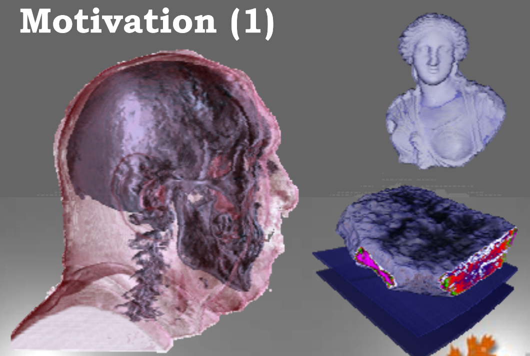
High-Quality Volume Graphics on Consumer PC Hardware

Joe Kniss — Gordon Kindlmann — Markus Hadwiger — Christof Rezk-Salama — Rüdiger Westermann

Volume Graphics Introduction

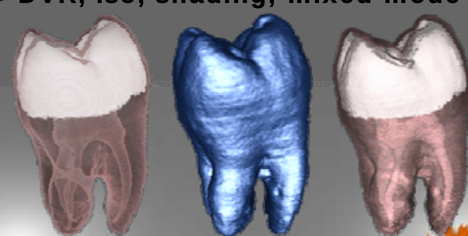
Motivation (1)



Motivation (2)

Scientific visualization

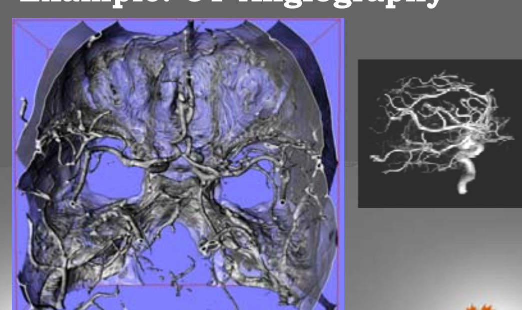
- DVR, iso, shading, mixed mode



Applications

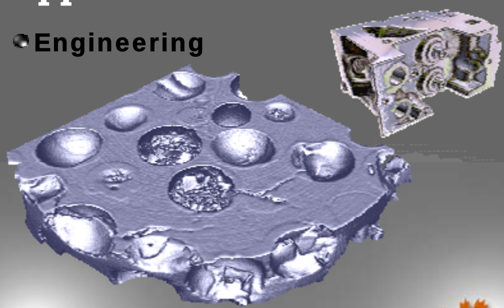
Medicine

Example: CT Angiography



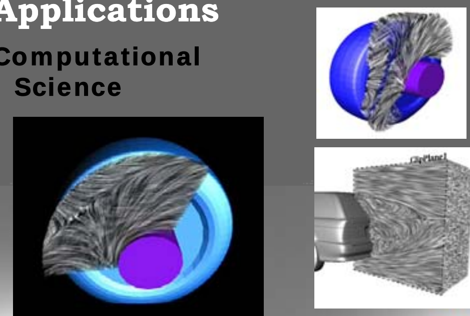
Applications

- Engineering



Applications

Computational Science

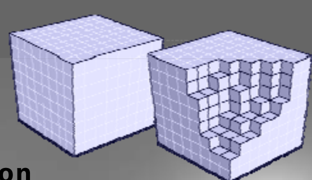


Volume Data

Continuous scalar field in 3D

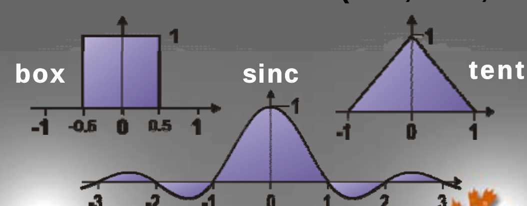
$$s = f(x, y, z); \quad x, y, z \in \mathbb{R}$$

- Discrete volume: voxels
- Sampling
- Reconstruction



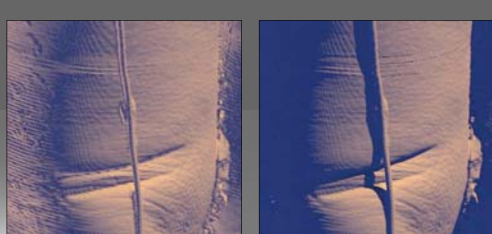
Sampling & Reconstruction

- Rectilinear 3D grid; scalar values
- Convolution of samples with reconstruction filter (box, tent, ...)



Surface extraction

- Extract Geometry, then render

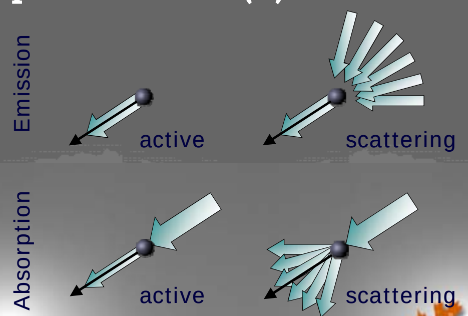


Without shadows With shadows

Direct Volume Rendering

- Render volume **without extracting any surfaces** (DVR)
- Map scalar values to **optical properties** (color, opacity)
- Need optical model
- Solve **volume rendering integral** for viewing rays into the volume

Optical Models (1)



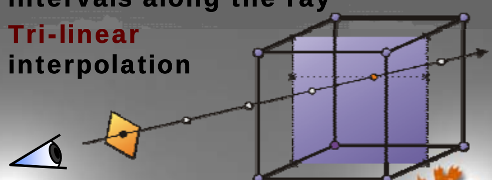
Optical Models (2)

Light interaction with density volume of particles

- **Absorption only**
- **Emission only**
- **Absorption + emission**
- **Scattering + shading / shadowing**
- **Multiple scattering**

Ray Casting

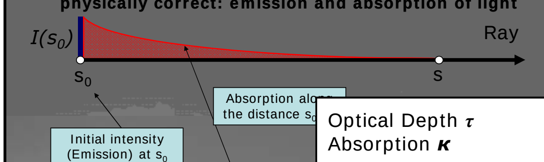
- **Numerical approximation of volume rendering integral**
- **Resample volume at equi-spaced intervals along the ray**
- **Tri-linear interpolation**



Integration

What must be integrated?

physically correct: emission and absorption of light



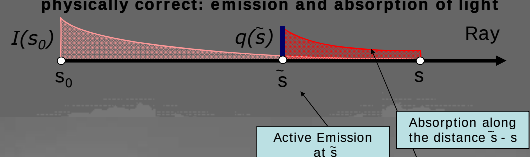
$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

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

Integration

What must be integrated?

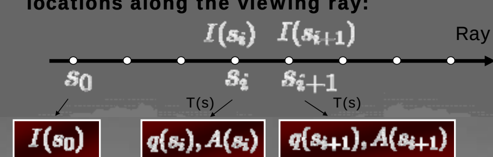
physically correct: emission and absorption of light



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

Discrete Solution

Resample the scalar field at discrete locations along the viewing ray:



Back-to-front Compositing with $\alpha = A(s_i)$

$$I(s_{i+1}) = \alpha \cdot q(s_{i+1}) + (1-\alpha) I(s_i)$$
$$= q(s_{i+1}) \text{ OVER } I(s_i)$$

Alpha Blending

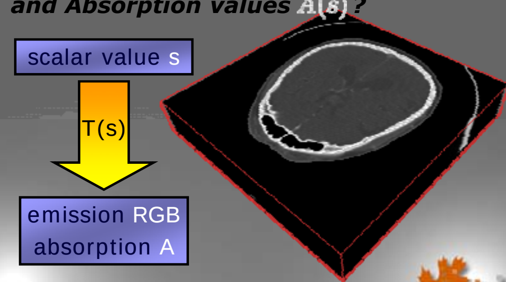
- **Numerical approximation** of volume rendering integral
- **Back-to-front compositing**

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

- **New composited color is color at current location blended with previous composited color**
- **Opacity-weighted colors!**

Classification

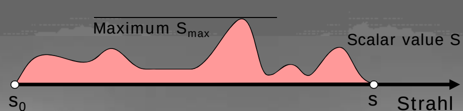
How do I obtain the emission values $q(s)$ and Absorption values $A(s)$?



Alternative Compositing

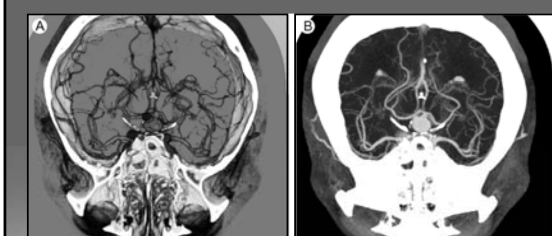
Maximum Intensity Projection (MIP)

- no **Emission** and **Absorption**
- pixel value is the **maximum** scalar value along the viewing ray



- **Advantage:** transfer function not required!
- **Drawback:** misleading depth information!

Maximum Intensity Projection



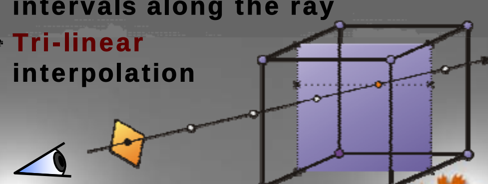
Emission/Absorption Maximum Intensity Proj.

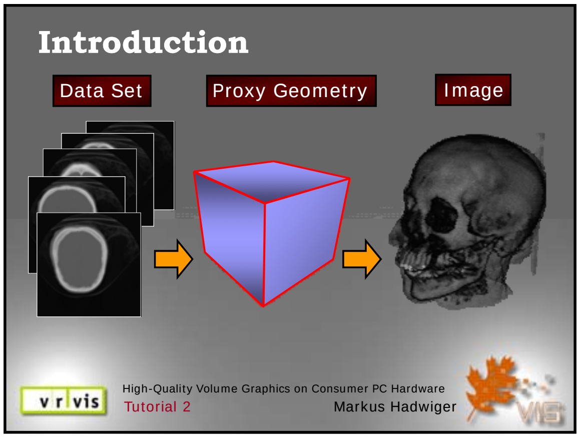
Volume Graphics on GPUs

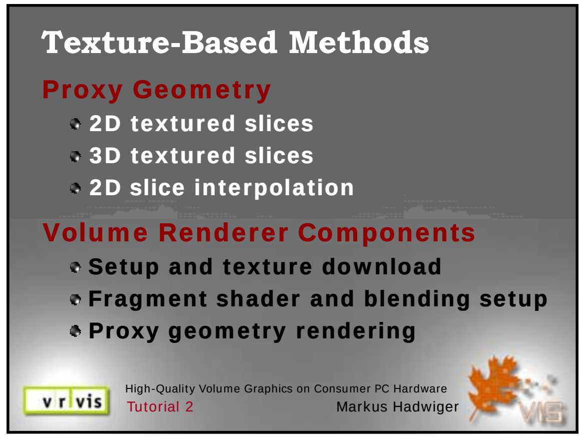
Texture-Based Methods

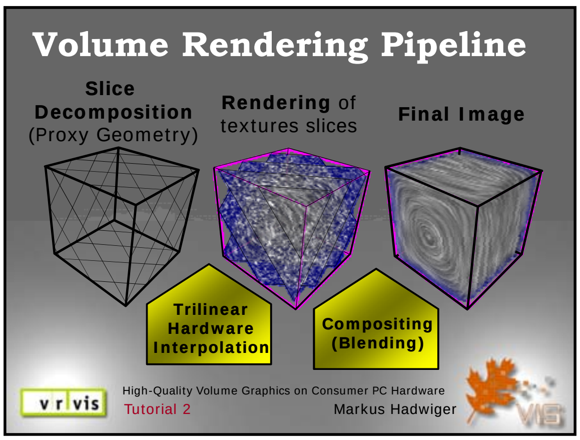
Ray Casting

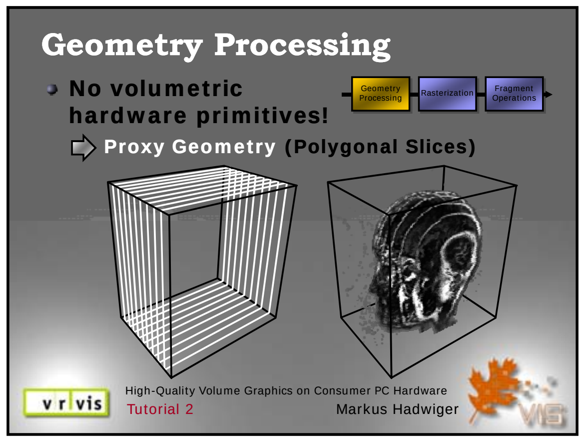
- **Numerical approximation** of volume rendering integral
- **Resample volume at equi-spaced intervals along the ray**
- **Tri-linear interpolation**

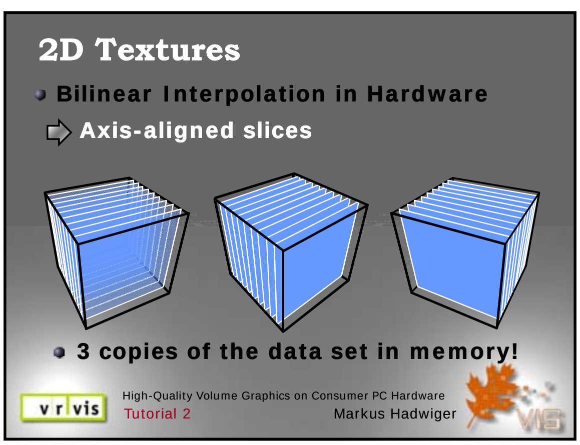


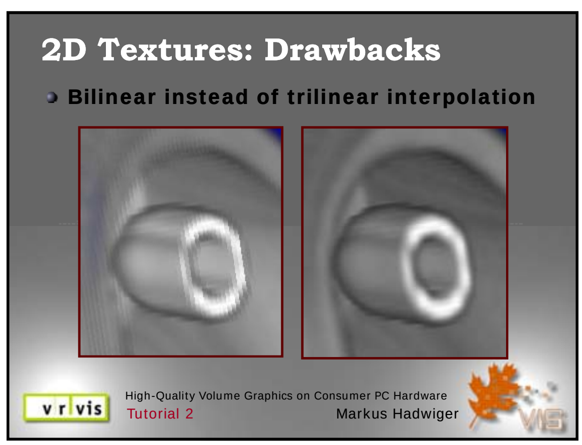

Introduction
Data Set
Proxy Geometry
Image
v r vis
High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2
Markus Hadwiger


Texture-Based Methods
Proxy Geometry
2D textured slices
3D textured slices
2D slice interpolation
Volume Renderer Components
Setup and texture download
Fragment shader and blending setup
Proxy geometry rendering
v r vis
High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2
Markus Hadwiger


Volume Rendering Pipeline
Slice Decomposition (Proxy Geometry)
Rendering of textures slices
Final Image
Trilinear Hardware Interpolation
Compositing (Blending)
v r vis
High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2
Markus Hadwiger

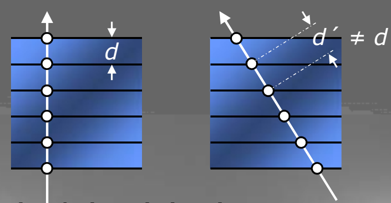

Geometry Processing
No volumetric hardware primitives!
Proxy Geometry (Polygonal Slices)
Geometry Processing
Rasterization
Fragment Operations
v r vis
High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2
Markus Hadwiger


2D Textures
Bilinear Interpolation in Hardware
Axis-aligned slices
3 copies of the data set in memory!
v r vis
High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2
Markus Hadwiger


2D Textures: Drawbacks
Bilinear instead of trilinear interpolation
v r vis
High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2
Markus Hadwiger

2D Textures: Drawbacks

- **Inconsistent sampling rate**



- **Emission / Absorbtion incorrect**
- **Supersampling not possible!**

2D Textures: Drawbacks

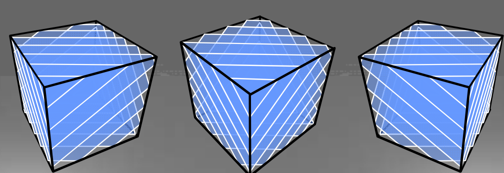
Select slice stack for viewpoint

- **Locations of sampling points may change abruptly!**

- **Stack switches are noticeable**

Better: 3D Texturen

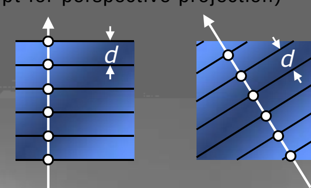
- **Trilinear Interpolation in Hardware**
  - ⇨ **Viewport aligned slices**



- **Volume is one texture block in memory**

Advantages:

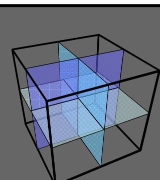
- **consistent sampling rate**
  (except for perspective projection)



- **Supersampling by increasing the number of slices**

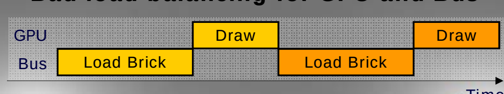
Bricking

- **Dataset is too large to fit into local video memory**
- ⇨ **Subdivide the data into Smaller chunks (bricks)**



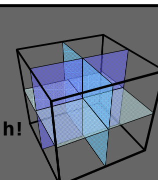
Bottleneck: Bus-Bandwidth

- **Bad load balancing for GPU and Bus**



Bricking

Possible Solutions

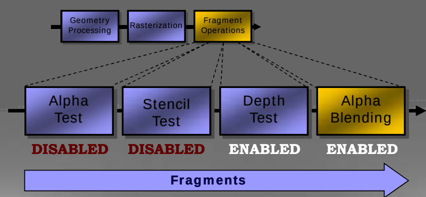


- **Keep the bricks small enough!**
  More than one brick must fit into local video memory!
  - **Texture transfer and rendering must be performed in parallel.**
  - **Higher CPU load for slice decomposition!**
  - **Effective load balancing is still difficult!**

Alternative:
Use 2D multi-textures instead of 3D textures!

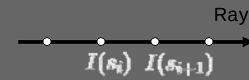
Compositing

- **Fragment Operations:**



Compositing

- **Ray Integration**



Back-to-front Compositing

$$I(s_{i+1}) = \alpha \cdot q(s_{i+1}) + (1-\alpha) I(s_i)$$
$$= q(s_{i+1}) \text{ OVER } I(s_i)$$

Alpha Blending

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```


Compositing

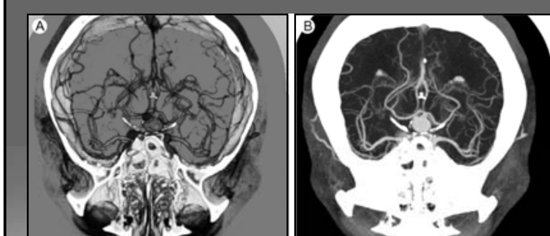
- **Maximum Intensity Projections**

OpenGL Extension: GL_EXT_blend_minmax

Alpha Blending: Maximum Operator

```
glEnable(GL_BLEND);
#if defined GL_EXT_blend_minmax
        glBlendEquationEXT(GL_MAX_EXT);
        glBlendFunc(GL_SRC_COLOR, GL_DST_COLOR);
#endif // GL_EXT_blend_minmax
```


Compositing



Emission/Absorption Maximum Intensity Proj.

Opacity Correction

- **Sampling distance depends on slice distance: adjust integration**
- **May depend on viewing direction (2D-textured slices)**

```
// determine cosine via dot-product; vectors must be normalized!
float correction_cosine = DotProduct3( view_vector, stack_vector );
// determine correction factor
float opacity_correction_factor = ( correction_cosine != 0.0f ) ?
    1.0f/correction_cosine :  1.0f;
```


Texture Definition

Volume textures

- **Palette indices (pre-class.)**
- **Intensity texture (post-class.)**

Transfer function tables

- **Texture palette**
- **Transfer function texture (1D, ...)**
- **Pre-integration texture (2D)**

Per-Volume Setup

Fragment shader configuration

- Transfer function application, shading, slice interpolation, ...

Blending mode configuration

- Final conversion fragment → pixel

Texture unit configuration (3D)

- When view-aligned slices are used
- All slices use the same texture

Fragment Shader Config (1)

NVIDIA GeForce 3 / 4

```
// configure texture shaders
glTexEnvi( GL_TEXTURE_SHADER_NV, GL_SHADER_OPERATION_NV, ... );
...
// enable texture shaders
glEnable( GL_TEXTURE_SHADER_NV );
// configure register combiners
glCombinerParameteriNV( GL_NUM_GENERAL_COMBINERS_NV, 1 );
glCombinerInputNV( GL_COMBINER0_NV, ... );
glCombinerOutputNV( GL_COMBINER0_NV, ... );
glFinalCombinerInputNV( ... );
...
// enable register combiners
glEnable( GL_REGISTER_COMBINERS_NV );
```


Fragment Shader Config (2)

ATI Radeon 8500

```
// configure fragment shader
GLuint shader_name = glGenFragmentShadersATI( 1 );
glBindFragmentShaderATI( shader_name );
glBeginFragmentShaderATI();
   glSampleMapATI( GL_REG_0_ATI, GL_TEXTURE0_ARB, GL_SWIZZLE_STR_ATI );
   glColorFragmentOp2ATI( GL_MUL_ATI, ... );
   ...
glEndFragmentShaderATI();
// enable fragment shader
glEnable( GL_FRAGMENT_SHADER_ATI );
```


Per-Slice Setup

Texture unit configuration (2D)

- When object-aligned slices are used
- Each slice uses different texture

- Slice interpolation needs two adjacent slice textures
- Fractional position between slices for slice interpolation

Generate Fragments

Render proxy geometry

- Submit slices in back-to-front order

2D textured slices

- Align slices with volume
- Specify texcoords for each slice

3D textured slices

- Align slices with viewport
- Use automatic texcoord generation