

Voreen – Volume Rendering Engine

Aaron Scherzinger, Tobias Brix

Version 4.5 alpha

Outline

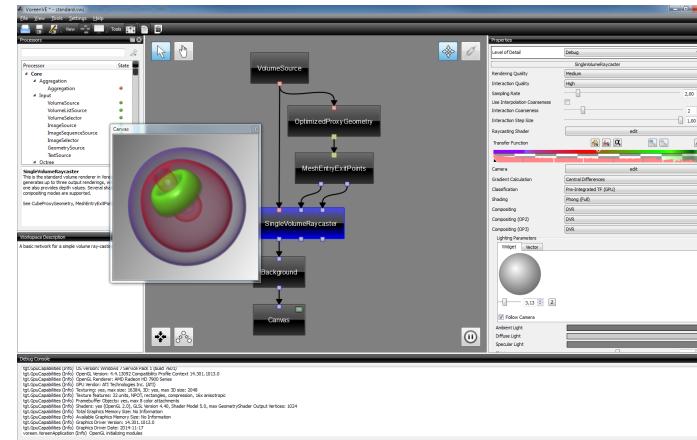
1. About Voreen
2. Obtaining Voreen
3. Project Structure
4. Property Linking
5. Extending Voreen
6. Additional Features

Outline

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

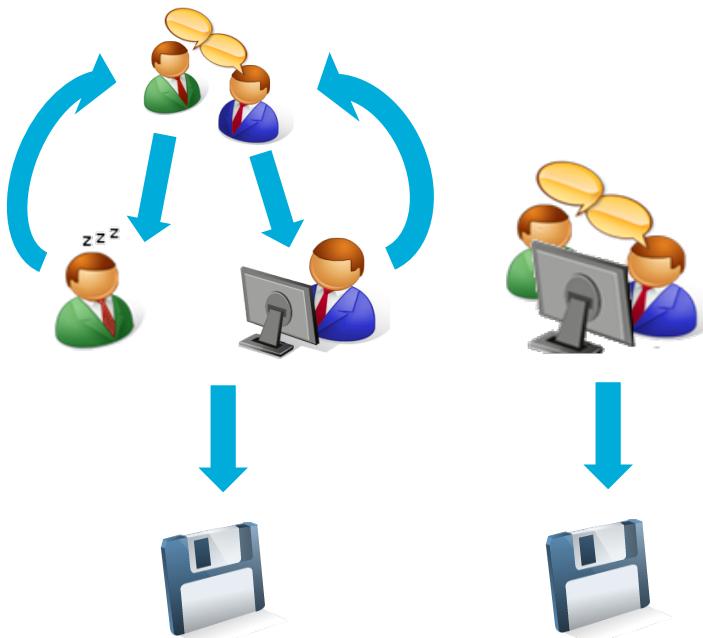
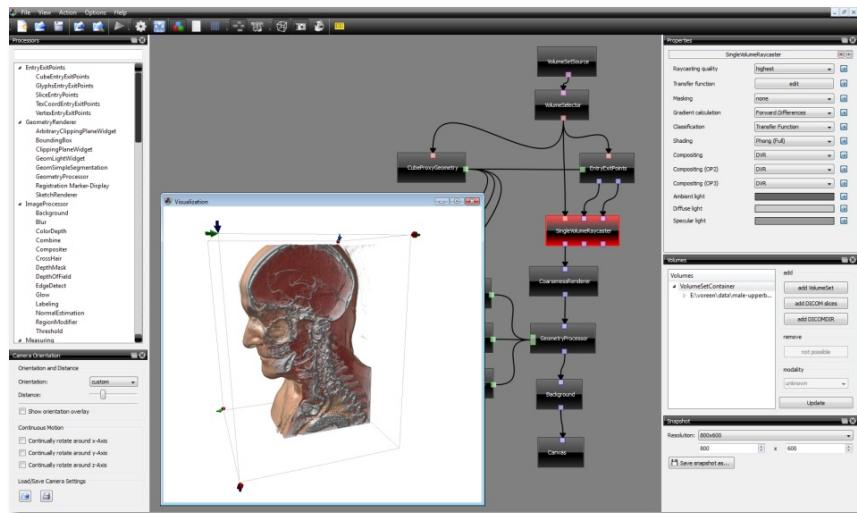
About Voreen

- Framework for **interactive visualization of volumetric data**
 - Initiated and maintained by the Visualization & Computer Graphics Research Group
 - Collaborative Research Centre 656 ‘Molecular Cardiovascular Imaging’
- Research platform with a focus on rendering / visualization, some preprocessing capabilities and analysis tools
- Processors can be reused as functional entities by exploiting the **data-flow metaphor**
- Integrates not only volume data (e.g., geometry, flow data, ...)
- Open source (GPL)
- Platform independent (Windows, Linux;
Mac OS X currently not supported)



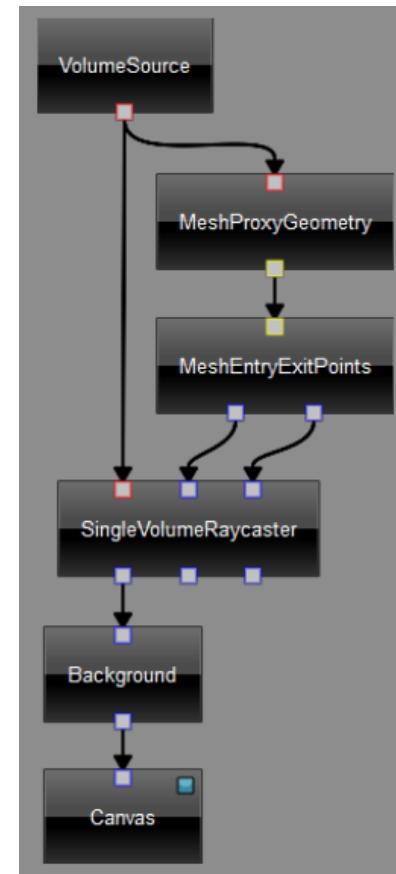
Collaborating with Domain Experts

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



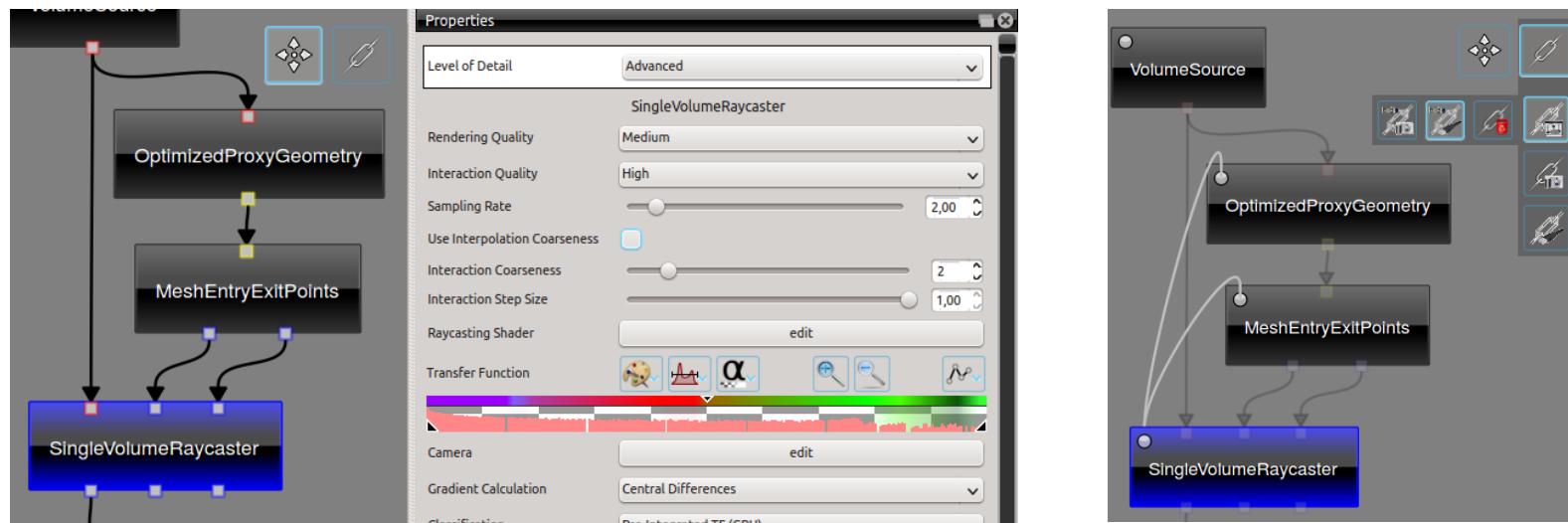
Data Flow Concept

- 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



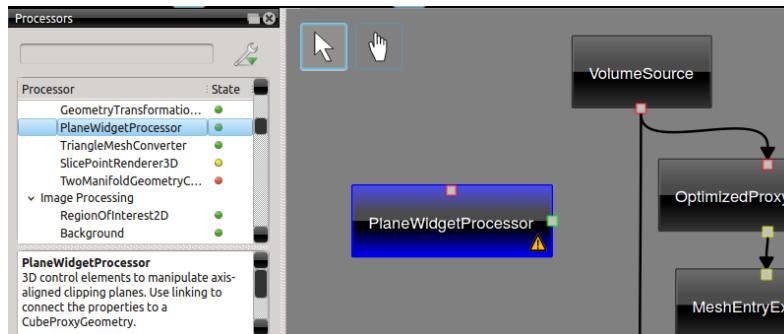
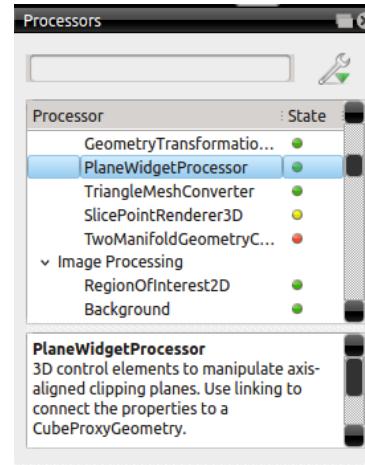
Data Flow Concept

- 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



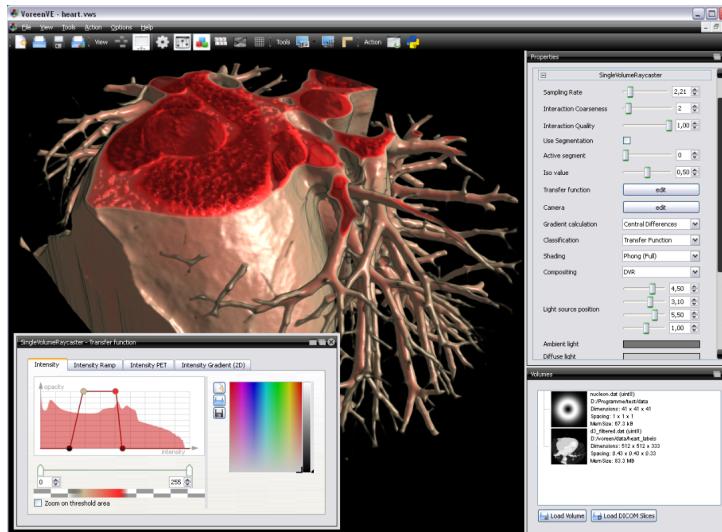
Data Flow Concept

- 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



Application Mode

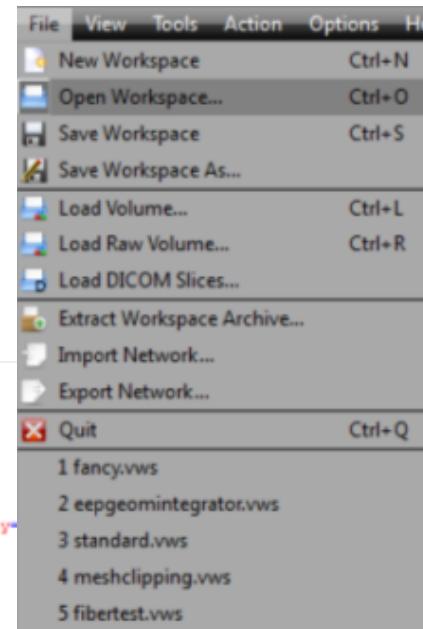
- As an addition to the network mode there exists an application mode.
- Revised and extended in development version 4.5 alpha
- Visibility of single properties can be configured
- Provides a user interface for the actual application domain as an abstraction from the underlying network



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="174" />
            </MetaData>
            <position x="-3.16516089" y="1.88449895" z="2.34805959" />
            <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="useFloatRenderTarget" value="false" />
    </Properties>
</Processor>
```



Technical Aspects

- Written in C++
- Exploits OpenGL / GLSL, OpenCL, OpenMP
- GUI optional (Qt)
- Support for several volume file formats (e.g., RAW, DICOM, 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

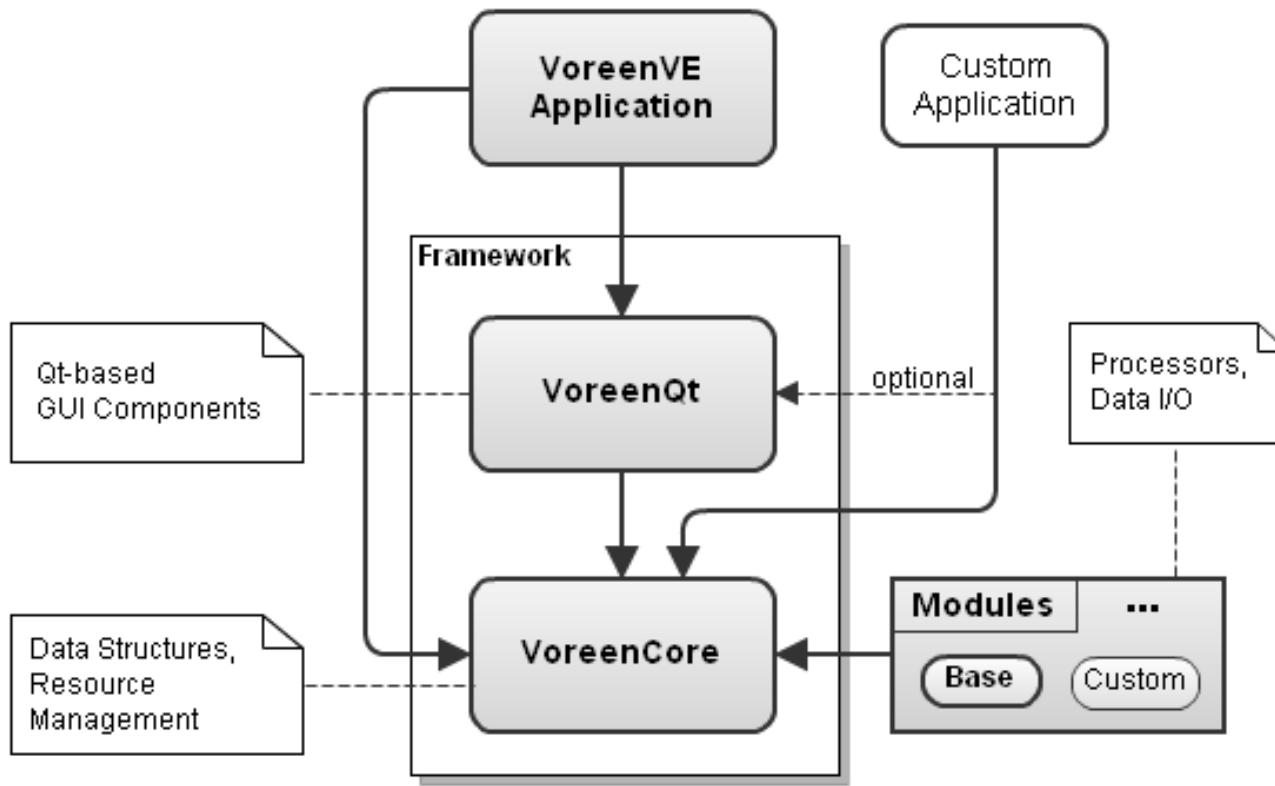
Obtaining Voreen

- Download of pre-built version or source code from <http://voreen.uni-muenster.de>
- Website: public versions up to 4.4
- Internal git repository: [4.5 alpha](#)
 - Public git repository will be available in 2015
- 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)



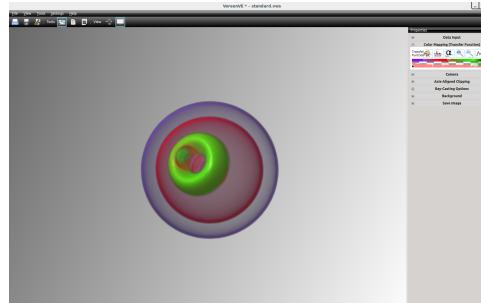
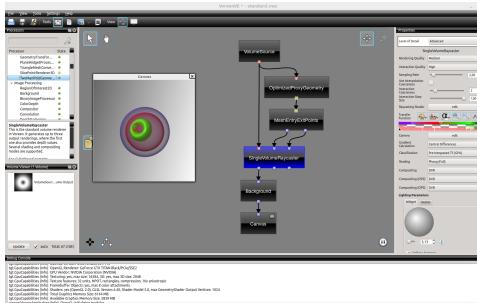
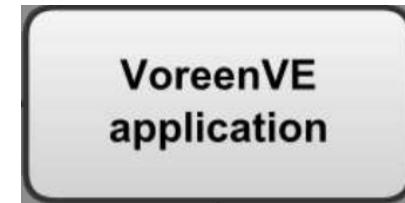
- Voreen **Qt library**

- Property widgets
- Processor widgets
- Graphical network editor



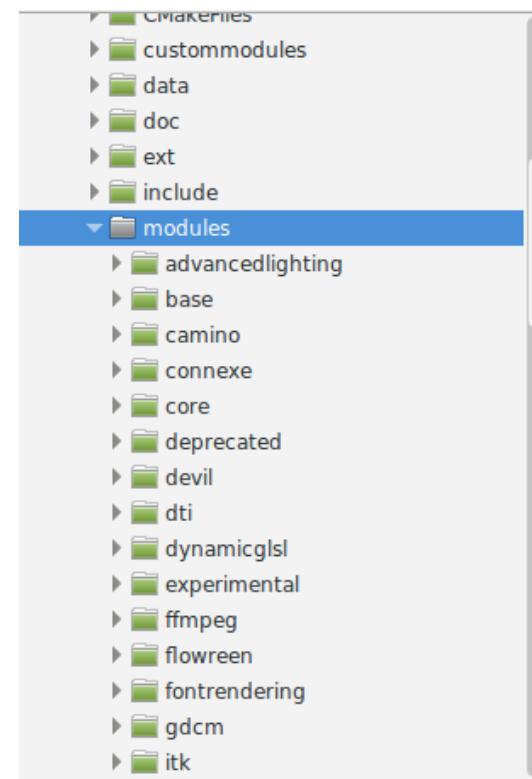
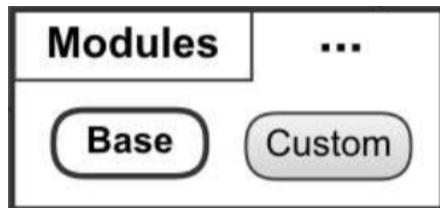
VoreenVE

- 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**

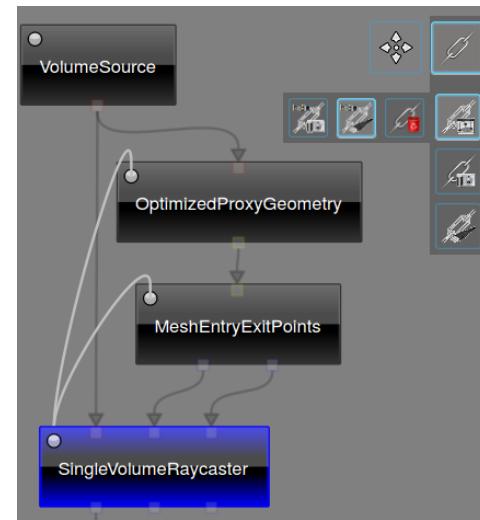


Outline

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

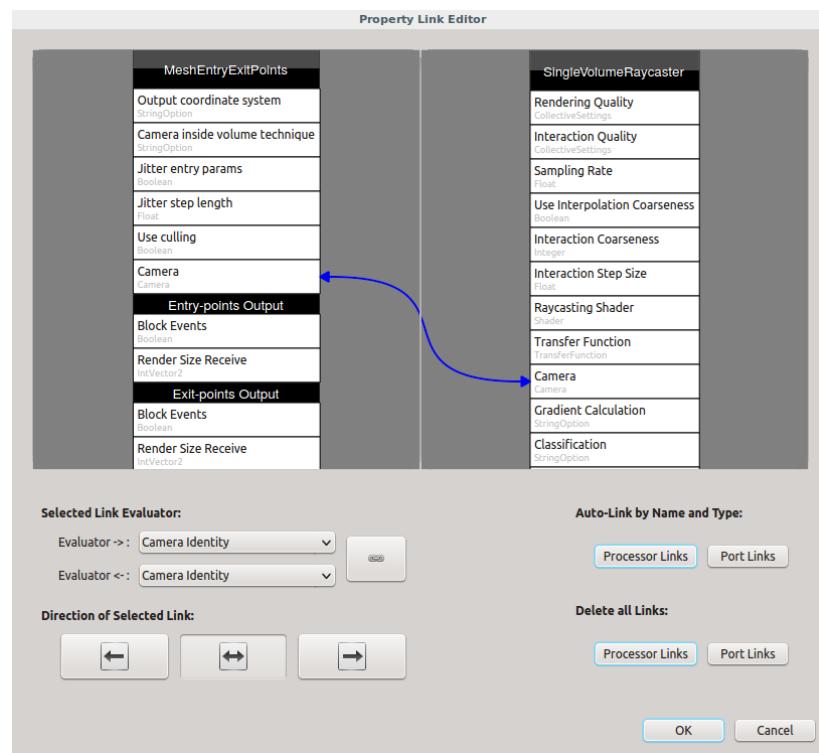
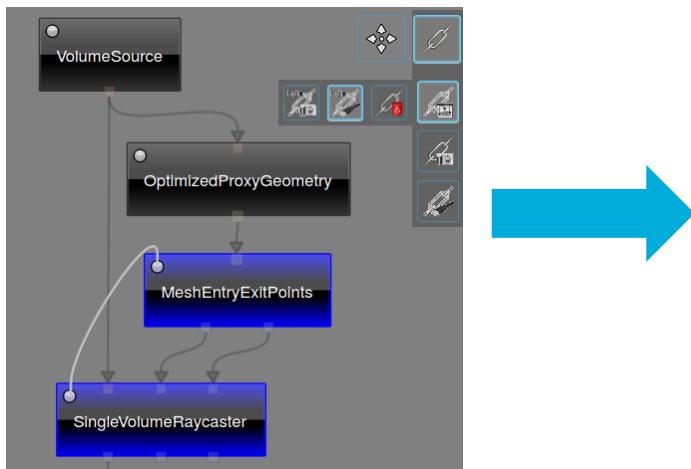
Property Linking

- 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 \leftrightarrow Integer \leftrightarrow 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

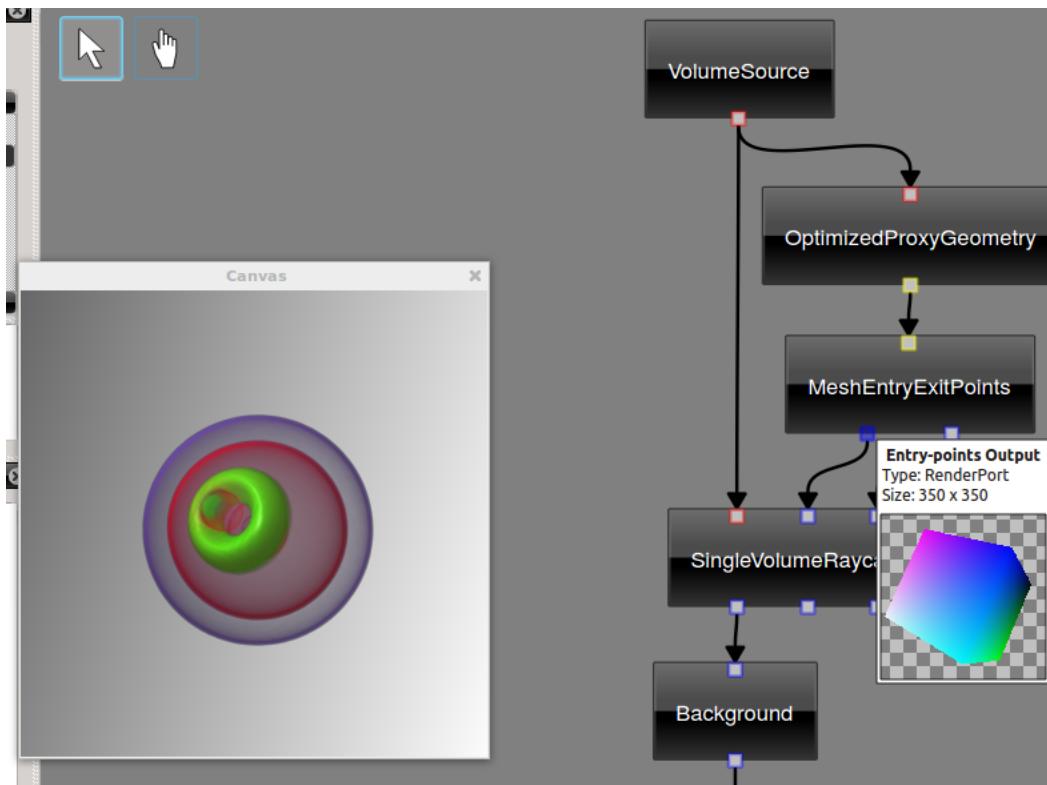
- Tutorials on the website (more to come)
 - Adding a module
 - Adding a processor
- Sample-module *voreen/modules/sample* can be used as a starting point
 - Documentation in the source code should be helpful
- Use existing processors as templates
 - *process()*-method does (almost) all the work, is 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

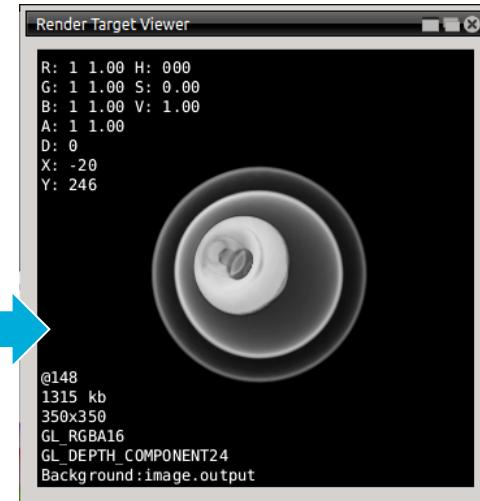
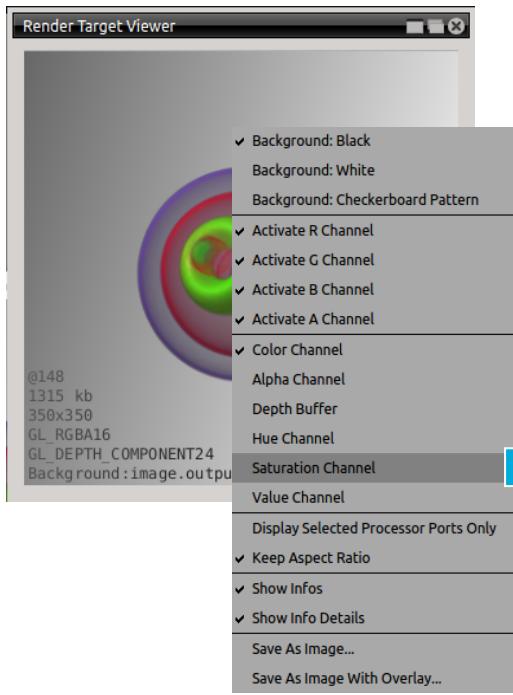
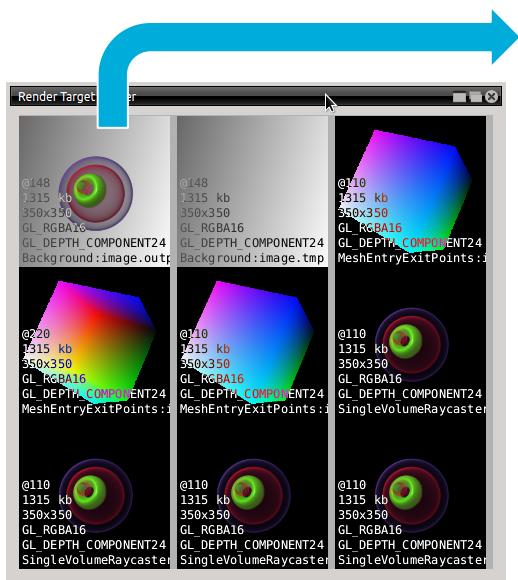
Visual Debugging

- 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

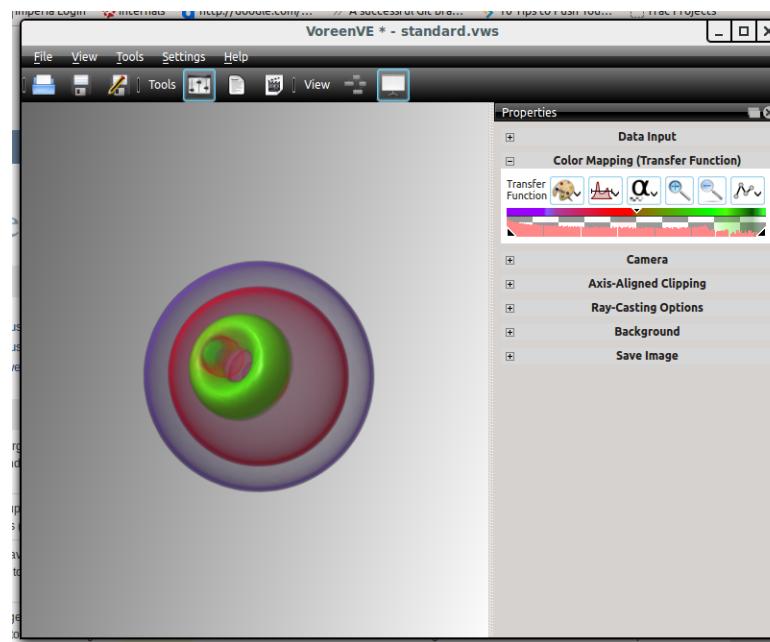
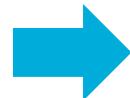
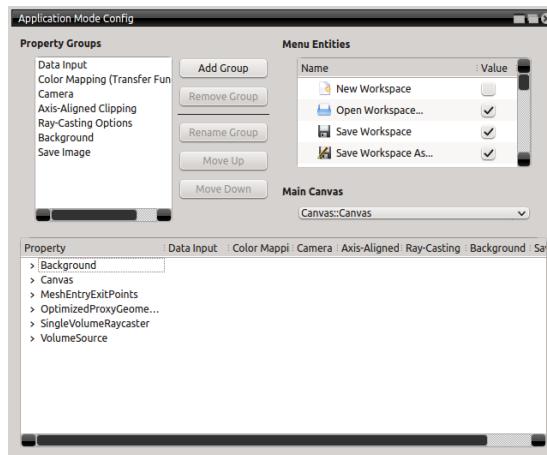
- Workspace serializes network topology and property states to XML
- Custom data can be serialized by implementing the *Serializable* interface and overwriting the *serialize-* and *deserialize-*methods
- Serializer supports primitive data types, *tgt* data types (e.g., vectors, matrices, ...), and STL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

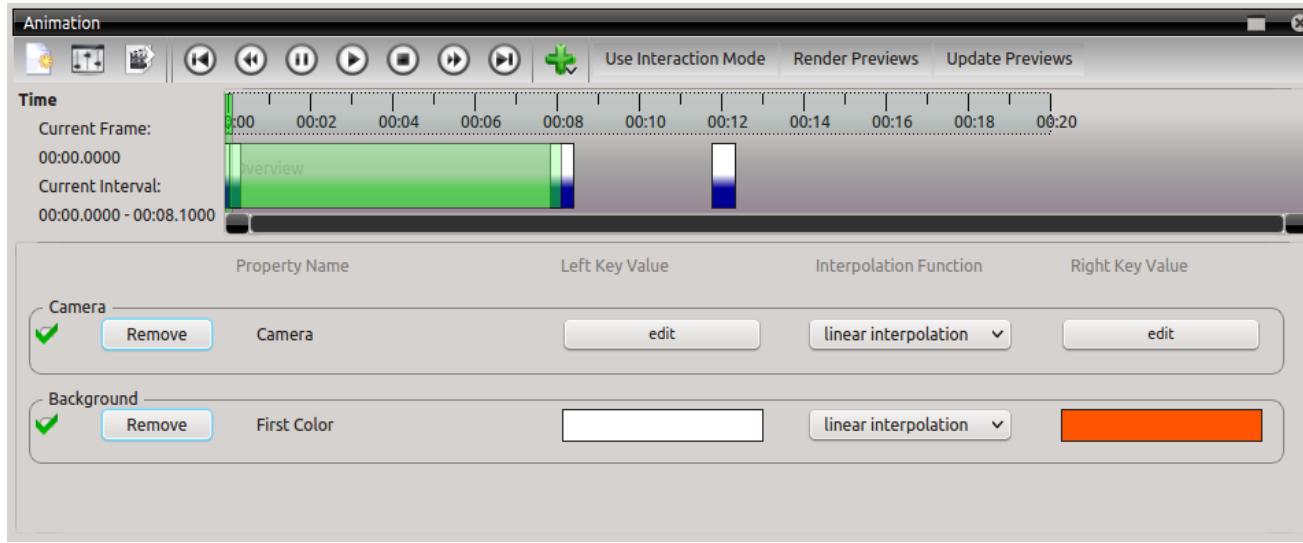
Application Mode Configuration

- 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



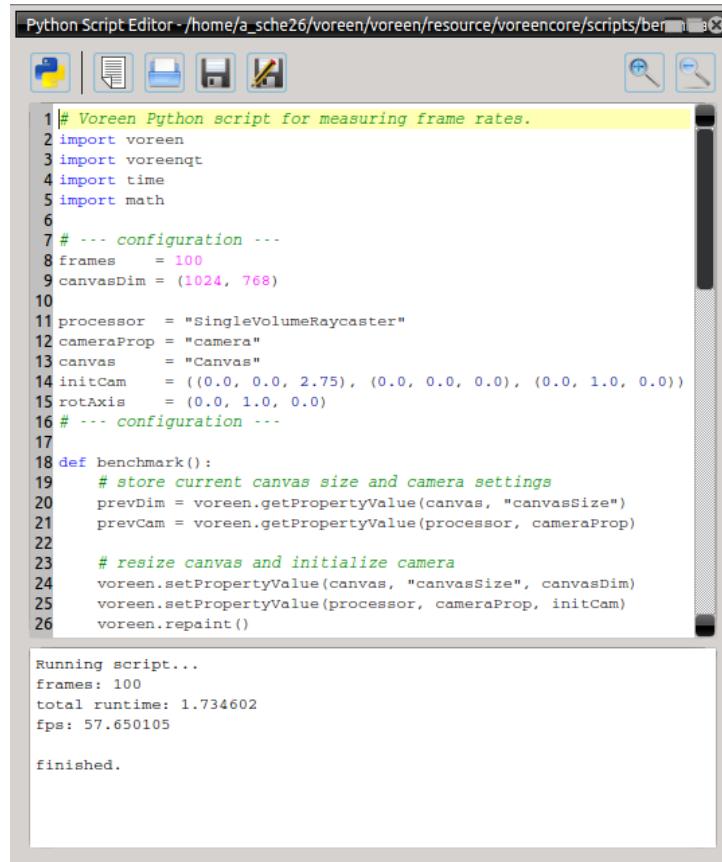
Animation

- 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



The screenshot shows a Python Script Editor window titled "Python Script Editor - /home/a_scher26/voreen/voreen/resource/voreencore/scripts/benchmark.py". The script content is as follows:

```
# Voreen Python script for measuring frame rates.
import voreen
import voreengt
import time
import math

# --- configuration ---
frames = 100
canvasDim = (1024, 768)

processor = "SingleVolumeRaycaster"
cameraProp = "camera"
canvas = "Canvas"
initCam = ((0.0, 0.0, 2.75), (0.0, 0.0, 0.0), (0.0, 1.0, 0.0))
rotAxis = (0.0, 1.0, 0.0)
# --- configuration ---

def benchmark():
    # store current canvas size and camera settings
    prevDim = voreen.getPropertyValue(canvas, "canvasSize")
    prevCam = voreen.getPropertyValue(processor, cameraProp)

    # resize canvas and initialize camera
    voreen.setPropertyValue(canvas, "canvasSize", canvasDim)
    voreen.setPropertyValue(processor, cameraProp, initCam)
    voreen.repaint()

Running script...
frames: 100
total runtime: 1.734602
fps: 57.650105

finished.
```

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
- *Fontrendering*
- *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.