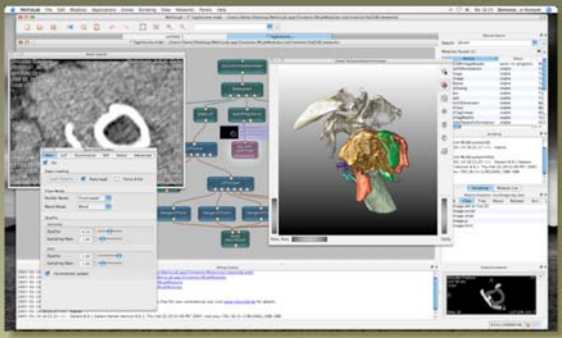
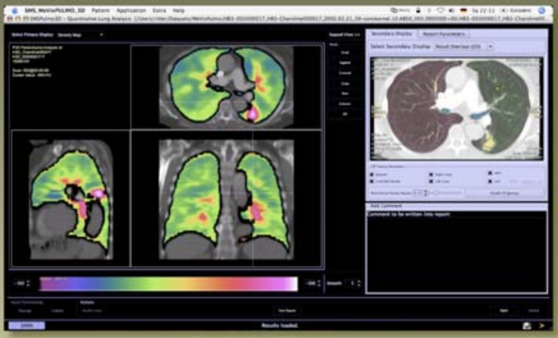
Visual Programming for Prototyping of Medical Imaging Applications





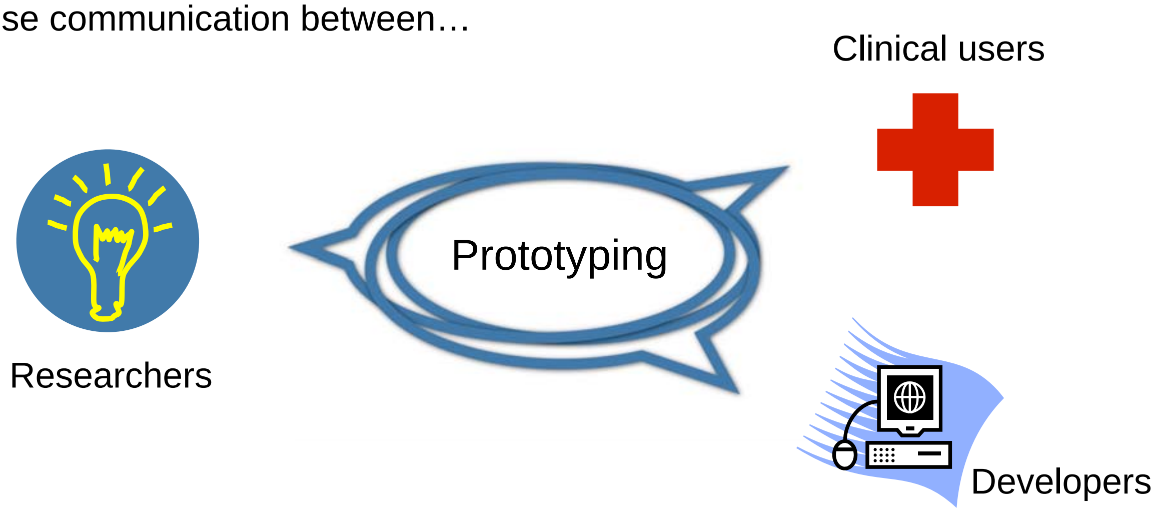
Felix Ritter, MeVis Research Bremen, Germany

Overview

- Introduction to MeVisLab
- Visual Programming
- Image Processing / VIsualization Examples
- VTK / ITK Integration
- MeVisLab SDK Features
- GUI Scripting

Prototyping in Medical Imaging Research

Innovation in clinical medical imaging requires close communication between…



Clinical users

Researchers

Prototyping

Developers

Prototyping serves as a common language!

MeVisLab Prototyping Platform

- MeVisLab is:
  - Medical Image Processing and Visualization Platform
  - Research and Development Tool
  - Rapid Application Prototyping Environment

  - Cross-platform (Windows, Mac OS X, Linux)
  - Free for non-commercial usage

Related Visualization Platforms

- Amira
- Analyze
- AVS Express
- IBM Data Explorer/OpenDX
- Khoros/VisQuest
- LONI
- SCIRun
- …

MeVisLab Development Platform

C++

Graphical

Scripting

Research and development in MeVisLab ...

... on the module level

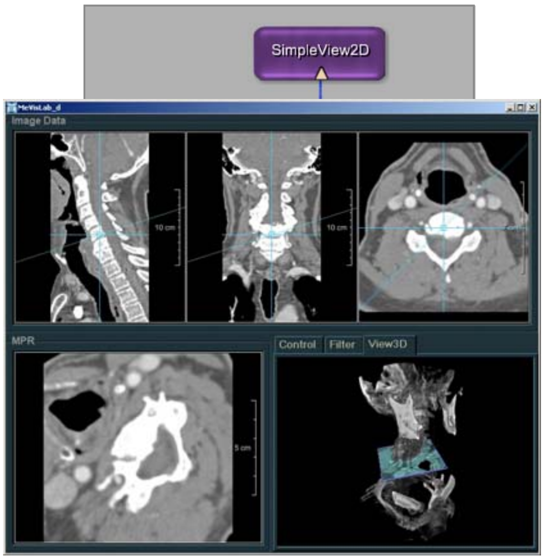
- Powerful toolbox libraries
- Efficient Interfaces

... on the network level

- Flexibility and modularity
- Module toolbox

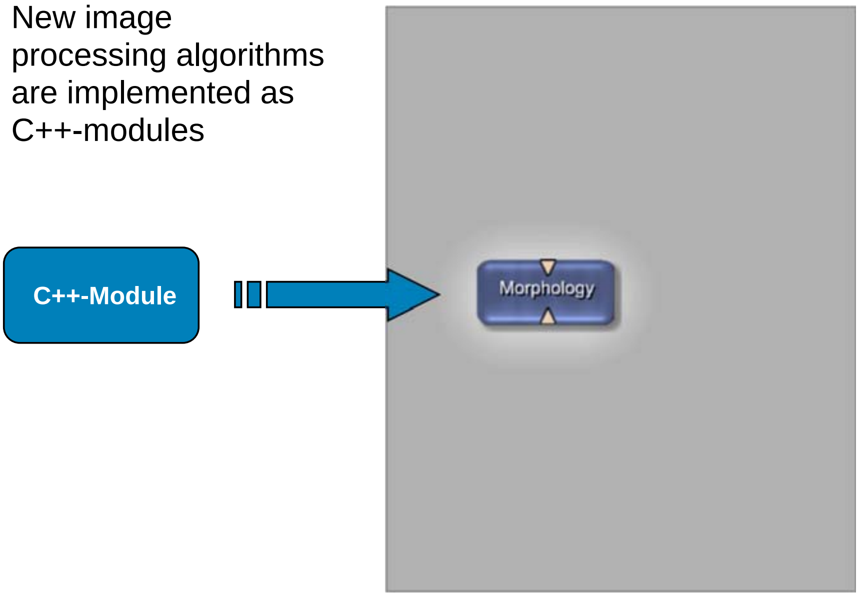
... on the application level

- Interactive, efficient application framework



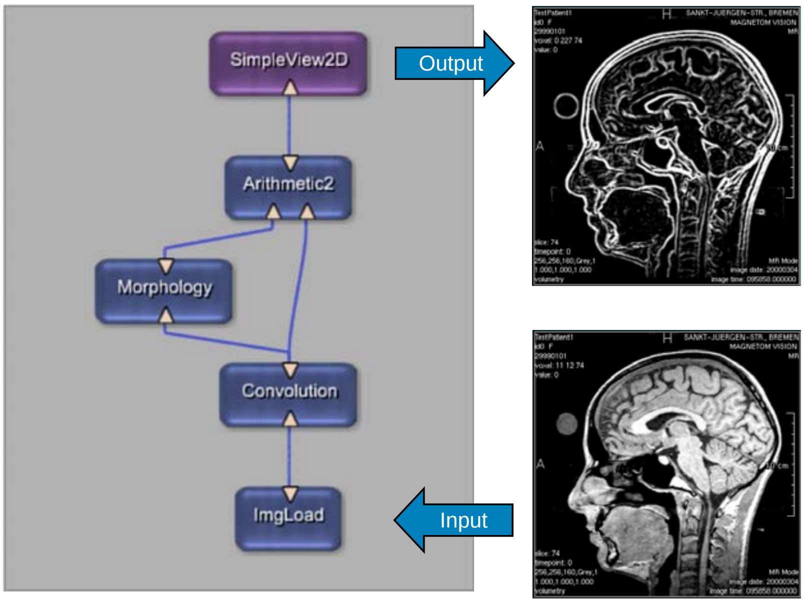
Different application development interfaces at different levels:

New image processing algorithms are implemented as C++-modules



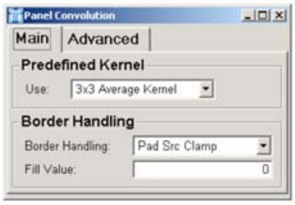
Different application development interfaces at different levels:

Individual image processing modules are combined to powerful networks using a graphical user interface



Different application development interfaces at different levels:

Each image processing module can be controlled using its own parameter panel



Panel Convolution

Main | Advanced

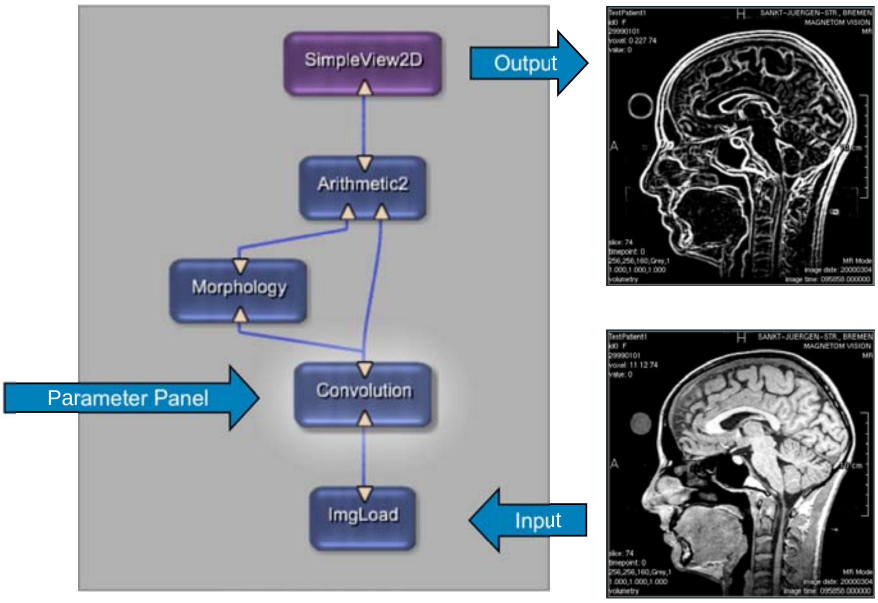
Predefined Kernel

Use: 3x3 Average Kernel

Border Handling

Border Handling: Pad Src Clamp

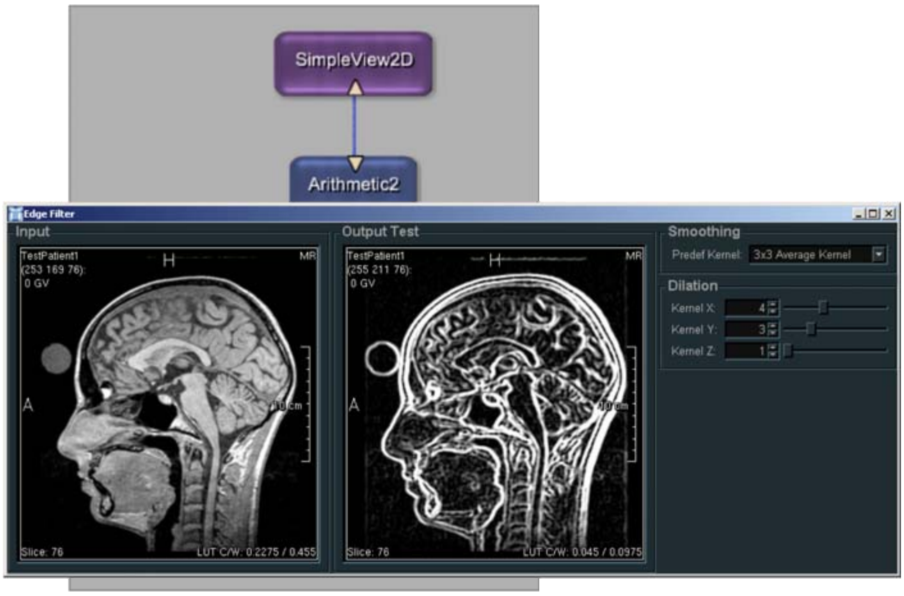
Fill Value: 0



Different application development interfaces at different levels:

An application prototype is designed using a powerful scripting language

```
Horizontal "Edge Filter" {
  Box "Input" {
    Viewer viewIn.self
  }
  Box "Output" {
    Viewer viewOut.self
  }
  Vertical {
    Box "Smoothing" {
      Field conv.PredefKernel
    }
    Box "Dilation" {
      layout = Vertical
      Field morph.KernelX
      Field morph.KernelY
      Field morph.KernelZ
    }
  }
}
```



Smoothing

Predef Kernel: 3x3 Average Kernel

Dilation

Kernel X: 4

Kernel Y: 3

Kernel Z: 1

Available Modules

- 450 Image Processing Modules
- 300 Open Inventor Modules
- 400 Macro Modules
- 300 ITK Modules
- 1000 VTK Modules

Image Processing

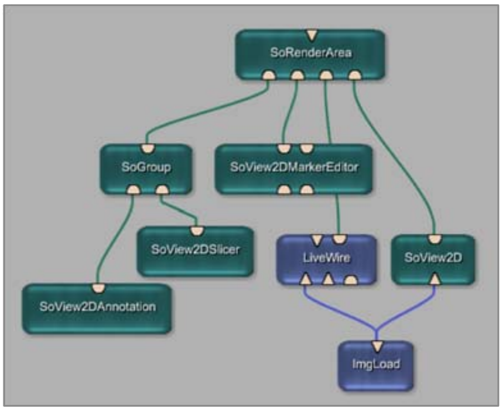
- MeVis Image Processing Library (ML)
- Page oriented and request driven
- Priority controlled caching
- General image concept:
  - Various data types (int, float, complex, tensors)
  - x/y/z/color/time/user dimensions
- Medical image properties:
  - DICOM coordinate system and tags
- C++ Interface and Wizard available for integration of new algorithms

Image Processing

- Filters
  - Diffusion filters
  - Morphology filters
  - Kernel filters
- Segmentation
  - Region growing
  - Live wire
  - Fuzzy connectedness
  - Threshold
  - Manual contours
- Transformations
  - Affine transformations
  - Distance transformations
  - Radon transform
  - Manual registration
- Statistics
  - Histograms
  - Global image statistics
  - Box counting dimension
- Other
  - Unary/binary arithmetic
  - Resampling/reformatting
  - Oblique and curved MPR
  - Dynamic data analysis
  - Noise/test pattern generators

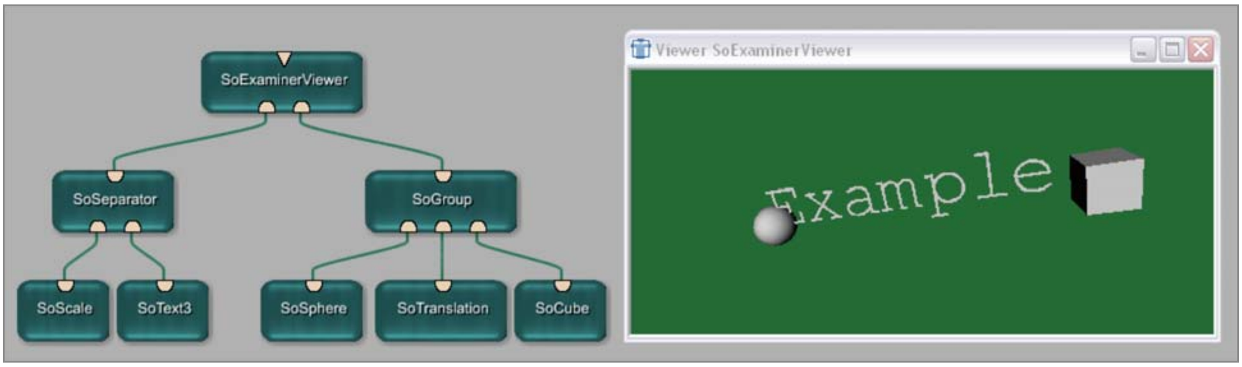
Open Inventor (OIV)

- Direct Open Inventor node support
- Open Inventor:
  - Scene graph paradigm
  - Object, rendering, transformation, property, … nodes
  - Based on OpenGL
  - Well documented
- Extensions to support 2D image viewing/manipulation
- Mixed ML/Open Inventor modules
- www.mevislab.de/inventor



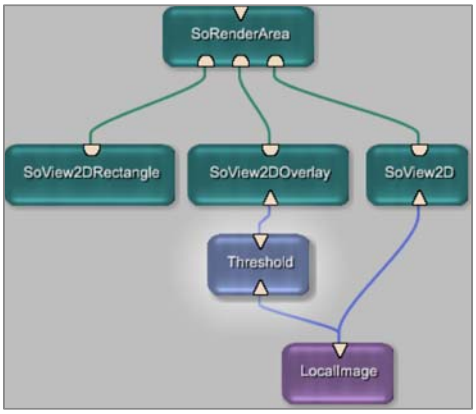
Open Inventor Scene Graph

- Scene objects are represented by nodes
- Size and position is defined by transformation nodes
- A rendering node represents the root of the scene graph



2D Viewers

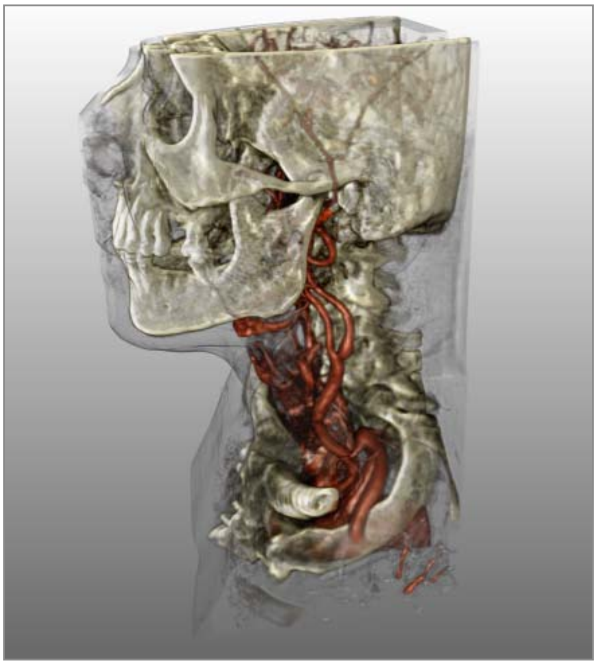
- Modular 2D Viewer Library (SoView2D)
- Hardware accelerated using textures and shaders
- Supports interactive LUT even on large images
- Extension mechanism supports:
  - Overlays
  - Markers
  - ROIs
  - Contours
  - User extensions can add drawing and event handling

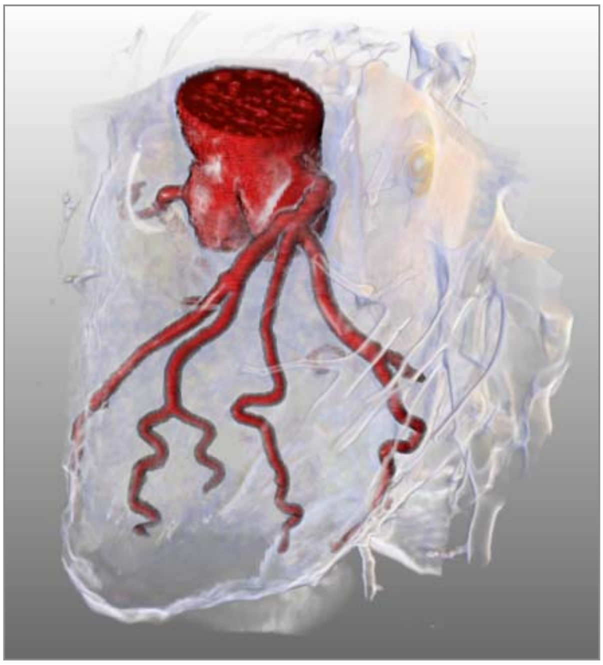


Volume Rendering

- Advanced Volume Rendering modules
  - MIP, DVR, Shaded DVR
  - Tone Shading, Silhouette and Boundary Enhancement
  - Tagged/Labeled Objects
  - Per Object Shading
  - Large data visualization via multi-resolution data octree

Volume Rendering Examples



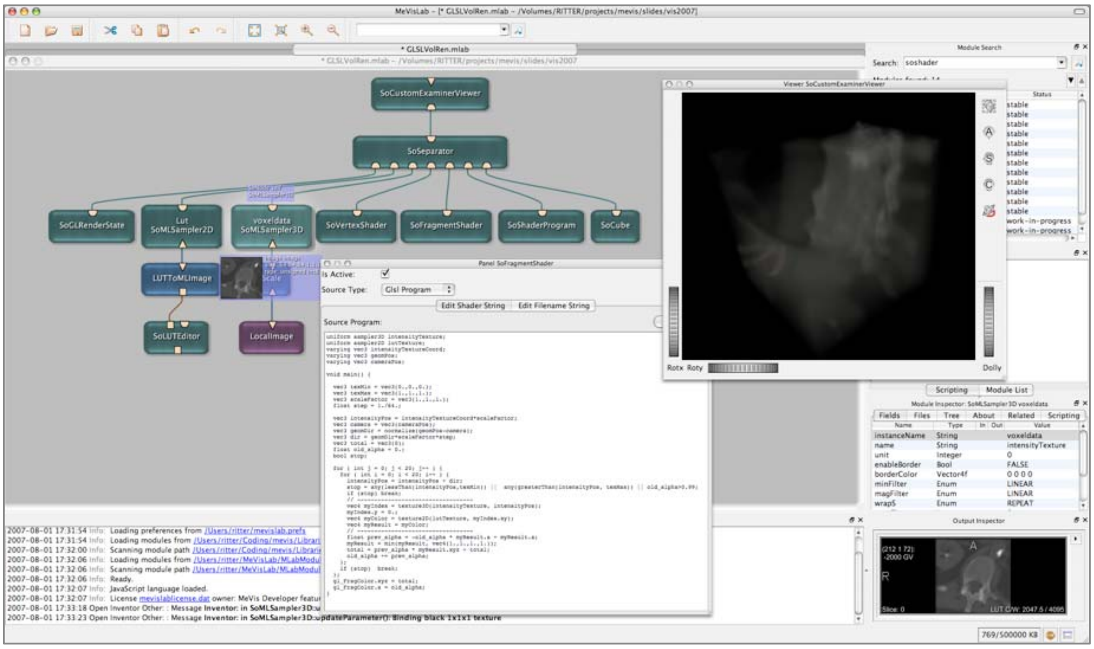


Prototyping GLSL Shaders

- Support for OpenGL Shading Language
- Enables prototyping of advanced visualization / image processing algorithms
- Textures are loaded using ML image pipeline
- Support for OpenGL framebuffer objects
- Textures may be loaded from the graphics card and directed into the ML image pipeline

Prototyping GLSL Shaders

Simple volume ray casting using GLSL shader framework



Panel SoFragmentShader

Is Active: ☑

Source Type: Glsl Program

Edit Shader String | Edit Filename String

Source Program:

```
uniform sampler3D intensityTexture;
uniform sampler2D lutTexture;
varying vec3 intensityTextureCoord;
varying vec3 geomPos;
varying vec3 cameraPos;

void main() {

  vec3 texMin = vec3(0.,0.,0.);
  vec3 texMax = vec3(1.,1.,1.);
  vec3 scaleFactor = vec3(1.,1.,1.);
  float step = 1./64.;

  vec3 intensityPos = intensityTextureCoord*scaleFactor;
  vec3 camera = vec3(cameraPos);
  vec3 geomDir = normalize(geomPos-camera);
  vec3 dir = geomDir*scaleFactor*step;
  vec3 total = vec3(0);
  float old_alpha = 0.;
  bool stop;

  for ( int j = 0; j < 20; j++ ) {
    for ( int i = 0; i < 20; i++ ) {
      intensityPos = intensityPos + dir;
      stop = any(lessThan(intensityPos,texMin)) ||  any(greaterThan(intensityPos, texMax)) || old_alpha>0.99;
      if (stop) break;
      // ----------------------------------
      vec4 myIndex = texture3D(intensityTexture, intensityPos);
      myIndex.y = 0.;
      vec4 myColor = texture2D(lutTexture, myIndex.xy);
      vec4 myResult = myColor;
      // ----------------------------------
      float prev_alpha = -old_alpha * myResult.a + myResult.a;
      myResult = min(myResult, vec4(1.,1.,1.,1.));
      total = prev_alpha * myResult.xyz + total;
      old_alpha += prev_alpha;
    };
    if (stop)  break;
  };
  gl_FragColor.xyz = total;
  gl_FragColor.a = old_alpha;
}
```

```
2007-08-01 17:31:54 Info:  Loading preferences from /Users/ritter/mevislab.prefs
2007-08-01 17:31:54 Info:  Loading modules from /Users/ritter/Coding/mevis/Librar
2007-08-01 17:32:00 Info:  Scanning module path /Users/ritter/Coding/mevis/Librarie
2007-08-01 17:32:06 Info:  Loading modules from /Users/ritter/MeVisLab/MLabModul
2007-08-01 17:32:06 Info:  Scanning module path /Users/ritter/MeVisLab/MLabModul
2007-08-01 17:32:06 Info:  Ready.
2007-08-01 17:32:07 Info:  JavaScript language loaded.
2007-08-01 17:32:07 Info:  License mevislablicense.dat owner: MeVis Developer featu
2007-08-01 17:33:18 Open Inventor Other: : Message Inventor: in SoMLSampler3D::u
2007-08-01 17:33:23 Open Inventor Other: : Message Inventor: in SoMLSampler3D::updateParameter(): Binding black 1x1x1 texture
```

Module Inspector: SoMLSampler3D voxeldata

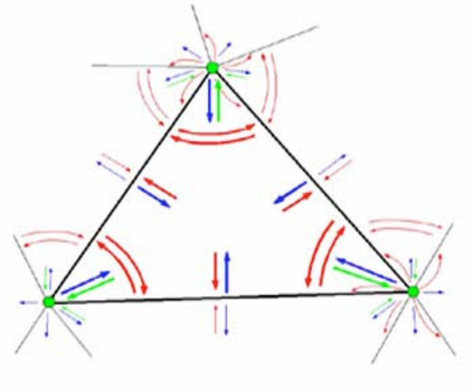
Fields | Files | Tree | About | Related | Scripting

Name	Type	In	Out	Value
instanceName	String			voxeldata
name	String			intensityTexture
unit	Integer			0
enableBorder	Bool			FALSE
borderColor	Vector4f			0 0 0 0
minFilter	Enum			LINEAR
magFilter	Enum			LINEAR
wrapS	Enum			REPEAT

Winged Edge Mesh Library (WEM)

- Data structure proposed by Baumgart, 1975
- Mesh consists of Nodes, Edges and Faces
- Dense pointer structure of incident primitives
- Fast access to neighboring structures

Pointer links in a neighborhood:



WEM Modules Overview

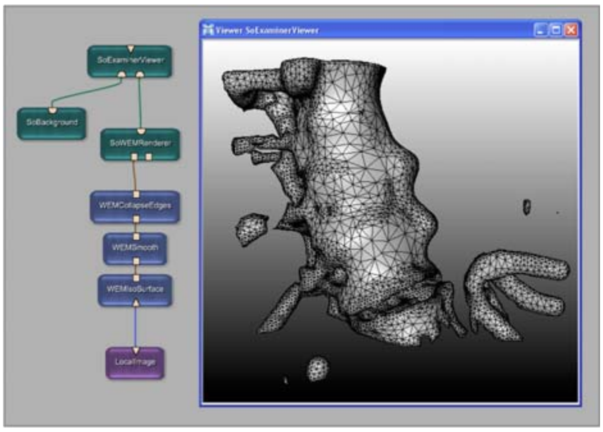
- Generation:
  - WEMIsoSurface
- Processing:
  - WEMCollapseEdges
  - WEMSmooth
  - WEMPurge
  - WEMClip
  - …
- Rendering:
  - SoWEMRenderer
    - Different Render Modes
    - Optional Coloring by LUT Values

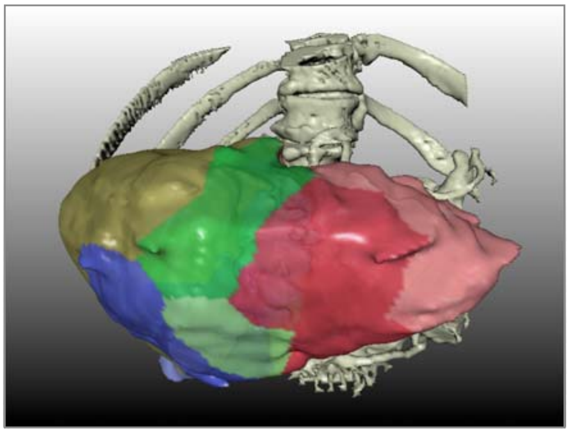
… and many more, type in ‘WEM’ in the search field.

WEM Screenshots

Network with iso surface generation and polygon reduction

A liver surface colored by a LUT in bone context



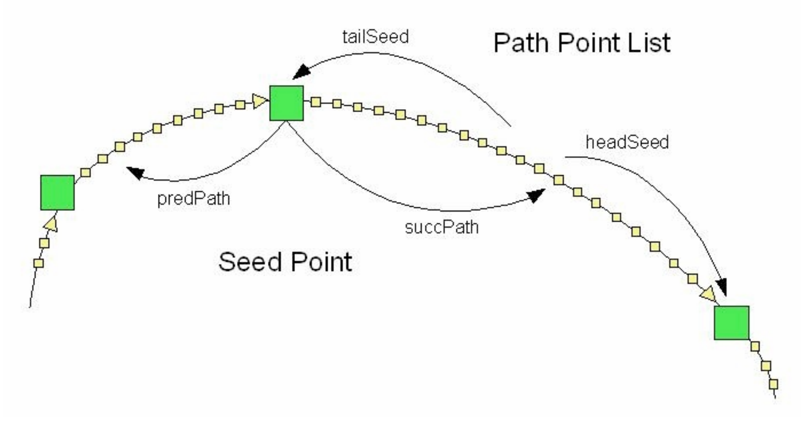


Contour Segmentation Objects (CSO)

- CSO library provides data structures and modules for interactive or automatic generation of contours in voxel images
- Contours can be analyzed, maintained, grouped and converted back into a voxel image
- Contours may „communicate“ with each other
- Contours can be displayed in 2D and 3D
- CSOs are 3D objects (world coordinates)
- CSOGroups group contours which share a set of attributes

Contour Segmentation Objects

- CSO consists of a number of seed points and a number of path point lists



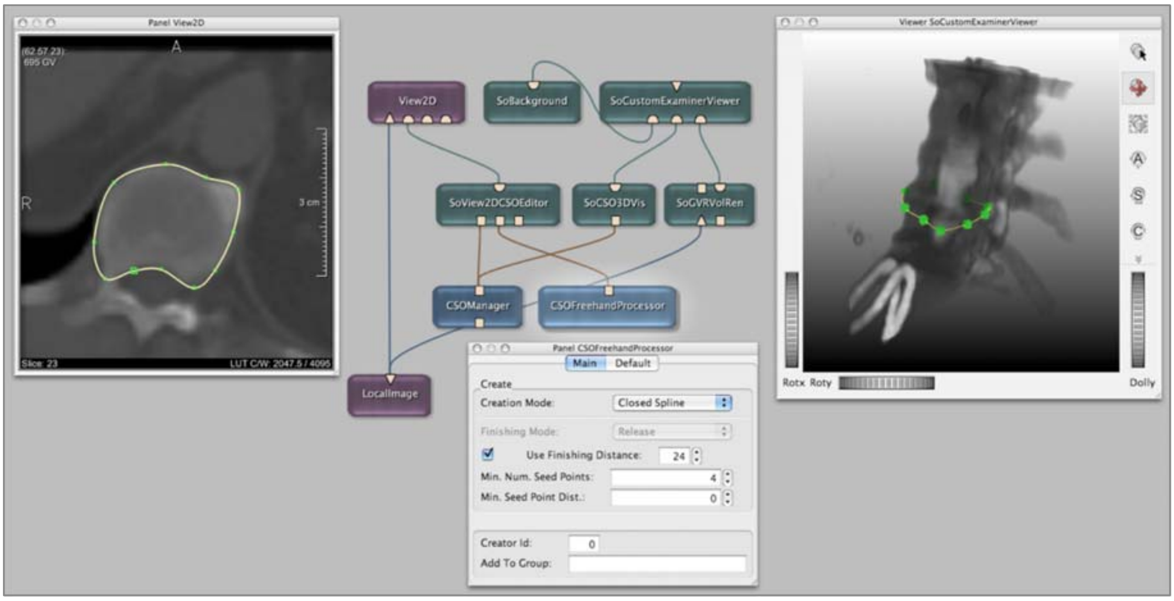
CSO Modules Overview

- Generation (without interaction):
  - CSOIsoGenerator
- Processing (with interaction):
  - CSOFreehandProcessor
  - CSOLiveWireProcessor
  - CSOIsoProcessor
  - CSOBulgeProcessor
  - …
- Rendering
  - SoView2DCSOEditor
  - SoCSO3DVis
- Misc
  - CSOConvertToImage
  - CSOConvertTo3DMask
  - CSOFilter
  - CSOManager
  - CSOLoad / CSOSave
  - …

… and many more, type in ‘CSO’ in the search field.

CSO Screenshot

Visualizing a contour in 2D slices and within a 3D volume rendering



Create

Creation Mode: Closed Spline

Finishing Mode: Release

☑ Use Finishing Distance: 24

Min. Num. Seed Points: 4

Min. Seed Point Dist.: 0

Creator Id: 0

Add To Group:

DICOM Support

- Import of 2D/3D/4D DICOM datasets
- MeVisLab DICOM Service runs as Windows Service or UNIX Daemon and receives data from PACS even when user is logged out
- Export of DICOM slices to disk
- DICOM-Store allows to send data to PACS

ITK Wrapper

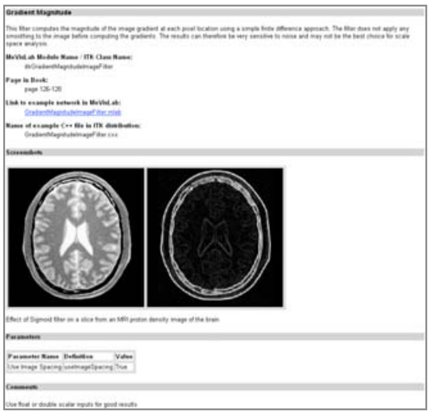
- ITK – Insight Toolkit (www.itk.org)
- Open Source Library for Medical Image Processing and Registration
- about 200 Modules for Standard Image Processing such as
  - Image Arithmetics
  - Kernel-based and Diffusion Filtering
  - Levelset and Segmentation Filtering
  - Warping, Resampling Filters
- about 90 Modules Registration-Related Algorithms
  - Interpolators
  - Metrics
  - Optimizers
  - Transformations
- A few hundred other classes such as functions etc.

ITK Book Examples

ITK Book Example → Corresponding Website (screenshots generated with MeVisLab) → MeVisLab Network

www.itk.org/ItkSoftwareGuide.pdf

www.mevislab.de/index.php?id=35



Gradient Magnitude

This filter computes the magnitude of the image gradient at each pixel location using a simple finite difference approach. The filter does not apply any smoothing to the image before computing the gradients. The results can therefore be very sensitive to noise and may not be the best choice for scale space analysis.

MeVisLab Module Name / ITK Class Name:
itkGradientMagnitudeImageFilter

Page in Book:
page 126-128

Link to example network in MeVisLab:
GradientMagnitudeImageFilter.mlab

Name of example C++ file in ITK distribution:
GradientMagnitudeImageFilter.cxx

Screenshots

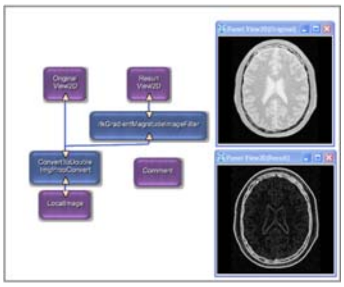
Effect of Sigmoid filter on a slice from an MRI proton density image of the brain

Parameters

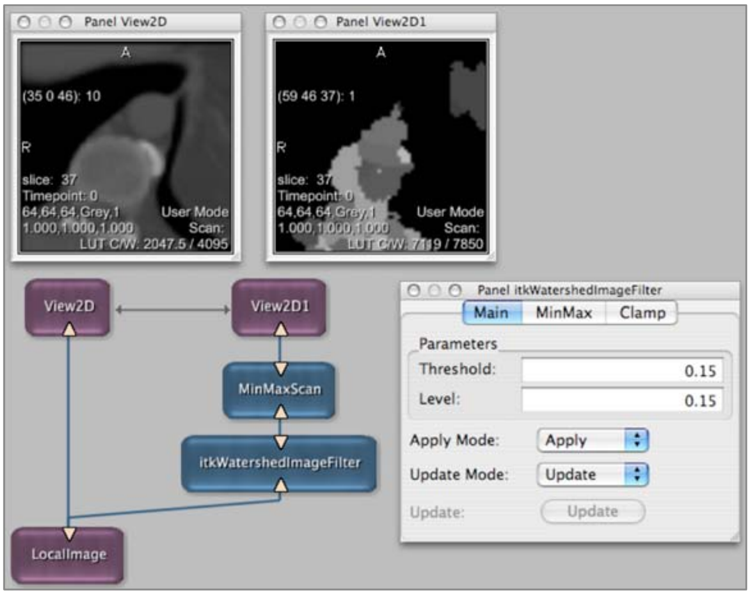
Parameter Name	Definition	Value
Use Image Spacing	useImageSpacing	True

Comments

Use float or double scalar inputs for good results



ITK Example



Threshold: 0.15

Level: 0.15

Apply Mode: Apply

Update Mode: Update

Update:

Smooth integration with
ML image processing
⇒ ITK modules behave
like normal ML modules

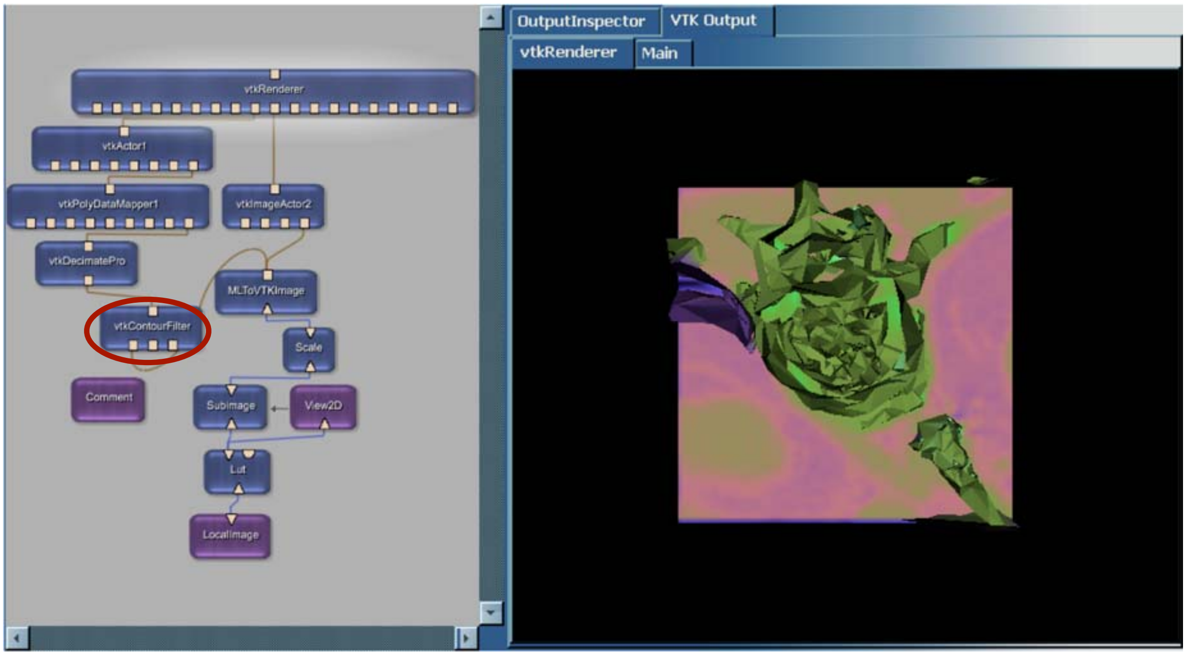
Each filter has additional
controls for:

- Clamping of image values
- Min / Max setting
- Update / Apply handling

VTK Wrapper

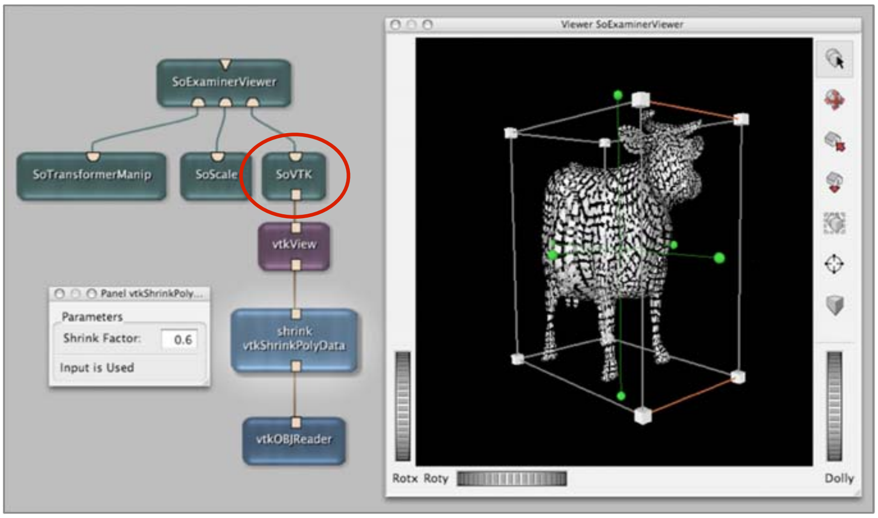
- VTK – Visualization Toolkit (www.vtk.org)
- Visualization, Image Processing and Filtering Library for images, meshes, grids, data sets etc.
- about 1000 Modules for
  - 2D/3D Image Processing
  - Grid, Mesh, Surface, and Data Filtering
  - Pickers
  - Properties and Actors
  - Mappers
  - Renderers, Widgets, Viewers
  - Sources, Readers and Writers
  - Transformations

VTK Example 1: Contour Filter



VTK Example 2: VTK / OIV mix

SoVTK module allows VTK rendering as part
of an Open Inventor scene graph



Parameters
Shrink Factor: 0.6
Input is Used

Automatic wrapper generation

- The ITK and VTK libraries are integrated into MeVisLab using a generic wrapping approach
- This approach facilitates updates to new library versions and makes almost all algorithms of ITK/VTK instantly available
- Other platforms do this wrapping manually and offer a less extensive ITK/VTK integration

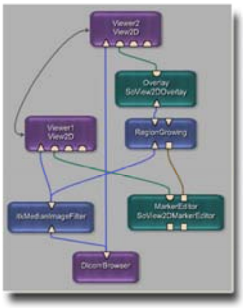
MeVisLab SDK

- Allows to extend MeVisLab with
  - ML Modules
  - Open Inventor Modules
  - Macro Modules
  - ITK and VTK Modules
- Efficient user interface development
- JavaScript / Python scripting languages

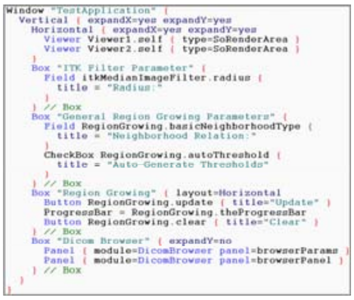
Scripting (MDL)

- User interfaces are created with the „Module Definition Language“ (MDL)
- Abstract hierarchical GUI language
- Interpreted at run-time, allows rapid prototyping
- www.mevislab.de/fileadmin/docs/html/mdl/

GUI Scripting Example

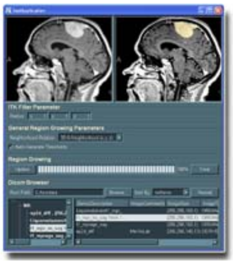


Module Network

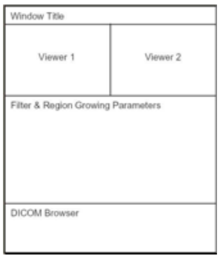


```
Window "TestApplication" {
  Vertical { expandX=yes expandY=yes
    Horizontal { expandX=yes expandY=yes
      Viewer Viewer1.self { type=SoRenderArea }
      Viewer Viewer2.self { type=SoRenderArea }
    }
    Box "ITK Filter Parameter" {
      Field itkMedianImageFilter.radius {
        title = "Radius:"
      }
    } // Box
    Box "General Region Growing Parameters" {
      Field RegionGrowing.basicNeighborhoodType {
        title = "Neighborhood Relation:"
      }
      CheckBox RegionGrowing.autoThreshold {
        title = "Auto-Generate Thresholds"
      }
    } // Box
    Box "Region Growing" { layout=Horizontal
      Button RegionGrowing.update { title="Update" }
      ProgressBar = RegionGrowing.theProgressBar
      Button RegionGrowing.clear { title="Clear" }
    } // Box
    Box "Dicom Browser" { expandY=no
      Panel { module=DicomBrowser panel=browserParams }
      Panel { module=DicomBrowser panel=browserPanel }
    } // Box
  }
}
```

MDL Script



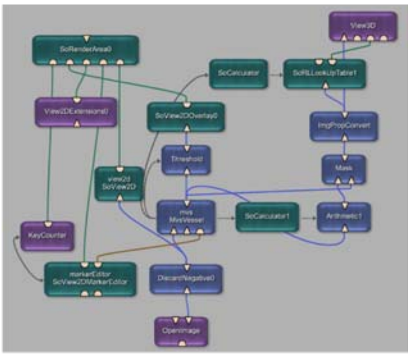
Graphical User Interface

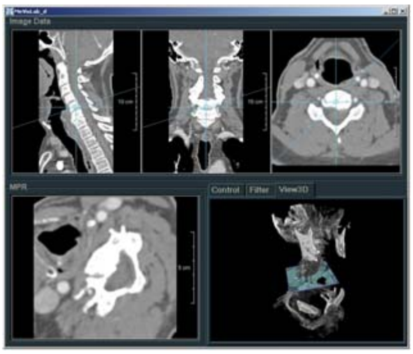


Schematic Representation

Application Prototyping

- Hide network complexity
- Design user interfaces
- Scripting for dynamic components





JavaScript / Python Integration

- Scripting can be used to program dynamic behaviour both on network and user interface level
  - Adding modules at run-time
  - Parameter computations and synchronization
  - Dynamic user interfaces
  - External processes
- JavaScript / Python bindings are available
- www.mevislab.de/fileadmin/docs/html/script/

Summary

- MeVisLab allows to learn about Medical Imaging and Visualization without C++ knowledge
- Visual Programming allows easy exploration of algorithms
- Open Inventor, ITK and VTK integrations offer a vast amount of available modules

Getting MeVisLab

- Get your free copy of MeVisLab at:
  www.mevislab.de
- The examples from this presentation are available at:
  www.mevislab.de/vis2007/

Licensing

- MeVisLab is free for non-commercial usage
- Many algorithms presented in this tutorial can be explored with the free edition of MeVisLab
- Full MeVisLab SDK is available at academic and commercial rates
  - Evaluation version available

Acknowledgments

I would like to thanks my colleagues at MeVis Research
for their contributions to this presentation:

T. Boskamp, O. Konrad, F. Link,
J. Rexilius and W. Spindler