Tutorial

on SlicerRT and 3D Slicer modules

Agenda

- Acquire and build SlicerRT (on an existing Slicer)
- Use python console in Slicer
- Create extension and module skeletons
  - Use extension template
  - Use scripted module template
- Use SlicerRT
- Write a simple scripted module
- Write a simple CLI module

Acquire and build SlicerRT

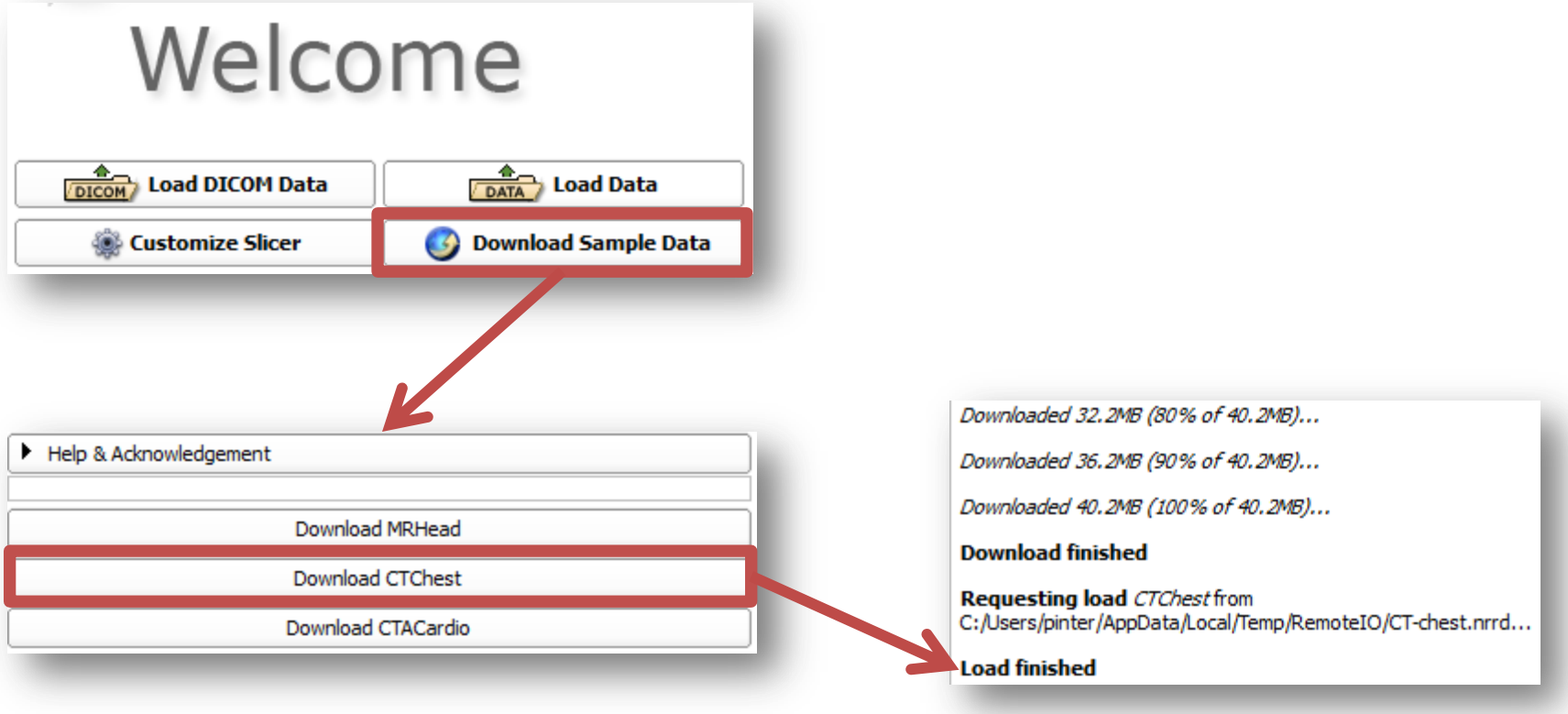
- We must have a 3D Slicer built in Release mode before starting
- Get SlicerRT from the repository
  - `mkdir SlicerRT`
  - `svn co https://subversion.assembla.com/svn/slicerrt/trunk/SlicerRt/src SlicerRT`
- Configure SlicerRT
  - `mkdir SlicerRT_R64-bin`
  - `cd SlicerRT_R64-bin`
  - `../cmake-2.8.10.2-Linux-i386/cmake -DCMAKE_CXX_COMPILER=/usr/bin/g++-4.6 -DCMAKE_BUILD_TYPE=Release -DSlicer_DIR=/home/slicer/Support/Slicer4-SuperBuild-Debug/Slicer-build ../SlicerRT`
- Build SlicerRT
  - `make`

Next item on agenda

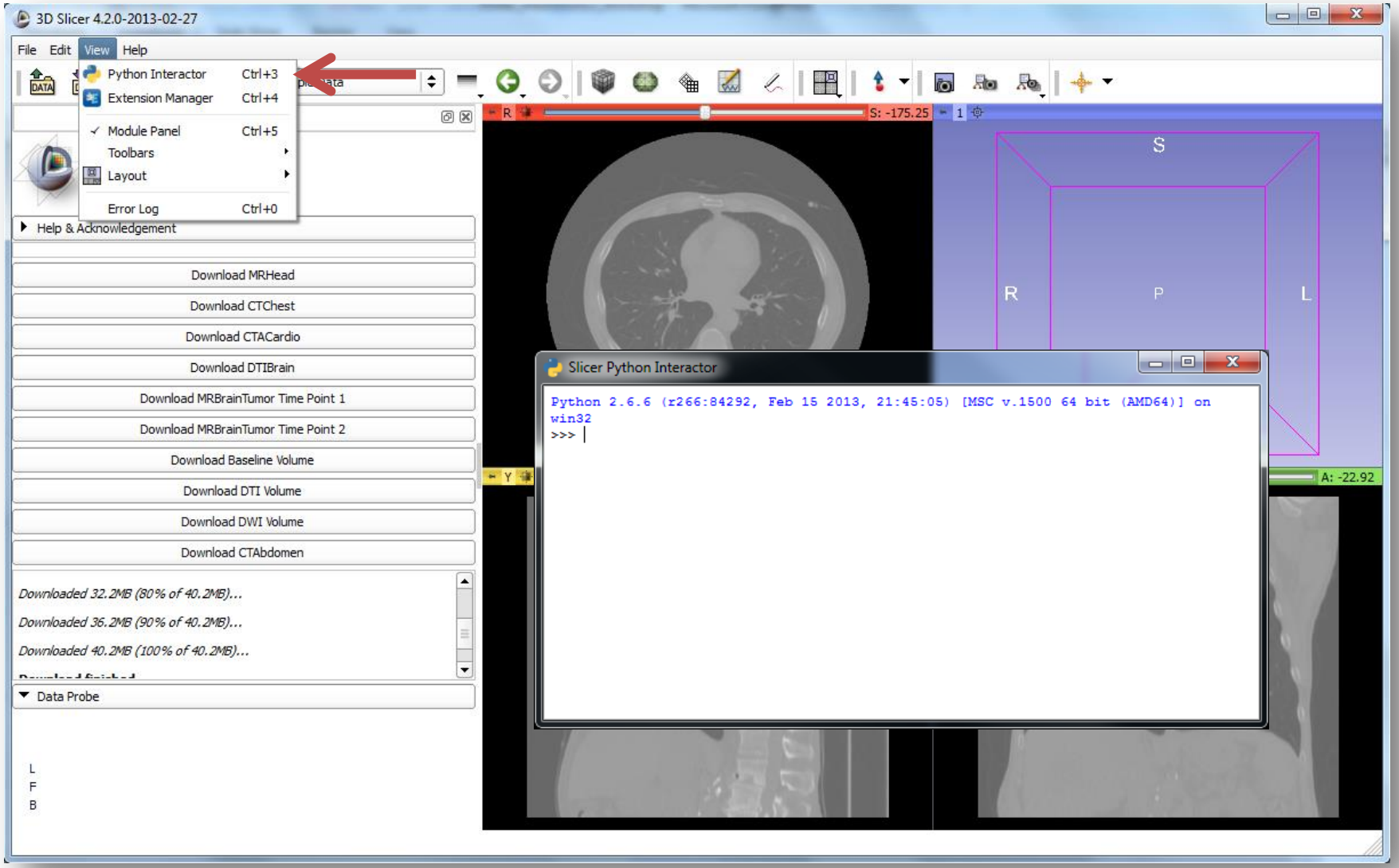
- Acquire and build SlicerRT (on an existing Slicer)
- **Use python console in Slicer**

Launch Slicer and load data

- `cd ~/Support/Slicer4-SuperBuild-Debug/Slicer-build`
- `./Slicer`



Introducing the python console



Accessing the MRML scene and nodes

- Accessing MRML scene directly
  - `c = slicer.mrmlScene.GetNodesByName('CTChest')`
  - `c.GetNumberOfItems()`
  - `v = c.GetItemAsObject(0)`

  Also useful: `slicer.mrmlScene.GetNodeByID()`

- Using utility functions
  - `v = slicer.util.getNode('CTChest')`

  OR
  - `v = slicer.util.getNode('CT*')`

  Also useful (returns a list!): `slicer.util.getNodes('CT*')`

Manipulating MRML objects

- Setting window/level values programmatically
  - `d = v.GetDisplayNode()`
  - `d.SetAutoWindowLevel(0)`
  - `d.SetWindowLevel(350,40)`

- Setting color palette programmatically
  - `c = slicer.util.getNode('Ocean')`
  - `d.SetAndObserveColorNodeID(c.GetID())`

- Slicer classes reference

http://www.slicer.org/doc/html/classes.html

Next item on agenda

- Acquire and build SlicerRT (on an existing Slicer)
- Use python console in Slicer
- **Create extension and module skeletons**
  - **Use extension template**
  - **Use scripted module template**

Create extension skeleton

- Setting window/level values programmatically
  - `cd ~/Support`
  - `python ~/Support/Slicer/Utilities/Scripts/ModuleWizard.py --template ~/Support/Slicer/Extensions/Testing/SuperBuildExtensionTemplate --target ./MyExtension MyExtension`
  - `cd MyExtension`
  - `python ~/Support/Slicer/Utilities/Scripts/ModuleWizard.py --template ~/Support/Slicer/Extensions/Testing/ScriptedLoadableExtensionTemplate/ScriptedLoadableModuleTemplate --target ./MyScriptedModule MyScriptedModule`

Create extension skeleton cont'd #1

- Edit super-build Cmake file
  - `gedit CMakeLists &`

Add under the other `add_subdirectory...` lines

  - `add_subdirectory(MyScriptedModule)`

Quit editor

  - `cd ..`
  - `mkdir MyExtension_R64-bin`
  - `cd MyExtension_R64-bin`

Create extension skeleton cont'd #2

- Configure extension skeleton
  - ```
    ../cmake-2.8.10.2-Linux-i386/cmake
    -DCMAKE_CXX_COMPILER=/usr/bin/g++-4.6 --
    DSlicer_DIR=/home/slicer/Support/Slicer4-
    SuperBuild-Debug/Slicer-build -
    DMyExtension_SUPERBUILD=0 ../MyExtension
    ```
- Build extension skeleton
  - ```
    make
    ```

Add new modules to the paths

- Start Slicer
- Add modules to the additional module paths
  - Go to *Edit / Application settings / Modules*
  - Click the “>>” button next to the *Additional module paths* section
  - Add the following directories:
    - *~/Support/MyExtension_R64-bin/lib/Slicer-4.2/qt-loadable-modules*
    - same location but the directory *qt-scripted-module*

Next item on agenda

- Acquire and build SlicerRT (on an existing Slicer)
- Use python console in Slicer
- Create extension and module skeletons
  - Use extension template
  - Use scripted module template
- **Use SlicerRT**

Add SlicerRT to the paths

- Start Slicer
- Add SlicerRT to the additional module paths
  - Go to *Edit / Application settings / Modules*
  - Click the “>>” button next to the *Additional module paths* section
  - Add all sub-directories from:
    - */home/slicer/Support/SlicerRT_R64-bin/inner-build/lib/Slicer-4.2*
  - Press *Restart*

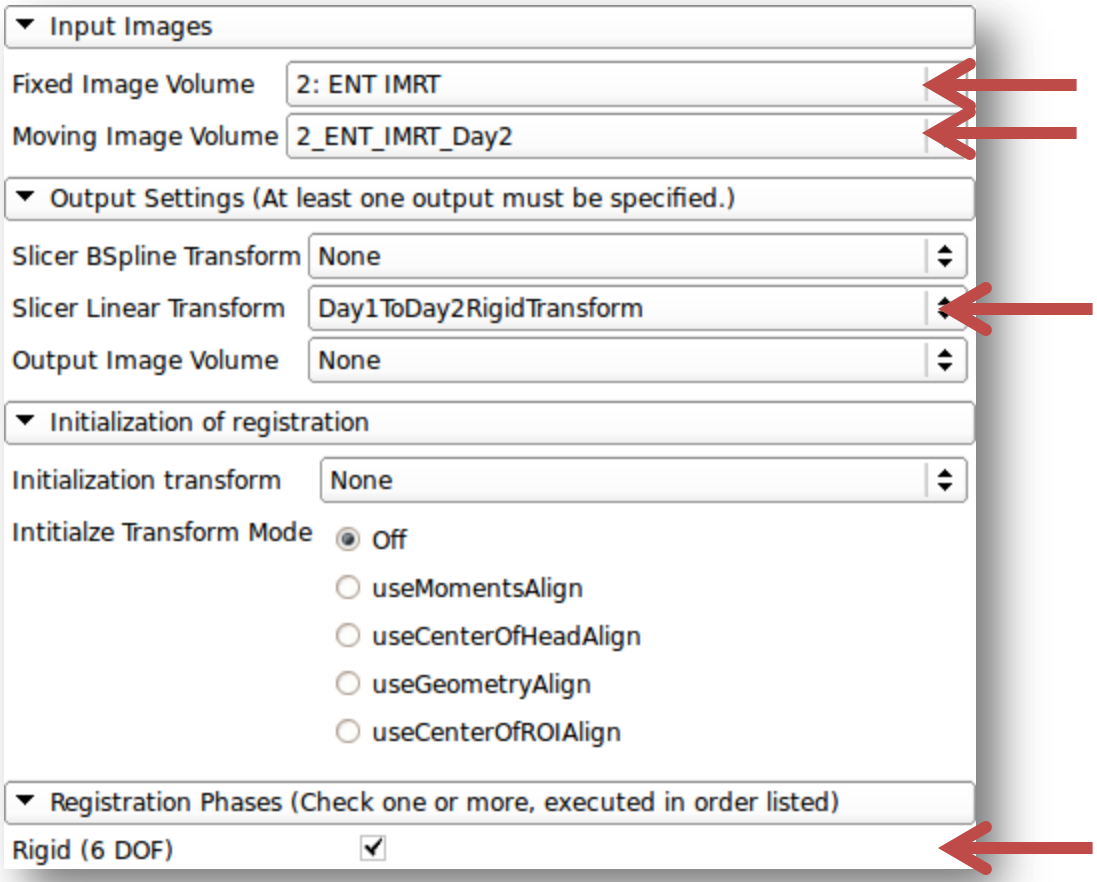
Load RT data

- Go to *module list / Testing / TestCases / SlicerRT Demo RSNA2012 Self Test*
- Click *Load data*
- Click 

(This step is not necessary, it just shows the DICOM browser which has to be used when loading one's own data, not downloaded automatically from script)
- When the data is loaded, save the scene for faster future loading using 

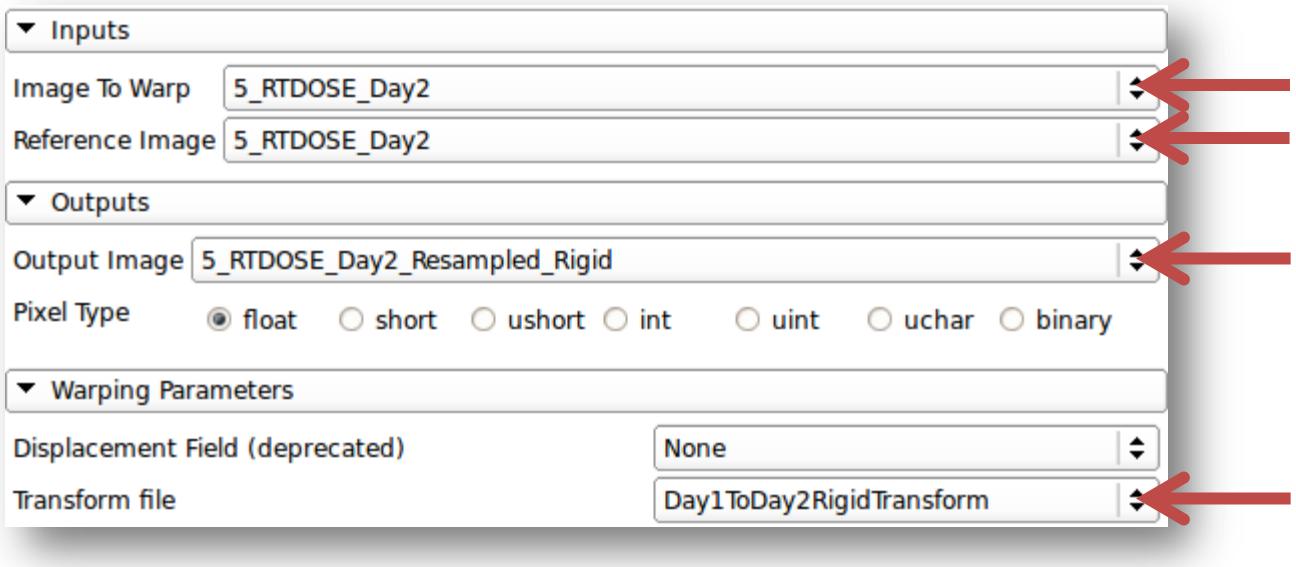
Registering CT volumes

- Go to *module list / Registration / General Registration (BRAINS)*
- Set the module parameters as seen in this image an click *Apply*



Resampling the day 2 dose volume

- Go to *module list / Registration / Resample Image (BRAINS)*
- Set the module parameters as seen in this image an click *Apply*



Accumulating doses

- Go to *module list / Radiotherapy / Dose Accumulation*
- Un-check *Show dose volumes only* checkbox
- Select reference dose volume *5: RTDOSE*
- Check volumes
  - 5: RTDOSE
  - 5_RTDOSE_Day2
- Open combobox in the *Output* section
- Select *Create new Volume*
- Click *Apply*
- Accumulate these volumes in the same way
  - 5: RTDOSE
  - 5_RTDOSE_Day2_Resampled_Rigid

Computing dose volume histograms

- Go to *Radiotherapy / Dose Volume Histogram*
- Select the first accumulated dose volume from the list
- Select *PTV* as the structure set
- Click *Compute DVH*
- Select second accumulated dose volume
- Compute DVH
- Do it for other structures, such as *BRSTEM* or *optOptic*
- Click *Select Chart*
- Click *Create new Chart*
- Check *Show/hide all*

Next item on agenda

- Acquire and build SlicerRT (on an existing Slicer)
- Use python console in Slicer
- Create extension and module skeletons
  - Use extension template
  - Use scripted module template
- Use SlicerRT
- **Write a simple scripted module**

Write our scripted module #1

- Open *MyExtension/MyScriptedModule/MyScriptedModule.py*
- Rename the module and put it in our extension category

```
class MyScriptedModule:
  def __init__(self, parent):
    parent.title = "Center of Masses" # T
    parent.categories = ["MyExtension"]
```

- Create widgets to specify inputs:
  Under `def setup(self):` look for *input volume selector*, and change

```
self.inputSelector.addAttribute( "vtkMRMLScalarVolumeNode", "LabelMap", 1 )
self.inputSelector.selectNodeUponCreation = False
```

  and

```
self.inputSelector.setToolTip( "Pick the first input to the algorithm." )
parametersFormLayout.addRow("Input Volume 1: ", self.inputSelector)
```

- Create second input selector
  - Rename *outputSelector* to *input2Selector* everywhere *setUp* and *onApplyButton* and *onSelect* functions
  - Set the same settings as for the first one

Write our scripted module #2

- Look for *class MyScriptedModuleLogic* on the bottom
- Insert this code above the *Run* function

```
def getCenterOfMass(self,volumeNode):
  centerOfMass = [0,0,0]

  numberOfStructureVoxels = 0
  sumX = sumY = sumZ = 0

  volume = volumeNode.GetImageData()

  for z in xrange(volume.GetExtent()[4], volume.GetExtent()[5]+1):
    for y in xrange(volume.GetExtent()[2], volume.GetExtent()[3]+1):
      for x in xrange(volume.GetExtent()[0], volume.GetExtent()[1]+1):
        voxelValue = volume.GetScalarComponentAsDouble(x,y,z,0)
        if voxelValue>0:
          numberOfStructureVoxels = numberOfStructureVoxels+1
          sumX = sumX + x
          sumY = sumY + y
          sumZ = sumZ + z

  if numberOfStructureVoxels > 0:
    centerOfMass[0] = sumX / numberOfStructureVoxels
    centerOfMass[1] = sumY / numberOfStructureVoxels
    centerOfMass[2] = sumZ / numberOfStructureVoxels

  print('Center of mass for \'' + volumeNode.GetName() + '\': ' + repr(centerOfMass))
  return centerOfMass
```

Write our scripted module #3

- Insert this code in the *Run* function above the *return* statement

```
def run(self,inputVolume,input2Volume):
  """
  Run the actual algorithm
  """
  center1 = self.getCenterOfMass(inputVolume)
  center2 = self.getCenterOfMass(input2Volume)

  self.translation = []
  for i in [0,1,2]:
    self.translation.append(center2[i] - center1[i])

  return True
```

- Displaying the output

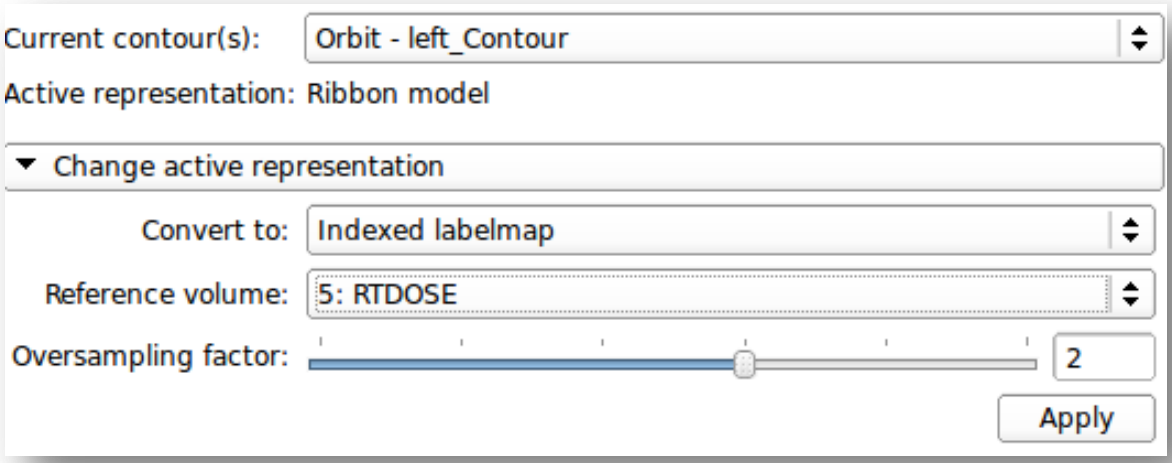
```
 def onApplyButton(self):
   logic = MyScriptedModuleLogic()
   print("Run the algorithm")
   logic.run(self.inputSelector.currentNode(), self.outputSelector.currentNode())
   self.outputField.setText('(' + repr(logic.translation[0]) + ', ' + repr
(logic.translation[1]) + ', ' + repr(logic.translation[2]) + ')')
```

- Create the text field under the *Apply* button

```
self.outputLabel = qt.QLabel()
parametersFormLayout.addRow(self.outputLabel)
```

Try our scripted module

- Build the extension (go to its bin directory and *make*)
- Load data using the demo module (load the scene we saved)
- Convert two structures, for example the two eyeballs (*Orbit…*) using the *Radiotherapy / Contours* module



- Go to our module in *MyExtension / Center of Masses*
- Set the two labelmaps as input
- Press *Apply*

Last item on agenda

- Acquire and build SlicerRT (on an existing Slicer)
- Use python console in Slicer
- Create extension and module skeletons
  - Use extension template
  - Use scripted module template
- Use SlicerRT
- Write a simple scripted module
- **Write a simple CLI module**

Write our CLI module #1

- Open *MyExtension/SuperCLIModuleTemplate/SuperCLIModuleTemplate.xml*
- Rename the module and put it in our extension category

```
<executable>
  <category>MyExtension</category>
  <title>Morphology</title>
```

Write our CLI module #2

- Create controls

```
<parameters>
  <string-enumeration>
    <name>operation</name>
    <longflag>operation</longflag>
    <label>Operation</label>
    <description><![CDATA[Type of operation]]></description>
    <default>Dilate</default>
    <element>Dilate</element>
    <element>Erode</element>
  </string-enumeration>
  <label>IO</label>
  <description><![CDATA[Input/output parameters]]></description>
  <image>
    <name>inputVolume</name>
    <label>Input Volume</label>
    <channel>input</channel>
    <index>0</index>
    <description><![CDATA[Input volume]]></description>
  </image>
  <image>
    <name>outputVolume</name>
    <label>Output Volume</label>
    <channel>output</channel>
    <index>1</index>
    <description><![CDATA[Output Volume]]></description>
  </image>
</parameters>
```

Write our CLI module #3

- Open *MyExtension/SuperCLIModuleTemplate/SuperCLIModuleTemplate.cxx*
- Include needed headers

```
#include "itkBinaryBallStructuringElement.h"
#include "itkGrayscaleDilateImageFilter.h"
#include "itkGrayscaleErodeImageFilter.h"
```

- Define used types (replace *FilterType* definition)

```
typedef itk::BinaryBallStructuringElement<
  InputPixelType, 3> StructuringElementType;
typedef itk::GrayscaleDilateImageFilter<
  InputImageType, OutputImageType, StructuringElementType> DilateFilterType;
typedef itk::GrayscaleErodeImageFilter<
  InputImageType, OutputImageType, StructuringElementType> ErodeFilterType;
```

Write our CLI module #4

- Add processing code

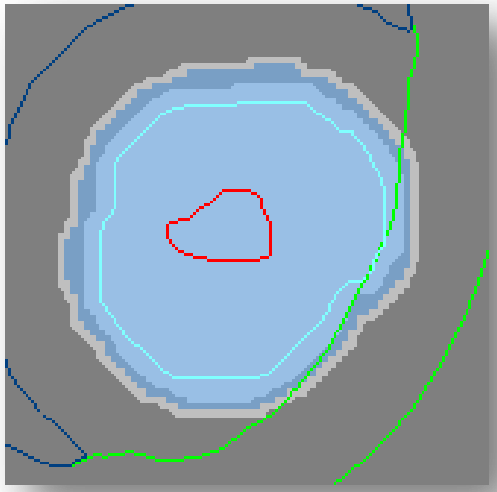
```
reader->SetFileName( inputVolume.c_str() );

typename OutputImageType::Pointer outputImage = NULL;
if (operation == "Dilate")
{
  typename DilateFilterType::Pointer filter = DilateFilterType::New();
  filter->SetInput( reader->GetOutput() );
  filter->Update();
  outputImage = filter->GetOutput();
}
else if (operation == "Erode")
{
  typename ErodeFilterType::Pointer filter = ErodeFilterType::New();
  filter->SetInput( reader->GetOutput() );
  filter->Update();
  outputImage = filter->GetOutput();
}
else
{
  return EXIT_FAILURE;
}

typename WriterType::Pointer writer = WriterType::New();
writer->SetFileName( outputVolume.c_str() );
writer->SetInput( outputImage );
writer->SetUseCompression(1);
writer->Update();
```

Try our CLI module

- Load data using the demo module (load the scene we saved)
- Convert one structure, for example *PTV* using the *Contours* module
- Go to our module in *MyExtension / Morphology*
- Set the *PTV* labelmap as input
- Create output volume
- Press *Apply*
- Change *Operation to Erode*
- Create output volume
- Press *Apply*



- Show volumes in the viewers: access slice toolbar, expand and turn on lock ( ), select output volumes and *PTV* labelmap

Congratulations!

Thanks for attending!