Westfälische
Wilhelms-Universität
Münster

Voreen – Volume Rendering Engine

Aaron Scherzinger, Dominik Drees

Version 5.0

living.knowledge
WWU Münster

01.08.2018

voreen
volume rendering engine

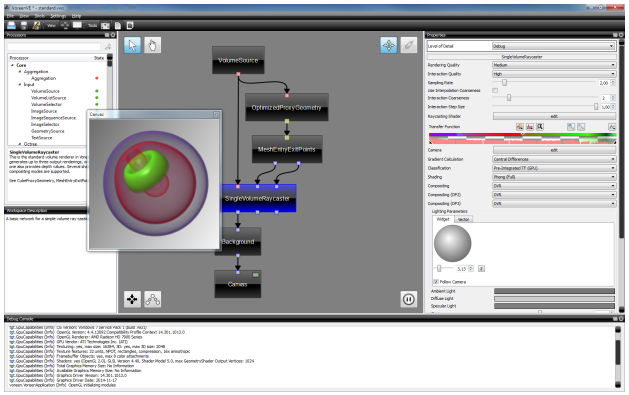
Outline

1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

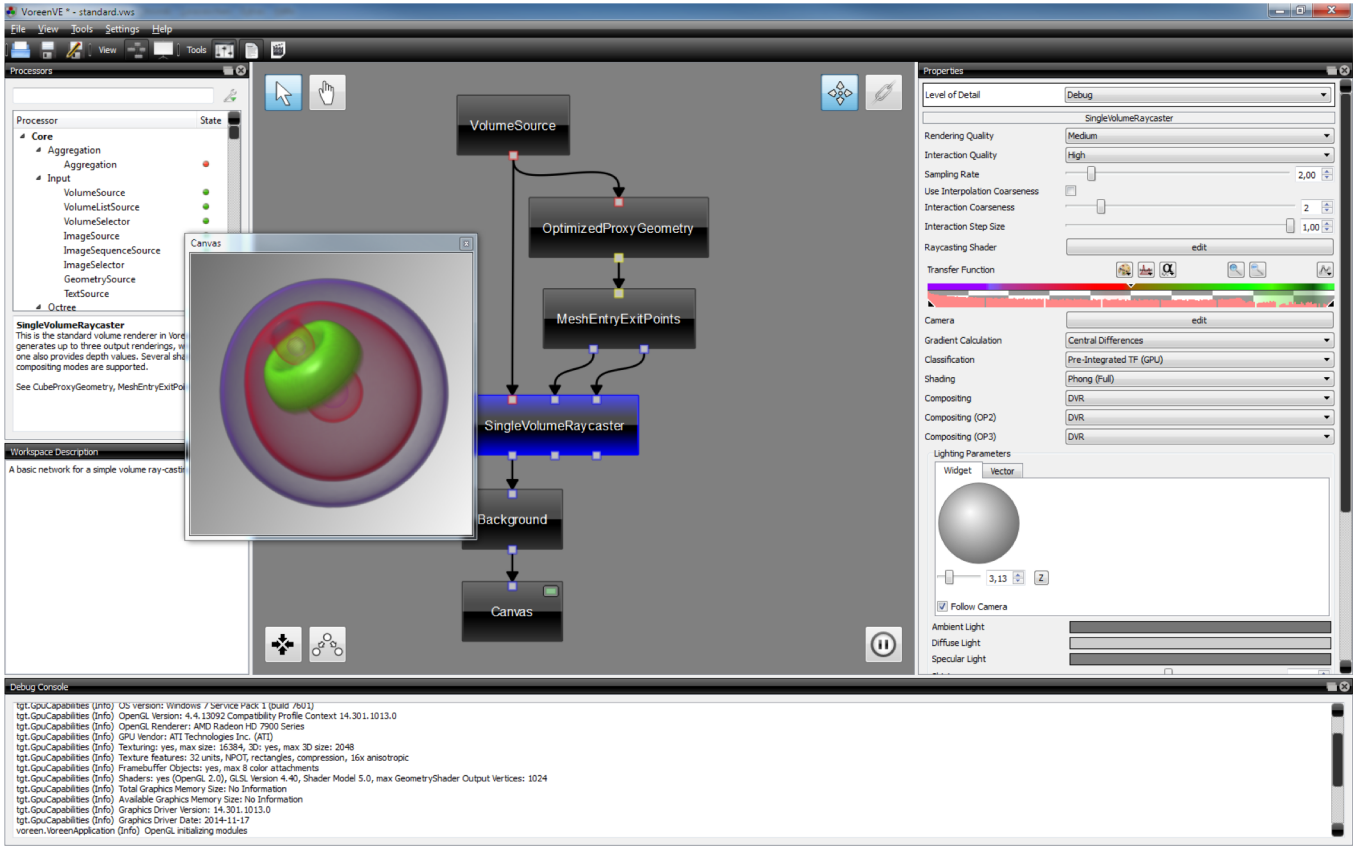
Outline

About Voreen

- Framework for interactive visualization of volumetric data
  - Originally initiated and maintained by the Visualization & Computer Graphics Research Group at the University of Münster in the Collaborative Research Centre 656 'Molecular Cardiovascular Imaging'
  - Now maintained and developed collaboratively by the Pattern Recognition and Image Analysis Group (https://uni-muenster.de/PRIA) and the VISualization and graphIX (VISIX) research group (https://uni-muenster.de/VISIX)
- Open source (GPL) research platform with a focus on rendering / visualization, some preprocessing capabilities and analysis tools
- Functional entities can be reused by exploiting the data-flow metaphor
- Integrates not only volume data (e.g., geometry, flow data, ...)
- Platform independent (Windows, Linux; Mac OS X currently not supported)

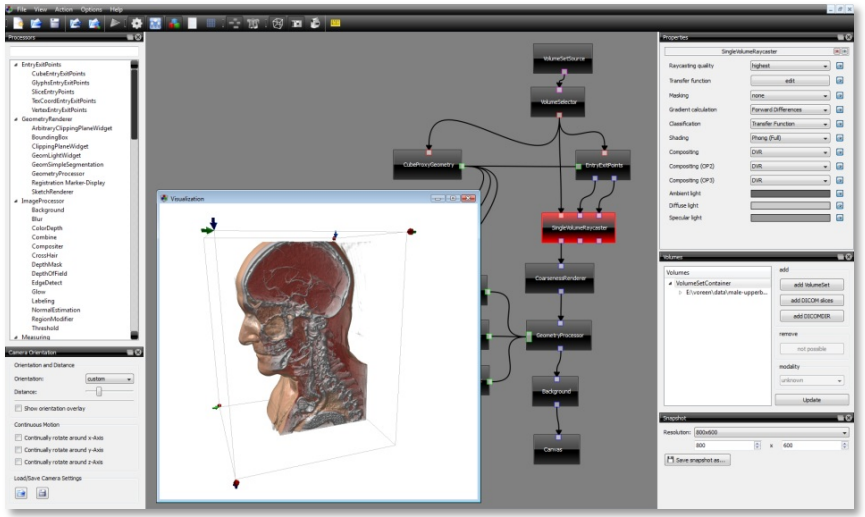


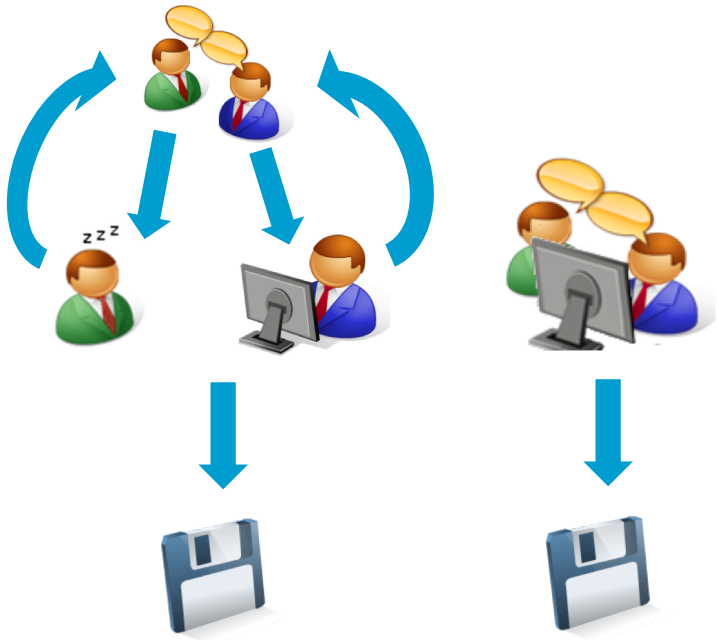
About Voreen



Collaborating with Domain Experts

- Insightful visualizations can only be generated collaboratively
- Challenging for domain experts and computer scientist



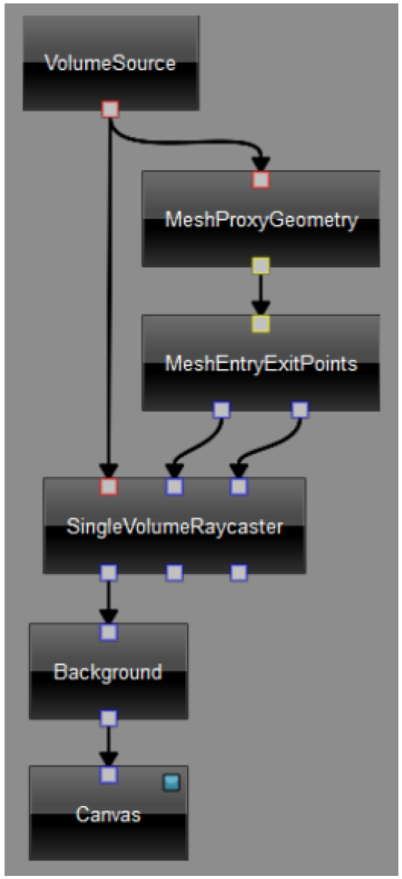


Data Flow Concept

- Data flow network for visual rapid prototyping
  - Modular concept, reusability
- Data (e.g., volume data, geometry, images, …) is transmitted through the network

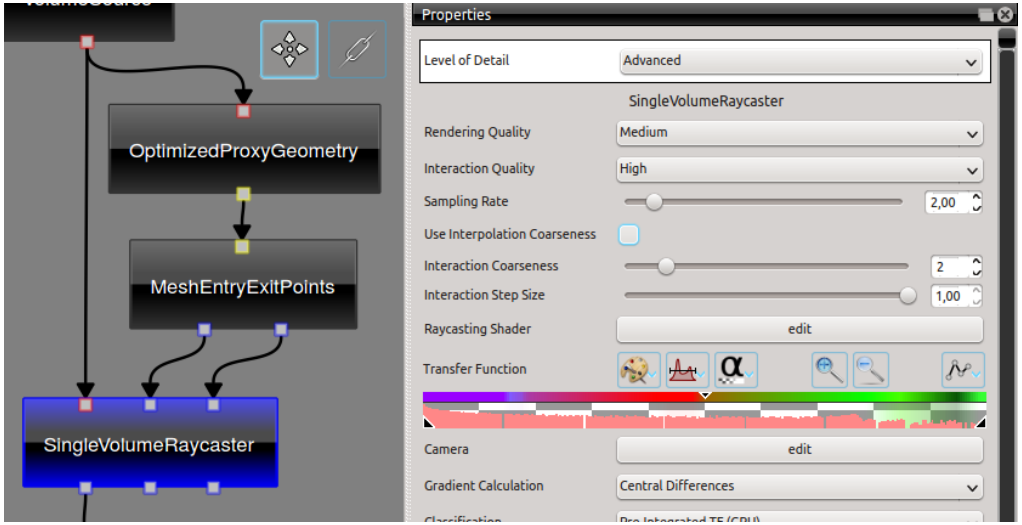
- *Processors*: Entities that perform computations (e.g., rendering, geometry processing, data import)
- Connected by *ports*
  - Different types, e.g., *ImagePort*, *GeometryPort*, …

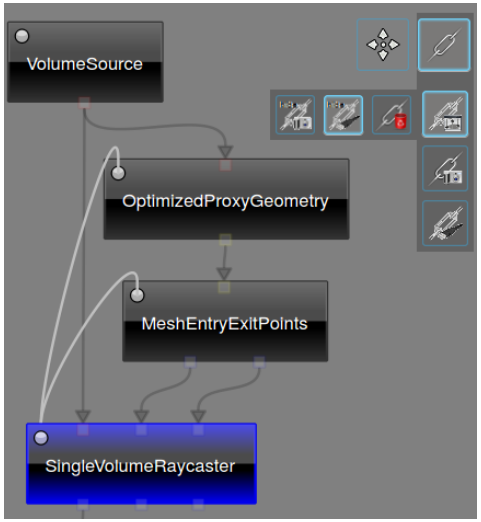
- Central *network evaluator*
  - Determines evaluation order
  - Manages resources



Data Flow Concept

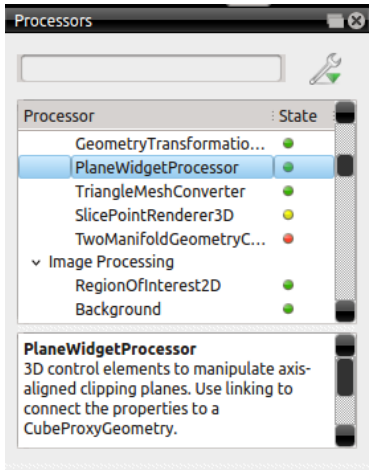
- Configuration of processors through *properties*
  - e.g., lighting, camera, transfer function
- Specify processor behavior
- Interactive manipulation of network behavior through interaction with properties
- *Linking* of properties for synchronization between processors



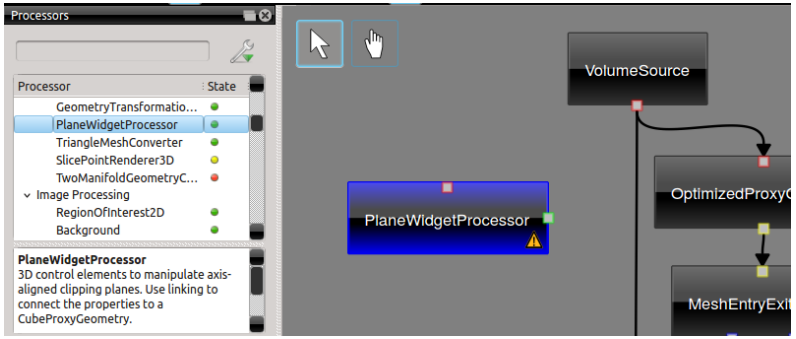


Data Flow Concept

- Reusability of processors
  - Processors are organized in processor list
  - List can be searched and sorted by type, module, name, ...
  - State flags depict the processor's stability
    (= experimental, = testing, = stable)

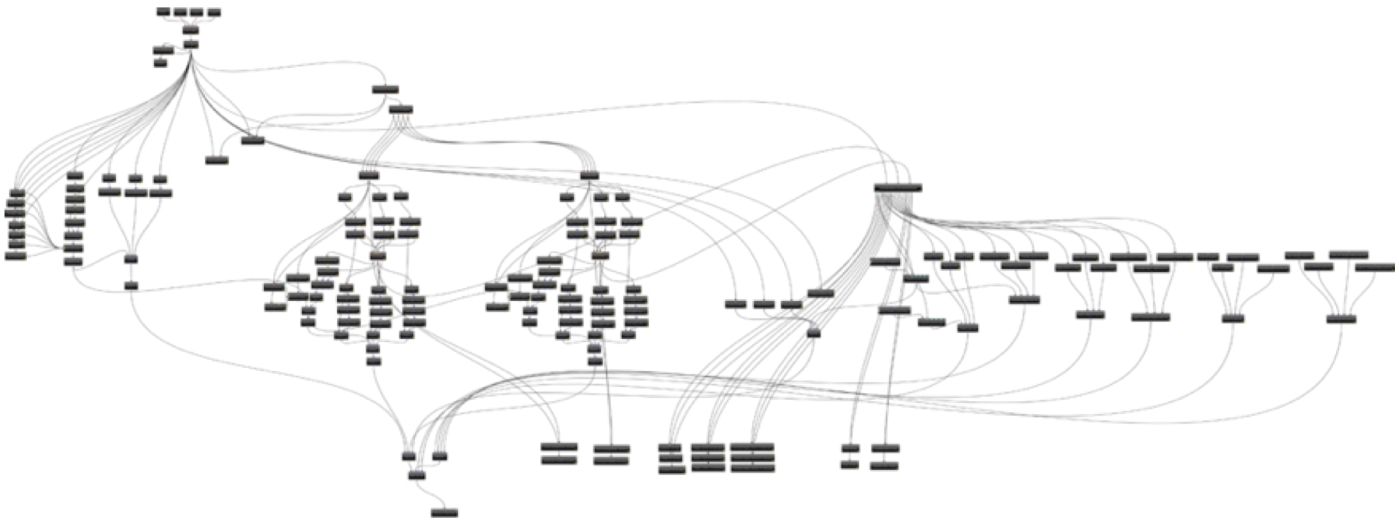


- Processors can be dragged into the network to create a new network or extend an already existing network



Data Flow Concept

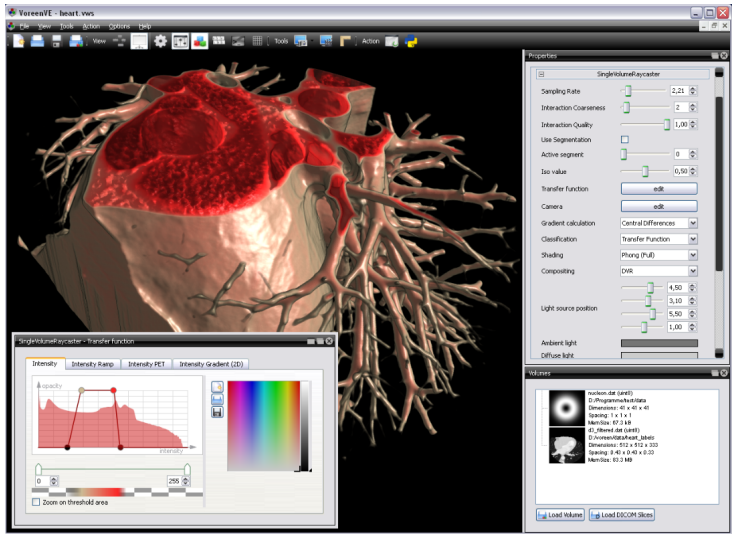
- Drawbacks: networks may become large and confusing for domain experts
- Large number of components and properties, settings, ...



- Solution: provide a more streamlined application user interface for a created workspace

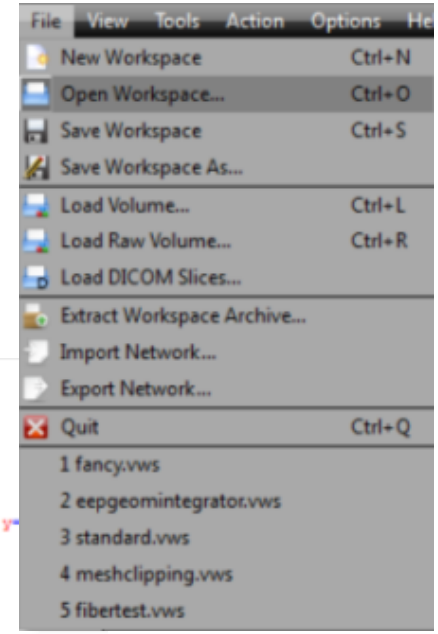
Application Mode

- As an addition to the network mode there exists an application mode.
- Revised and extended in Voreen 5.0
- Visibility of single properties can be configured
- Provides a user interface for the actual application domain as an abstraction from the underlying network



Workspaces

- The current session is serialized within the XML- based Voreen workspace format *.vws*
  - Network topology
  - Property states
  - Processor layout
  - Loaded volumes
  - …




```
<Processor type="MeshEntryExitPoints" name="MeshEntryExitPoints" id="ref8">
    <MetaData>
        <MetaItem name="ProcessorGraphicsItem" type="PositionMetaData" x="-205" y="-174" />
    </MetaData>
    <Properties>
        <Property name="camera" adjustProjectionToViewport="true" id="ref16">
            <MetaData>
                <MetaItem name="EditorWindow" type="WindowStateMetaData" visible="false" x="-955" y=
            </MetaData>
            <position x="-3.16516089" y="1.88449895" z="2.34805989" />
            <focus x="-0.14060999" y="-0.205892" z="0.00218" />
            <upVector x="0.3551189" y="0.87701076" z="-0.3236399" />
        </Property>
        <Property name="filterJitterTexture" value="true" />
        <Property name="jitterEntryPoints" value="false" />
        <Property name="jitterStepLength" value="0.005" />
        <Property name="supportCameraInsideVolume" value="true" />
        <Property name="useFloatRenderTargets" value="false" />
    </Properties>
</Processor>
```

Technical Aspects

- Written in C++
- Exploits OpenGL / GLSL and (optionally) OpenCL, OpenMP
- GUI optional (Qt 5)

- Support for several volume file formats (e.g., RAW, DICOM, HDF5, TIFF-Stacks, ...)

- Main renderer: OpenGL / GLSL volume ray-casting

- Support for out-of-core data sets using an octree data structure and an OpenCL volume ray-casting approach

Outline

1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

Obtaining Voreen

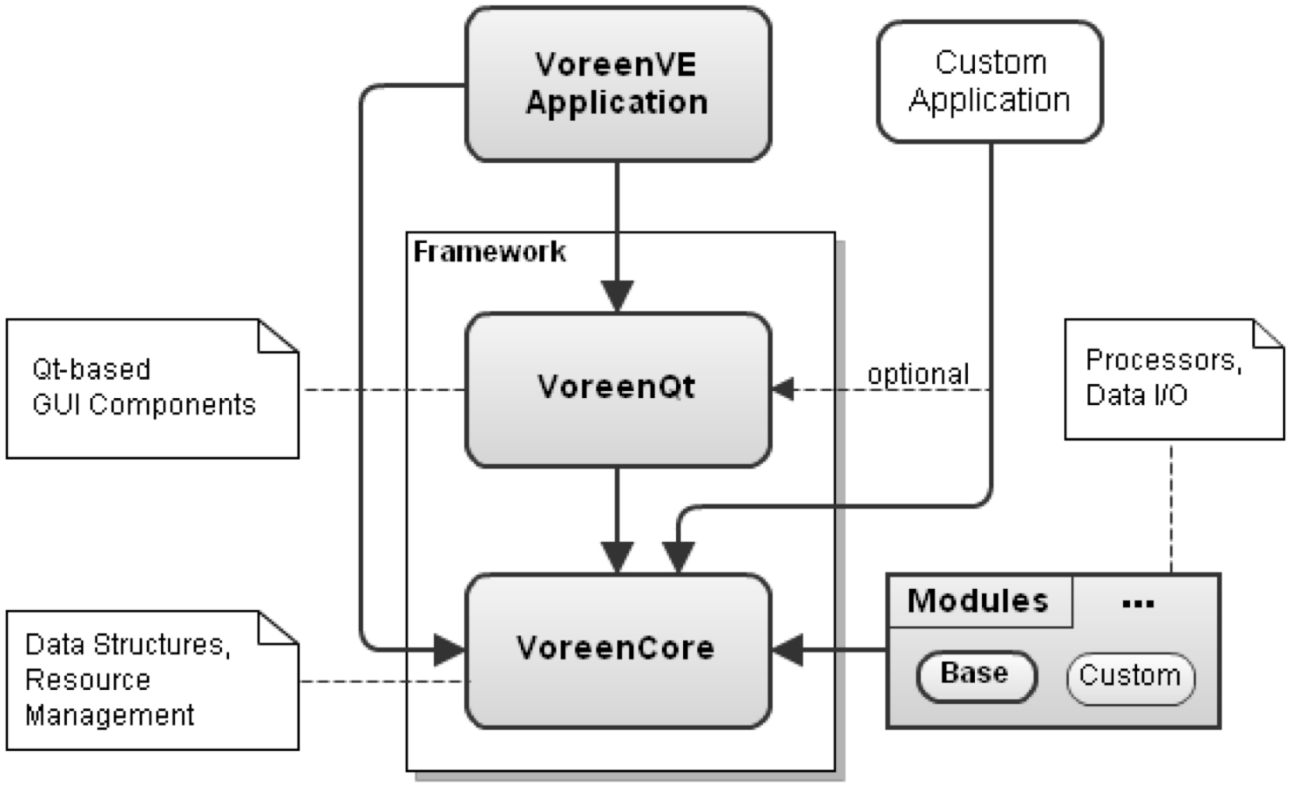
- Download of pre-built version or source code from http://voreen.uni-muenster.de
- Current public version 5.0

- Configuration via the CMake build system
- Instructions for building Voreen from source can be found on the website

Outline

1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

Voreen Architecture



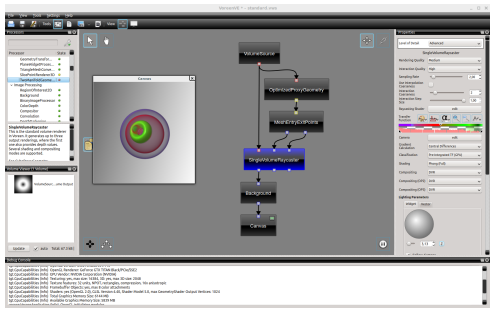
Framework

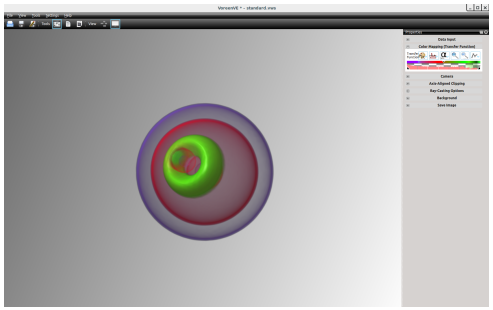
- Voreen core library
  - Ports, Properties
  - Processor base classes
    - *Processor*, *VolumeProcessor*, *RenderProcessor*, *ImageProcessor*, ...
  - Data structures
    - Data flow network, volumes, geometries, ...
  - Network handling
  - Minimal external dependencies (OpenGL, GLEW, Boost, TinyXML)

- Voreen Qt library
  - Property widgets
  - Processor widgets
  - Graphical network editor

VoreenVE

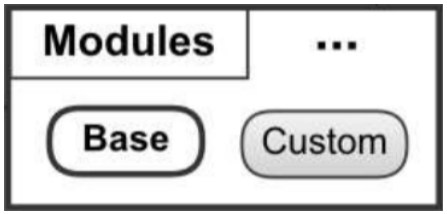
- Visualization environment for rapid prototyping
  - Auto-generated property widgets (Voreen Qt Library)
  - Visual debugging
    - Inspection of intermediate rendering results
  - Runtime shader editing
- Application mode for domain experts
  - Hides the underlying network
  - Visibility of single properties can be configured

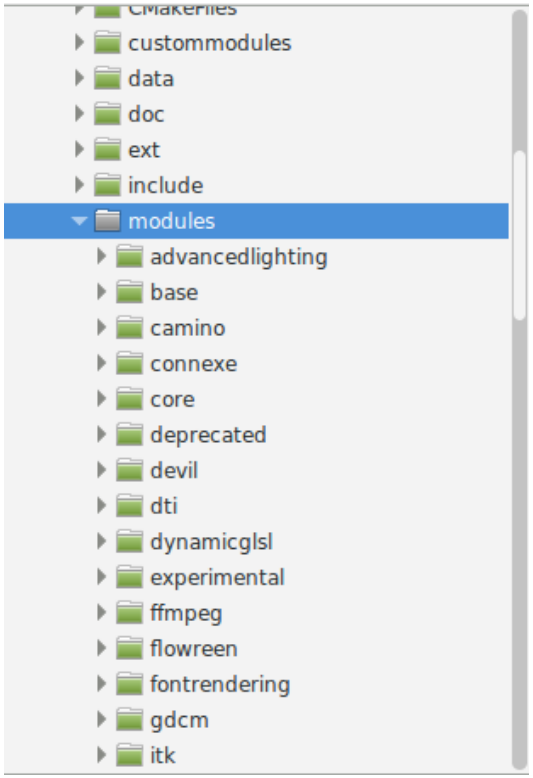




Modules

- Recommended way to extend Voreen
- Encapsulate rendering and data processing functionality
  - Processors
  - Data reader and writers
- Are included / excluded from the build process using CMake configuration options
- May contain external libraries
- Dedicated directory for custom modules



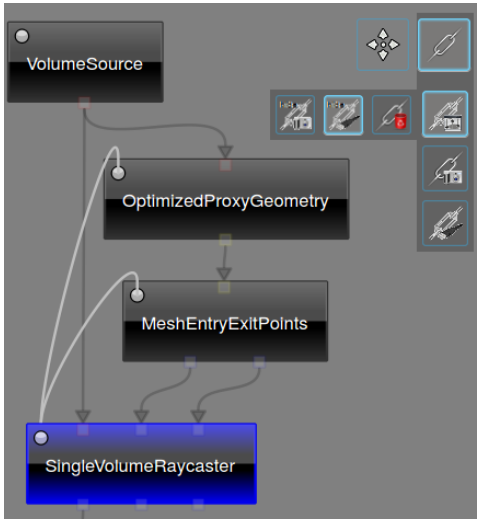


Outline

1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

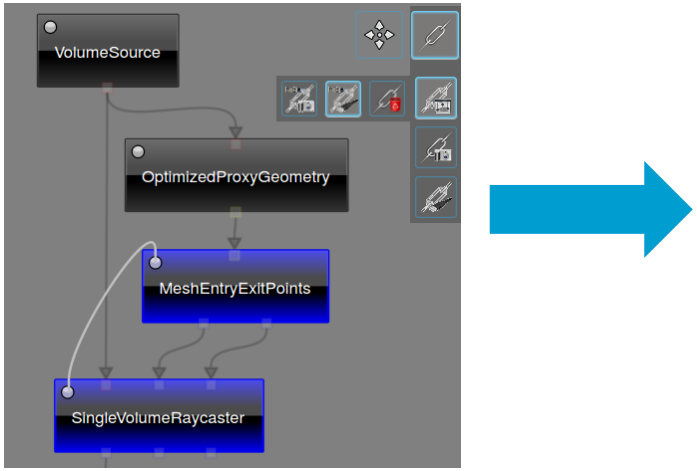
Property Linking

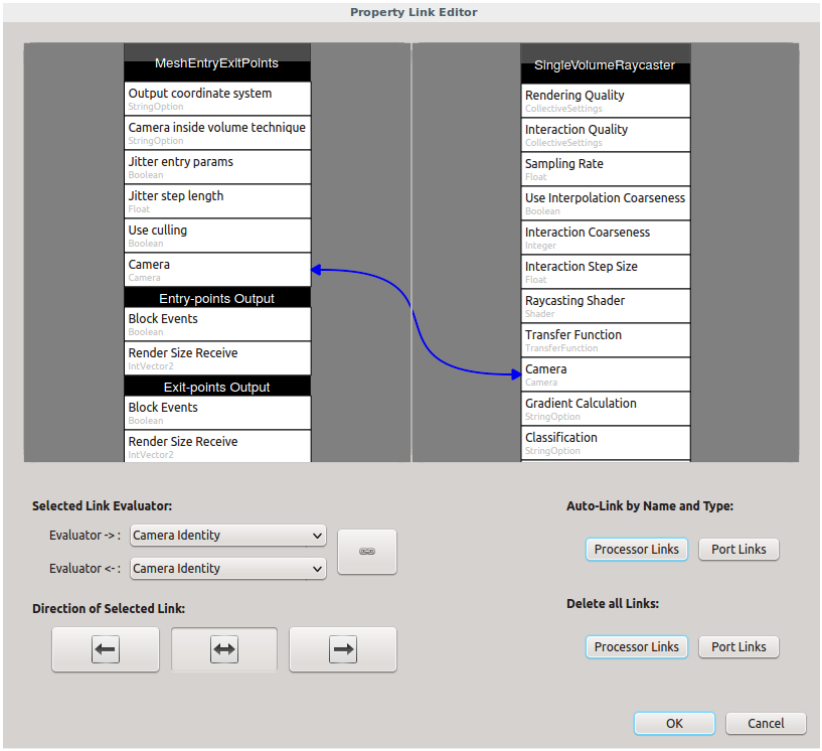
- Properties of the same type can be linked (value synchronization)
  - Within or across processors
  - Uni- or bidirectional
  - Cycle prevention
- Linking of differing, but compatible property types is also possible
  - Float ⟷ Integer ⟷ Boolean
- Linking of more complex properties (e.g., transfer functions)
- (Optional) auto-linking of camera properties



Managing Links in VoreenVE

- Network editor provides *linking layer*
  - Links are represented by arrows
  - Port connections are faded out
  - Dragging a line between processors opens *linking dialog*





Outline

1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

Extending Voreen

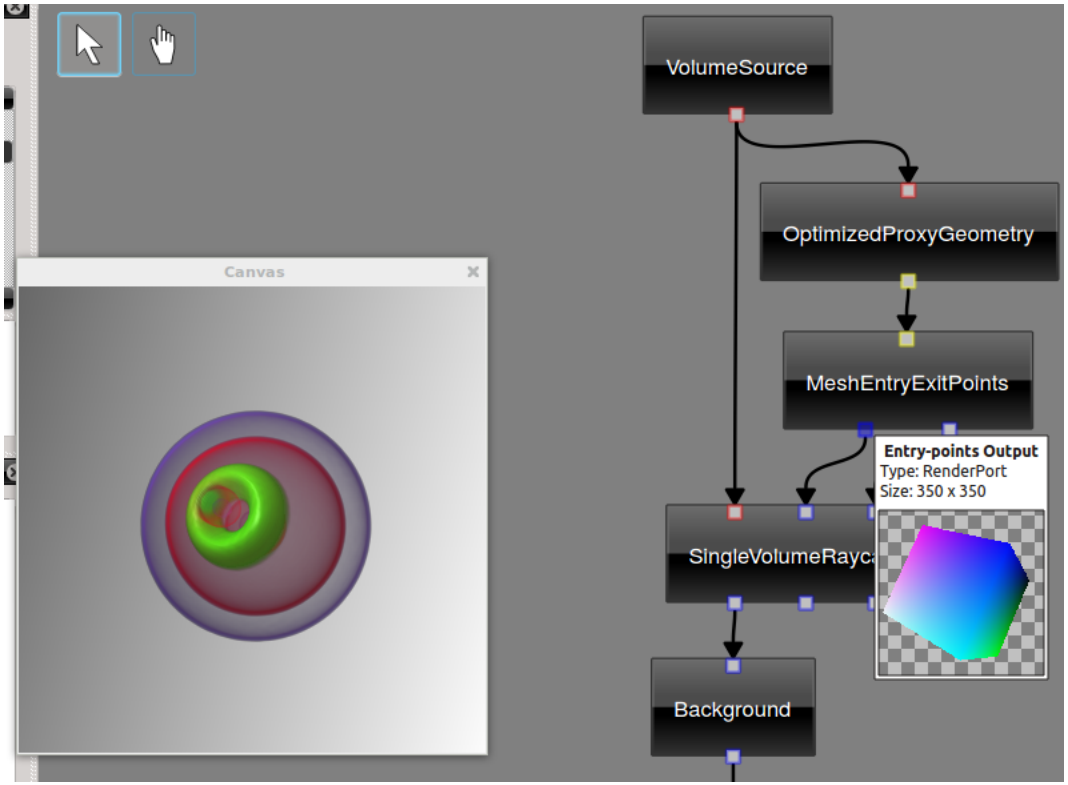
- Tutorials on the website (more to come)
  - Adding a module
  - Adding a processor
- Sample-module *voreen/modules/sample* can be used as a starting point
  - Documentation in the source code should be helpful
- Use existing processors as templates
  - *process()*-method does (almost) all the work, is called during network evaluation
  - Adding ports / properties using *addPort()* and *addProperty()* in constructor
  - Callback-functions for performing actions on property changes can be realized using *MemberFunctionCallback*
- ...

Outline

1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

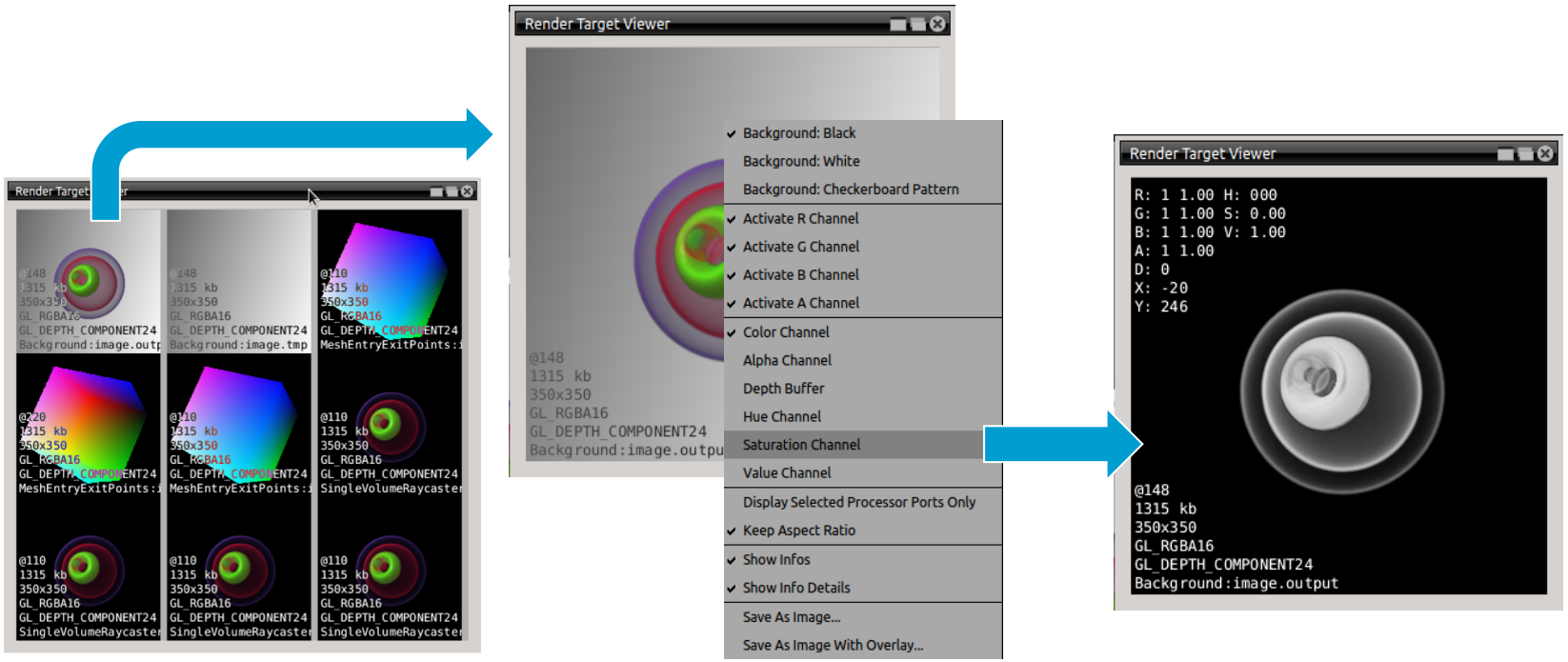
Visual Debugging

- By hovering over render ports, their content can be inspected



Visual Debugging

- Render target viewer allows to inspect the color, alpha, depth layer , ...
- For all render targets (e.g., *RenderPort* objects)



Serialization

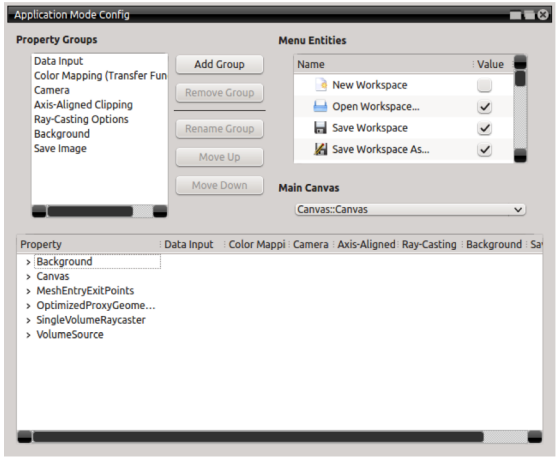
- Workspace serializes network topology and property states to XML
- Custom data can be serialized by implementing the *Serializable* interface and overwriting the *serialize*- and *deserialize*-methods
- Serializer supports primitive data types, *tgt* data types (e.g., vectors, matrices, ...), and STL containers

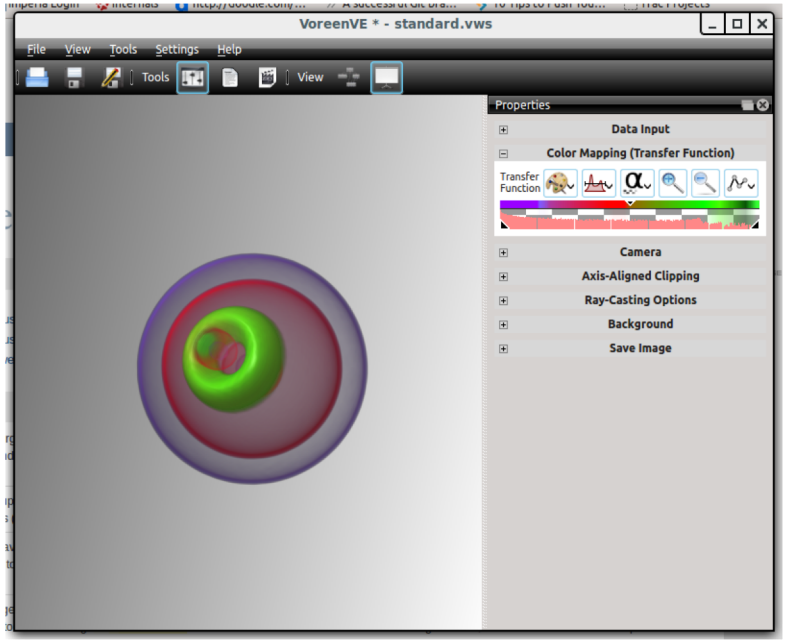
```
std::vector< std::pair<float, tgt::vec3> > myData_;

void TestProcessor::serialize(XmlSerializer& s) const {
    s.serialize("MyData", myData_);
}
void TestProcessor::deserialize(XmlDeserializer& d) {
    d.deserialize("MyData", myData_);
}
```

Application Mode Configuration

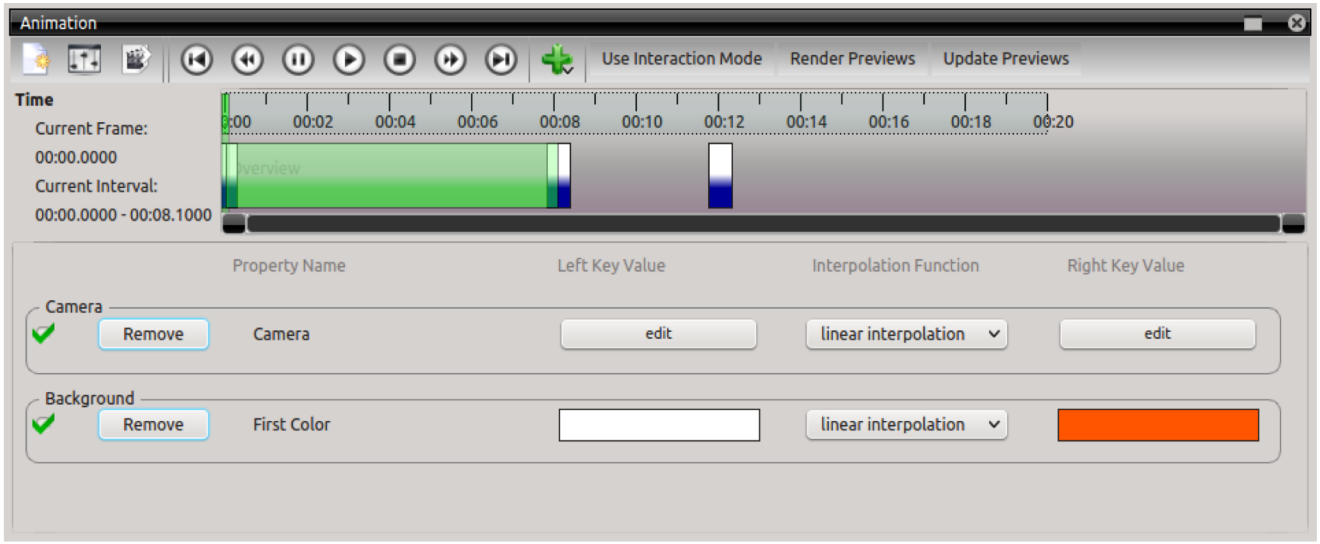
- Configures the application mode by creating property groups and adding selected properties to a specific group
- Properties of multiple processors can be grouped by functionality in the interface
- Independent from network topology





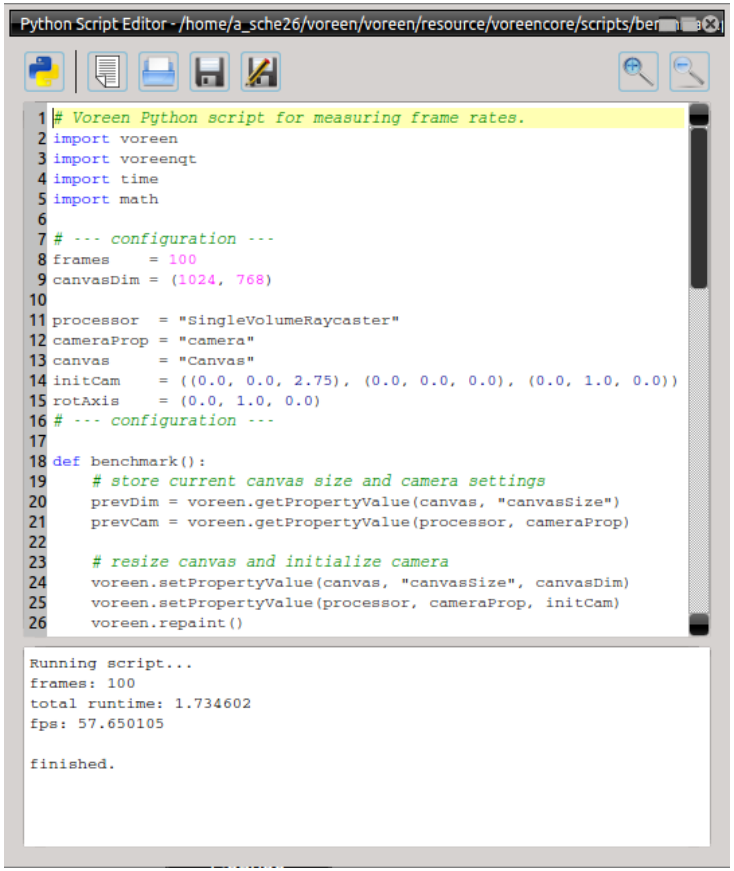
Animation

- Allows to animate (almost) all properties that have been added to the application mode configuration
- User specifies key frames for which the property value is stored
- Automatic interpolation of values in intervals between key frames
- Video export



Python Scripting

- Generic read / write access to almost all types of properties, including cameras
- Volume and transfer function loading
- Canvas snapshots
- Integrated Python editor
- Since Voreen 5.0: Python 3
  (earlier versions of Voreen: Python 2.7)



Selected Modules

- *Base*
  - Base functionality, standard rendering processors (2D and 3D)
  - Volume and geometry processors
  - Clipping
  - Bounding boxes
  - Image processors (post processing etc.)
  - Volume I/O
- *Ffmpeg*
  - Video export
- *OpenCL*
  - Rendering of large data sets (≥ 30 GB)
- *OpenMP*
  - Parallel code execution for various processors

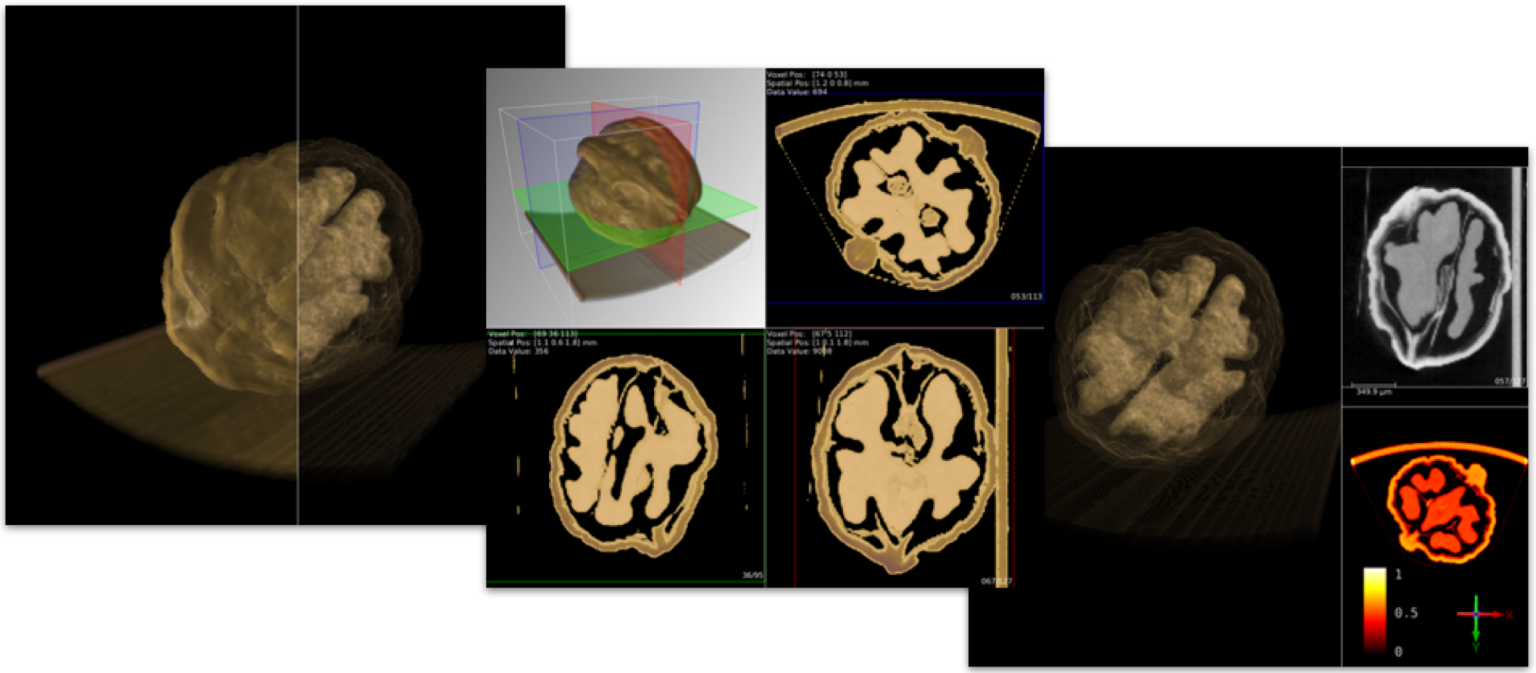
Selected Modules

- *Plotting*
  - Multiple plotting functions (2D / 3D)
- *RandomWalker*
  - Semi-automatic 3D segmentation of volume data sets

- ... and many more.

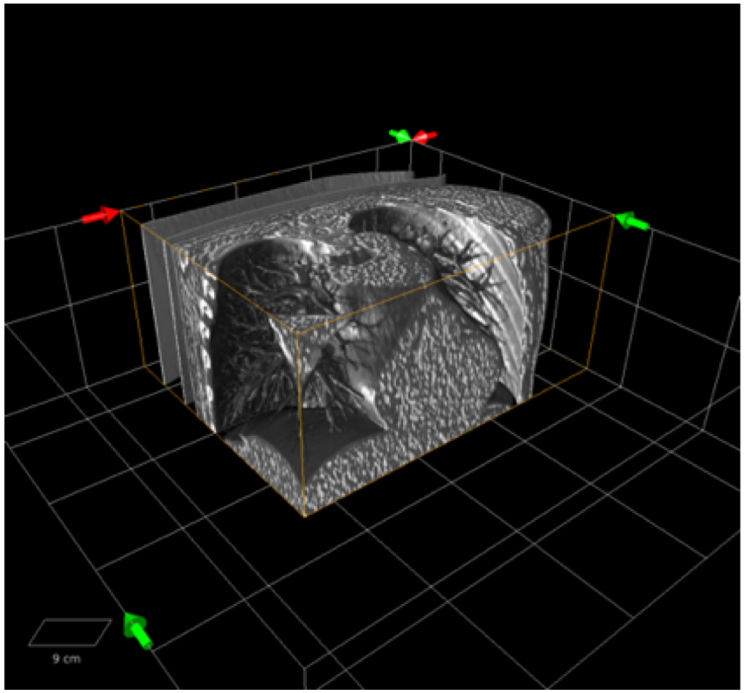
Selected Functionality Examples

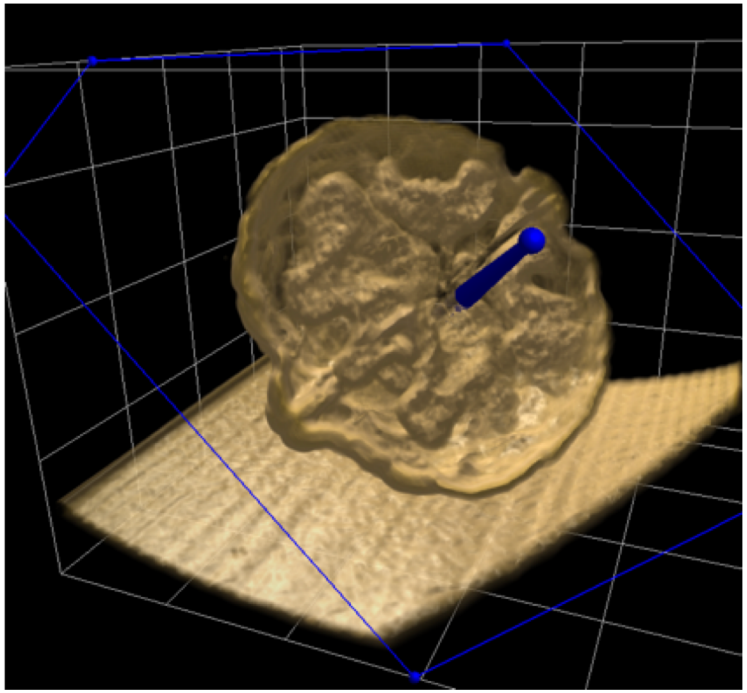
- Configurable views: Splitter, triple view, quad view, tabs, ...



Selected Functionality Examples

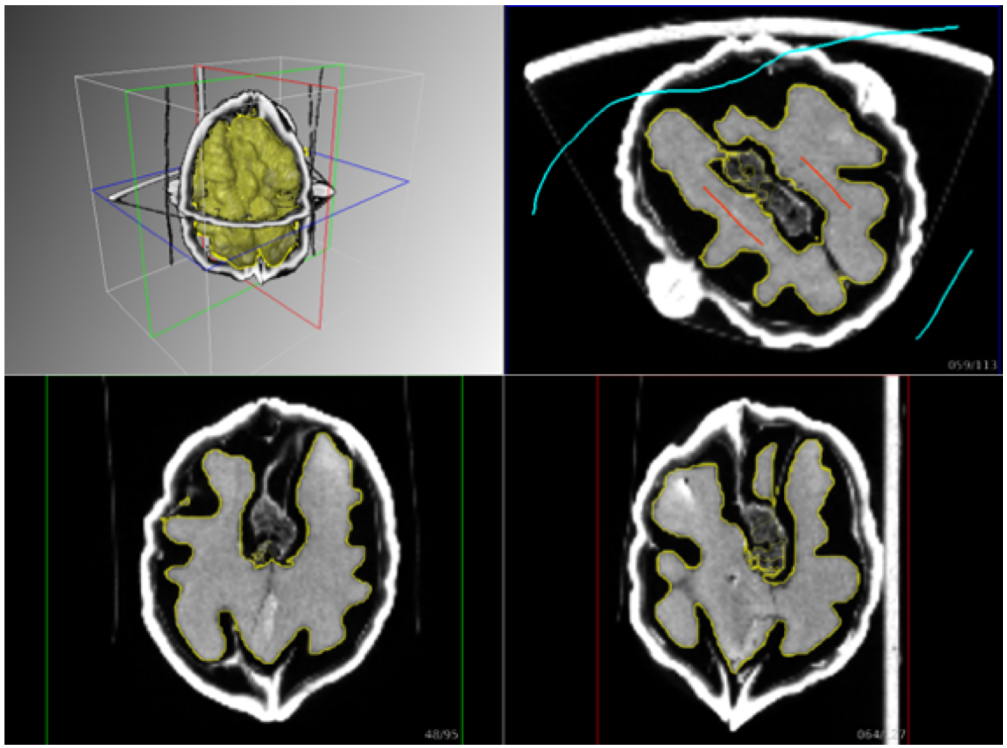
- Interactive clipping: Axis-aligned clipping, arbitrary clipping, on-screen handles





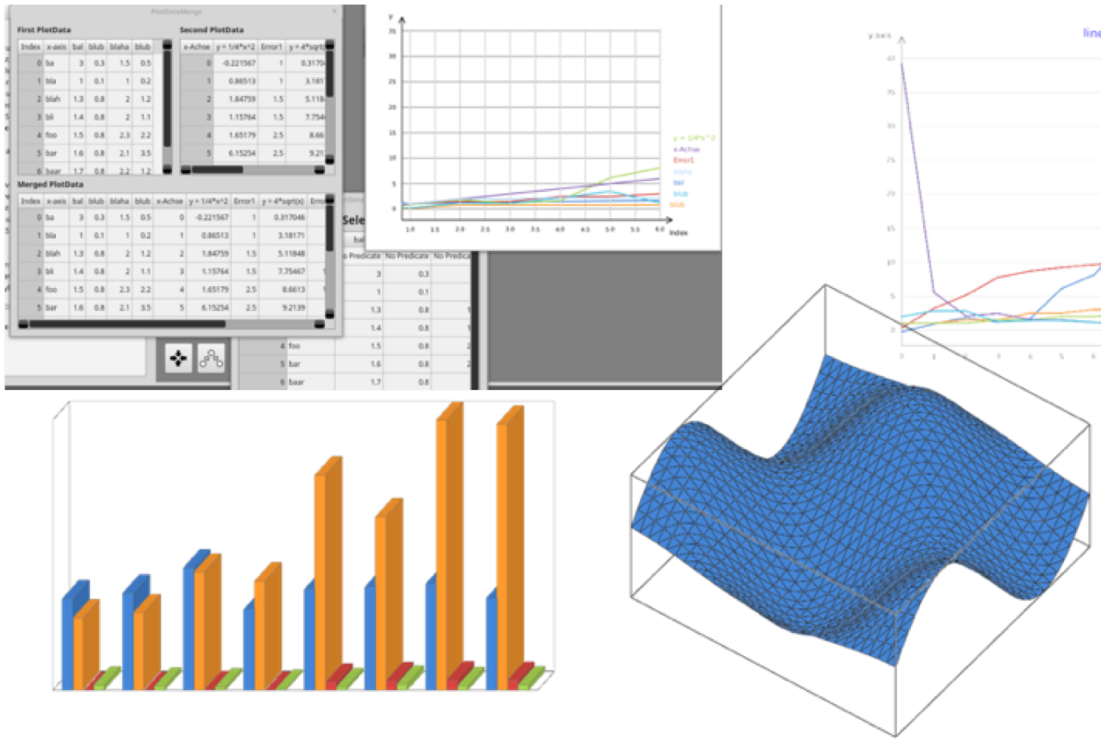
Selected Functionality Examples

- Random walker: semi-automated volume segmentation



Selected Functionality Examples

- Plotting: Support for CSV files, line plots, bar plots, 3D surface plots, ...

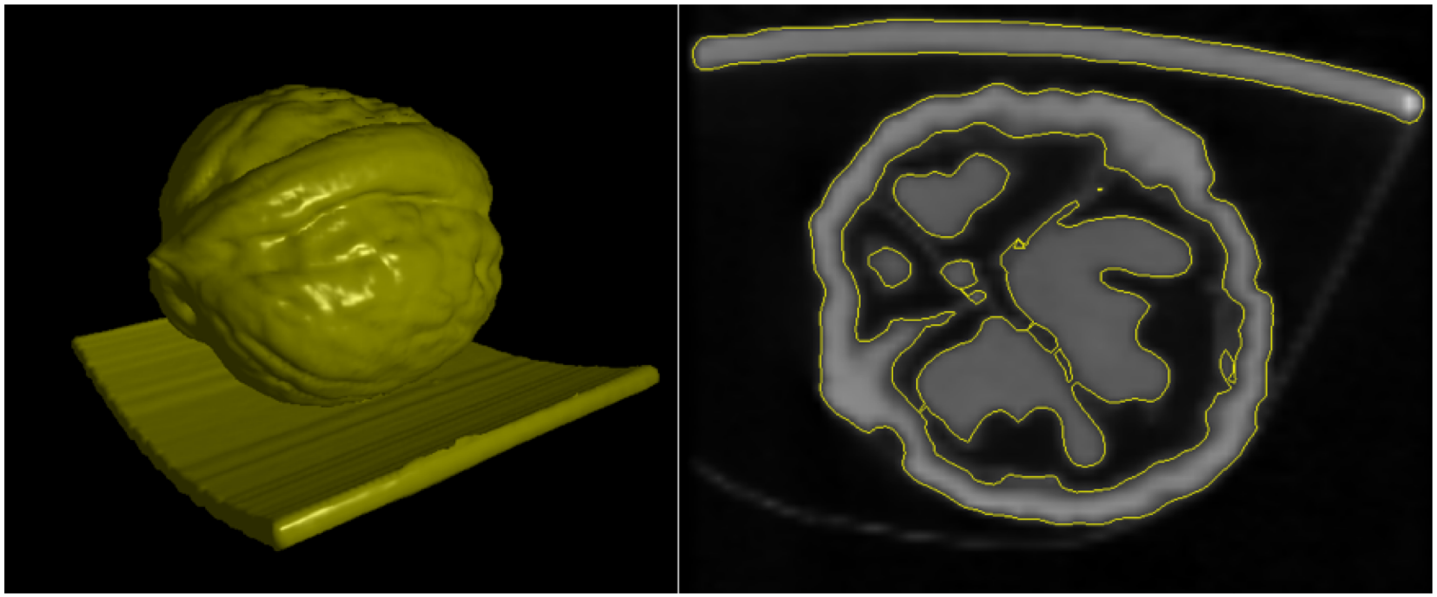


Selected Functionality Examples

- Large Volume Visualization:
  - Interactive 3D and 2D visualization of multi-channel volume data (e.g., lightsheet microscopy image stacks)
  - Support for TIFF / OME TIFF image stacks
  - HDF5 file support (incl. compression)
  - Rendering of large data sets (100 GB and more)

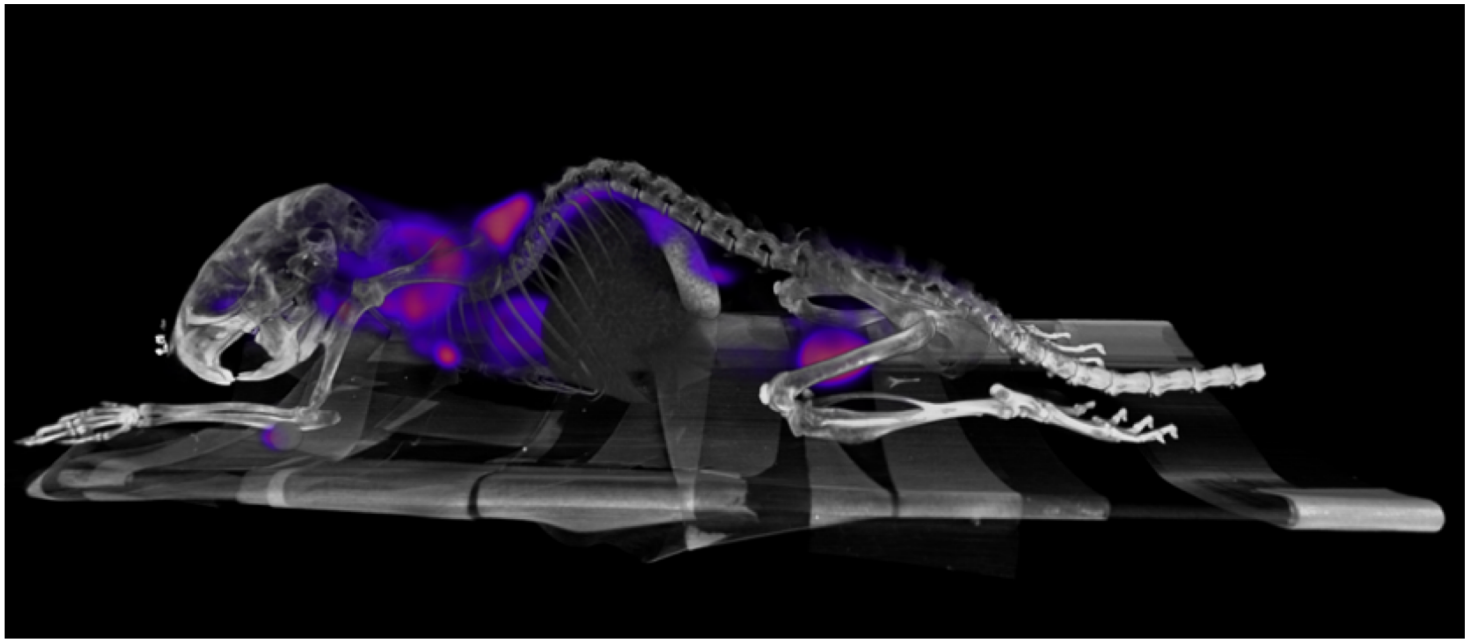
Selected Functionality Examples

- Surface Extraction based on iso values



Selected Functionality Examples

- Multivolume Raycasting: Simultaneous 3D visualization of multi-modal datasets



Selected Functionality Examples

- Volume Registration: Landmark registration, interactive (manual) registration