Lecture 17
ITK Pipeline

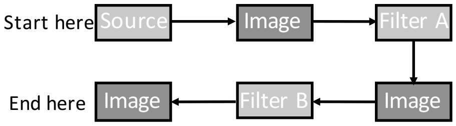
Methods in Medical Image Analysis - Spring 2016
18-791 (CMU ECE) : 42-735 (CMU BME) : BioE 2630 (Pitt)
Dr. John Galeotti

Based in part on Shelton's slides from 2006

The Pipeline

- ITK is organized around *data objects* and *process objects*
  - You should now be somewhat familiar with the primary data object, **`itk::Image`**
  - Today we'll talk about how to do cool things to images, using process objects
- A *pipeline* is a series of process objects that operate on one or more data objects
- The data objects "flow" along the pipeline

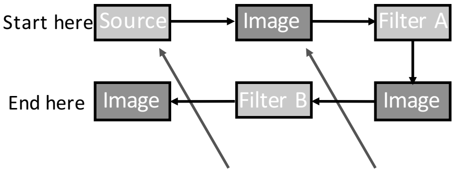
The pipeline idea



The pipeline consists of:

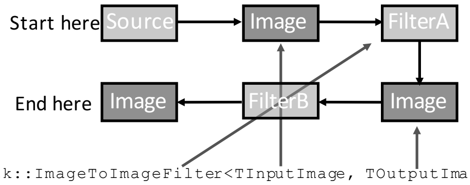
- Data objects
- Process object (things that create data objects)

Image sources



`itk::ImageSource<TOutputImage>`
The base class for all process objects that produce images without an input image

Image to image filters

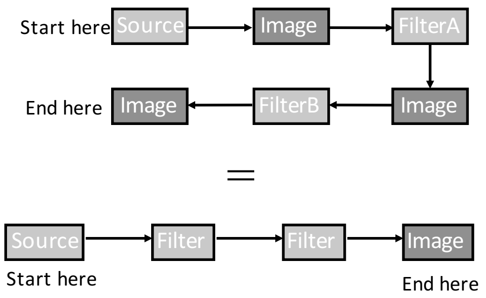


`itk::ImageToImageFilter<TInputImage, TOutputImage>`
The base class for all process objects that produce images when provided with an image as input.

Input and output

- ImageSource's do not require input, so they have only a **`GetOutput()`** function
- ImageToImageFilter's have both **`SetInput()`** and **`GetOutput()`** functions

Ignoring intermediate images



How this looks in code

```
SrcType::Pointer src = SrcType::New();
FilAType::Pointer filterA = FilAType::New();
FilBType::Pointer filterB = FilBType::New();

src->SetupTheSource();
filterA->SetInput( src->GetOutput() );
filterB->SetInput( filterA->GetOutput() );

ImageType::Pointer im = filterB->GetOutput();
```

When execution occurs

- The previous page of code **only** sets up the pipeline - i.e., what connects to what
- This **does not** cause the pipeline to execute
- In order to “run” the pipeline, you must call `Update()` on the last filter in the pipeline

Propagation of Update()

- When `Update()` is called on a filter, the update propagates back “up” the pipeline until it reaches a process object that does not need to be updated, or the start of the pipeline

When are process objects updated?

- If the input to the process object has changed
- If the process object itself has been modified - e.g., I change the radius of a Gaussian blur filter

How does it know?

Detecting process object modification

- The easy way (when writing your own proces object) is to use

  `itkSetMacro(MemberName, type);`

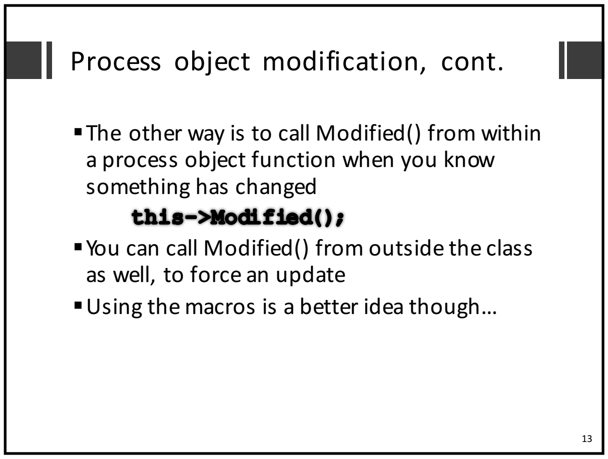
  which produces the function

  `void SetMemberName(type);`

  that calls `Modified()` for you when a new value is set in the class.
- For example, the compiler turns this line of code:

  `itkSetMacro(DistanceMin, double);`

  into a member function, `SetDistanceMin()`, that sets member variable `m_DistanceMin`.

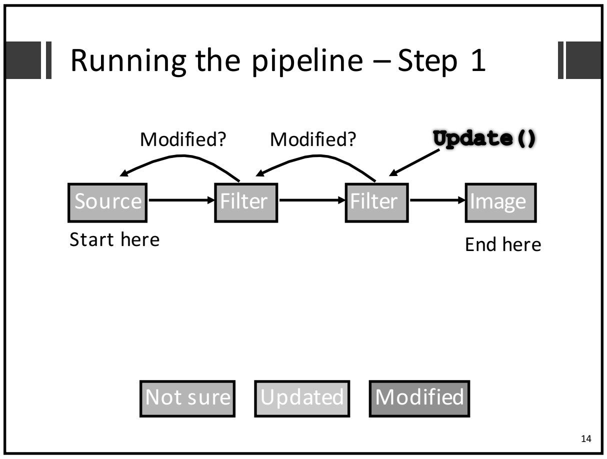


Process object modification, cont.

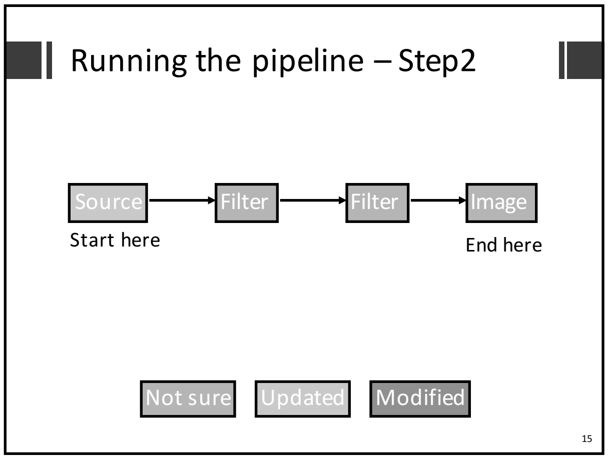
- The other way is to call Modified() from within a process object function when you know something has changed

```
this->Modified();
```

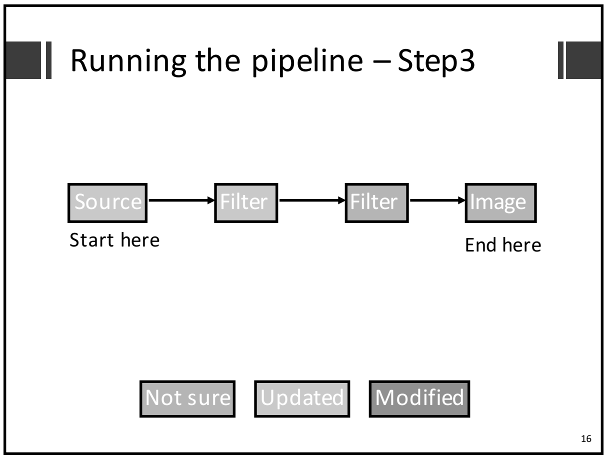
- You can call Modified() from outside the class as well, to force an update
- Using the macros is a better idea though…



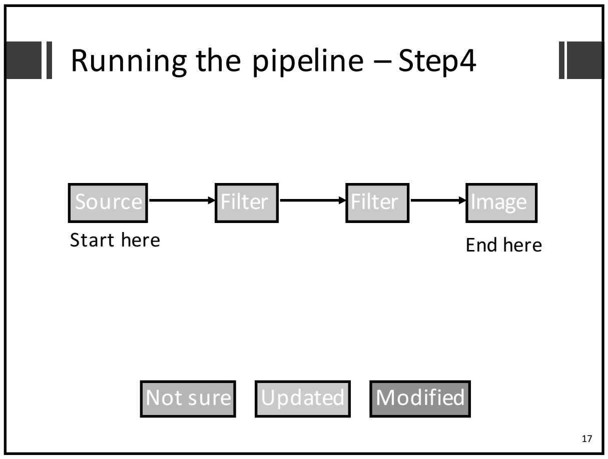
Running the pipeline – Step 1



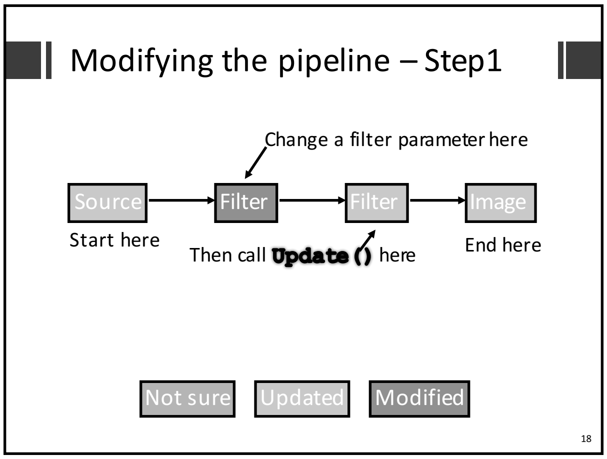
Running the pipeline – Step2



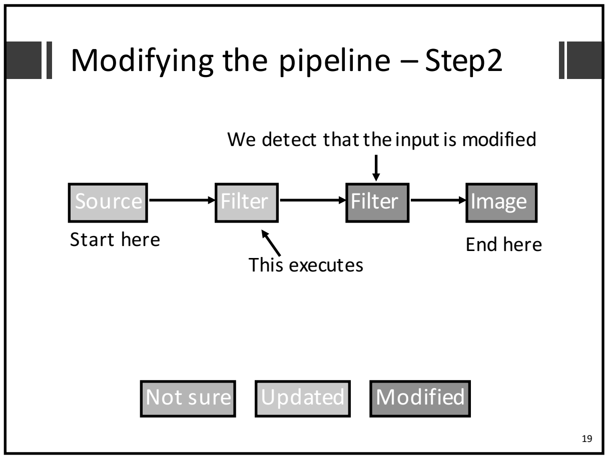
Running the pipeline – Step3



Running the pipeline – Step4



Modifying the pipeline – Step1



Modifying the pipeline – Step2

We detect that the input is modified

Source → Filter → Filter → Image

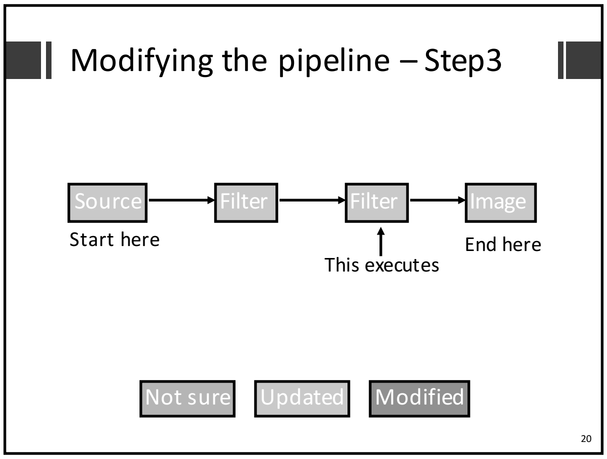
Start here

This executes

End here

Not sure | Updated | Modified

19



Modifying the pipeline – Step3

Source → Filter → Filter → Image

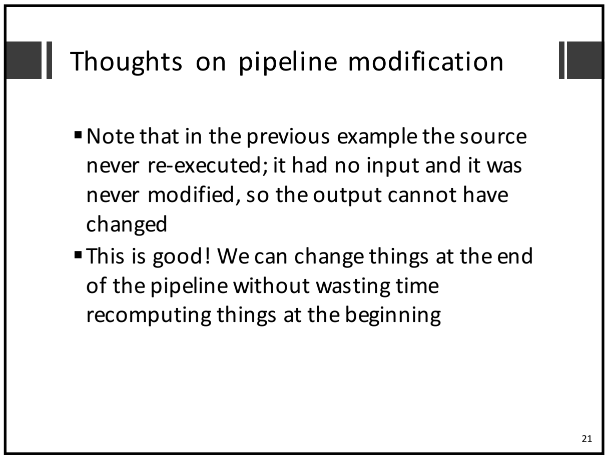
Start here

This executes

End here

Not sure | Updated | Modified

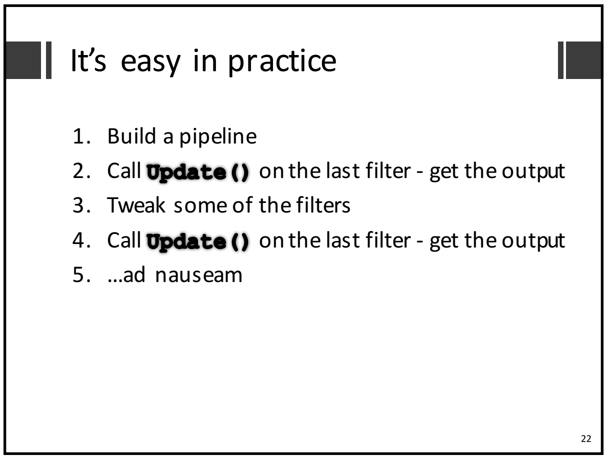
20



Thoughts on pipeline modification

- Note that in the previous example the source never re-executed; it had no input and it was never modified, so the output cannot have changed
- This is good! We can change things at the end of the pipeline without wasting time recomputing things at the beginning

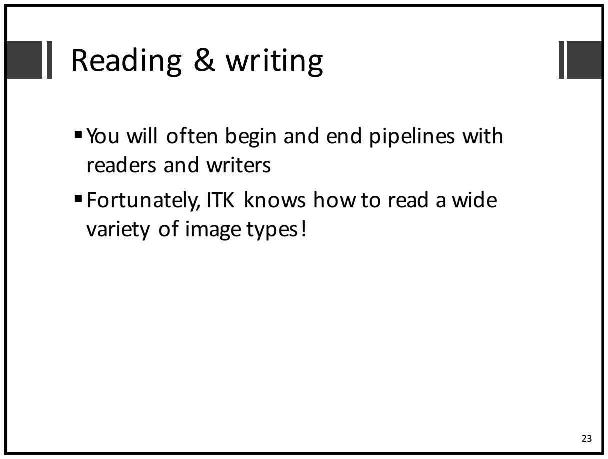
21



It’s easy in practice

1. Build a pipeline
2. Call **`Update()`** on the last filter - get the output
3. Tweak some of the filters
4. Call **`Update()`** on the last filter - get the output
5. …ad nauseam

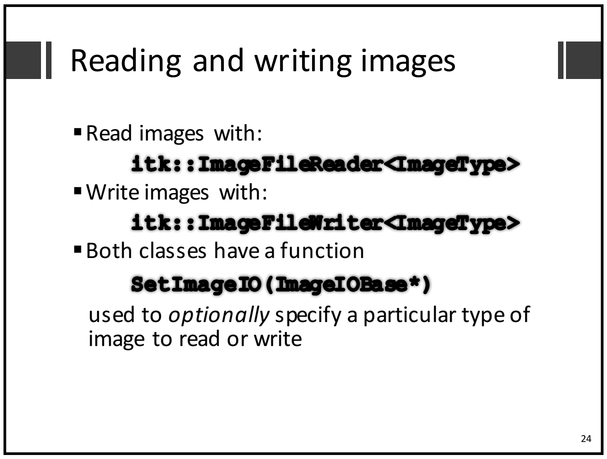
22



Reading & writing

- You will often begin and end pipelines with readers and writers
- Fortunately, ITK knows how to read a wide variety of image types!

23



Reading and writing images

- Read images with:
  `itk::ImageFileReader<ImageType>`
- Write images with:
  `itk::ImageFileWriter<ImageType>`
- Both classes have a function
  `SetImageIO(ImageIOBase*)`

  used to *optionally* specify a particular type of image to read or write

24

Reading an image (4.1.2)

- Create a reader
- If you know the file format (optional):
  - Create an instance of an **`ImageIOBase`** derived class (e.g. **`PNGImageIO`**)
  - Pass the IO object to the reader
- Set the file name of the reader
- Update the reader

Reader notes

- The **`ImageType`** template parameter is the type of image you want to convert the stored image *to*, not necessarily the type of image stored in the file
- ITK assumes a valid conversion exists between the stored pixel type and the target pixel type

Writing an image

- Almost identical to the reader case, but you use an **`ImageFileWriter`** instead of a reader
- Output format can be specified with an IO object (optional)
  - If you've already created an IO object during the read stage, you can recycle it for use with the writer

More read/write notes

- ITK actually has several different ways of reading files - what I've presented is the simplest conceptually
- Remember, you can read files without knowing their format a-priori
  - Just don't specify any IO objects.
- Many more details are in ch. 7 of the software guide.

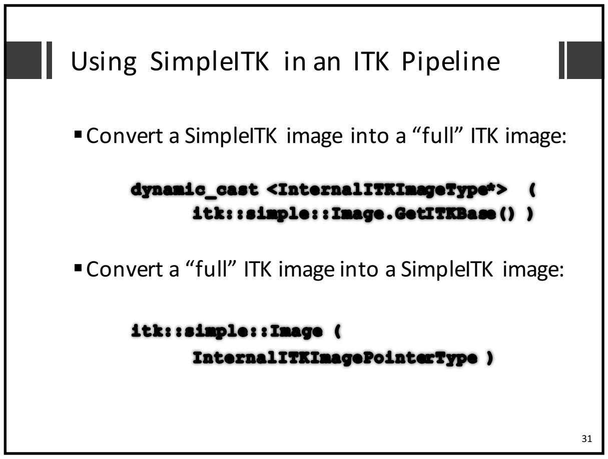
SimpleITK Pipeline

It doesn't have one!

- SimpleITK's interface does NOT use a pipeline
- Every time you call a filter in SimpleITK, it re-executes.
- You manually execute each filter every time you think it is necessary
- You also manually pass the updated output from one filter to the input of the next filter

Combining ITK and SimpleITK

- You can combine ITK with SimpleITK!
- For example:
  - Use SimpleITK to quickly read and preprocess images
  - Use "full" ITK to perform a complex registration
  - Use SimpleITK to save the results
- This is really easy in C++
- We just need to integrate SimpleITK into our ITK pipeline



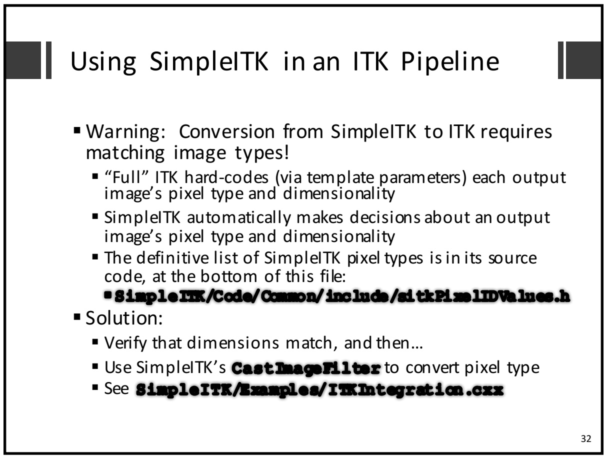
Using SimpleITK in an ITK Pipeline

- Convert a SimpleITK image into a "full" ITK image:

```
dynamic_cast <InternalITKImageType*>  (
      itk::simple::Image.GetITKBase() )
```

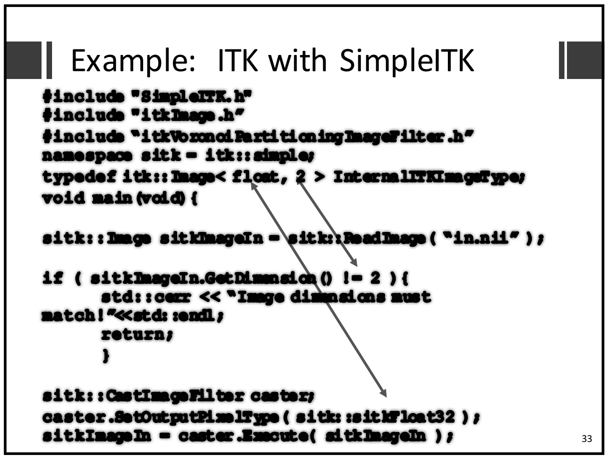
- Convert a "full" ITK image into a SimpleITK image:

```
itk::simple::Image (
      InternalITKImagePointerType )
```



Using SimpleITK in an ITK Pipeline

- Warning: Conversion from SimpleITK to ITK requires matching image types!
  - "Full" ITK hard-codes (via template parameters) each output image's pixel type and dimensionality
  - SimpleITK automatically makes decisions about an output image's pixel type and dimensionality
  - The definitive list of SimpleITK pixel types is in its source code, at the bottom of this file:
    - `SimpleITK/Code/Common/include/sitkPixelIDValues.h`
- Solution:
  - Verify that dimensions match, and then…
  - Use SimpleITK's `CastImageFilter` to convert pixel type
  - See `SimpleITK/Examples/ITKIntegration.cxx`



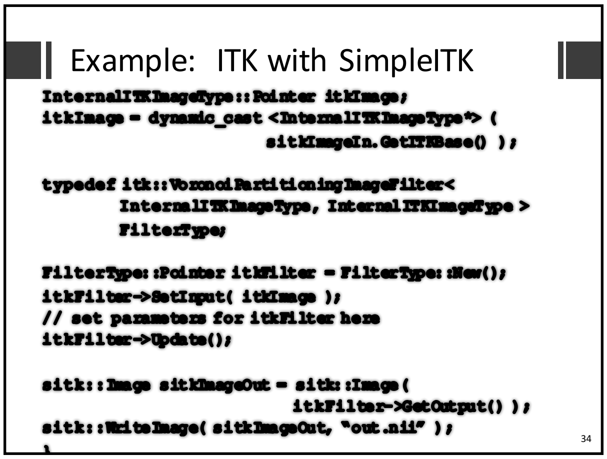
Example: ITK with SimpleITK

```
#include "SimpleITK.h"
#include "itkImage.h"
#include "itkVoronoiPartitioningImageFilter.h"
namespace sitk = itk::simple;
typedef itk::Image< float, 2 > InternalITKImageType;
void main(void){

sitk::Image sitkImageIn = sitk::ReadImage( "in.nii" );

if ( sitkImageIn.GetDimension() != 2 ){
      std::cerr << "Image dimensions must
match!"<<std::endl;
      return;
      }

sitk::CastImageFilter caster;
caster.SetOutputPixelType( sitk::sitkFloat32 );
sitkImageIn = caster.Execute( sitkImageIn );
```



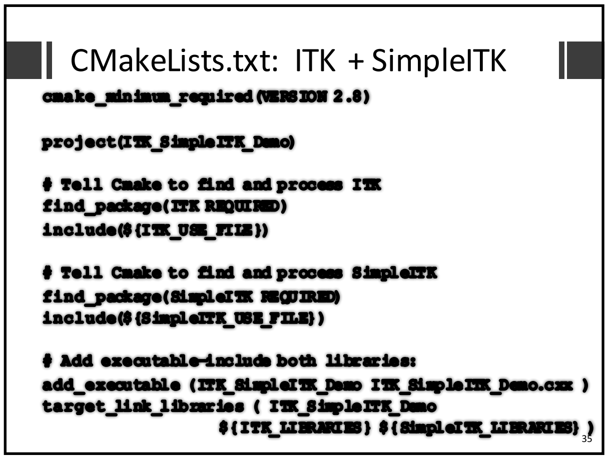
Example: ITK with SimpleITK

```
InternalITKImageType::Pointer itkImage;
itkImage = dynamic_cast <InternalITKImageType*> (
                        sitkImageIn.GetITKBase() );

typedef itk::VoronoiPartitioningImageFilter<
      InternalITKImageType, InternalITKImageType >
      FilterType;

FilterType::Pointer itkFilter = FilterType::New();
itkFilter->SetInput( itkImage );
// set parameters for itkFilter here
itkFilter->Update();

sitk::Image sitkImageOut = sitk::Image(
                           itkFilter->GetOutput() );
sitk::WriteImage( sitkImageOut, "out.nii" );
}
```



CMakeLists.txt: ITK + SimpleITK

```
cmake_minimum_required(VERSION 2.8)

project(ITK_SimpleITK_Demo)

# Tell Cmake to find and process ITK
find_package(ITK REQUIRED)
include(${ITK_USE_FILE})

# Tell Cmake to find and process SimpleITK
find_package(SimpleITK REQUIRED)
include(${SimpleITK_USE_FILE})

# Add executable-include both libraries:
add_executable (ITK_SimpleITK_Demo ITK_SimpleITK_Demo.cxx )
target_link_libraries ( ITK_SimpleITK_Demo
                 ${ITK_LIBRARIES} ${SimpleITK_LIBRARIES} )
```