Rapid Prototyping with Inviwo for Medical Applications

Martin Falk, Daniel Jönsson

Tutorial @VCBM September 2019

LINKÖPING UNIVERSITY

SeRC Swedish e-Science Research Centre

VCBM BRNO · CZECH REPUBLIC · 2019

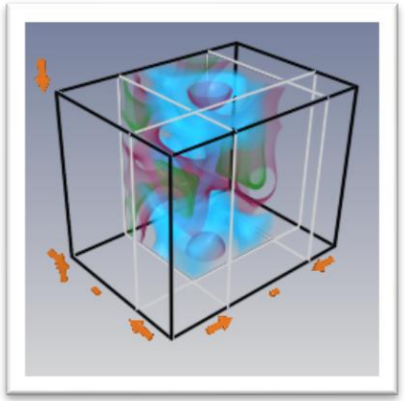
Inviwo

What can one do with Inviwo?

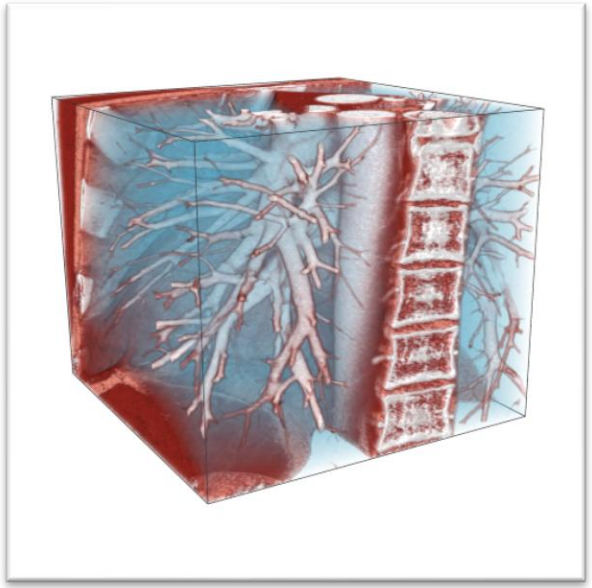
– Quick demonstration –

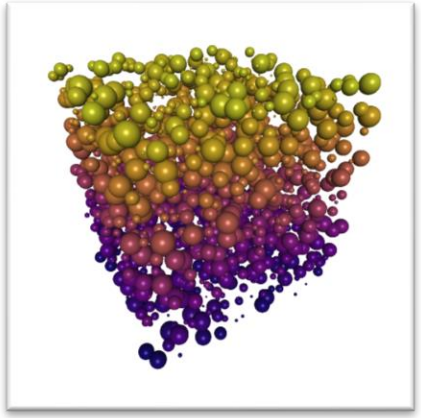
LINKÖPING UNIVERSITY

Outline



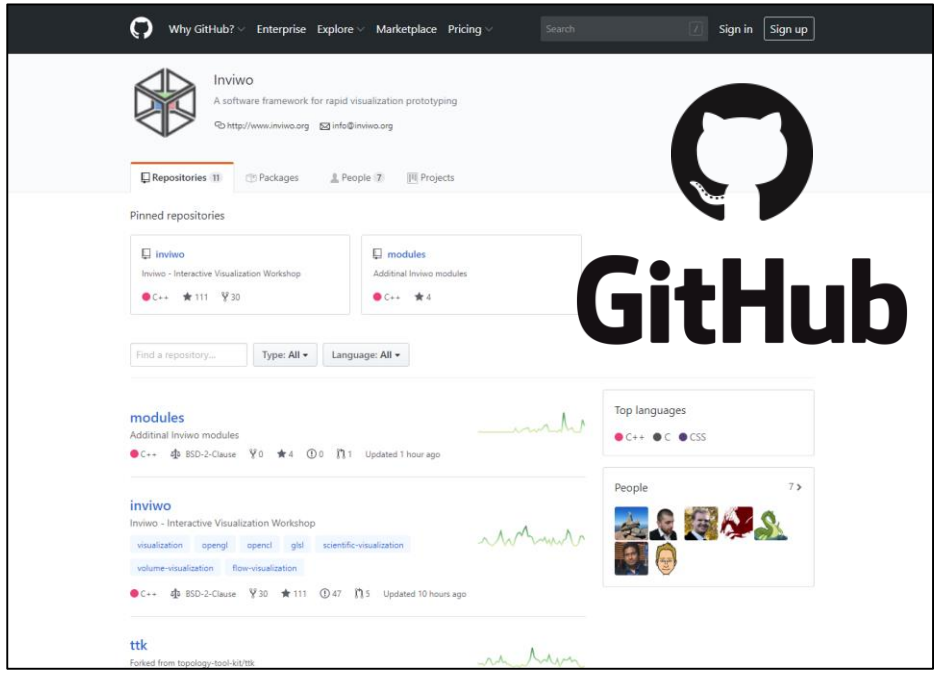
- Getting started
- Overview
- High-level abstractions
  - Visualization pipeline creation/editing
- Mid-level abstractions
  - Python
  - Web
- Low-level abstractions
  - Data handling
  - C++
- Application use cases





Getting started

- Recommended way: github.com/inviwo
- Just wanna try it out (binaries): inviwo.org



github.com/inviwo

inviwo.org

Building from source

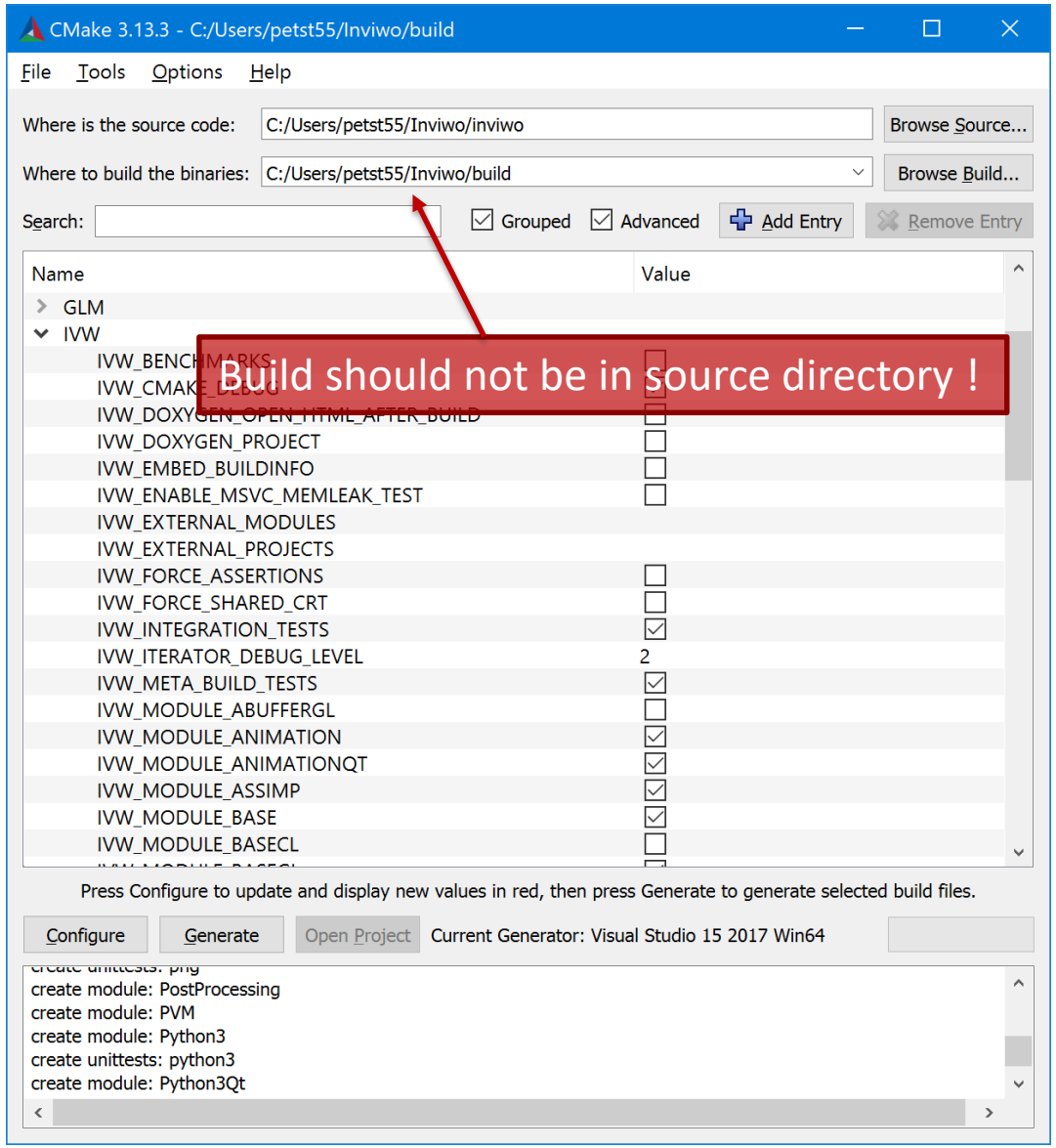
a. Setup software environment
    1. Git client (SourceTree, GitKraken…)
    2. CMake cmake.org/download/
    3. Qt www.qt.io/download
    4. C++17 compiler of your choice (MSVC/Clang/Gcc)
    5. Optional: Python 3.x
b. Get source code

```
git clone --recurse-submodules https://github.com/inviwo/inviwo.git
```

Building from source (cont.)

c. Run CMake
   1. Specify source/build directories
   2. Configure and generate project

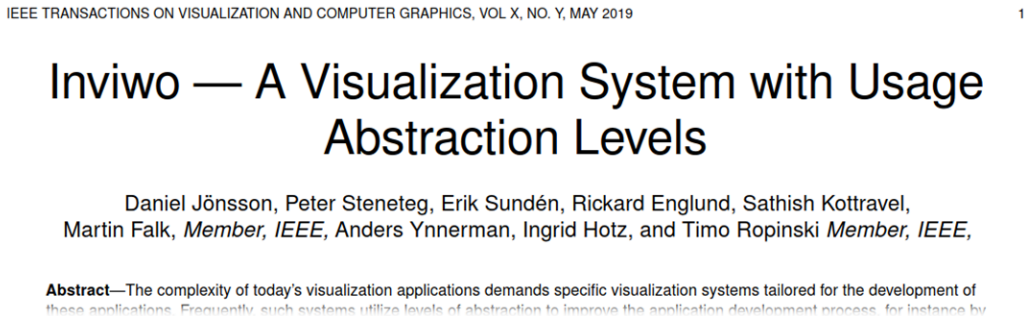
d. Open in chosen developer IDE
   1. Compile
   2. Run and make magic ☺



Overview

Overview

- Research software
- Developed by Visualization groups at
  - LiU, KTH, UULM
- Liberal license (Simplified BSD)
  - you can use it commercially



IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL X, NO. Y, MAY 2019 1

Inviwo — A Visualization System with Usage Abstraction Levels

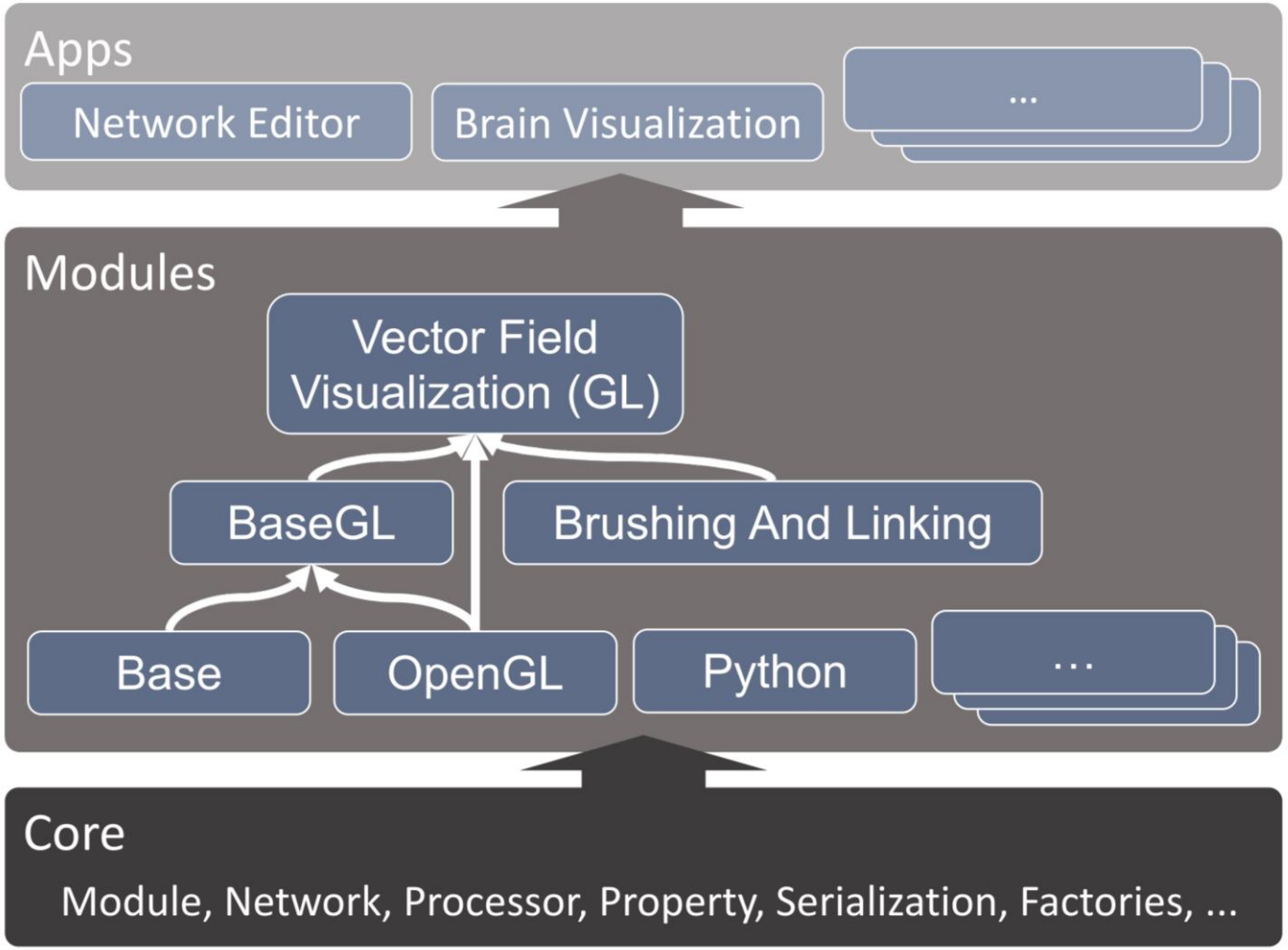
Daniel Jönsson, Peter Steneteg, Erik Sundén, Rickard Englund, Sathish Kottravel, Martin Falk, *Member, IEEE,* Anders Ynnerman, Ingrid Hotz, and Timo Ropinski *Member, IEEE,*

Abstract—The complexity of today's visualization applications demands specific visualization systems tailored for the development of

D. Jönsson, *et al.*, "Inviwo – A Visualization System with Usage Abstraction Levels" in *IEEE Transactions on Visualization & Computer Graphics*, 2019.
doi: 10.1109/TVCG.2019.2920639

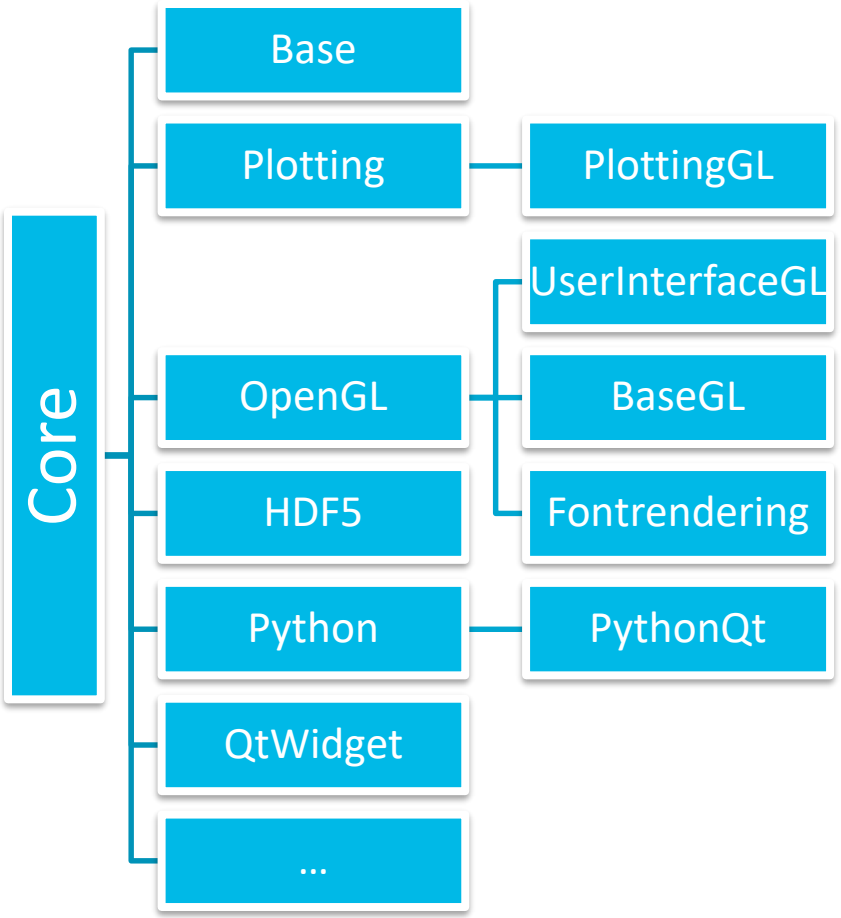
Architecture

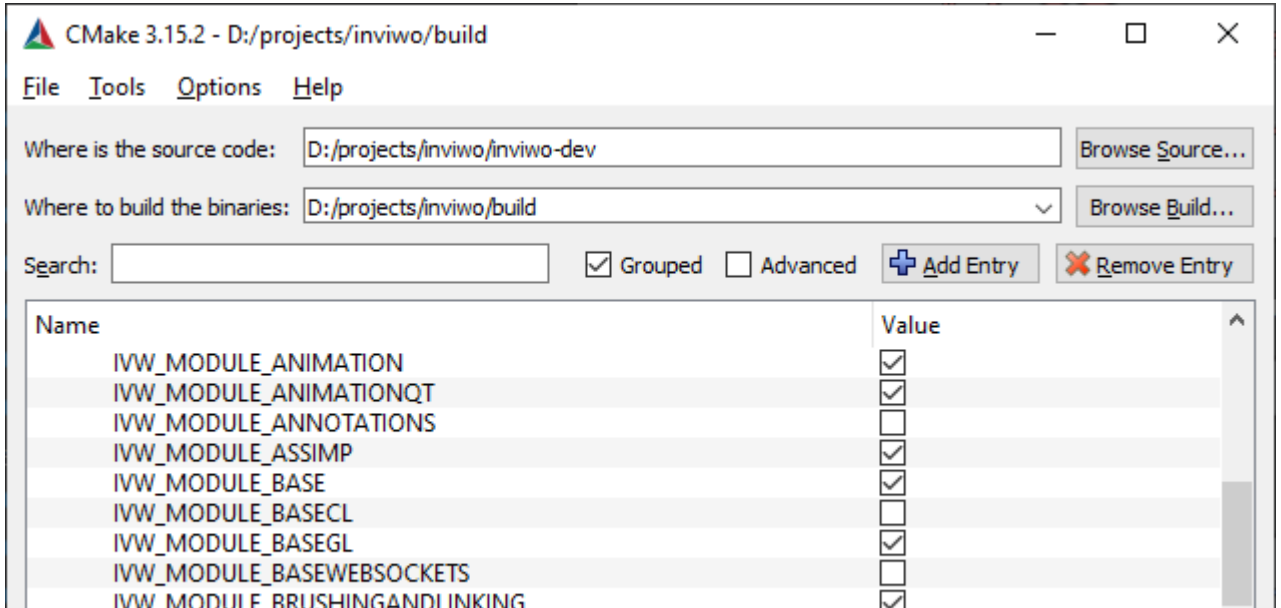
- Pure C++ Core
- Modular system (plugins)
- Multiple Apps



Module system (plugins)

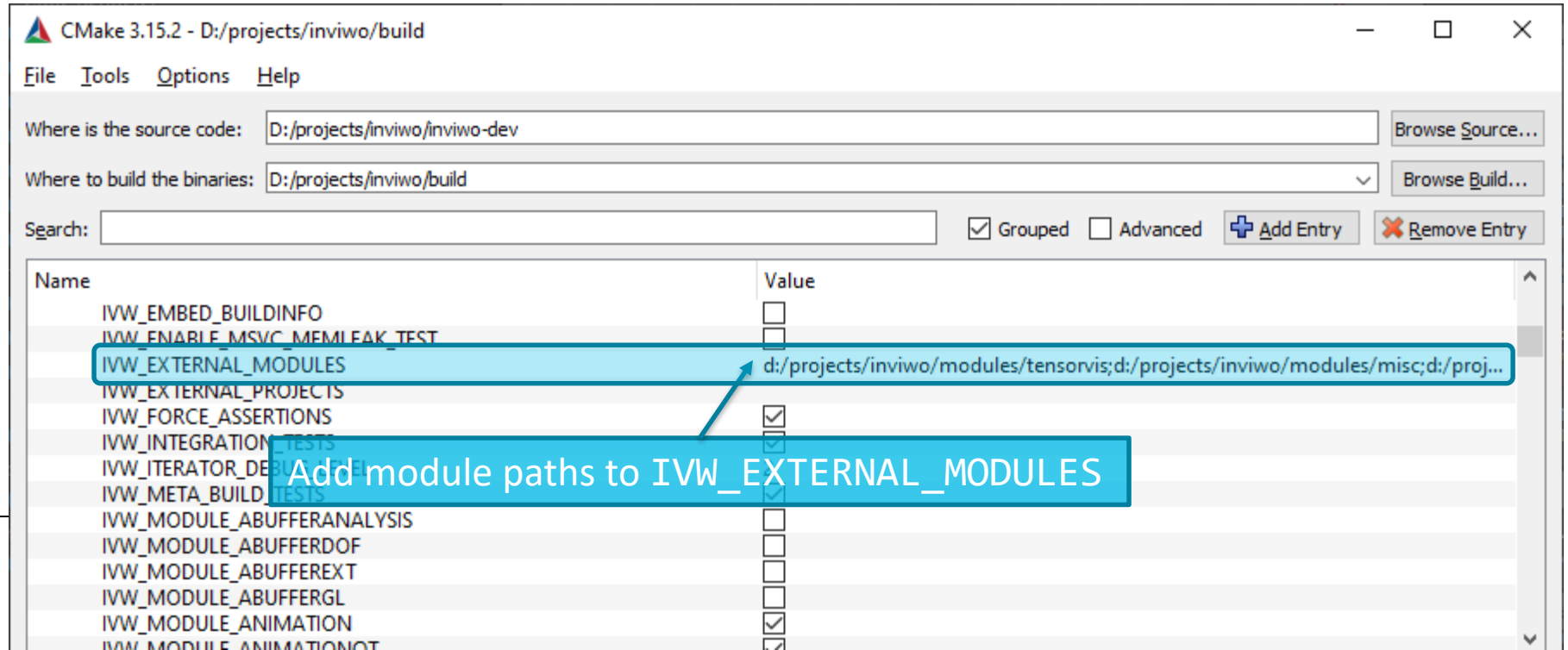
- Modules depend on Core and/or other modules
- Modules can wrap external libs
  - Data readers, OpenGL, …
- Enable them in CMake (`IVW_MODULE_...`)



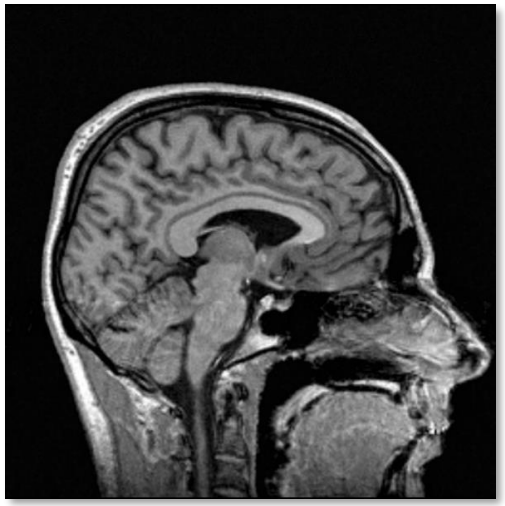


External modules – github.com/inviwo/modules

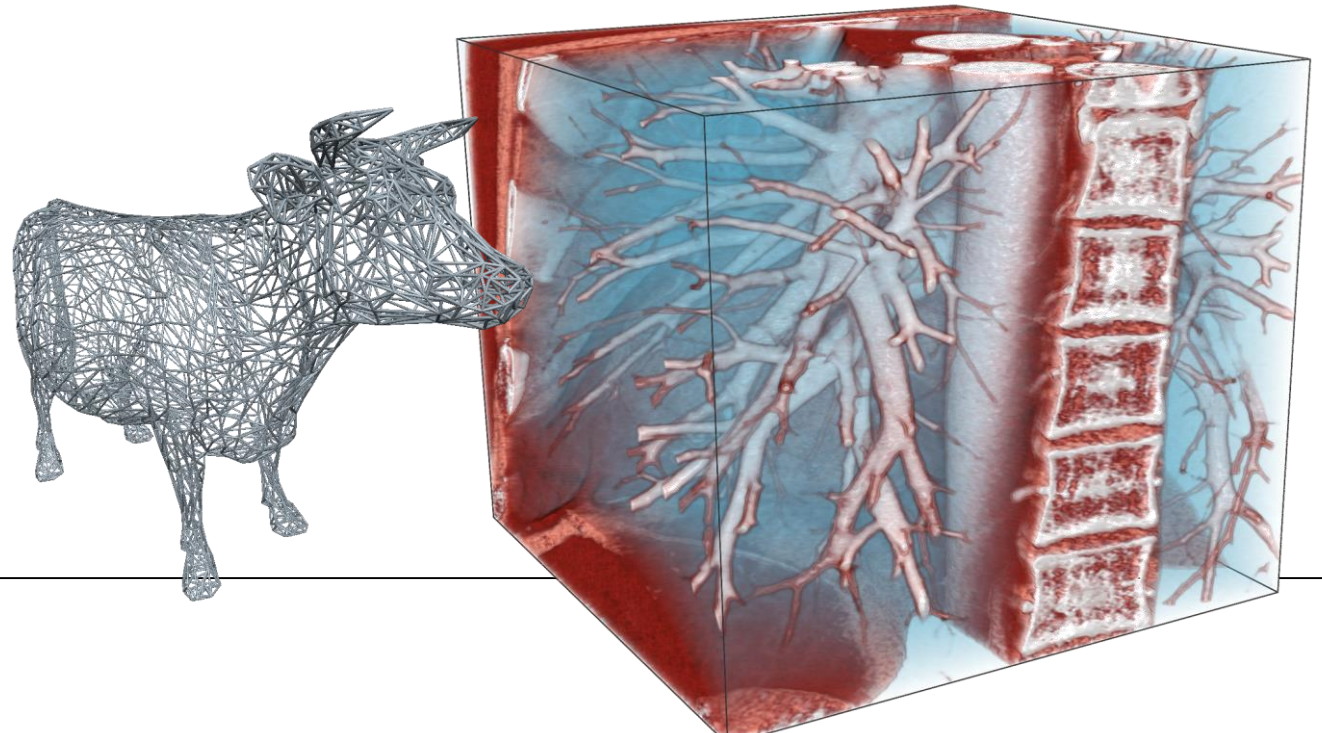
- Useful modules with external dependencies (e.g. boost)
  - OpenMesh
  - DICOM (Grassroots)
  - Topology Toolkit (TTK)
  - Tensor vis



Supported data formats in existing modules



- Imaging data
  - DICOM, H5, TIFF stack, RAW, nifty, pvm, …
- Mesh data via Assimp
  - Collada, 3Ds, fbx, obj, …
- Various image formats
  - Jpeg, TIFF, PNG, …



- Not enough?
  - Write your own!

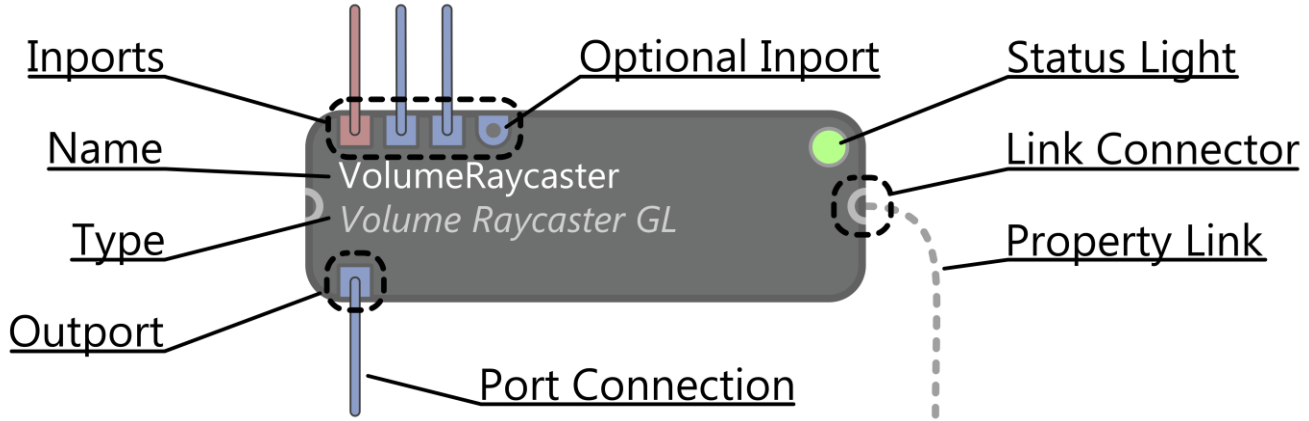
High-level abstractions

– The network editor

LINKÖPING UNIVERSITY

Processor

- Uses data from inports
- Operates on data
- Outputs result on outports

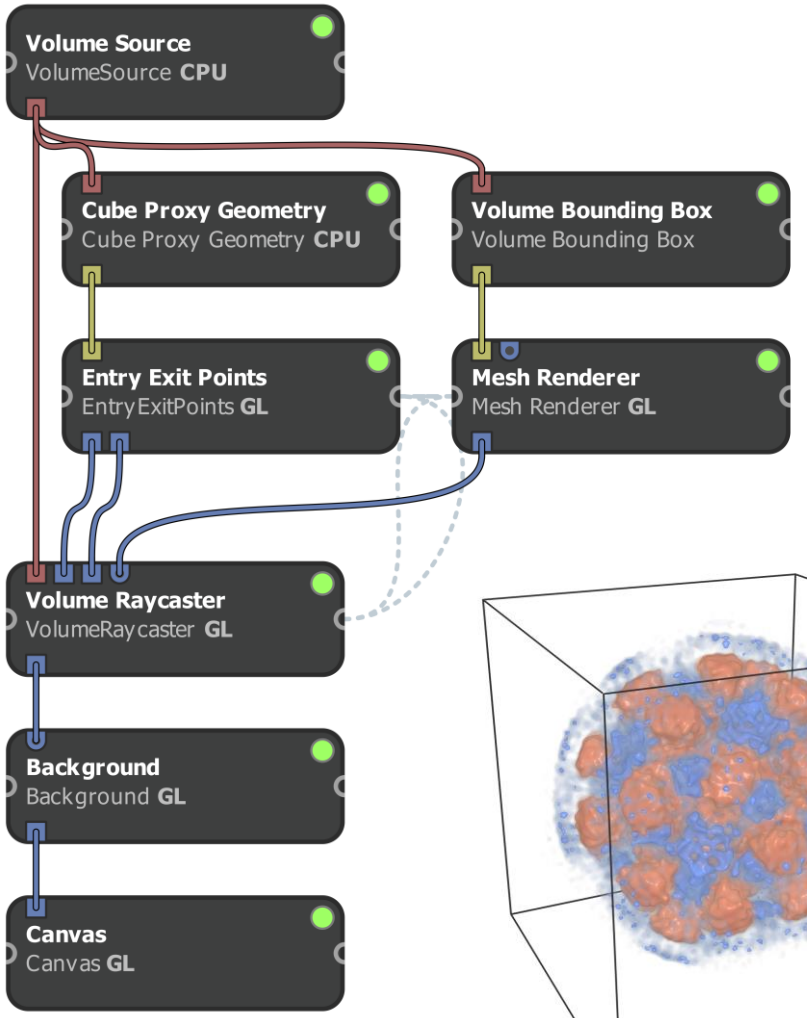


- Source Processor has no inports
- Sink Processor has no outports
- Evaluation starts at Source Processor and ends at Sink Processors

Visualization pipeline – processor network

- Models a data-flow graph
- Data enters at the top
- Result at the bottom
- “Stateless”

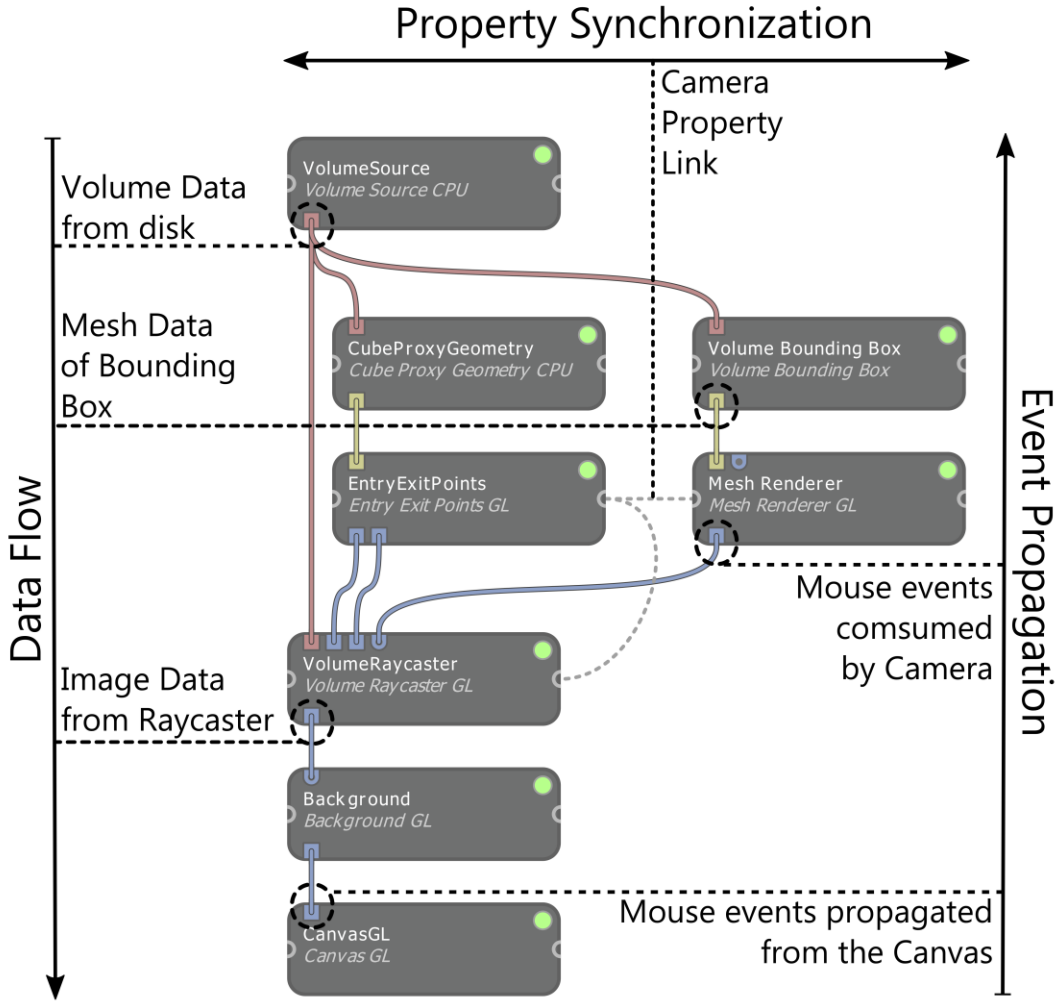
$$f_{p4}\left(g_{p3}\left(h_{p1}(input_1), k_{p2}(input_2)\right)\right)$$



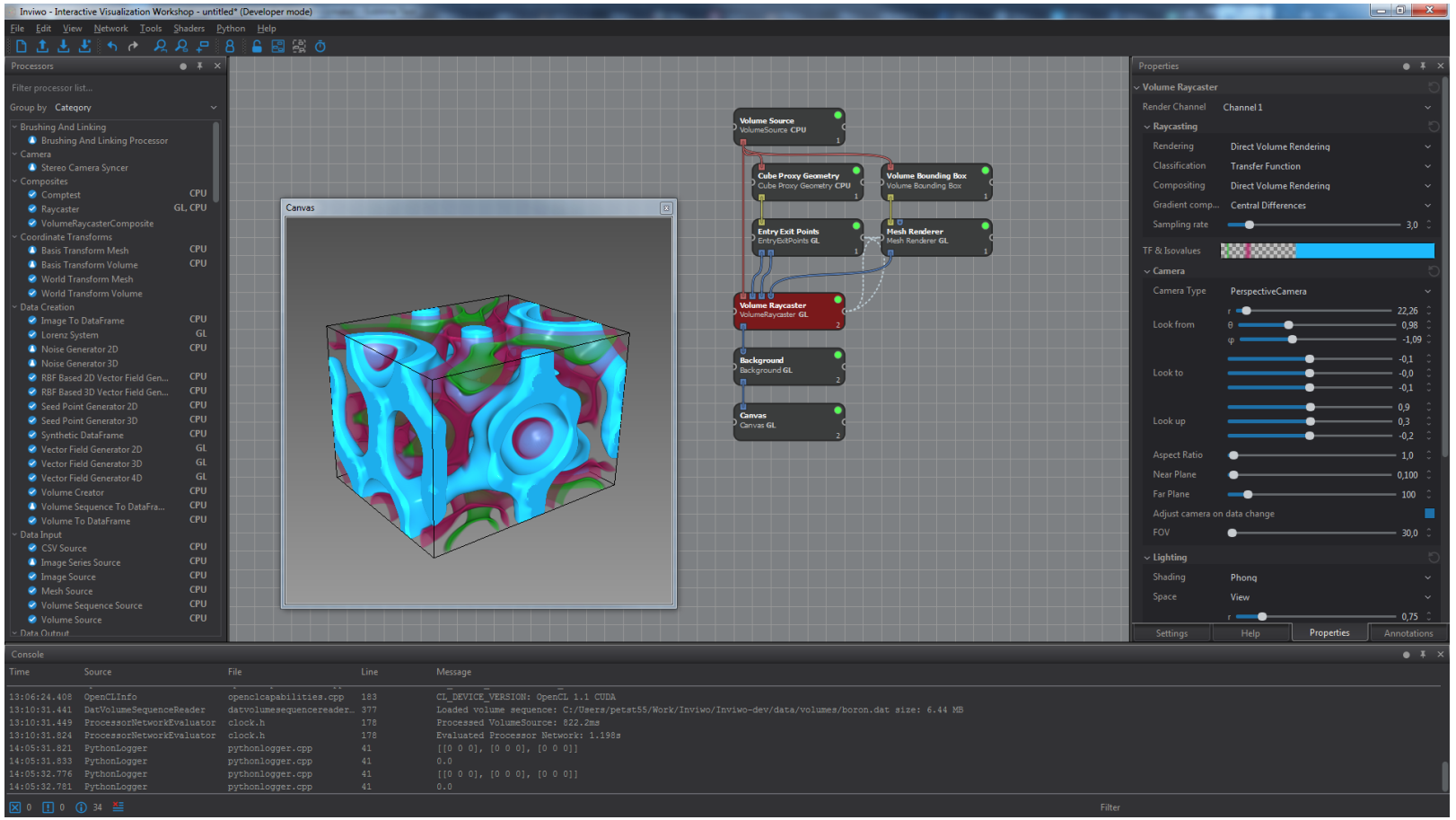
Feline calicivirus

Data/information flow

- Processors
  - Lazy evaluation
- Connections
  - Pass data downwards
  - Pass events upwards
- Links
  - Synchronize property states between processors



Network editor – Create/edit visualization pipelines



Inviwo - Interactive Visualization Workshop - untitled* (Developer mode)

File Edit View Network Tools Shaders Python Help

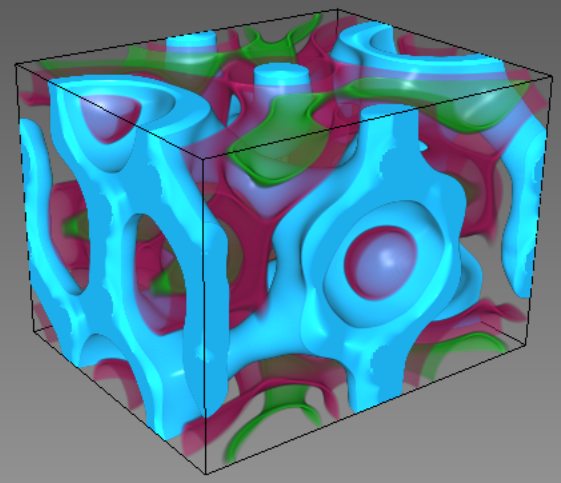
Processors

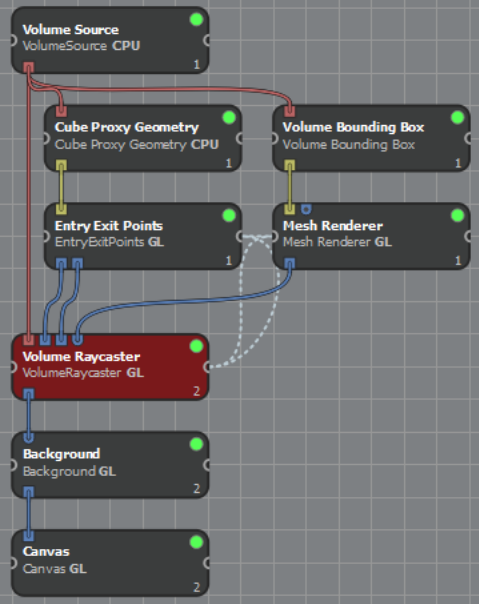
Filter processor list...

Group by Category

- Brushing And Linking
  - Brushing And Linking Processor
- Camera
  - Stereo Camera Syncer
- Composites
  - Comptest CPU
  - Raycaster GL, CPU
  - VolumeRaycasterComposite
- Coordinate Transforms
  - Basis Transform Mesh CPU
  - Basis Transform Volume CPU
  - World Transform Mesh
  - World Transform Volume
- Data Creation
  - Image To DataFrame CPU
  - Lorenz System GL
  - Noise Generator 2D CPU
  - Noise Generator 3D
  - RBF Based 2D Vector Field Gen... CPU
  - RBF Based 3D Vector Field Gen... CPU
  - Seed Point Generator 2D CPU
  - Seed Point Generator 3D CPU
  - Synthetic DataFrame CPU
  - Vector Field Generator 2D GL
  - Vector Field Generator 3D GL
  - Vector Field Generator 4D GL
  - Volume Creator CPU
  - Volume Sequence To DataFra... CPU
  - Volume To DataFrame CPU
- Data Input
  - CSV Source CPU
  - Image Series Source CPU
  - Image Source CPU
  - Mesh Source CPU
  - Volume Sequence Source CPU
  - Volume Source CPU
- Data Output

Canvas





Properties

Volume Raycaster

Render Channel: Channel 1

Raycasting

- Rendering: Direct Volume Rendering
- Classification: Transfer Function
- Compositing: Direct Volume Rendering
- Gradient comp...: Central Differences
- Sampling rate: 3,0
- TF & Isovalues

Camera

- Camera Type: PerspectiveCamera
- Look from: r 22,26; θ 0,98; φ -1,09
- Look to: -0,1; -0,0; -0,1
- Look up: 0,9; 0,3; -0,2
- Aspect Ratio: 1,0
- Near Plane: 0,100
- Far Plane: 100
- Adjust camera on data change
- FOV: 30,0

Lighting

- Shading: Phong
- Space: View
- r 0,75

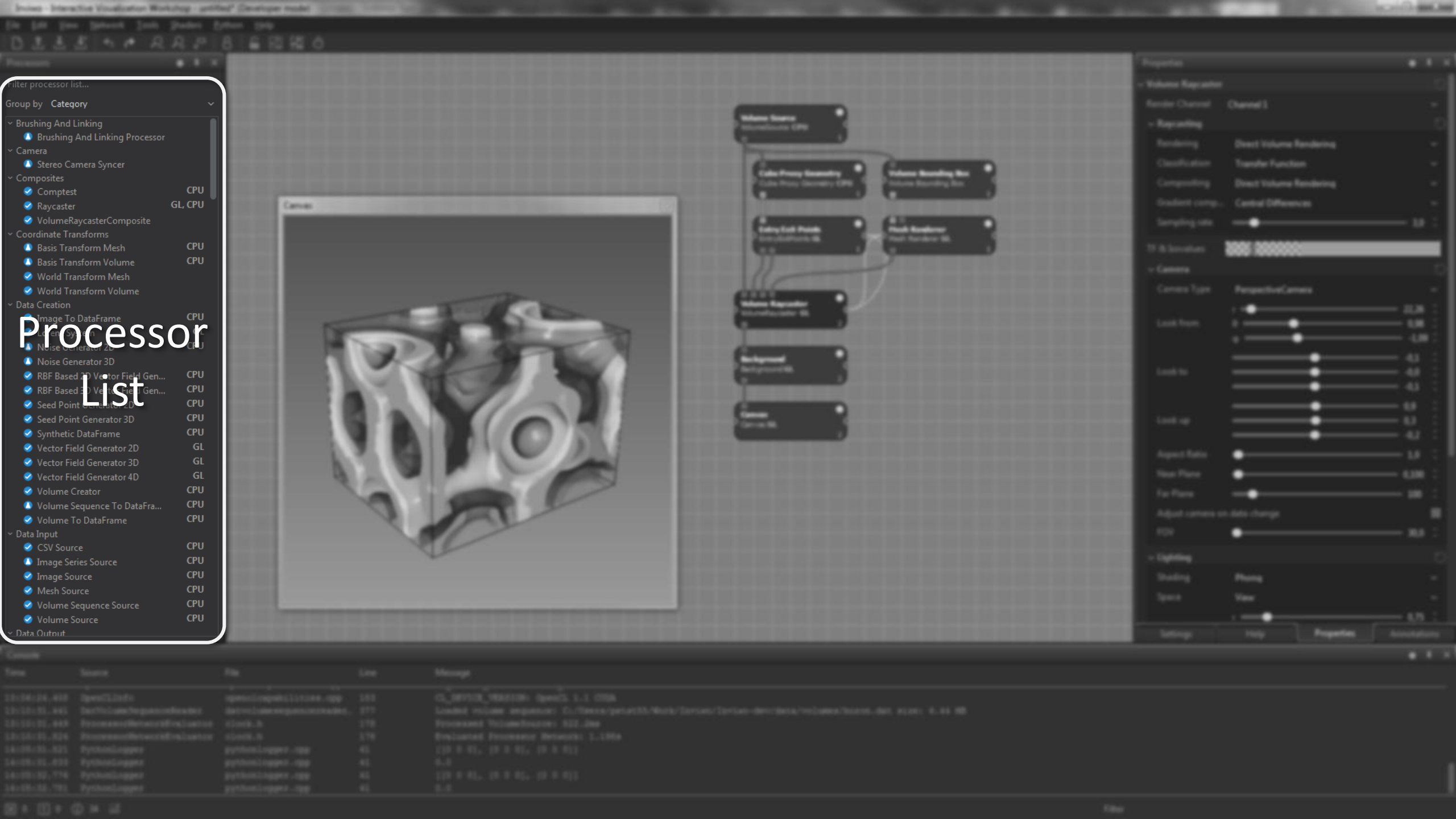
Settings Help Properties Annotations

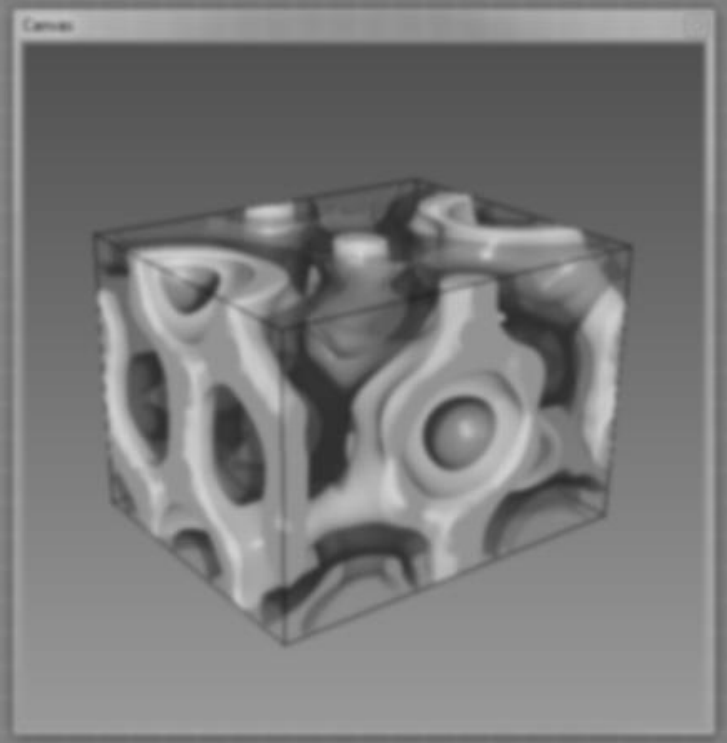
Console

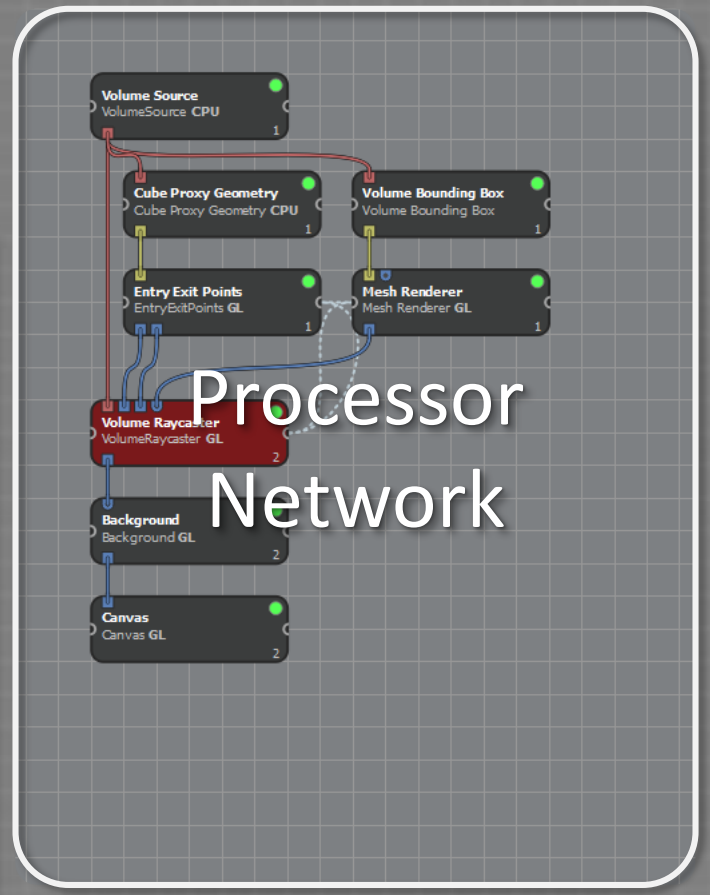
Time	Source	File	Line	Message
13:06:24.408	OpenCLInfo	openclcapabilities.cpp	183	CL_DEVICE_VERSION: OpenCL 1.1 CUDA
13:10:31.441	DatVolumeSequenceReader	datvolumesequencereader...	377	Loaded volume sequence: C:/Users/petst55/Work/Inviwo/Inviwo-dev/data/volumes/boron.dat size: 6.44 MB
13:10:31.449	ProcessorNetworkEvaluator	clock.h	178	Processed VolumeSource: 822.2ms
13:10:31.824	ProcessorNetworkEvaluator	clock.h	178	Evaluated Processor Network: 1.198s
14:05:31.821	PythonLogger	pythonlogger.cpp	41	[[0 0 0], [0 0 0], [0 0 0]]
14:05:31.833	PythonLogger	pythonlogger.cpp	41	0.0
14:05:32.776	PythonLogger	pythonlogger.cpp	41	[[0 0 0], [0 0 0], [0 0 0]]
14:05:32.781	PythonLogger	pythonlogger.cpp	41	0.0

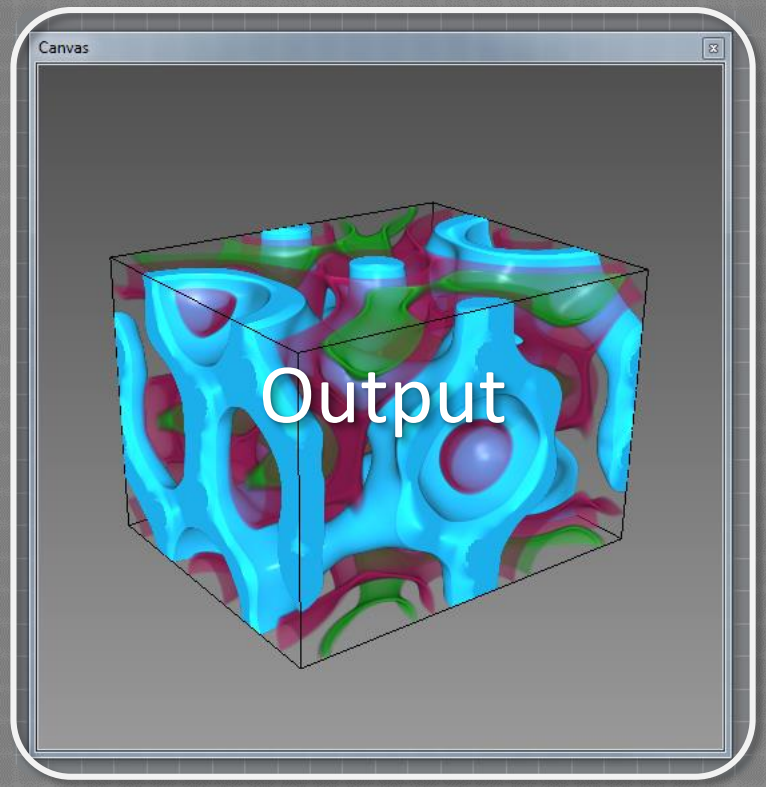
0 0 34

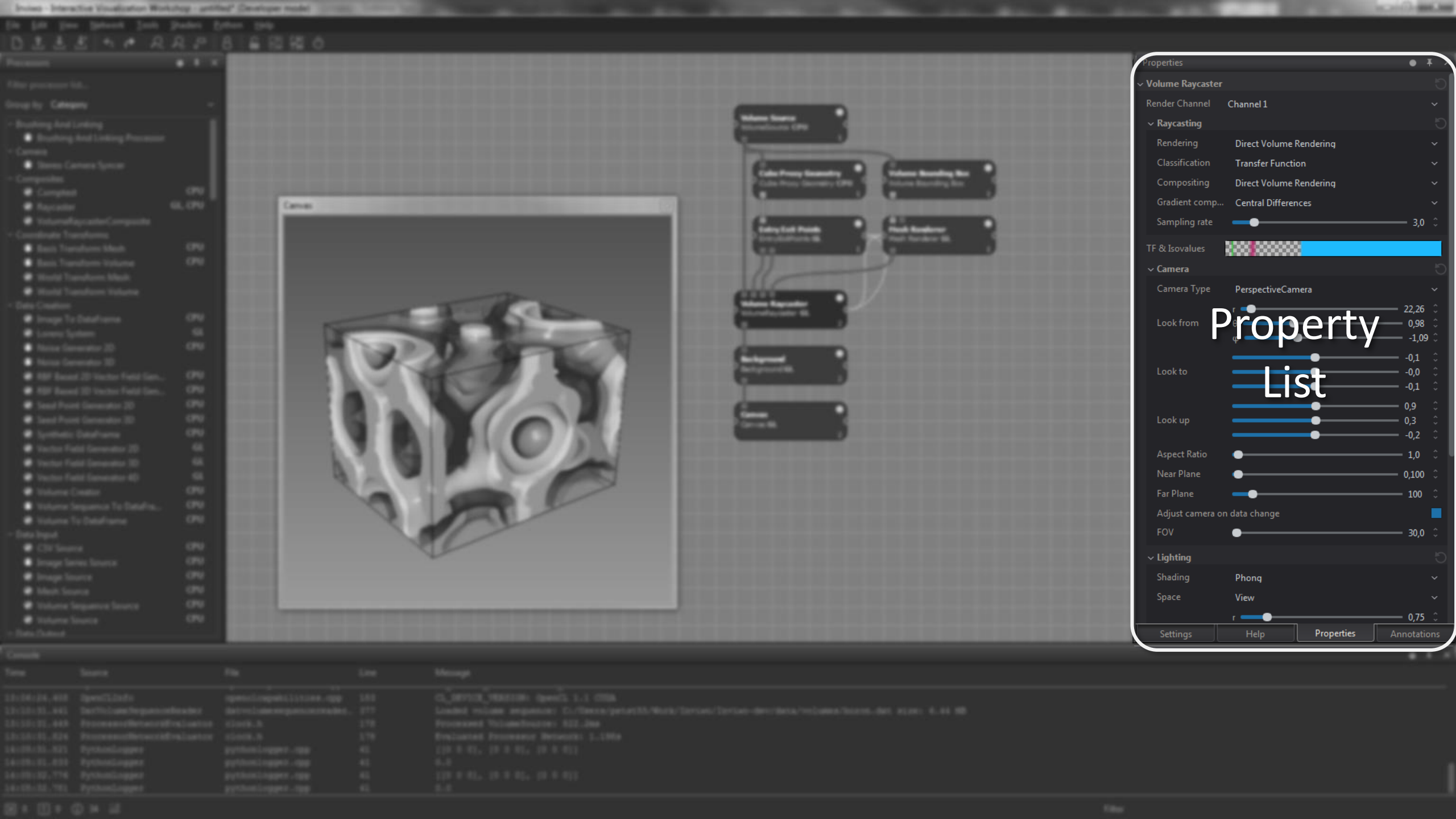
Filter

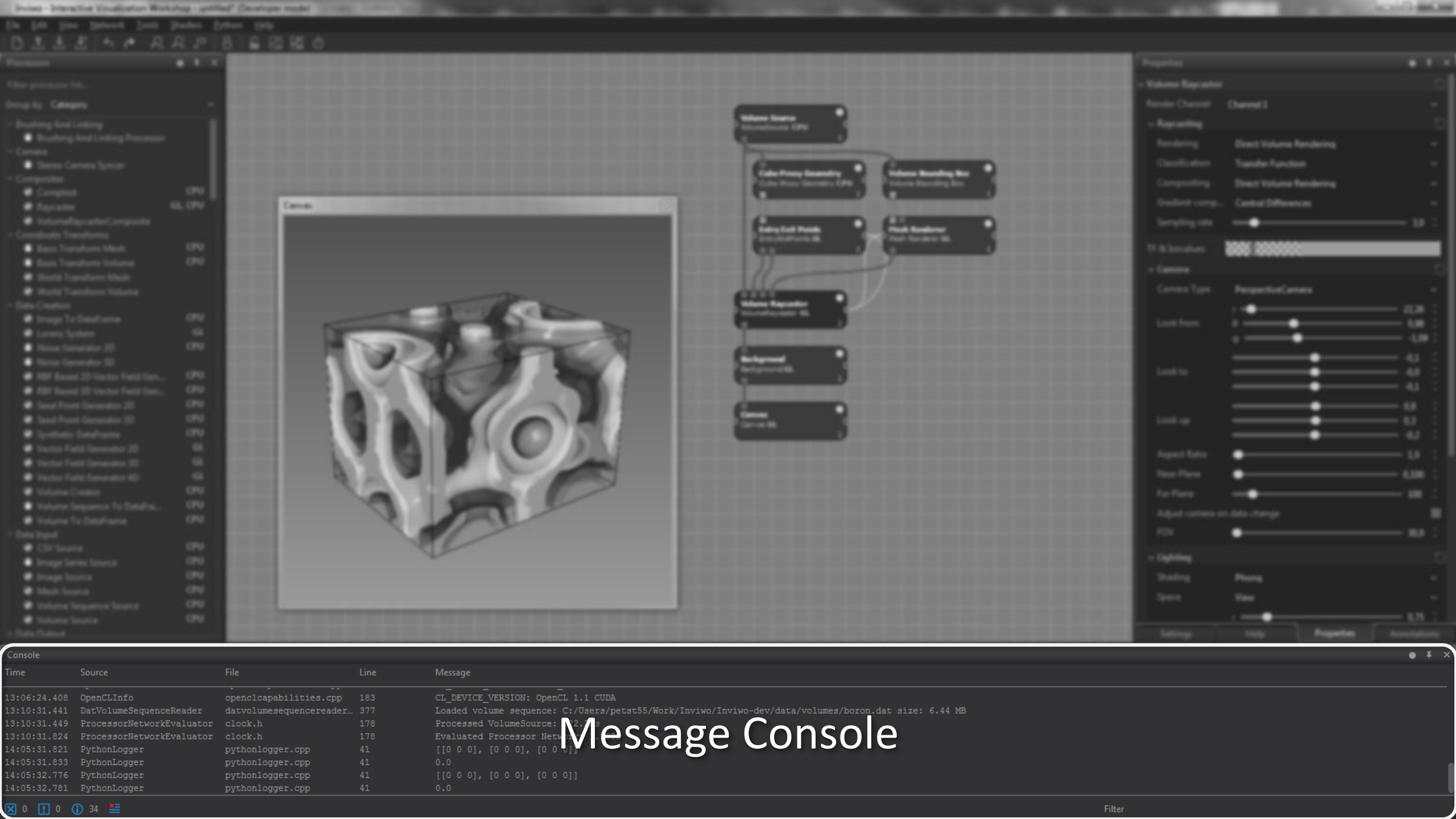

Filter processor list...
Group by Category
Brushing And Linking
Brushing And Linking Processor
Camera
Stereo Camera Syncer
Composites
Comptest CPU
Raycaster GL, CPU
VolumeRaycasterComposite
Coordinate Transforms
Basis Transform Mesh CPU
Basis Transform Volume CPU
World Transform Mesh
World Transform Volume
Data Creation
Image To DataFrame CPU
Noise Generator 3D
RBF Based 3D Vector Field Gen... CPU
Seed Point Generator 3D CPU
Synthetic DataFrame CPU
Vector Field Generator 2D GL
Vector Field Generator 3D GL
Vector Field Generator 4D GL
Volume Creator CPU
Volume Sequence To DataFra... CPU
Volume To DataFrame CPU
Data Input
CSV Source CPU
Image Series Source CPU
Image Source CPU
Mesh Source CPU
Volume Sequence Source CPU
Volume Source CPU
Data Output
Processor List




Volume Source
VolumeSource CPU
Cube Proxy Geometry
Cube Proxy Geometry CPU
Volume Bounding Box
Volume Bounding Box
Entry Exit Points
EntryExitPoints GL
Mesh Renderer
Mesh Renderer GL
Volume Raycaster
VolumeRaycaster GL
Background
Background GL
Canvas
Canvas GL
Processor Network

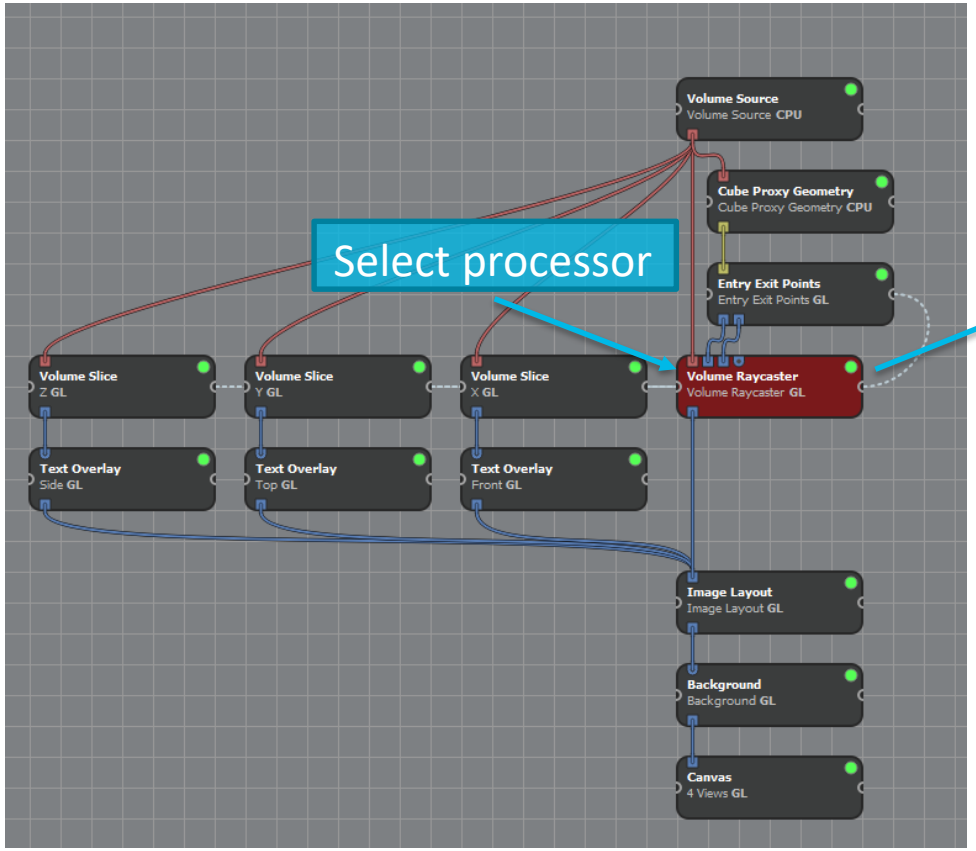

Canvas
Output

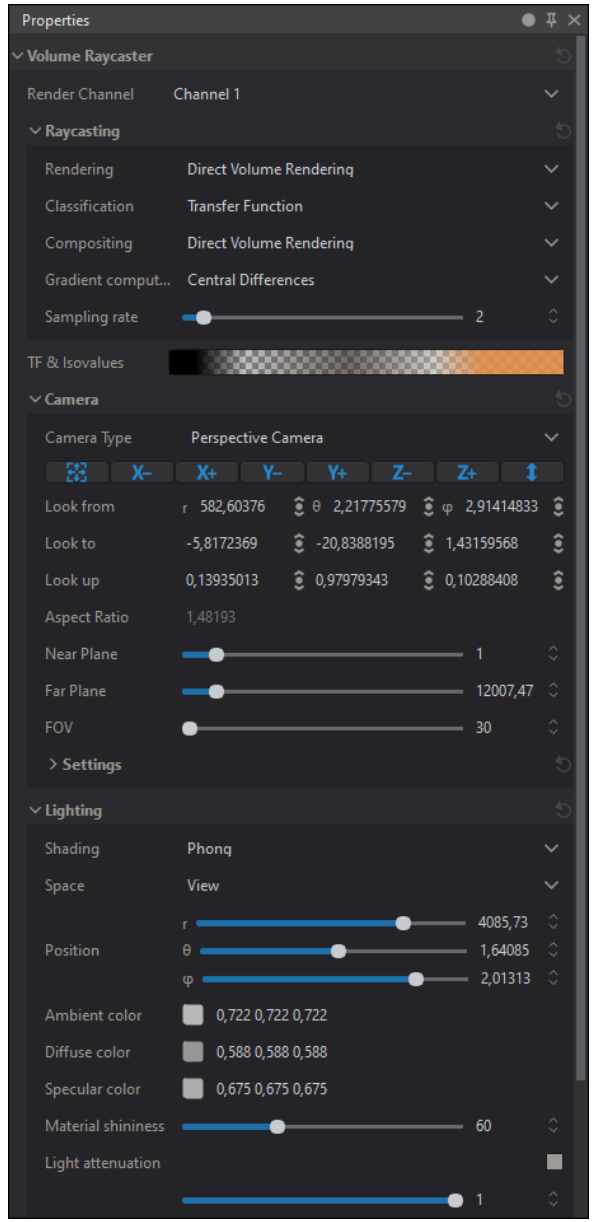

Property
List
Properties
Volume Raycaster
Render Channel
Channel 1
Raycasting
Rendering
Direct Volume Rendering
Classification
Transfer Function
Compositing
Direct Volume Rendering
Gradient comp...
Central Differences
Sampling rate
3,0
TF & Isovalues
Camera
Camera Type
PerspectiveCamera
Look from
22,26
0,98
-1,09
Look to
-0,1
-0,0
-0,1
Look up
0,9
0,3
-0,2
Aspect Ratio
1,0
Near Plane
0,100
Far Plane
100
Adjust camera on data change
FOV
30,0
Lighting
Shading
Phong
Space
View
0,75
Settings
Help
Properties
Annotations


Console
Time Source File Line Message
13:06:24.408 OpenCLInfo openclcapabilities.cpp 183 CL_DEVICE_VERSION: OpenCL 1.1 CUDA
13:10:31.441 DatVolumeSequenceReader datvolumesequencereader… 377 Loaded volume sequence: C:/Users/petst55/Work/Inviwo/Inviwo-dev/data/volumes/boron.dat size: 6.44 MB
13:10:31.449 ProcessorNetworkEvaluator clock.h 178 Processed VolumeSource:
13:10:31.824 ProcessorNetworkEvaluator clock.h 178 Evaluated Processor Netw
14:05:31.821 PythonLogger pythonlogger.cpp 41 [[0 0 0], [0 0 0], [0 0 0]]
14:05:31.833 PythonLogger pythonlogger.cpp 41 0.0
14:05:32.776 PythonLogger pythonlogger.cpp 41 [[0 0 0], [0 0 0], [0 0 0]]
14:05:32.781 PythonLogger pythonlogger.cpp 41 0.0
Filter
Message Console

Properties

- Configurable processor parameters

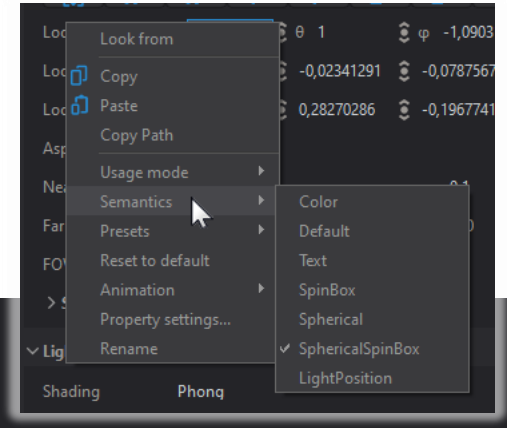




Property	Data types	Description
StringProperty	String	Text input
BoolProperty	Boolean	Checkbox
OptionProperty	enum, string, float/double, integer	List item selection
OrdinalProperty	float/double, integer, vector {2,3,4} , matrix{2,3,4}{2,3,4}	Bounded number input
MinMaxProperty	float/double, integer, vector {2,3,4}	Range with start-end values
ButtonProperty	-	Callback action on click
TransferFunctionProperty	TransferFunction/Iso-values	Map values to colors and opacity. Colormap
FileProperty DirectoryProperty FilePatternProperty	string(s)	File/folder paths. Also with pattern matching (image###.*)
EventProperty	Event	Map user input to callbacks
ListProperty	Any property type	User can add/remove properties dynamically
CameraProperty	Camera (position, direction…)	Camera interaction
CompositeProperty		Group of properties

Property Semantics

Accessible via context menu of the property

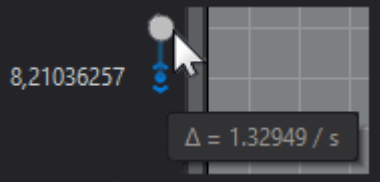


Ordinal double property

Default

Text

Spinbox

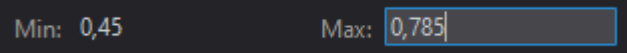


Angle

Min max property

Default

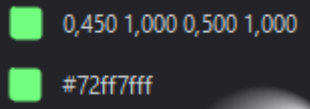
Text



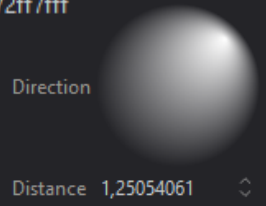
Ordinal vec4 property

Spinbox

Color



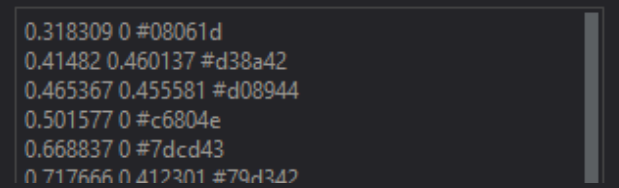
Position



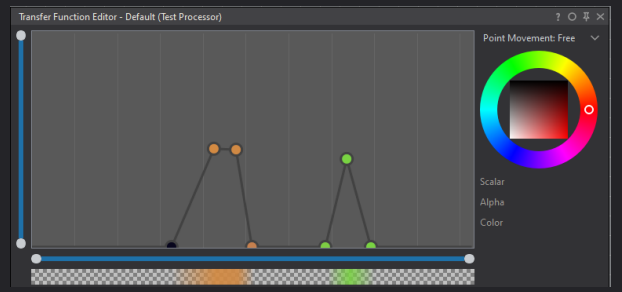
Transfer function property

Default

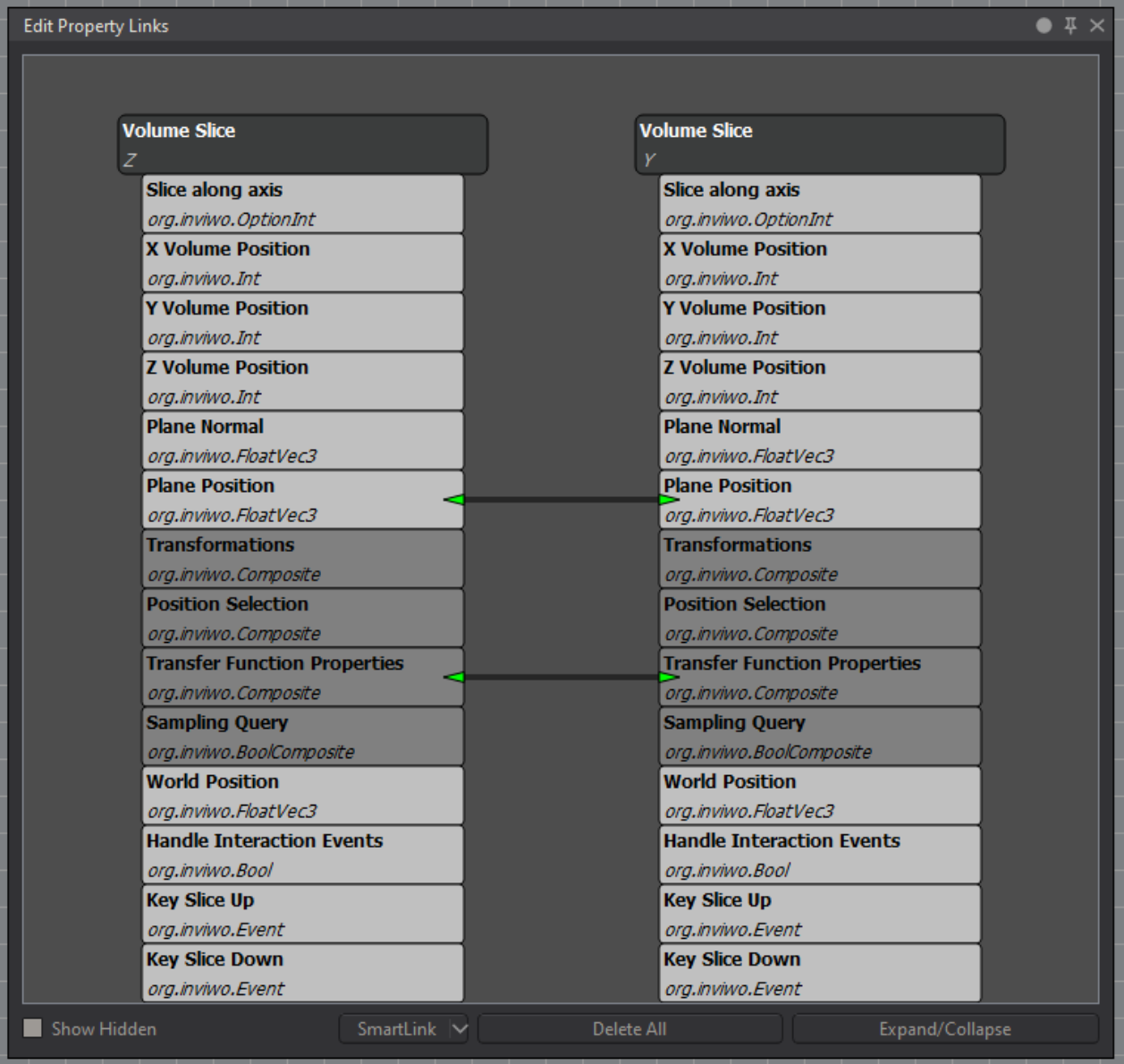
Text

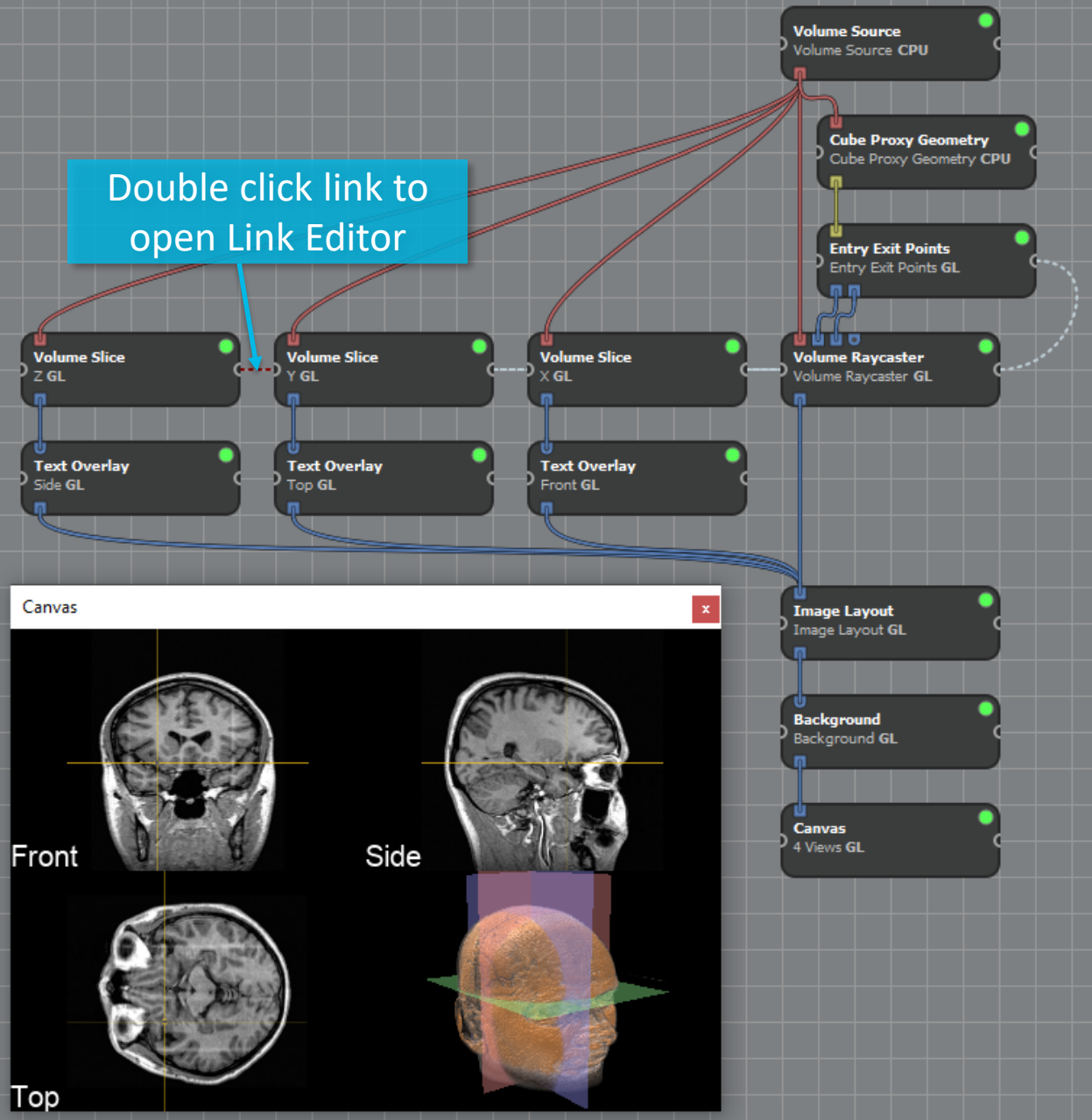


Widget



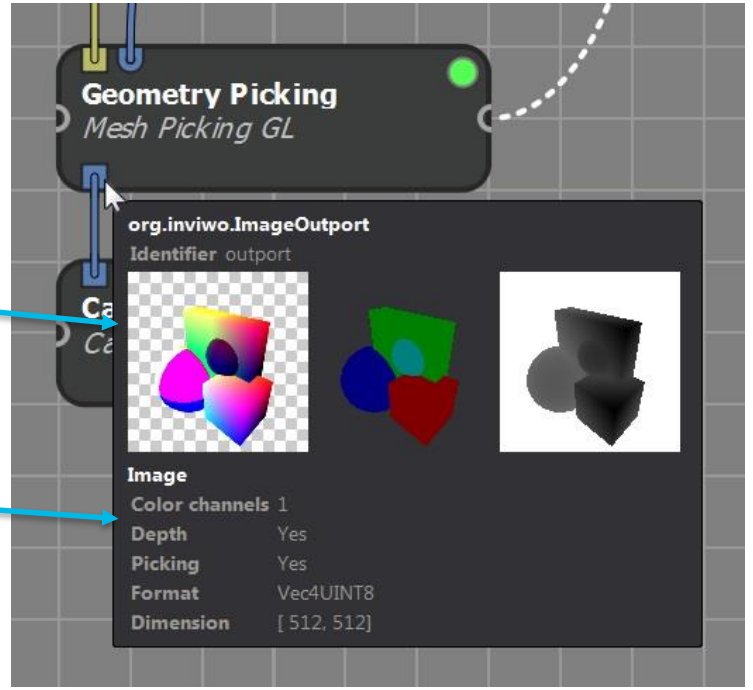
Property Linking





Visual debugging

- Concept: **Port inspector**
  - Is an Inviwo workspace
- Concept: **Port trait**
  - Programmatically show data attributes
  - Defines port color
- Easily add one for your port type
  - yourmodule/data/workspaces/portinspectors
  - Register in module constructor




```
registerPortInspector(PortTraits<ImageOutport>::classIdentifier(),
                      app->getPath(PathType::PortInspectors,
                      "/imageportinspector.inv"));
```

Mid-level abstractions

– Python and Web integration

LINKÖPING UNIVERSITY

Python

- Network-level scripting
  - Built-in Python editor for running scripts
- Processor-level
  - Interactive prototyping via PythonProcessor
    - Processor from a python script
- Python package
  - Run Inviwo from within python

- Integrated data handling
  - volume/image/layer/buffer <-> numpy

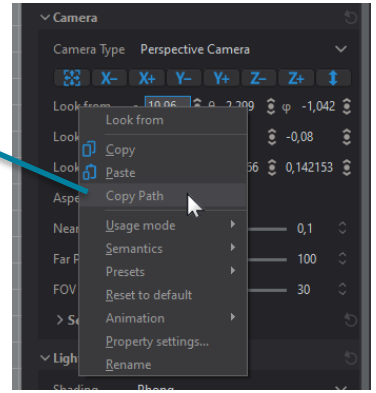
Python Editor

Python Editor - D:/projects/inviwo/inviwo-dev/data/scripts/canvassnapshot.py*

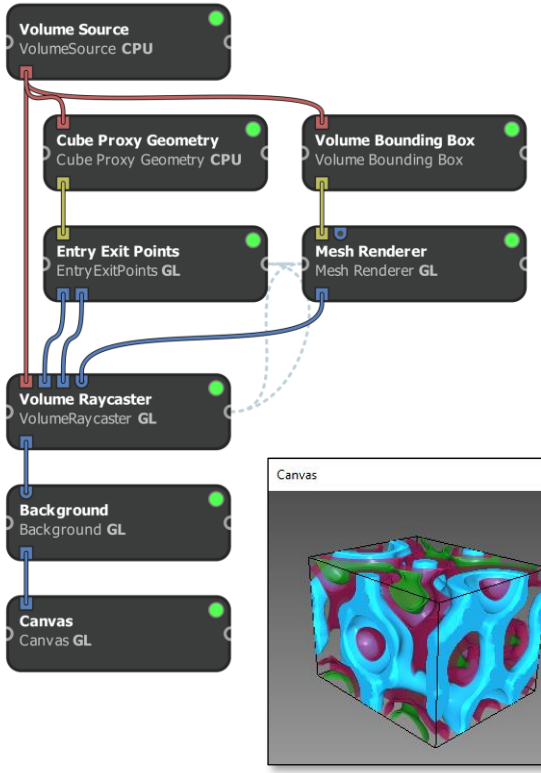
```
1  # Inviwo Python script
2  import inviwopy
3
4  network = inviwopy.app.network
5
6  # change camera position
7  network.VolumeRaycaster.camera.lookFrom = inviwopy.glm.vec3(0, 0, -20)
8
9  # adjust a transfer function
10 tf = network.VolumeRaycaster.isotfComposite.tf
11 for p in tf.value:
12     print(p)
13     p.alpha = min(p.alpha + 0.2, 1)
14
15 # take a snapshot
16 network.Canvas.snapshot("snapshot.png")
17
```

```
1 <TFPrimitive: 0.0277578, #00000000>
2 <TFPrimitive: 0.0319476, #1bc92468>
3 <TFPrimitive: 0.0416367, #09090900>
4 <TFPrimitive: 0.107889, #0e0e0e00>
5 <TFPrimitive: 0.127136, #ba1a657d>
6 <TFPrimitive: 0.14599, #16161600>
7 <TFPrimitive: 0.343699, #1d1d1d00>
8 <TFPrimitive: 0.358112, #1ebafdff>
```

Executed Successfully (153.933 ms)



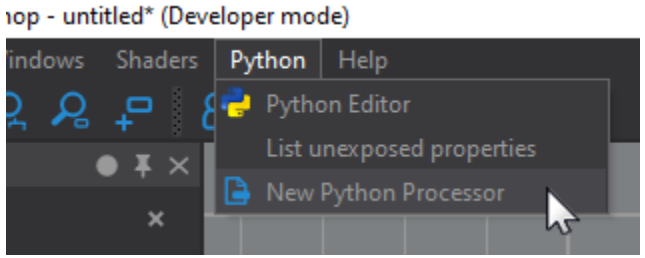
Access property path via context menu



More examples in inviwo/data/scripts/

Python processor

- Utilizing everything python offers
  - data import, parsing, …
- Skeleton processor via menu
  - Ports and properties can be added

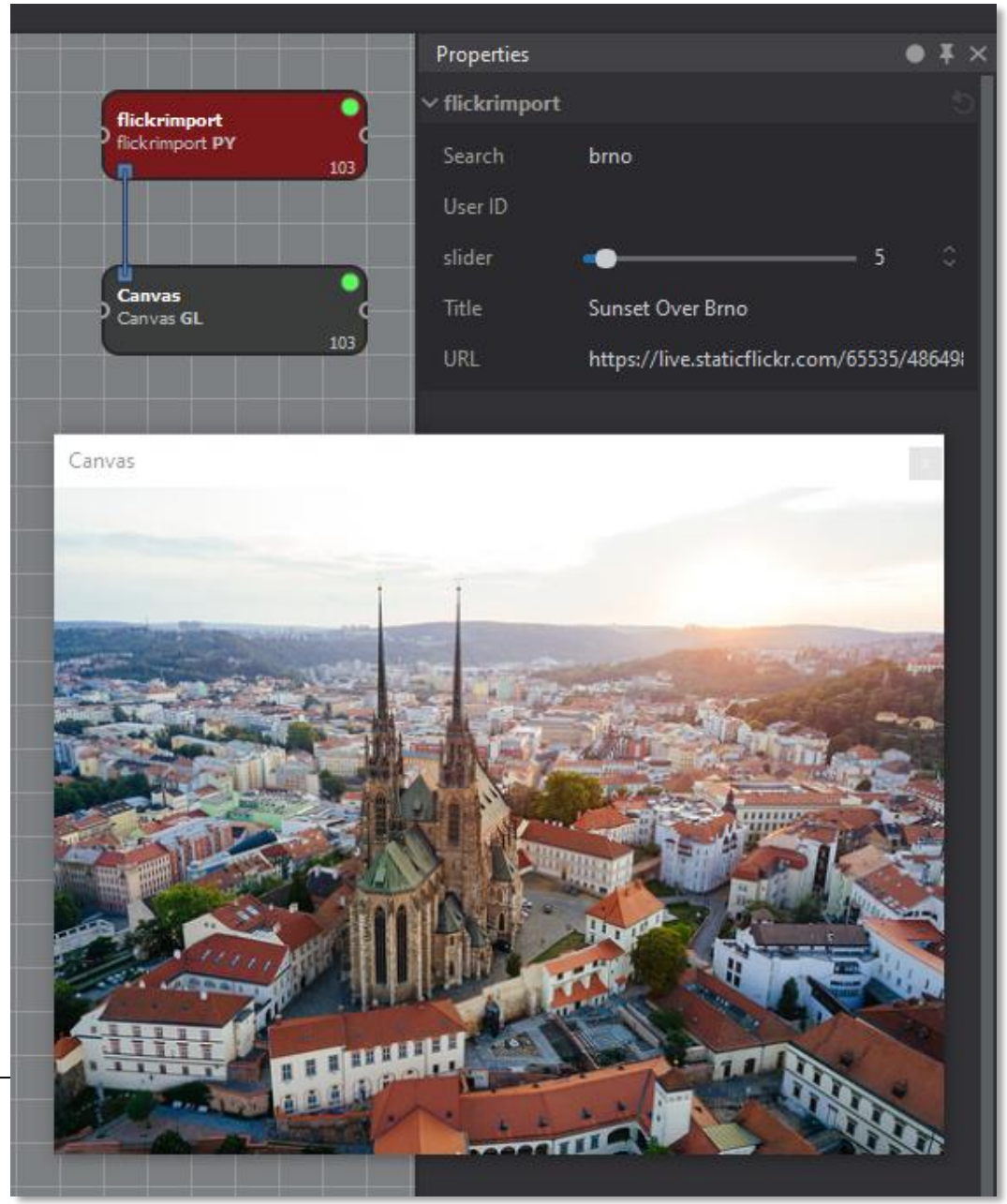



```python
# Name: PythonTestProcessor
import inviwopy as ivw

class PythonTestProcessor(ivw.Processor):
    def __init__(self, id, name):
        ivw.Processor.__init__(self, id, name)
        self.inport = ivw.data.ImageInport("inport")
        self.addInport(self.inport, owner=False)
        self.outport = ivw.data.ImageOutport("outport")
        self.addOutport(self.outport, owner=False)

        self.slider = ivw.properties.IntProperty("slider", "slider", 0, 0
        self.addProperty(self.slider, owner=False)

    @staticmethod
    def processorInfo():
        return ivw.ProcessorInfo(
            classIdentifier = "org.inviwo.pythontestprocessor",
            displayName = "PythonTestProcessor",
            category = "Python",
            codeState = ivw.CodeState.Stable,
            tags = ivw.Tags.PY
        )

    def getProcessorInfo(self):
        return test.processorInfo()

    def initializeResources(self):
        print("init")

    def process(self):
        print("process: ", self.slider.value)
        self.outport.setData(self.inport.getData())
```

Python processor demo

Python processor – flickr_api

```
# Name: FlickrImportMinimal

import inviwopy as ivw

import numpy as np
import flickr_api

import requests
from PIL import Image

class FlickrImportMinimal(ivw.Processor):
    def __init__(self, id, name):
        # flickr authentication (see flickr_api docs)

        flickr_api.enable_cache()
        self.photos = []
        ...
```

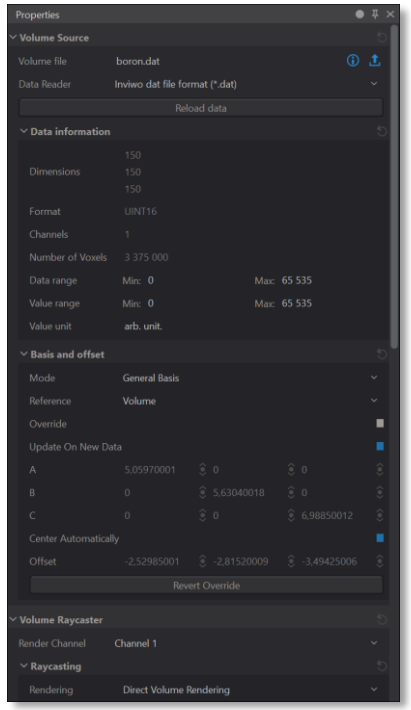


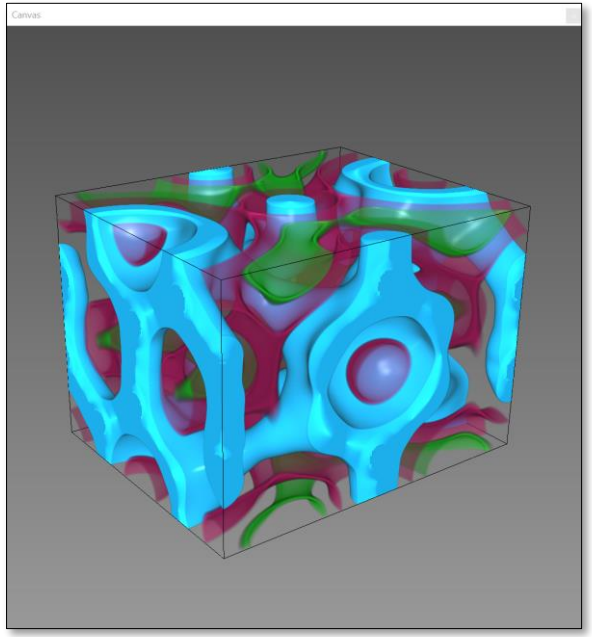
Python processor – flickr_api (cont.)

```
...
def initializeResources(self):
    self.photos = []
    w = flickr_api.Walker(flickr_api.Photo.search, tags="brno", media="photos")
    for i,photo in enumerate(w):
        self.photos.append({'url' : photo.getPhotoFile('Medium'), 'title' : photo.title})

    self.slider.maxValue = len(self.photos) - 1

def process(self):
    r = requests.get(self.photos[self.slider.value]['url'], stream=True)
    r.raw.decode_content = True

    img = Image.open(r.raw).transpose(Image.FLIP_TOP_BOTTOM)
    npImg = np.array(img).transpose((1, 0, 2))

    image = ivw.data.Image(ivw.data.Layer(npImg))

    self.outport.setData(image)
```

Python package – run Inviwo in Python

```
import inviwopy as ivw
import inviwopyapp as qt

if __name__ == '__main__':
    # Inviwo requires a logCentral
    lc = ivw.LogCentral()
    # create and register a console logger
    cl = ivw.ConsoleLogger()
    lc.registerLogger(cl)

    # create the inviwo application
    app = qt.InviwoApplicationQt()
    app.registerModules()

    # load a workspace
    app.network.load(app.getPath(ivw.PathType.Workspaces) + "/boron.inv")

    # start the event loop
    app.run()
```

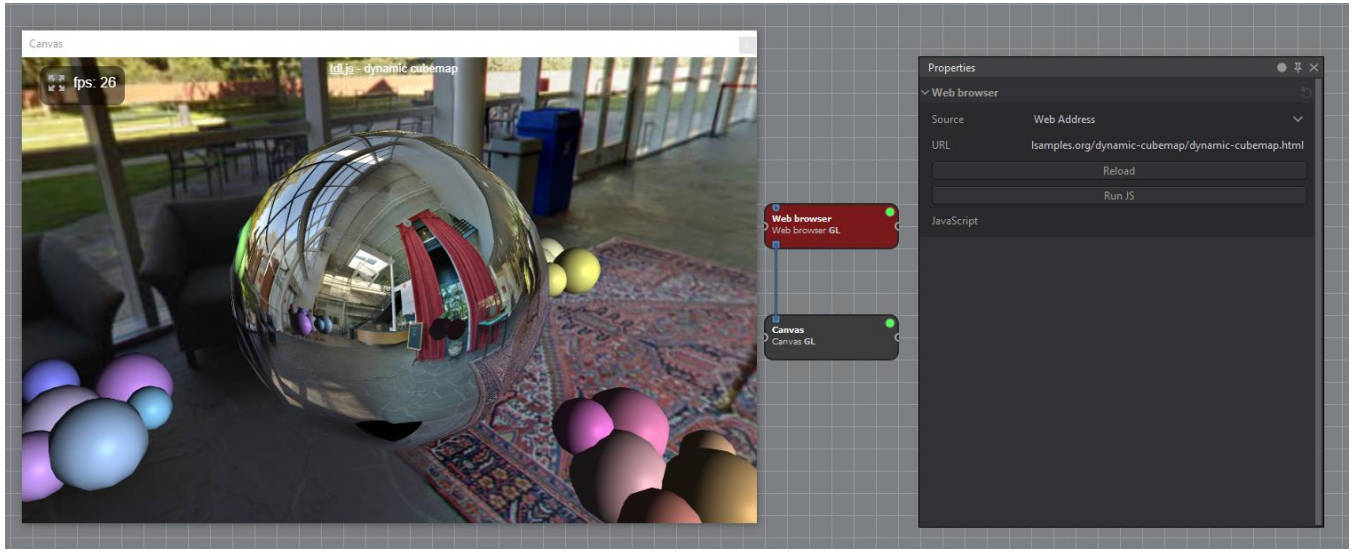


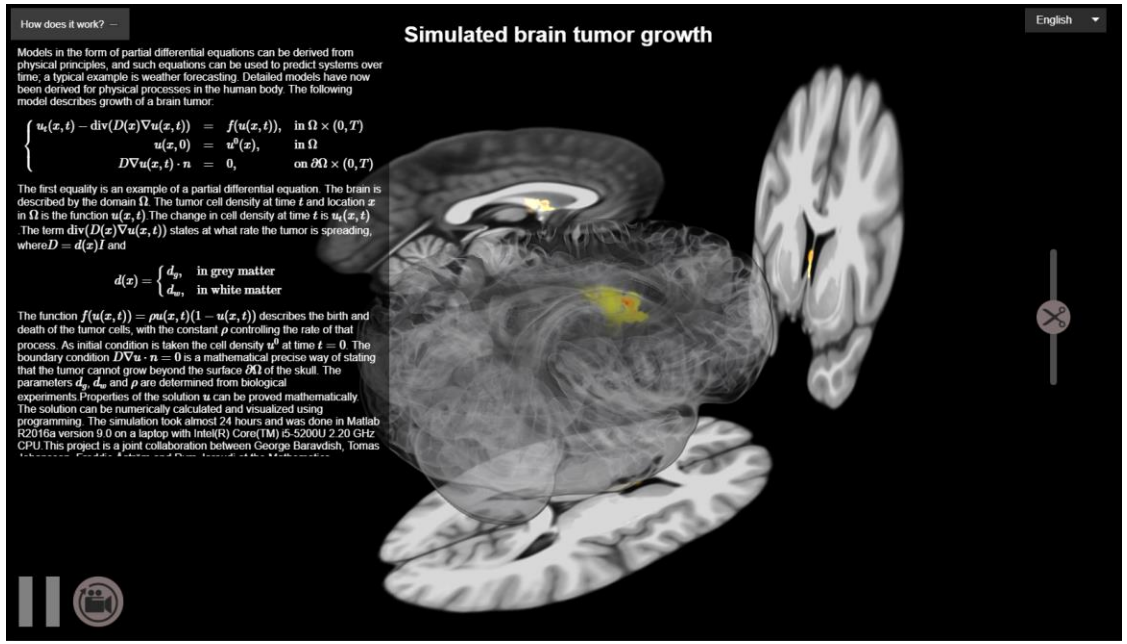


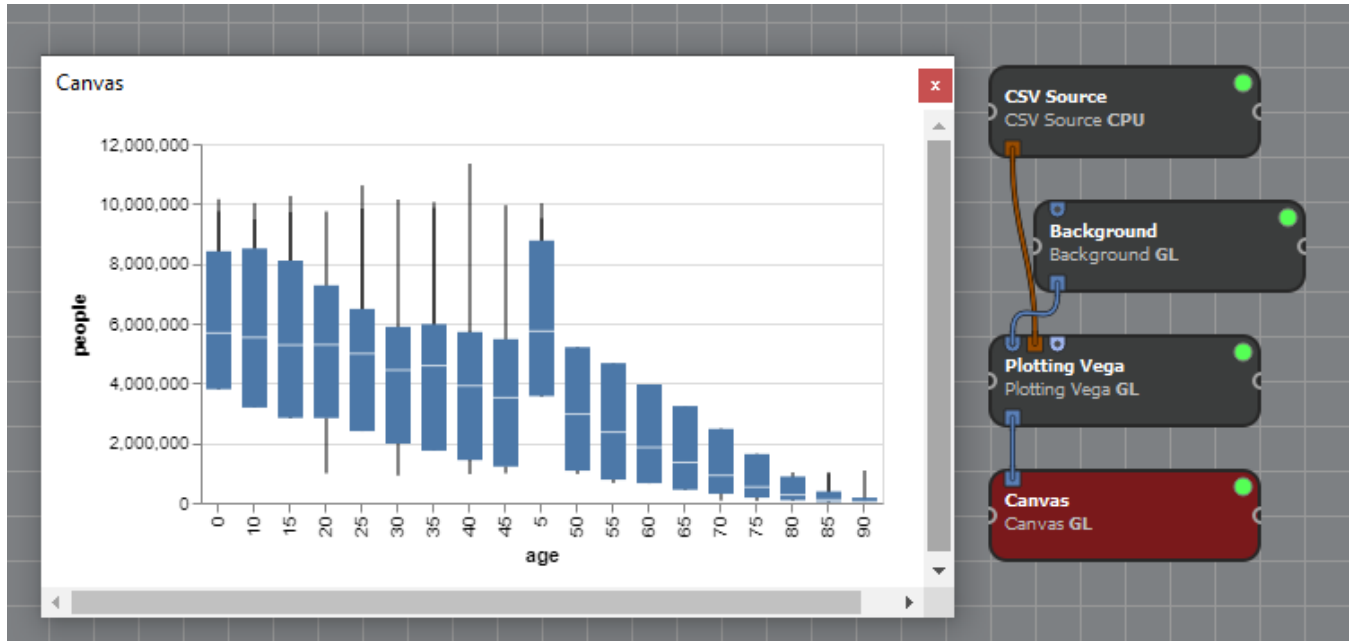
For full example see inviwo/apps/inviwopyapp/inviwo.py

Web integration

- Run HTML5/WebGL in Inviwo
  - User interface
  - Plotting
  - Etc.

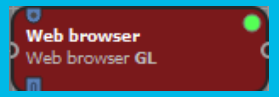






How does it work?

Inviwo



Property changed

Javascript/JSON

Set property

Mouse/Key events

WebBrowser::invokeEvent(…)

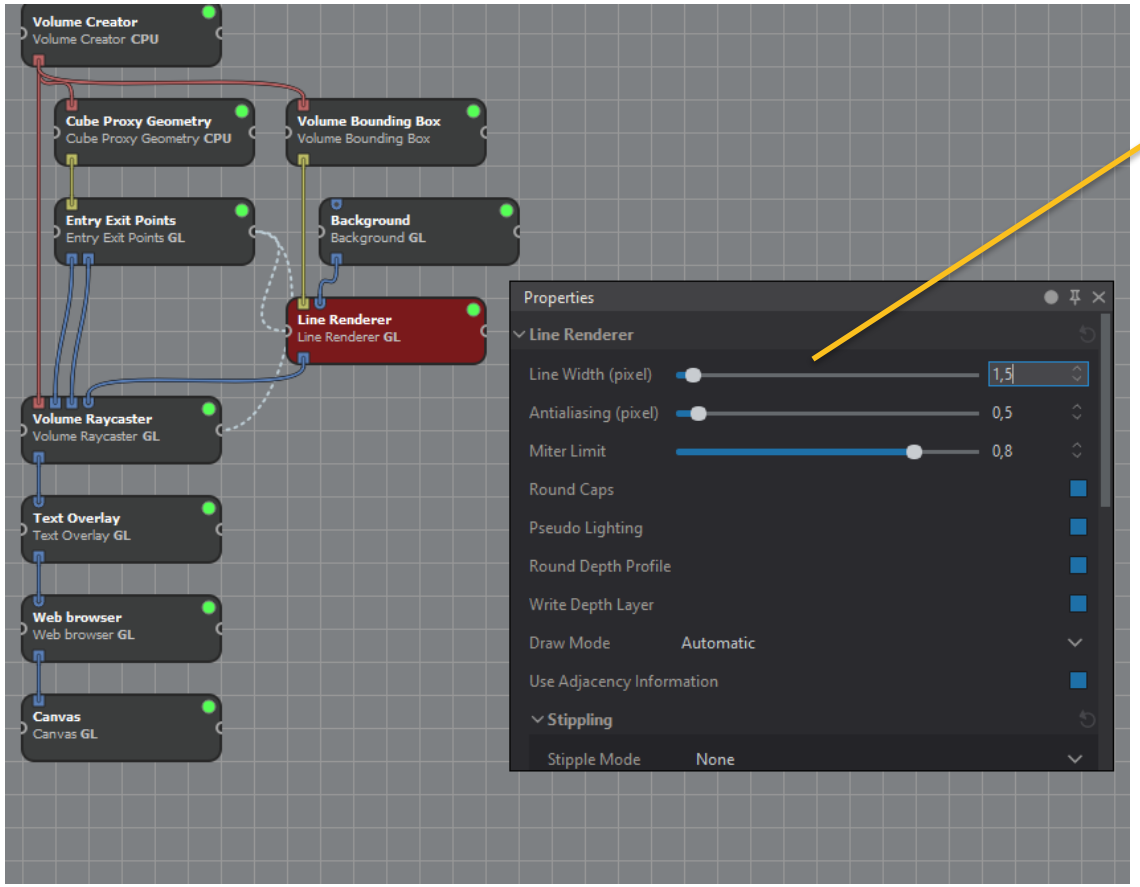
WebBrowser::process()

Webpage rendered

Chromium Embedded Framework
HTML/CSS/Javascript

Code example

See: File->Example Workspaces->WebBrowser



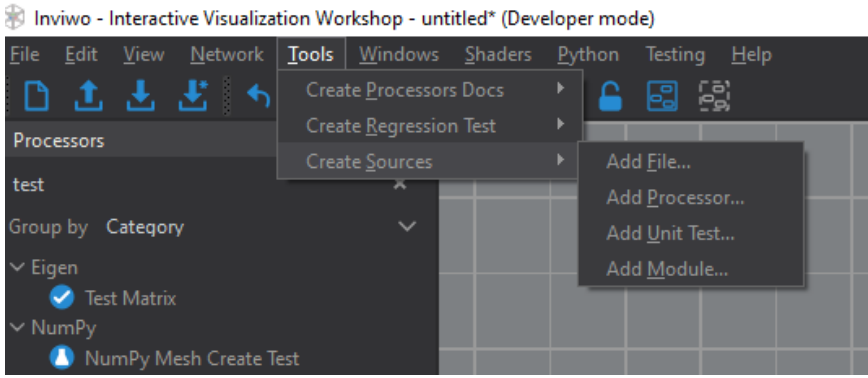

```
<html>
…
<body>
<p>Line width<input type="range" min="1" max="100" value="50" class="slider“
id="lineWidth"></p>

<script language="JavaScript">
// Initialize Inviwo API so that we can use it to synchronize properties
var inviwo = new InviwoAPI();

// Example: Get property from Inviwo
inviwo.getProperty("LineRenderer.lineWidth",
      function(prop) {
            // Get HTML element and set its value
            var elem = document.getElementById("lineWidth");
            elem.value = prop.value;
      }
);

// Example: Subscribe to property changes

// Callback for property changes, need to be in global scope.
function syncLineWidth(prop) {
      var elem = document.getElementById("LineRenderer.lineWidth");
      elem.min = prop.minValue;
      elem.max = prop.maxValue;
      elem.step = prop.increment;
      elem.value = prop.value;
}
// Update html element when the corresponding Inviwo property changes
inviwo.subscribe("LineRenderer.lineWidth", syncLineWidth);
</script>
```

Low-level abstractions

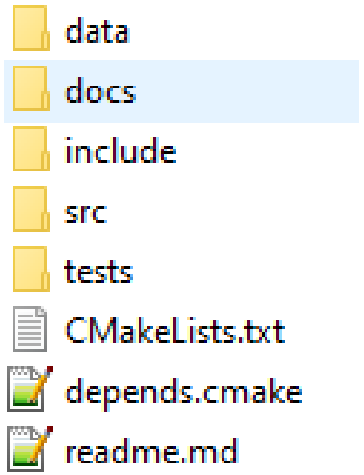
– the C++ world

LINKÖPING UNIVERSITY

Creating a module

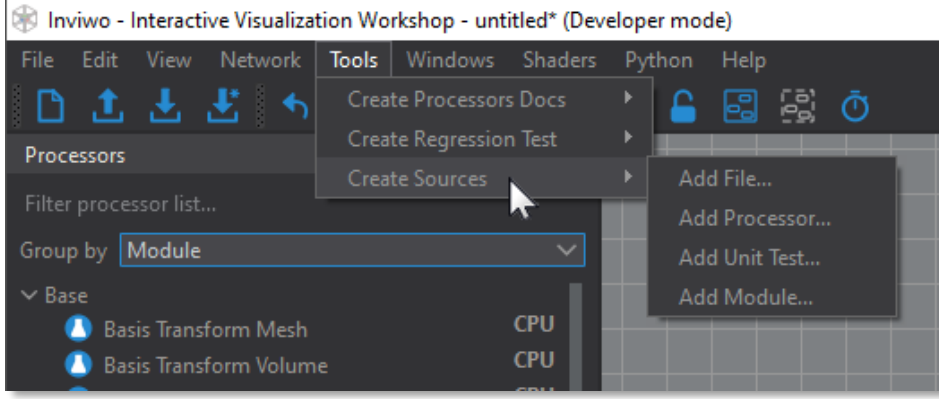


- Option 1: Tools->Create Sources->Add Module
  - Go to **`yourrepo/modules/`** and specify module name (`MyModule`)
  - Run Cmake
- Option 2: Command line
  - Go to **`build/bin/<config>/`**
  - Run **`./inviwo-meta-cli -m path/MyModule`**
  - Run CMake



- Creates module structure and CMake file

Creating a processor



- Option 1: Tools->Create Sources->Add Processor
  - Go to **`yourmodule/src/processors`** and specify Processor name (**`FancyProcessor`**)
  - Run CMake
- Option 2: Command line
  - Go to **`build/bin/<config>/`**
  - Run **`./inviwo-meta-cli -p path/FancyProcessor`**
  - Run CMake

- Creates header and source files
  - Added to **`mymodule/CMakeLists.txt`**
  - Processor registered in **`MyModule`** constructor

```
registerProcessor<MyFancyProcessor>();
```

Processor – skeleton (cont.)

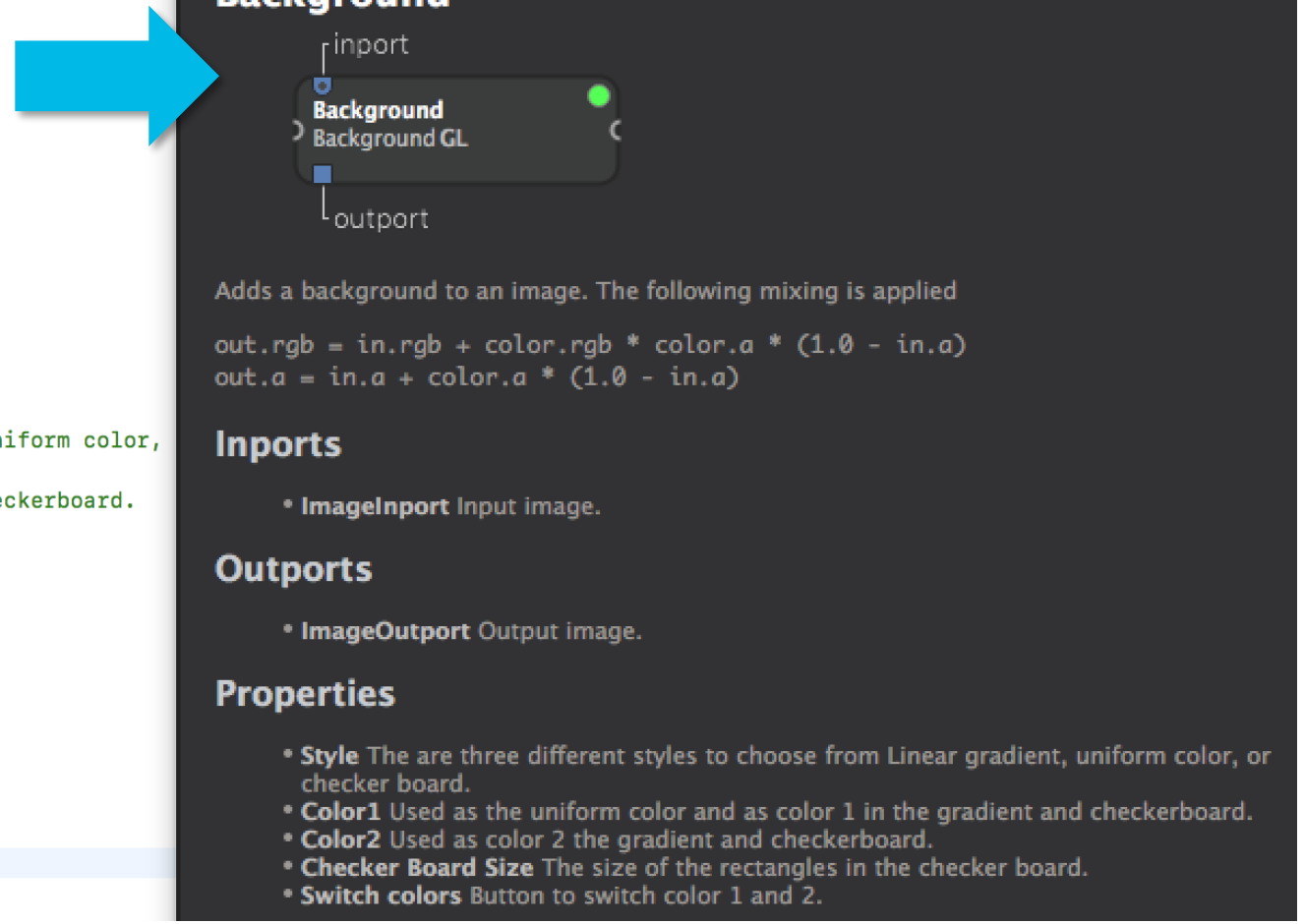
```
#pragma once

#include <inviwo/fancything/fancythingmoduledefine.h>
#include <inviwo/core/common/inviwo.h>
#include <inviwo/core/processors/processor.h>
#include <inviwo/core/properties/ordinalproperty.h>
#include <inviwo/core/ports/imageport.h>

class IVW_MODULE_FANCYTHING_API FancyProcessor : public Processor {
public:
    FancyProcessor();
    virtual ~FancyProcessor() = default;

    virtual void process() override;

    virtual const ProcessorInfo getProcessorInfo() const override;
    static const ProcessorInfo processorInfo_;

private:
    ImageOutport outport_;
    FloatVec3Property position_;
};
```

Processor – skeleton (cont.)

```
// The Class Identifier has to be globally unique. Use a reverse DNS naming scheme
const ProcessorInfo FancyProcessor::processorInfo_{
    "org.inviwo.FancyProcessor",  // Class identifier
    "Fancy Processor",            // Display name
    "Undefined",                  // Category
    CodeState::Experimental,      // Code state
    Tags::None,                   // Tags
};
const ProcessorInfo FancyProcessor::getProcessorInfo() const { return processorInfo_; }

FancyProcessor::FancyProcessor()
    : Processor()
    , outport_("outport")
    , position_("position", "Position", vec3(0.0f), vec3(-100.0f), vec3(100.0f)) {

    addPort(outport_);
    addProperty(position_);
}

void FancyProcessor::process() { outport_.setData(myImage); }
```

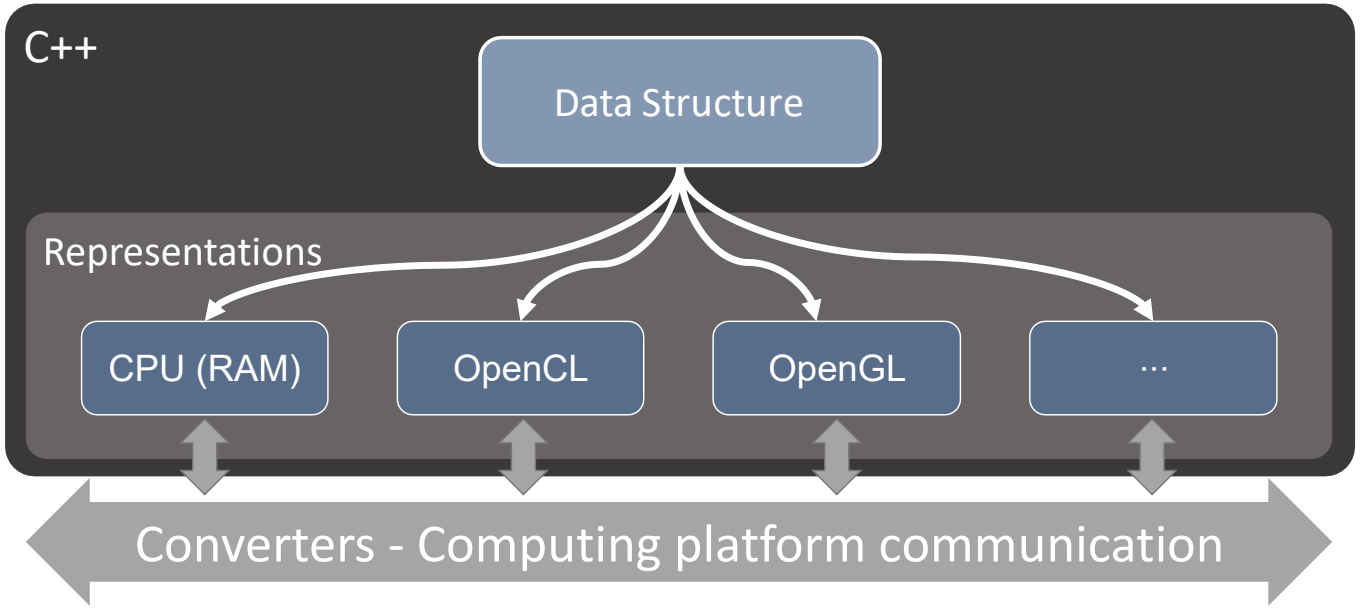
Documentation

```
/** \docpage{org.inviwo.Background, Background}
 * ![](org.inviwo.Background.png?classIdentifier=org.inviwo.Background)
 * Adds a background to an image.
 * The following mixing is applied
 *
 *     out.rgb = in.rgb + color.rgb * color.a * (1.0 - in.a)
 *     out.a = in.a + color.a * (1.0 - in.a)
 *
 * ### Inports
 *   * __ImageInport__ Input image.
 *
 * ### Outports
 *   * __ImageOutport__ Output image.
 *
 * ### Properties
 *   * __Style__ The are three different styles to choose from Linear gradient, uniform color,
 *     or checker board.
 *   * __Color1__ Used as the uniform color and as color 1 in the gradient and checkerboard.
 *   * __Color2__ Used as color 2 the gradient and checkerboard.
 *   * __Checker Board Size__ The size of the rectangles in the checker board.
 *   * __Switch colors__ Button to switch color 1 and 2.
 */

/**
 * \brief Adds a background to an image.
 *
 */
class IVW_MODULE_BASEGL_API Background : public Processor {
public:
    Background();
    virtual ~Background();
```



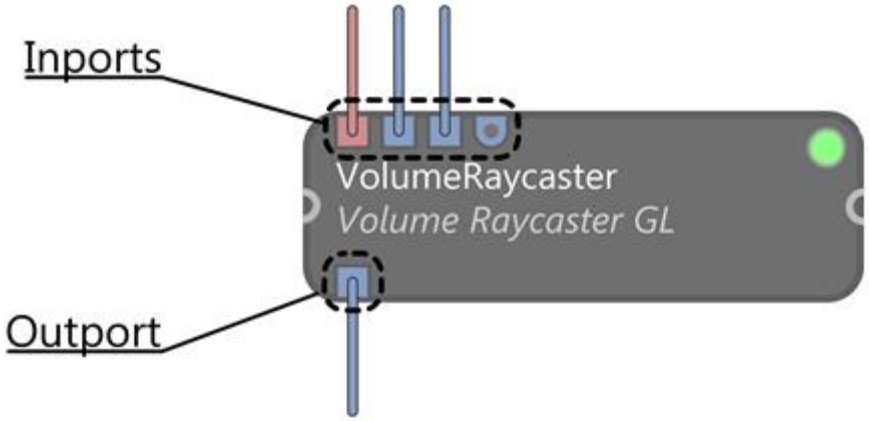
Data structures

- Volume
- Layer
- Image
  - Consists of layers
  - Color, depth, picking
- Buffer
- Mesh
  - Buffers
  - Index Buffers

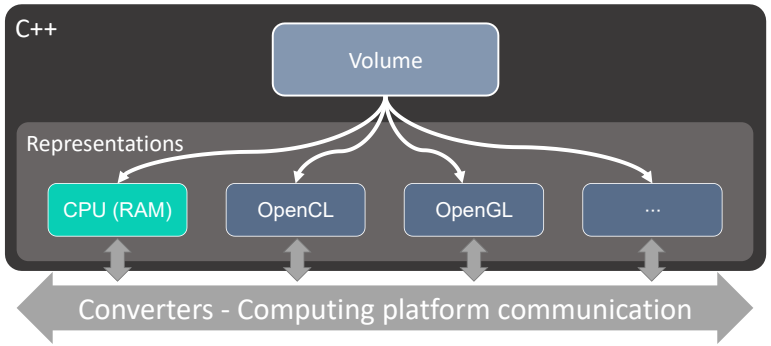


How to access data

- Ports hold data (shared pointer)
  - Inport is const (`std::shared_ptr<const Volume>`)
  - Assign new data to outport (`std::shared_ptr<Volume>`)
- Data representations
  - Type-erased data (void*, and size)
  - VolumeRAMPrecision<type>
  - VolumeGL
  - VolumeCL
- Dispatching to obtain types (similar to VTK)



How to access data – VolumeRAM



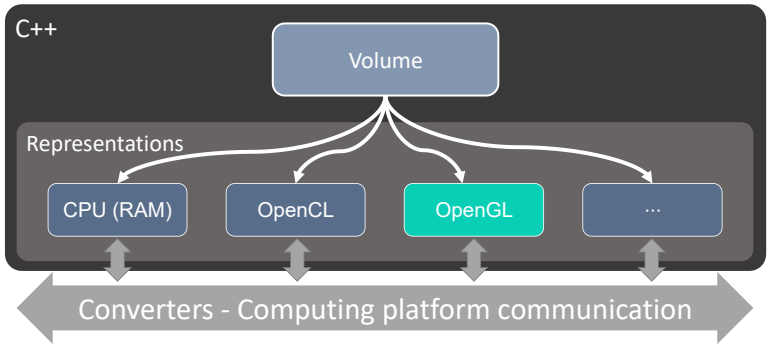

```
auto scaleVolume = [](auto ram) {
    using ValueType = util::PrecisionValueType<decltype(ram)>;
    const size3_t dims = ram->getDimensions();

    auto dstRam = std::make_shared<VolumeRAMPrecision<typename ValueType>>(dims);

    const ValueType* srcData = ram->getDataTyped();
    ValueType* dstData = dstRam->getDataTyped();
    for (size_t i = 0; i < (dims.x * dims.y * dims.z); ++i) {
        dstData[i] = srcData[i] * ValueType(2);
    }
    return std::make_shared<Volume>(dstRam);
};

auto volume = inport_.getData()->getRepresentation<VolumeRAM>()
    ->dispatch< std::shared_ptr<Volume> >( scaleVolume );

outport_.setData(volume);
```

```
VolumeInport inport_("inport");
VolumeOutport outport_("outport");
```

Return type of dispatch

Dispatch function;
gets called with correctly deduced type

How to access data – VolumeGL



- Access OpenGL texture ID

```
auto volume = inport.getData();
const VolumeGL* volumeGL = volume->getRepresentation<VolumeGL>();

std::shared_ptr<const Texture3D> tex = volumeGL->getTexture();
GLuint textureID = tex->getID();

glBindTexture(GL_TEXTURE_3D, textureID);
```

- Upload data

```
auto dstVolume = std::make_shared<Volume>(size3_t(1, 1, 1), DataFormat<float>::get());

VolumeGL* volumeGL = dstVolume->getEditableRepresentation<VolumeGL>();
float* data = ...;
volumeGL->getTexture()->uploadAndResize(data, size3_t(32, 32, 32));
```

Application show cases

– what’s possible

LINKÖPING UNIVERSITY

Inviwo.org

Need help?

inviwo.slack.com

SeRC
Swedish e-Science Research Centre

ELLIIT

LINKÖPING UNIVERSITY

KTH VETENSKAP OCH KONST

universität ulm

D. Jönsson, *et al.*, “Inviwo - A Visualization System with Usage Abstraction Levels” in *IEEE Transactions on Visualization & Computer Graphics*, 2019.
doi: 10.1109/TVCG.2019.2920639