Perform moving mesh simulations based on 3D RTE as explained in Gruenwald et al. 2021

Prerequisites

- Visual Studio (https://visualstudio.microsoft.com/en/downloads/) full version with all packages
- Python and any IDE like Spyder or Sublime Text 3
  - Install: pip, numpy wheel, matplotlib, scipy
- UCD files (export of Tomtec ImageArena) for ventricular cavities at all time steps

Step 0: Use ucd2stl MATLAB code to convert UCD to STL files

- Pat3_DS5ja_RV_KL_00.ucd
- Pat3_DS5ja_RV_KL_01.ucd
- Pat3_DS5ja_RV_KL_02.ucd
- Pat3_DS5ja_RV_KL_03.ucd
- Pat3_DS5ja_RV_KL_04.ucd
- Pat3_DS5ja_RV_KL_05.ucd
- Pat3_DS5ja_RV_KL_06.ucd
- Pat3_DS5ja_RV_KL_07.ucd
- Pat3_DS5ja_RV_KL_08.ucd
- Pat3_DS5ja_RV_KL_09.ucd
- Pat3_DS5ja_RV_KL_10.ucd
- Pat3_DS5ja_RV_KL_11.ucd
- Pat3_DS5ja_RV_KL_12.ucd
- Pat3_DS5ja_RV_KL_13.ucd
- Pat3_DS5ja_RV_KL_14.ucd
- Pat3_DS5ja_RV_KL_15.ucd
- Pat3_DS5ja_RV_KL_16.ucd
- Pat3_DS5ja_RV_KL_17.ucd
- Pat3_DS5ja_RV_KL_18.ucd
- Pat3_DS5ja_RV_KL_19.ucd
- Pat3_DS5ja_RV_KL_header

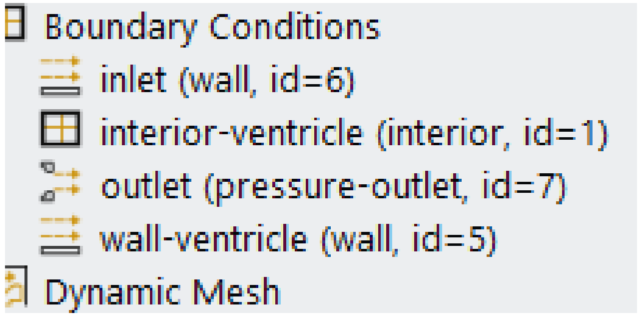
- ventricle_0
- ventricle_1
- ventricle_2
- ventricle_3
- ventricle_4
- ventricle_5
- ventricle_6
- ventricle_7
- ventricle_8
- ventricle_9
- ventricle_10
- ventricle_11
- ventricle_12
- ventricle_13
- ventricle_14
- ventricle_15
- ventricle_16
- ventricle_17
- ventricle_18
- ventricle_19

Step 1: Load one timestep and mesh in ANSYS

- Can be done through conversion of STL to stp (with Creo or Solidworks)
- Other approaches (Spaceclaim) also possible
- No specific requirements for meshing

Step 2: Use UDF udf_extPts.c through ANSYS Fluent

- Open FLUENT: Node Memory is used -> user defined -> memory -> turn on node Memory and adjust to number of used cores
- Check and adjust the zone IDs of the outer walls (here: Inlet, outlet and wall-ventricle) in the UDF (if necessary)




```c
/* Define zones of venctricle, inlet and outlet*/
#define ventricle 5
#define inlet 6
#define outlet 7
```

Step 2: Use UDF udf_extPts.c through ANSYS Fluent

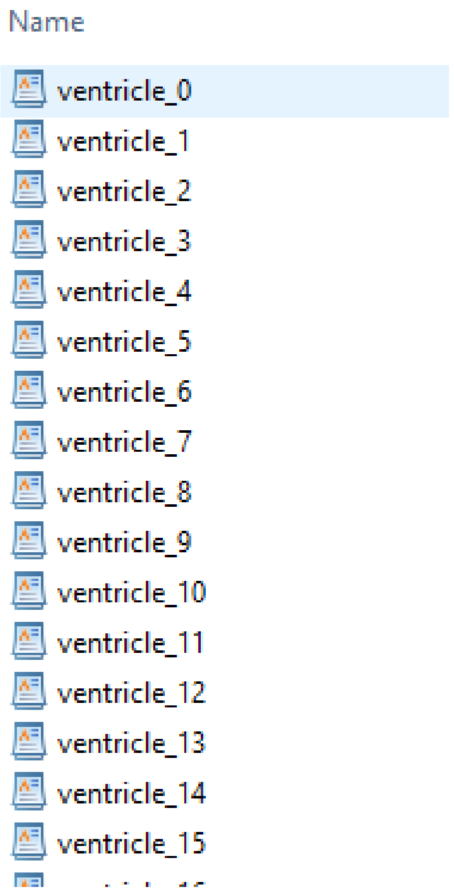
- Use of UDF in ANSYS Fluent
  - User defined -> functions -> compiled -> choose udf_extPts
  - User defined -> execute on demand -> First_AssignID
  - File „surface“ is created in the simulation folder
  - Number of nodes for each core are shown in the console (Will be necessary for udf_exeDynMesh)

surface file

```
0 6.485890 16.681522 28.130762
1 6.862917 16.265652 28.036011
2 6.569711 15.943861 27.625748
3 7.079221 15.724083 27.819656
4 7.156017 16.587324 28.446129
5 7.513730 15.881123 28.214535
6 7.481522 16.395054 28.535452
7 7.545926 15.367387 27.893742
8 7.887307 16.074398 28.607815
```

Step 3: Create STL folder

- Copy and rename existing STL files (of all time steps) into the folder STL
  - "ventricle_n" with n starting from 0
  - Folder STL is in the same location as code ps_detNPts.py

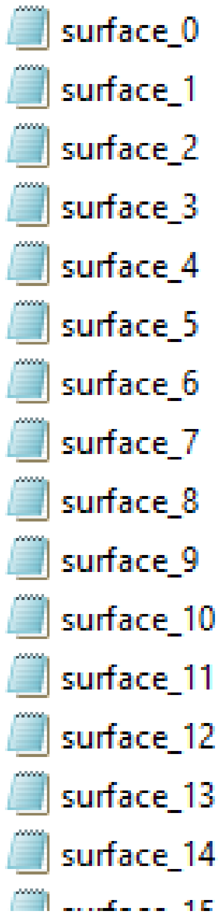


Step 4: Execute ps_detNPts.py ➔ STL to PTS

- With python e.g. in Sublime Text 3 or Spyder
- If first execution: Answer Do you need to generate correlation file? Yes: 1. No: 0. with “1” beantwortet
- The reference frame ID is the number of the STL-file in the folder STL, that has been loaded and meshed with ANSYS (see step 1).
- Output:
  - Number of Faces
  - allPtsNum

Step 4: Execute ps_detNPts.py ➔ STL to PTS

- First timepoint of cycle as Start FrameID (commonly 0)
- Last used timepoint as End FrameID
- Code creates "surface_" files in a folder PTS



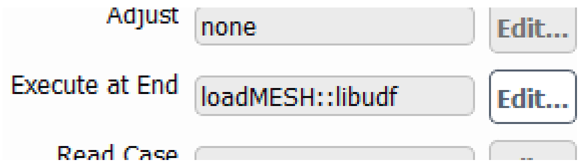
Step 5: Execute ps_intNPts.py ➔ PTS to UDFPTS

- Execute the ps_intNPts code in (same location as PTS folder)
- Set number of stl files per cycle as „Total number of frames in a cycle”
- Set number of interpolation steps (timesteps form simulation) „Number of intermediate frames“
- Choose a temporal interpolation method between the frames
- Output of Total Number of Frames after interpolation: # of timesteps for one cycle
- Code creates udfsurface.asc files in folder UDFPTS ➔ necessary for mesh movement

Step 6: Adjust udf_exeDynMesh.c

- Before execution several parameters in „dynMesh“ UDF have to be set:
  - nTimeSteps: Number of timesteps= maximum index of udf_surface (output of ps_intNPts.py)
  - allPtsNum: Number of all nodes, output of ps_detNPts.py
  - PtsNum[]: number of points per core (saved as array); Console output after execution of UDF First_Assign_ID (see slide 6)

Step 6: Execute udf_exeDynMesh.c



- User defined -> functions -> compiled -> choose dynMesh
- User defined -> functions -> function hooks -> execute at end -> add loadMesh
- Put folder UDFPTS into folder „Fluent“ of the simulation (same location of the „surface“ file, slide 6)
- User defined -> functions -> execute on demand -> First_load Mesh

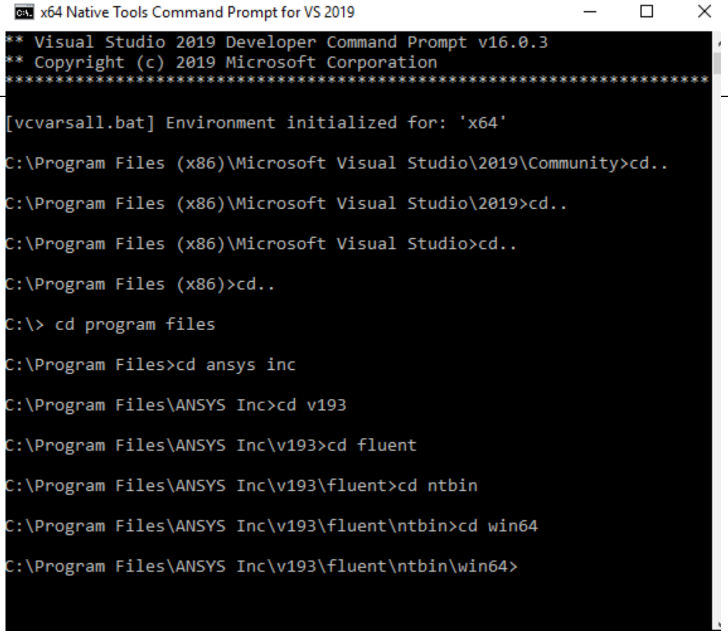
If compiling doesn‘t work

1. Compute locally(Desktop)

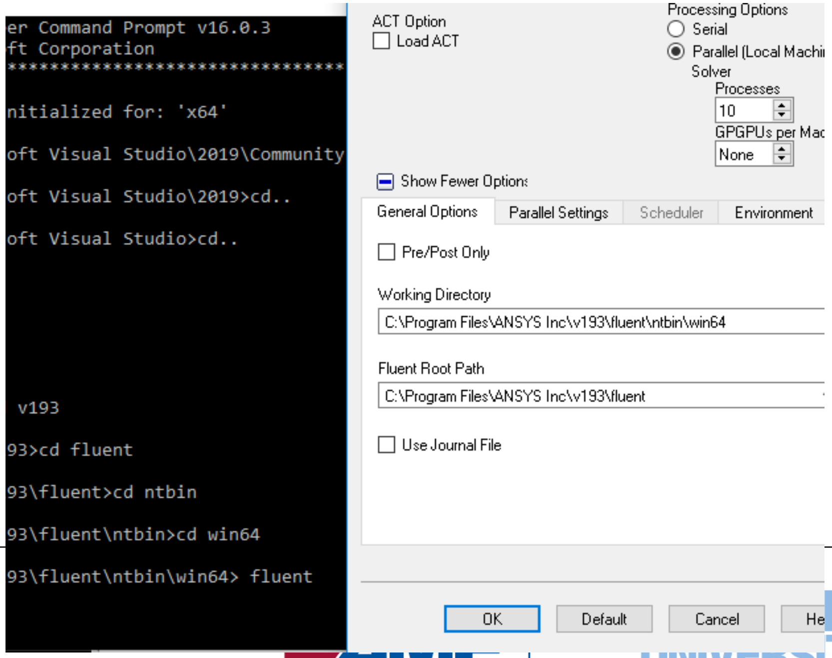
2. Compiling of UDFs

compiling with Native visual command prompt

Go to path and open fluent: enter fluent + ENTER

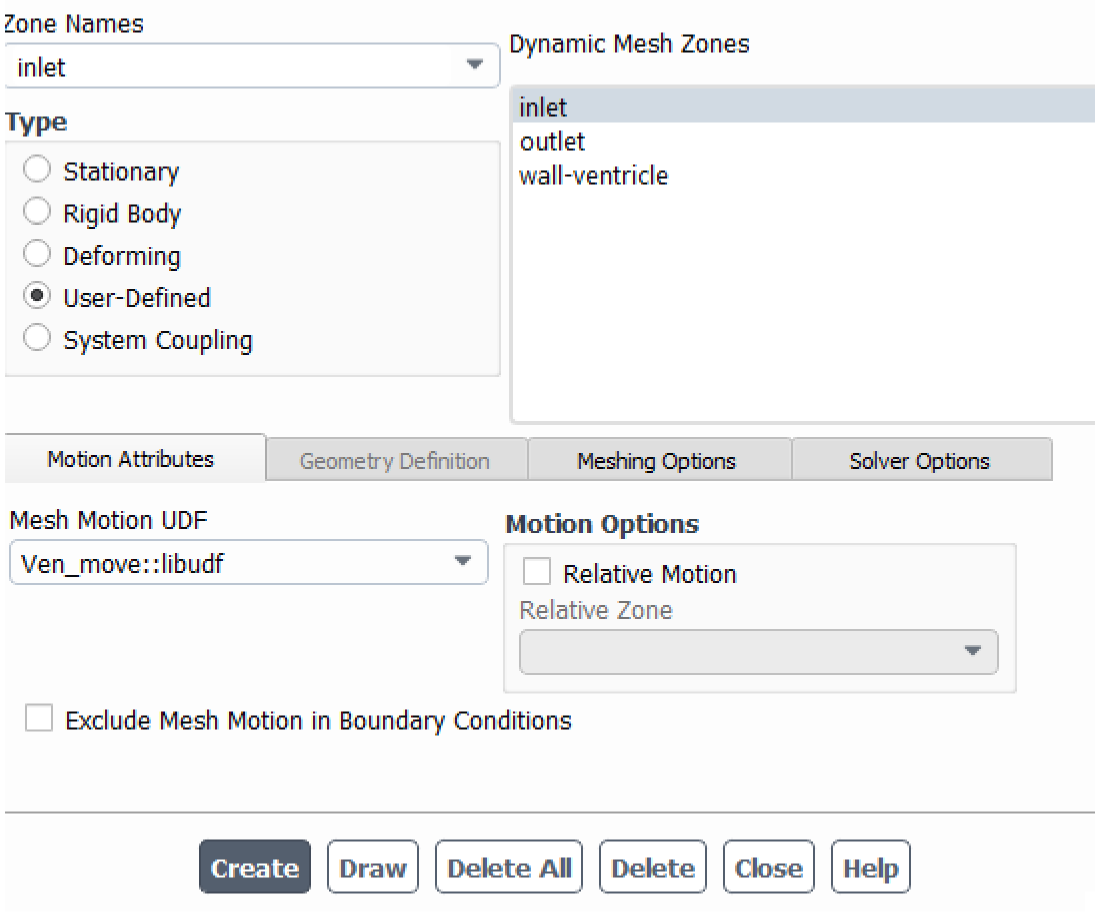


In launcher: set core numbers
Working directory where simulation has been saved

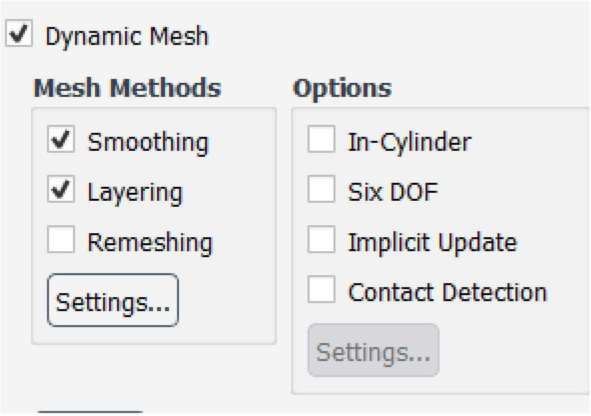


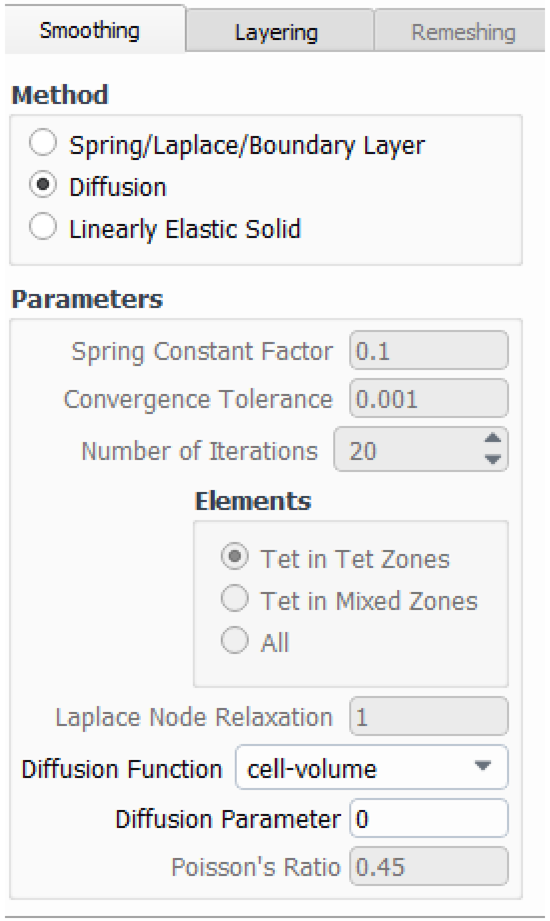
Use udf_exeDynMesh.c

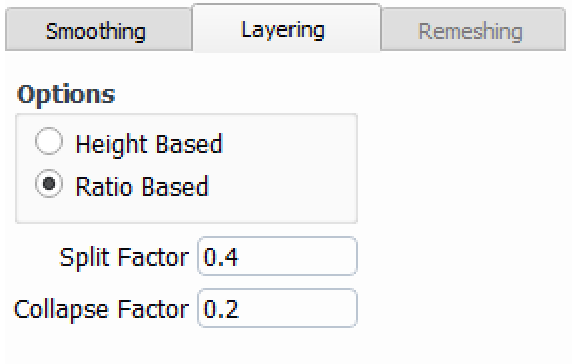
- Dynamic Mesh -> Dynamic Mesh zones -> create/edit -> user defined funktion for inlet, outlet, wall-ventricle erstellen



Dynamic Mesh Settings

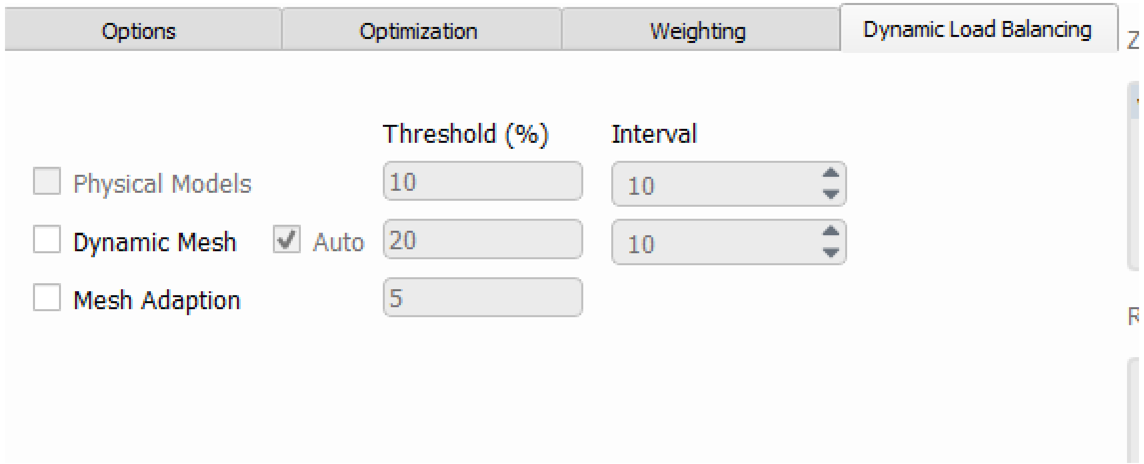






Prohibit repartitioning

- Parallel -> Partition/Load Balance ->



Mesh motion

- Before simulation start → preview mesh motion
  - Preview Mesh Motion
- If mesh movement doesn‘t work (e.g. negative cell volumes):
  - Fluent in serial model→ Neustart Fluent → Fluent Launcher ***serial***
  - Recompute mesh nodes (steps 2 -5)
  - Settings: Dynamic Mesh → Mesh Methods → Remeshing with default settings

Valve opening/closing

- In „Dynamic Mesh“ → „Events“ set three events
  - Inlet to wall
  - Outlet to pressure-outlet
  - Profile to pressure-outlet

- Initialise simulation, then firstloadmesh, then simulation starten
- At end of cycle NO initialisation, but
  - Inlet to pressure-inlet, Outlet to wall
  - Event – time adjusted to MV closing of next cycle
- start w/o initialisation

Contact

Michael Neidlin

neidlin@ame.rwth-aachen.de or

michael.neidlin@gmail.com