06/08/2013

Using ITK images with MITK

Joseph Görres

dkfz.
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Different image representations

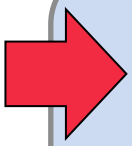
Why do we have different image types?

Library	Task
ITK	3D Image Processing
VTK	Rendering
OpenCV	2D Image Processing

MITK ➔ bridge between different image types
➔ implements additional concepts:
(e.g. for DataManager)

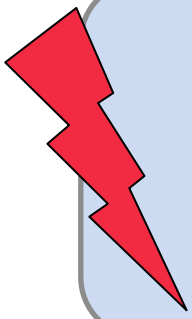
ITK image

- template<class TPixel, unsigned int VImageDimension = 2>
  class itk::Image< TPixel, VImageDimension >



The image type is templated:

We need to know
pixel type and image dimension
at compile time



We usually do NOT know them at compile time:

The bridge to ITK compiles all supported pixel types. At runtime the pixel type is investigated and then the right method/class is called

Image Cast

There are two possible directions:

From ITK to MITK:

- PixelType is already set at compiletime
- MITK can get it by calling mitk::MakePixelType<>()

```
image->Initialize(
  MakePixelType<itk::Image<TPixel,VDimension> >()
);
```

From MITK to ITK:

- PixelType is usually unknown at compiletime
- We need to prepare for several pixel types

→ Multiplexer

Multiplexer

```
#define mitkPixelTypeMultiplex0( function, ptype ) \
{ \
 if ( ptype.GetComponentType() == itk::ImageIOBase::CHAR )\
   function<char>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::UCHAR)\
   function<unsigned char>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::SHORT )\
   function<short>( ptype, ); \
 else if ( ptype.GetComponentType() == itk::ImageIOBase::USHORT )\
   function<unsigned short>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::INT )\
   function<int>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::UINT )\
   function<unsigned int>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::LONG )\
   function<long int>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::ULONG )\
   function<unsigned long int>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::FLOAT )\
   function<float>( ptype, );\
 else if ( ptype.GetComponentType() == itk::ImageIOBase::DOUBLE )\
   function<double>( ptype, );\
}\
```

ITK → MITK

AccessByItk

- macro that provides a type independent “cast”
- it is necessary to implement a templated method

```
template<TPixel, VImageDimension>
MyAccessMethod(itk::Image<TPixel, VImageDimension>*
itkImage)
{
...
}
```

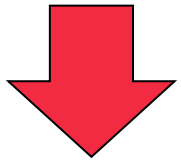
- once the method is defined, the macro can be used:

```
AccessByItk(mitkImage, MyAccessMethod)
```

ITK → MITK

AccessByItk

- If we use the basic macro, we create a lot of code
- `MyAccessMethod` will be compiled for all supported image type



- If we know the image dimension:
  AccessFixedDimensionByItk

- If we know the pixel type:
  AccessFixedTypeByItk

ITK → MITK

If the itk::Image type is known, we can also use:

CastToItkImage (method)

- Casts the pixel type automatically when different

GrabItkImageMemory (method)

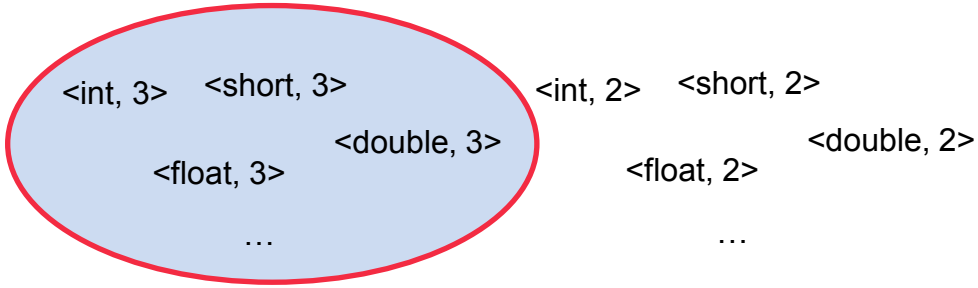
- Lets the mitk::Image manage the itk::Image memory
- Itk::Image remains valid until mitk::Image frees its memory

ImageToItk (filter)

Rules of thumb

- Be aware of the complexity when “casting” to an itk::Image

→ Narrow down possible types and dimensions



- Think about image memory before using macros

→ “Casts” usually do NOT copy image memory for efficiency

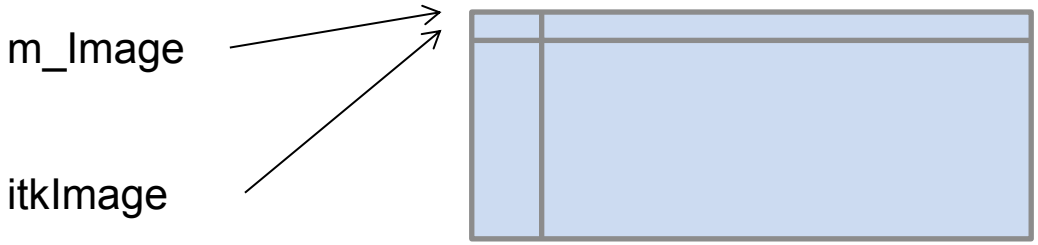
→ Only cast once, mostly there is no reason to cast back

```
CastToItkImage(mitkImage,itkImage);
itkImage->ApplyMagicFilter();
CastToMitkImage(itkImage,mitkImage);
```

Final Slide

- Think about your scope when using AccessByItk

→ The given itk::Image points to the same image memory as your mitk::Image, which could be in your method scope



- More information can be found on:
  http://docs.mitk.org/nightly/group__Adaptor.html#MitkToItk