Entwicklung interaktiver Bildverarbeitungssysteme mit MITK und CTK

Andreas Fetzer, Michael Müller
Marco Nolden, Sascha Zelzer

Deutsches Krebsforschungszentrum, Heidelberg

Bildverarbeitung für die Medizin 2013

Ablauf

- Toolkits für die medizinische Bildverarbeitung *Marco Nolden*
- Die MITK Workbench *Andreas Fetzer*
- ITK/VTK/MITK Konzepte *Sascha Zelzer*

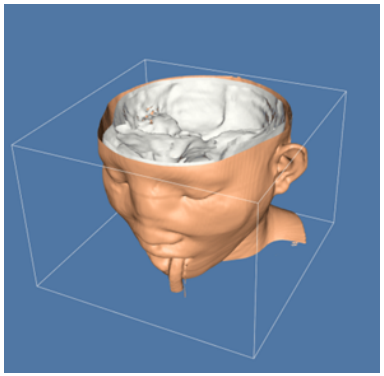
- Setup der MITK Umgebung *Michael Müller*
- Modularisierung in MITK *Sascha Zelzer*

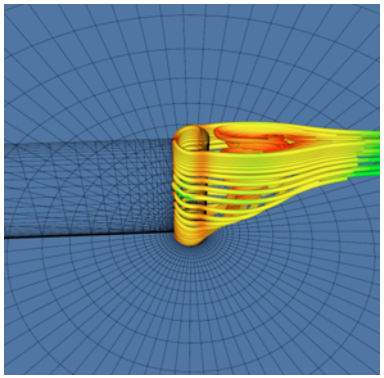
- Erstellen eines Beispiel-Plugins *Michael Müller*
- MITK Projekte *Marco Nolden*
- Prozesse, Dokumentation, Getting started *Marco Nolden*

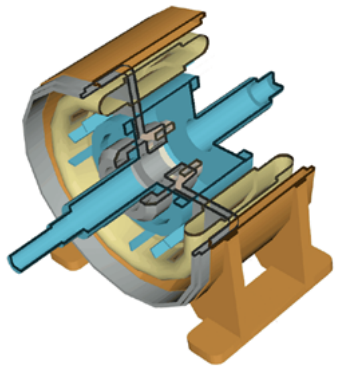
Toolkits: VTK, ITK, . . .

VTK

- The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. Will Schroeder, Ken Martin, and Bill Lorensen (1993)
- Supported by GE Research
- 1998: Foundation of Kitware Inc.
- Implemented in C++, Bindings for TCL, Python, Java ...
- A lot of visualization classes , simple image processing and GUI components







ITK

- Insight Segmentation and Registration Toolkit
- Call by NLM and NIH in 1999
- Six prime contractors from industry and science: GE, Kitware, University of North Carolina ...
- Data structures and algorithms for image segmentation and registration in medical applications
- No GUI oder application components
- Generic programming, “advanced” C++ features
- Open Source process, Insight Journal

DICOM:
Digital Imaging and Communication in Medicine

- Standard for medical imaging:
  file formats and network protocols
- Integration of image modalities (X-Ray, CT, MR . . .), storage systems (PACS) and other devices (printers, CD recorders . . .)
- Current version 3 dates back to 1993
- Challenges for the developer
  - Overall complexity of the standard (>5k pages and growing)
  - Conformance is variable
  - Non-standardized extensions (i.e. private tags) are quite common
  - Most $3D$ and $3D + t$ data are still encoded as 2D images

DICOM toolkits: DCMTK and GDCM

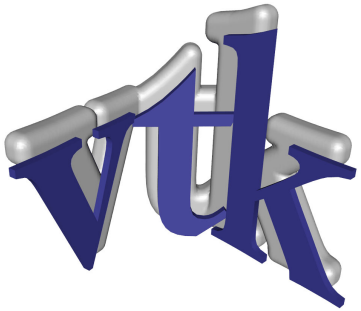
- DCMTK: the DICOM toolkit
- Started as reference implementation 1993
- Covers large parts of the standard
- Network functionality, low-level data handling

- GDCM: Grassroots DICOM
- Focus on parts 3, 5, 6 and 10: interpretation of image data
- Heuristics for “exotic” data
- GDCM is used by ITK for image reading

Qt

- Cross-platform application framework for C++
- GUI and non-GUI parts
- Very powerful, very mature, first version 1992
- Development tools: Qt Creator, Qt Designer
- Used in MITK for the workbench UI and the plugin framework

Motivation



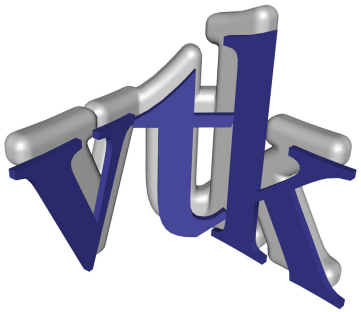
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

Motivation



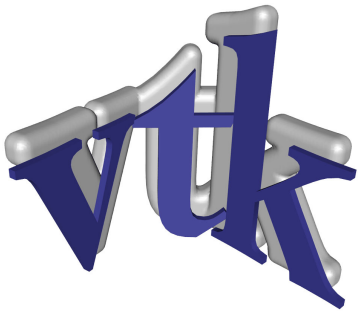
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

Motivation



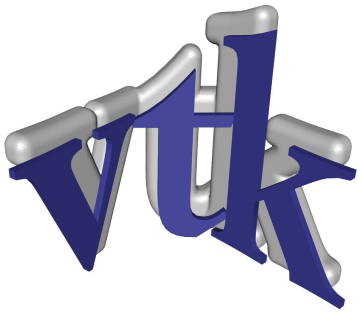
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

Motivation



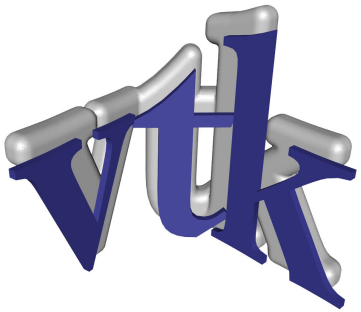
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

Motivation



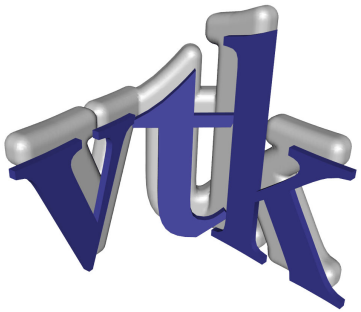
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

Motivation



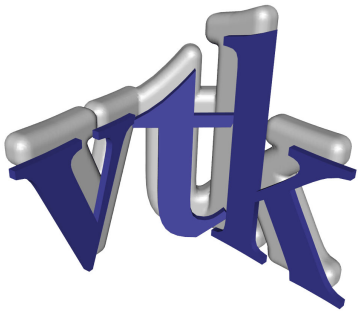
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

Motivation



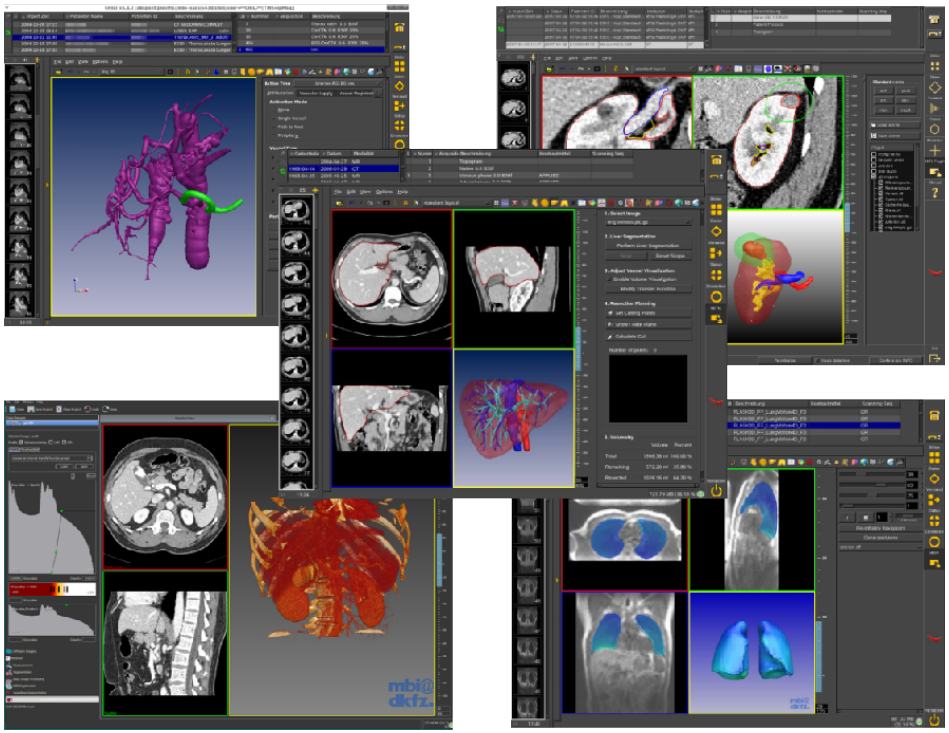
We had powerful toolkits for visualization and segmentation/registration

But:

insufficient support for *interactive multi-view software*

- uses as much as possible from ITK & VTK
- adds features outside the scope of boths
- is not at all a competitor to ITK or VTK

MITK

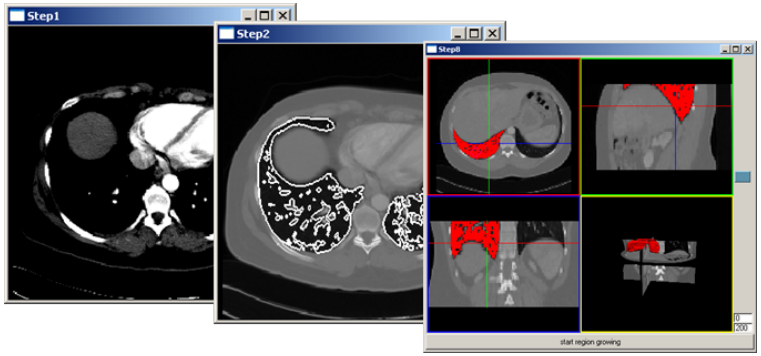


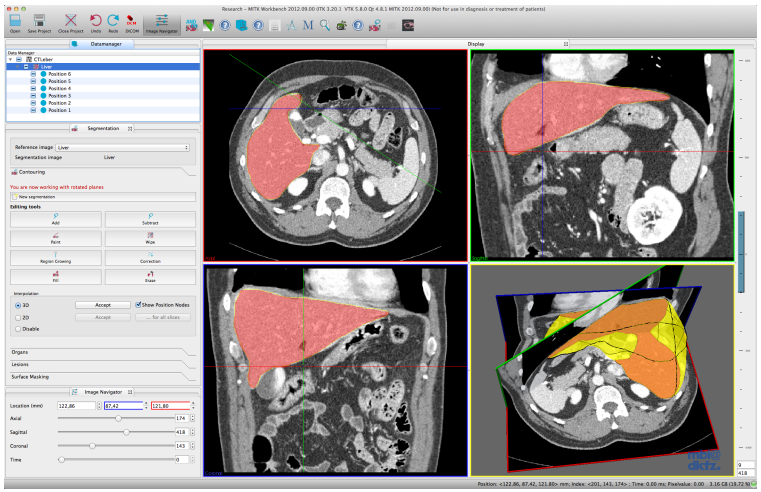
- Software library, developed since 2002
- Base for all applications of the department
- Ca. 500.000 lines of code (open source)
- Established in research and education (about 200 users in research)

MITK

MITK can be used in different ways:

- Develop new applications from scratch
  $\rightarrow$ MITK is used as a Software toolkit
- Extend the open source MITK application with new modules and plugins
  $\rightarrow$ MITK is used as a application framework
- Use the existing MITK Workbench (segmentation, registration, measurement, visualization ...)





CTK: The Common Toolkit

Kitware

DKFZ

OpenMAF

- Joint effort of multiple intitutions
- Planning Kick-off 2009 in Heidelberg
- Coding started 2010, six Hackfests so far

MedINRIA

GIMIAS

NiftyView

OpenXIP

CTK topics

DICOM I/O

- Query/Retrieve based on DCMTK
- Local data management
- UI components

Intraoperability

- Plug-in framework
- Command line modules
- Event bus

User Interfaces

- "Medical Imaging" Widgets
- Testing framework

Application Hosting

- DICOM Part 19
- Support for hosts and applications
- Goal: reference implementation

CTK topics

DICOM I/O

- Query/Retrieve based on DCMTK
- Local data management
- UI components

Intraoperability

- Plug-in framework
- Command line modules
- Event bus

User Interfaces

- "Medical Imaging" Widgets
- Testing framework

Application Hosting

- DICOM Part 19
- Support for hosts and applications
- Goal: reference implementation

CTK topics

DICOM I/O

- Query/Retrieve based on DCMTK
- Local data management
- UI components

Intraoperability

- Plug-in framework
- Command line modules
- Event bus

User Interfaces

- "Medical Imaging" Widgets
- Testing framework

Application Hosting

- DICOM Part 19
- Support for hosts and applications
- Goal: reference implementation

CTK topics

DICOM I/O

- Query/Retrieve based on DCMTK
- Local data management
- UI components

Intraoperability

- Plug-in framework
- Command line modules
- Event bus

User Interfaces

- "Medical Imaging" Widgets
- Testing framework

Application Hosting

- DICOM Part 19
- Support for hosts and applications
- Goal: reference implementation

MITK

- C++ framework / toolkit
- Open source BSD-style license, almost identical to VTK / ITK
- Supports Linux, Windows, Mac OS X
- Visual Studio, XCode, QtCreator, Eclipse
  MSVC, MinGW, GCC, Clang

MITK

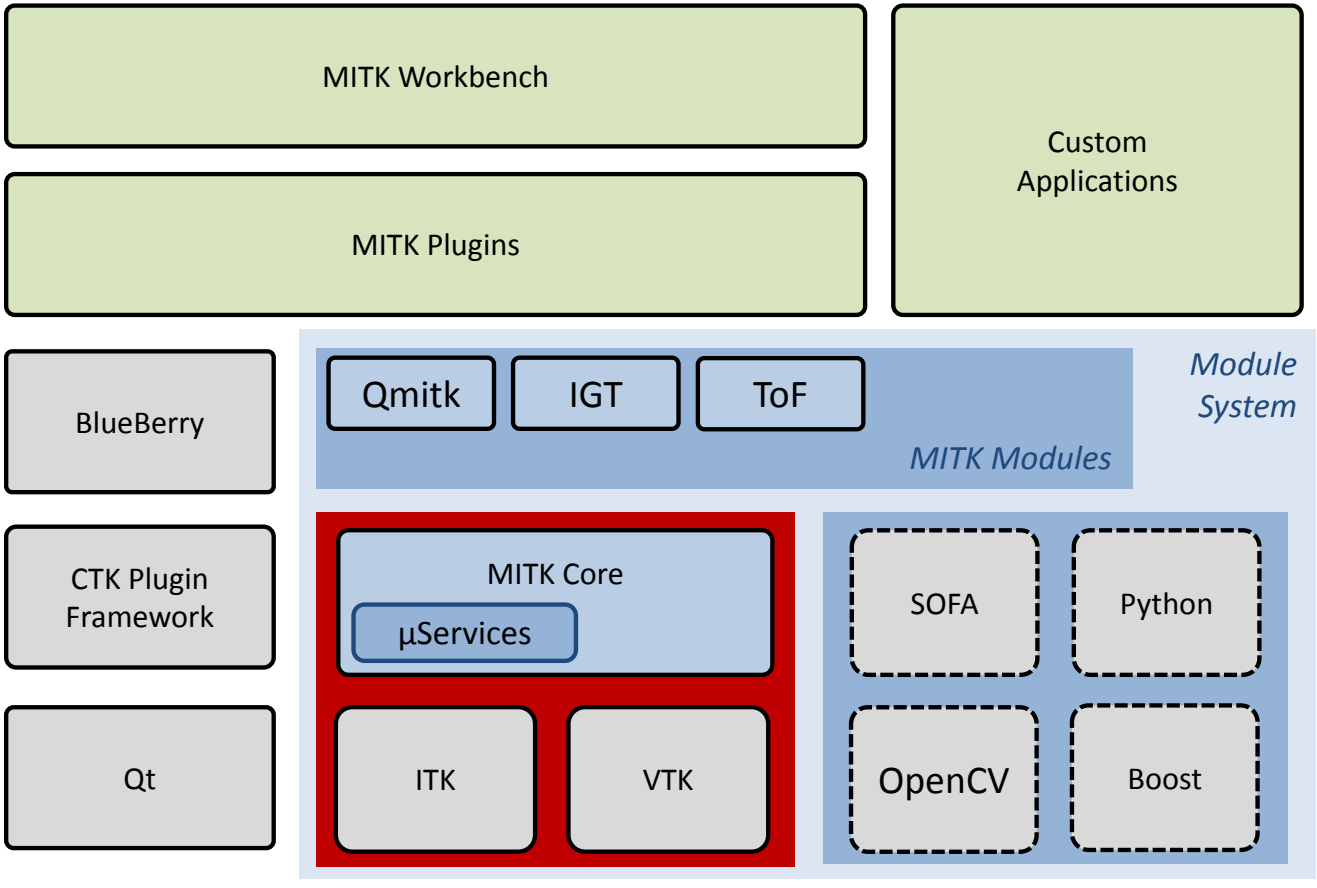
- C++ framework / toolkit
- Open source BSD-style license, almost identical to VTK / ITK
- Supports Linux, Windows, Mac OS X
- Visual Studio, XCode, QtCreator, Eclipse
  MSVC, MinGW, GCC, Clang

MITK

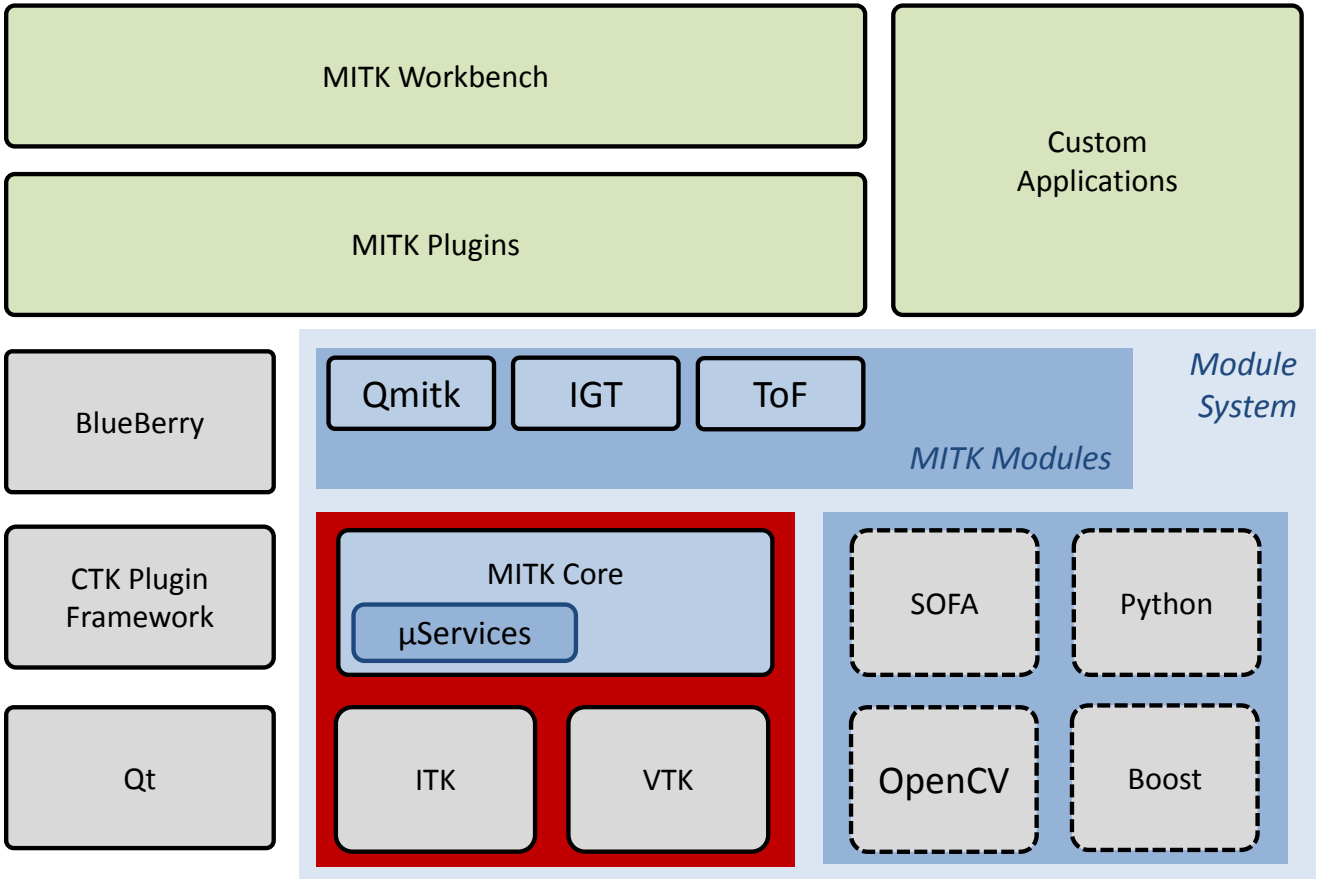
- C++ framework / toolkit
- Open source BSD-style license, almost identical to VTK / ITK
- Supports Linux, Windows, Mac OS X
- Visual Studio, XCode, QtCreator, Eclipse
  MSVC, MinGW, GCC, Clang

MITK

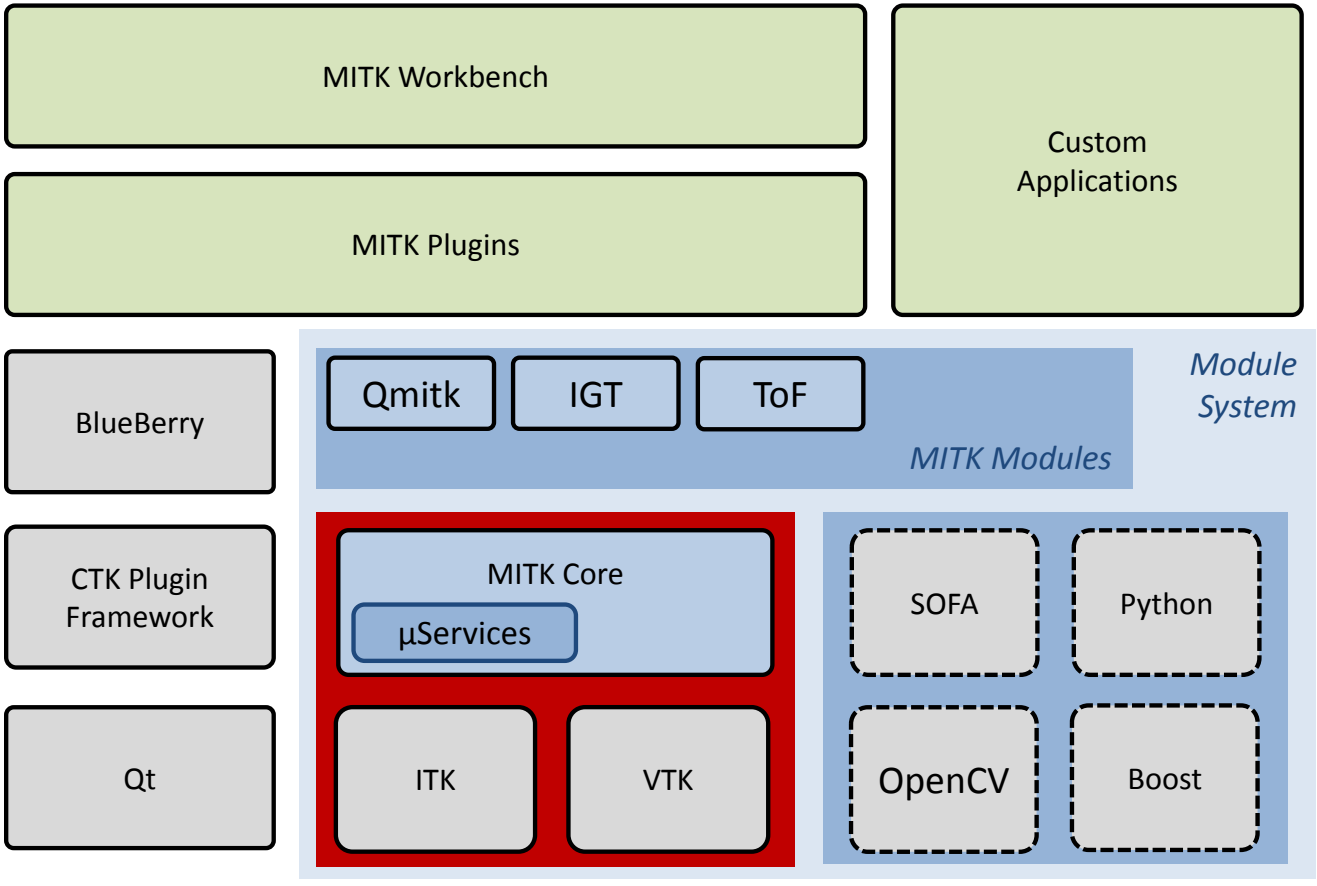
- C++ framework / toolkit
- Open source BSD-style license, almost identical to VTK / ITK
- Supports Linux, Windows, Mac OS X
- Visual Studio, XCode, QtCreator, Eclipse
  MSVC, MinGW, GCC, Clang



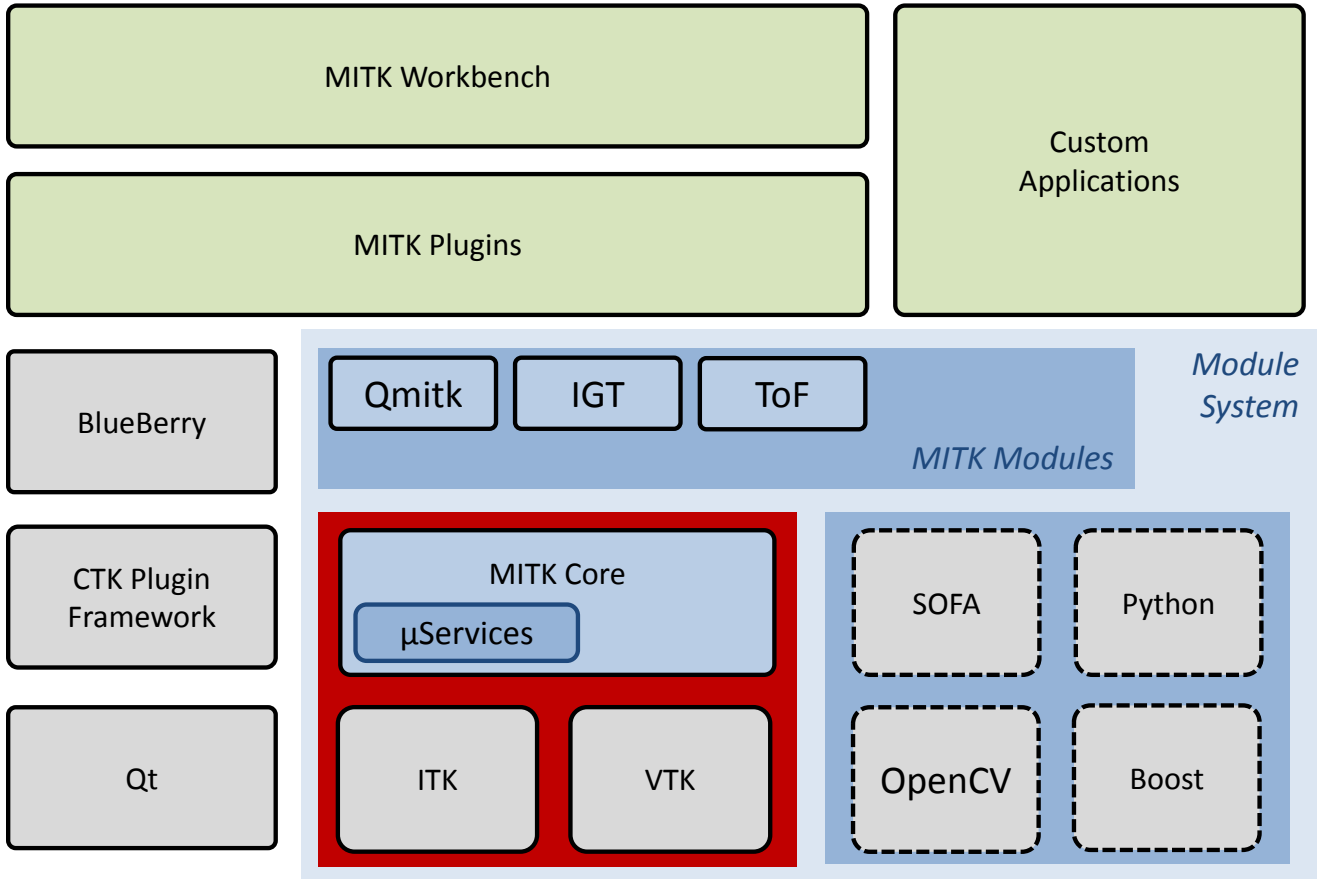
- MITK Core does not depend on a GUI toolkit
- Core provides data structures, rendering pipeline, interaction system, adaptors
- MITK-application-level provides Qt4 widgets and applications
- MITK Workbench serves as an application platform



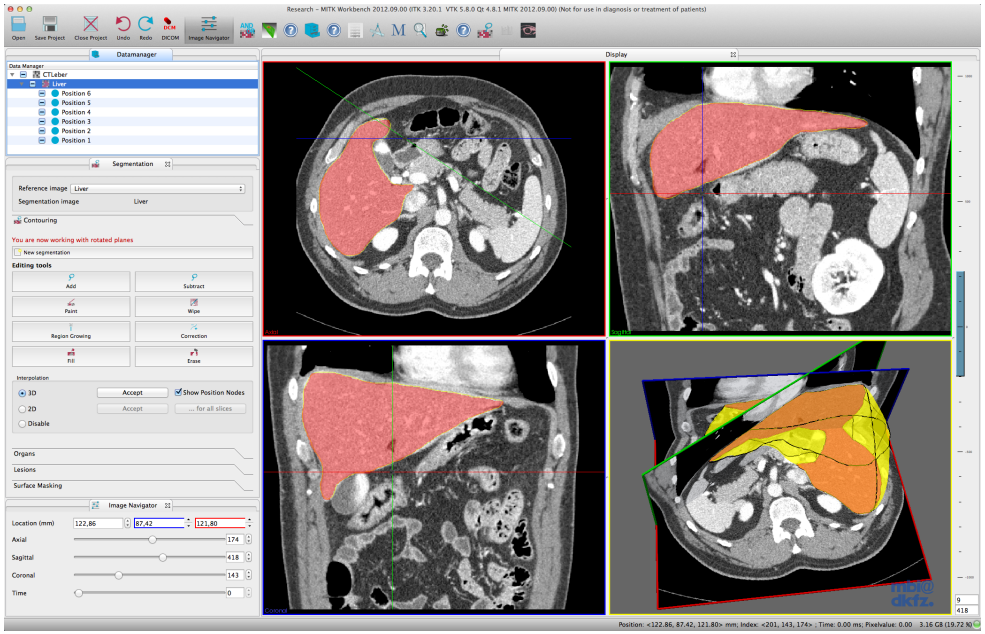
- MITK Core does not depend on a GUI toolkit
- Core provides data structures, rendering pipeline, interaction system, adaptors
- MITK-application-level provides Qt4 widgets and applications
- MITK Workbench serves as an application platform



- MITK Core does not depend on a GUI toolkit
- Core provides data structures, rendering pipeline, interaction system, adaptors
- MITK-application-level provides Qt4 widgets and applications
- MITK Workbench serves as an application platform



- MITK Core does not depend on a GUI toolkit
- Core provides data structures, rendering pipeline, interaction system, adaptors
- MITK-application-level provides Qt4 widgets and applications
- MITK Workbench serves as an application platform

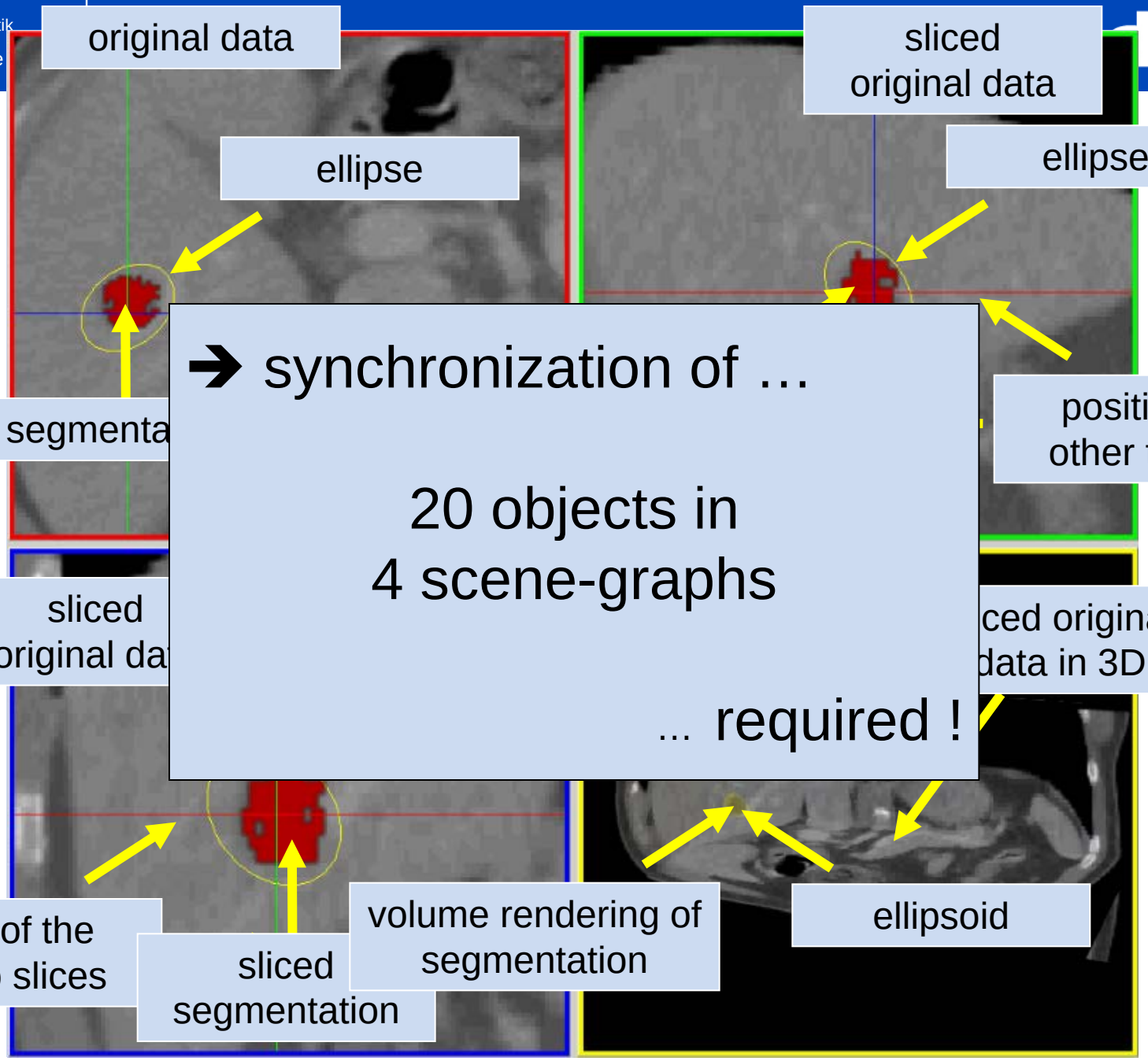

Research – MITK Workbench 2012.09.00 (ITK 3.20.1 VTK 5.8.0 Qt 4.8.1 MITK 2012.09.00) (Not for use in diagnosis or treatment of patients)
Open
Save Project
Close Project
Undo
Redo
DICOM
Image Navigator
Datamanager
Data Manager
CTLeber
Liver
Position 6
Position 5
Position 4
Position 3
Position 2
Position 1
Segmentation
Reference image
Liver
Segmentation image
Liver
Contouring
You are now working with rotated planes
New segmentation
Editing tools
Add
Subtract
Paint
Wipe
Region Growing
Correction
Fill
Erase
Interpolation
3D
2D
Disable
Accept
Show Position Nodes
Organs
Lesions
Surface Masking
Image Navigator
Location (mm)
122,86
87,42
121,80
Axial
Sagittal
Coronal
Time
Display

Thank you!

MITK: Concepts

ITK/VTK/MITK – a quick overview

Sascha Zelzer

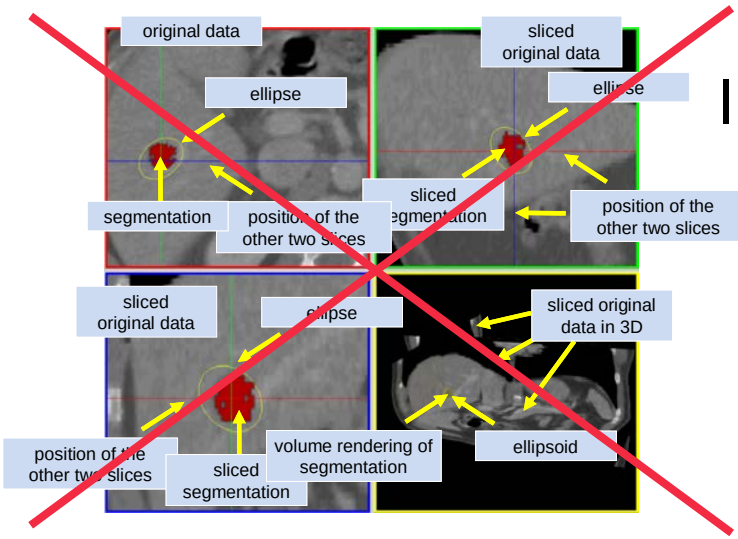


➔ synchronization of …

20 objects in
4 scene-graphs

… required !

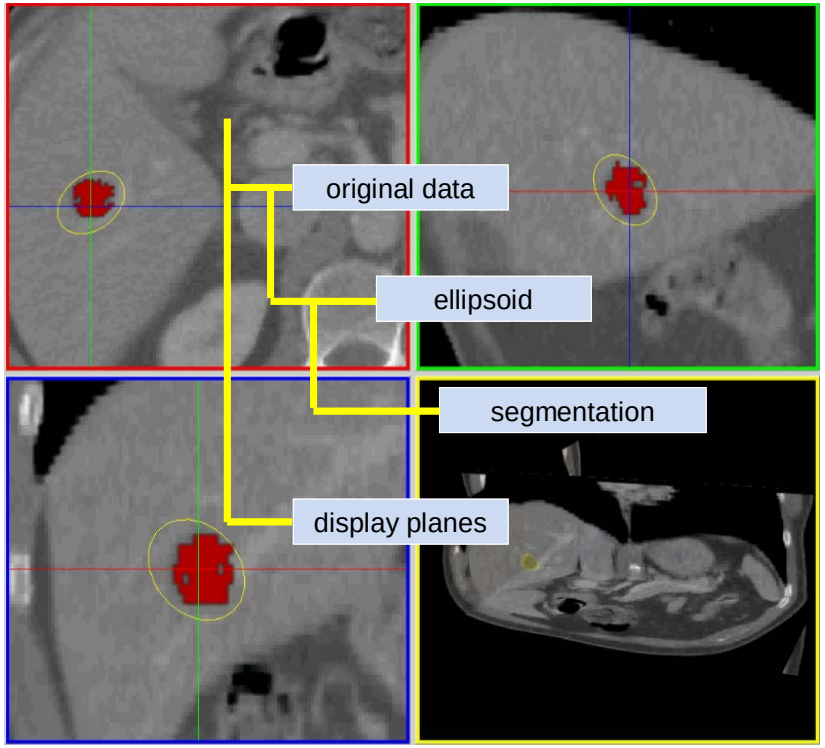
Getting out of the maze …



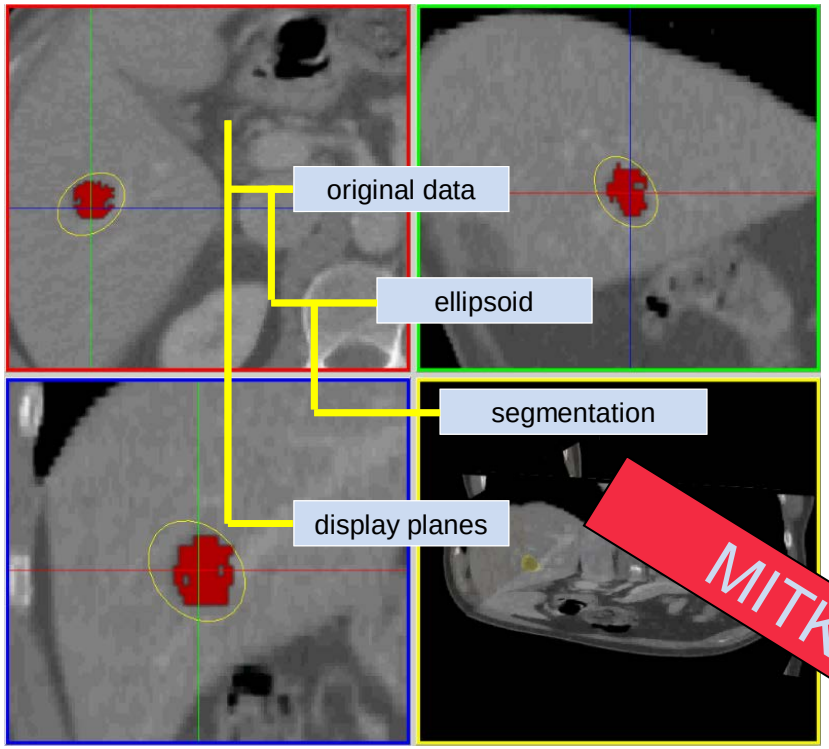
Instead of creating **many** scene-graphs with **even more** elements …

… create a **single data repository**

with a **few data-objects**!



MITK: Data repository instead of scene-graphs

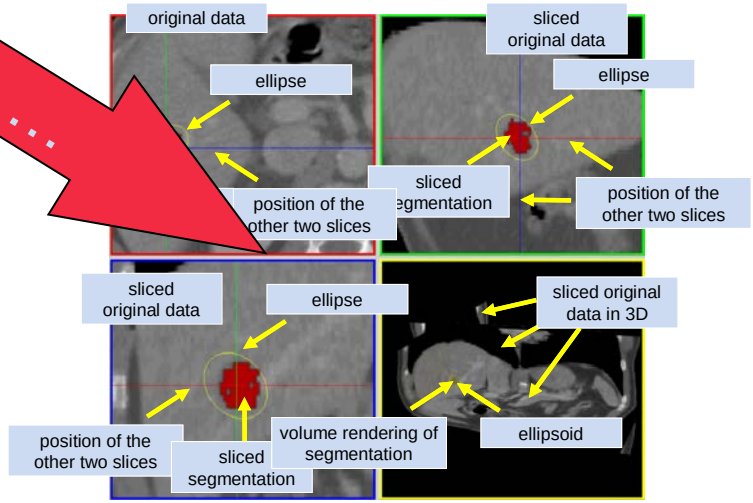


MITK takes the data repository …

and builds …

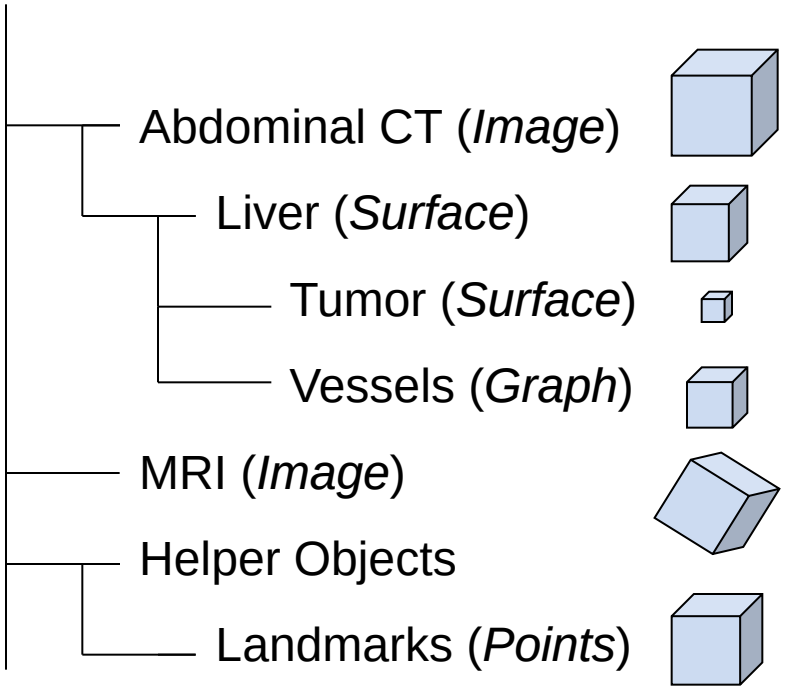
➔ VTK scene graphs

MITK creates …

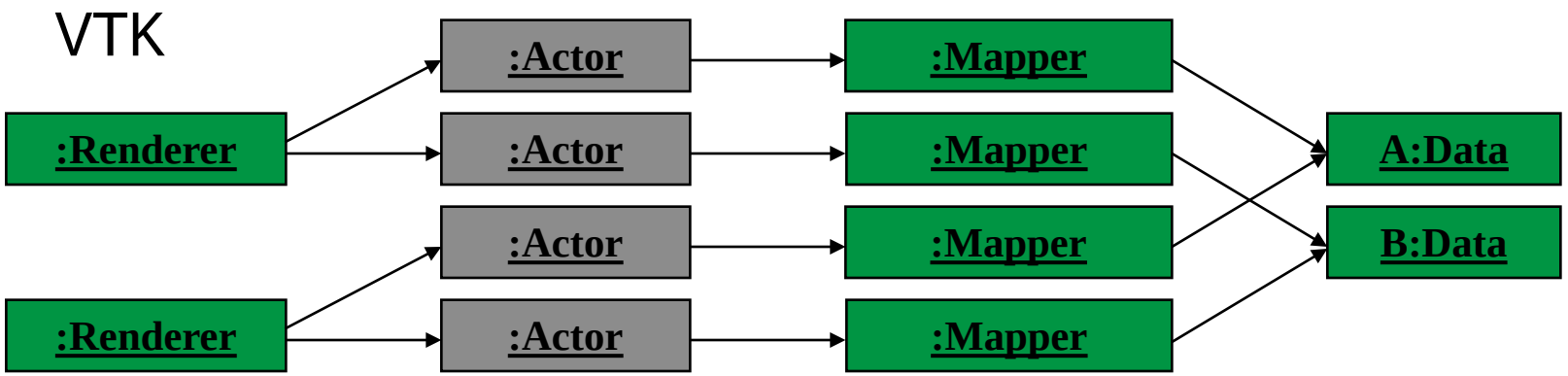


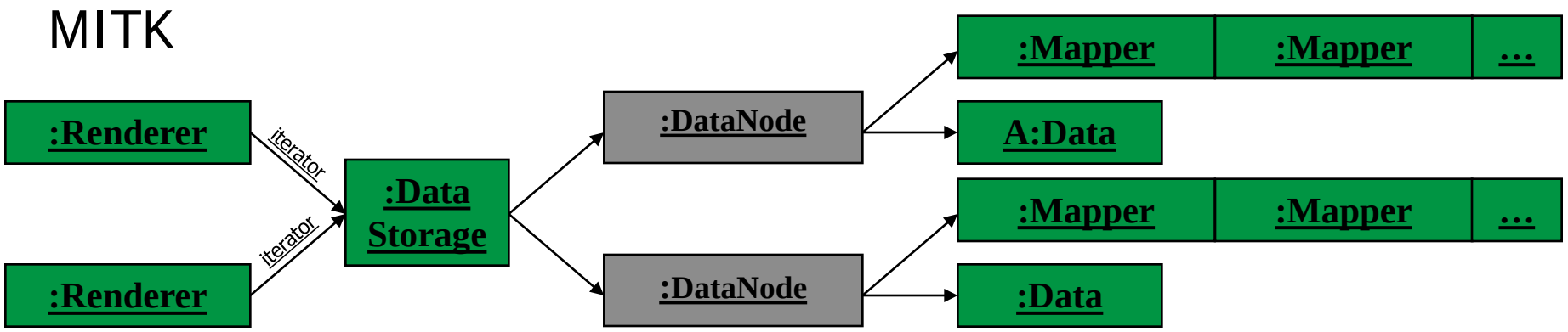
Data repository

- Repositories for sharing data objects between modules
- Any number of data objects
- Any kind of data objects
- Data objects with geometry frame (bounding-box, transform, etc.)

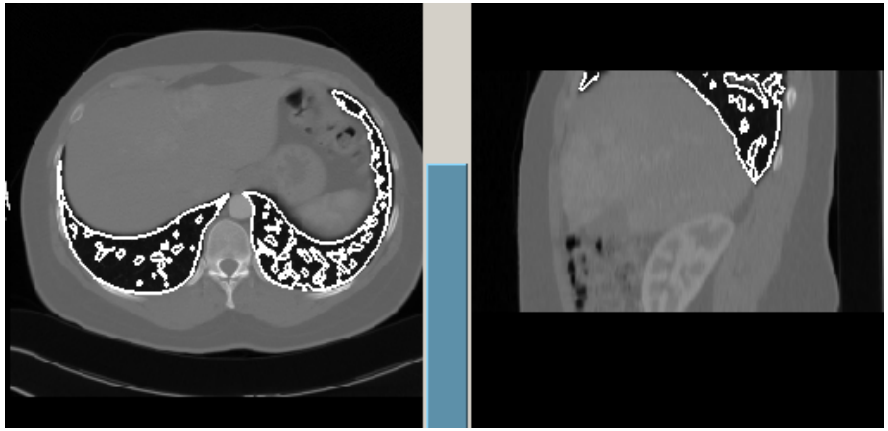


Rendering VTK vs. MITK





Rendering the data repository



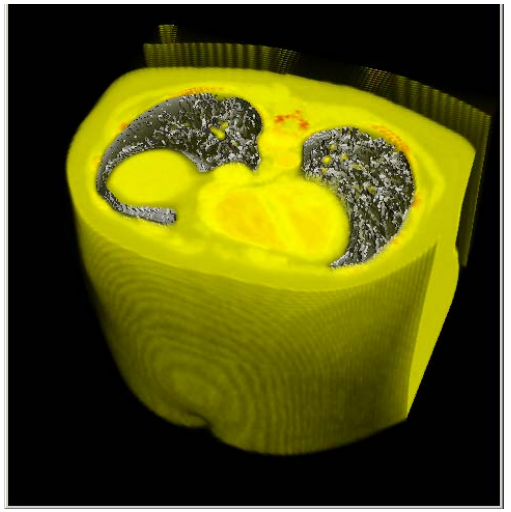
RenderWindows:

- **single** RenderWindow class
- **different types** of views
  - ➔ 2D/3D
  - ➔ special views definable (e.g., for AR)

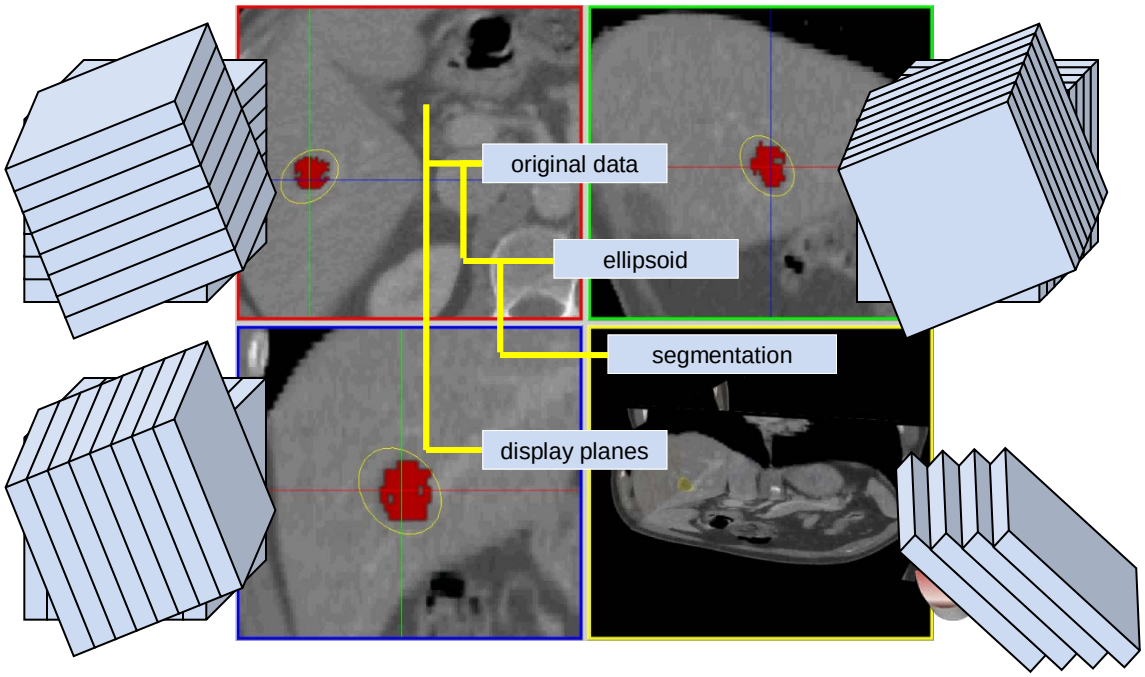
```
renderer->SetMapperID(mitk::BaseRenderer::Standard3D);
```

- **point** to the **data repository**
  - ➔ **any number of views** on the data:

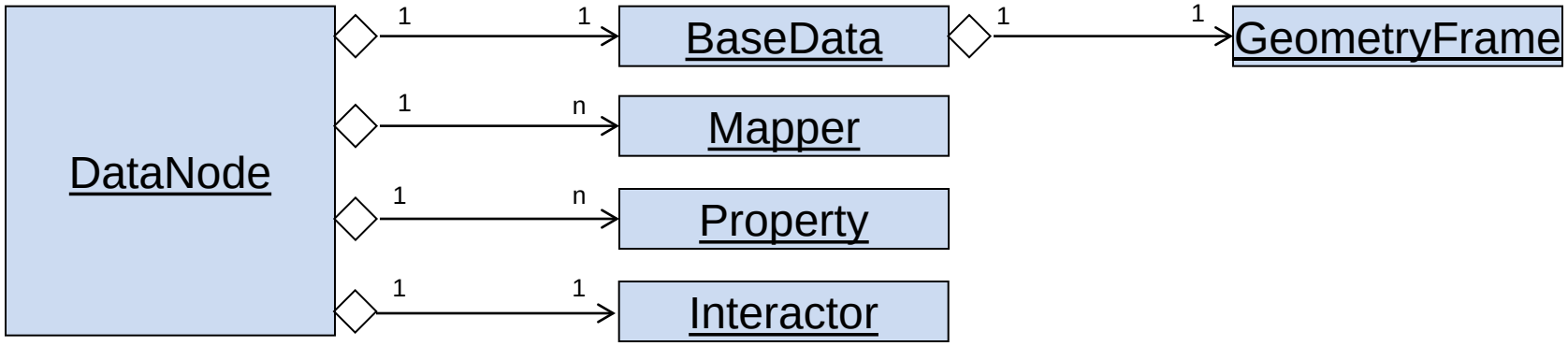
```
renderer1->SetData(repository);
renderer2->SetData(repository);
...
```



Defining
how we want to see the data …



The nodes in the data repository



BaseData: the actual data: images, surfaces, etc.

GeometryFrame: position and orientation in space

Mappers: render the data into a renderwindow

Properties: define how to draw the data

Interactor: defines user interaction with the data

How to add a new data type

Extension for new data types:

➔ derive data class

➔ derive mapper

➔ create file I/O

➔ Register mapper / I/O handler at factory

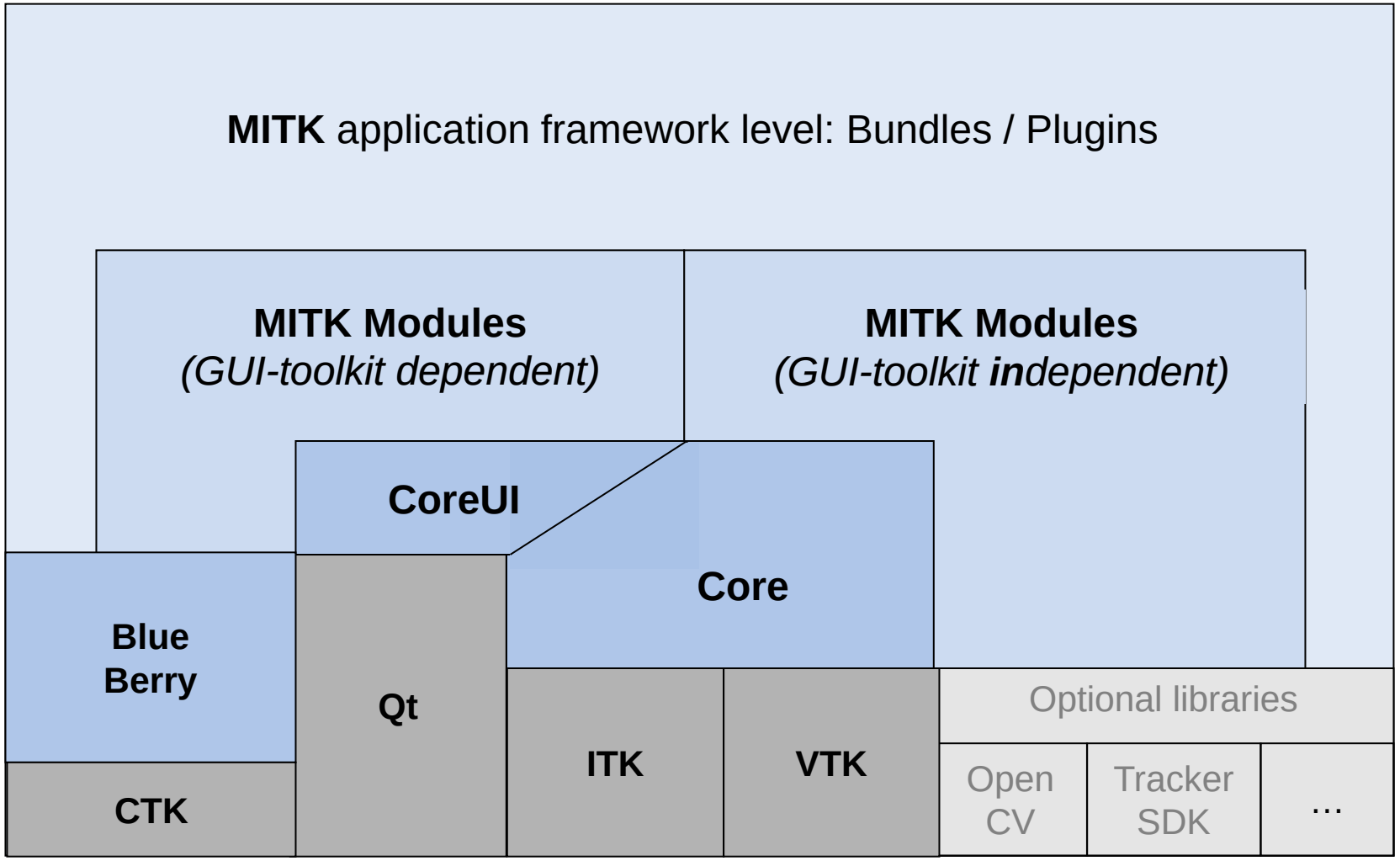
Example:

- attributed vessel graphs

[DKFZ and University of Tübingen]

MITK Architecture

MITK Architecture



MITK Core

- Access to ITK and VTK data structures and algorithms
- Data Management (data object container, properties, scene management)
- Time steps for data objects
- Spatial object location (geometries)
- Loading / saving of different file formats
- Rendering (mappers, update management, render properties)
- Interaction (statemachine based)
- Undo/Redo

MITK Modules

- Diffusion Imaging
- IGT
- Ultrasound
- ToF
- GPGPU
- Registration
- Segmentation
- Qt Widgets

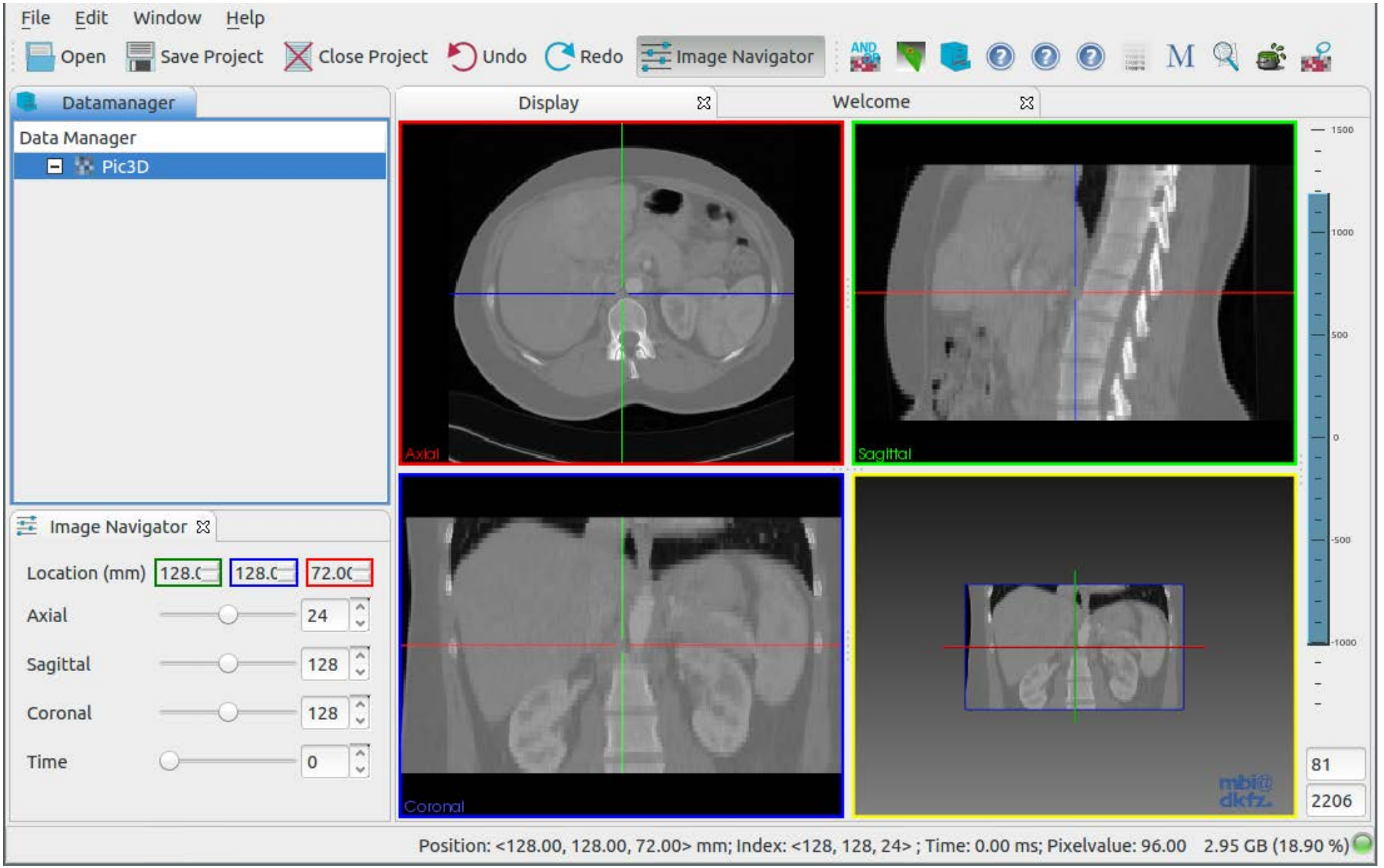
- Many more…

Application framework

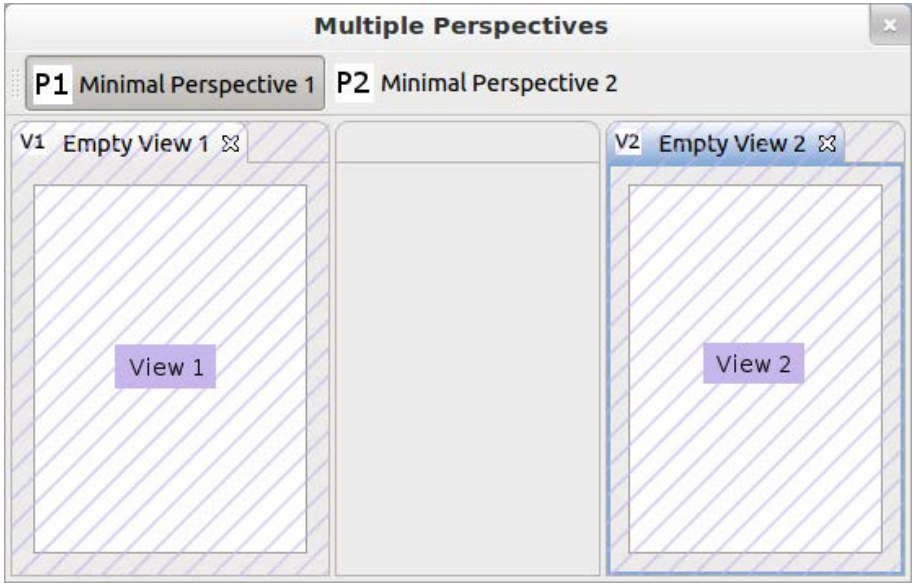
The MITK application framework is based on BlueBerry:

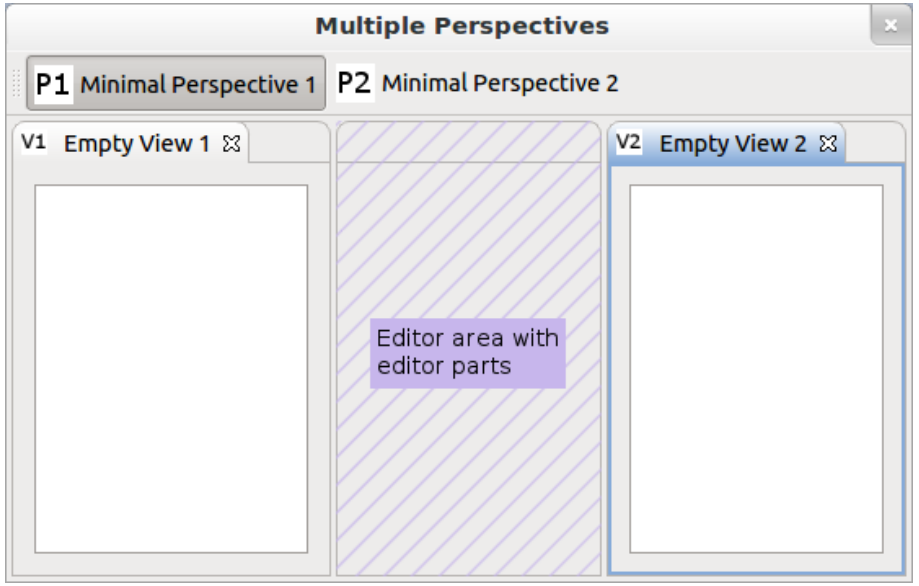
- Framework for creating modular and extensible end-user apps
  - Customizable application frame
  - Application state can be saved and restored on next startup
  - Independent „plug-Ins“ for specific problems
- The *MITK Workbench* is the main MITK application
  - Bundles a set of useful general purpose plug-ins
  - Shared repository for data objects

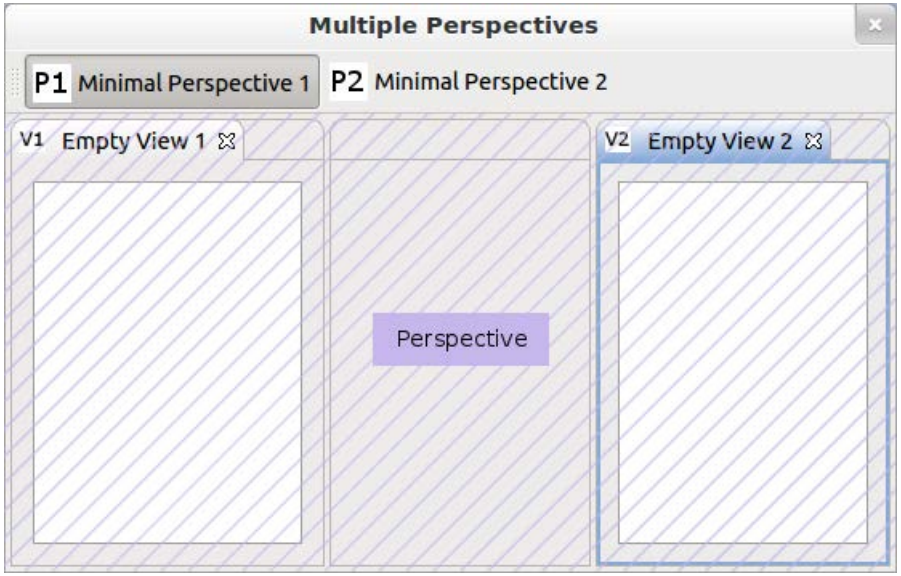
MITK Workbench



MITK Workbench – Views and Editors

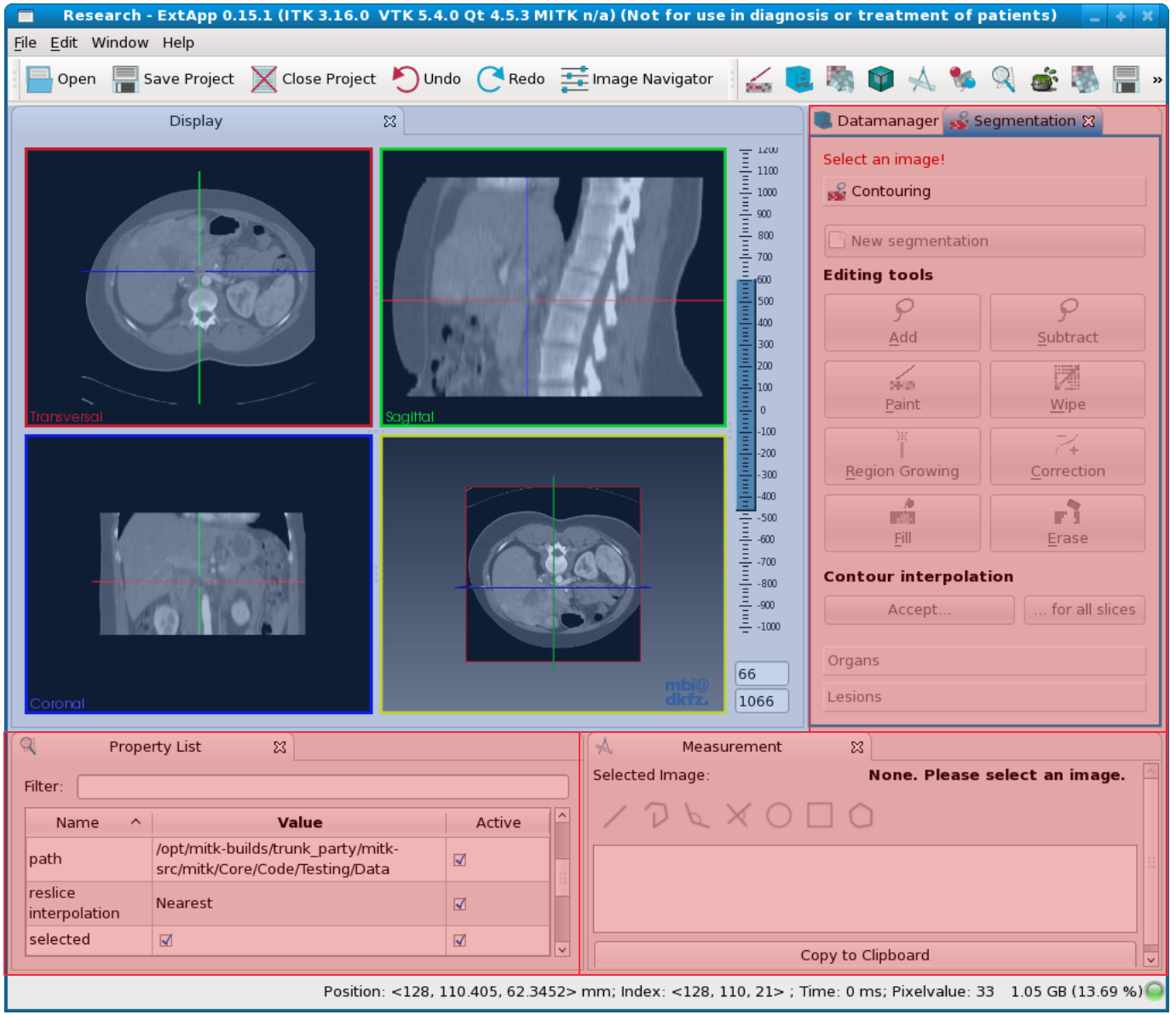






MITK Workbench – Views and Editors

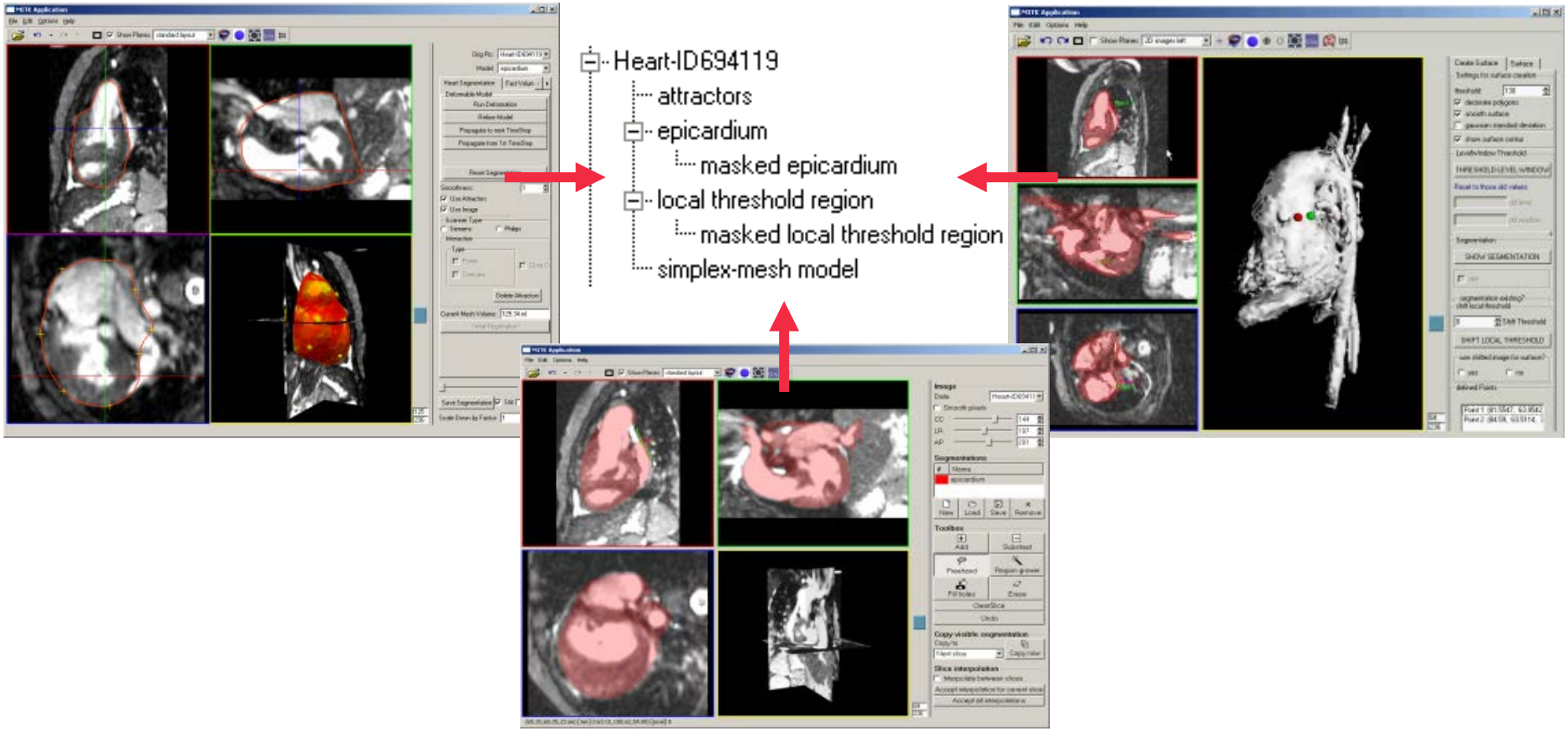
Editors



Views

Communication and sharing data

- Views/Editors are usually independent from each other
- They share data via the data repository
- Make use of the BlueBerry selection service



ITK, VTK, MITK: Gemeinsamkeiten

Gemeinsamkeiten

- Toolkits
- Objekt-orientierte Klassenbibliotheken
- C++
- Unterstützung vieler Compiler
- Plattform unabhängig
- GUI-Toolkit unabhängig
- Open source / BSD-style Lizenz

Gemeinsamkeiten: Überblick

- Download und Build-Prozess
- Instantiierung und Pointer
- Daten-Pipeline
- Kommunikation

Download und Build

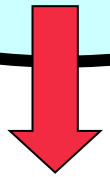
Voraussetzungen

C++ Compiler:

GCC 4.x
VC9
VC10
(Clang 3.x)

Plattform-unabhängiges Make-Tool:

CMake



Erzeugt Projekt bzw. Makefiles für die eingesetzte Plattform/den verwendeten Compiler/die verwendete Entwicklungsumgebung.

CMake

- Skriptsprache zur Beschreibung der Bestandteile eines (C++)-Projekts
- Getrennte Dateibäume für Quelltexte und Binärdateien
- Plattformen:
  Windows (Visual Studio, Borland, MinGW …), Linux, Mac OS X, Eclipse …
- Erzeugt spezifische Projekt/Makefiles (nicht nur) für die Übersetzung der Programme und Bibliotheken
- Auch: Testgenerierung und -steuerung, Aufruf von Doxygen, LaTeX , Erstellung von Installern …

CMake Befehle

Variablen:

- **set**(VAR [VALUE])
- **list**(APPEND VAR “NochEinWert”)

Output:

- **message**(“Wert von VAR: ${VAR}”)

Ausführbares Programm erstellen:

- **add_executable**(MeinProg Quelle1.cpp Quelle2.cpp)

Bibliothek erstellen:

- **add_library**(MeineLib LibQuelle1.cpp …)

- **target_link_libraries**(MeinProgramm MeineLib)
- **include_directories(**<Pfad für c++ header>)

CMake Makros

Makros definieren

macro(NAME parameter1 parameter2 …)
…
endmacro()

In MITK z.B.:

MITK_CREATE_MODULE()
MITK_CREATE_CTK_PLUGIN()

Definiert in mitk/CMake/*.cmake

CMake Modularisierung

- Hauptdatei heisst immer „CMakeLists.txt“
- **include**(anderesCMakeFile.cmake):
  direktes Einbinden der anderen Datei an dieser Stelle
- **add_subdirectory**(directory):
  „directory“ muss wieder CMakeLists.txt enthalten. Wird meistens für Unterprojekte verwendet.
- In MITK für „Anwender“: files.cmake in (fast) jedem Verzeichnis mit Quelltexten

CMake in MITK: files.cmake

```
set( CPP_FILES
  TutorialFunctionality.cpp
)

set( MOC_H_FILES
  TutorialFunctionality.h
)

set(UI_FILES
  TutorialFunctionalityControls.ui
)
```

- CPP_FILES: C++ Quelltexte
- MOC_H_FILES: Q_OBJECT Header, die vom Meta-Object-Compiler moc vorübersetzt werden müssen
- UI_FILES: vom Qt-Designer generierte Forms

Instantiierung von Klassen und Pointer

Instantiierung

Instantiierung von ITK/VTK/MITK-Klassen:

Statt

~~new className;~~

className::New();

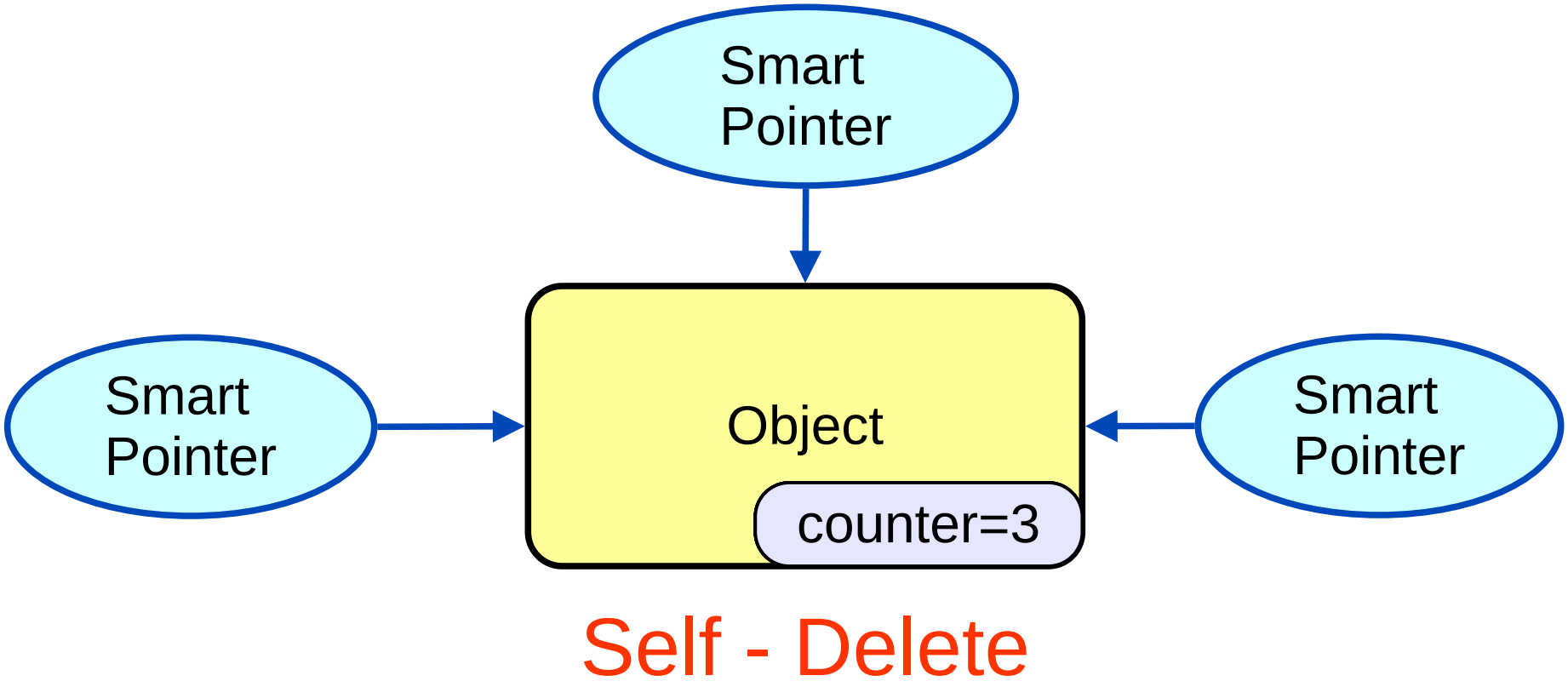
Instantiierung

```
//VTK:
vtkSmartPointer<vtkRenderer> renderer =
  vtkSmartPointer<vtkRenderer>::New();
//ITK/MITK:
itk::Command::Pointer command =
  itk::Command::New();
```

ACHTUNG:

“*” statt “::Pointer” *bei Instantiierung* führt sehr bald zum Absturz!!

::Pointer ist ein Smart Pointer



Daten-Pipeline

Daten-Pipeline

Objekt-orientierter Ansatz:

- Daten-Klassen

  und

- Algorithmen-Klassen

➔ Algorithmen nicht als Funktionen, sondern als Klassen realisiert !

Daten-Pipeline

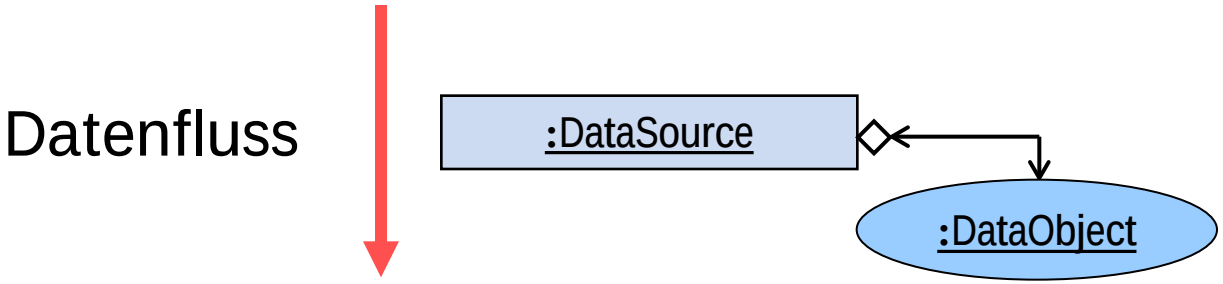
Algorithmen-Klassen:

- Oberbegriff: **ProcessObject**
- Daten-Erzeuger: **Source**
- Daten-Verarbeiter: **Filter**
  (sind somit ebenfalls Sourcen)

Daten-Pipeline

Sourcen **besitzen** ihre Ausgabe-Daten-Objekte!

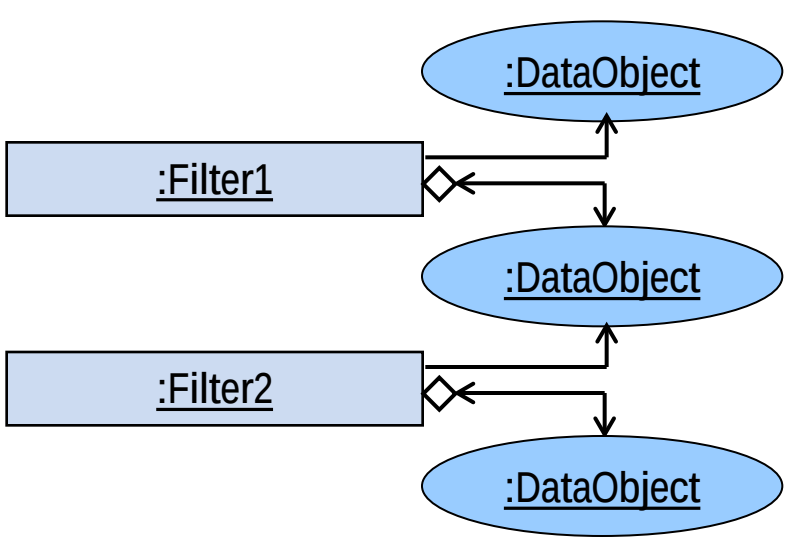
Umgekehrt **kennt** das Ausgabe-Daten-Objekte **seinen Erzeuger** (Source)!



Daten-Pipeline

Filter **kennen** zudem ihre Eingabe-Daten-Objekte:

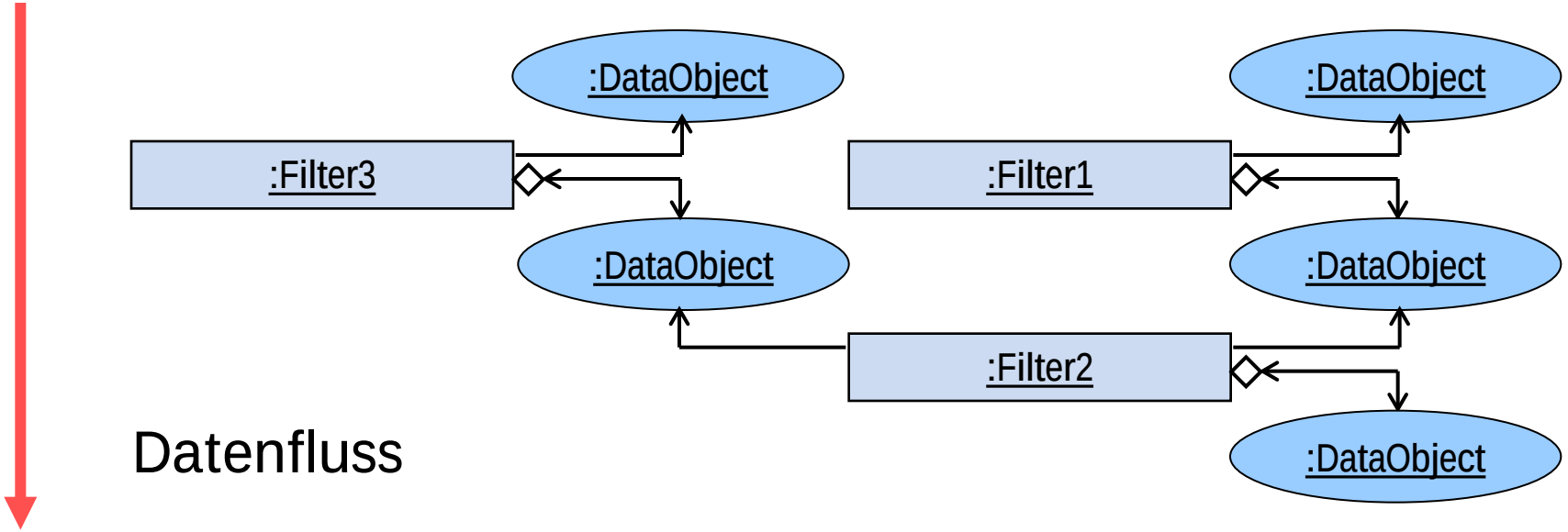
Datenfluss



➔ **Daten-Pipeline!!**

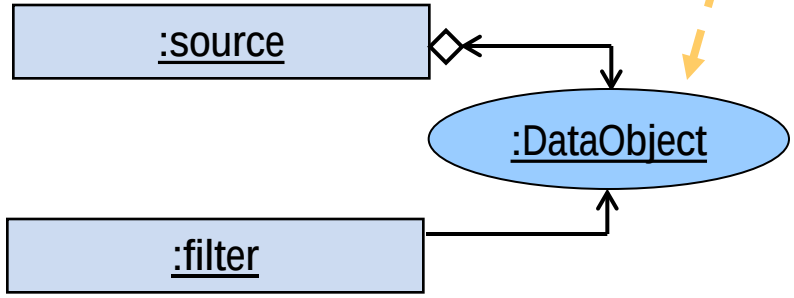
Daten-Pipeline

Filter können auch **mehrere** Eingabe-/ Ausgabe-Daten-Objekte haben:



Verbinden von Filtern

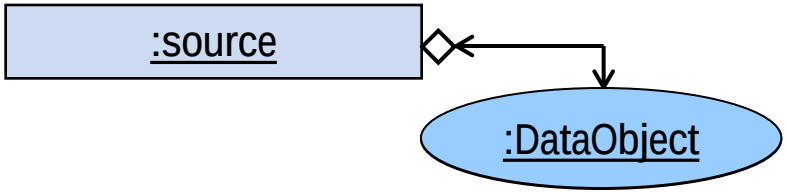
```
SourceType::Pointer source = SourceType::New();
FilterType::Pointer filter = FilterType::New();
filter->SetInput(source->GetOutput());
```



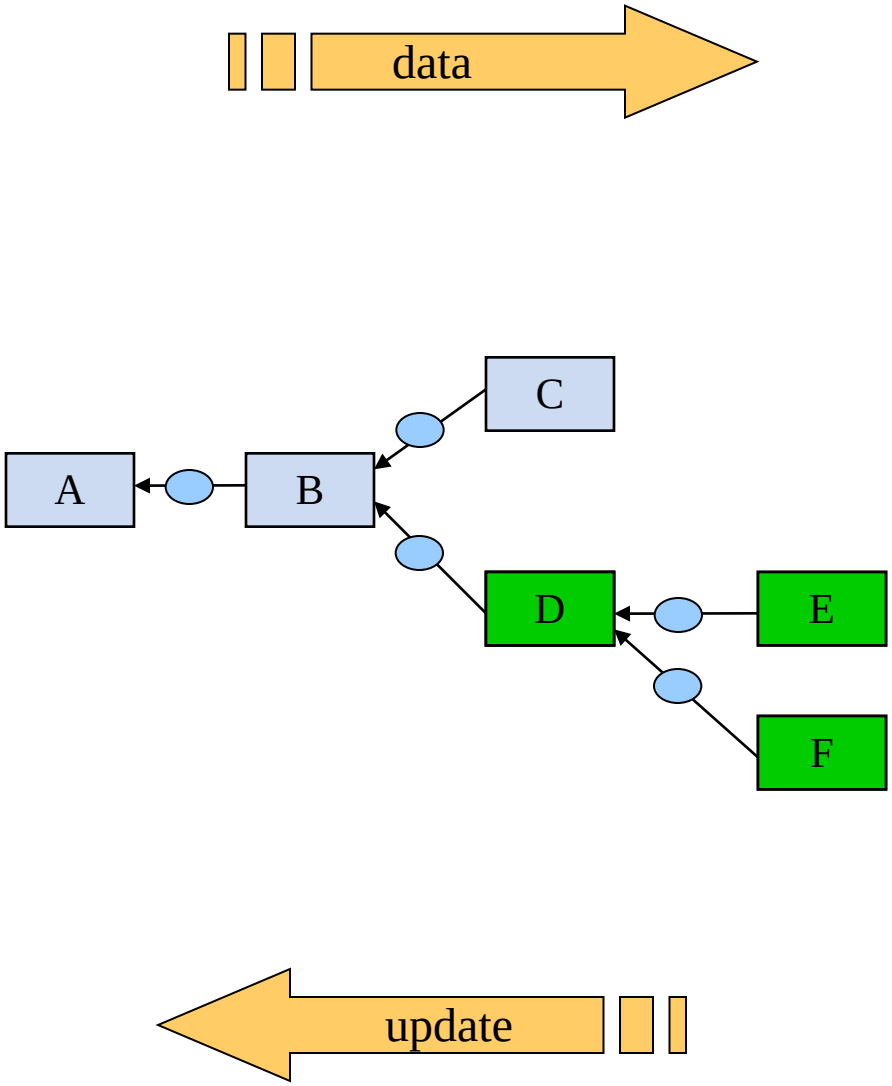
➔ ***Daten-Objekte meist “unsichtbar”!***

Update/Leere Daten-Objekte

```
SourceType::Pointer source = SourceType::New();
// Achtung: Daten-Objekt ist zunächst leer!!
// Garantie für aktuelles Daten-Objekt: Update()!
source->Update();
```



Pipeline update …



➔ Keine unnötigen Neuberechnungen !!

Adaptor Classes: Connecting ITK, VTK data classes to MITK

Access MITK images as ITK images

- ITK images are templated
- mitk::Image is not templated

```
// access method
template < … >
MyAccessMethod( itk::Image<…>* itkImage, … )
{
…
}

// calling the access method
AccessByItk(mitkImage, MyAccessMethod, …)
```

Example code in mitk/Examples/Tutorial/Step6.cpp

Converting images ITK⇔MITK

mitk::Image to itk::Image cast:

```
mitk::CastToItkImage(mitk::Image, itk::Image<…>&)
```

- converts data type if neccessary
- otherwise references memory of data array
- dimension of itk::Image must equal dimension of mitkImage

itk::Image<> to mitk::Image cast:

```
mitk::Image::Pointer
mitk::ImportItkImage(const itk::Image<…>&)
```

- references memory of itk::Image

Connecting MITK images to VTK

… not a conversion, just accessing:

```
vtkImageData*
mitk::Image::GetVtkImageData(int time=0)
```

Surfaces MITK ⇔ VTK

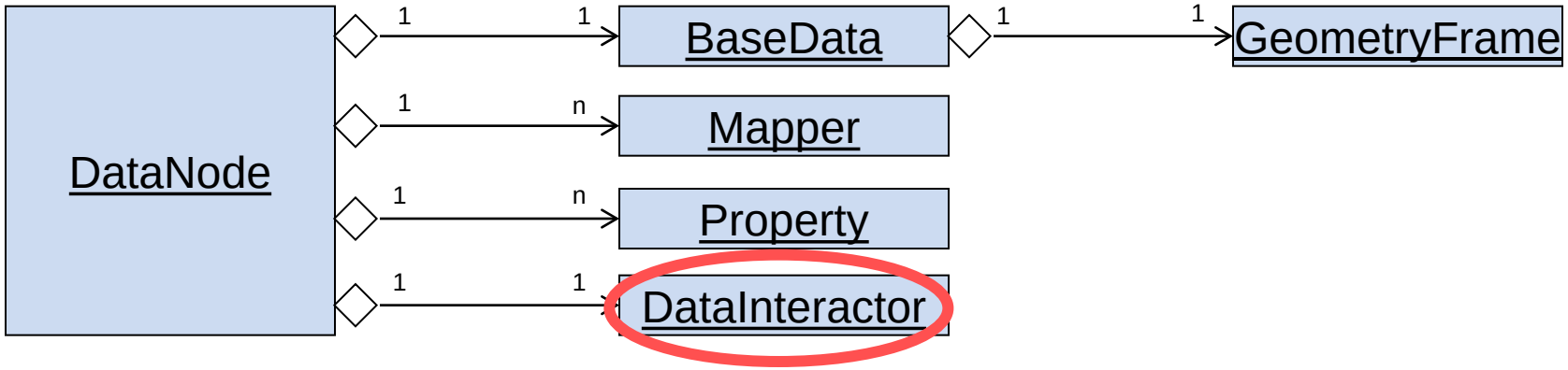
Again: not a conversion, just accessing:

```
vtkPolyData*
mitk::Surface::GetVtkPolyData(int time=0)
```

```
mitk::Surface::SetVtkPolyData
  (vtkPolyData*, int time=0)
```

Interaction and Undo

Interactors



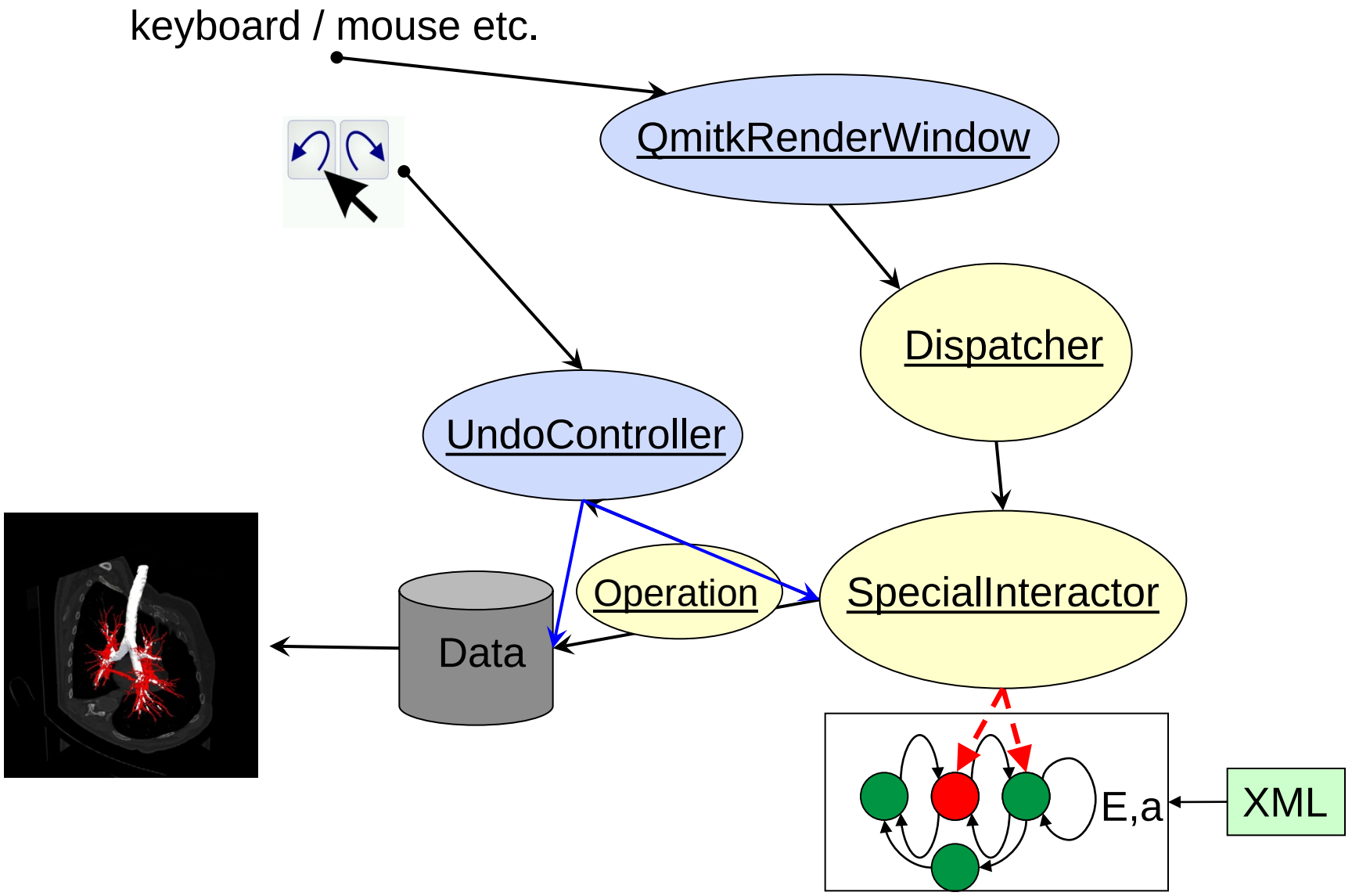
Interactors:

➔ behavior defined in state-machines

➔ undo-/redo concept

➔ dimension/geometry-independent definition:
(often) identical interaction code for 2D and 3D

Interaction Scheme



Thank you!

Questions?

MITK
Modular Development

Sascha Zelzer

Division of Medical and Biological Informatics, DKFZ Heidelberg

dkfz.
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Levels of Modularity and Interoperability

Modularity refers to the logical decomposition of a large system into smaller collaborating pieces.

General modularity concepts:

- Reusable executables (legacy code)
- Shared libraries
- Reusable software components as “Services”
- Library / Component / Service repositories

Levels of Modularity and Interoperability

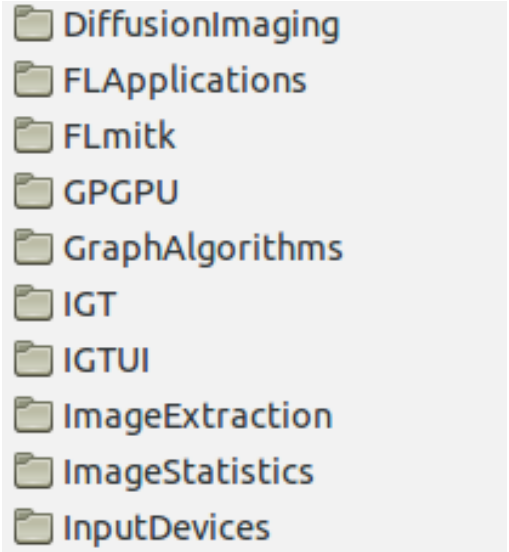
MITK supports different “modularity approaches”

- MITK Modules (shared libraries)
- MITK Micro Services (Modules + Service Registry)
- CTK Plug-ins (OSGi concepts)
- BlueBerry Application Framework (think of Eclipse RCP)
- Command Line Interface (CTK)

MITK Modules

MITK Modules

- Shared libraries
- MITK CMake macros for easy creation and orchestration
- Automatic dependency checking and error reporting

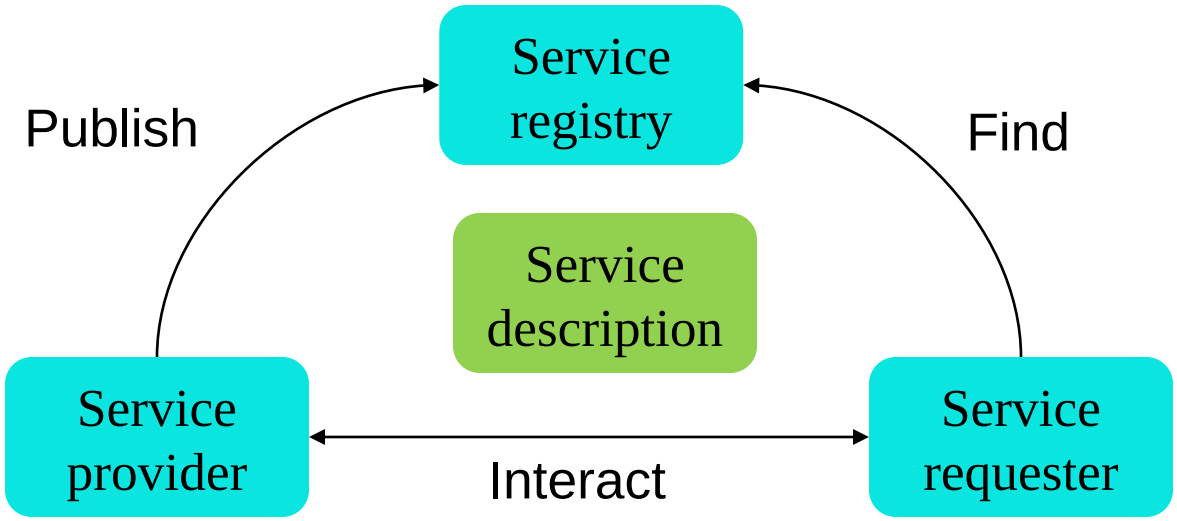



```
SET(CPP_FILES
  mitkImageStatisticsCalculator.cpp
  …
)

MITK_CREATE_MODULE(ImageStatistics
  DEPENDS Mitk ImageExtraction PlanarFigure)
```

MITK Micro Services

MITK modules can publish services and bind services from other modules:



API is very close to the OSGi Service Layer specs.

MITK Micro Services

- Think of a service as: “work done for another”
- Services imply a contract between the provider of the service and its consumers.

A service-oriented approach promotes:

- Less coupling between providers and conusmers leading to higher reuse of components
- More emphasis in interfaces
- Clear description of dependencies
- Support for multiple competing implementations
- Powerful service look-up mechanisms based on service properties and a LDAP query language

MITK Micro Services

Service Interface

```
struct MyService {
virtual ~MyService();
virtual void DoSomething()=0;
};

US_DECLARE_SERVICE_INTERFACE
(MyService,“org.ms.mysrv/1.0”)
```

Service provider

```
class MyServiceImpl :
public MyService {
  void DoSomething(){…};
};
```

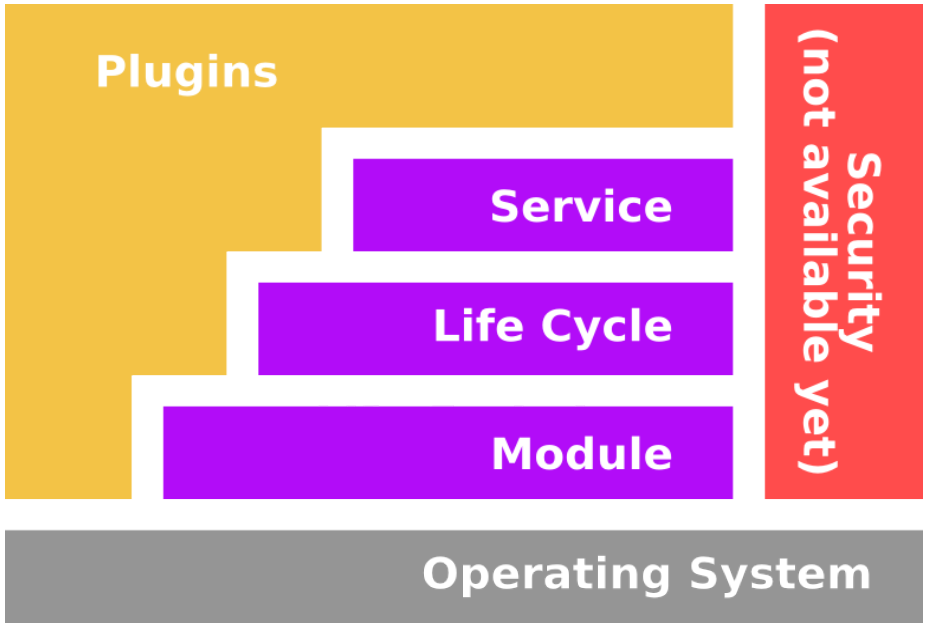
```
mitk::ModuleContext* mc = mitk::GetModuleContext();
MyService* myService = new MyServiceImpl();
mc->RegisterService<MyService>(myService);
```

```
mitk::ServiceReference ref =
  mc->GetServiceReference<MyService>();
if (ref) {
 mc->GetService<MyService>(ref)->DoSomething(); }
```

CTK Plugin Framework

Plugin Framework

- Dynamic Plugin Framework (based on OSGi)
- Enables service oriented architectures
- CTK provides basic plugins for distributed/large-scale applications



Plugin Framework

Highlighted Features:

- **Reduced complexity**

  plug-ins hide their internals and communicate through well-defined services
- **Adaptive**

  supports optional and dynamic service dependencies
- **Versioning**

  plug-ins and their dependencies are versioned
- **Simple**

  the API is surprisingly simple
- **Non intrusive**

  no special interface required for services

OSGi Specifications

- OSGi Core Specifications are small
- OSGi Service Compendium defines many optional services:

Implemented OSGi specifications in CTK

- Log Service Specification

  Provides a general purpose message logger.
- Metatype Service Specification

  Provides a unified way to describe metadata about services.
- Configuration Admin Service Specification

  Allows to set the configuration information of deployed plugins.
- Event Admin Service Specification

  Inter-plugin communication mechanism based on a event publish and subscribe model.

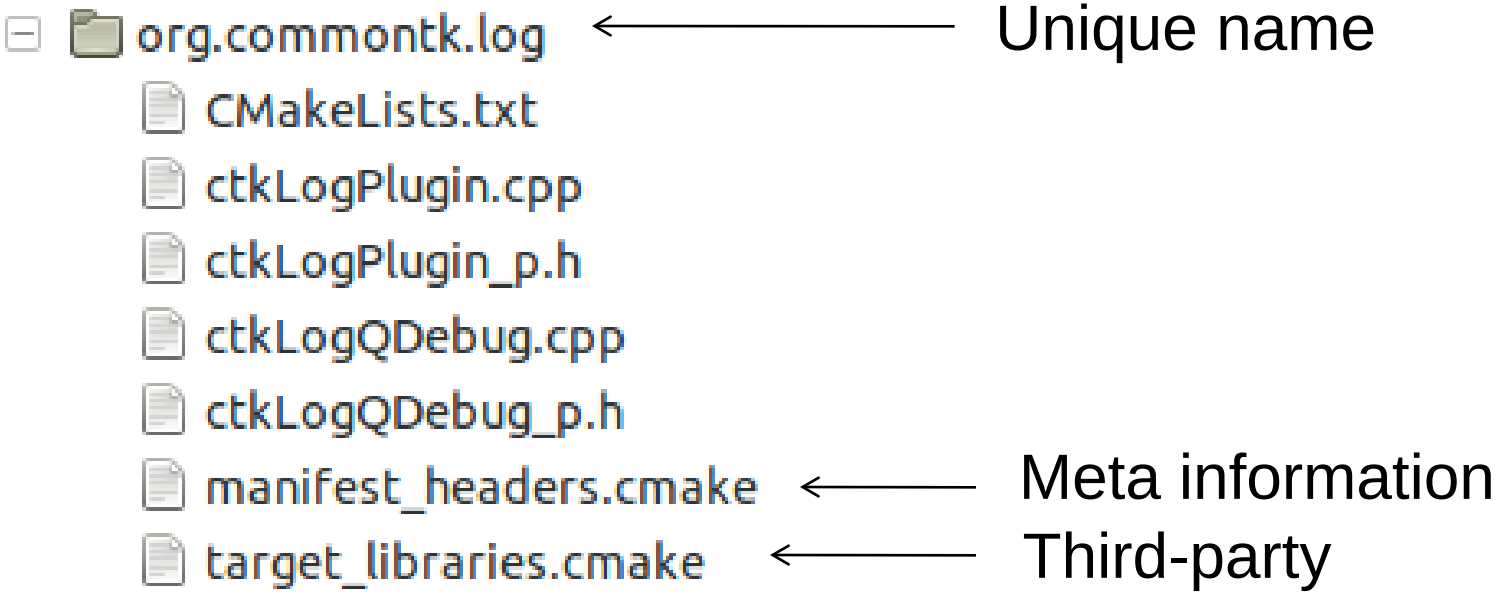
Event Admin

- Event publisher: sends events related to a specific topic
- Event handler: expresses interest in one or more topics

Features

- Synchronous or asynchronous event delivery
- Event from different threads are sent in parallel
- Event handler blacklisting

Plug-in structure



OSGi meta information

manifest_headers.cmake

```
set(Plugin-Name "A human readable name for your plug-in")
set(Plugin-Version "x.x.x")
set(Plugin-Vendor "A human readable name for the vendor of the plug-in")
set(Plugin-ContactAddress "Could be a web page, a email adress, etc.")
set(Require-Plugin <list-of-plugin-symbolic-names>)
```

BlueBerry Application Framework

BlueBerry

BlueBerry

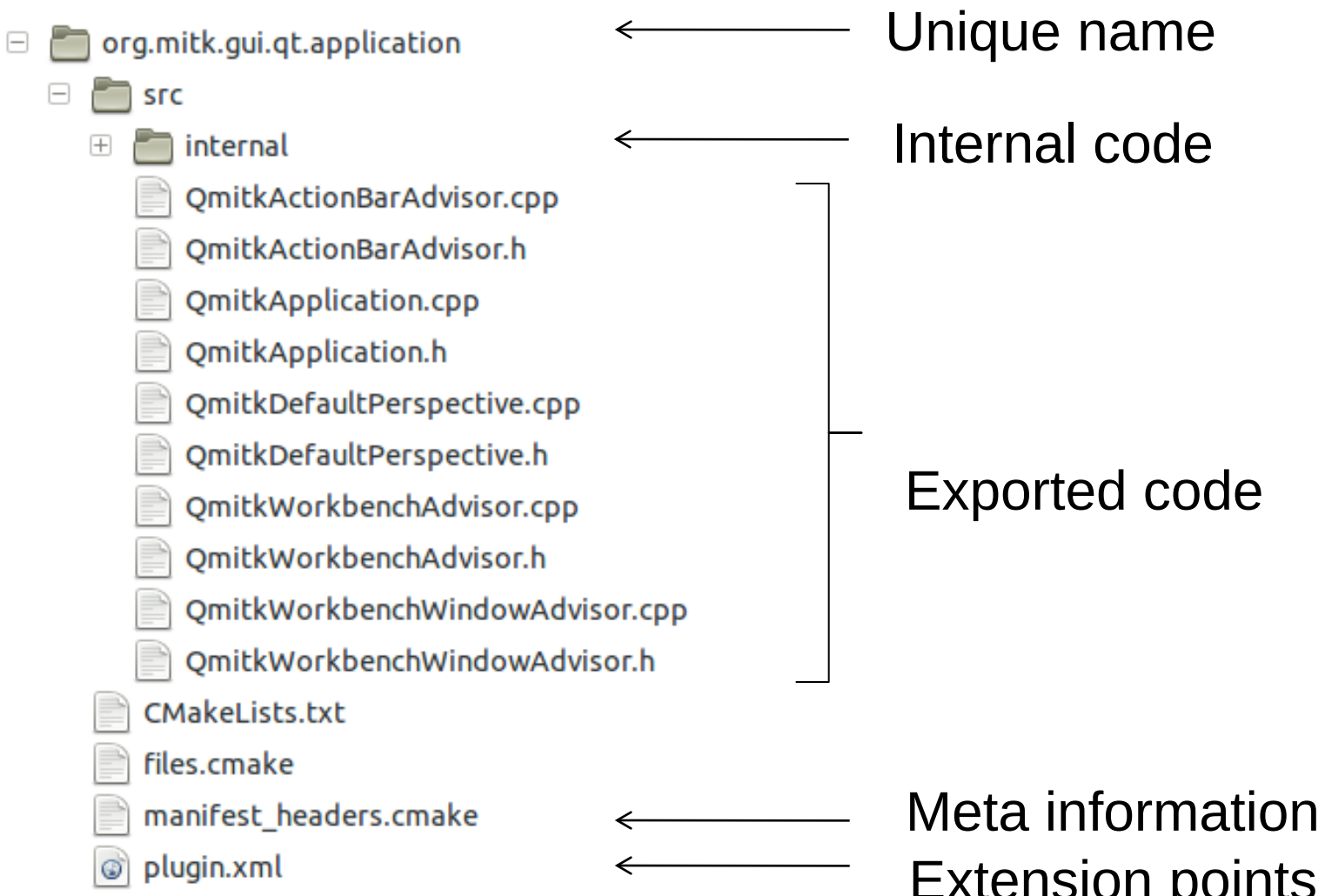
- Application framework based on CTK plug-ins
- Design and extensibility inspired by the Eclipse RCP
- Shipped with MITK, but it is indepenent of MITK

What is BlueBerry?

BlueBerry Features

- Provides an application framework similar to the Eclipse RCP in C++
- Based on the CTK plug-in model
- Solutions for lazy loading of plug-ins (scalability)
- Flexible extension mechansims
- Provides general purpose plug-ins for extensible applications

Plug-in structure



Define extensions

plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
  <extension point="org.blueberry.osgi.applications">
    <application id="org.mitk.qt.application">
      <run class="QmitkApplication"/>
    </application>
  </extension>

  <extension point="org.blueberry.ui.perspectives">
    <perspective
      id="org.mitk.coreapp.defaultperspective"
      name="Core App Home"
      class="QmitkDefaultPerspective">
    </perspective>
  </extension>

</plugin>
```

Registering executable extensions

MyPluginActivator.cpp

```
#include "QmitkPropertyListView.h"
#include "../QmitkDataManagerView.h"
#include "../QmitkDataManagerPreferencePage.h"
#include "../QmitkDataManagerHotkeysPrefPage.h"

namespace mitk {

void PluginActivator::start(ctkPluginContext* context)
{
  BERRY_REGISTER_EXTENSION_CLASS(QmitkDataManagerView, context)
  BERRY_REGISTER_EXTENSION_CLASS(QmitkPropertyListView, context)
  BERRY_REGISTER_EXTENSION_CLASS(QmitkDataManagerPreferencePage, context)
  BERRY_REGISTER_EXTENSION_CLASS(QmitkDataManagerHotkeysPrefPage, context)
}
```

Conclusion

Advantages

- An application made out of plug-ins
- Stronger encapsulation & loose coupling
- Exchangeable software modules
  - CTK services
  - BlueBerry extension points
- A bunch of concepts for creating Rich Client applications
  - Flexible application layout
  - Reusable „views“ and programmable „perspectives“
- A lot of MITK imaging plug-ins

Conclusion

Costs

- Writing XML files for extension points
- Code for tracking services (they can come and go as they want)
- Little overhead for the plug-in management
- Components are tied to the BlueBerry application framework

Command Line Interface

Command Line Interface

- Integration of (legacy) stand-alone command line programs
- Executables must provide an XML file describing the command line parameters
- Allows (limited) communication via standard input/output channels

Examples

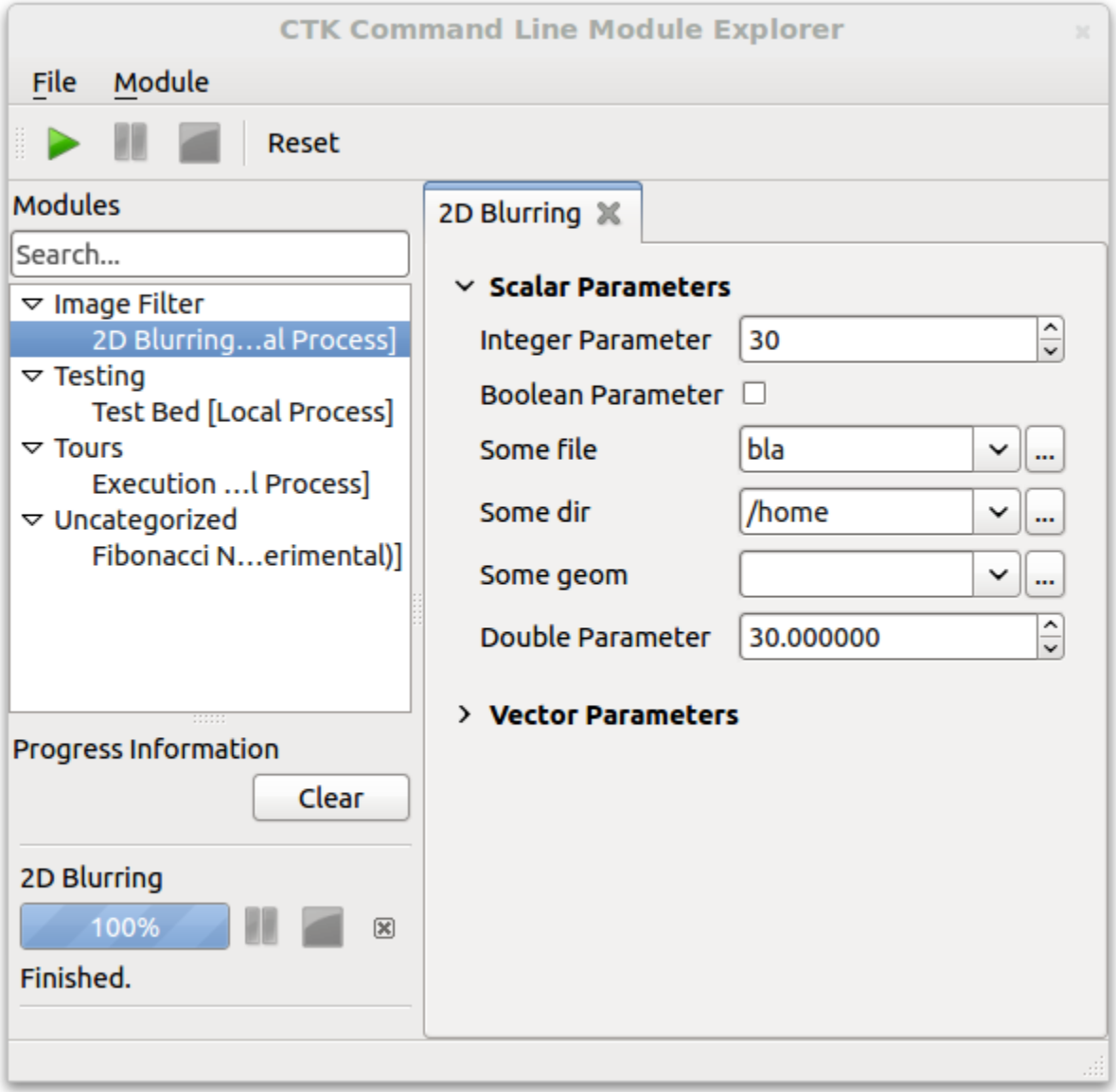
- ITK based registration programs
- Compiled Matlab code
- Slicer CLI modules

```
<?xml version="1.0" encoding="utf-8"?>
<executable>
  <category>Image Filter</category>
  <title>2D Blurring</title>
  <description>
Blur a 2d image.
  </description>
  <version>1.0</version>
  <documentation-url></documentation-url>
  <license></license>
  <contributor>Sascha Zelzer</contributor>

  <parameters>
    <label>Scalar Parameters</label>
    <description>
    Variations on scalar parameters
    </description>
    <integer>
      <name>integerVariable</name>
      <flag>i</flag>
      <longflag>integer</longflag>
      <description>
      An integer without constraints
      </description>
      <label>Integer Parameter</label>
      <default>30</default>
    </integer>
    <boolean>
      <name>booleanParam</name>
      <flag>b</flag>
```

Command Line Interface

- Automatic GUI generation based on the XML description
- Command line module explorer in CTK
- MITK integration via org.mitk.gui.qt.cmdline modules



Questions

dkfz.

Thank you!

Questions?

MITK projects

Marco Nolden

Medical and Biological Informatics
German Cancer Research Center, Heidelberg (D)

dkfz. DEUTSCHES KREBSFORSCHUNGSZENTRUM IN DER HELMHOLTZ-GEMEINSCHAFT

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

dkfz.

MITK GPGPU programming with OpenCL

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

MITK OpenCL Module

dkfz.

- Provides basic classes for comfortable OpenCL usage
- Primary aim: image processing filter
  - ITK-like filter interface
  - Filter pipeline implementation
    - special methods for two GPU filter
    - *on-demand* data copy
- Follows the structure of OpenCL:

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Module Setup

- Requirements:
  - OpenCL capable device
    - NVIDIA GeForce 8600 and newer
    - ATI Radeon 4600 and newer
  - OpenCL library, OpenCL header
    (available through an OpenCL SDK)
- Configuration:
  - CMake Flag: MITK_USE_OPENCL
  - Extra Flags: MITK_OPENCL_INC, MITK_OPENCL_LIB
- Module Usage:
  - module name mitkOcl

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

MITK OpenCL Module : Structure

dkfz.

Data Objects

- **OclImage**
  - Encapsulates mitk::Image

- **OclMemoryObject**
  - generic memory object
  - holds an void* buffer and corresponding cl_mem pointer pointing to (global) graphics memory

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

MITK OpenCL Module : Structure

dkfz.

Specialized objects

- **OclContextManager**
  - singleton class
  - creates/destroys context
  - `OclContextManager::GetInstance()->GetContext();`

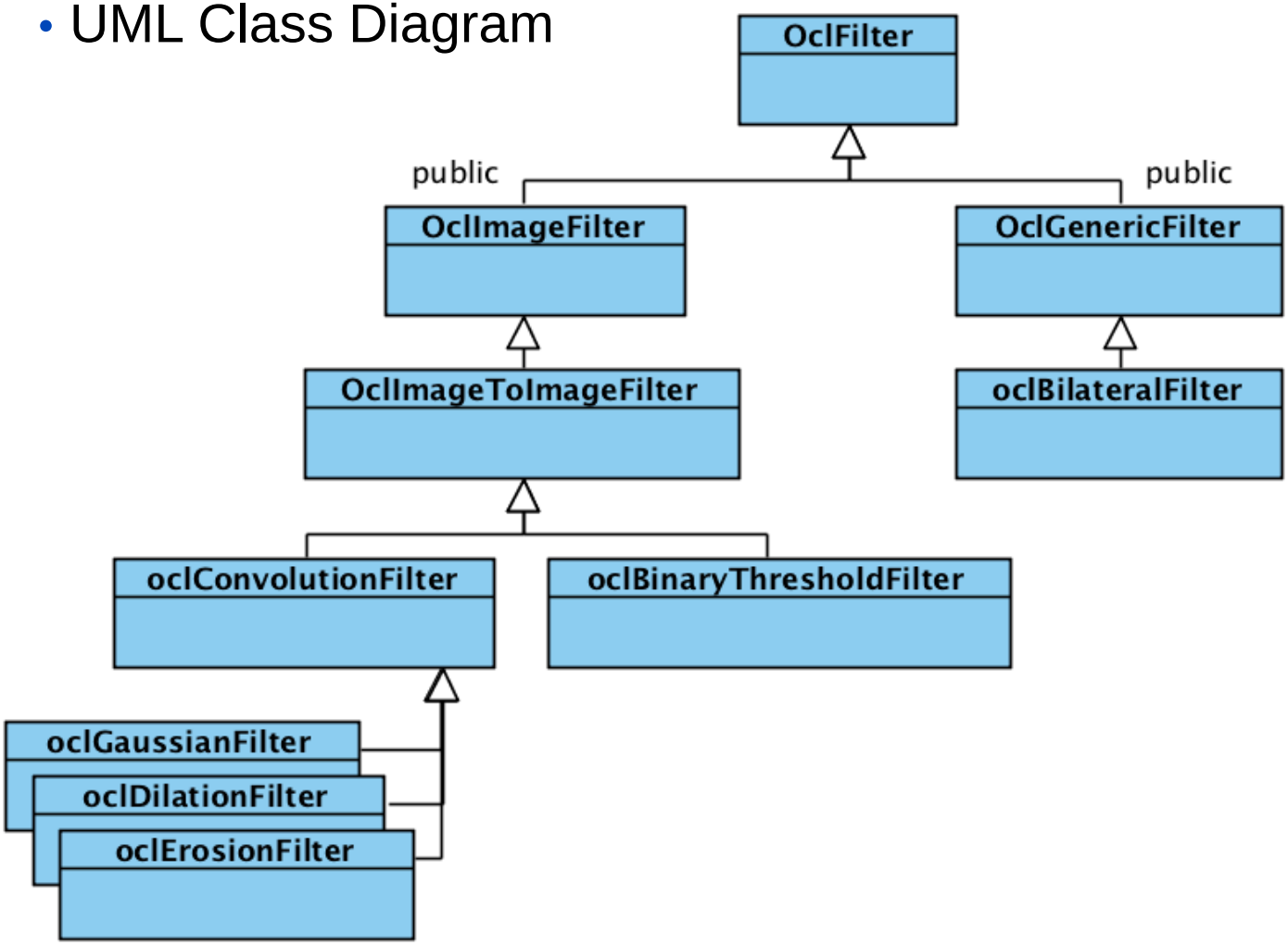
- **OclRessourceManager**
  - each filter – unique ID
  - manager holds compiled binaries (avoiding multiple code compile)

- **mitkOclUtils**.h
  - OCL_CHECK_ERR(int) macro
  - methods for compiling, platform information

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

MITK OpenCL

- UML Class Diagram



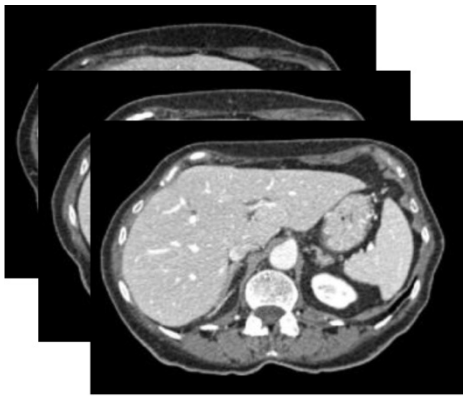
Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

dkfz.

Image guided therapy with MITK

Motivation

The Problem:

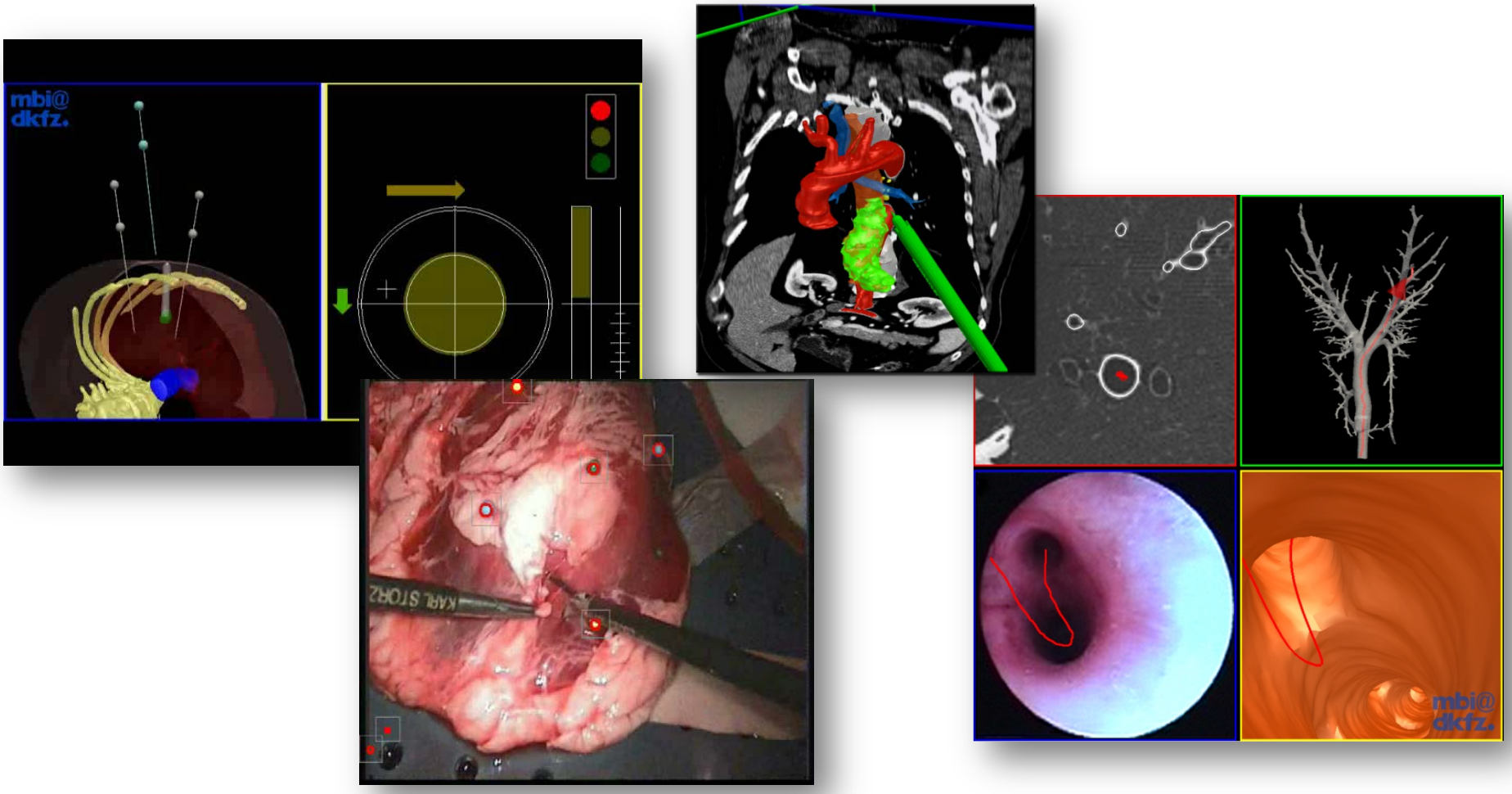


Planning → Intervention

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Motivation

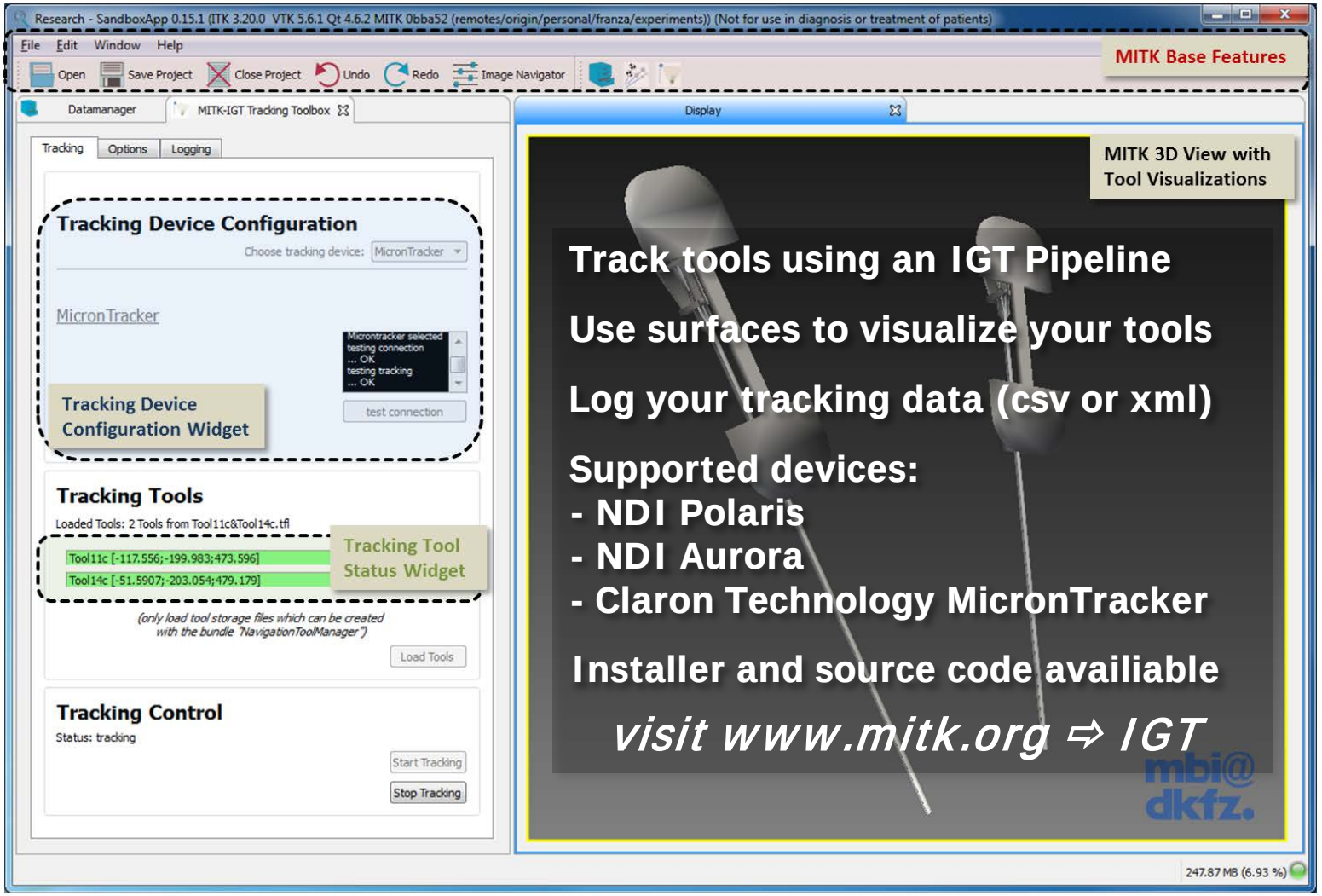
The Solution: Image Guided Therapy



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Tracking Technology

- MITK-IGT: Software to use tracking systems



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Outline

- Motivation
- **Requirements**
- Structure
- Outlook

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Requirements

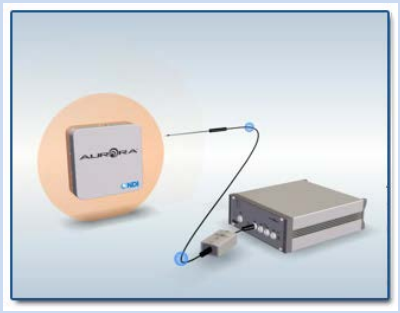
- Hardware control of tracking systems

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Requirements

- Hardware control of tracking systems

Tracking systems

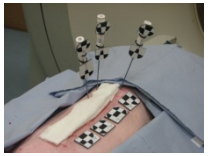


Tracking tools

Requirements

- Hardware control of tracking systems

- Real-time localization of multiple objects

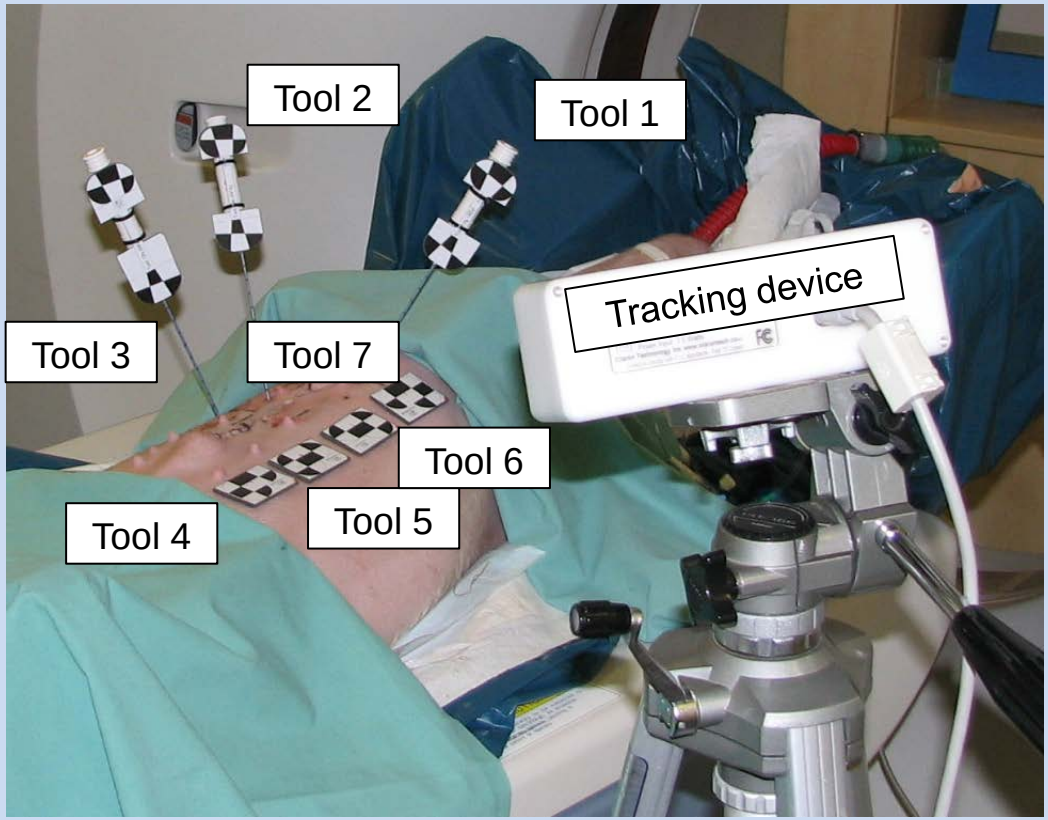


Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Requirements

- Real-time localization of multiple objects

Multiple tools for each tracking device

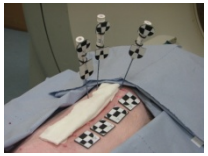


Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

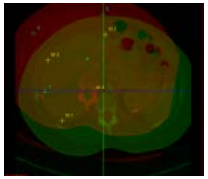
Requirements

- Hardware control of tracking systems

- Real-time localization of multiple objects



- Registration

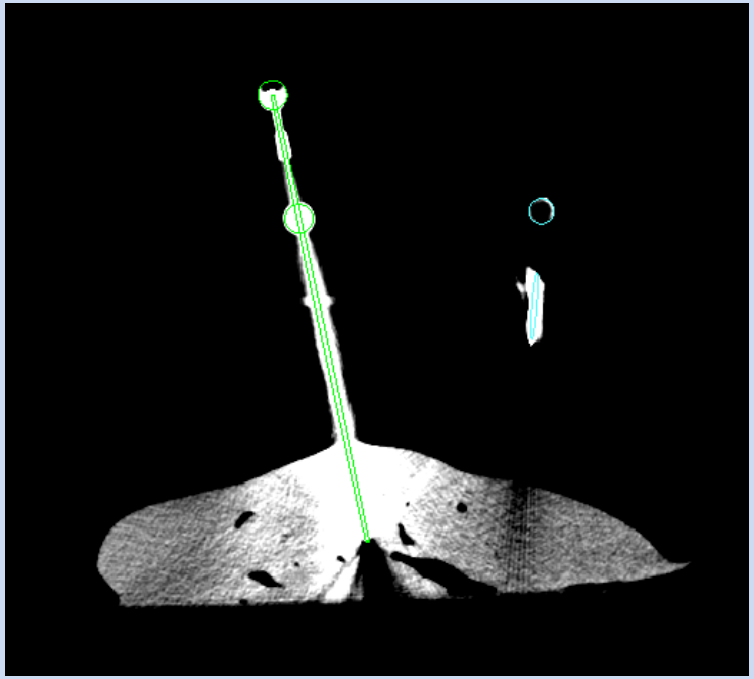


Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Requirements

- Registration

Image to patient registration and object registration

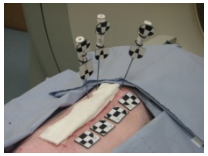


Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

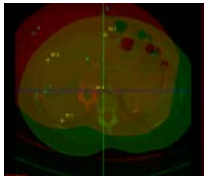
Requirements

- Hardware control of tracking systems

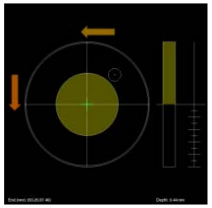
- Real-time localization of multiple objects



- Registration



- Visualization

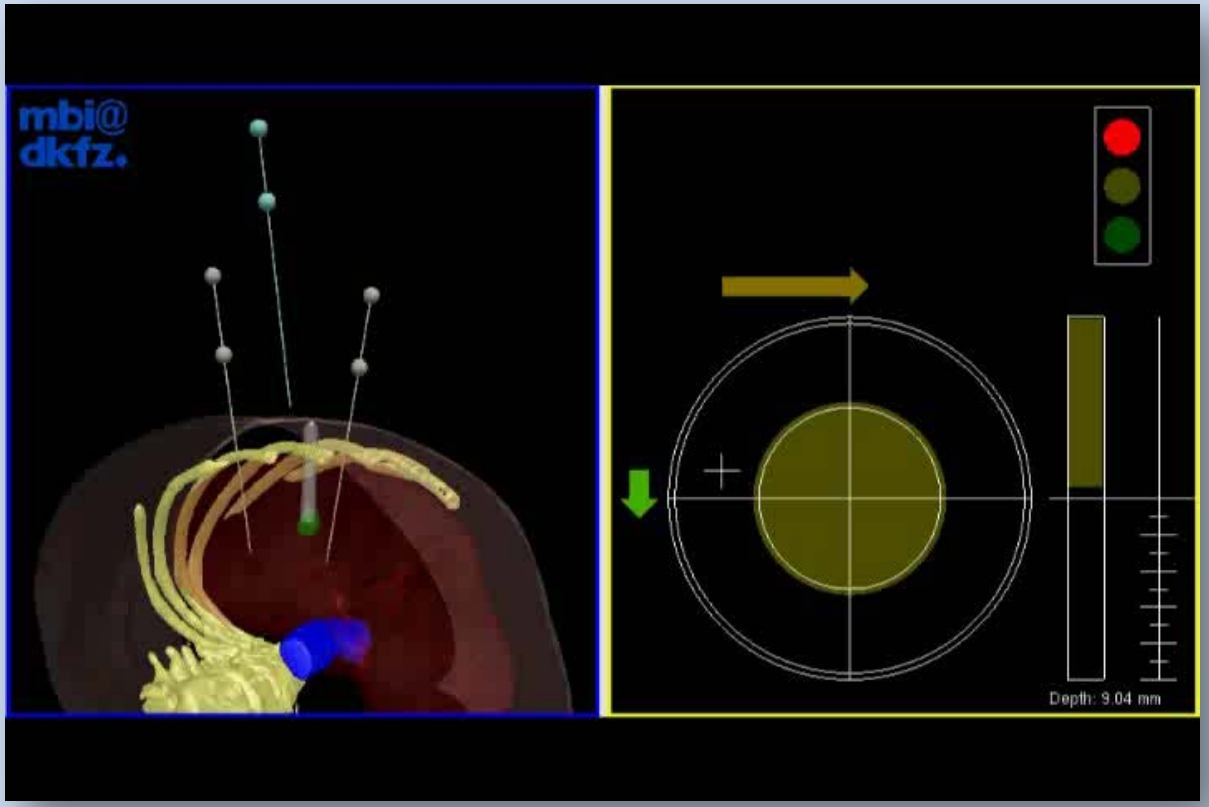


Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Requirements

- Visualization

Efficiently show pose information of tracked tool
-> Visualization filter



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Outline

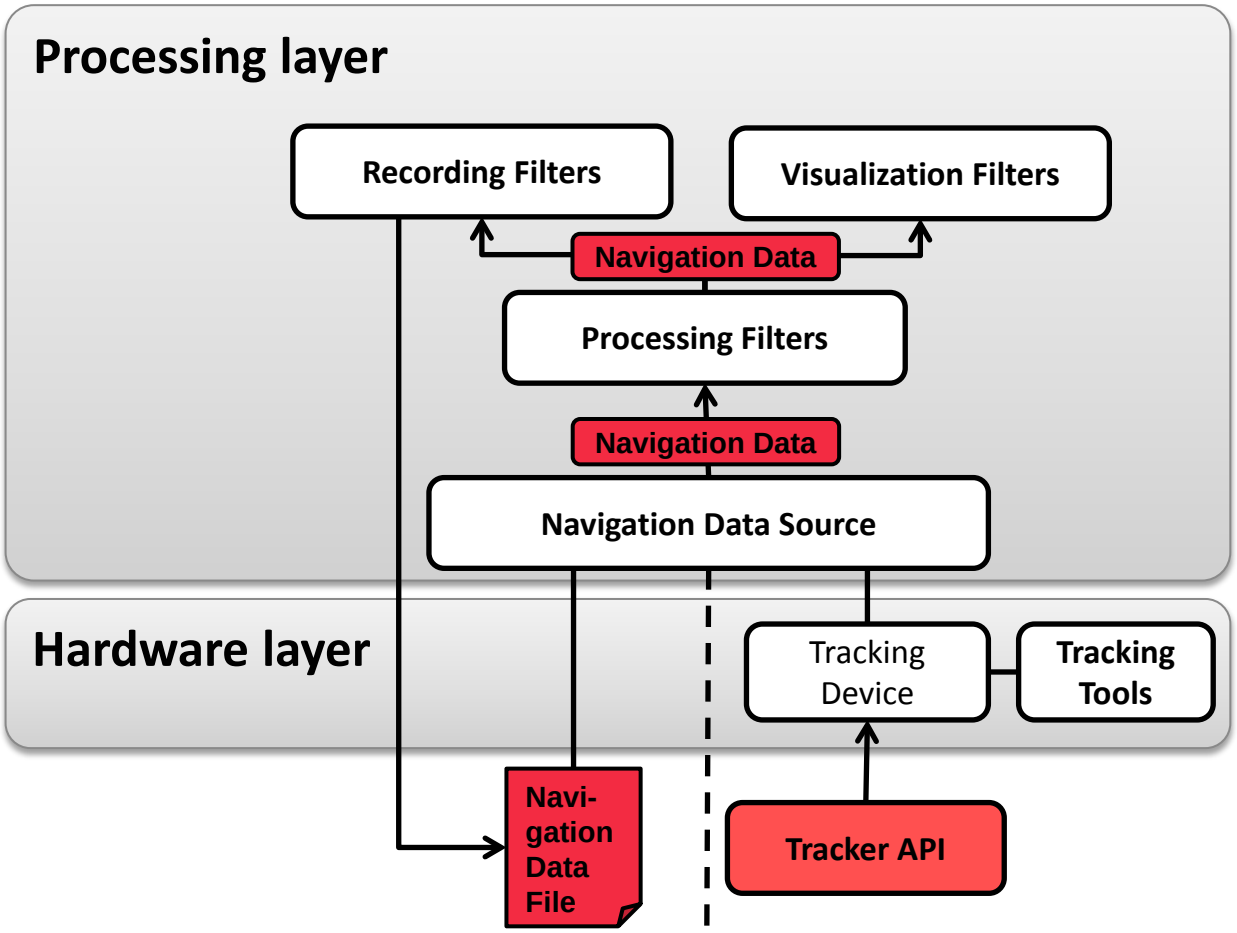
- Motivation
- Requirements
- **Structure**
  - Hardware layer
  - Processing layer
- Outlook

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Overview

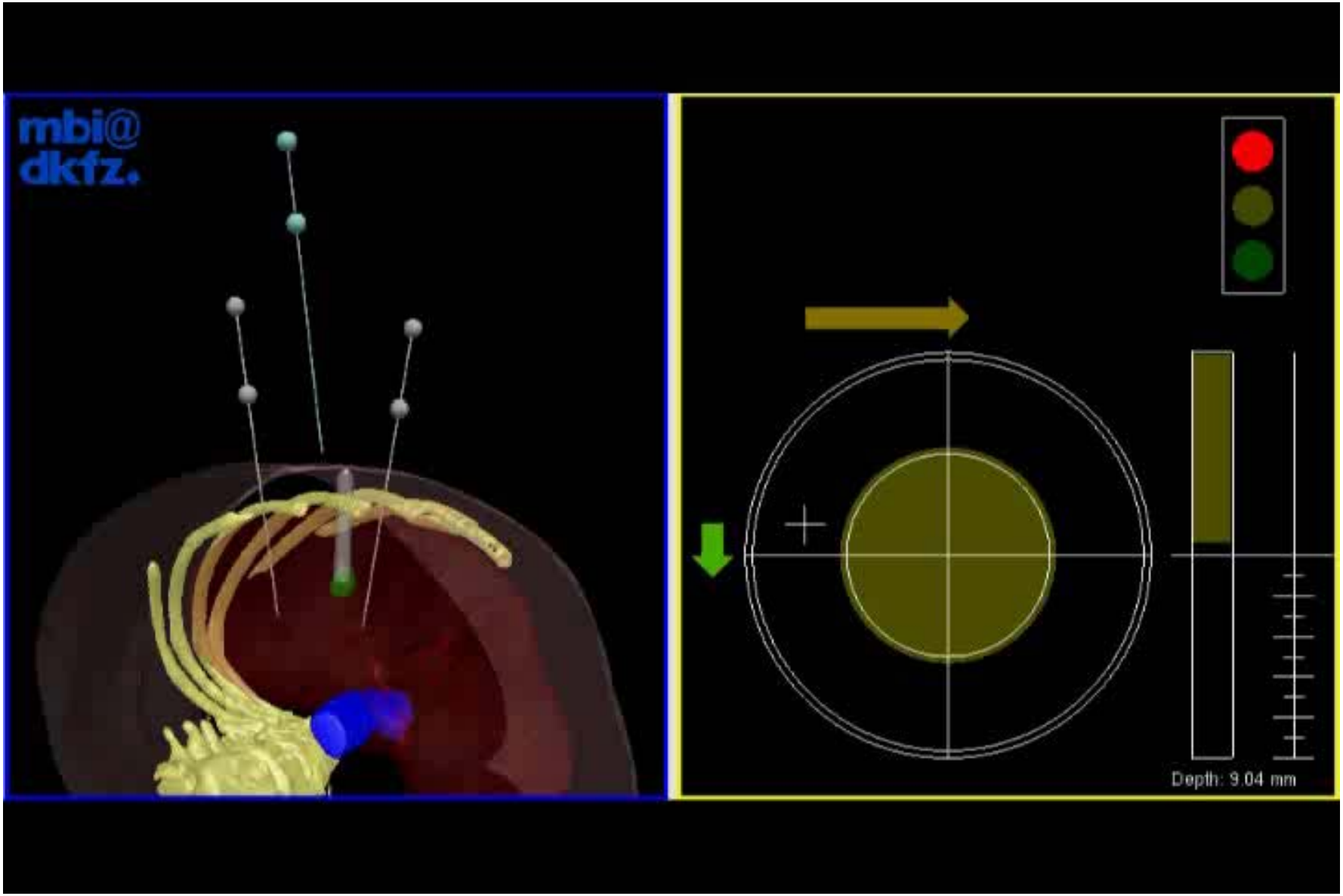
„Recording, Processing and Visualization of tracking data “

„Control of tracking devices“



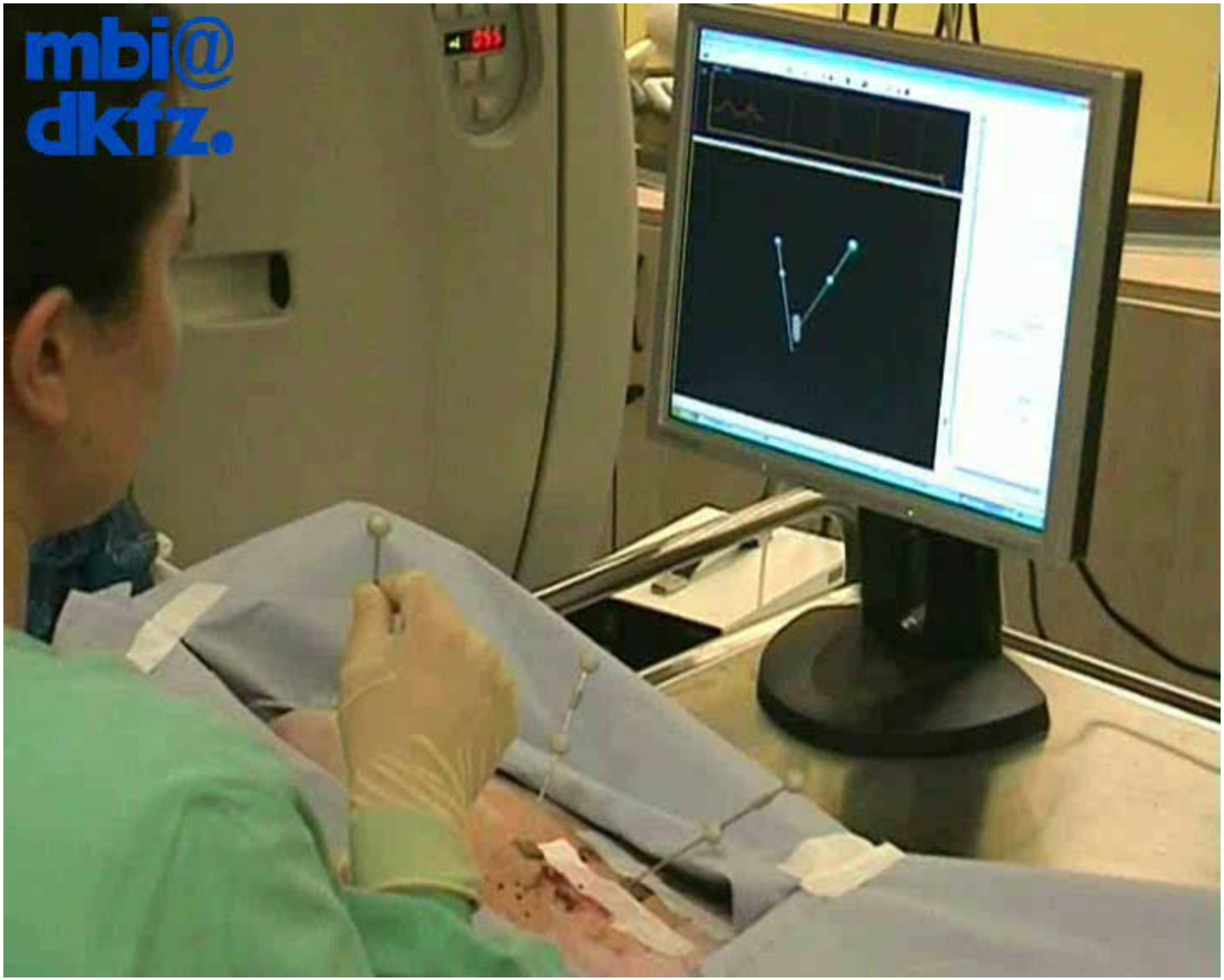
Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Liver Ablation



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Assisted liver tumor ablation – Pig experiments

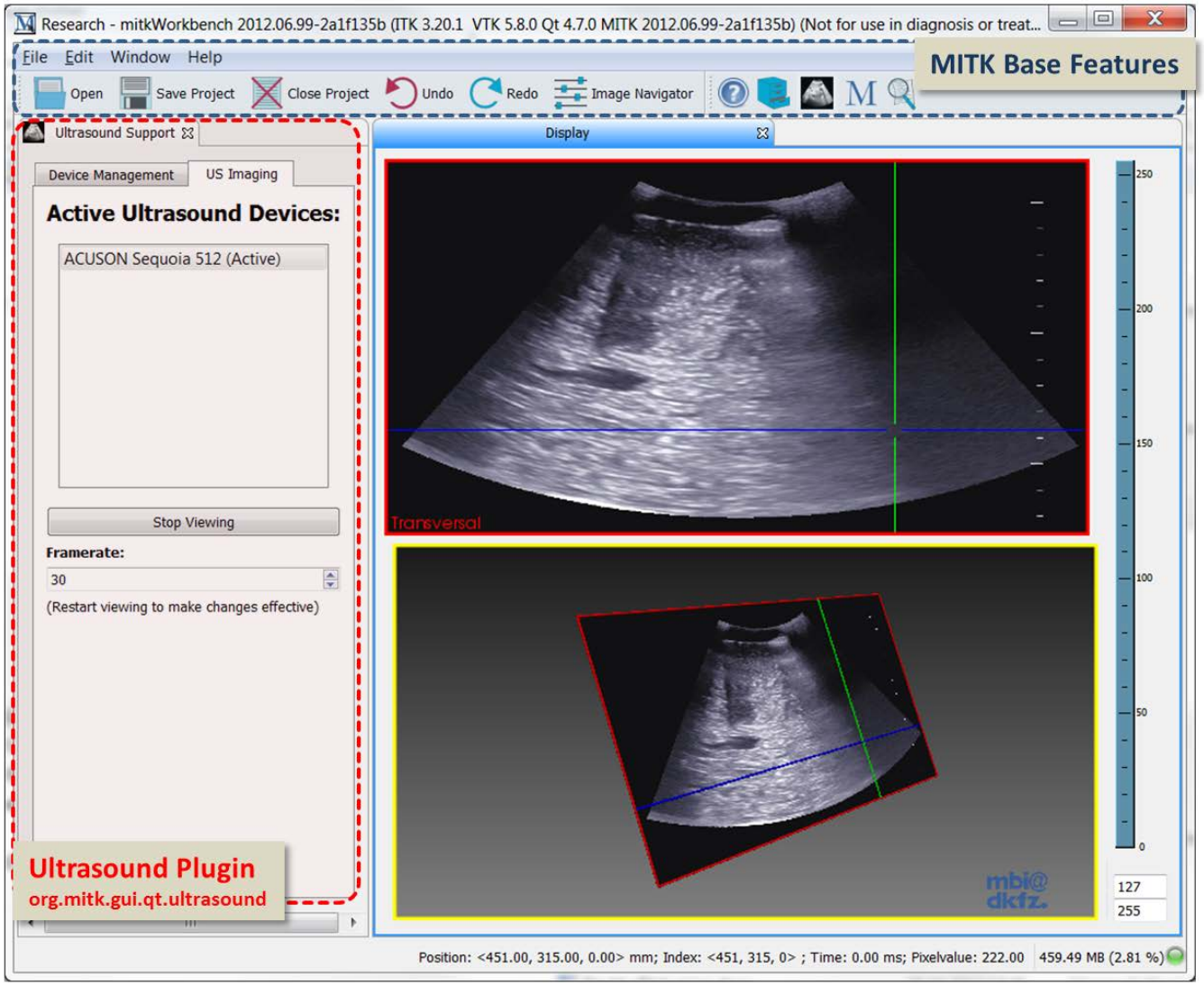


Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

MITK-US: US Image Processing

MITK module for support of ultrasound devices (MITK-US)

Software demo @ BVM2013



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

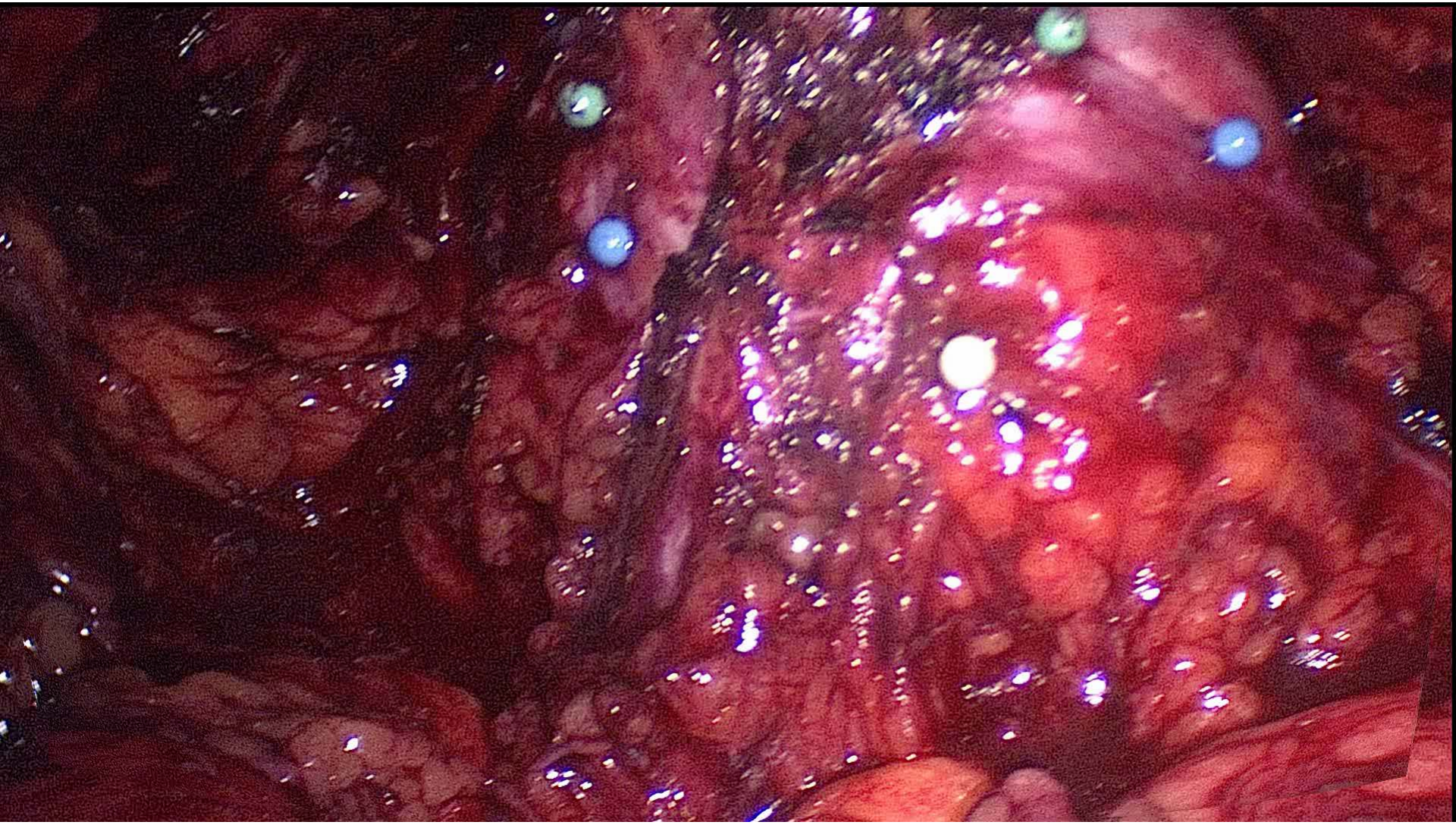
dkfz.

Augmented Reality

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Laparoscopic Prostatectomy

dkfz.



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

dkfz.

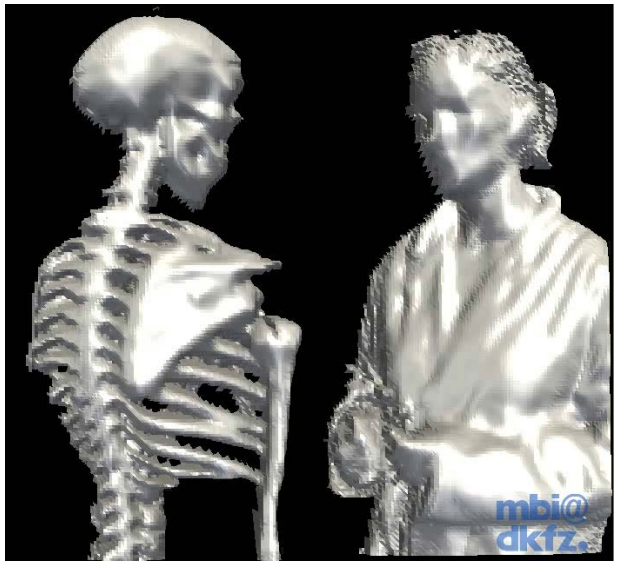
MITK Time-of-Flight

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

ToF Surface-Generation

Depth-Field

Original surface



Surface reconstruction

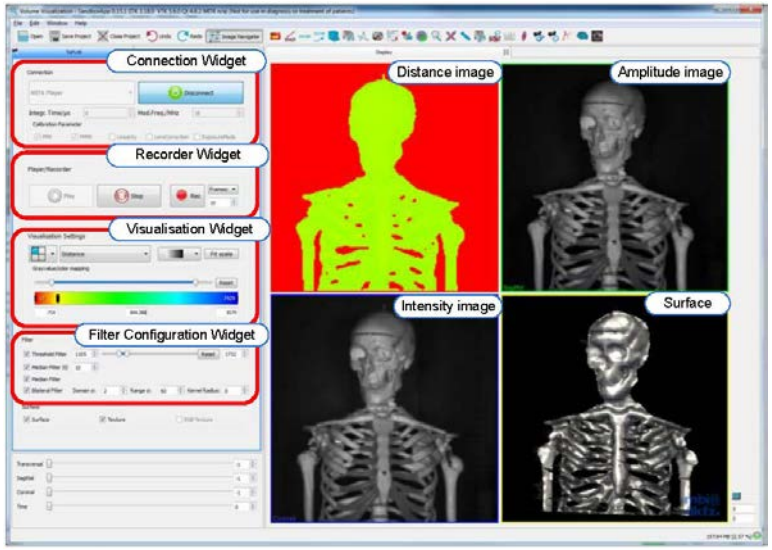
Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

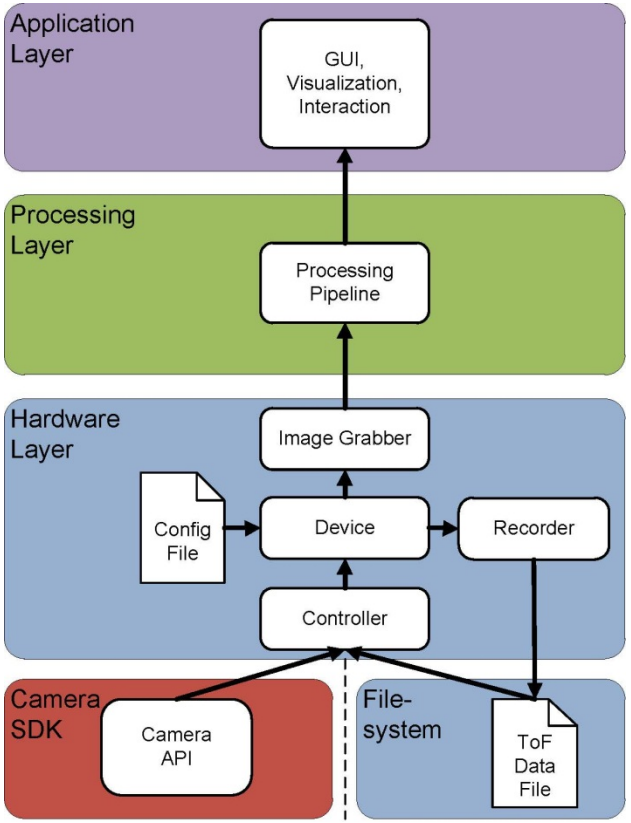
Time of Flight (ToF) - Imaging

dkfz.

MITK-ToF – open-source toolkit for range image processing

- Abstracts hardware communication from data processing / interaction
- Re-usable GUI components
- Flexible, fast data processing





Int J CARS (2011): “MITK-ToF – Range data within MITK”

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

dkfz.

MITK Special Interactions

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Headtracking

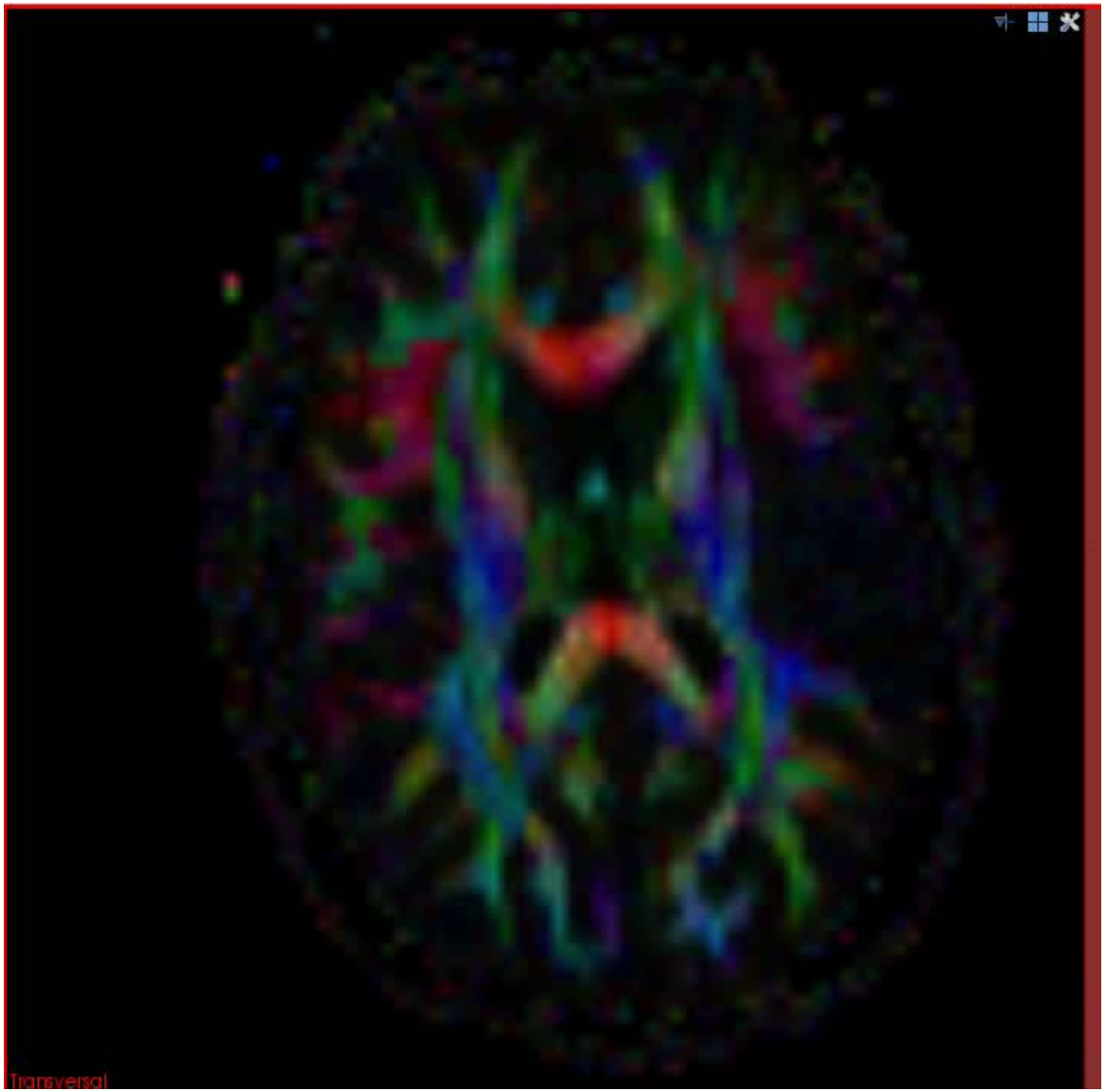
Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

dkfz.

MITK: extending data types

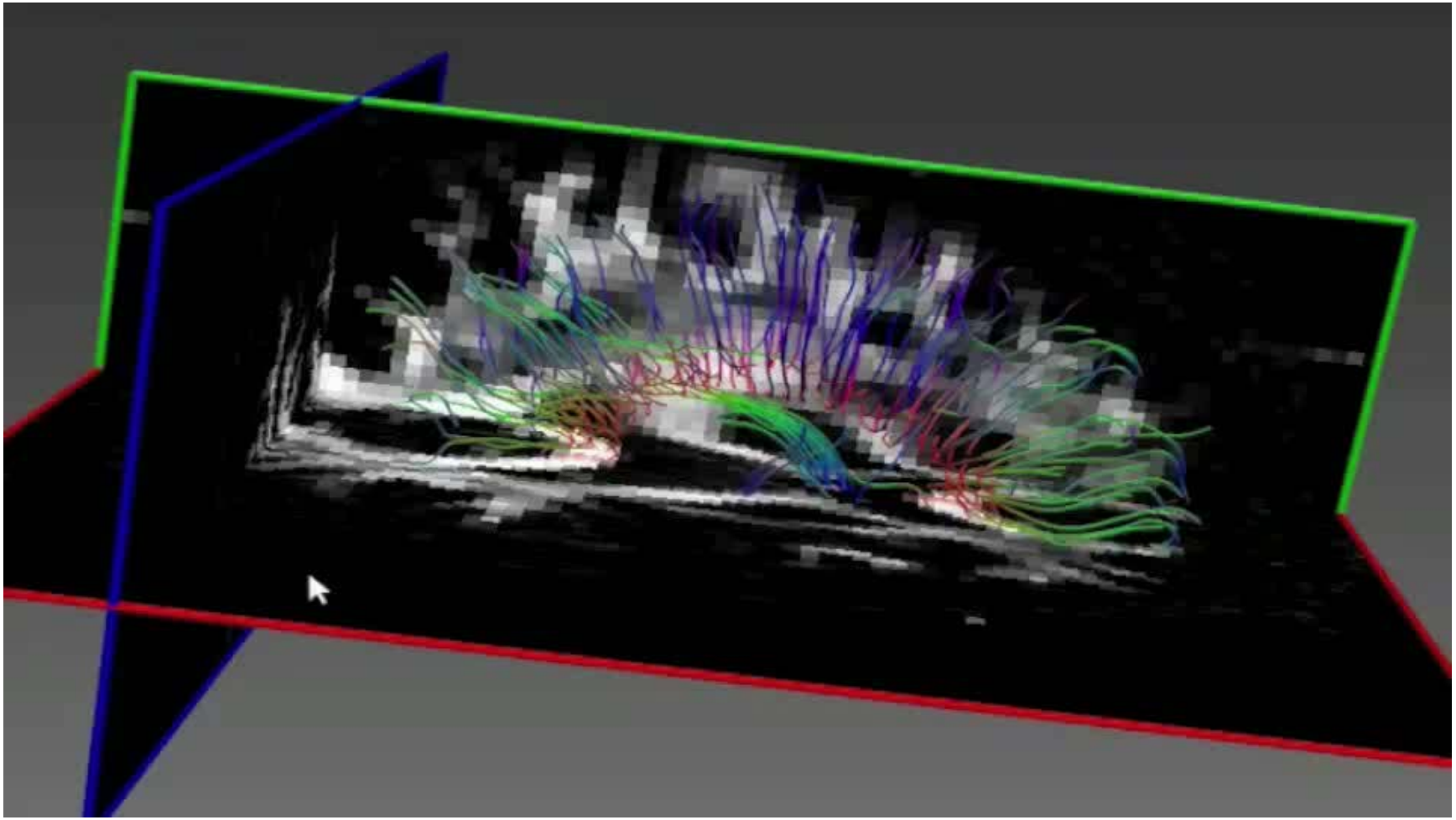
Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Diffusion Imaging (Q-Balls)



Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

Visualisation of Nerve Bundles



MITK – Getting started

Marco Nolden

Medical and Biological Informatics
German Cancer Research Center, Heidelberg (D)

dkfz. DEUTSCHES KREBSFORSCHUNGSZENTRUM IN DER HELMHOLTZ-GEMEINSCHAFT

Marco Nolden
Medical and Biological Informatics
Heidelberg (D)

MITK links

- Homepage: http://www.mitk.org
- Tutorial: http://docs.mitk.org/2012.12/TutorialPage.html
- Bug tracking: http://bugs.mitk.org
- Quality control: http://cdash.mitk.org
- Mailing list: http://mitk.org/Mailinglist
- Source code managed by git:
  http://mitk.org/git/?p=MITK.git;a=summary
  https://github.com/MITK

Vielen Dank!

Feedback bitte!

Andreas Fetzer: a.fetzer@dkfz-heidelberg.de
Michael Müller: michael.mueller@dkfz-heidelberg.de
Marco Nolden: m.nolden@dkfz-heidelberg.de
Sascha Zelzer: s.zelzer@dkfz-heidelberg.de