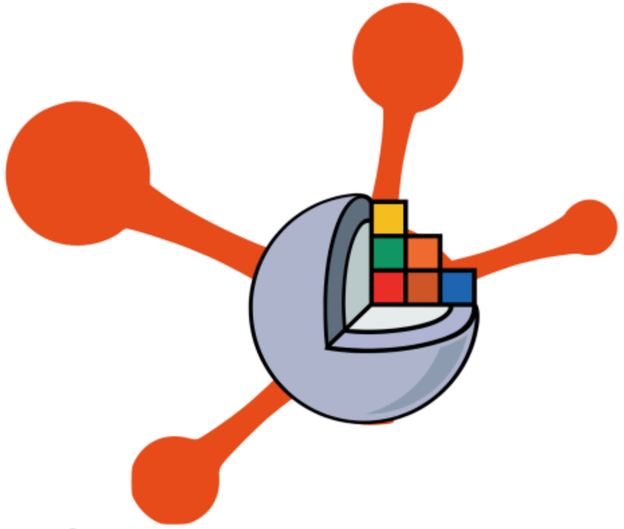
Integrating Simulations in 3D Slicer with SlicerSOFA

Rafael Palomar

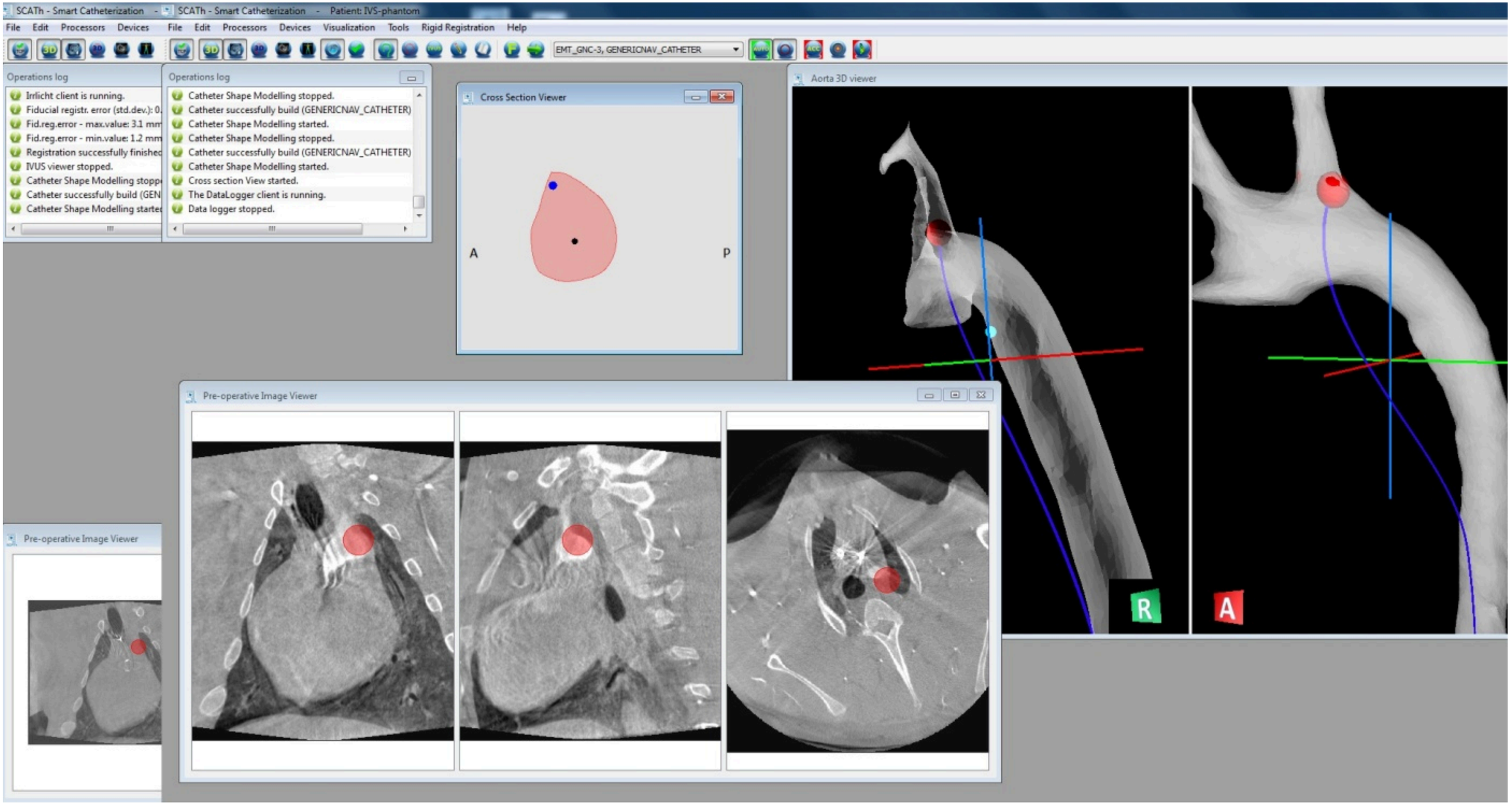


Slicer **SOFA**

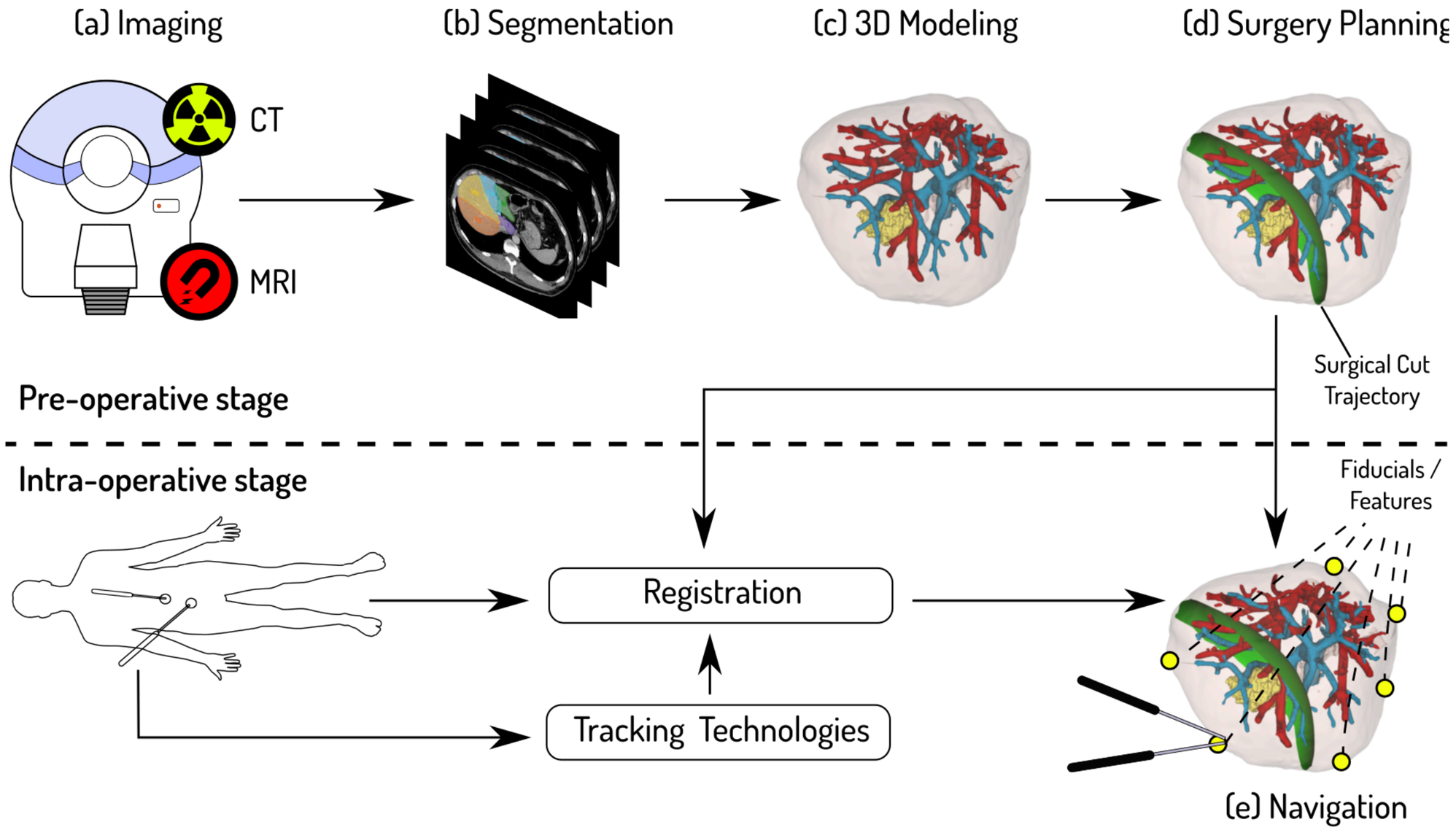
Introduction

Context: Image-Guided Therapies (I)

Context: Image-Guided Therapies (II)



Context: Image-Guided Therapies Workflow



To simulate or not. That’s the question

To simulate or not. That's the question

Simulations as accurate predictors

- Difficulty to align initial conditions
- Difficulty to accurately model complex systems
- Computationally expensive

Simulations as providers of non-linear behavior

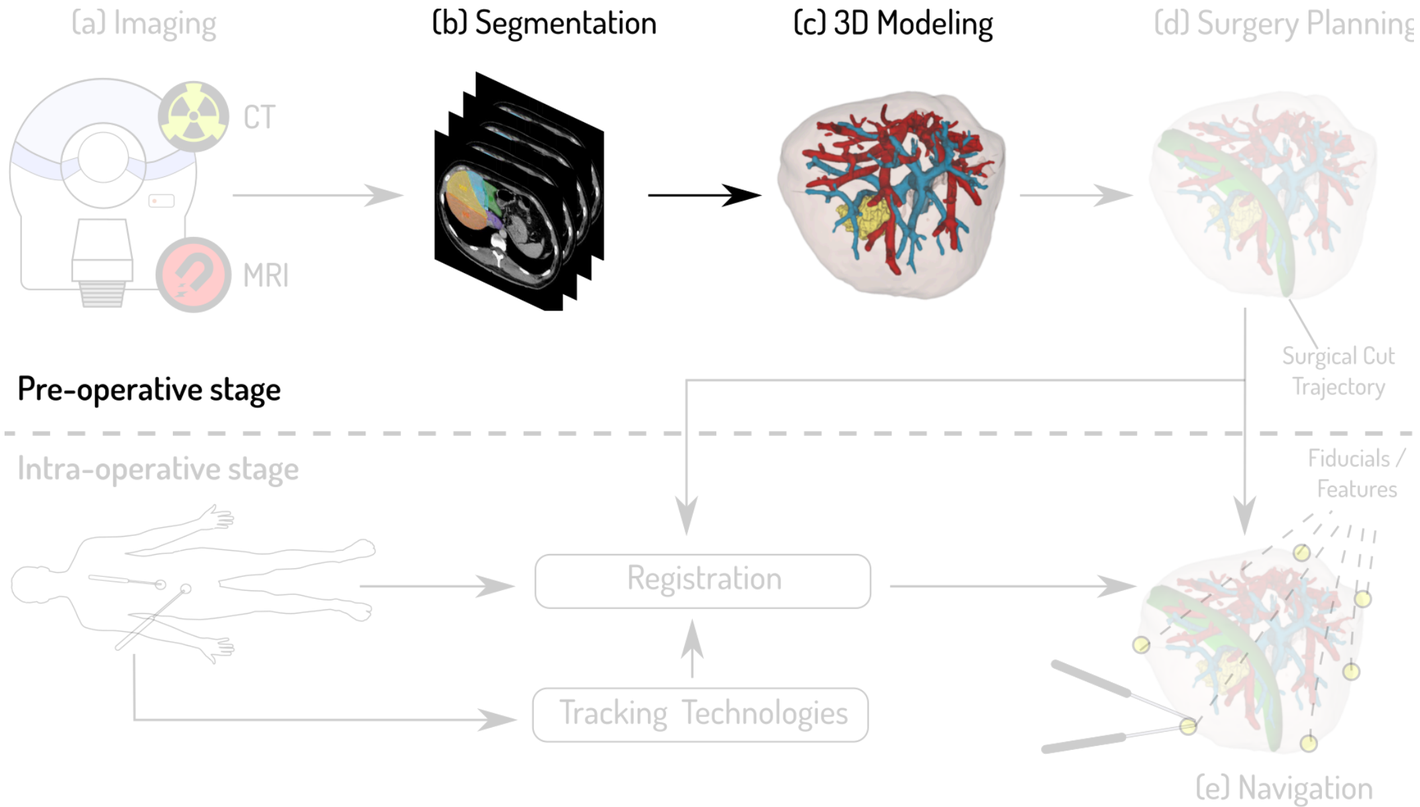
- Non-linear better than linear
- Generating plausibility over reality
- Fast computation over precision

Example of SOFA providing non-linear behavior

(Stephane Cotin et al.)

Mesh Models Generation

From Segmentation to 3D Models



Segmentations

- Voxel-based objects

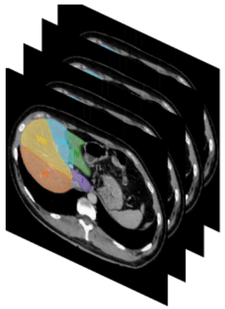
3D Surface models

- Triangle-based meshes
- Represent the shell of objects
- Useful for visualization

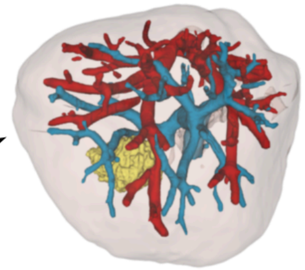
3D Tetrahedral meshes

- Tetrahedra-based (i.e., pyramid)
- Useful for simulation

From Segmentation to 3D Models



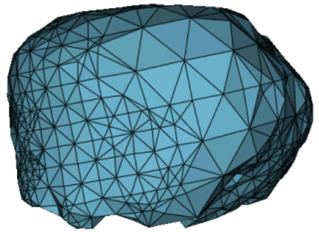
Surface Models



- Segmentations Module
- Model Maker Module
- Surface Toolbox Module

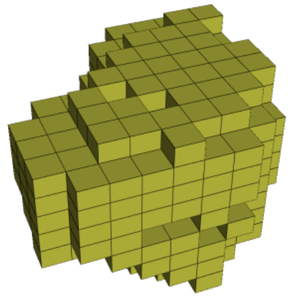
*SOFA can automatically generate these Sparse Grids from Surface Models

Volumetric Mesh Models



- Segment Mesher Extension
  - Tetgen
  - Cleaver
- GMesh external

Sparse Grids



From Segmentation to 3D Models

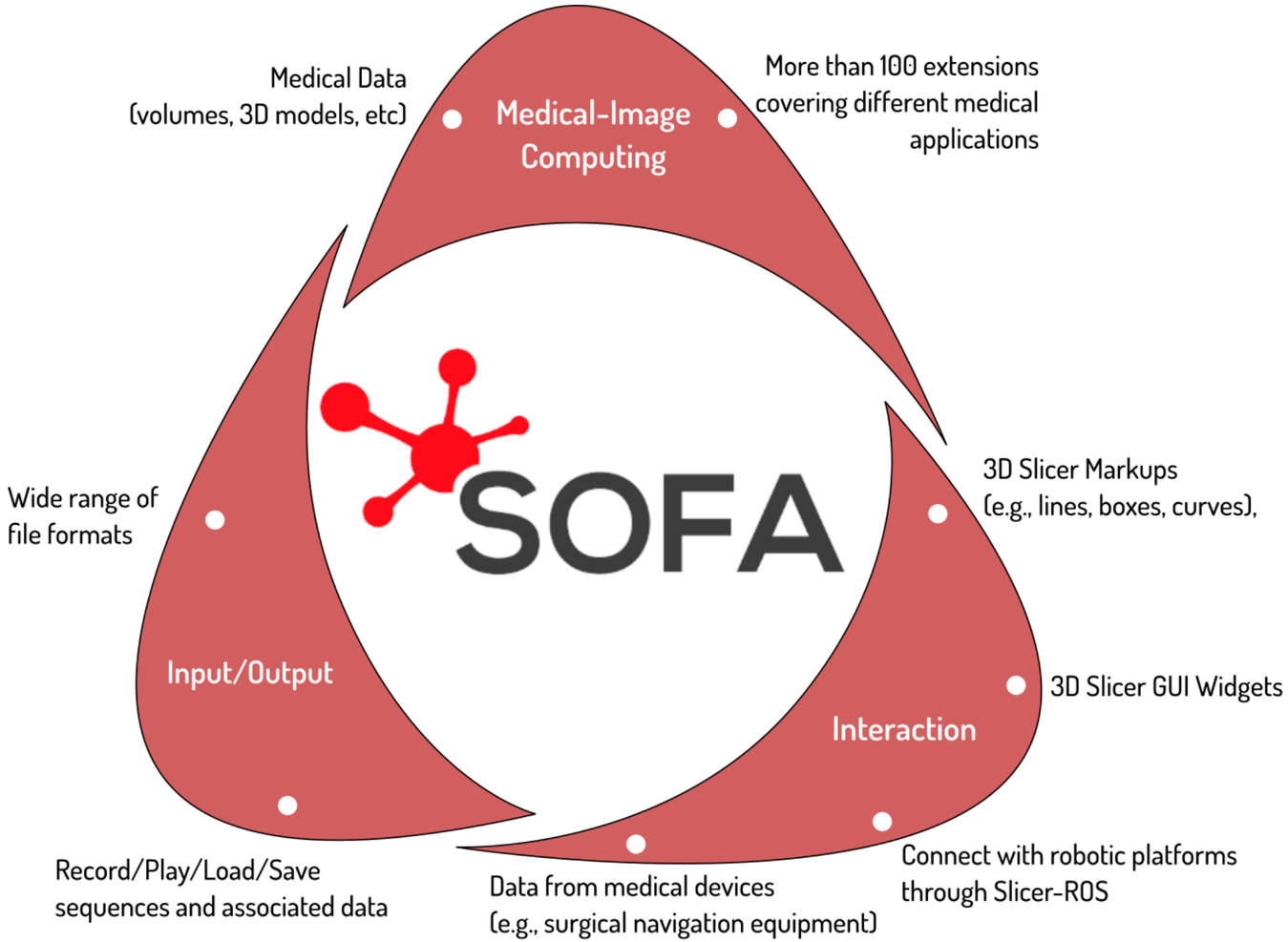
- Simulation applications could use a combination of models
  - Simple tetrahedral meshes (or sparse grids) for computations
  - Translation of results over to more detailed surface meshes
- There are no definitive methods
  - Variability of underlying data (e.g., quality of segmentation)
  - Purpose of the output mesh (e.g., visualization, computation, simulation)
- Post processing is often applied
  - Smoothing
  - Decimation

Tips

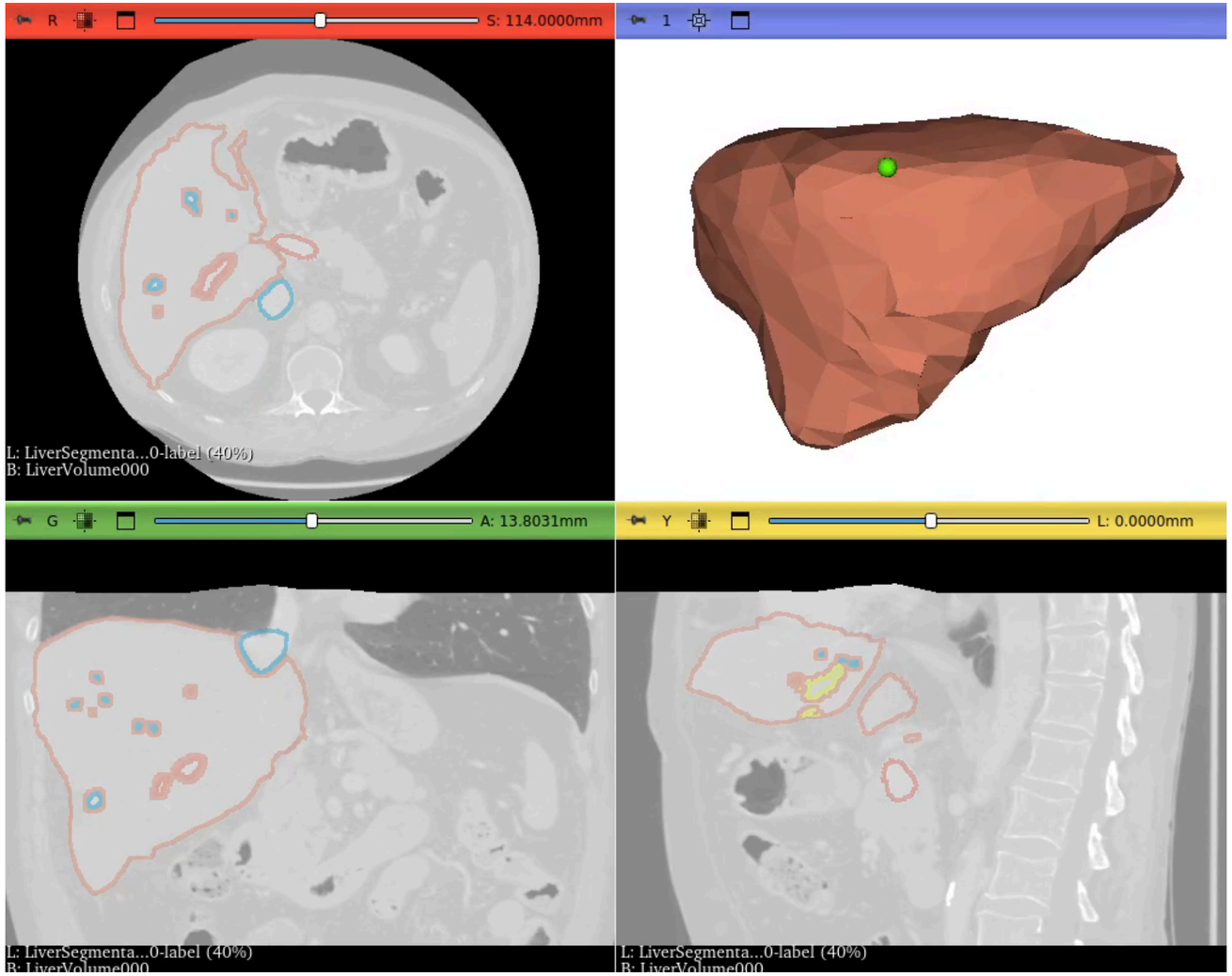
- For segmentations, watch for noise particles, non-connected components and sharp edges.
- For mesh models, watch for topological correctness, volumes/areas, regularity of triangles.

SlicerSOFA

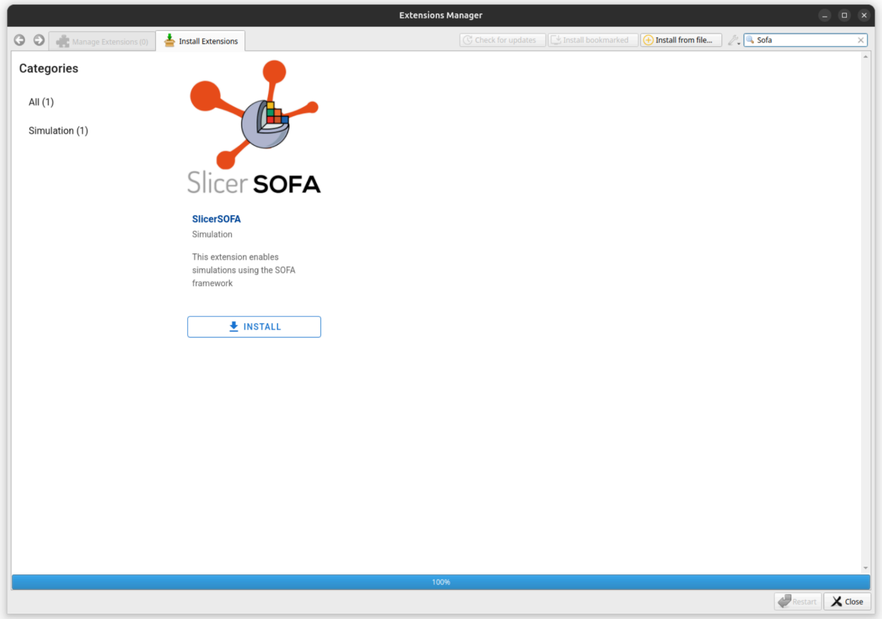
SlicerSOFA Features



SlicerSOFA


R
S: 114.0000mm
L: LiverSegmenta...0-label (40%)
B: LiverVolume000
1
G
A: 13.8031mm
L: LiverSegmenta...0-label (40%)
Y
L: 0.0000mm
L: LiverSegmenta...0-label (40%)

What is SlicerSOFA?

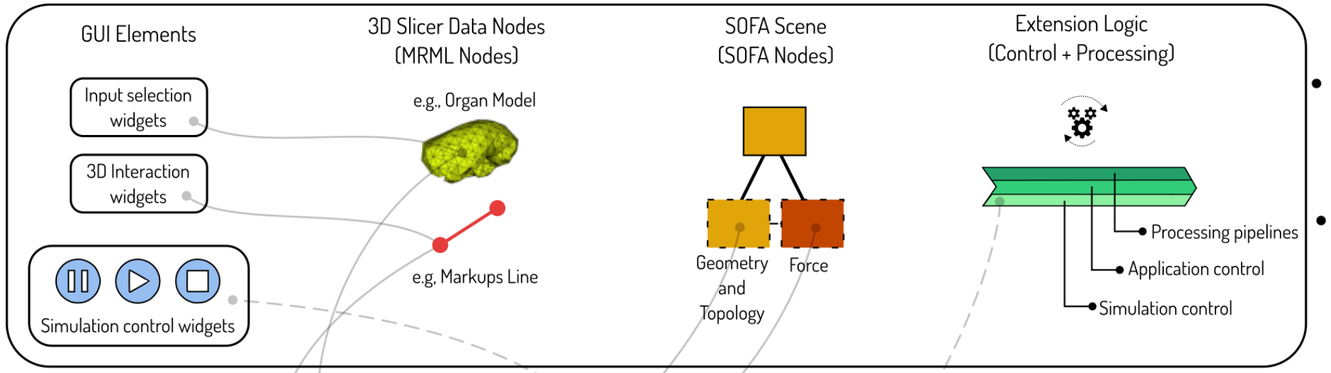


SlicerSOFA is a 3D Slicer Extension

- Open-source: BSD and LGPL licenses
- Available through the **Slicer Extension Manager** as downloadable binary
- Source code available at https://github.com/Slicer/SlicerSOFA
- Issue tracker at https://github.com/Slicer/SlicerSOFA/issues

SlicerSOFA Architecture

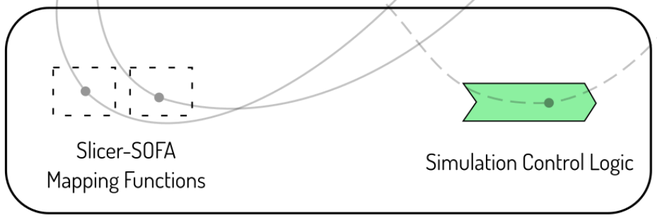
Example of custom application developed using Slicer-SOFA



- Application leveraging simulation, visualization and medical-image computing
- Can communicate with other 3D Slicer Extensions

Slicer-SOFA Extension

Extension Module



- Abstracts Slicer-SOFA communication mechanisms
- Abstracts simulation control
- Enables simulation recording

Extension Infrastructure

- Enables communication between SOFA and 3D Slicer through Python

SlicerSOFA Development Status (06/2025)

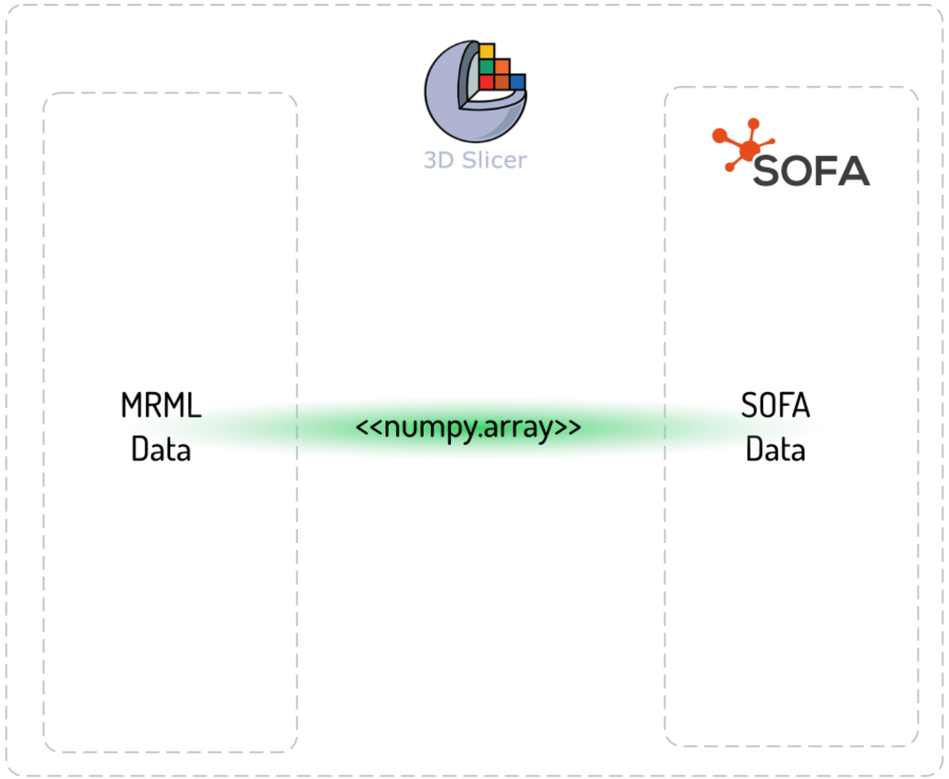
	SOFA Version	Windows	Linux	MacOS
Slicer v5.8.1 (stable)	v24.06	Dev+Pack	Dev+Pack	Dev only
Slicer v5.9 (preview)	v24.06	Dev+Pack	Dev+Pack	Dev only

https://github.com/Slicer/SlicerSOFA

Practical SlicerSOFA

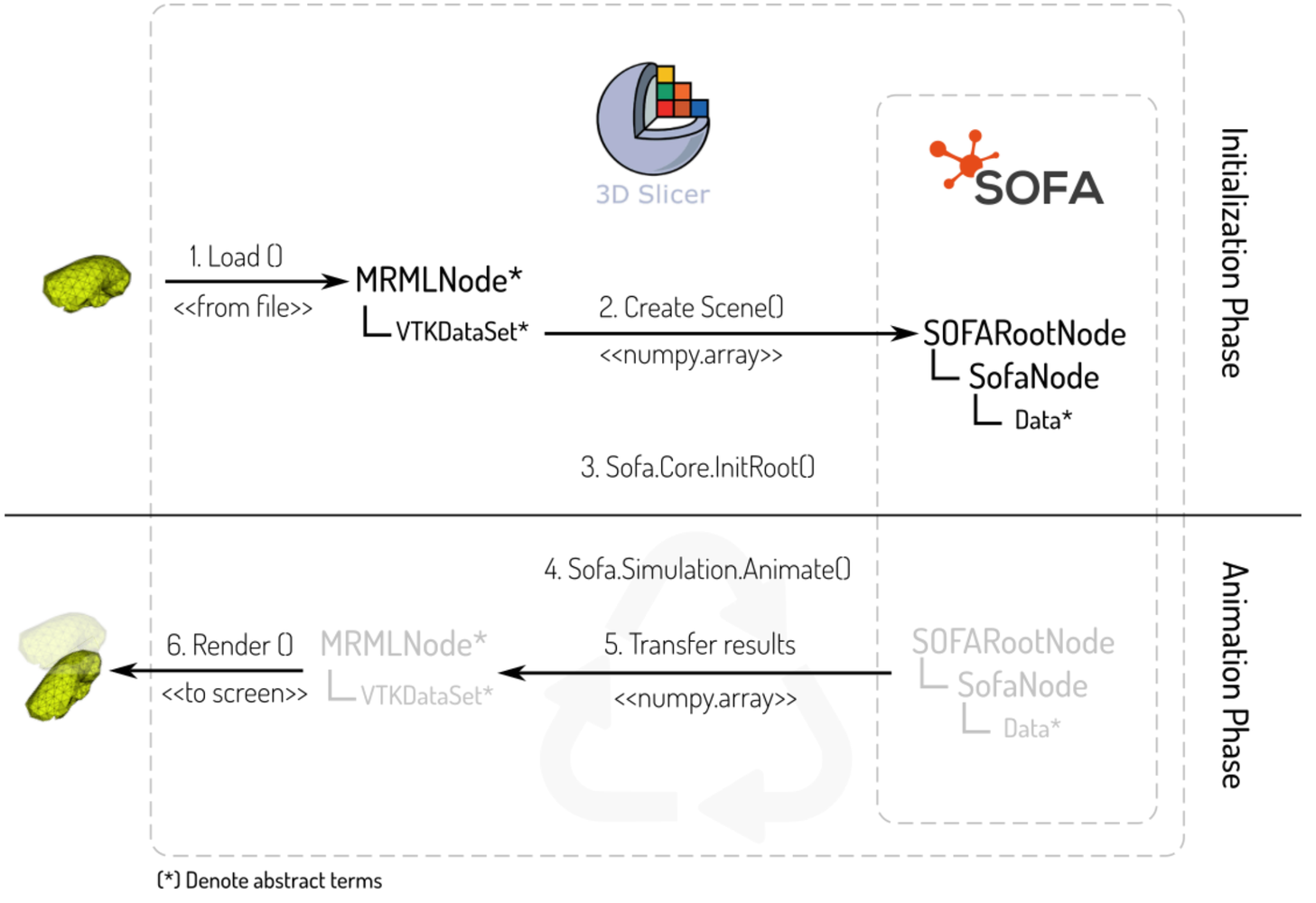
Integrating Slicer and SOFA

Integrating Slicer and SOFA (I)



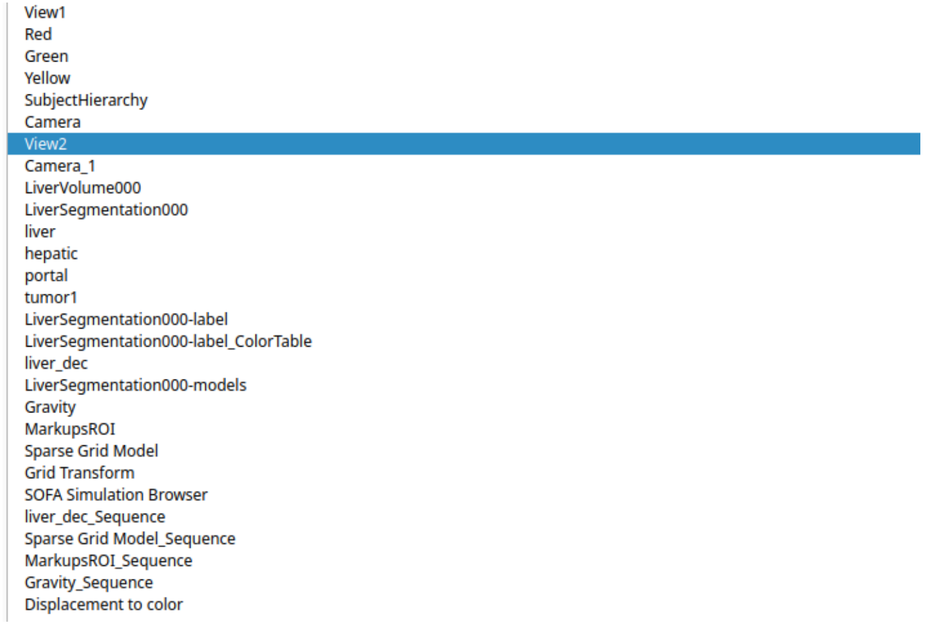
- Everything runs in the Slicer process
- Slicer data and SOFA data are communicated using `numpy arrays` as common language:
  - Efficient
  - Powerful
  - Guarantees contiguous memory

Integrating Slicer and SOFA (II)



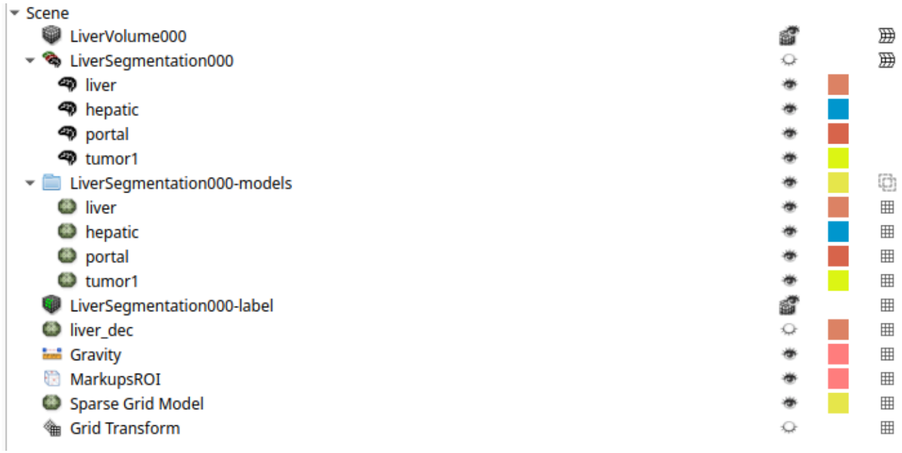
Understanding Data Components in 3D * Understanding Data Components in 3D Slicer: MRML (I)

- **Medical Reality Modeling Language (MRML)**
  - Data model to represent all data sets used in medical software applications
- **MRML Scene**
  - Contains a list of **MRML Nodes**
  - Nodes can reference (link to) each other



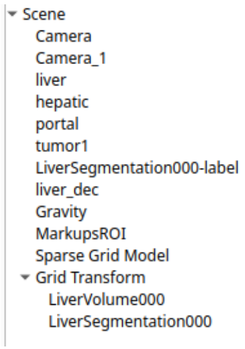
Understanding Data Components in 3D Slicer: MRML (II)

- **Medical Reality Modeling Language (MRML)**
  - Data model to represent all data sets used in medical software applications
- **MRML Scene**
  - Contains a list of **MRML Nodes**
  - Nodes can reference (link to) each other



Understanding Data Components in 3D Slicer: MRML (III)

- **Medical Reality Modeling Language (MRML)**
  - Data model to represent all data sets used in medical software applications
- **MRML Scene**
  - Contains a list of **MRML Nodes**
  - Nodes can reference (link to) each other



Understanding Data Components in 3D Slicer: MRML (IV)

- **MRML Nodes**
  - Each node has:
    - Unique ID and name
    - Custom attributes
    - Data-type-specific properties
  - **Node Types** include:
    - Image Volumes
    - **Meshes**
    - Point Sets
    - Transformations
    - and more…

Simple node creation in 3D Slicer

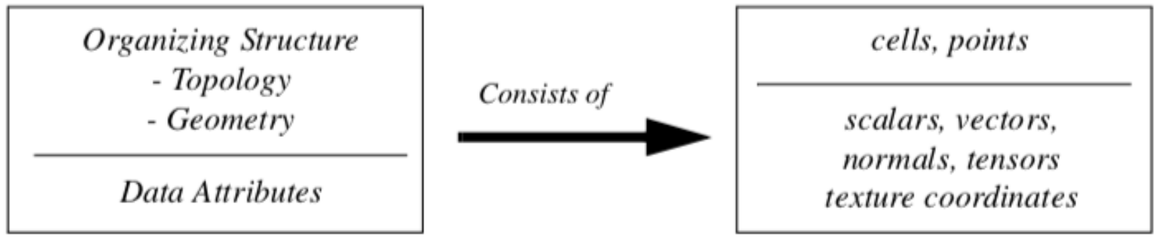
```python
# Scrip: 001_simple_node_creation.py (ctrl+g in slicer)

# Create a model node (Method I)
modelNode = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLModelNode')

# Create a model node (Method II)
modelNode = slicer.vtkMRMLModelNode()
slicer.mrmlScene.AddNode(modelNode)

# Create display node (regardless of the method)
print(modelNode)
```

https://apidocs.slicer.org/v5.8/classvtkMRMLNode.html

Understanding Data Components in 3D Slicer: VTK datasets (I)

VTK Datasets

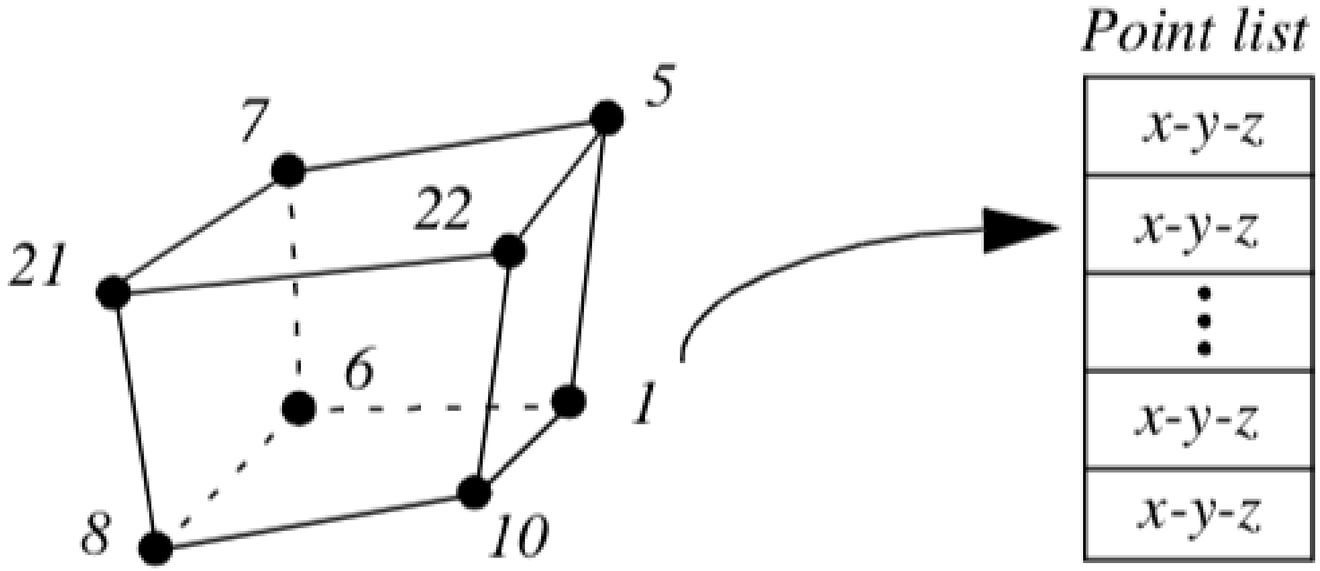
- Data objects with an **organizing structure** and **associated data attributes**
- The structure has two parts: **geometry** and **topology**
- [**Dataset Geometry**] Specification of position in 3D space (**points**)
- [**Dataset Topoology**] The set of properties invariant under certain geometric transformations (**cells**)



https://examples.vtk.org/site/VTKBook/05Chapter5/#characterizing-visualization-data

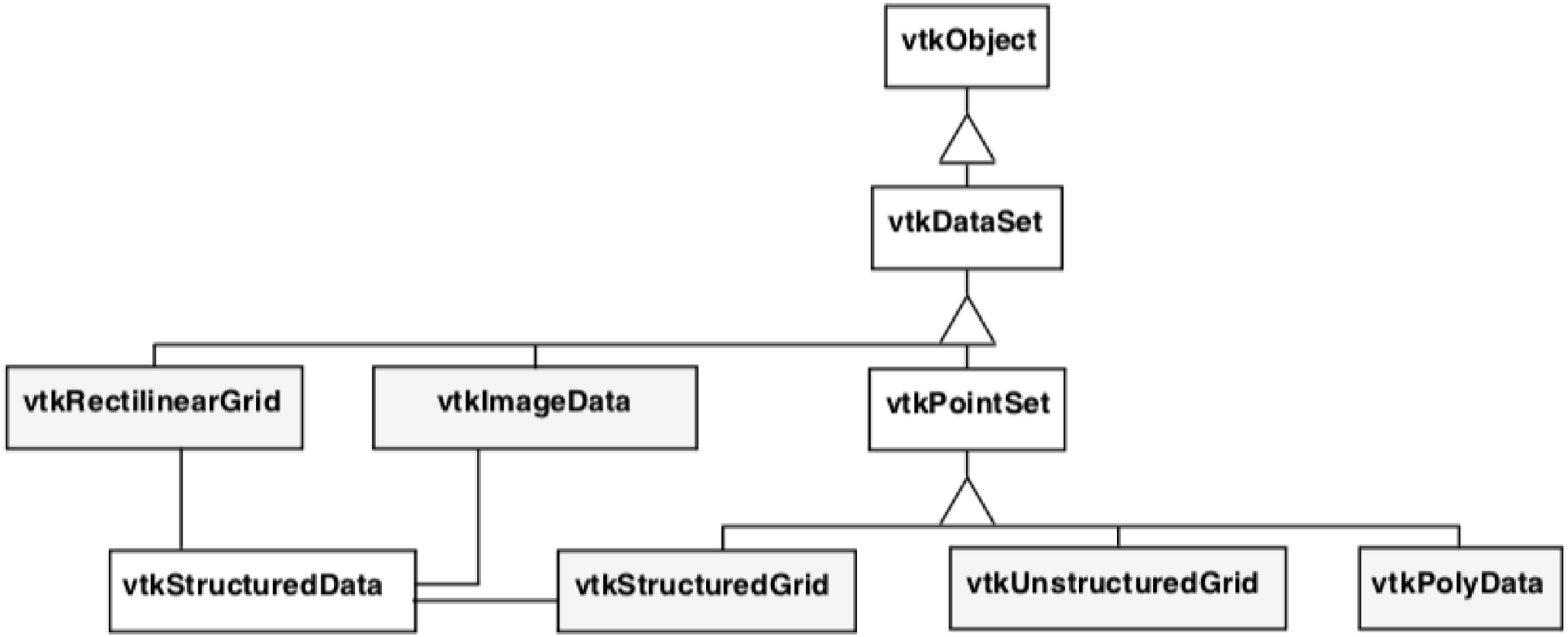
Understanding Data Components in 3D Slicer: VTK datasets (II)

Definition:
Type: hexahedron
Connectivity: (8,10,1,6,21,22,5,7)



https://examples.vtk.org/site/VTKBook/05Chapter5/#54-cell-types

Understanding Data Components in 3D Slicer: VTK datasets (III)



https://examples.vtk.org/site/VTKBook/05Chapter5/#dataset-representation

Understanding Data Components in 3D Slicer: VTK datasets (IV)

`vtkPolyData`

- Used for surface models (e.g., for visualization, not for simulation)
- Requires explicit representation of both geometry and topology.
- **Geometry Representation**:
- Points are stored using the `vtkPoints` class.
- **Cell Type Management**:
  - `vtkPolyData` maintains four separate cell lists:
    - **Vertices**: `vtkVertex`, `vtkPolyVertex`.
    - **Lines**: `vtkLine`, `vtkPolyLine`.
    - **Polygons**: `vtkTriangle`, `vtkQuad`, `vtkPolygon`.
    - **Triangle Strips**: `vtkTriangleStrip`.

https://examples.vtk.org/site/VTKBook/05Chapter5/#dataset-representation

vtkPolyData

```python
# Script: 002_vtkpolydata_1.py (ctrl+g in slicer)
#   requires the SlicerSOFA extension

# Load the dataset
import SampleData
liverScene = SampleData.downloadSample('LiverSimulationScene')

# Get model from MRML Scene
modelNode = slicer.util.getNode('liver_dec')

# Get polydata from model
polyData = modelNode.GetPolyData()

# Iterate over points in the mesh (geometry)
for i in range(polyData.GetNumberOfPoints()):
    print(polyData.GetPoint(i))

# Iterate over cells in the mesh (topology)
for i in range(polyData.GetNumberOfCells()):
    print(polyData.GetCellType(i),
          polyData.GetCell(i).GetPointId(0),
          polyData.GetCell(i).GetPointId(1),
          polyData.GetCell(i).GetPointId(2))
```

Output:

```
(129.05188821670617, 25.76884416954813, 39.73223040375034)
(118.80156312926272, 35.7303511720099, 36.28580029038706)
(132.42859446014708, 38.59746416817071, 41.155659542019805)
(129.12104482620379, 14.465276201463498, 48.71041794474367)
...
5 5 1 0
5 2 0 1
5 1 4 2
5 2 4 10
5 12 1 5
...
```

`vtkUnstructuredGrid`

- The most general VTK dataset type for representing complex topology and geometry
- Can represent tetrahedral meshes (e.g., used for simulations)
- **Explicit Representation of Geometry and Topology**
  - **Points** are stored using **vtkPoints**
  - **Cells** are stored using **vtkCellArray**

https://examples.vtk.org/site/VTKBook/05Chapter5/#dataset-representation

vtkUnstructuredGrid

```python
# Script: 003_vtkunstructuredgrid_1.py (ctrl+g in slicer)
#   requires the SlicerSOFA extension

# Load the dataset
import SampleData
liverScene = SampleData.downloadSample('RightLungLowTetra')

# Get model from MRML Scene
modelNode = slicer.util.getNode('RightLung')

# Get the unstructured grid from model
unstructuredGrid = modelNode.GetMesh()

# Iterate over points in the mesh (geometry)
for i in range(unstructuredGrid.GetNumberOfPoints()):
    print(unstructuredGrid.GetPoint(i))

# Iterate over cells in the mesh (topology)
for i in range(unstructuredGrid.GetNumberOfCells()):
    print(unstructuredGrid.GetCellType(i),
          unstructuredGrid.GetCell(i).GetPointId(0),
          unstructuredGrid.GetCell(i).GetPointId(1),
          unstructuredGrid.GetCell(i).GetPointId(2),
          unstructuredGrid.GetCell(i).GetPointId(3))
```

Output:

```
(129.05188821670617, 25.76884416954813, 39.73223040375034)
(118.80156312926272, 35.7303511720099, 36.28580029038706)
(132.42859446014708, 38.59746416817071, 41.155659542019805)
(129.12104482620379, 14.465276201463498, 48.71041794474367)
...
5 5 1 0
5 2 0 1
5 1 4 2
5 2 4 10
5 12 1 5
...
```

Understanding Data Components in SOFA

Understanding Data Components in SOFA (I)

- All SOFA scenes start with the creation of a root node
- Plugins must be explicitly loaded (`RequiredPlugin` type nodes)
- Knowing which plugins are needed is a matter of knowing SOFA, and sometimes, trial and fail.

```python
# Script: 004_sofa_nodex.py (ctrl+g in slicer)
#   requires the SlicerSOFA extension
import Sofa.Core
import numpy as np

#Create the root node
rootNode = Sofa.Core.Node('root')

plugins=["Sofa.Component.IO.Mesh",
         "Sofa.Component.StateContainer",
         "Sofa.Component.Mapping.NonLinear",
         "Sofa.Component.Topology.Container.Constant",
         "Sofa.Component.Topology.Mapping",
         #Other plugins
         ]

for plugin_name in plugins:
    rootNode.addObject('RequiredPlugin', name=plugin_name)


inputNode = rootNode.addChild('InputSurfaceNode')
container = inputNode.addObject('TriangleSetTopologyContainer',
                                name='Container',
                                position=np.zeros(10*3).reshape(-1,3),
                                triangles=np.zeros(100))
```

Understanding Data Components in SOFA (II)

- Nodes can be addressed by
  1. Direct access through variables (e.g., `subnode`).
  2. Array notation (e.g., `node['subnode_name']`)
  3. Array notation with `.` separator (e.g., =node['subnode$_{name.subsubnodename}$']

```
# Script: 005_sofa_nodes.py (ctrl+g in slicer)
#   requires the SlicerSOFA extension
import Sofa.Core
import numpy as np

#Create the root node
rootNode = Sofa.Core.Node('root')

# Required plugins can be added here

inputNode = rootNode.addChild('InputSurfaceNode')
container = inputNode.addObject('TriangleSetTopologyContainer',
                                name='Container', position=np.zeros(10*3).reshape(-1,3),
                                triangles=np.zeros(100))

# Access to container
print(container)
print(rootNode['InputSurfaceNode']['Container'])
print(rootNode['InputSurfaceNode.Container'])
print(rootNode['Container']) #This won't work!
print(inputNode['Container'])#This will work!
```

Understanding Data Components in SOFA (III)

- Node **data** read acces
  1. Direct access through variables (e.g., `subnode.attribute.array()`).
  2. Array notation (e.g., `node['subnode_name']['attribute].array()`).
  3. Array notation with `.` separator (e.g., `node['subnode_name.subsubnode_name.attribute'].array()`).

```python
# Script: 006_sofa_nodes.py (ctrl+g in slicer)
#   requires the SlicerSOFA extension
import Sofa.Core
import numpy as np

#Create the root node
rootNode = Sofa.Core.Node('root')

# Required plugins can be added here

inputNode = rootNode.addChild('InputSurfaceNode')
container = inputNode.addObject('TriangleSetTopologyContainer',
                                name='Container', position=np.zeros(10*3).reshape(-1,3),
                                triangles=np.zeros(100))

# Access to container geometry
print(container.position.array())
print(rootNode['InputSurfaceNode']['Container']['position'].array())
print(rootNode['InputSurfaceNode.Container.position'].array())
```

Understanding Data Components in SOFA (IV)

- Node **data** write
  1. Direct access through variables and attributes (e.g., `subnode.attribute`). For scalar values
  2. Through `.writeable()` arrays. For vector data.

```python
# Script: 007_sofa_nodes.py (ctrl+g in slicer)
#   requires the SlicerSOFA extension
import Sofa.Core
import numpy as np

#Create the root node
rootNode = Sofa.Core.Node('root')

# Required plugins can be added here

inputNode = rootNode.addChild('InputSurfaceNode')
container = inputNode.addObject('TriangleSetTopologyContainer',
                                name='Container', position=np.zeros(10*3).reshape(-1,3),
                                triangles=np.zeros(100))

# Access to container geometry
rootNode.dt = 0.1
with container.position.writeable() as geometry:
    geometry[:] = np.random.rand(10*3).reshape(-1,3) #Note: Copy!!
    geometry[0][0] = 10
```

https://sofapython3.readthedocs.io/en/latest/content/modules/Sofa/generated/Sofa.Core/classes/Sofa.Core.Dat

Using **numpy** arrays to Communicate Slicer and SOFA

Using `numpy` arrays to Communicate Slicer and SOFA (I)

- **NumPy** is a powerful library for numerical computations in Python.
- **ndarray** is the core data structure in NumPy, representing multidimensional homogeneous arrays.
- Arrays can be created from lists or tuples, or functions like `np.zeros`, `np.ones`, `np.arange`, `np.linspace`.

```python
import numpy as np

# Create a 1D array
a = np.array([1, 2, 3, 4, 5])

# Create a 2D array
b = np.array([[1, 2, 3], [4, 5, 6]])

print("1D array:", a)
print("2D array:\n", b)
```

Output:

```
1D array: [1 2 3 4 5]
2D array:
 [[1 2 3]
 [4 5 6]]
```

https://numpy.org/doc/stable/reference/generated/numpy.array.html

Using `numpy` arrays to Communicate Slicer and SOFA (II)

- NumPy arrays are stored in contiguous blocks of memory, enabling efficient computation.
- Each array has a data type (`dtype`) that defines the type of elements in the array.
- Common dtypes include `np.int32`, `np.float64`, etc.

```python
import numpy as np

a = np.array([1, 2, 3], dtype=np.int32)
print("Array:", a)
print("dtype:", a.dtype)
print("Item size:", a.itemsize)
print("Total size (nbytes):", a.nbytes)
```

Output:

```
Array: [1 2 3]
dtype: int32
Item size: 4
Total size (nbytes): 12
```

Using `numpy` arrays to Communicate Slicer and SOFA (III)

- Slices are **views**, not copies, meaning they reference the same memory. One can select subsets of data efficiently without copying.

```python
import numpy as np

a = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Select first row
row = a[0, :]
print("First row:", row)

# Select first column
col = a[:, 0]
print("First column:", col)

# Select a subarray
subarray = a[0:2, 1:3]
print("Subarray:\n", subarray)
```

Output:

```
First row: [1 2 3]
First column: [1 4 7]
Subarray:
 [[2 3]
 [5 6]]
```

https://numpy.org/doc/stable/user/basics.indexing.html

Using `numpy` arrays to Communicate Slicer and SOFA (IV)

- Slices of arrays create **views**, not copies.
- Modifying a view will affect the original array.
- Use the `copy()` method to create a copy when needed.

```python
import numpy as np

a = np.array([1, 2, 3, 4, 5])

# Slicing creates a view
b = a[1:4]
print("Original slice:", b)

b[0] = 99
print("Modified slice:", b)
print("Original array after modifying slice:", a)

# Using copy to create a separate array
c = a[1:4].copy()
c[0] = 100
print("Copy modified:", c)
print("Original array after modifying copy:", a)
```

Output:

```
Original slice: [2 3 4]
Modified slice: [99  3  4]
Original array after modifying slice: [ 1 99  3  4  5]
Copy modified: [100   3   4]
Original array after modifying copy: [ 1 99  3  4  5]
```

Using `numpy` arrays to Communicate Slicer and SOFA (V)

- Assigning to a slice modifies the original array.
- Using `a[:] = something` replaces the contents in-place without changing the array object.
- In-place modifications can be efficient and sometimes necessary, as they preserve the size of the original array.

```python
import numpy as np

a = np.array([1, 2, 3, 4, 5])

# Assigning a new array to 'a' creates a new object
a = np.array([10, 20, 30, 40, 50])
print("After assignment, 'a' is:", a)

# Using a slice to modify 'a' in-place
a[:] = [100, 200, 300, 400, 500]
print("After in-place modification, 'a' is:", a)

# Verifying that the array object is the same
b = a
a[:] = [1, 2, 3, 4, 5]
print("After modifying 'a', 'b' is also changed:", b)
```

Output:

```
After assignment, 'a' is: [10 20 30 40 50]
After in-place modification, 'a' is: [100 200 300 400 500]
After modifying 'a', 'b' is also changed: [1 2 3 4 5]
```

Making a Simulation Tick

Making a Simulation Tick (I)

- Advancing the simulation requires an explicit call to `Sofa.Simulation.animate`
- Non-interactive simulations can utilize a simple loop

```
count = 0
while count < 350: #350 iterations
    count += 1
    Sofa.Simulation.animate(root, root.dt.value)
    slicer.app.processEvents() #Let Slicer process its own events
```

https://github.com/pieper/SlicerSOFA/blob/main/Experiments/scene_with_attachments.py

Making a Simulation Tick (II)

- Advancing the simulation requires an explicit call to `Sofa.Simulation.animate`
- interactive simulations can utilize a simple loop

```python
from qt import QTimer

# Initialization code goes here
# ...

iteration = 0
iterations = 30
simulating = True

def updateSimulation(): # Callback function
    global iteration, iterations, simulating

    # Update from Slicer to SOFA goes here
    # ...

    Sofa.Simulation.animate(rootSofaNode, rootSofaNode.dt.value)

    # Update from SOFA to Slicer goes here
    # ...

    # Iteration management
    iteration += 1
    simulating = iteration < iterations
    if simulating:
        QTimer.singleShot(10, updateSimulation)
```

https://doc.qt.io/qt-6/qtimer.html#singleShot

Complete Example

Complete Example

```python
# Script: 008_example_1.py
#  - requires SlicerSOFA
#  - requires adjusting path for mesh files

import numpy
import Sofa
import Sofa.Core
import Sofa.Simulation

from slicer.util import arrayFromModelPoints
from slicer.util import arrayFromModelPolyIds # For Polydata cells
from SlicerSofaUtils.Mappings import arrayFromModelGridCells # For tetrahedral unstructured grid cells
```

- Importing essential libraries
- Importing utility libraries from `slicer.util` from `SlicerSofaUtils.Mappings`

Complete Example

```
1  ##########################################
2  ###### Simulation Hyperparameters
3  ##########################################
4
5  # Input data parameters
6  liver_mesh_file = "/tmp/originalMesh.vtk"
7  sphere_surface_file = "/tmp/biggerCavity.obj"
8  originalMeshNode = None
9  sphereNode = None
10 liver_mass = 30.0
11 liver_youngs_modulus = 1.0 * 1000.0 * 0.001
12 liver_poisson_ratio = 0.45
13
14 # Simulatlon hyperparameters
15 root = None
16 dt = 0.01
17 collision_detection_method = "LocalMinDistance"
18 alarm_distance = 10.0
19 contact_distance = 0.8
20
21 # Simulation control parameters
22 iteration = 0
23 iterations = 30
24 simulating = True
```

Complete Example

```
1 ##############################################
2 ###### Load Simulation Data
3 ##############################################
4 def loadSimulationData():
5     global originalMeshNode
6     originalMeshNode = slicer.util.loadModel(liver_mesh_file)
7     sphereNode = slicer.util.loadModel(sphere_surface_file)
8     sphereNode.GetDisplayNode().SetRepresentation(slicer.vtkMRMLDisplayNode.WireframeRepresentation)
```

Complete Example

```python
##########################################
###### Create Sofa Scene
##########################################
def createSofaScene():
    global root

    # Create a root node
    root = Sofa.Core.Node("root")

    plugin_list = [
        "MultiThreading",
        "Sofa.Component.AnimationLoop",
        "Sofa.Component.Collision.Detection.Algorithm",
        "Sofa.Component.Collision.Detection.Intersection",
        "Sofa.Component.Collision.Geometry",
        "Sofa.Component.Collision.Response.Contact",
        "Sofa.Component.Constraint.Lagrangian.Correction",
        "Sofa.Component.Constraint.Lagrangian.Solver",
        "Sofa.Component.IO.Mesh",
        "Sofa.Component.LinearSolver.Direct",
        "Sofa.Component.Mass",
        "Sofa.Component.MechanicalLoad",
        "Sofa.Component.ODESolver.Backward",
        "Sofa.Component.SolidMechanics.FEM.Elastic",
        "Sofa.Component.StateContainer",
        "Sofa.Component.Topology.Container.Dynamic",
        "Sofa.Component.Topology.Mapping",
        "Sofa.Component.Visual",
        "Sofa.Component.Constraint.Projective",
    ]
    for plugin in plugin_list:
        root.addObject("RequiredPlugin", name=plugin)

```

Complete Example

```
#def createSofaScene() continues
    # The simulation scene
    with root.gravity.writeable() as gravity:
        gravity[:] = np.array([-9.81 * 10.0, 0.0, 0.0])
    root.dt = dt

    root.addObject("FreeMotionAnimationLoop")
    root.addObject("VisualStyle",
                   displayFlags=["showForceFields", "showBehaviorModels", "showCollisionModels", "showWireframe"])

    root.addObject("CollisionPipeline")
    root.addObject("ParallelBruteForceBroadPhase")
    root.addObject("ParallelBVHNarrowPhase")
    root.addObject(collision_detection_method, alarmDistance=alarm_distance, contactDistance=contact_distance)

    root.addObject("CollisionResponse", response="FrictionContactConstraint", responseParams=0.001)
    root.addObject("GenericConstraintSolver")

    scene_node = root.addChild("scene")
```

Complete Example

```
#def createSofaScene() continues
    #### Liver ####
    liver_node = scene_node.addChild("liver")

    liver_node.addObject('TetrahedronSetTopologyContainer', name="Container",
                         position=arrayFromModelPoints(originalMeshNode),
                         tetrahedra=arrayFromModelGridCells(originalMeshNode))

    liver_node.addObject("TetrahedronSetTopologyModifier")
    liver_node.addObject("EulerImplicitSolver")
    liver_node.addObject("SparseLDLSolver", template="CompressedRowSparseMatrixMat3x3d")
    liver_node.addObject("MechanicalObject")
    liver_node.addObject("TetrahedralCorotationalFEMForceField",
                         youngModulus=liver_youngs_modulus,
                         poissonRatio=liver_poisson_ratio)
    liver_node.addObject("UniformMass", totalMass=liver_mass)
    liver_node.addObject("LinearSolverConstraintCorrection")

    liver_collision_node = liver_node.addChild("collision")
    liver_collision_node.addObject("TriangleSetTopologyContainer")
    liver_collision_node.addObject("TriangleSetTopologyModifier")
    liver_collision_node.addObject("Tetra2TriangleTopologicalMapping")
    liver_collision_node.addObject("PointCollisionModel")
    liver_collision_node.addObject("LineCollisionModel")
    liver_collision_node.addObject("TriangleCollisionModel")
```

Complete Example

```
#def createSofaScene() continues
    #### Sphere ####
    sphere_node = scene_node.addChild("sphere")
    sphere_node.addObject("MeshOBJLoader", filename=sphere_surface_file, scale=1.0)
    sphere_node.addObject("TriangleSetTopologyContainer", src=sphere_node.MeshOBJLoader.getLinkPath())
    sphere_node.addObject("TriangleSetTopologyModifier")
    sphere_node.addObject("MechanicalObject")
    # NOTE: The important thing is to set bothSide=True for the collision models,
    # so that both sides of the triangle are considered for collision.
    sphere_node.addObject("TriangleCollisionModel", bothSide=True)
    sphere_node.addObject("PointCollisionModel")
    sphere_node.addObject("LineCollisionModel")
    sphere_node.addObject("FixedProjectiveConstraint")
```

Complete Example

```
#def createSofaScene() continues

    # Initialize the simulation
    Sofa.Simulation.init(root)

    with sphere_node.MechanicalObject.position.writeable() as sphereArray:
        sphereArray *= [-1,-1,1] #Note the LPS-to-RAS transform here!!
```

- Note that **part of the data** for the simulation is loaded by **SOFA**
- SOFA-loaded data and Slicer-loaded data are not in the same coordinate frame
- Requires a LPS-to-RAS conversion

Complete Example

```
##########################################
###### Update Simulation
##########################################
def updateSimulation():
    global iteration, iterations, simulating, root, originalMeshNode

    for step in range(10):
        Sofa.Simulation.animate(root, root.dt.value)

    # update model from mechanical state
    meshPointsArray = root['scene.liver'].getMechanicalState().position.array()
    modelPointsArray = slicer.util.arrayFromModelPoints(originalMeshNode)
    modelPointsArray[:] = meshPointsArray #Note the slice operator (copy!)
    slicer.util.arrayFromModelPointsModified(originalMeshNode)

    # iteration management
    iteration += 1
    simulating = iteration < iterations
    if iteration % 10 == 0:
        print(f"Iteration {iteration}")
    if simulating:
        qt.QTimer.singleShot(10, updateSimulation)
    else:
        print("Simlation stopped")

```

Complete Example

```
1 ##########################################
2 ###### Execution flowRequires a LPS-to-RAS conversion
3 ##########################################
4 loadSimulationData()
5 createSofaScene()
6 updateSimulation()
```