**John Fredy Ochoa Gómez**

**Programa de bioingeneniería**

**Universidad de Antioquia**

**Manipulación de la escena**

Para esta primera parte se debe tener cargada la escena **“slicer4minute”**

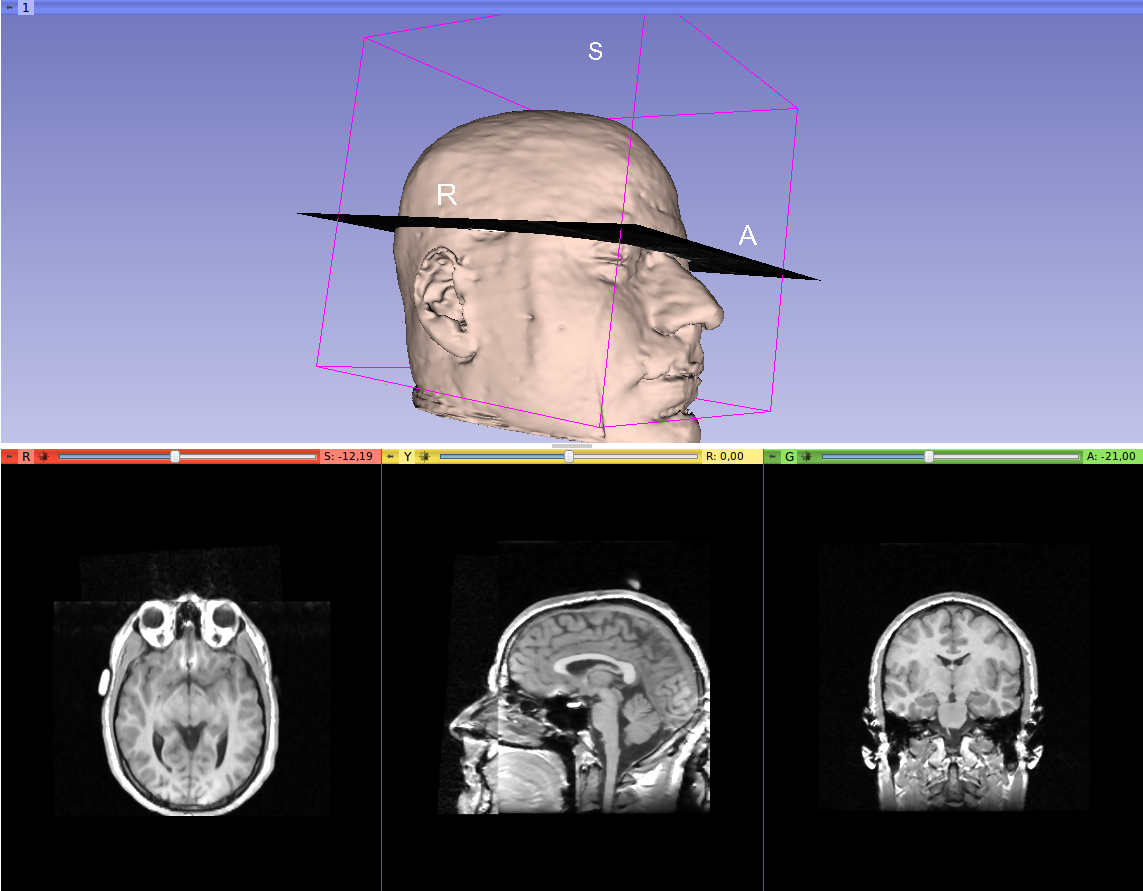
La visualización en medicina hace parte de la visualización científica la cual tiene como propósitos fundamentales:

* Exploración de datos
* Validación de hipótesis
* Presentación de resultados

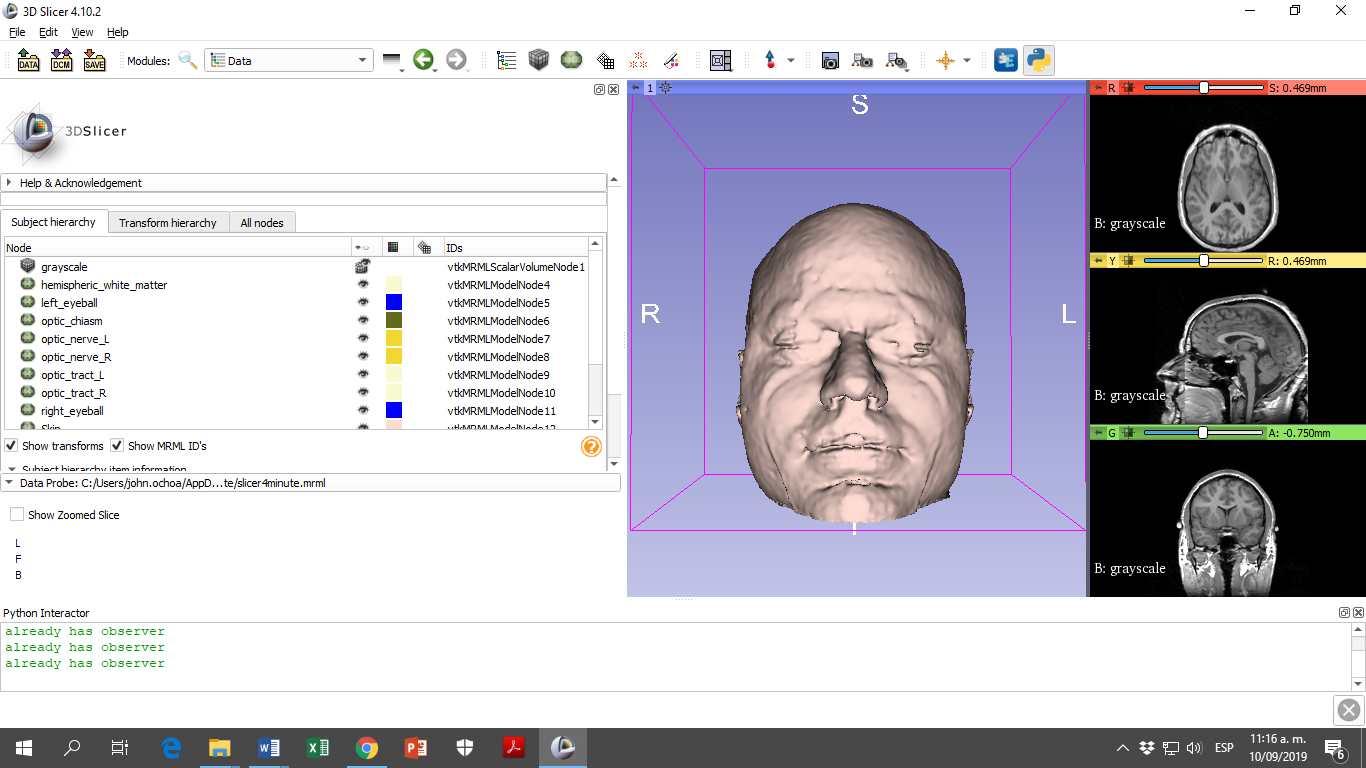
La visualización en medicina se basa en el trabajo con conjuntos de datos volumétricos los cuales tienen como limitante su naturaleza discreta.

El 3DSlicer es una plataforma de visualización de imágenes médicas multipropósito. Algunos elementos del 3DSlicer son:

* **Escena:** Representación gráfica de la información contenida en imágenes, puntos, transformaciones, medidas, superficies poligonales, etc. Toda esta información se ve en forma de árbol mediante el módulo “Data” del Slicer.

Por ejemplo, la siguiente escena

Se visualiza en el módulo “*Data*” como:



El objeto ***slicer.mrmlScene*** contiene toda la información de la escena en el Slicer.

Las clases que permiten seleccionar información cargada en el Slicer, por ejemplo, una lista de selección de imágenes, obtiene la información sobre las imágenes disponibles consultando internamente una referencia a la escena.

La escena actual se puede recuperar en una referencia llamada “*escena*” así:

***escena = slicer.mrmlScene;***

***print(type(escena))***

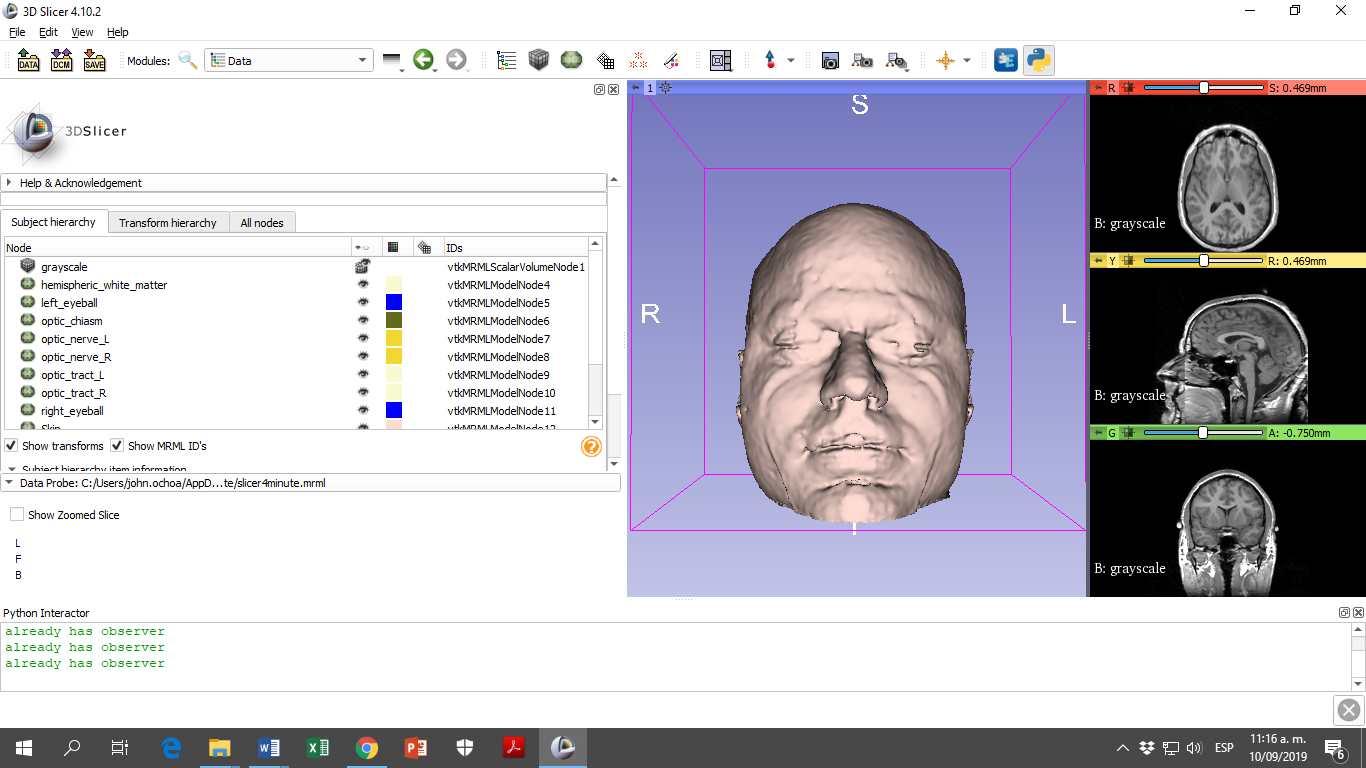
Algunos métodos de la clase vtkMRMLScene son:

Para el manejo de nodos:

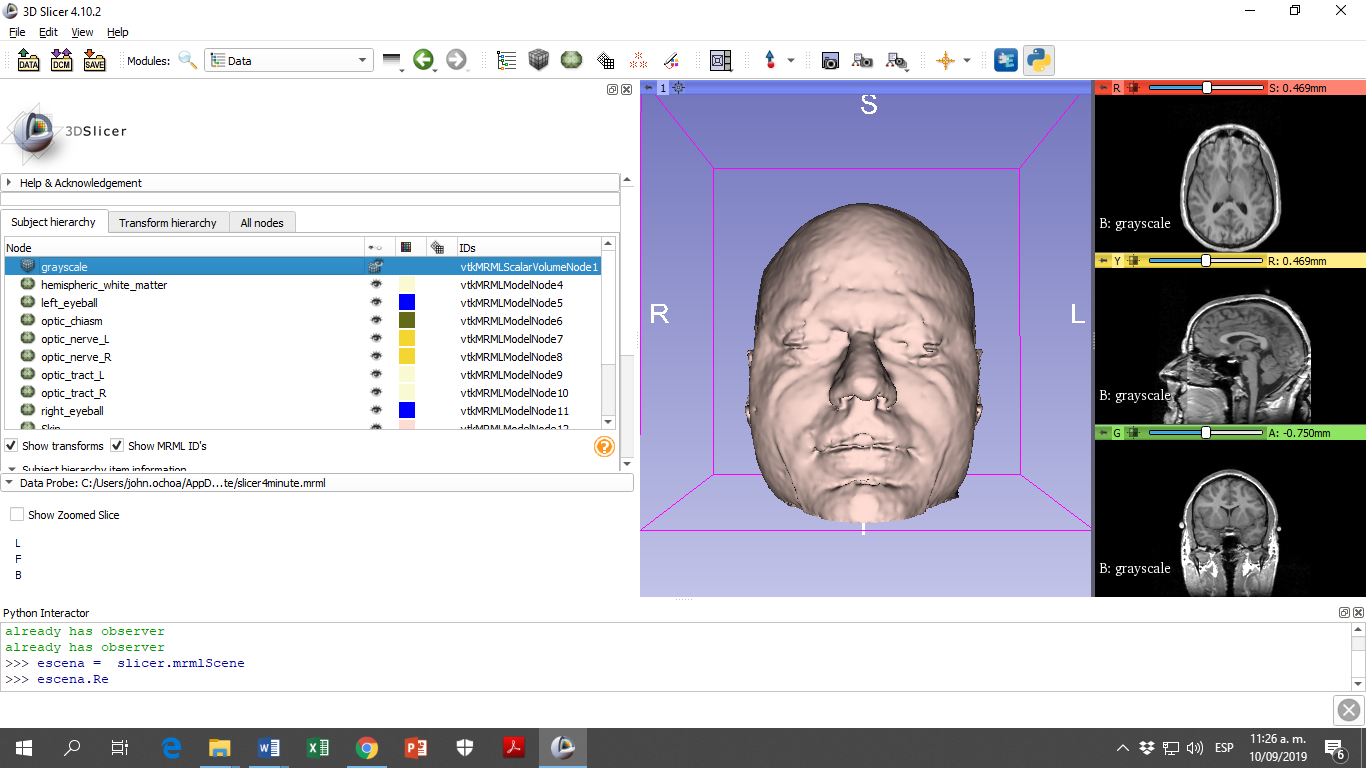
* AddNode(nodo): añade el nodo a la escena
* RemoveNode(nodo): elimina un nodo de la escena

Búsqueda de nodos:

* GetNthNode(numero\_nodo): permite obtener el nodo indicado por la variable entera 'numero\_nodo'
* GetNodeByID(id): permite obtener el nodo con identificador ID



Por ejemplo, es posible recuperar la resonancia que en la escena es ‘grayscale’



***volumen = escena.GetNodeByID('vtkMRMLScalarVolumeNode1')***

En esta referencia conocida como “*volumen*” se encuentran los datos que constituyen la imagen. Para obtener dichos datos se utiliza:

***datos\_imagenes = volumen.GetImageData();***

El código anterior devuelve un tipo de dato ***vtk.vtkImageData*** el cual es propio de vtk y consiste en un arreglo de datos (una matriz). A partir de este podemos conocer propiedades de la imagen.

Por otro lado, para construir un volumen se utiliza:

***volumen = slicer.vtkMRMLScalarVolumeNode()***

Es posible asociar una matriz de datos ***vtk.vtkImageData***, en este caso llamada 'imagen', a un volumen previamente creado mediante la instrucción:

***volumen.SetAndObserveImageData(imagen)***

**Manipulación de imágenes**

Dadas las capacidades de los equipos de tomografía y resonancia modernos es cada vez más común obtener información de una estructura mediante cortes de alta resolución y con iguales dimensiones. Esta información volumétrica permite realizar reconstrucciones tridimensionales haciendo uso de diferentes algoritmos.

En **vtk** estos volúmenes son representados como **vtkImageData** y en 3DSlicer como ***vtkMRMLScalarVolumeNode***. Hay que recordar que las librerías de 3DSlicer son una extensión de las de VTK por lo que un ***vtkMRMLScalarVolumeNode*** fue concebido para contener un ***vtkImageData***.

**vtkImageData**

Es la clase básica para almacenar imágenes en VTK. Está definida por cuatro elementos:

* Dimensiones
* Origen
* Espaciamiento o tamaño de los voxels
* Tipo de dato que puede ser flotante, entero, entero corto, etc

Una imagen se puede crear a partir de código Python así:

***imagen = vtk.vtkImageData()***

***imagen.SetDimensions([10,10,2]) #tamanio de la matriz***

***imagen.SetOrigin([0,0,0]) #origen de la imagen***

***imagen.SetSpacing([0.78,0.78,1.5]) #espaciamiento entre puntos de la matriz***

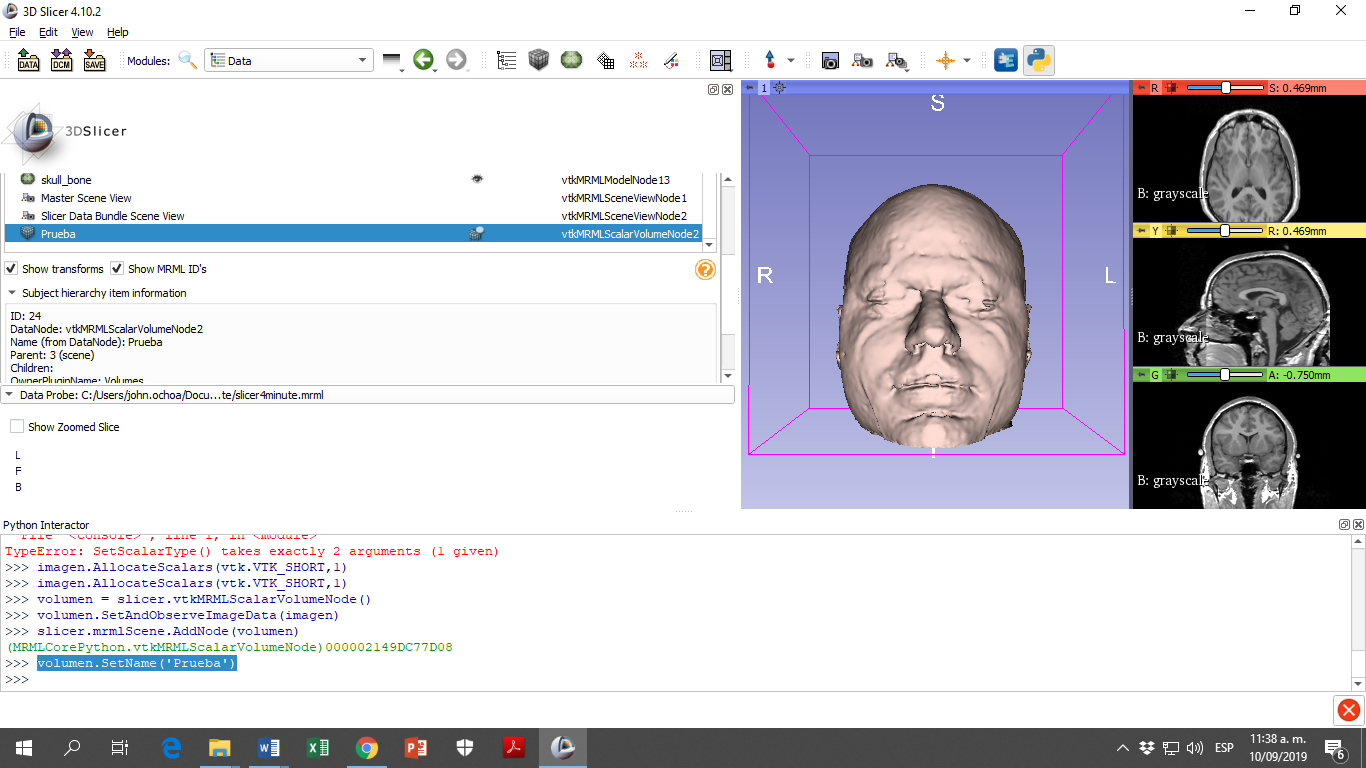
***imagen.AllocateScalars(vtk.VTK\_SHORT,1) #tipo de datos***

***volumen = slicer.vtkMRMLScalarVolumeNode()***

***volumen.SetAndObserveImageData(imagen)***

***volumen.SetName('Prueba')***

***slicer.mrmlScene.AddNode(volumen)***



Mediante los flujos de procesamiento disponibles en VTK es posible obtener las imágenes a partir de filtros. Algunos filtros disponibles son:

– Suavizado: vtkImageGaussianSmooth, vtkImageMedian3D

– Computo de gradientes y laplacianos: vtkImageGradient, vtkImageLaplacian

– Operaciones Fourier: vtkImageFFT, vtkImageRFT

– Remuestreo, reformateo: vtkImageResample, vtkImageReslice

– Permutación, giro: vtkImageFlip, vtkImagePermute

Es posible acceder a algunas propiedades de la imagen

***datos = imagen.GetPointData().GetScalars()***

***print(type(datos))***

***print(datos.GetRange()) #rango de valores posibles en las entradas de la matriz***

***print(datos.GetDataTypeMax())***

***print(datos.GetDataTypeMin())***

Los datos de una imagen se pueden manipular como un ndarray a partir del nombre de la imagen en la escena:

***matriz\_ndarray = slicer.util.array('Prueba')***

O también desde una vtkImageData

***matriz\_ndarray = vtk.util.numpy\_support.vtk\_to\_numpy(imagen.GetPointData().GetScalars())***

Proceso que es invertible

***matriz = vtk.util.numpy\_support.numpy\_to\_vtk(matriz\_ndarray)***

El ***vtkMRMLScalarVolumeNode*** tiene un par de matrices de transformación, diferentes a las matrices de datos, que permiten hacer las relaciones entre entradas de la matriz (i,j,k) y coordenadas en el espacio derecha-izquierda, anterior-posterior, superior-inferior (RAS)

***ras2ijk = vtk.vtkMatrix4x4()***

***ijk2ras = vtk.vtkMatrix4x4()***

***volumen.GetRASToIJKMatrix(ras2ijk)***

***volumen.GetIJKToRASMatrix(ijk2ras)***

Estas matrices son del tipo [***vtkMatrix4x4***](http://www.vtk.org/doc/nightly/html/classvtkMatrix4x4.html)que es una clase que representa y manipula matrices 4x4 útiles para el manejo de coordenadas homogéneas.

**Para resolver:**

Cómo puedo modificar las entradas de una matriz [***vtkMatrix4x4***](http://www.vtk.org/doc/nightly/html/classvtkMatrix4x4.html)usando el método ***setElement***?

**Fiducials o marcas**

En Slicer podemos ubicar marcas desde las cuales podemos extraer intensidades o valores de interés

El siguiente código imprime las posiciones RAS de un conjunto de marcas agregadas en la escena

***markup = slicer.mrmlScene. GetNodeByID ("vtkMRMLMarkupsFiducialNode1")***

***ras = [0,0,0]***

***numFids = markup.GetNumberOfFiducials()***

***for i in range(numFids):***

***markup.GetNthFiducialPosition(i, ras)***

***print str(ras)***

**Para resolver:**

De la imagen 4D obtener los valores de intensidad asociados con un fiducials para cada volumen. Probar con valores negros y blancos e intermedios para tener certeza de que se está obteniendo el valor indicado

**Correr módulos CLI**

Podemos acceder a los módulos mediante:

***slicer.modules.<TAB>***

Segmentación

***cliModule = slicer.modules.simpleregiongrowingsegmentation***

***n = cliModule.cliModuleLogic().CreateNode()***

***for groupIndex in range(n.GetNumberOfParameterGroups()):***

***for parameterIndex in range(n.GetNumberOfParametersInGroup(groupIndex)):***

***print('Parameter ({0}/{1}): {2} ({3})'.format(groupIndex, parameterIndex, n.GetParameterName(groupIndex, parameterIndex), n.GetParameterLabel(groupIndex, parameterIndex)))***

Parameter (0/0): smoothingIterations (Smoothing iterations)

Parameter (0/1): timestep (Timestep)

Parameter (1/0): iterations (Number of iterations)

Parameter (1/1): multiplier (Multiplier)

Parameter (1/2): neighborhood (Neighborhood Radius)

Parameter (1/3): labelvalue (Output Label Value)

Parameter (1/4): seed (Seeds)

Parameter (2/0): inputVolume (Input Volume)

Parameter (2/1): outputVolume (Output Volume)

***#parametros para la operacion de registro***

***parameters = {}***

***parameters['smoothingIterations'] = 5.0***

***parameters['timestep'] = 0.0625***

***parameters['iterations'] = 5***

***parameters['multiplier'] = 2.5***

***parameters['neighborhood'] = 1***

***parameters['labelvalue'] = 2***

***fiducials = slicer.mrmlScene.GetNodeByID('vtkMRMLMarkupsFiducialNode1')***

***parameters['seed'] = fiducials.GetID()***

***volumen\_entrada = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')***

***parameters['inputVolume'] = volumen\_entrada.GetID()***

***volumen\_salida = slicer.vtkMRMLLabelMapVolumeNode()***

***slicer.mrmlScene.AddNode(volumen\_salida)***

***parameters['outputVolume'] = volumen\_salida.GetID()***

***cliModule = slicer.modules.simpleregiongrowingsegmentation***

***cliNode = slicer.cli.run(cliModule,None,parameters,wait\_for\_completion=True)***

**Cuantificación**

***import numpy***

***volume = array('4D')***

***label = array('LabelMapVolume')***

***points = numpy.where( label == 1 ) # or use another label number depending on what you segmented***

***values = volume[points] # this will be a list of the label values***

***values.mean() # should match the mean value of LabelStatistics calculation as a double-check***

**Registro multivolumen**

**Cargar volumen 4D**

***#%%***

***fileName = "/Users/jfochoa/Desktop/clases\_semana/semana\_8/PDI/uroresonancia/anpaos/4D.hdr"***

***"""Try to read a 4D nifti file as a multivolume"""***

***print('trying to read %s' % fileName)***

***# use the vtk reader which seems to handle most nifti variants well***

***reader = vtk.vtkNIFTIImageReader()***

***reader.SetFileName(fileName)***

***reader.SetTimeAsVector(True)***

***reader.Update()***

***header = reader.GetNIFTIHeader()***

***qFormMatrix = reader.GetQFormMatrix()***

***if not qFormMatrix:***

***print('Warning: %s does not have a QFormMatrix - using Identity')***

***qFormMatrix = vtk.vtkMatrix4x4()***

***spacing = reader.GetOutputDataObject(0).GetSpacing()***

***timeSpacing = reader.GetTimeSpacing()***

***nFrames = reader.GetTimeDimension()***

***print(nFrames)***

***if header.GetIntentCode() != header.IntentTimeSeries:***

***intentName = header.GetIntentName()***

***if not intentName:***

***intentName = 'Nothing'***

***print('Warning: %s does not have TimeSeries intent, instead it has \"%s\"' % (fileName,intentName))***

***print('Trying to read as TimeSeries anyway')***

***units = header.GetXYZTUnits()***

***# try to account for some of the unit options***

***# (Note: no test data available but we hope these are right)***

***if units & header.UnitsMSec == header.UnitsMSec:***

***timeSpacing /= 1000.***

***if units & header.UnitsUSec == header.UnitsUSec:***

***timeSpacing /= 1000. / 1000.***

***spaceScaling = 1.***

***if units & header.UnitsMeter == header.UnitsMeter:***

***spaceScaling \*= 1000.***

***if units & header.UnitsMicron == header.UnitsMicron:***

***spaceScaling /= 1000.***

***spacing = [e \* spaceScaling for e in spacing]***

***# create frame labels using the timing info from the file***

***# but use the advanced info so user can specify offset and scale***

***volumeLabels = vtk.vtkDoubleArray()***

***volumeLabels.SetNumberOfTuples(nFrames)***

***frameLabelsAttr = ''***

***for i in range(nFrames):***

***frameId = 0 + timeSpacing \* 0.1 \* i***

***volumeLabels.SetComponent(i, 0, frameId)***

***frameLabelsAttr += str(frameId)+','***

***frameLabelsAttr = frameLabelsAttr[:-1]***

***# create the display node***

***mvDisplayNode = slicer.mrmlScene.CreateNodeByClass('vtkMRMLMultiVolumeDisplayNode')***

***mvDisplayNode.SetScene(slicer.mrmlScene)***

***slicer.mrmlScene.AddNode(mvDisplayNode)***

***mvDisplayNode.SetReferenceCount(mvDisplayNode.GetReferenceCount()-1)***

***mvDisplayNode.SetDefaultColorMap()***

***# spacing and origin are in the ijkToRAS, so clear them from image data***

***imageChangeInformation = vtk.vtkImageChangeInformation()***

***imageChangeInformation.SetInputConnection(reader.GetOutputPort())***

***imageChangeInformation.SetOutputSpacing( 1, 1, 1 )***

***imageChangeInformation.SetOutputOrigin( 0, 0, 0 )***

***imageChangeInformation.Update()***

***# QForm includes directions and origin, but not spacing so add that***

***# here by multiplying by a diagonal matrix with the spacing***

***scaleMatrix = vtk.vtkMatrix4x4()***

***for diag in range(3):***

***scaleMatrix.SetElement(diag, diag, spacing[diag])***

***ijkToRAS = vtk.vtkMatrix4x4()***

***ijkToRAS.DeepCopy(qFormMatrix)***

***vtk.vtkMatrix4x4.Multiply4x4(ijkToRAS, scaleMatrix, ijkToRAS)***

***mvNode = slicer.vtkMRMLMultiVolumeNode()***

***mvNode.SetIJKToRASMatrix(ijkToRAS)***

***mvNode.SetAndObserveDisplayNodeID(mvDisplayNode.GetID())***

***mvNode.SetAndObserveImageData(imageChangeInformation.GetOutputDataObject(0))***

***mvNode.SetNumberOfFrames(nFrames)***

***# set the labels and other attributes, then display the volume***

***mvNode.SetLabelArray(volumeLabels)***

***mvNode.SetLabelName("MultiVolumen")***

***mvNode.SetAttribute('MultiVolume.FrameLabels',frameLabelsAttr)***

***mvNode.SetAttribute('MultiVolume.NumberOfFrames',str(nFrames))***

***mvNode.SetAttribute('MultiVolume.FrameIdentifyingDICOMTagName','')***

***mvNode.SetAttribute('MultiVolume.FrameIdentifyingDICOMTagUnits','')***

***mvNode.SetName(str(nFrames)+' frames NIfTI MultiVolume')***

***#the node is inserted in the scene***

***slicer.mrmlScene.AddNode(mvNode)***

En el siguiente ejemplo permite llamar un módulo de Slicer trabajando sobre una imagen 4D previamente cargada y con ID ***vtkMRMLMultiVolumeNode1***

***escena = slicer.mrmlScene;***

***volumen4D = escena.GetNodeByID('vtkMRMLMultiVolumeNode1')***

***imagenvtk4D = volumen4D.GetImageData()***

***numero\_imagenes = volumen4D.GetNumberOfFrames()***

***print('imagenes: ' + str(numero\_imagenes))***

***#filtro vtk para descomponer un volumen 4D***

***extract1 = vtk.vtkImageExtractComponents()***

***extract1.SetInputData(imagenvtk4D)***

***#matrices de transformacion***

***ras2ijk = vtk.vtkMatrix4x4()***

***ijk2ras = vtk.vtkMatrix4x4()***

***#le solicitamos al volumen original que nos devuelva sus matrices***

***volumen4D.GetRASToIJKMatrix(ras2ijk)***

***volumen4D.GetIJKToRASMatrix(ijk2ras)***

***#creo un volumen nuevo que será el volumen fijo a registrar***

***volumenFijo = slicer.vtkMRMLScalarVolumeNode();***

***#le asigno las transformaciones***

***volumenFijo.SetRASToIJKMatrix(ras2ijk)***

***volumenFijo.SetIJKToRASMatrix(ijk2ras)***

***#le asigno el volumen 3D fijo***

***imagen\_fija = extract1.SetComponents(0)***

***extract1.Update()***

***volumenFijo.SetName('fijo')***

***volumenFijo.SetAndObserveImageData(extract1.GetOutput())***

***extract1.Update()***

***#anado el nuevo volumen a la escena***

***escena.AddNode(volumenFijo)***

***# extraigo la imagen movil***

***imagen\_movil = extract1.SetComponents(59) #Seleccionamos un volumen lejano***

***extract1.Update()***

***#Creo un volumen movil, y realizamos el mismo procedimiento que con el fijo***

***volumenMovil = slicer.vtkMRMLScalarVolumeNode();***

***volumenMovil.SetRASToIJKMatrix(ras2ijk)***

***volumenMovil.SetIJKToRASMatrix(ijk2ras)***

***volumenMovil.SetAndObserveImageData(extract1.GetOutput())***

***volumenMovil.SetName('movil')***

***escena.AddNode(volumenMovil)***

***#creamos la transformada para alinear los volumenes***

***transformadaSalida = slicer.vtkMRMLLinearTransformNode()***

***transformadaSalida.SetName('Transformada de registro')***

***slicer.mrmlScene.AddNode(transformadaSalida)***

***#parametros para la operacion de registro***

***parameters = {}***

***#parameters['InitialTransform'] = transI.GetID() #si hay una transformación inicial***

***parameters['fixedVolume'] = volumenFijo.GetID()***

***parameters['movingVolume'] = volumenMovil.GetID()***

***parameters['transformType'] = 'Rigid'***

***parameters['outputTransform'] = transformadaSalida.GetID()***

***cliNode = slicer.cli.run(slicer.modules.brainsfit,None,parameters, wait\_for\_completion=True)***

**Para resolver:**

¿Cómo se podrían registrar todos los volúmenes de la imagen 4D tomando la primera como fija y las otras como móviles?

¿Cómo se puede crear un volumen 4D que tenga los volúmenes corregidos luego del proceso de corregistro? (Se podría crear una copia y el filtro ***vtkImageReslice***)

¿Cómo se podría corregir deformaciones por la respiración?

**Graficar Valores**

***import numpy as np;***

***numero\_frames = 100;***

***data = np.zeros((100,2));***

***for i in range(numero\_frames):***

***data[i,0] = i #x data***

***data[i,1] = i\*\*2; #y data***

***#plot(narray, xColumnIndex=-1, columnNames=None, title=None, show=True, nodes=None)***

***##:param narray: input numpy array containing data series in columns.***

***# :param xColumnIndex: index of column that will be used as x axis.***

***# If it is set to negative number (by default) then row index will be used as x coordinate.***

***# :param columnNames: names of each column of the input array.***

***# :param title: title of the chart. Plot node names are set based on this value.***

***# :param nodes: plot chart, table, and list of plot series nodes.***

***# Specified in a dictionary, with keys: 'chart', 'table', 'series'.***

***# Series contains a list of plot series nodes (one for each table column).***

***# The parameter is used both as an input and output.***

***# :return: plot chart node. Plot chart node provides access to chart properties and plot series nodes***

***chartNode = slicer.util.plot(data, xColumnIndex=0, columnNames=['X', 'X^2']);***

***chartNode.SetXAxisTitle('X')***

***chartNode.SetYAxisTitle('Y')***

***chartNode.LegendVisibilityOff()***

***chartNode.SetTitle('Prueba')***

**Para resolver:**

De la imagen 4D obtener los valores de intensidad asociados con un fiducials para cada volumen. Graficar los valores 4D obtenidos

*Running CLI module*

Podemos acceder a los módulos mediante:

slicer.modules.<TAB>

segmentación

cliModule = slicer.modules.simpleregiongrowingsegmentation

n = cliModule.cliModuleLogic().CreateNode()

for groupIndex in range(n.GetNumberOfParameterGroups()):

for parameterIndex in range(n.GetNumberOfParametersInGroup(groupIndex)):

print('Parameter ({0}/{1}): {2} ({3})'.format(groupIndex, parameterIndex, n.GetParameterName(groupIndex, parameterIndex), n.GetParameterLabel(groupIndex, parameterIndex)))

Parameter (0/0): smoothingIterations (Smoothing iterations)

Parameter (0/1): timestep (Timestep)

Parameter (1/0): iterations (Number of iterations)

Parameter (1/1): multiplier (Multiplier)

Parameter (1/2): neighborhood (Neighborhood Radius)

Parameter (1/3): labelvalue (Output Label Value)

Parameter (1/4): seed (Seeds)

Parameter (2/0): inputVolume (Input Volume)

Parameter (2/1): outputVolume (Output Volume)

#parametros para la operacion de registro

parameters = {}

parameters['smoothingIterations'] = 5.0

parameters['timestep'] = 0.0625

parameters['iterations'] = 5

parameters['multiplier'] = 2.5

parameters['neighborhood'] = 1

parameters['labelvalue'] = 2

fiducials = slicer.mrmlScene.GetNodeByID('vtkMRMLMarkupsFiducialNode1')

parameters['seed'] = fiducials.GetID()

volumen\_entrada = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')

parameters['inputVolume'] = volumen\_entrada.GetID()

volumen\_salida = slicer.vtkMRMLLabelMapVolumeNode()

slicer.mrmlScene.AddNode(volumen\_salida)

parameters['outputVolume'] = volumen\_salida.GetID()

cliModule = slicer.modules.simpleregiongrowingsegmentation

cliNode = slicer.cli.run(cliModule,None,parameters,wait\_for\_completion=True)

Cuantificación

import numpy

volume = array('4D')

label = array('LabelMapVolume')

points = numpy.where( label == 1 ) # or use another label number depending on what you segmented

values = volume[points] # this will be a list of the label values

values.mean() # should match the mean value of LabelStatistics calculation as a double-check

numpy.savetxt('values.txt', values)

En nuestro caso, evaluaremos el filtro de difusión anisotropica

cliModule = slicer.modules.gradientanisotropicdiffusion

n = cliModule.cliModuleLogic().CreateNode()

for groupIndex in range(n.GetNumberOfParameterGroups()):

for parameterIndex in range(n.GetNumberOfParametersInGroup(groupIndex)):

print('Parameter ({0}/{1}): {2} ({3})'.format(groupIndex, parameterIndex, n.GetParameterName(groupIndex, parameterIndex), n.GetParameterLabel(groupIndex, parameterIndex)))

Parameter (0/0): conductance (Conductance)

Parameter (0/1): numberOfIterations (Iterations)

Parameter (0/2): timeStep (Time Step)

Parameter (1/0): inputVolume (Input Volume)

Parameter (1/1): outputVolume (Output Volume)

Parameter (2/0): useImageSpacing (Use image spacing)

#parametros para la operacion de registro

parameters = {}

parameters['conductance'] = 1.0

parameters['numberOfIterations'] = 10

parameters['timeStep'] = 0.05

volumen\_entrada = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')

volumen\_salida = slicer.vtkMRMLScalarVolumeNode()

slicer.mrmlScene.AddNode(volumen\_salida)

parameters['inputVolume'] = volumen\_entrada.GetID()

parameters['outputVolume'] = volumen\_salida.GetID()

cliModule = slicer.modules.gradientanisotropicdiffusion

cliNode = slicer.cli.run(cliModule,None,parameters,wait\_for\_completion=True)

*Get list of parameter names*

The following script prints all the parameter names of a CLI parameter node:

cliModule = slicer.modules.grayscalemodelmaker n=cliModule.cliModuleLogic().CreateNode() for groupIndex in range(n.GetNumberOfParameterGroups()): for parameterIndex in range(n.GetNumberOfParametersInGroup(groupIndex)): print('Parameter ({0}/{1}): {2} ({3})'.format(groupIndex, parameterIndex, n.GetParameterName(groupIndex, parameterIndex), n.GetParameterLabel(groupIndex, parameterIndex)))

*Running CLI in the background*

If the CLI module is executed using slicer.cli.run method then the CLI module runs in a background thread, so the call to grayModel will return right away and the user interface will not be blocked. The slicer.cli.run call returns a cliNode (an instance of [vtkMRMLCommandLineModuleNode](http://slicer.org/doc/html/classvtkMRMLCommandLineModuleNode.html)) which can be used to monitor the progress of the module.

In this example we create a simple callback that will be called whenever the cliNode is modified. The status will tell you if the nodes is Pending, Running, or Completed.

def printStatus(caller, event): print("Got a %s from a %s" % (event, caller.GetClassName())) if caller.IsA('vtkMRMLCommandLineModuleNode'): print("Status is %s" % caller.GetStatusString()) cliNode.AddObserver('ModifiedEvent', printStatus)

If you need to cancel the CLI, call

cliNode.Cance()

To get the log info for the process you can call

cliNode.GetOutputText()

and

cliNode.GetErrorText()

.

Proyecto

Opción 1:

* Construir un módulo que permita corregir movimiento y graficar curvas en volúmenes 4D (Podría utilizar o no elementos del módulo MultiVolumeSupport <https://www.slicer.org/wiki/Documentation/4.1/Modules/MultiVolumeExplorer> )

Opción 2: Proponer un proyecto de utilidad que haga uso del Slicer y deba programarse

Opción 3: Proponer un módulo de segmentación que utilice contornos activos ( Ver guía de referencia: <https://www.slicer.org/wiki/Modules:VMTKEasyLevelSetSegmentation> , <https://www.intechopen.com/books/medical-and-biological-image-analysis/active-contour-based-segmentation-techniques-for-medical-image-analysis> )

Información adicional se puede encontrar en: <https://www.slicer.org/wiki/Documentation/Nightly/ScriptRepository#Plotting>

Dejemos a los compañeros de próximos semestres material que enriquezca esta guía!

Examples

Start Slicer and bring up python console. Load a sample volume like this:

import SampleData sampleDataLogic = SampleData.SampleDataLogic() sampleDataLogic.downloadMRHead()

Get the volume node for that volume:

n = getNode('MRHead')

You can use Tab to see lists of methods for a class instance.

**Accessing Volume data as numpy array**

You can easily inspect and manipulate volume data using numpy and related code. In slicer you can do this:

a = array('MRHead')

and 'a' will be a pointer to the appropriate data (no data copying). If you get an error that 'array' is not defined then run 'import slicer.util' and use 'slicer.util.array'. Scalar volumes become three-dimensional arrays, while vector volumes become 4D, and tensor volumes are 5D. All arrays can be manipulated directly. After modification is completed, call Modified() method of the volume node to indicate that the image is modified and trigger display update.

The **array** method is intended for quick testing only, as multiple nodes may have the same name and various arrays may be retrieved from MRML nodes. In Slicer modules, it is recommended to use **arrayFromVolume** instead, which takes a MRML node as input.

volumeNode = getNode('MRHead') a = arrayFromVolume(volumeNode) # Increase image contrast a[:] = a \* 2.0 arrayFromVolumeModified(volumeNode)

If you don't process the data in-place but you have computation results in a numpy array, then you have to copy the contents of a numpy array into a volume, using **updateVolumeFromArray**:

import numpy as np import math def some\_func(x, y, z): return 0.5\*x\*x + 0.3\*y\*y + 0.5\*z\*z a = np.fromfunction(some\_func,(30,20,15)) volumeNode = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLScalarVolumeNode') volumeNode.CreateDefaultDisplayNodes() updateVolumeFromArray(volumeNode, a) setSliceViewerLayers(background=volumeNode)

*Passing Fiducials to CLIs via a Python Script*

import SampleData sampleDataLogic = SampleData.SampleDataLogic() head = sampleDataLogic.downloadMRHead() volumesLogic = slicer.modules.volumes.logic() headLabel = volumesLogic.CreateLabelVolume(slicer.mrmlScene, head, 'head-label') fiducialNode = slicer.vtkMRMLAnnotationFiducialNode() fiducialNode.SetFiducialWorldCoordinates((1,0,5)) fiducialNode.SetName('Seed Point') fiducialNode.Initialize(slicer.mrmlScene) fiducialsList = getNode('Fiducials List') params = {'inputVolume': head.GetID(), 'outputVolume': headLabel.GetID(), 'seed' : fiducialsList.GetID(), 'iterations' : 2} cliNode = slicer.cli.runSync(slicer.modules.simpleregiongrowingsegmentation, None, params)

*Running CLI in the background*

If the CLI module is executed using slicer.cli.run method then the CLI module runs in a background thread, so the call to grayModel will return right away and the user interface will not be blocked. The slicer.cli.run call returns a cliNode (an instance of [vtkMRMLCommandLineModuleNode](http://slicer.org/doc/html/classvtkMRMLCommandLineModuleNode.html)) which can be used to monitor the progress of the module.

In this example we create a simple callback that will be called whenever the cliNode is modified. The status will tell you if the nodes is Pending, Running, or Completed.

def printStatus(caller, event): print("Got a %s from a %s" % (event, caller.GetClassName())) if caller.IsA('vtkMRMLCommandLineModuleNode'): print("Status is %s" % caller.GetStatusString()) cliNode.AddObserver('ModifiedEvent', printStatus)

If you need to cancel the CLI, call

cliNode.Cance()

To get the log info for the process you can call

cliNode.GetOutputText()

and

cliNode.GetErrorText()

.

***#parametros para la operacion de registro***

***parameters = {}***

***#parameters['InitialTransform'] = transI.GetID() #si hay una transformación inicial***

***parameters['fixedVolume'] = volumenFijo.GetID()***

***parameters['movingVolume'] = volumenMovil.GetID()***

***parameters['transformType'] = 'Rigid'***

***parameters['outputTransform'] = transformadaSalida.GetID()***

***cliNode = slicer.cli.run(slicer.modules.brainsfit,None,parameters, wait\_for\_completion=True)***

*Get list of parameter names*

The following script prints all the parameter names of a CLI parameter node:

cliModule = slicer.modules.grayscalemodelmaker n=cliModule.cliModuleLogic().CreateNode() for groupIndex in range(n.GetNumberOfParameterGroups()): for parameterIndex in range(n.GetNumberOfParametersInGroup(groupIndex)): print('Parameter ({0}/{1}): {2} ({3})'.format(groupIndex, parameterIndex, n.GetParameterName(groupIndex, parameterIndex), n.GetParameterLabel(groupIndex, parameterIndex)))

#### Accessing slice vtkRenderWindows from slice views

The example below shows how to get the rendered slice window.

lm = slicer.app.layoutManager() redWidget = lm.sliceWidget('Red') redView = redWidget.sliceView() wti = vtk.vtkWindowToImageFilter() wti.SetInput(redView.renderWindow()) wti.Update() v = vtk.vtkImageViewer() v.SetColorWindow(255) v.SetColorLevel(128) v.SetInputConnection(wti.GetOutputPort()) v.Render()

## How to assign a volume to a Slice view ?

Assuming the MRHead sample data has been loaded, you could do the following:

red\_logic = slicer.app.layoutManager().sliceWidget("Red").sliceLogic() red\_cn = red\_logic.GetSliceCompositeNode() red\_logic.GetSliceCompositeNode().SetBackgroundVolumeID(slicer.util.getNode('MRHead').GetID())

Discussion: <http://slicer-devel.65872.n3.nabble.com/Assign-volumes-to-views-tt4028694.html>

## How to access vtkRenderer in Slicer 3D view ?

renderer = slicer.app.layoutManager().threeDWidget(0).threeDView().renderWindow().GetRenderers().GetFirstRenderer()

## How to get VTK rendering backend ?

backend = slicer.app.layoutManager().threeDWidget(0).threeDView().renderWindow().GetRenderingBackend()

## How to access displayable manager associated with a Slicer 2D or 3D view ?

As originally explained [here](http://slicer-devel.65872.n3.nabble.com/How-to-get-the-point-of-a-3D-model-based-on-the-fiducial-position-td4031760.html#a4031762), you could use the method getDisplayableManagers() available in any [qMRMLThreeDView](http://slicer.org/doc/html/classqMRMLThreeDView.html) and [qMRMLSliceView](http://slicer.org/doc/html/classqMRMLSliceView.html).

lm = slicer.app.layoutManager() for v in range(lm.threeDViewCount): td = lm.threeDWidget(v) ms = vtk.vtkCollection() td.getDisplayableManagers(ms) for i in range(ms.GetNumberOfItems()): m = ms.GetItemAsObject(i) if m.GetClassName() == "vtkMRMLModelDisplayableManager": print(m)

## How to center the 3D view on the scene ?

layoutManager = slicer.app.layoutManager() threeDWidget = layoutManager.threeDWidget(0) threeDView = threeDWidget.threeDView() threeDView.resetFocalPoint()

## Should I use 'old style' or 'new style' python classes in my scripted module ?

When python classes have no superclass specified they are 'old style' as described here [[1]](http://docs.python.org/2/reference/datamodel.html#new-style-and-classic-classes).

In general it doesn't matter for the classes in a scripted module, since they won't be subclassed either old or new style should be the same.

For other python code in slicer where you might be subclassing, it's better to use new style classes. See the class hierarchies in the [EditorLib](https://github.com/Slicer/Slicer/tree/master/Modules/Scripted/EditorLib) and the [DICOMLib](https://github.com/Slicer/Slicer/tree/master/Modules/Scripted/DICOM/DICOMLib) for examples.

## How to harden a transform ?

>>> n = getNode('Bone') >>> logic = slicer.vtkSlicerTransformLogic() >>> logic.hardenTransform(n)

How can I run slicer operations from a batch script?

Slicer --no-main-window --python-script /tmp/test.py

Contents of /tmp/test.py

# use a slicer scripted module logic from SampleData import SampleDataLogic SampleDataLogic().downloadMRHead() head = slicer.util.getNode('MRHead') # use a vtk class threshold = vtk.vtkImageThreshold() threshold.SetInputData(head.GetImageData()) threshold.ThresholdBetween(100, 200) threshold.SetInValue(255) threshold.SetOutValue(0) # use a slicer-specific C++ class erode = slicer.vtkImageErode() erode.SetInputConnection(threshold.GetOutputPort()) erode.SetNeighborTo4() erode.Update() head.SetAndObserveImageData(erode.GetOutputDataObject(0)) slicer.util.saveNode(head, "/tmp/eroded.nrrd") exit()