

Lib/CollaboratiVesselLogic.py

```
1 import os
2 import logging
3
4 import numpy as np
5
6 import slicer
7 from slicer.ScriptedLoadableModule import ScriptedLoadableModuleLogic
8 from slicer.parameterNodeWrapper import parameterNodeWrapper
9 from slicer import vtkMRMLScalarVolumeNode
10
11
12 # In order to separate logic from widget, we must modify the path:
13 # Git clone clDice in Lib/clDice jocpae/clDice, MIT licensed
14 # Self-bootstrap sys.path so the import works no matter what
15 import sys as _sys
16 _LIB_DIR = os.path.dirname(os.path.abspath(__file__))
17 if _LIB_DIR not in _sys.path:
18     _sys.path.insert(0, _LIB_DIR)
19
20
21 # Try/Catch for user friendly errors
22 try:
23     import skimage.morphology
24     HAS_SKIMAGE = True
25 except ImportError:
26     HAS_SKIMAGE = False
27
28 try:
29     import itk
30     from itk import TubeTK
31     HAS_TUBETK = True
32 except ImportError:
33     HAS_TUBETK = False
```

```
34
35 try:
36     import scipy
37     HAS_SCIPY = True
38 except ImportError:
39     HAS_SCIPY = False
40
41 try:
42     import SimpleITK
43     HAS_SITK = True
44 except ImportError:
45     HAS_SITK = False
46
47 try:
48     from PIL import Image
49     HAS_PIL = True
50 except ImportError:
51     HAS_PIL = False
52
53
54 # Helpers
55 from helpers import (
56     rasToIjk,
57     scalarArray,
58     pushArray,
59     createSegmentationFromMask,
60     exportBinarySlices,
61 )
62 from evaluation import (
63     computeDice,
64     computeClDice,
65     evaluateAgainstChunks,
66 )
67 from pipeline import (
68     stageRemoveBackground,
69     stageFrangi,
```

```
70     stageExpandCenterlines, # TODO: revist
71     stageLocalIntensityProb,
72     stageGrowFromPMap,
73     stageReconnect,
74     stageClosing,
75 )
76
77 #
78 # CollaboratiVesselParameterNode
79 #
80
81
82 @parameterNodeWrapper
83 class CollaboratiVesselParameterNode:
84     """
85     The parameters needed by module.
86     """
87     inputVolume: vtkMRMLScalarVolumeNode
88
89
90 # =====
91 # CollaboratiVesselLogic
92 # Transposed from main file for clarity
93 # =====
94
95
96 class CollaboratiVesselLogic(ScriptedLoadableModuleLogic):
97     """Implements the computation;
98     The interface should be such that other python code can import
99     this class and make use of the functionality without requiring an instance of the Widget.
100
101     Uses ScriptedLoadableModuleLogic base class, available at:
102     https://github.com/Slicer/Slicer/blob/main/Base/Python/slicer/ScriptedLoadableModule.py
103     """
104
105     # Helpers live in sibling modules; re-expose them on the class so the
```

```
106 # widget can keep calling self.logic.X without binding to those modules.
107 rasToIjk = staticmethod(rasToIjk)
108 pushArray = staticmethod(pushArray)
109 createSegmentationFromMask = staticmethod(createSegmentationFromMask)
110 exportBinarySlices = staticmethod(exportBinarySlices)
111 evaluateAgainstChunks = staticmethod(evaluateAgainstChunks)
112
113 def __init__(self) -> None:
114     """Called when the logic class is instantiated. Can be used for initializing member variables."""
115     ScriptedLoadableModuleLogic.__init__(self)
116
117     # Pipeline outputs are written here by runPipeline; widget reads
118     # self.logic.state.get("binaryMask") etc., so initialise empty.
119     self.state = {}
120
121     # Check imports
122     if not HAS_TUBETK:
123         raise RuntimeError("ITKTubeTK is required for ridge extraction. Install and restart:
slicer.util.pip_install('itk-tubetk')")
124
125     if not HAS_SKIMAGE:
126         raise RuntimeError("skimage is required for cLDICE. Install and restart: slicer.util.pip_install('scikit-
image')")
127
128     if not HAS_SCIPY:
129         raise RuntimeError("Scipy is required for smoothing. Install and restart: slicer.util.pip_install('scipy')")
130
131     if not HAS_SITK:
132         raise RuntimeError("SimpleITK is required for operations. Install and restart:
slicer.util.pip_install('simpleitk')")
133
134     if not HAS_PIL:
135         raise RuntimeError("PIL needed for img generation. Install and restart: slicer.util.pip_install('Pillow')")
136
137
138 def getParameterNode(self):
```

```
139         return CollaboratiVesselParameterNode(super().getParameterNode())
140
141
142
143
144     def runPipeline(self, volumeNode, vesselSeedIJKs, backgroundSeedIJKs, outsideIJKs, params):
145         """Run the full pipeline and cache outputs in self.state.
146         "params" parameters are to be defined on call from ui, with defaults here
147         """
148         vessel_size      = int(params.get("vessel_size", 4))
149         vessel_std        = int(params.get("vessel_std", 3))
150         structure_strength= int(params.get("structure_strength", 4))
151         frangi_tiles      = max(1, int(params.get("frangi_tiles_per_axis", 2)))
152         ridge_threshold   = float(params.get("ridge_threshold", 0.01))
153         ridge_iters       = int(params.get("ridge_iterations", 3))
154         auto_seed_rounds  = int(params.get("auto_seed_rounds", 2))
155         auto_seed_max     = int(params.get("auto_seed_max", 200))
156
157
158         log = []
159         spacing = volumeNode.GetSpacing()
160         origin = volumeNode.GetOrigin()
161
162         raw = scalarArray(volumeNode)
163
164         # 1. Frangi vesselness on raw ROI
165         frangi = stageFrangi(raw, spacing, vessel_size, vessel_std, structure_strength,
166                             tiles_per_axis=frangi_tiles)
167
168         # 2. Grey value outside filter
169         # We assume the distribution of background points is substantially different
170         roi = stageRemoveBackground(scalarArray(volumeNode), outsideIJKs, backgroundSeedIJKs)
171         # Remove frangi points near the outside edge
172         frangi = frangi * (roi > 0)
173         tile_note = (f"{frangi_tiles}^3={frangi_tiles**3} tile(s)")
174
```

```
175     log.append(  
176         f"Frangi ({tile_note}):  $\sigma$  sum "  
177         f"[{max(0.5, (vessel_size-vessel_std)/2.0):.1f}], "  
178         f"[{max(1.0, float(vessel_size+vessel_std)):.1f}] vox"  
179     )  
180  
181     # 2. Localized intensity probability  
182     # NOTE: this step, because of its locality, means that a strong signal  
183     # will drown out surrounding ones. One of the reasons behind needing to  
184     # rebuild the network  
185     p_intensity, stats = stageLocalIntensityProb(roi, vessel_size, vesselSeedIJKs)  
186     log.append(f"Intensity:  $\mu_v$ ={stats['mu_v']:.1f}+/-{stats['sig_v']:.1f}, ")  
187  
188     # Combine frangi with intensity map  
189     probMap = (frangi * p_intensity).astype(np.float32, copy=False)  
190  
191     # Save mem  
192     # del p_intensity  
193     # del raw  
194  
195     # soft_mask = probMap  
196     # mask = soft_mask >= 0.65  
197  
198     # 3. reconstruct vasculature  
199     mask, soft_mask = stageGrowFromPMap(  
200         probMap, spacing, origin, vesselSeedIJKs,  
201         vessel_size, vessel_std,  
202         n_rounds=auto_seed_rounds,  
203         max_new_seeds=auto_seed_max,  
204         ridge_threshold=ridge_threshold,  
205         ridge_iters=ridge_iters)  
206  
207     log.append(f"Tube growth: {int(mask.sum()):,} voxels (final)")  
208  
209     # Step 1: small morphological closing for fragment-level reconnection  
210
```

```
211     mask = stageClosing(mask)
212
213     # Step 2: your existing skeleton-endpoint bridging for larger gaps with evidence
214     mask = stageReconnect(mask, probMap, vessel_size,
215                           max_gap_voxels=vessel_size * 3,
216                           min_path_prob=0.1)
217
218     # Final threshold
219     # Use a percentile of the non-zero soft mask:
220     thresh = 0.12
221     mask = soft_mask >= thresh
222     log.append(f"Thresholded at: {thresh}")
223
224
225     self.state = {
226         "spacing": spacing, "origin": origin,
227         "vesselSeedIJKs": vesselSeedIJKs,
228         "backgroundSeedIJKs": backgroundSeedIJKs,
229         "log": log,
230         # The volumes we want to be able to visualize
231         "frangi": frangi,
232         "p_intensity": p_intensity,
233         "probMap": probMap,
234         "soft_mask": soft_mask,
235         # The final mask
236         "binMask": mask,
237     }
238     return self.state
239
240
241
242
243     # TODO: Old, from example code
244     # Remove if no needed, but check first if anything in here is missing in the main code
245     def process(self,
246               inputVolume: vtkMRMLScalarVolumeNode,
```

```
247         outputVolume: vtkMRMLScalarVolumeNode,
248         imageThreshold: float,
249         invert: bool = False,
250         showResult: bool = True) -> None:
251     """
252     Run the processing algorithm.
253     Can be used without GUI widget.
254     :param inputVolume: volume to be thresholded
255     :param outputVolume: thresholding result
256     :param imageThreshold: values above/below this threshold will be set to 0
257     :param invert: if True then values above the threshold will be set to 0, otherwise values below are set to 0
258     :param showResult: show output volume in slice viewers
259     """
260
261     if not inputVolume or not outputVolume:
262         raise ValueError("Input or output volume is invalid")
263
264     import time
265
266     startTime = time.time()
267     logging.info("Processing started")
268
269     # Compute the thresholded output volume using the "Threshold Scalar Volume" CLI module
270     cliParams = {
271         "InputVolume": inputVolume.GetID(),
272         "OutputVolume": outputVolume.GetID(),
273         "ThresholdValue": imageThreshold,
274         "ThresholdType": "Above" if invert else "Below",
275     }
276     cliNode = slicer.cli.run(slicer.modules.thresholdscalarvolume, None, cliParams, wait_for_completion=True,
update_display=showResult)
277     # We don't need the CLI module node anymore, remove it to not clutter the scene with it
278     slicer.mrmlScene.RemoveNode(cliNode)
279
280     stopTime = time.time()
281     logging.info(f"Processing completed in {stopTime-startTime:.2f} seconds")
```

