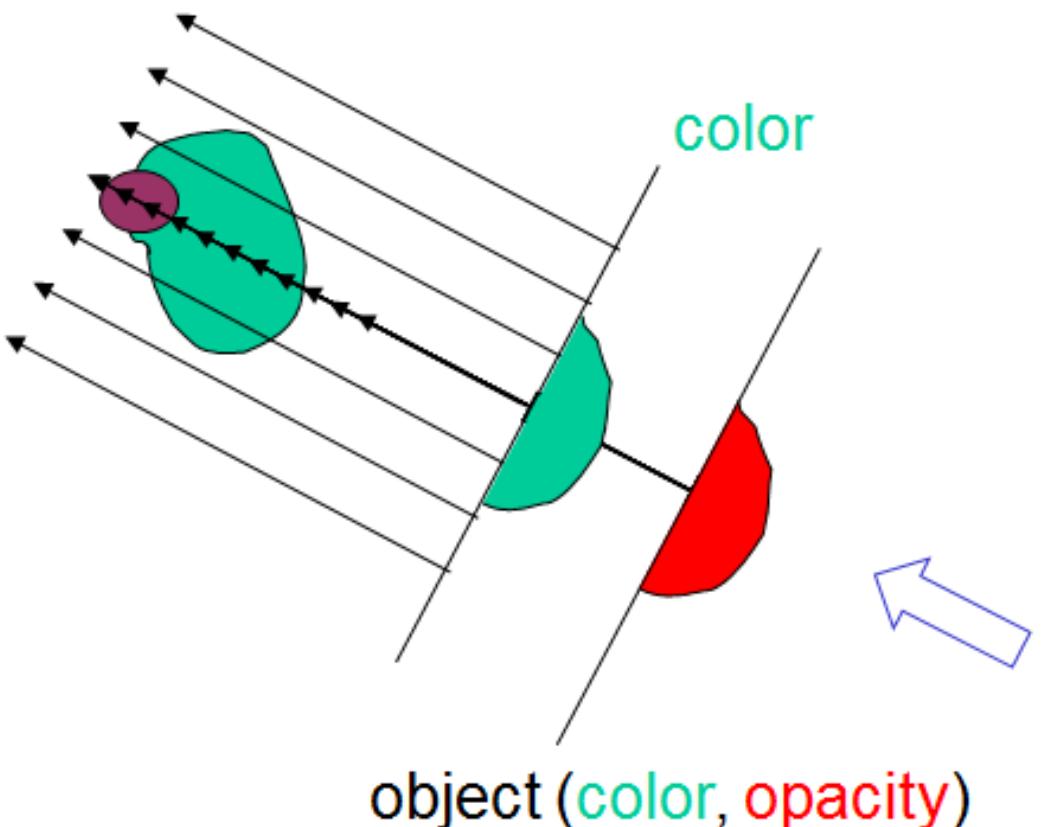


Raycasting

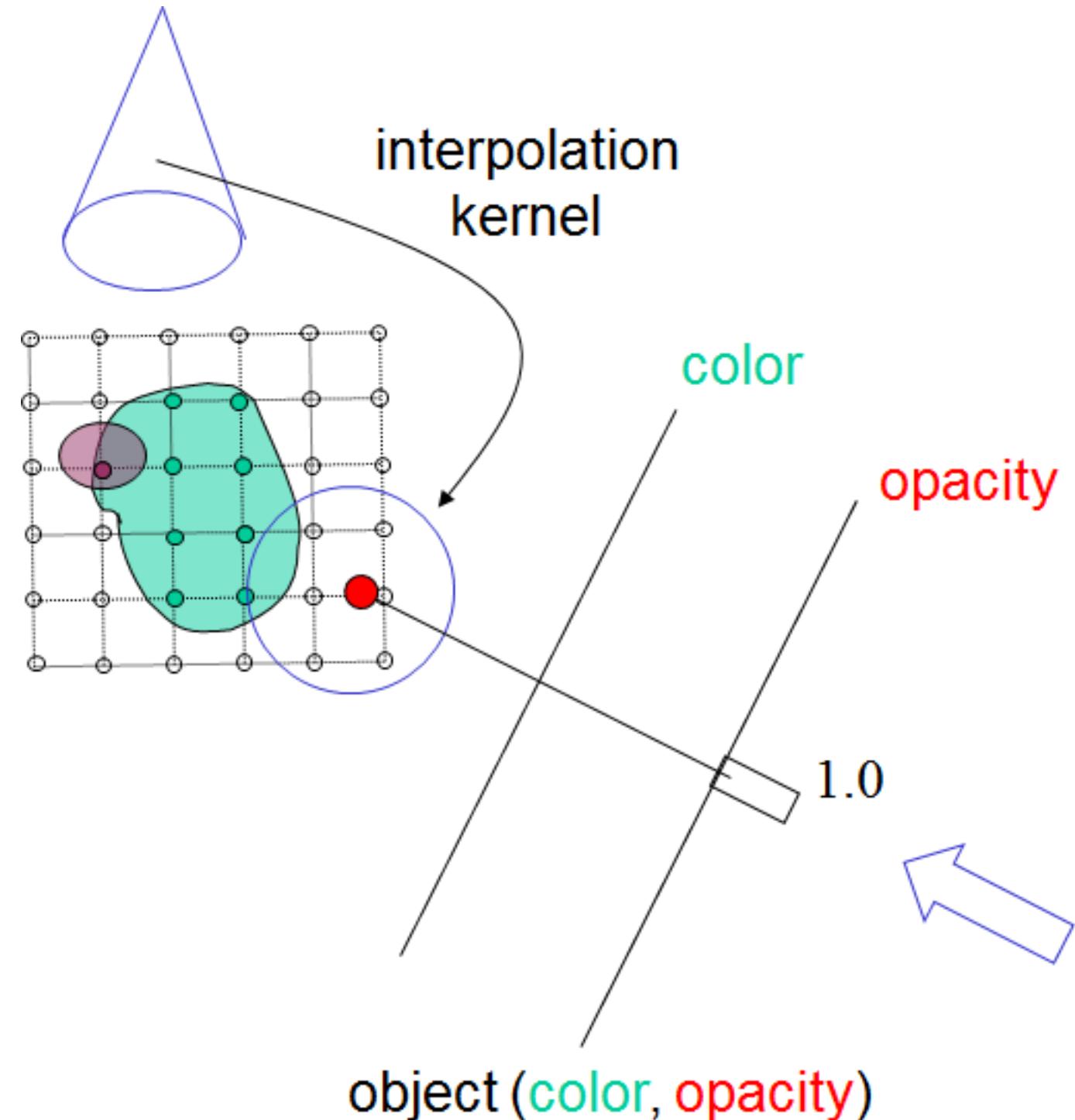
- Approach
 - Cast a ray into the volume
 - Sampling at certain intervals
 - Often equidistant sampling
 - More sophisticated sampling schemes exist
 - At each sampling location
 - Interpolation from voxel grid
 - Nearest neighbor
 - Trilinear
 - Or more sophisticated (Gaussian, cubic spline)

Raycasting

- Volumetric ray integration (*compositing*)
 - Tracing of rays
 - Accumulation of color & opacity along ray

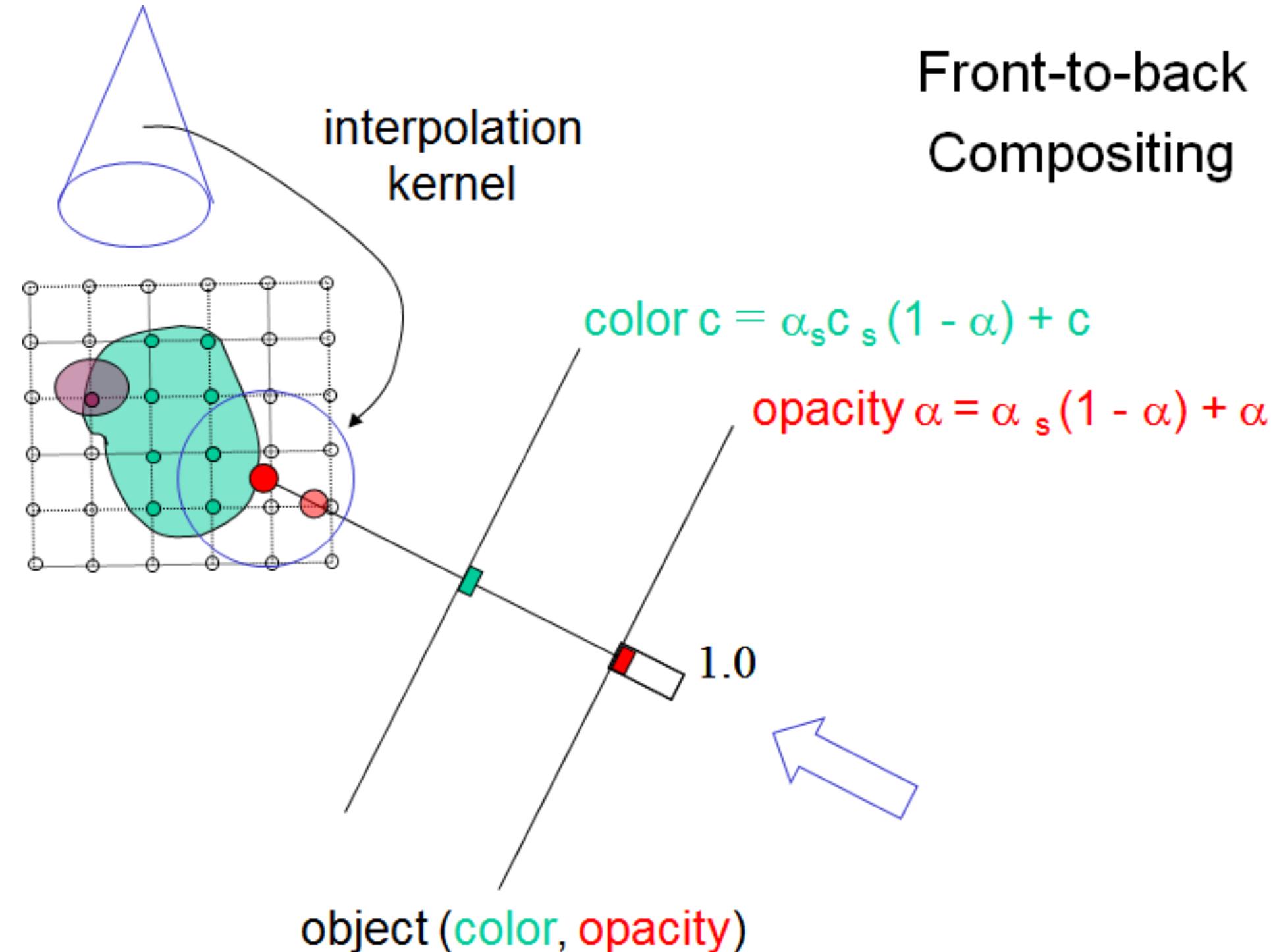


Raycasting



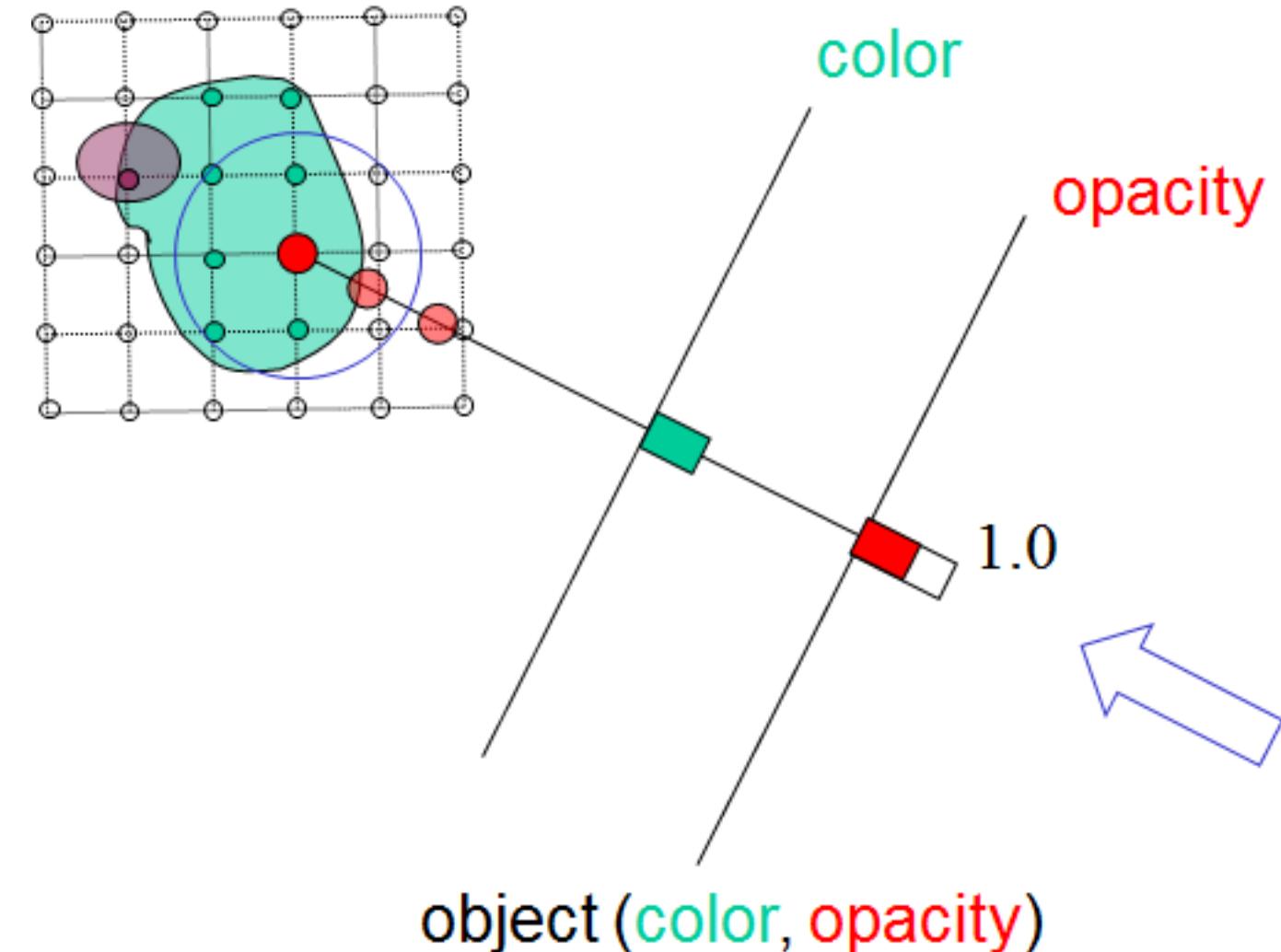
Volumetric
Compositing

Raycasting



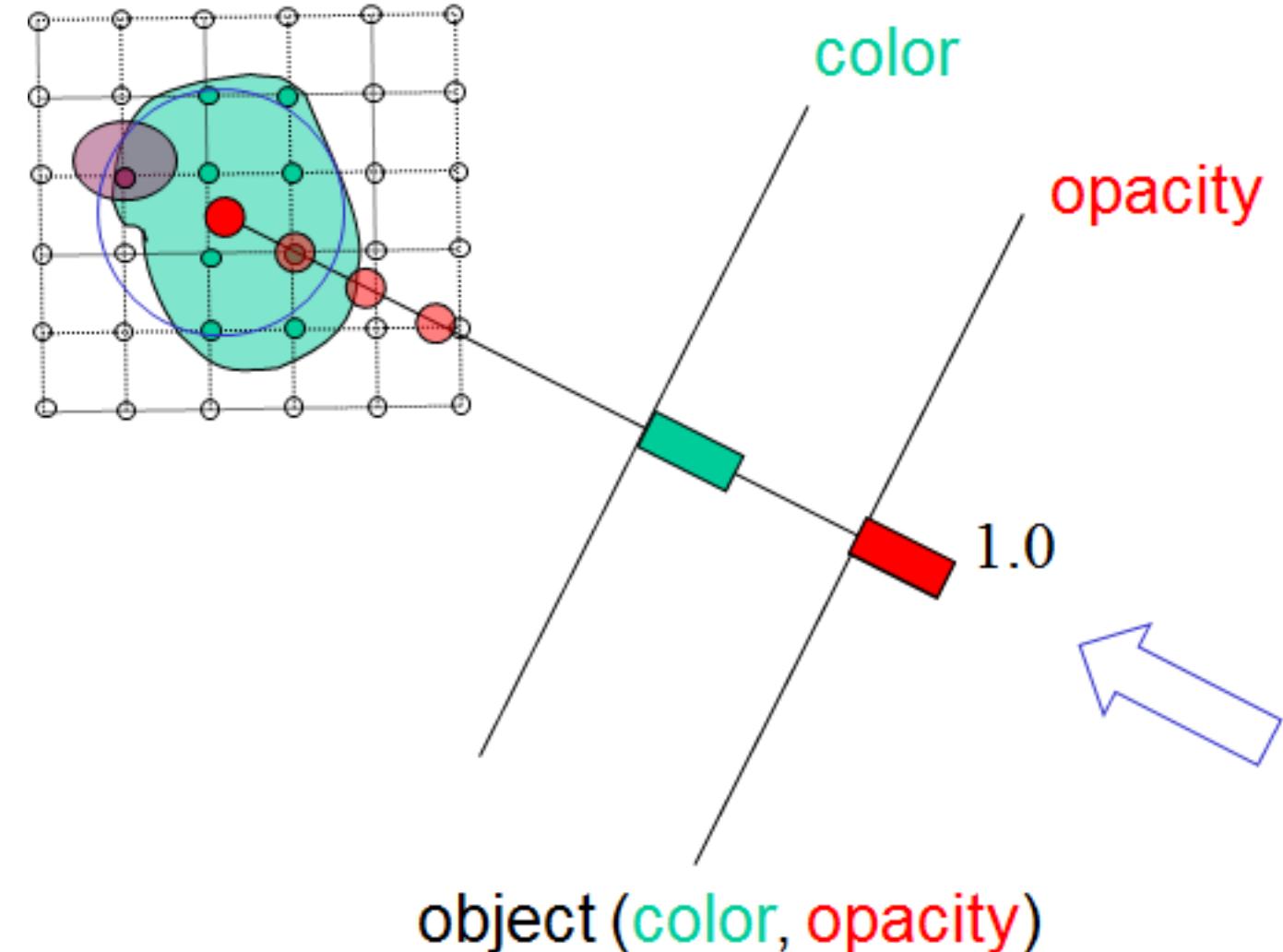
Raycasting

Front-to-back
Compositing



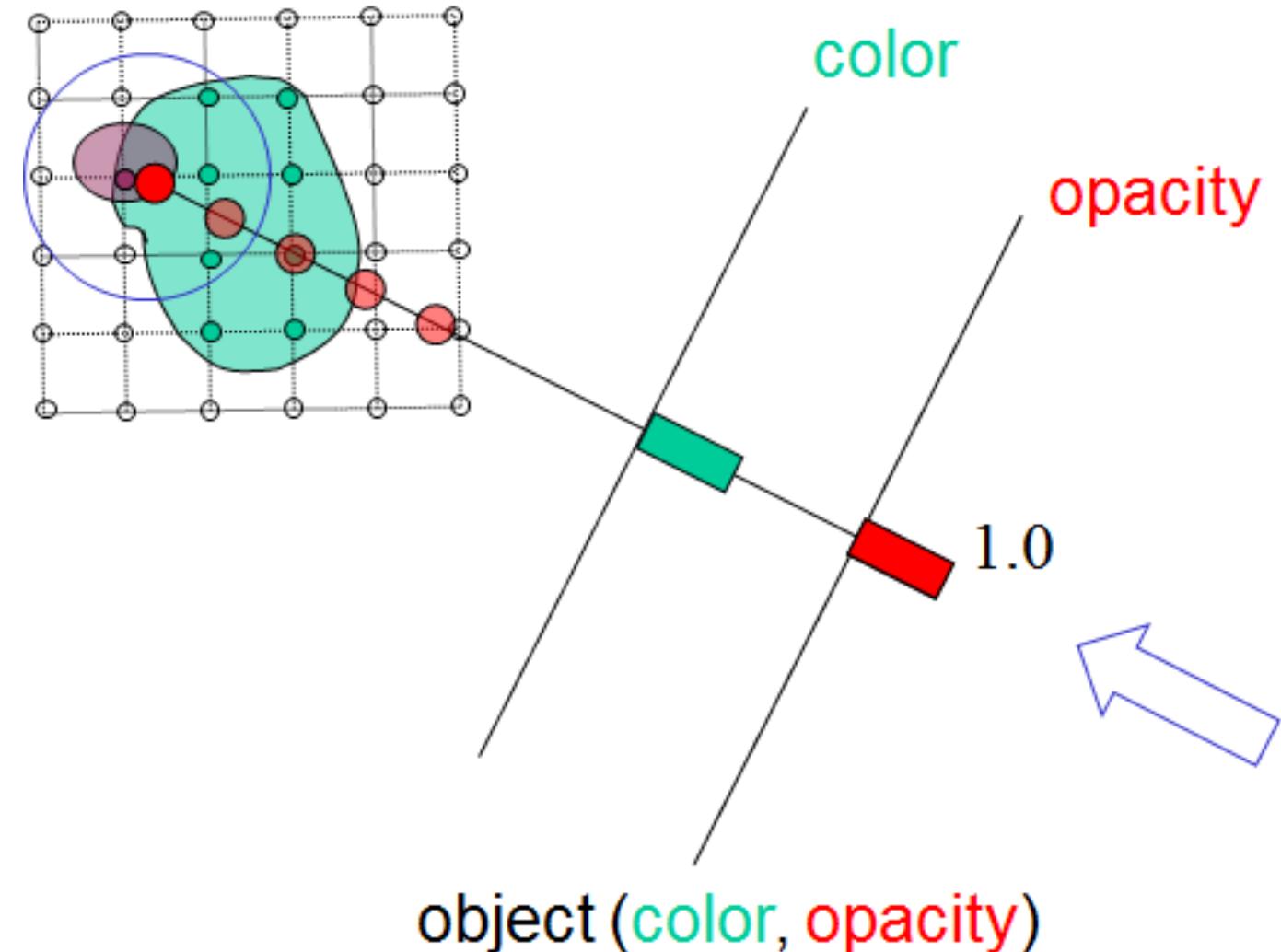
Raycasting

Front-to-back
Compositing



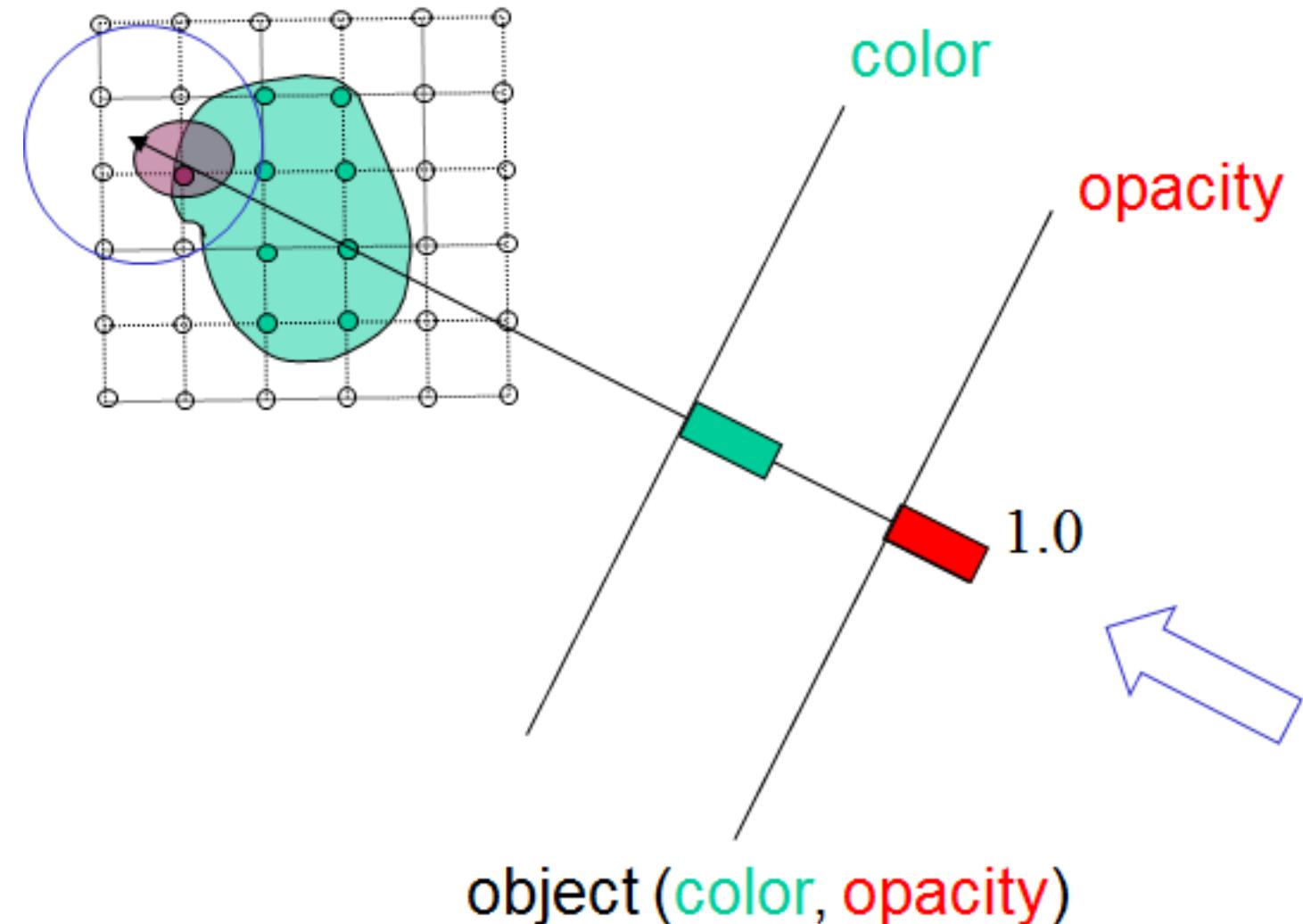
Raycasting

Front-to-back
Compositing



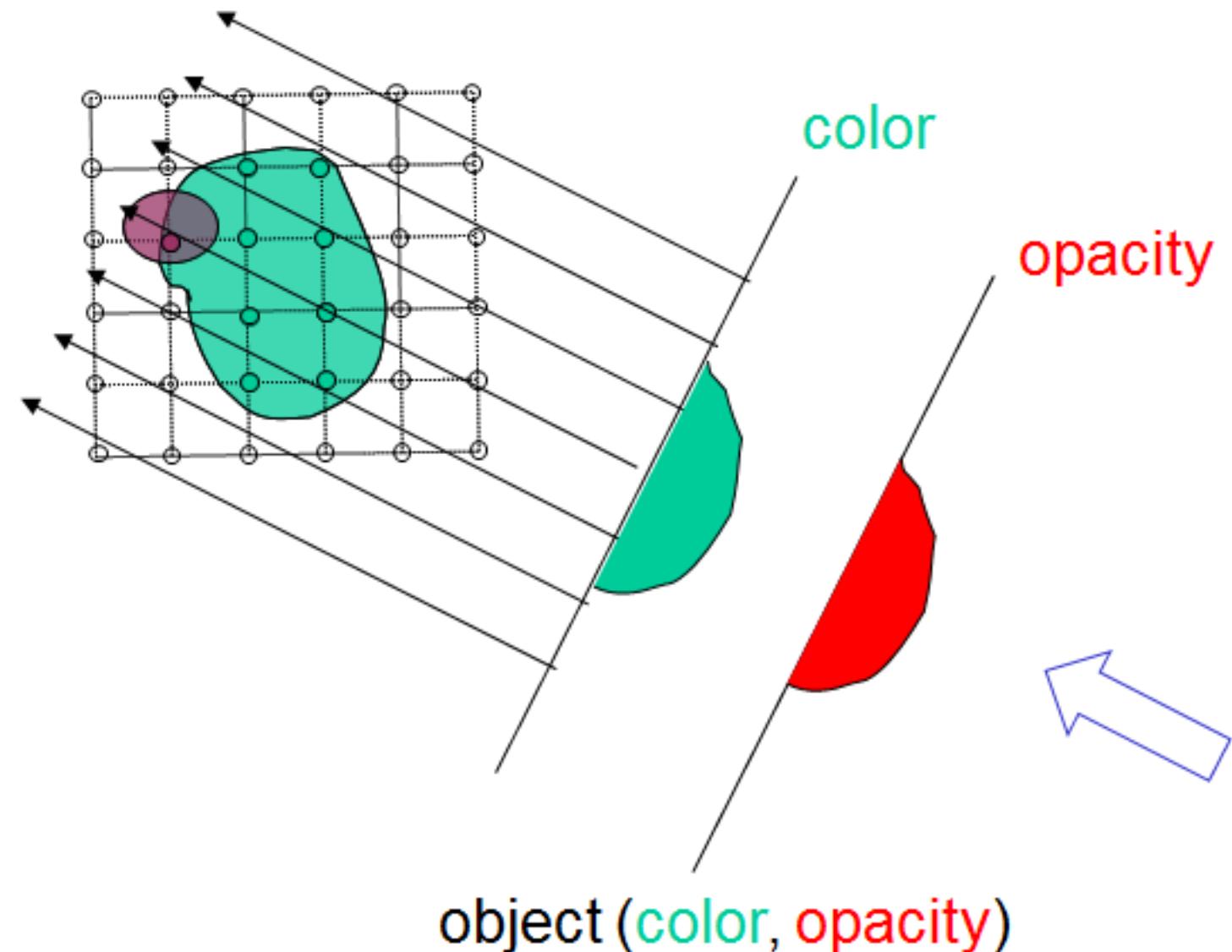
Raycasting

Front-to-back
Compositing



Raycasting

Front-to-back
Compositing



Raycasting

- Processing steps
 - Ray discretization
 - Sampling of intensities
 - Assignment of color and opacity
 - Adjustment of transfer functions
 - Integration

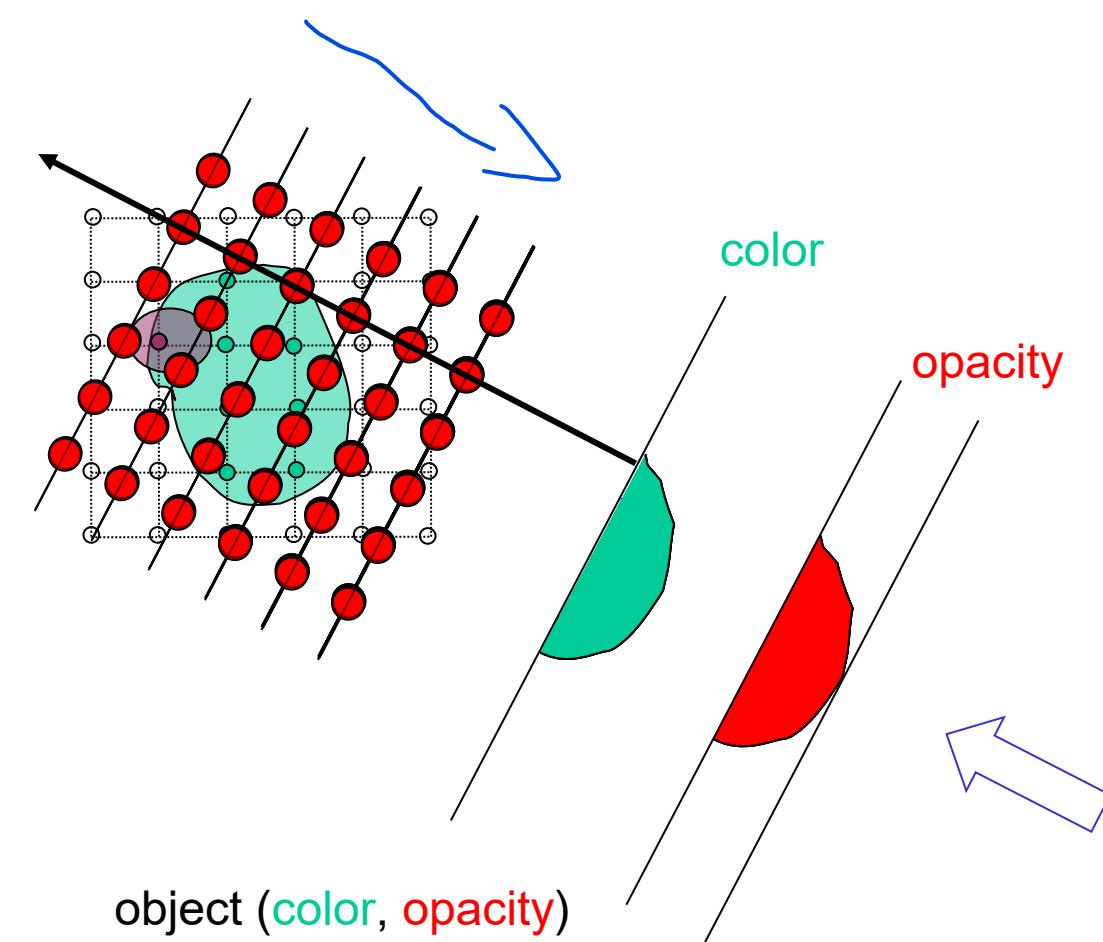
Texture-based Volume Rendering

Texture-based DVR

- Characteristics
 - Object order approach
 - Based on graphics hardware
 - Rasterization
 - Texturing
 - Blending
 - Use proxy geometry
 - Due to missing volumetric primitives
 - Calculate slices through volume

Texture-based DVR

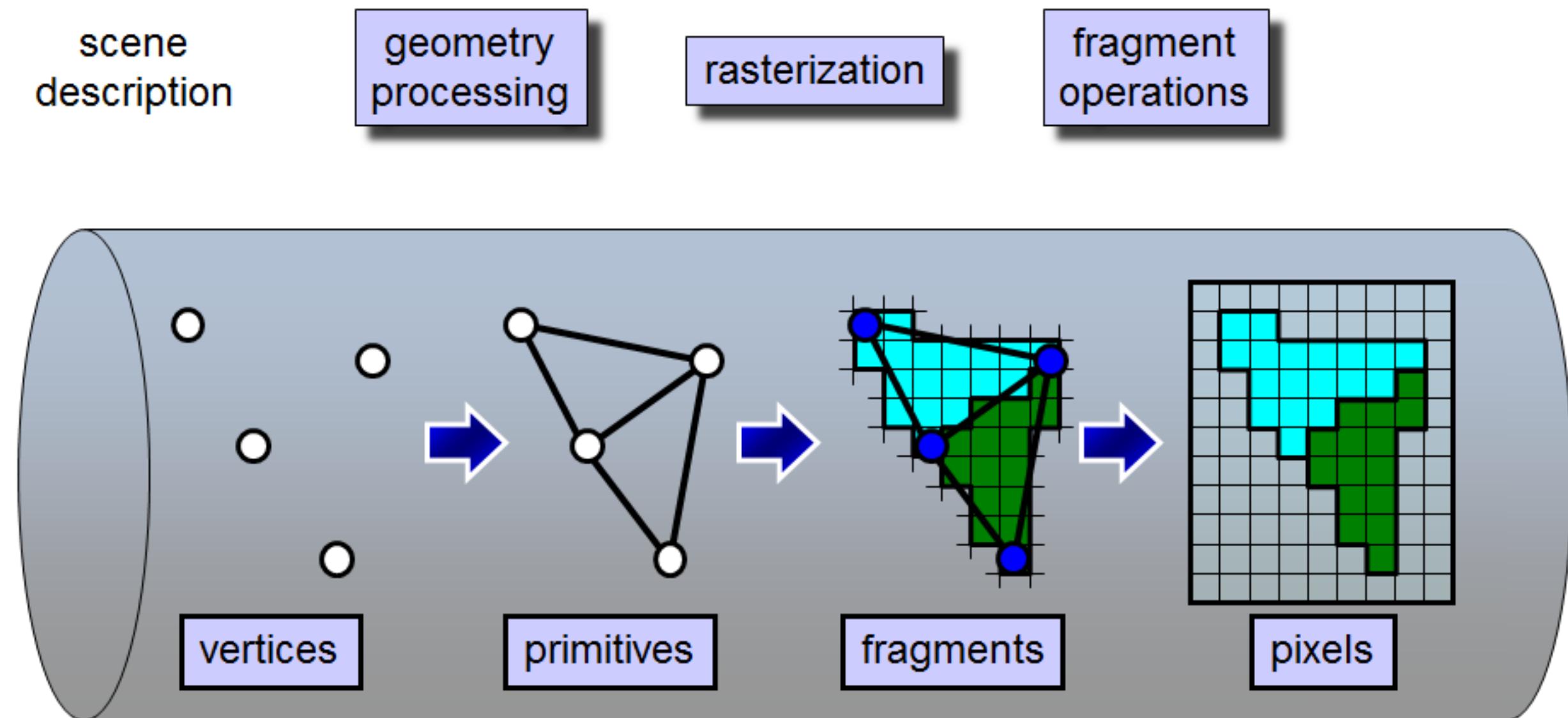
- Slice-based rendering



Ergebnis => Überlagerung der Textur durch die Schnitte im Volumen (Back to Front)

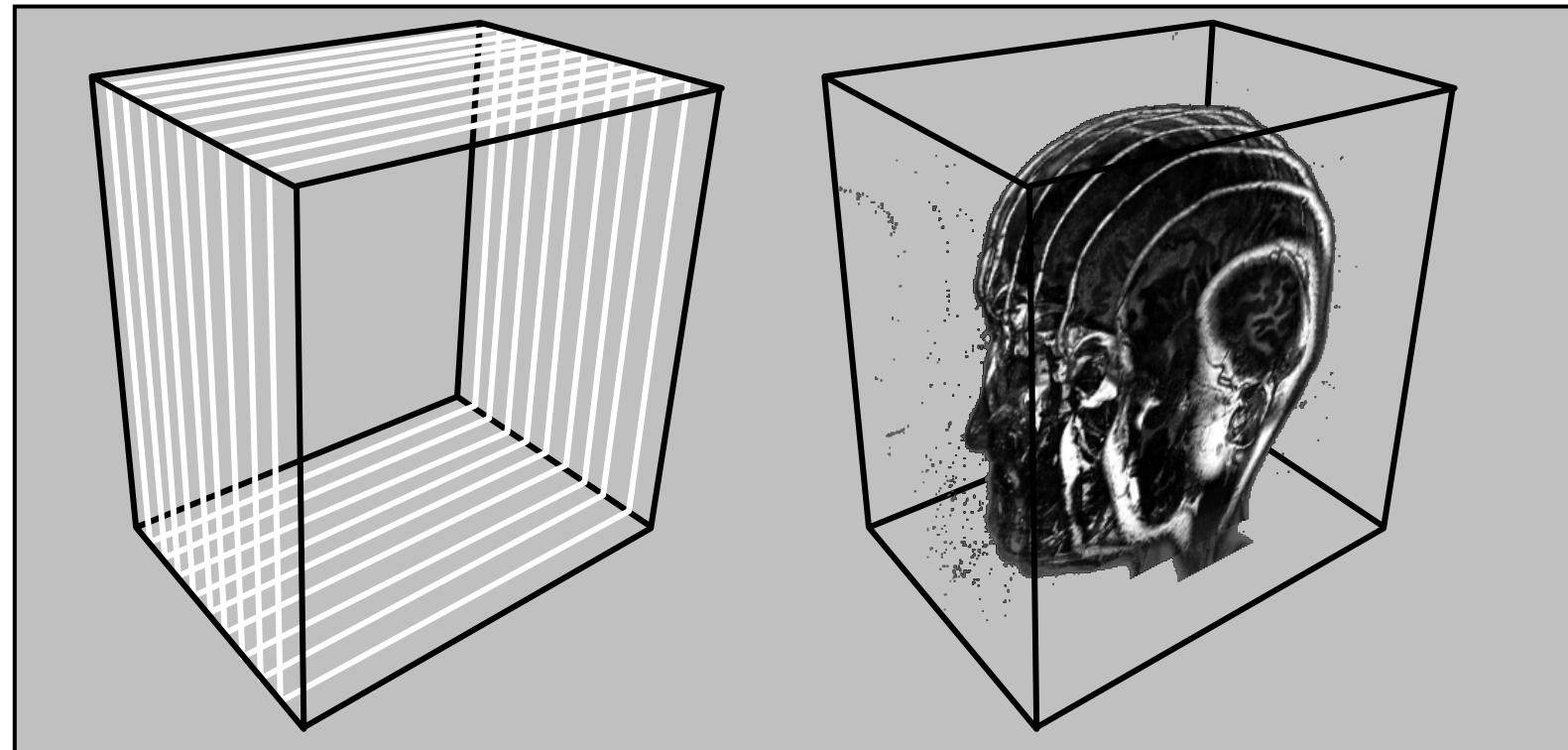
Texture-based DVR

- Rendering pipeline (recap)



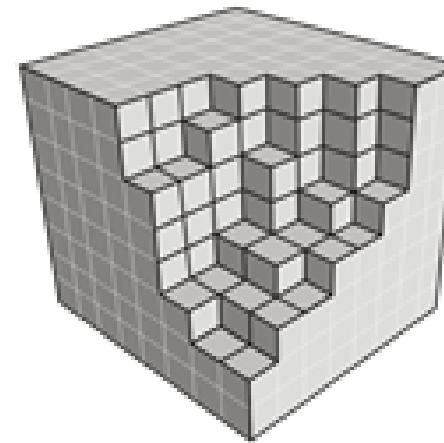
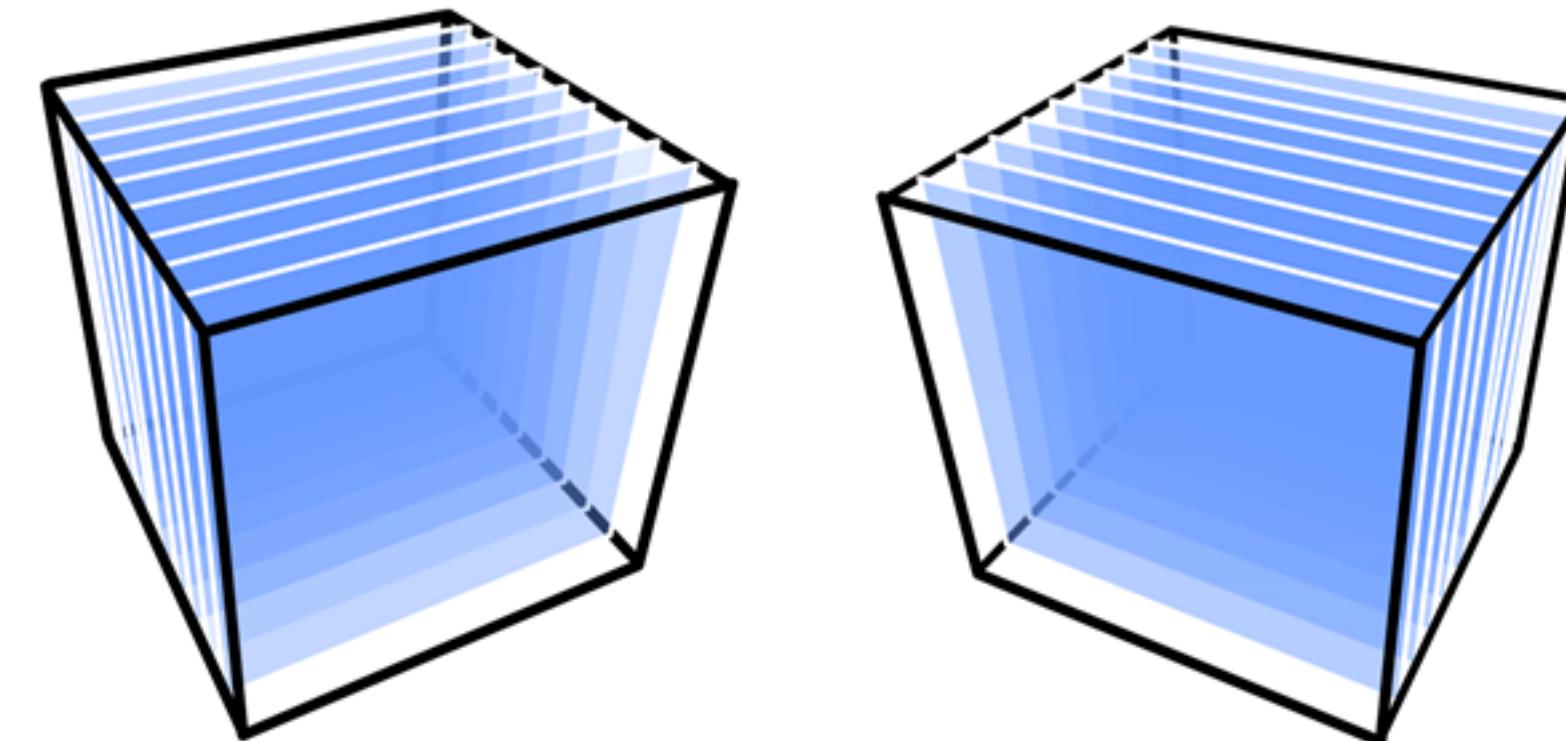
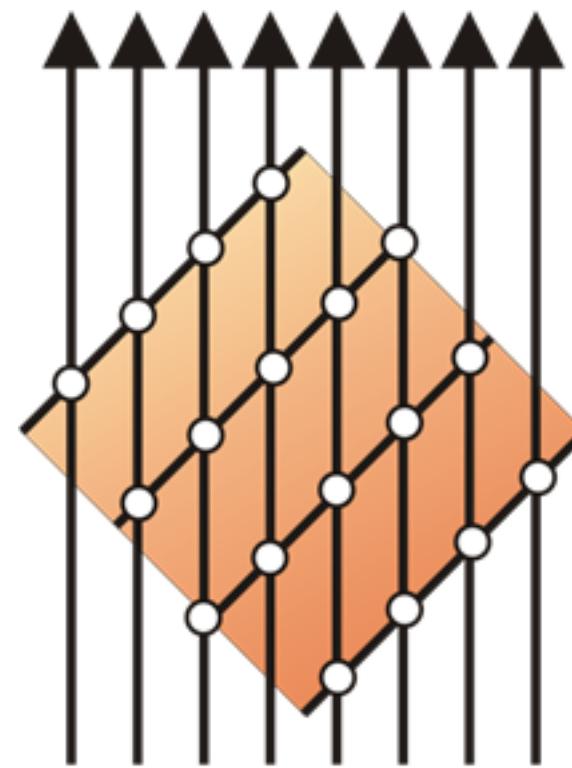
Texture-based DVR

- Proxy geometry
 - Stack of texture-mapped slices
 - Generate fragments
 - Most often back-to-front traversal



Texture-based DVR

- 2D textured slices
 - Object-aligned slices
 - Three stacks of 2D textures
 - Bilinear interpolation

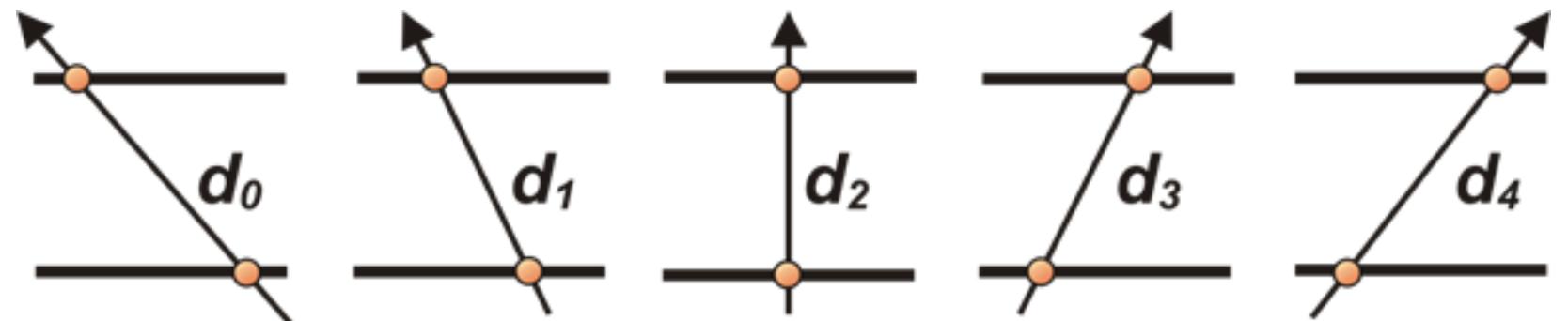


Note: Scalar data on
uniform rectilinear grid

Texture-based DVR

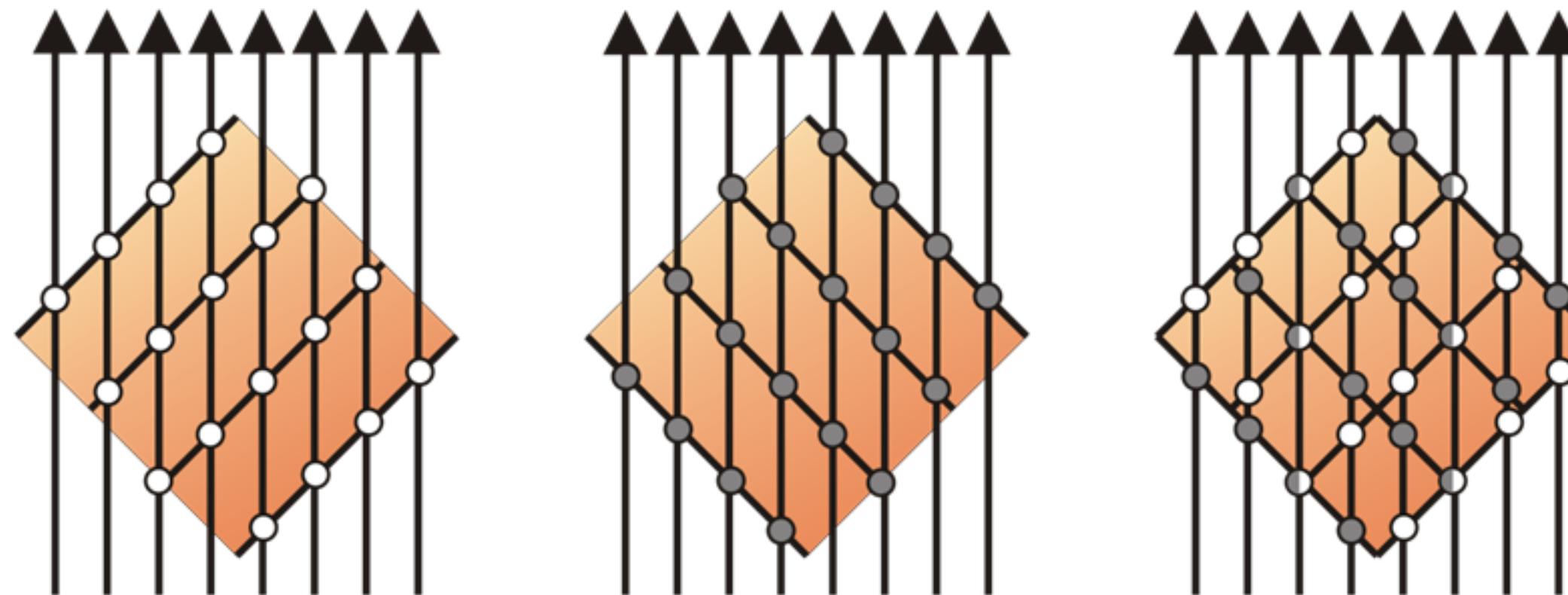
- Stack of 2D textures
 - Works on older graphics hardware without 3D textures
 - Needs 3 stacks of textures: 3 times texture memory!
 - Only bilinear interpolation within slices
 - Fast
 - Problems with image quality
 - Sampling distance between slices along a ray
 - Depends on viewing direction
 - Apparent brightness changes if opacity is not corrected

abhängig von Blickwinkel damit nicht zwischen die Schichten geschaut werden kann



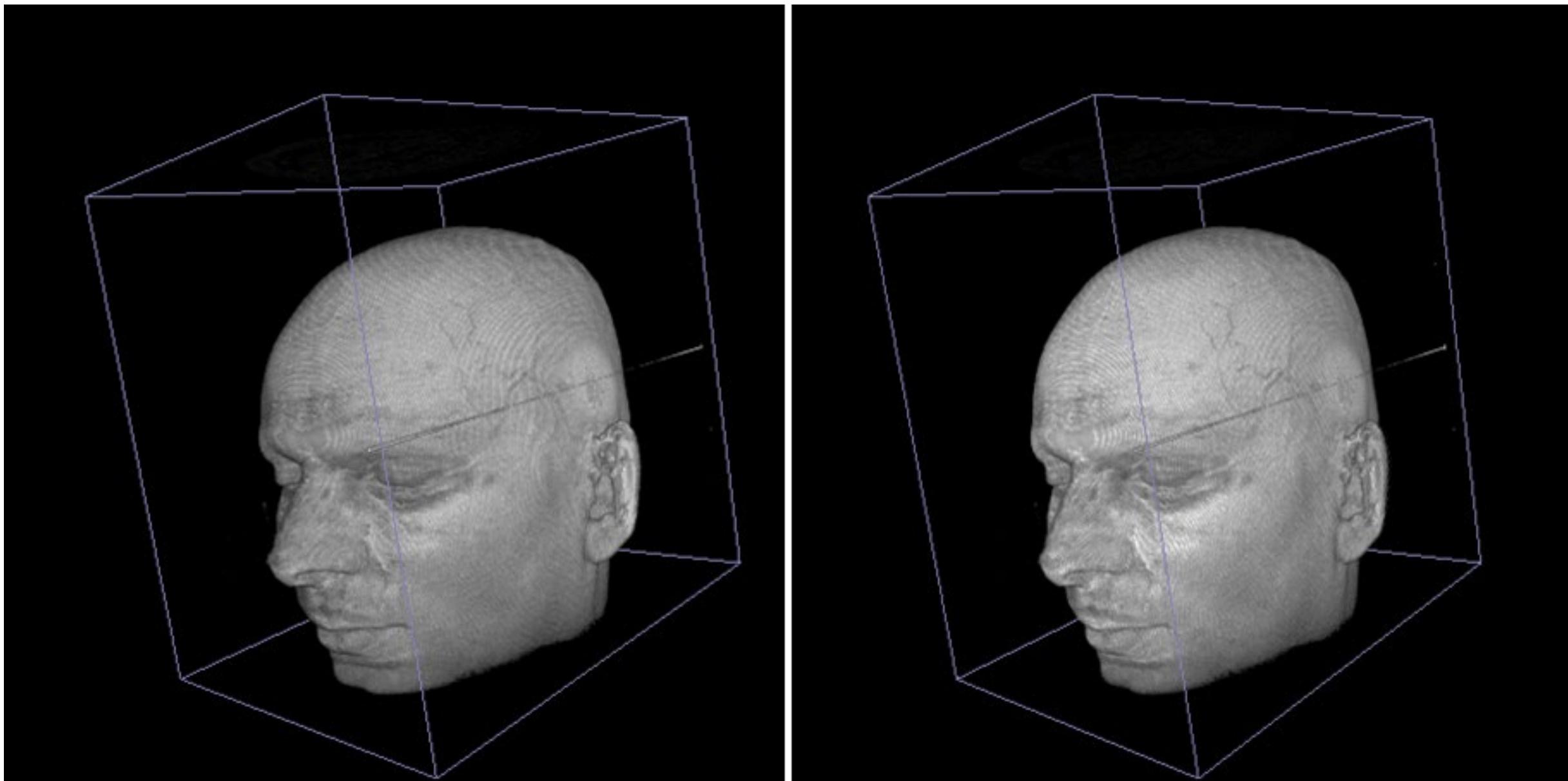
Texture-based DVR

- Stack of 2D textures
 - Artifacts when stack is viewed close to 45 degrees
 - Locations of sampling points may change abruptly



Texture-based DVR

- Brightness changes in 2D texture rendering



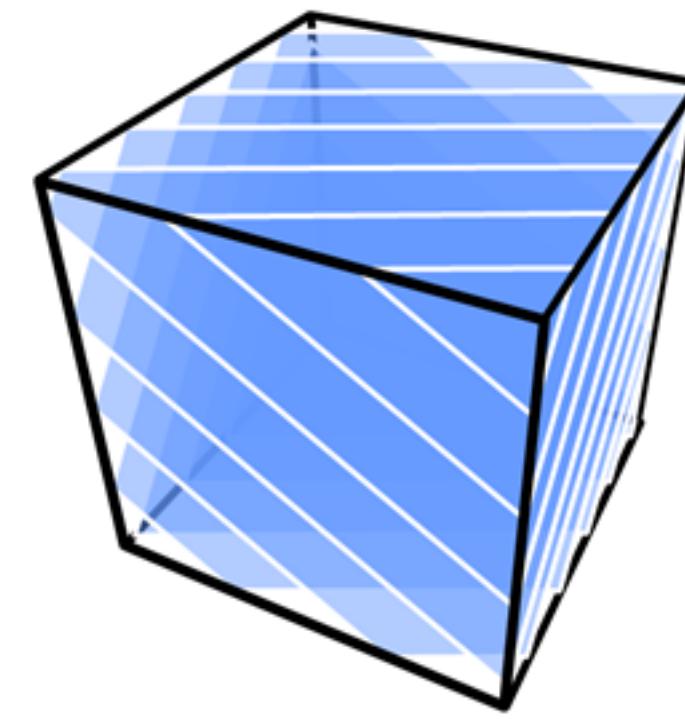
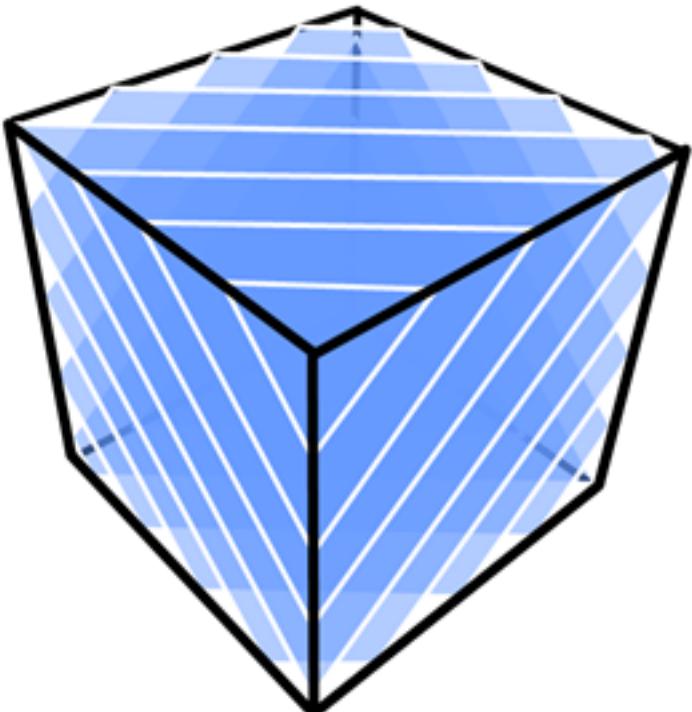
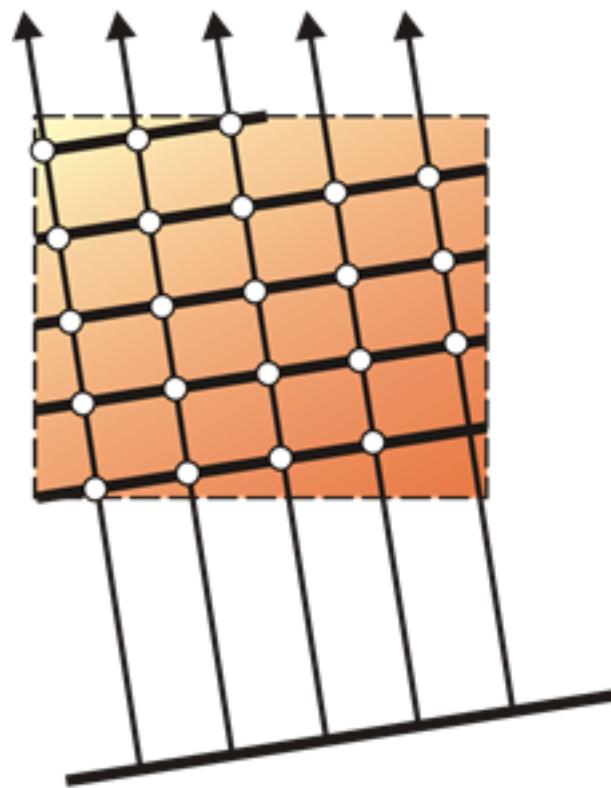
Texture-based DVR

- Artifacts
 - When stack is viewed close to 45 degrees



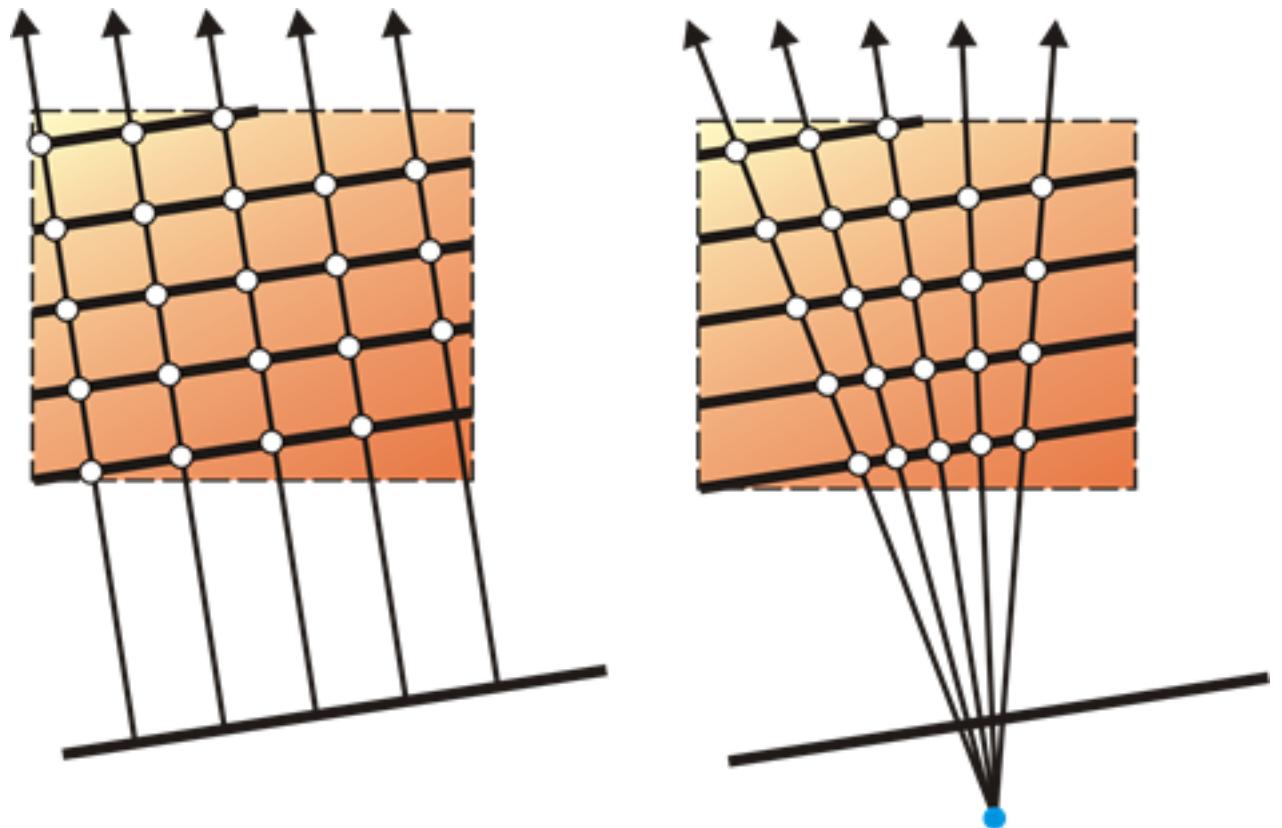
Texture-based DVR

- 3D textured slices
 - View-aligned slices
 - Single 3D texture
 - Trilinear interpolation



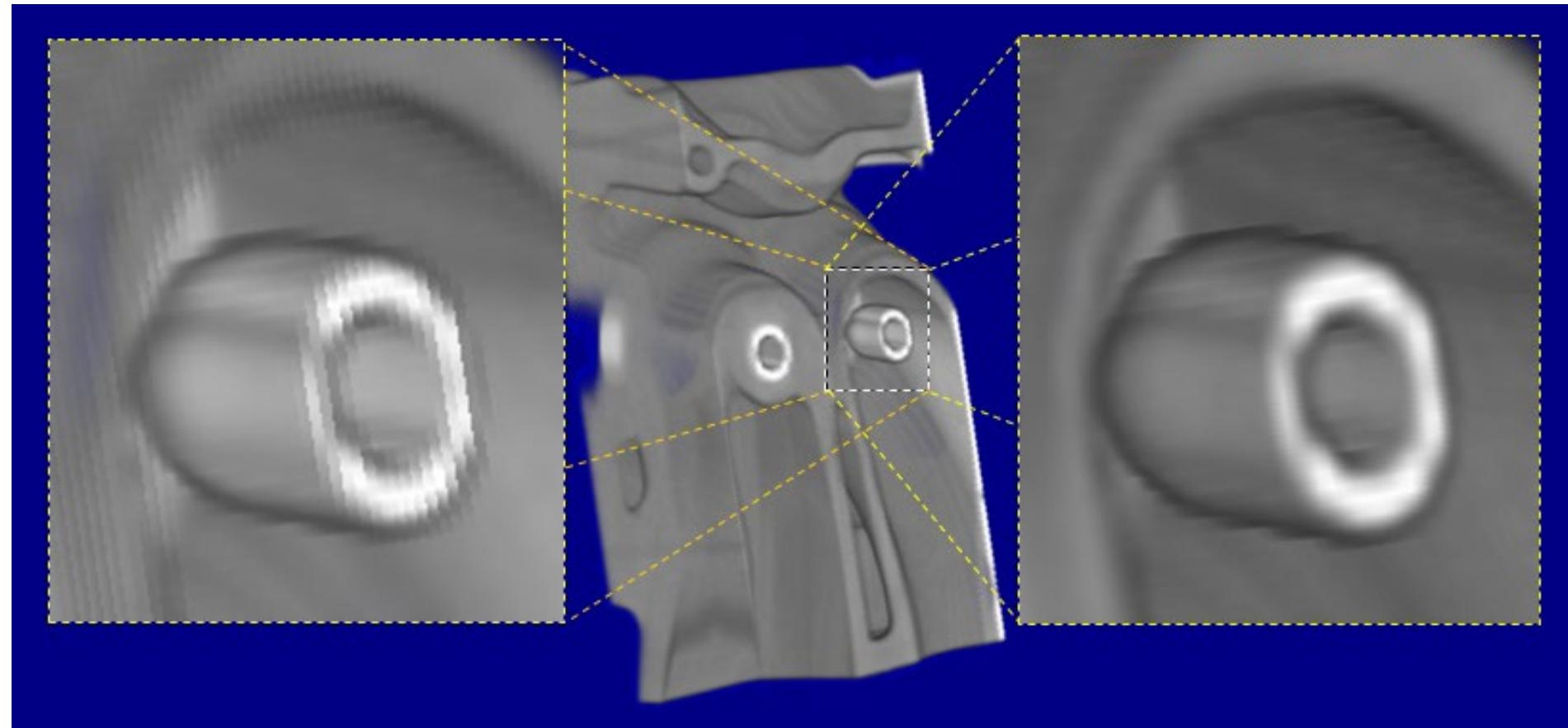
Texture-based DVR

- 3D textures
 - Needs support for 3D textures
 - Data set stored only once (not 3 stacks!)
 - Trilinear interpolation within volume
 - Slower
 - Good image quality
 - Constant Euclidian distance between slices along a ray
 - Constant sampling distance (except for perspective projection)

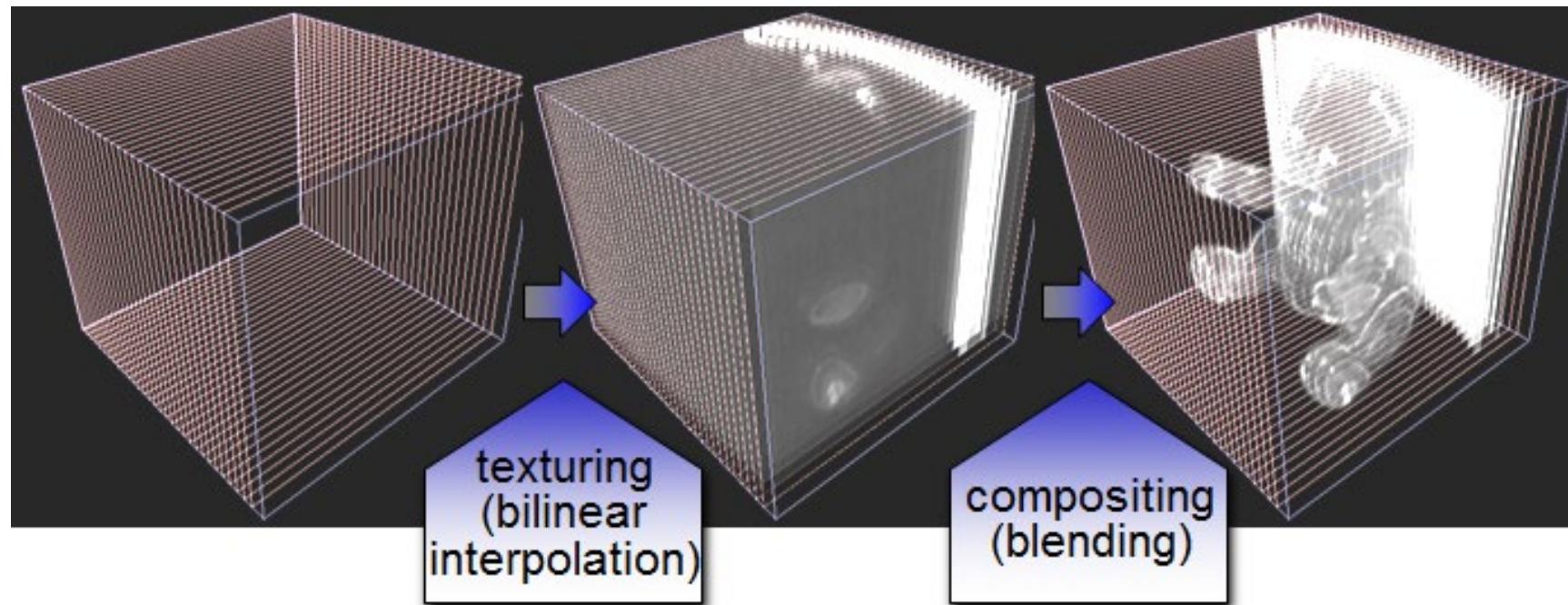


Texture-based DVR

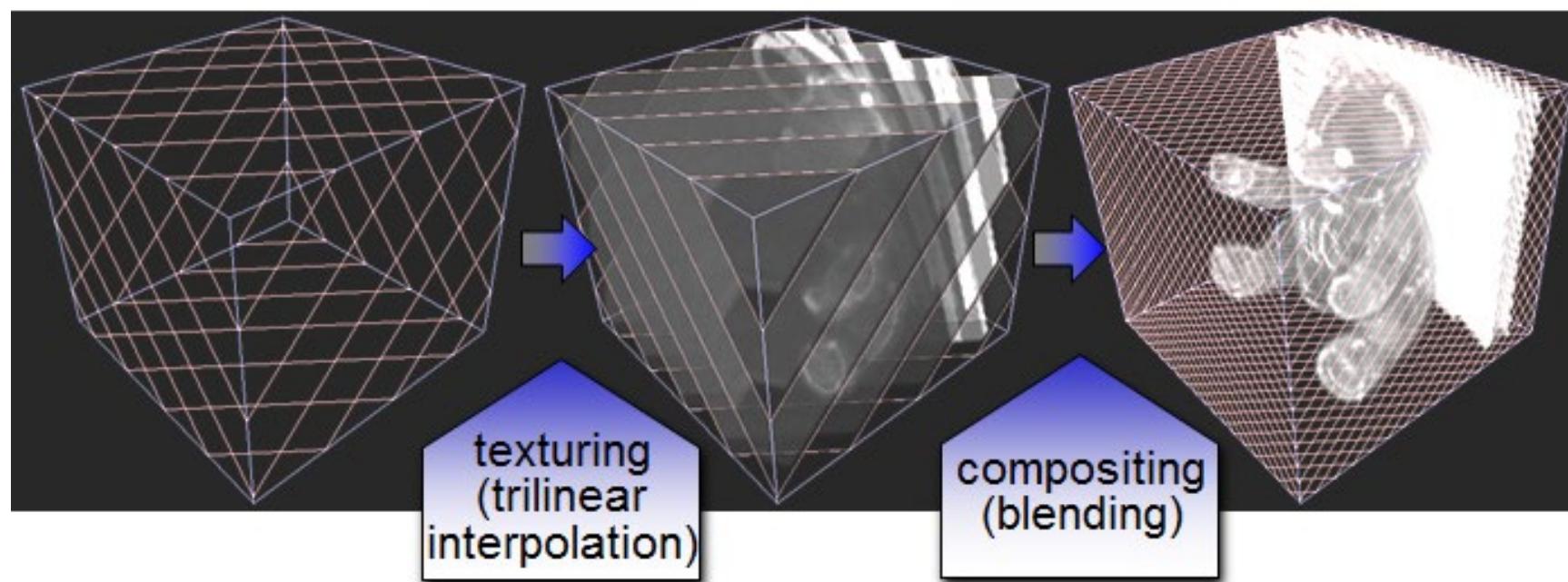
- 3D textures
 - No artifacts due to inappropriate viewing angles
 - Increase sampling rate → more slices
 - Easy with 3D textures



Texture-based DVR



2D textures
axis-aligned



3D texture
view-aligned

Texture-based DVR

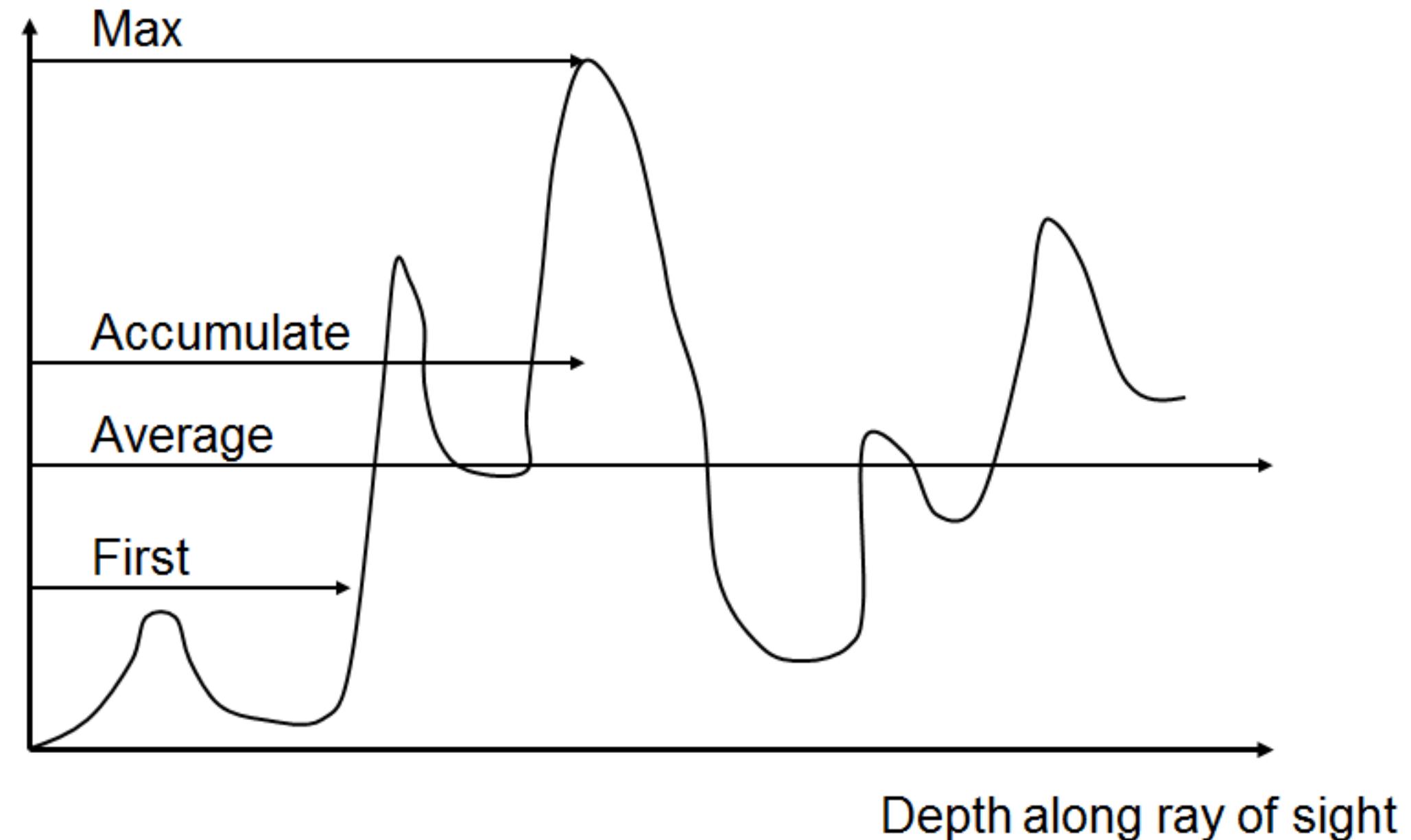
- Advantages
 - Exploits graphics hardware
 - Fast for moderately sized data sets
 - Combines surface and volumetric representations
 - Allows for mixture with opaque geometries
- Disadvantages
 - Limited by texture memory
 - Use bricking at the cost of additional texture downloads
 - Brute force representation by slices
 - No acceleration techniques as for ray casting
 - Rasterization speed and memory access can be problematic

6.5.3 (Other) Compositing Schemes

Compositing Schemes

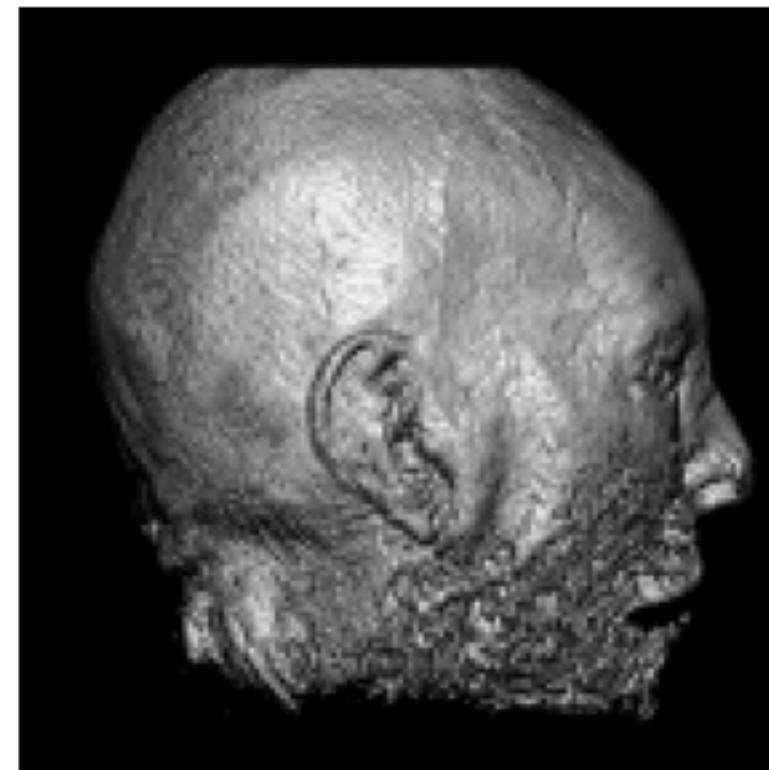
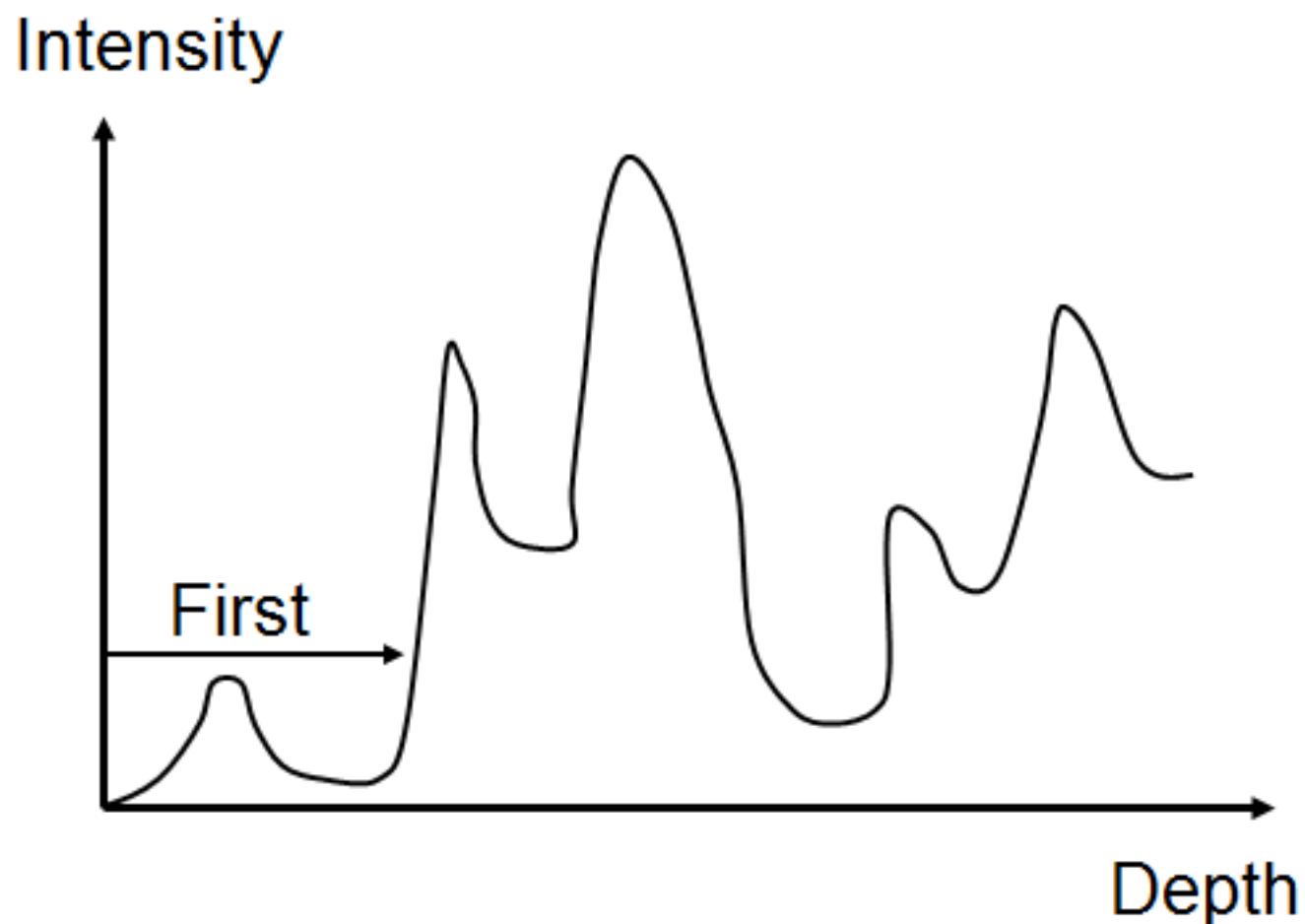
- Other approaches for determining the final color exist
 - Simpler calculations
 - Different visualization results
 - Application dependent
- Variations of composition schemes
 - First
 - Average
 - Maximum intensity projection
 - Accumulate (i.e. evaluation of the integral, chapter 6.5.2)

Compositing Schemes



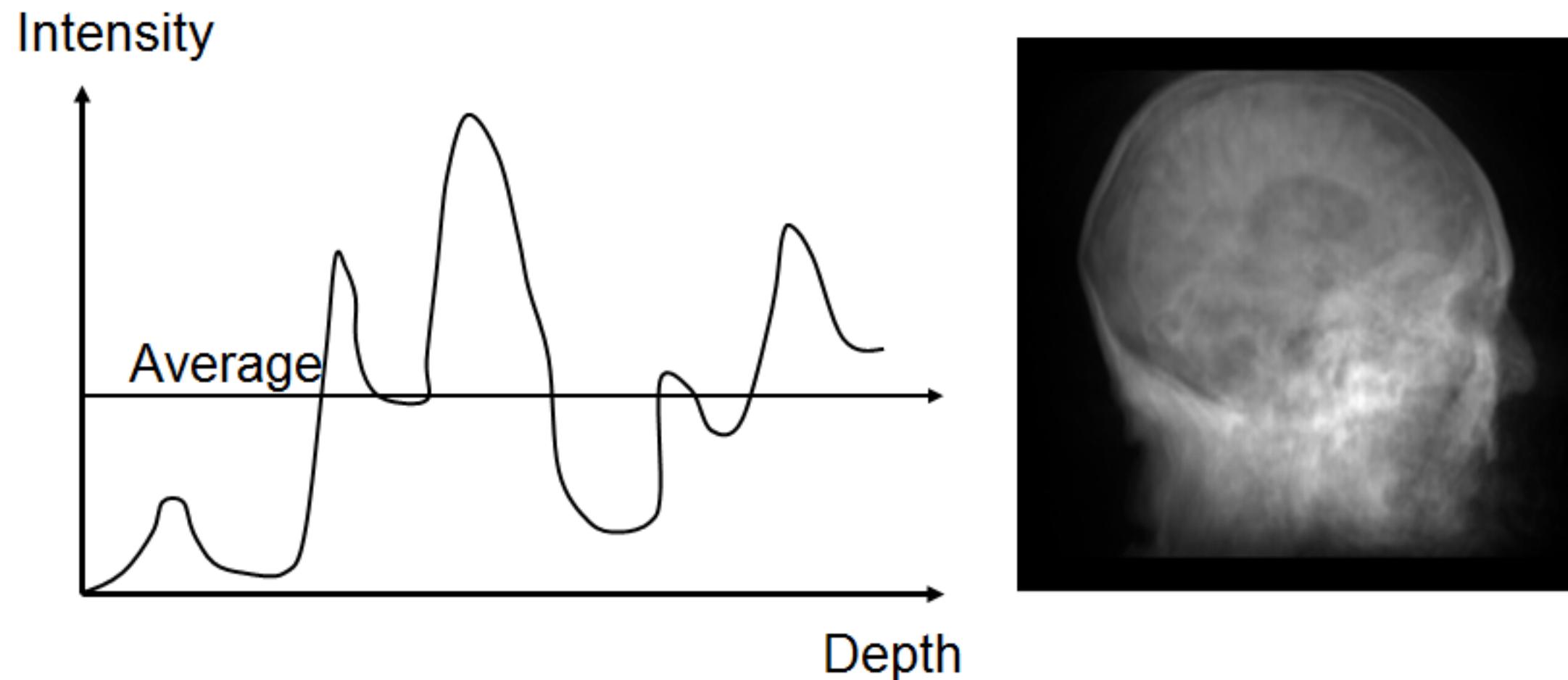
Compositing Schemes

- First hit of iso-value along a ray
 - Extracts iso-surfaces



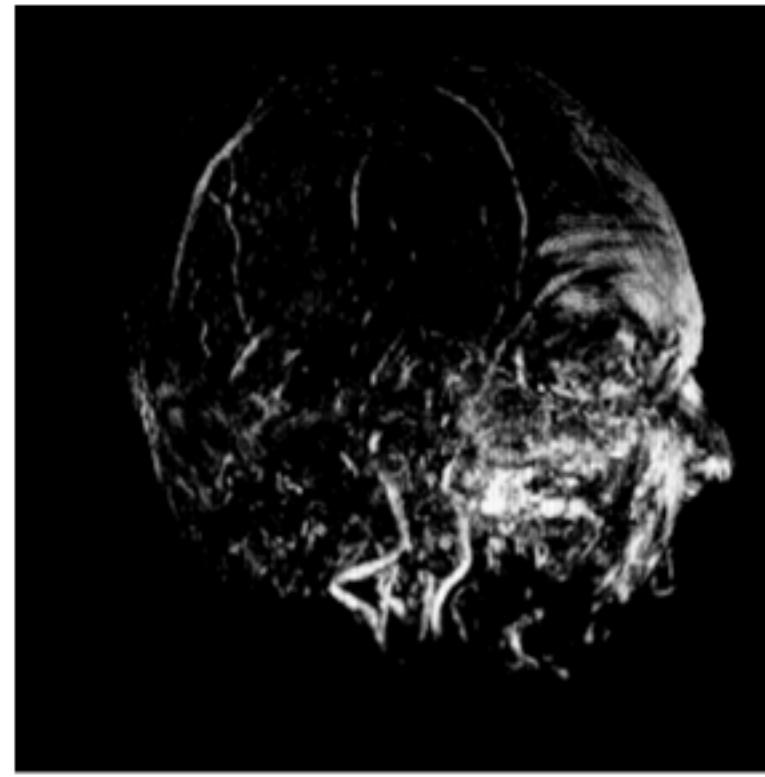
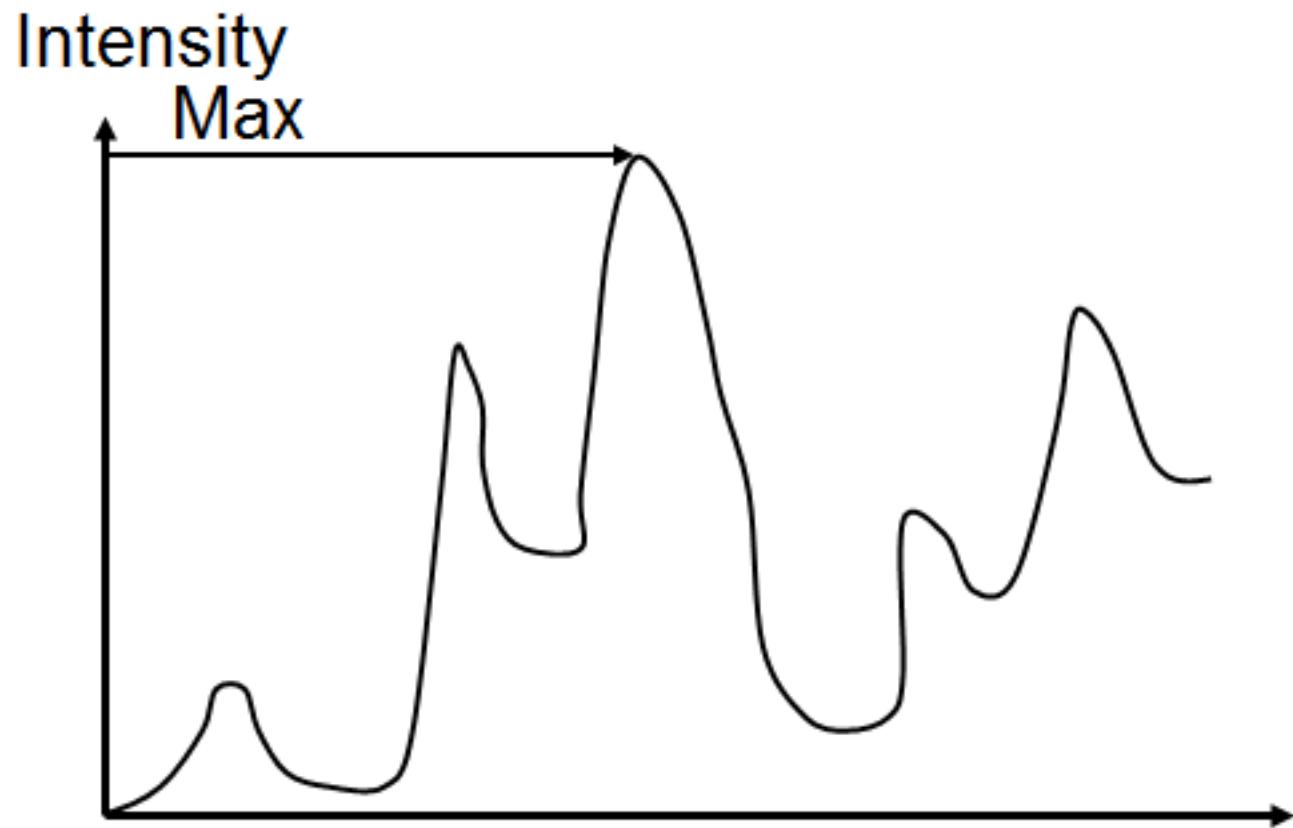
Compositing Schemes

- Average of all values along a ray
 - Produces basically an X-ray picture



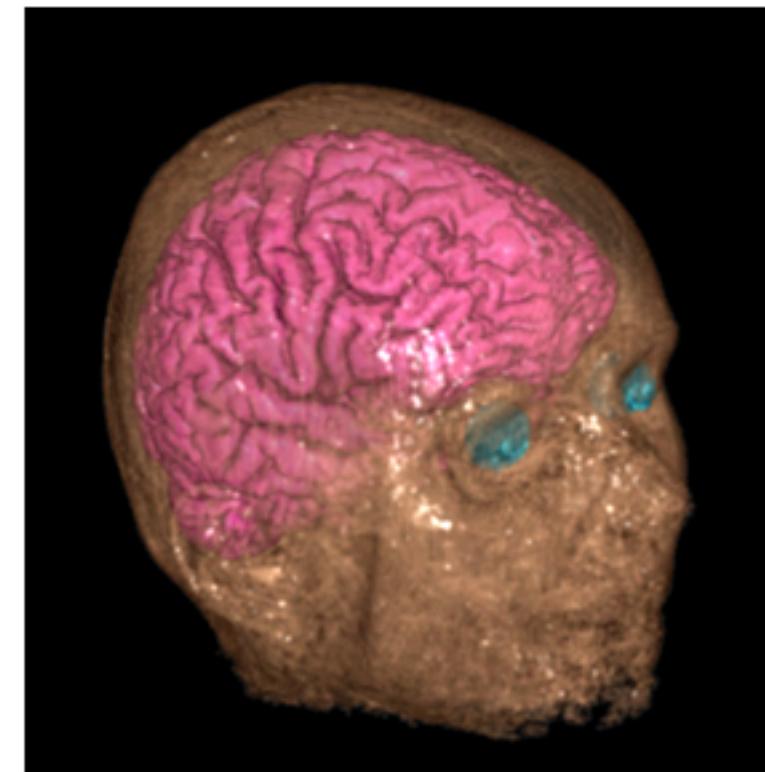
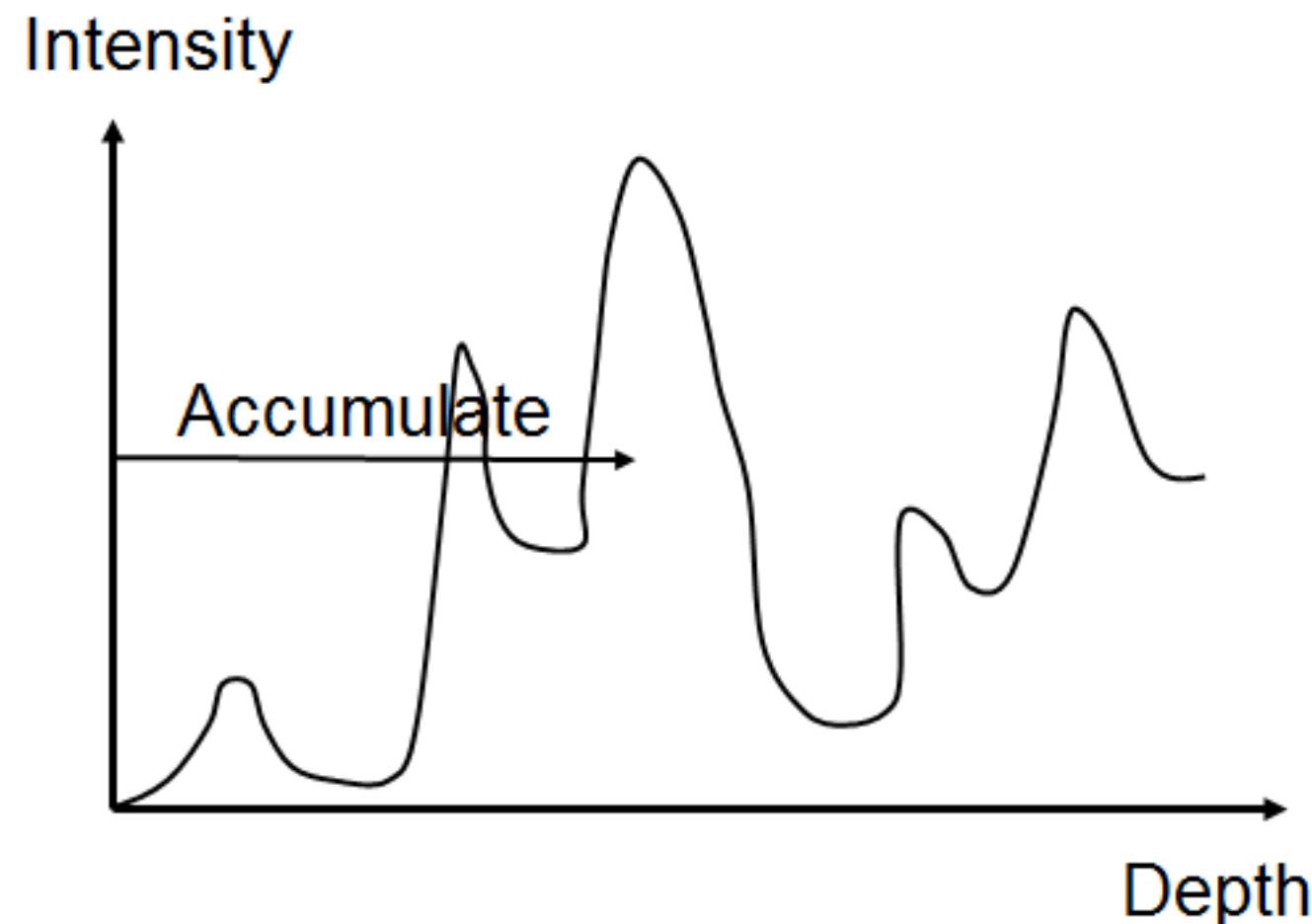
Compositing Schemes

- Maximum value along a ray
 - Maximum intensity projection (MIP)
 - Good for structures with values higher than the rest



Compositing Schemes

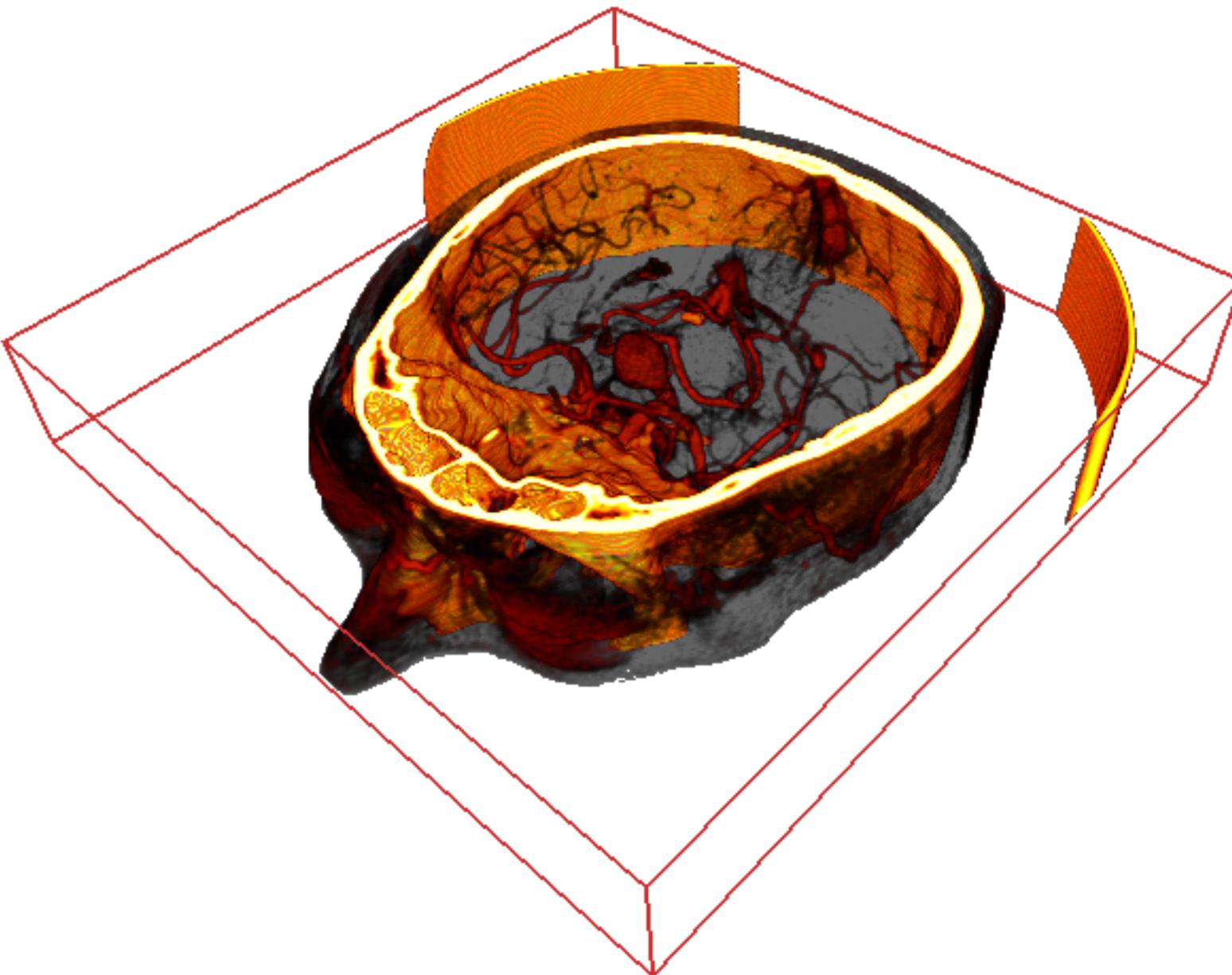
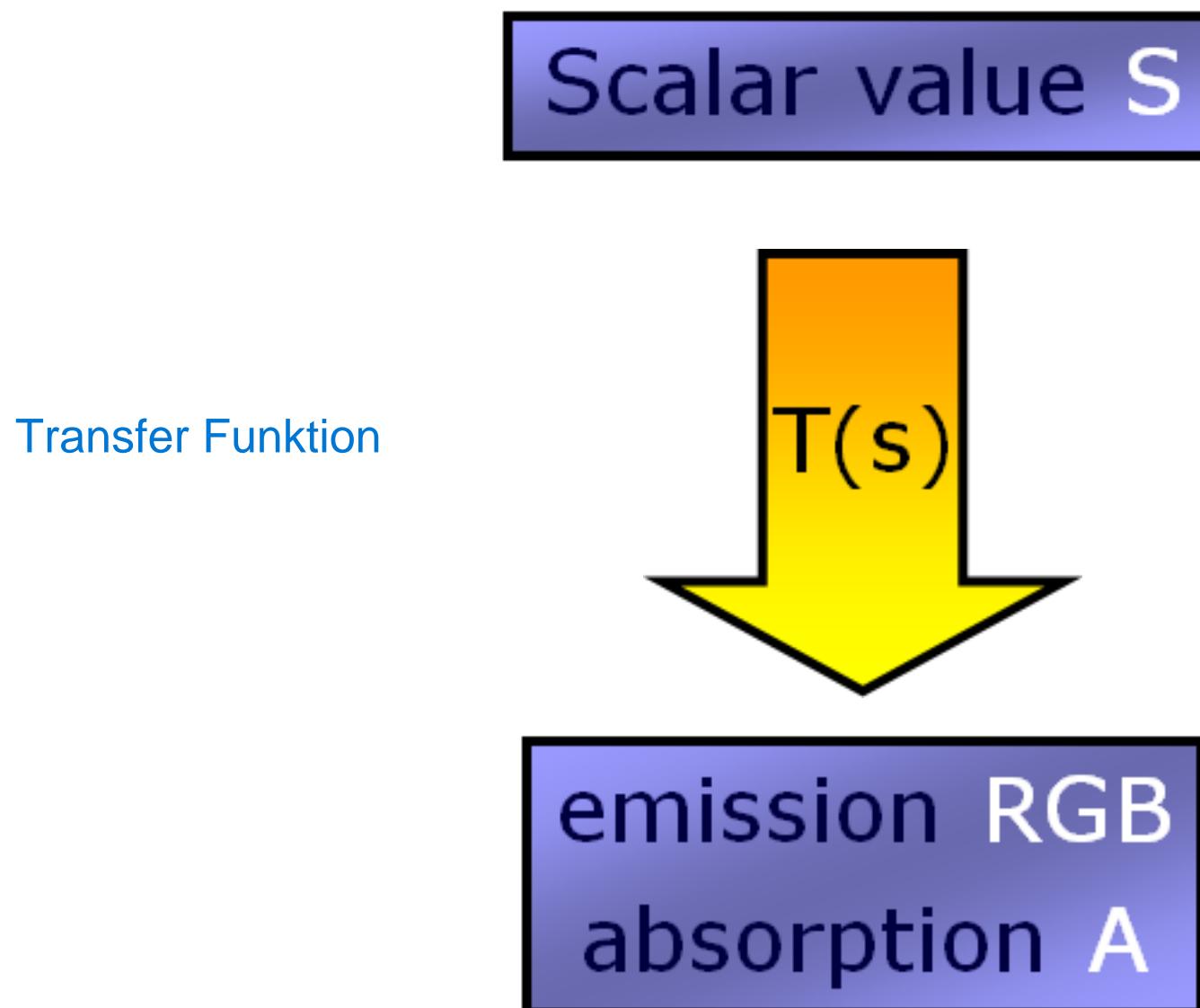
- Accumulate all values along a ray
 - Emission-absorption model (integration of volume rendering equation)



6.5.4 Classification

Classification

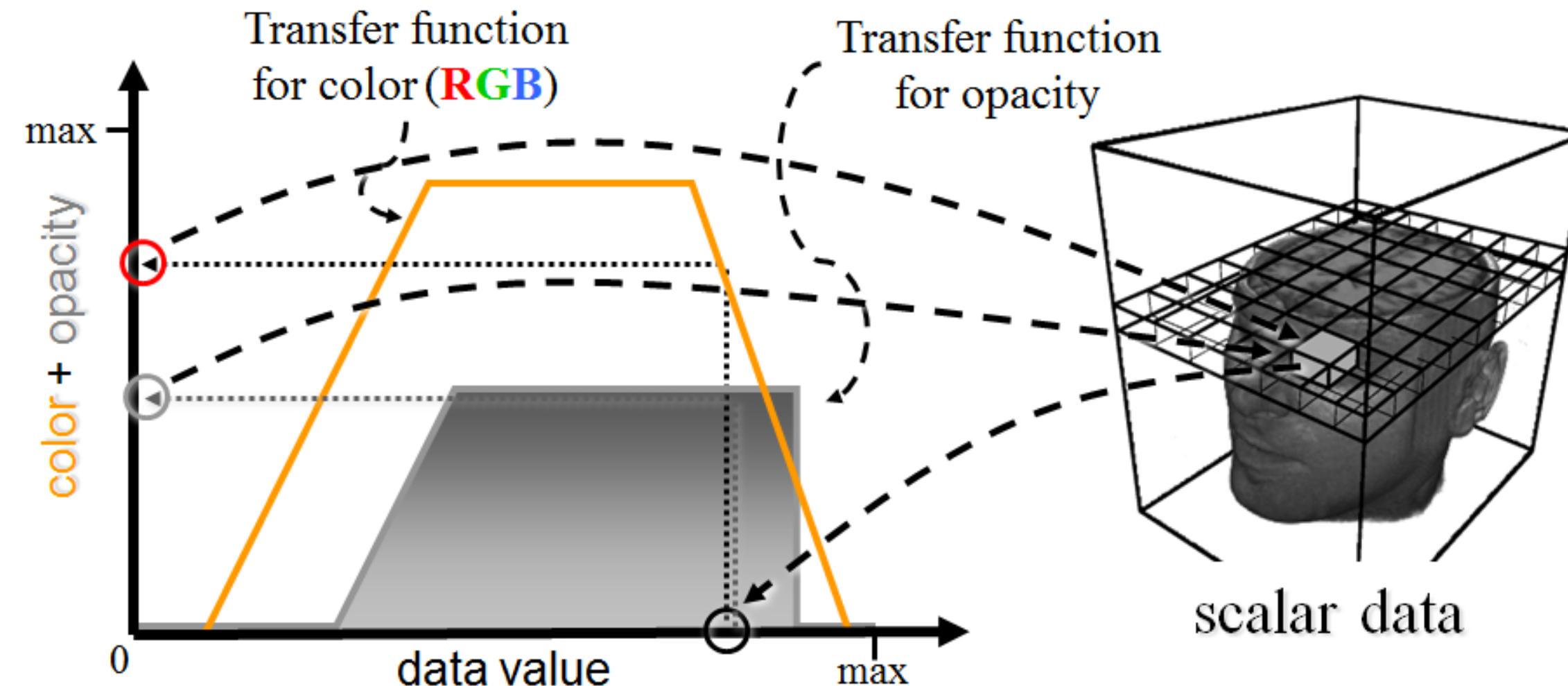
Wie werden Werte gesetzt dass es richtig Visualisiert wird



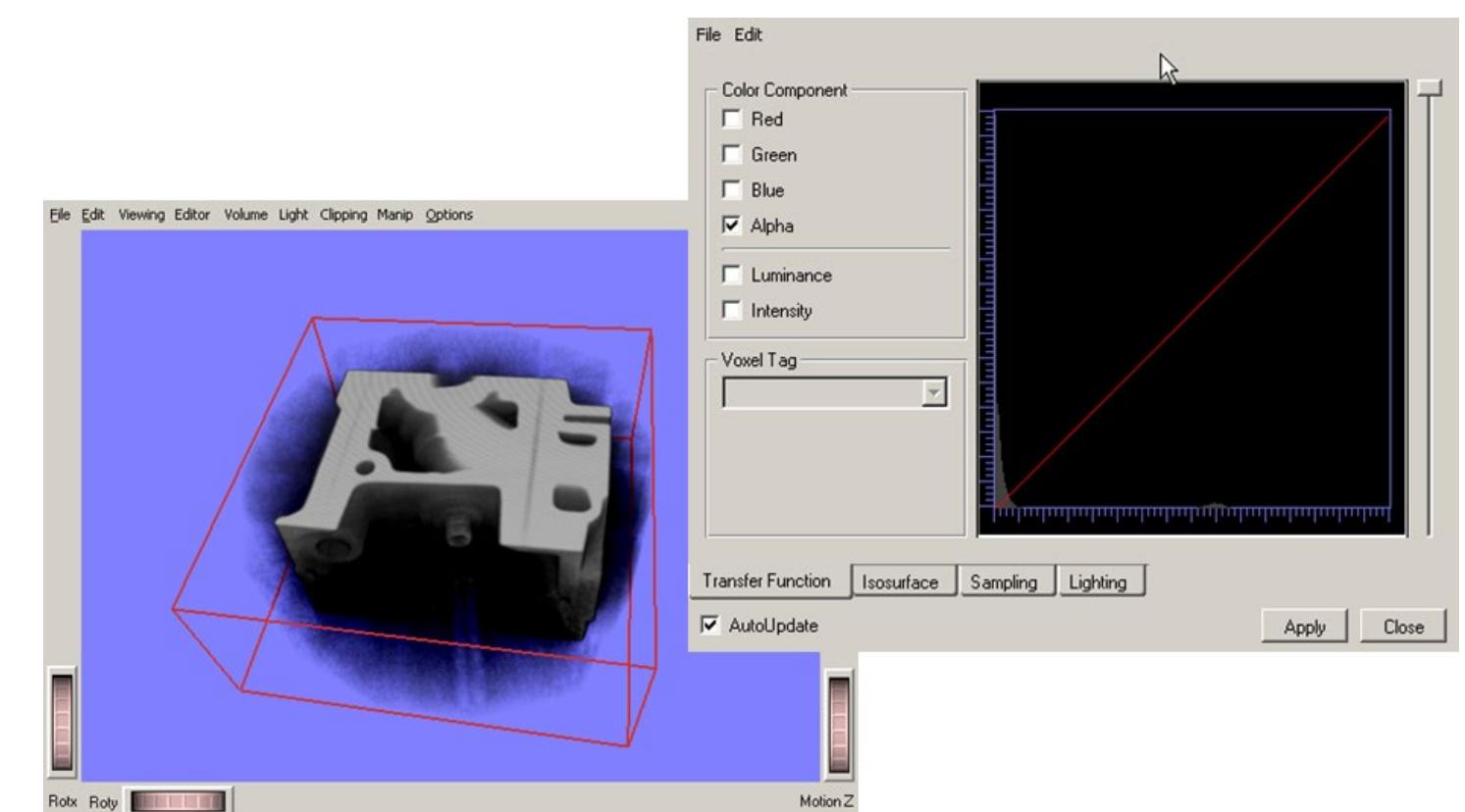
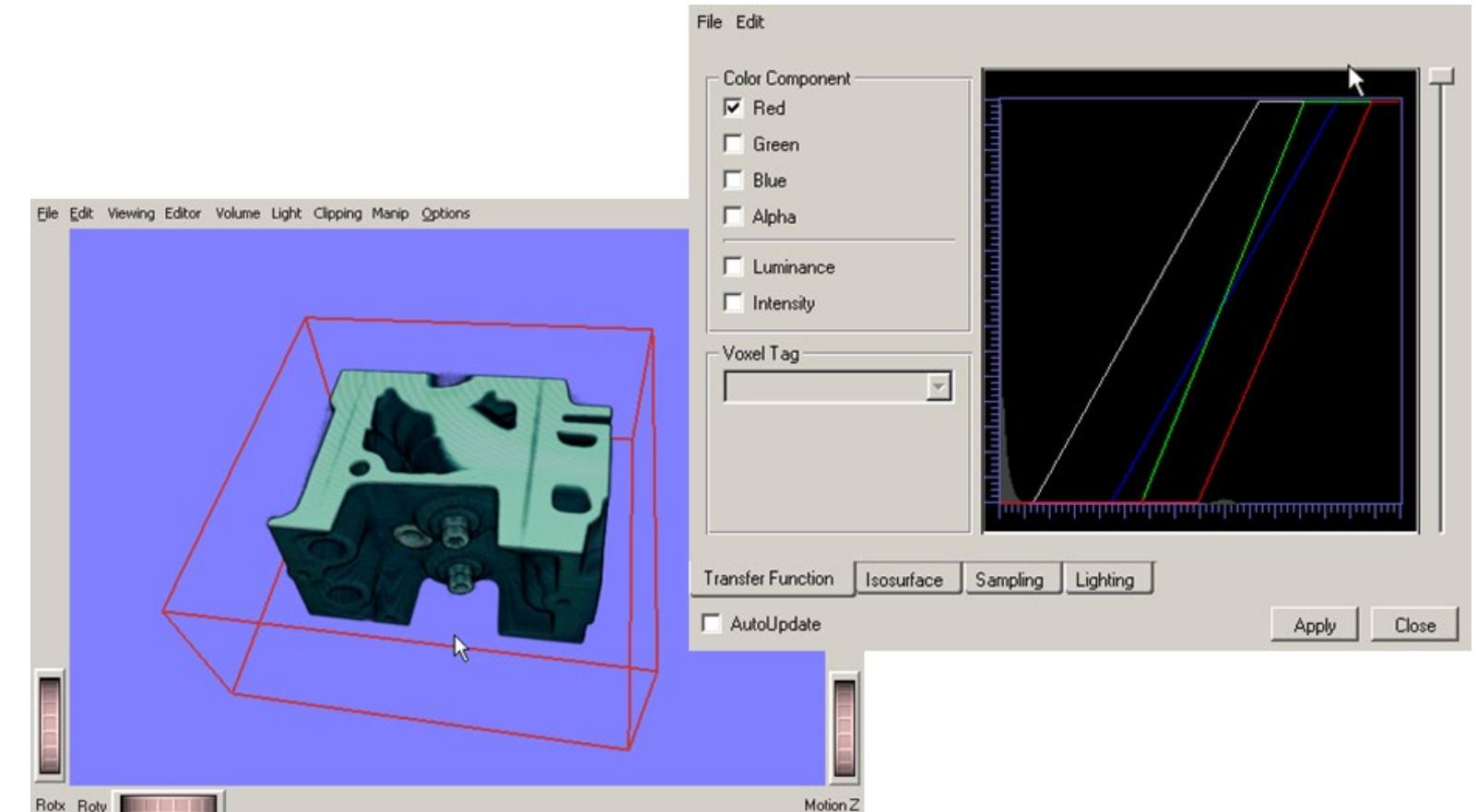
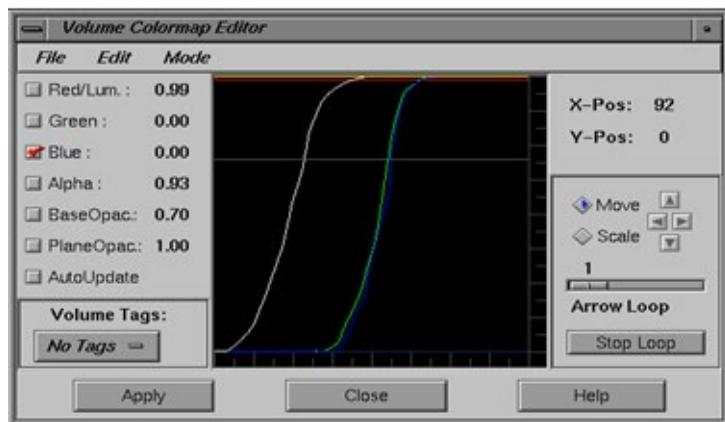
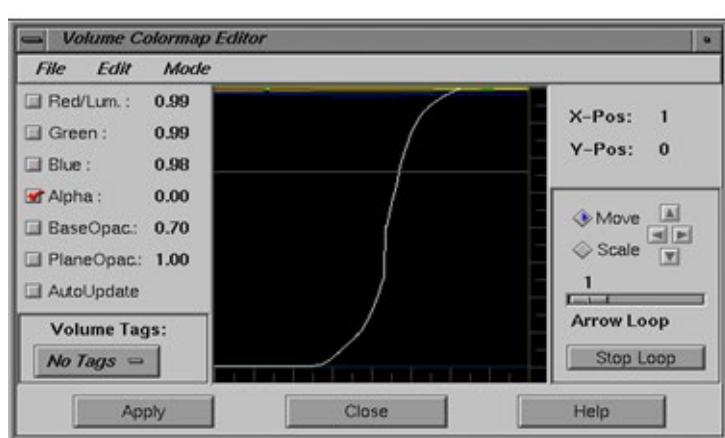
Classification

- Transfer functions
 - Map data value to color and opacity

look up Wert von Farbe/Grauwert für Scalar data



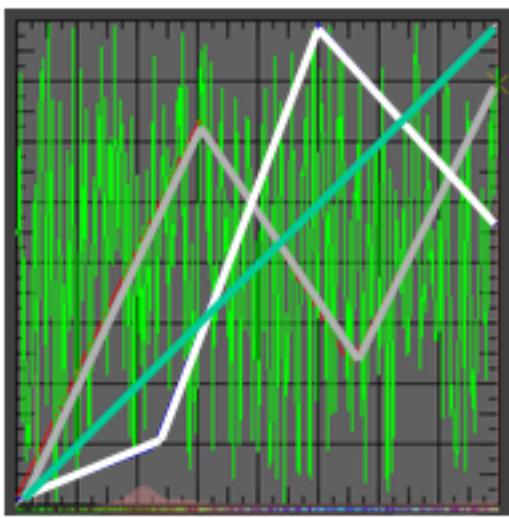
Classification Examples



Classification

Pre- vs. post-classification

transfer functions



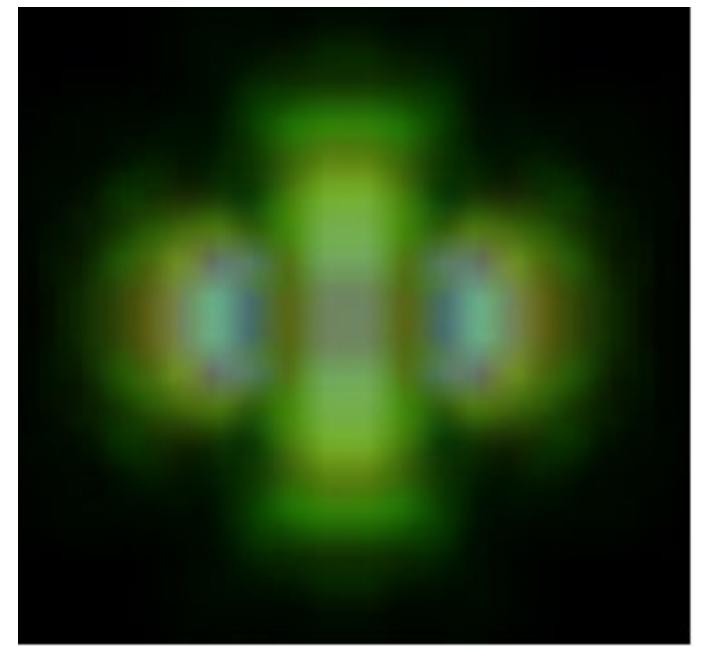
voxels

classification



interpolation

pre-classification

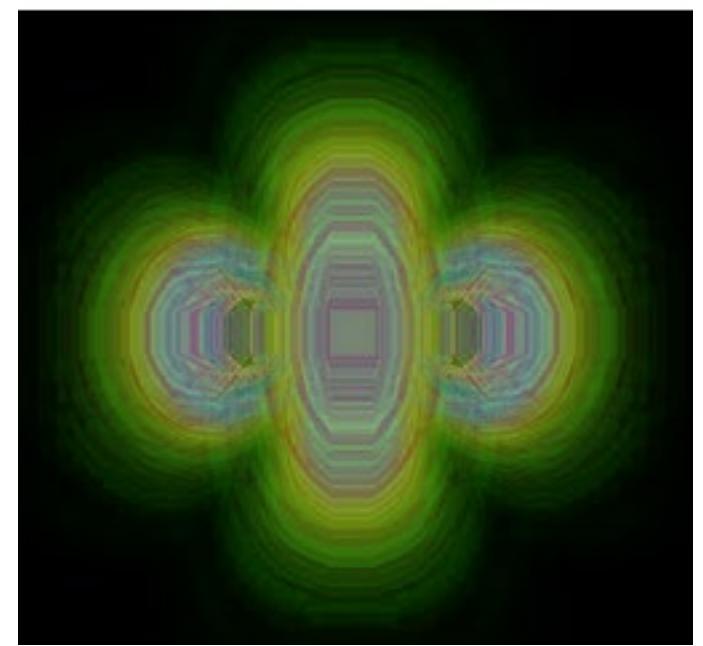


post-classification

classification



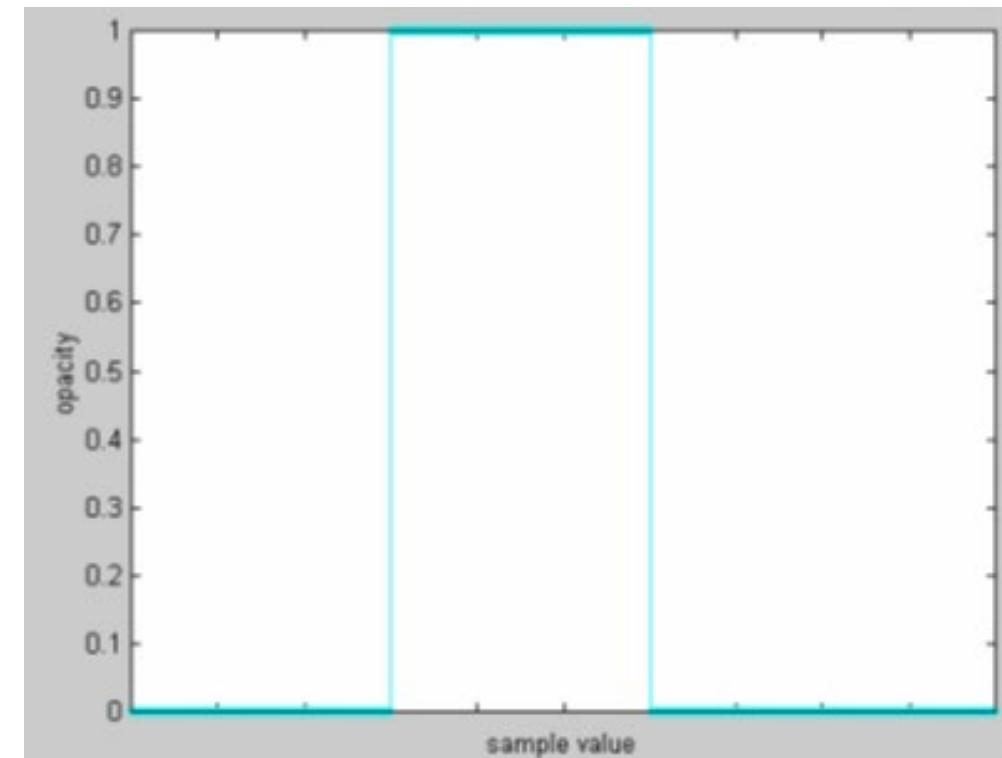
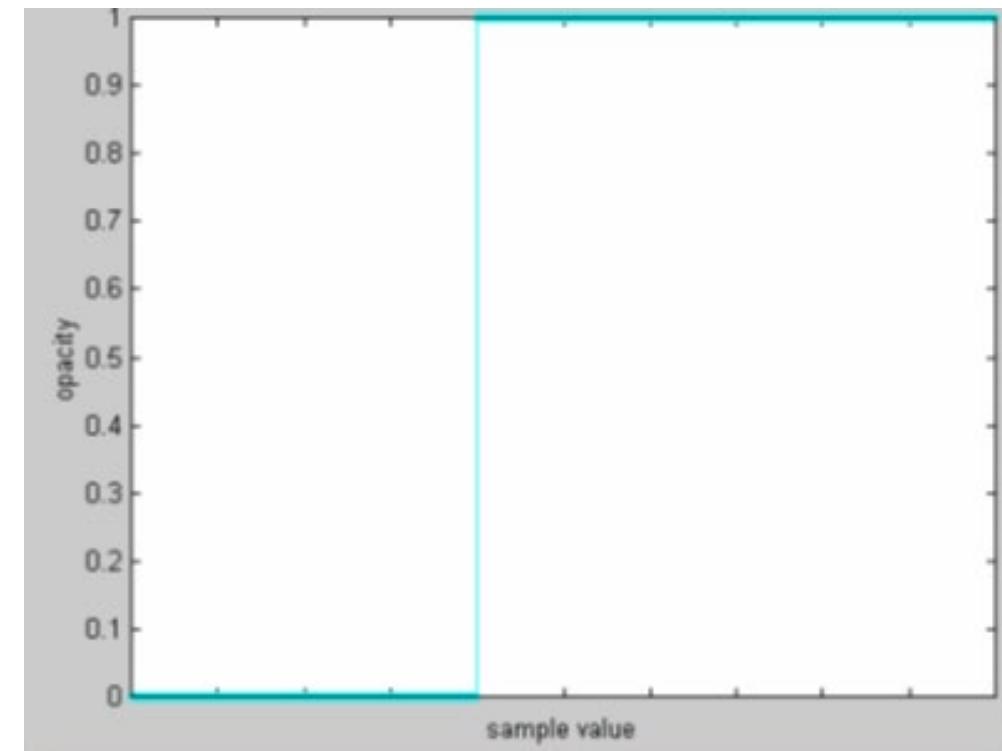
interpolation



Classification

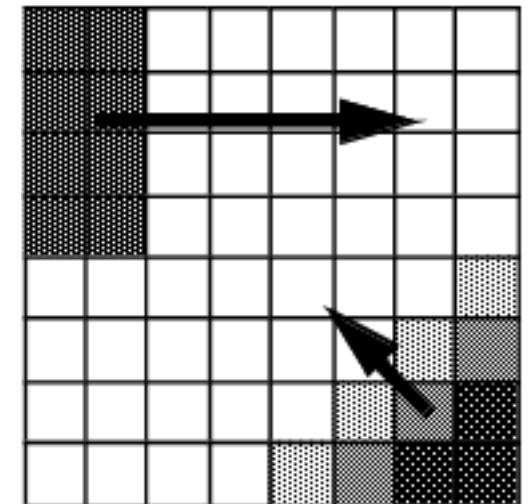
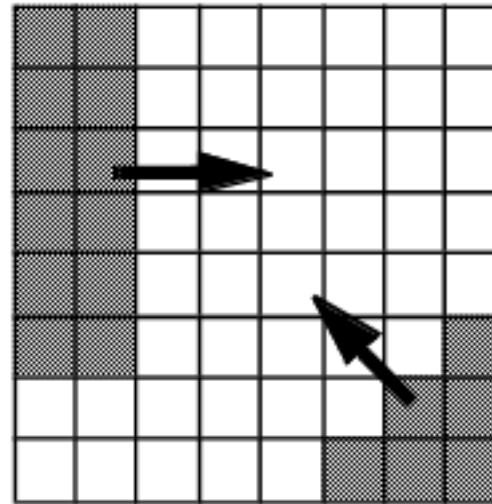
Isovalue contour surfaces

- Render opaquely all voxels with values > threshold
 - Unable to display multiple concentric surfaces
- Use a window instead of a threshold
 - Too narrow → holes
 - Too wide → restricted display of multiple surfaces
 - Generates artifacts that are not part of the original data



Classification

- Considerations
 - Usually not only interested in a particular iso-surface but in regions of "change"
 - Feature extraction
 - High value of opacity in regions of change
 - Homogenous regions less interesting - transparent
 - Surface "strength" depends on gradient
 - Gradient of the scalar field is taken into account



Classification

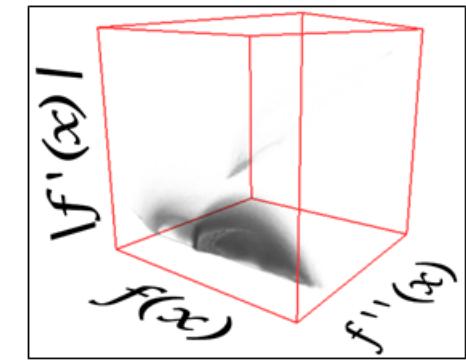
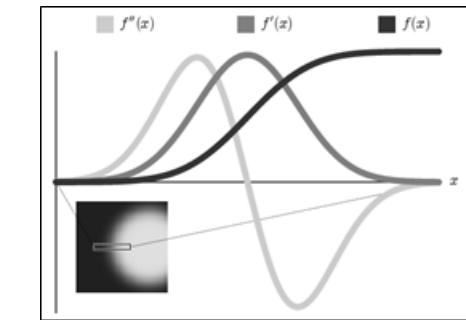
- In order to emphasize iso-surfaces
 - Opacity of $a_v = \max$ to voxel with isovalue f_v
 - Opacity to zero to remaining voxels
 - To avoid aliasing artifacts
 - Assign opacities close to a_v to voxels with values close to f_v
- Most pleasing if transition region stays constant
 - Opacity fall off at a rate inversely proportional to magnitude of local gradient
 - e.g.

$$\alpha(x_i) = \alpha_v \begin{cases} 1 & \text{if } f(x_i) = f_v \text{ and } |\nabla f(x_i)| = 0 \\ 1 - \frac{1}{r} \left| \frac{f_v - f(x_i)}{|\nabla f(x_i)|} \right| & \text{if } f_v - f(x_i) \leq r |\nabla f(x_i)| \\ 0 & \text{otherwise} \end{cases}$$

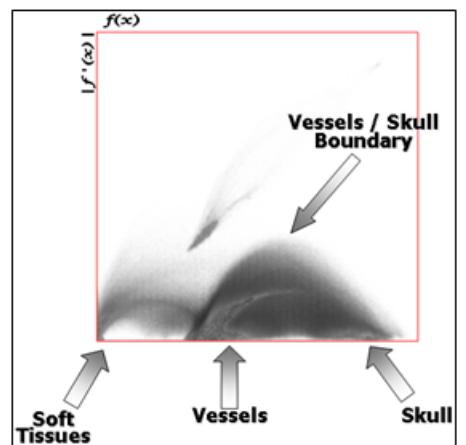
Classification

Multidimensional transfer functions

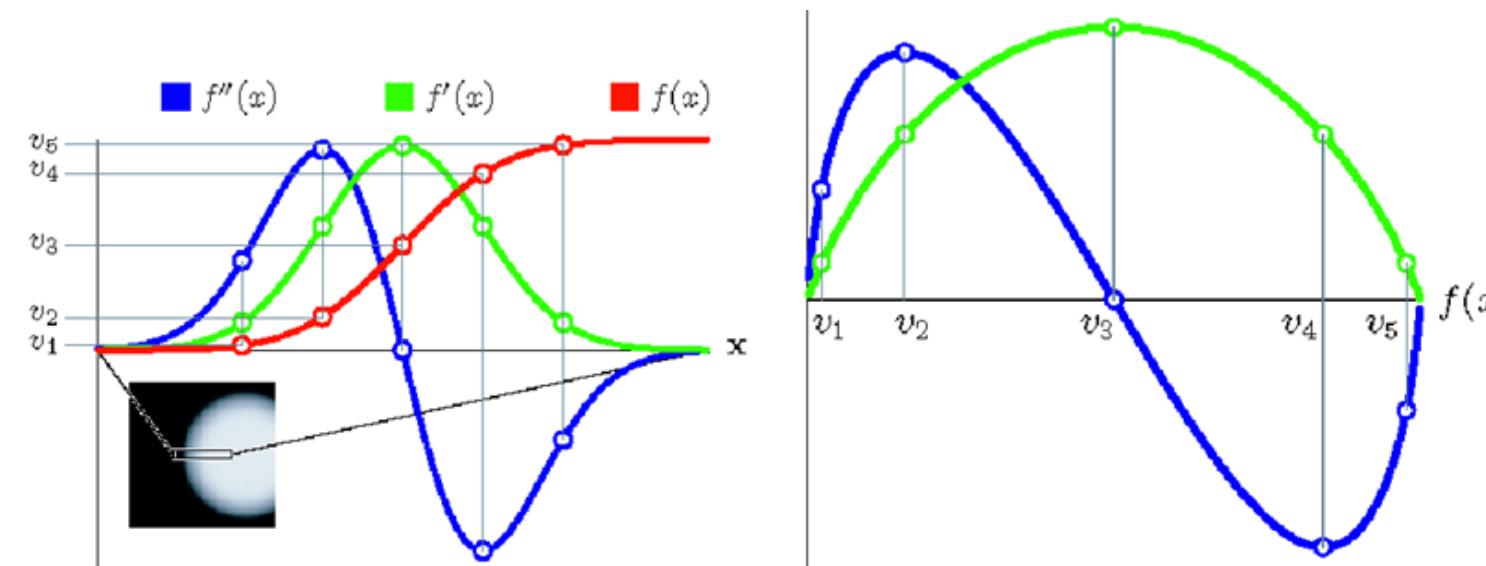
- Problem: How to identify boundary regions/surfaces
- Approach: 2D/3D transfer functions, depending on
 - Scalar value, gradient magnitude
 - Second derivative along the gradient direction



2D representation by integration in direction of $f''(x)$



Representation in a 3D histogram

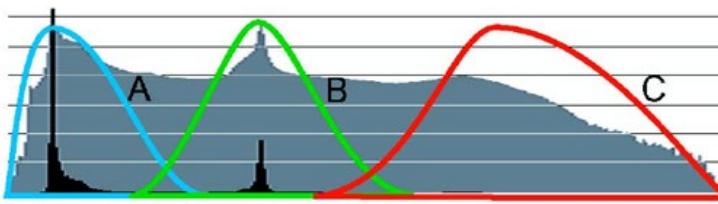


Classification

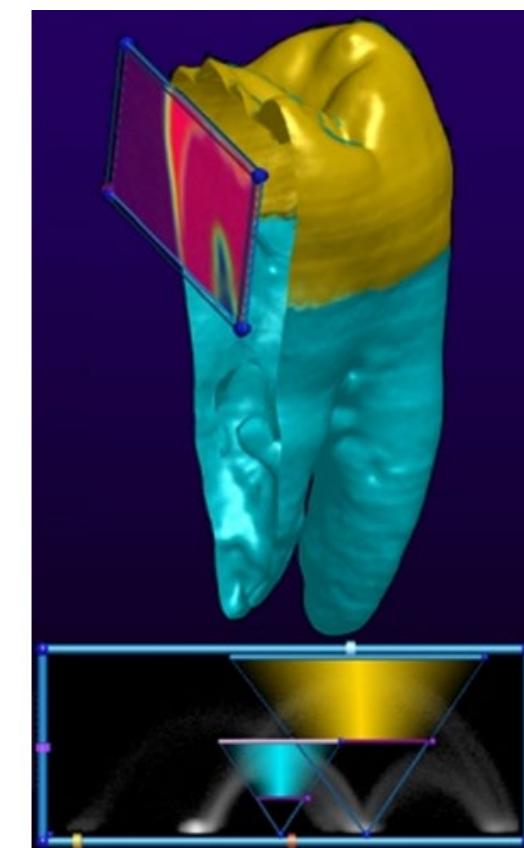
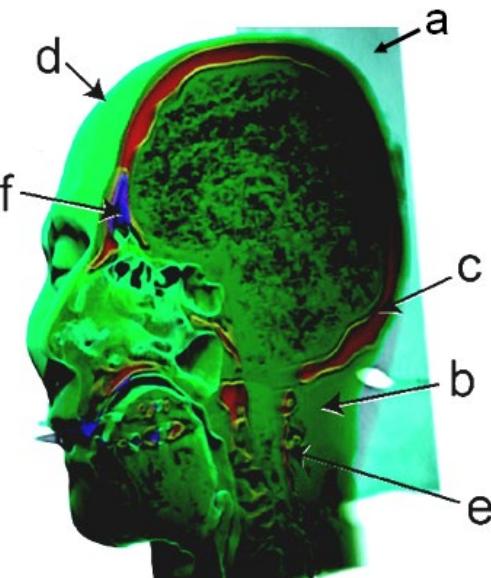
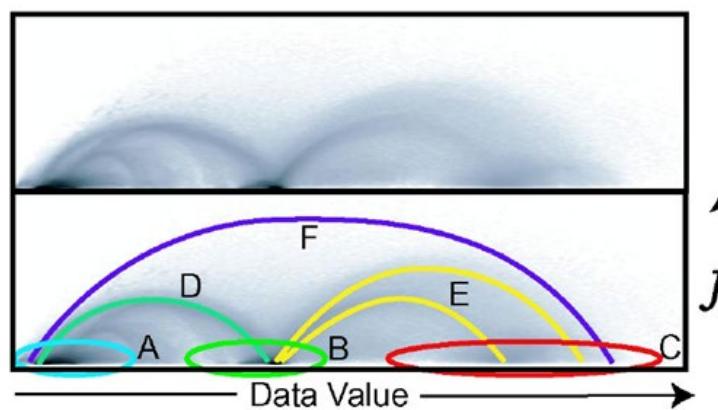
Multidimensional transfer functions

- Yields high quality transfer functions at the expense of restricted usability

1 transfer function with 1D histogram: basic materials in colored regions (A,B,C)



2D transfer function with 2D joint histogram:
materials (A,B,C), material boundaries (D,E,F)



Segmentation

Segmentation

zu meinem Datensatz noch Datensätze die Angeben wo andere Informationen sind (Color, Structure, ...)

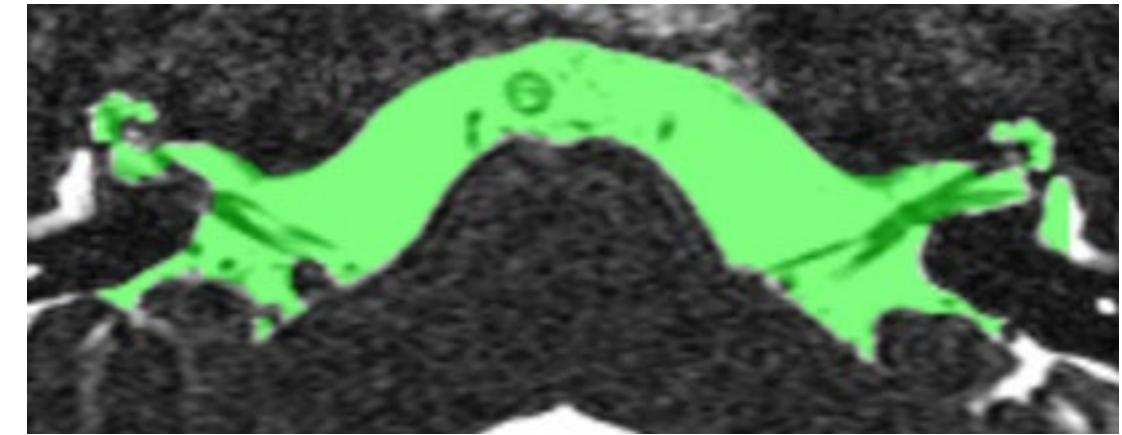
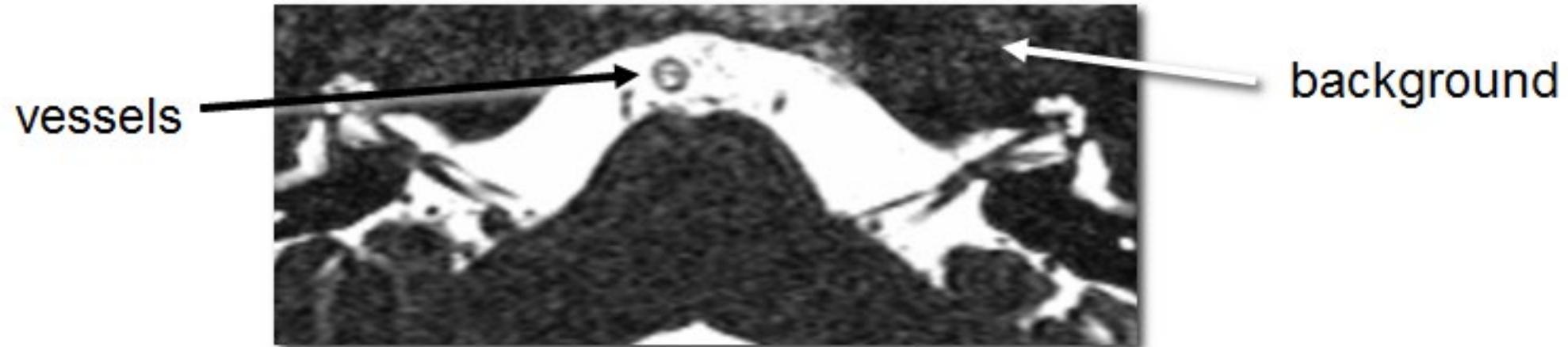
Explicit separation of structures

- Application areas
 - If classification with a transfer function fails
 - E.g. different structures, identical data values
 - In case of indirect volume rendering
 - E.g. if polygonal models are reconstructed out of volume data
- Label voxels indicating a type
 - Preprocessing
 - Semi-automatic process
- Note the difference
 - Implicit segmentation \leftrightarrow classification with color and opacity
 - Explicit segmentation \leftrightarrow labeling of structures

Segmentation

Example

- Implicit separation of background ↔ vessels impossible
 - Both structures have identical data values



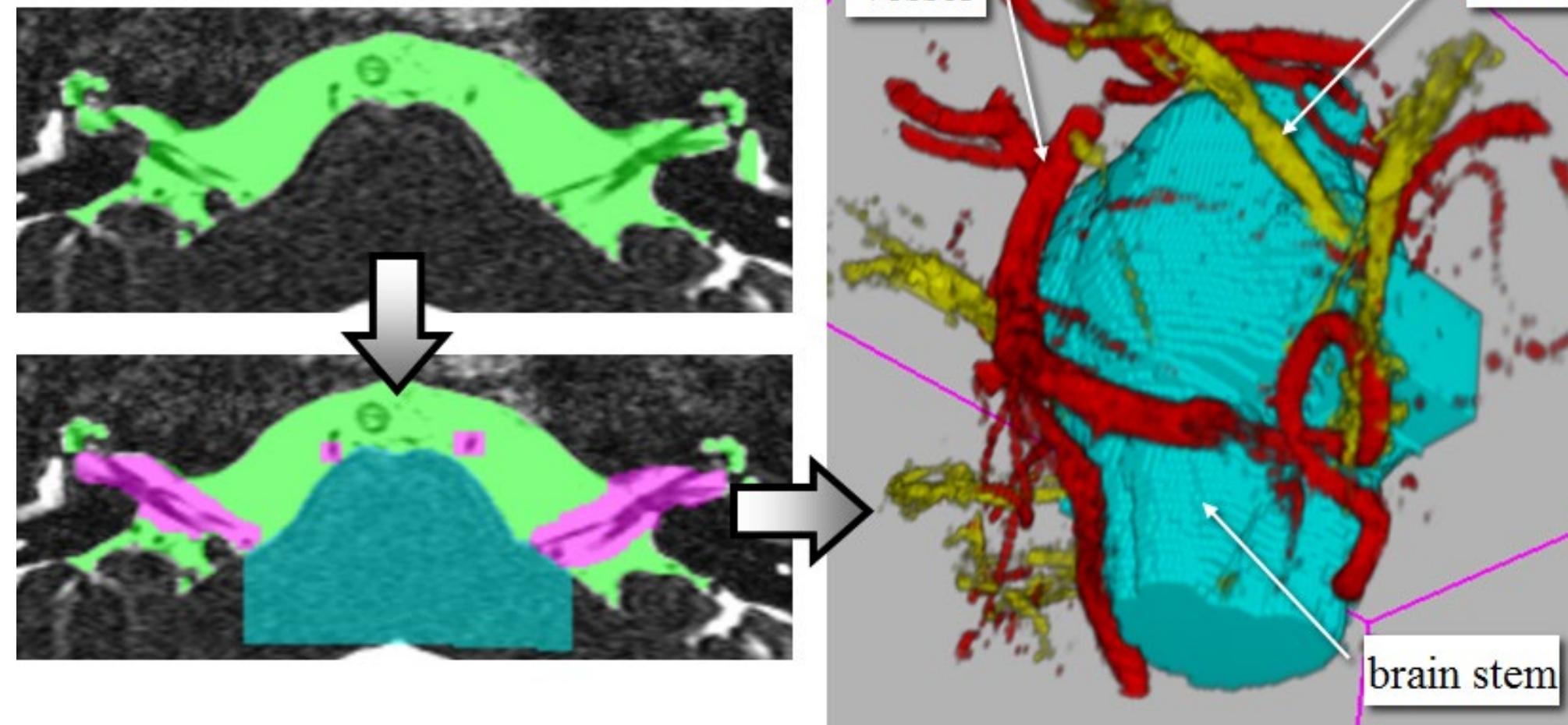
- Explicit segmentation of white area including vessels

Segmentation

Example (cont.)

- TF 1: background to transparent
- TF 2: explicitly segmented area

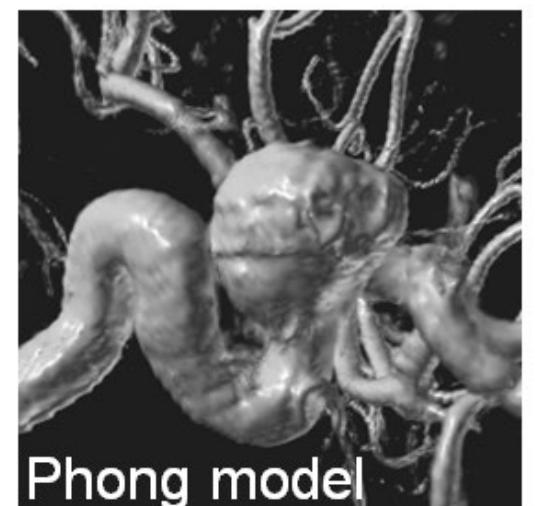
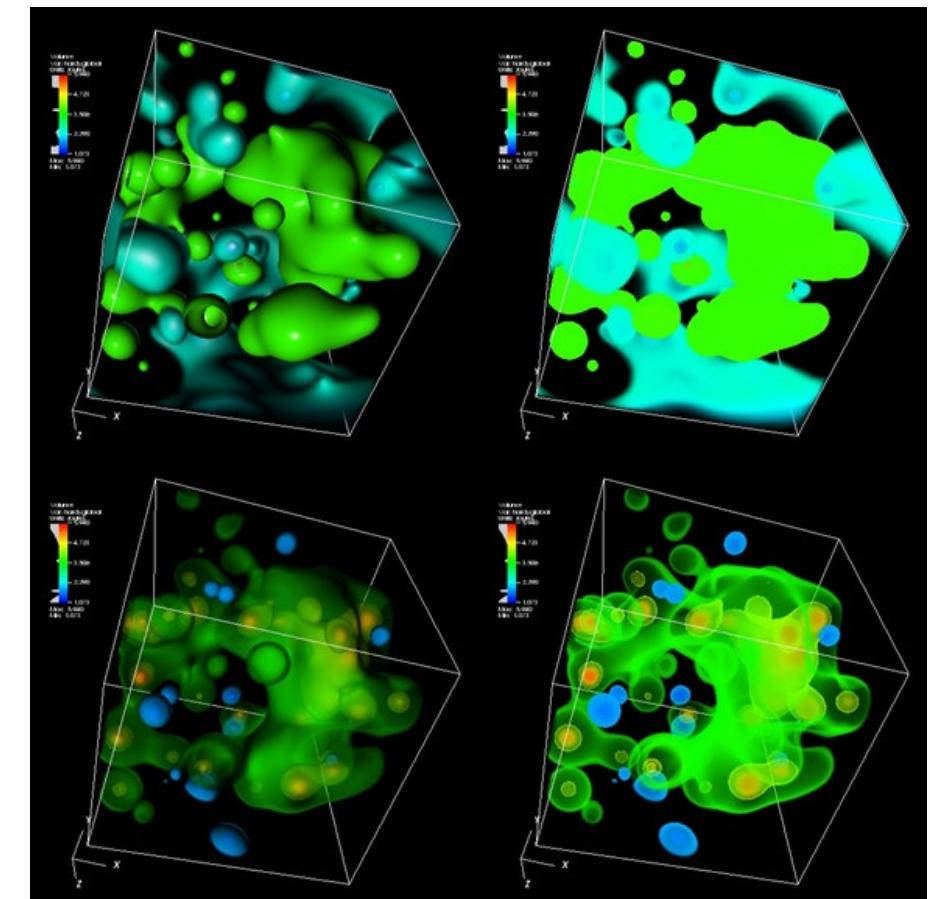
jedem der Label kann eine eigene Transferfunktion zuweisen



Volumetric Shading

Volumetric Shading

- Shading
 - Simulate reflection of light, simulate effect of color
 - Make use of human visual system's ability to efficiently deal with shaded objects
- What is the normal vector in a scalar field?
 - Use the gradient (perpendicular to iso-surface)
 - Numerical computation of the gradient
 - Central difference
 - Not isotropic - length is 1 to $\sqrt{3}$
 - Needs normalization
 - Intermediate difference (forward/backward difference)
 - Very cheap
 - Noisy data means less good gradients
 - Also not isotropic



$$K_a = 0.1$$

$$K_d = 0.5$$

$$K_s = 0.4$$

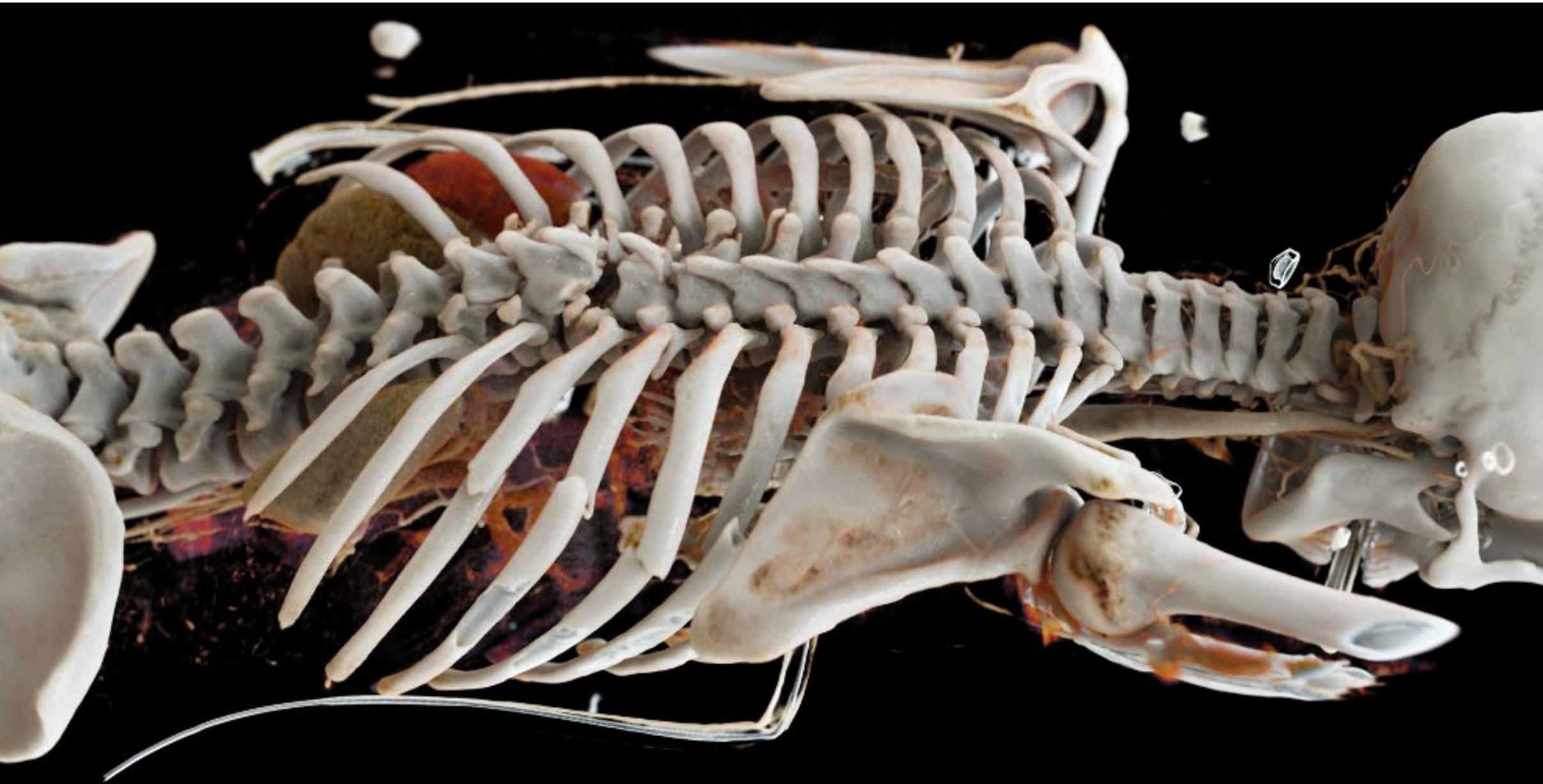
6.5.5 Outlook

Cinematic Rendering

Physically based rendering

- Photorealistic rendering based on physically correct lighting calculations
- Light transport equation
 - $L_o(p, \omega_o) = L_o(p, \omega_o) + \int_{S^2} f(p, \omega_o, \omega_i)L_i(p, \omega_i)|\cos \Theta_i|d\omega_i$
 - Can be extended to include volumetric scattering
 - Cannot be evaluated analytically (except for the most simplest scenes)
 - Monte-Carlo Ray-Tracing
 - $E[F_N] = E \left[\frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)} \right] = \int_a^b f(x)dx$
 - Requires many samples to yield a good estimate
 - Computers have become reasonably fast to do this

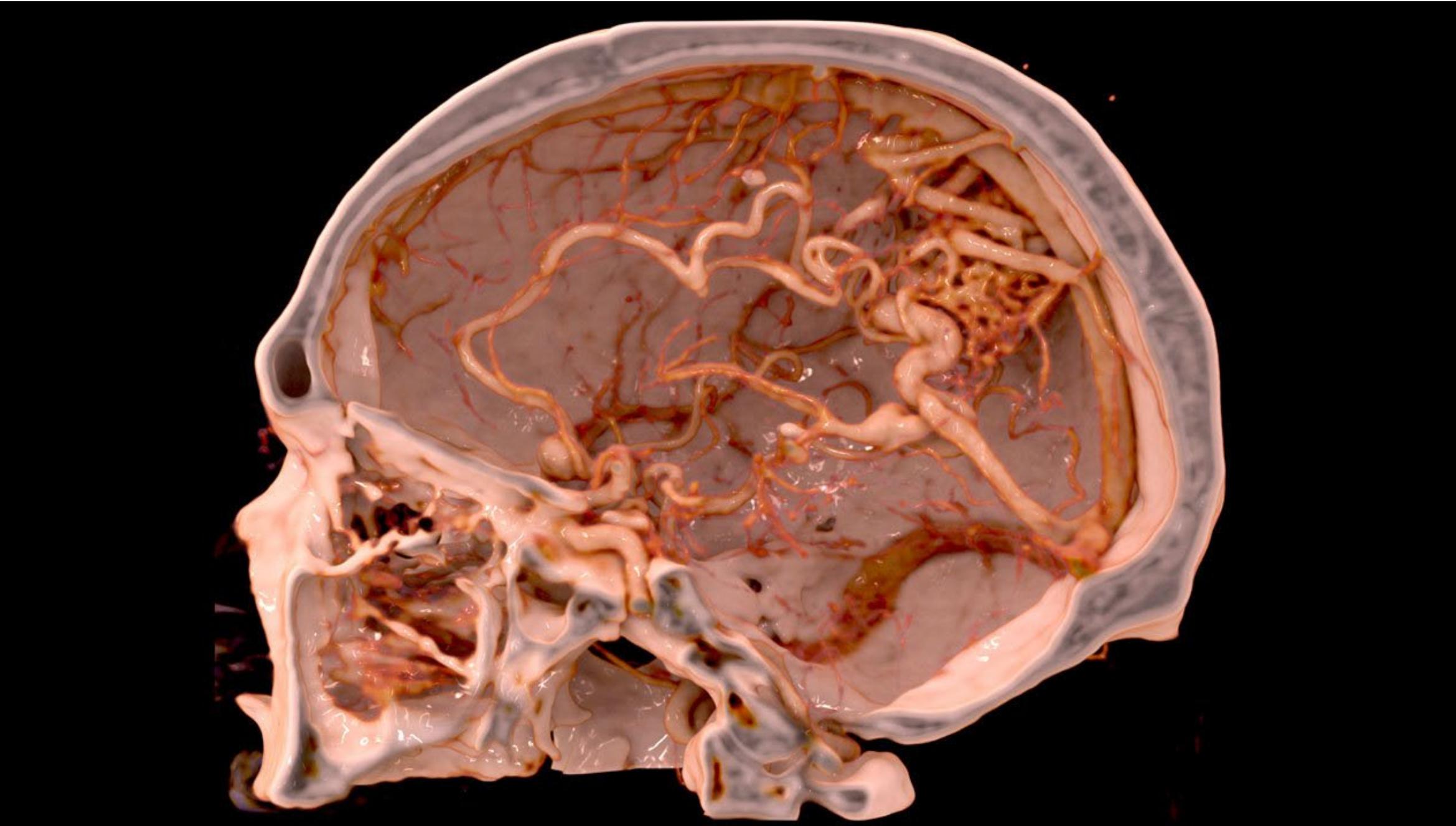
Cinematic Rendering



Source: <https://www.aec.at/postcity/en/cinematic-rendering/>

Prof. Dr. Matthias Teßmann

Cinematic Rendering



Source: <https://www.healthcare.siemens.de/magazine/msc-cinematic-rendering.html>

Prof. Dr. Matthias Teßmann

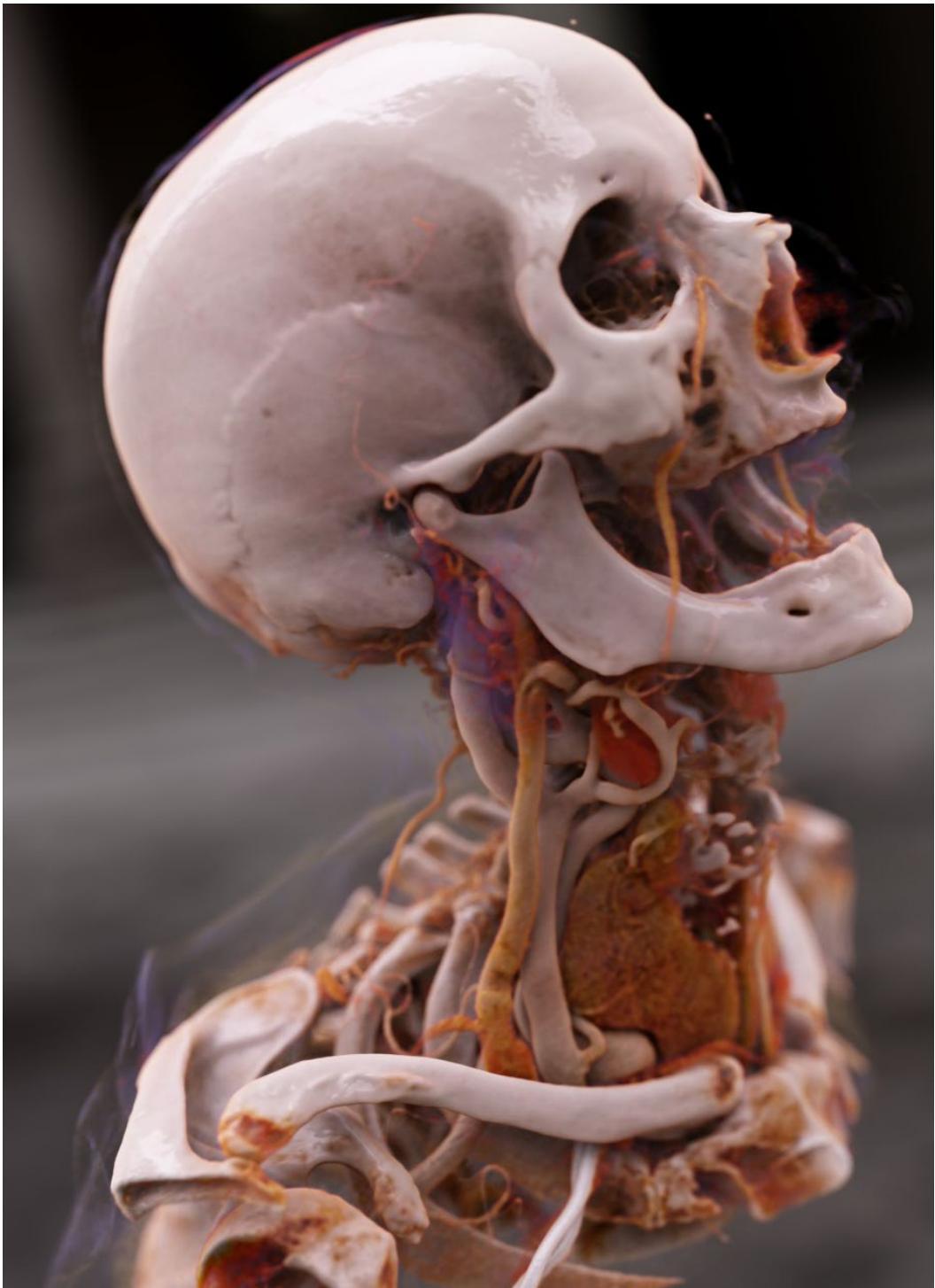
Cinematic Rendering



Source: <https://www.healthcare.siemens.de/magazine/mso-cinematic-rendering.html>

Prof. Dr. Matthias Teßmann

Cinematic Rendering



Source: <https://www.siemens.com/innovation/en/home/pictures-of-the-future/health-and-well-being/medical-imaging-cinematic-vrt.html>

Prof. Dr. Matthias Teßmann