MITK in the context of NA-MIC

The Medical Imaging Interaction Toolkit

Div. of Medical and Biological Informatics, DKFZ Heidelberg

Open-source Toolkits

Powerful toolkits for

- Visualization: VTK (www.vtk.org)
- Segmentation/registration: ITK (www.itk.org)

But:

insufficient support for interactive, multi-view software

MITK ...

- uses parts of NA-MIC: ITK & VTK
- adds features outside the scope of boths
  ➔ is not at all a competitor to VTK or ITK
Medical Imaging Interaction Toolkit (MITK)
- open-source C++ toolkit based on ITK/VTK
- coordination of visualizations and interactions
- combine modules developed independently from each other

<table>
<thead>
<tr>
<th>Application / PACS Viewer Plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MITK module layer</strong>: building blocks; functionality blocks</td>
</tr>
<tr>
<td><strong>GUI-toolkit</strong> (Qt, FLTK, …)</td>
</tr>
<tr>
<td><strong>MITK</strong>: coordination interactivity</td>
</tr>
<tr>
<td><strong>ITK</strong>: algorithms segmentation registration</td>
</tr>
<tr>
<td><strong>VTK</strong>: visualization</td>
</tr>
</tbody>
</table>

Object-oriented C++ framework / toolkit
- BSD-style license, almost identical to VTK / ITK
- Supports
  - Linux and Windows
  - gcc 3.3, 4.2, VC7.1, VC8, VC9
  - Latest VTK release
  - Latest two ITK releases
- MITK-core does not depend on a GUI toolkit
- MITK-application-level provides
  - Qt3 base application
  - Many Qt3 widgets
  - FLTK example
  - Qt4 is work in progress
- MITK's core is GUI independent

MITK uses NA-MIC tools and software process

**CMake:** config and build system

**Subversion:** version management

**Bugzilla:** bug tracking

**Doxygen:** documentation

**SourceForge:** mailing list

**DART:** automatic builds and test runs
What MITK does –
a quick overview

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
**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.)

**Abdominal CT (**Image**)**
**Liver (**Surface**)**
**Tumor (**Surface**)**
**Vessels (**Graph**)**
**MRI (**Image**)**
**Helper Objects**
**Landmarks (**Points**)**

---

**Rendering the data repository**

**RenderWindows:**

- **single** RenderWindow class
- **different types** of views
  - 2D/3D
  - special views definable (e.g., for AR)
    ```cpp
    renderer->SetMapperID(mitk::BaseRenderer::Standard3D);
    ```
- **point** to the data repository
  - any number of views on the data:
    ```cpp
    renderer1->SetData(repository);
    renderer2->SetData(repository);
    ...
    ```
Defining *how* we want to see the data …

- original data
- ellipsoid
- segmentation
- display planes

Render and interact on curved planes
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

Controlling rendering with properties

- Rendering specific properties

<table>
<thead>
<tr>
<th>Generic</th>
<th>Image</th>
<th>PointSet</th>
<th>Surface</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>opacity</td>
<td>line width</td>
<td></td>
</tr>
<tr>
<td>layer</td>
<td>color</td>
<td>pointsize</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>use color</td>
<td>selected color</td>
<td></td>
</tr>
<tr>
<td></td>
<td>binary</td>
<td>color</td>
<td></td>
</tr>
<tr>
<td></td>
<td>outline binary</td>
<td>contour color</td>
<td></td>
</tr>
<tr>
<td></td>
<td>texture</td>
<td>close</td>
<td></td>
</tr>
<tr>
<td></td>
<td>interpolation</td>
<td>show points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>volumerendering</td>
<td>show distances</td>
<td></td>
</tr>
<tr>
<td></td>
<td>levelwindow</td>
<td>distance decimal digits</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LookupTable</td>
<td>show angles</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TransferFunction</td>
<td>show distant lines</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>line width</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scalar mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>wireframe line width</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>material</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>scalar visibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>color mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>representation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interpolation</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>MITK application framework level</th>
<th>MITK data processing level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MITK</strong> application toolkit level</td>
<td><strong>MITK</strong> application toolkit level</td>
</tr>
<tr>
<td><em>(GUI-toolkit dependent)</em></td>
<td><em>(GUI-toolkit independent)</em></td>
</tr>
<tr>
<td>GUI-toolkit</td>
<td>Optional add-on libraries</td>
</tr>
<tr>
<td>ITK</td>
<td>VTK</td>
</tr>
<tr>
<td>OpenCV</td>
<td>PACS-Plugin</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

## Data Processing level

- access to ITK and VTK data structures and algorithms
- Access to other libraries (OpenCV, ANN, TinyXML,…)
- Tree / Graph data structures and algorithms
- Spatial object location (Geometries)
- Time steps for data objects
- Loading / saving of different file formats
- Interface to tracking systems
**Application toolkit level**

- Rendering
  - Mappers, Update Management, Render Properties
- Data Management
  - Object Container, Object Properties, Scene Management
- Interaction
  - Statemachine based
- Undo/Redo
- Processing of tracking data
- Qt Widgets
  - TreeNodeSelector, StandardViews, PropertyEditor, LevelWindow, Renderwindow, SlicerControls, Navigationviews,…

**Application framework**

Base application (*MITK-MainApp)*:
- Container for functionalities
  - independent „Plug-Ins“ for specific problems
- Shared repository for data objects
- Persistence:
  - Application state can be saved and restored on next startup
- Interface to CHILI-PACS Workstation
MITK functionality modules

**Functionality** = a module with …
- an identification (icon/tooltips/…)
- a workspace area
- a control area
- a option dialog
- a help page (manual)
- the algorithmic implementation
Combining functionality blocks

- Functionalities are independent from each other
- They communicate via the data repository
- Tracking component allows access to different tracking systems:
  - NDI Polaris/Aurora
  - Microntracker
  - Our own video based Inside-Out-Tracking algorithm
- Filter pipelines for tracking coordinates (Kalman-filter,...)
- Logging & replay of tracking data
- Geometry classes to manage different coordinate systems
  - (not yet open source)
Embedding in PACS/tele-conferencing system

Integration in PACS/telemedicine system CHILI® as a PlugIn:
- PACS
- Connection to modalities
- DICOM import/export
- DICOM “unification”
- Data transfer
- Tele-radiology
- Management of results from image processing

→ facilitates clinical integration

How to get started
Download options:
- anonymous svn:
  `svn co http://svn.mitk.org/trunk/mitk/
- zipped archive (v 0.8)
  https://sourceforge.net/projects/mitk/

Tutorial:
http://mitk.org/documentation/
A small functionality

We'll have a look at a very simple functionality for region growing:

0. (create a functionality)
1. select an image
2. set some seed points
3. react, when a GUI button is pressed
4. run a region grower from ITK
5. display the result in MITK

(can be downloaded at mitk.org)
A small functionality

1. selection of an image
2. set some seed points
3. react, when a GUI button is pressed
4. run a region grower from ITK
5. display the result in MITK
A small functionality

1. selection of an image
2. set some seed points
3. react, when a GUI button is pressed
4. run a region grower from ITK
5. display the result in MITK

mitk::Image

AccessByItk macro

templated method for ITK code

itk::Image<TPixel, VImageDimension>

mitk::ImportItkImage()

mitk::Image

mitk::Image::ReadImage();

// set some properties
// 
// itk::ImageIOFactory::LoadImageProperties() somehow
// nowNode->setProperty("halo", mitk::BoolProperty::New(true));
// nowNode->setProperty("hace", mitk::StringProperty::New("deep segmentation"));
// nowNode->setProperty("color", mitk::ColorProperty::New(1,0,0,0,0));
// 
// somehow: AccessByItk<ITKImage>(nowNode, nowNode, nowNode); // mitk::Image::ReadImage() somehow
// // EndAccessByItk
// 
// // add result to data tree
// 
// 
// // EndAccessByItk
OSGi-based extensibility for MITK-applications:

- OSGi: component model originally designed for Java
- Basic building blocks are bundles (aka plugins) and services
- Easy extensibility through loose coupling
- Every plugin can define its own extension points
  - general concept for extensions
  - plugins within plugins at no additional costs
- MITK will provide a set of core bundles and services for complex imaging tasks and interactions
Other enhancements

Hot topics:

- Release of two functionalities for registration
- Transition of the Qt3 MITK code to Qt4
- OSGi-based application platform providing views/editors, perspectives and GUI-services (openCherry plugins)
- Python scripting
Thank you!

www.mitk.org