User's Guidance of GMGS-3D

By Junpeng Wang
(junpeng.wang@tum.de)

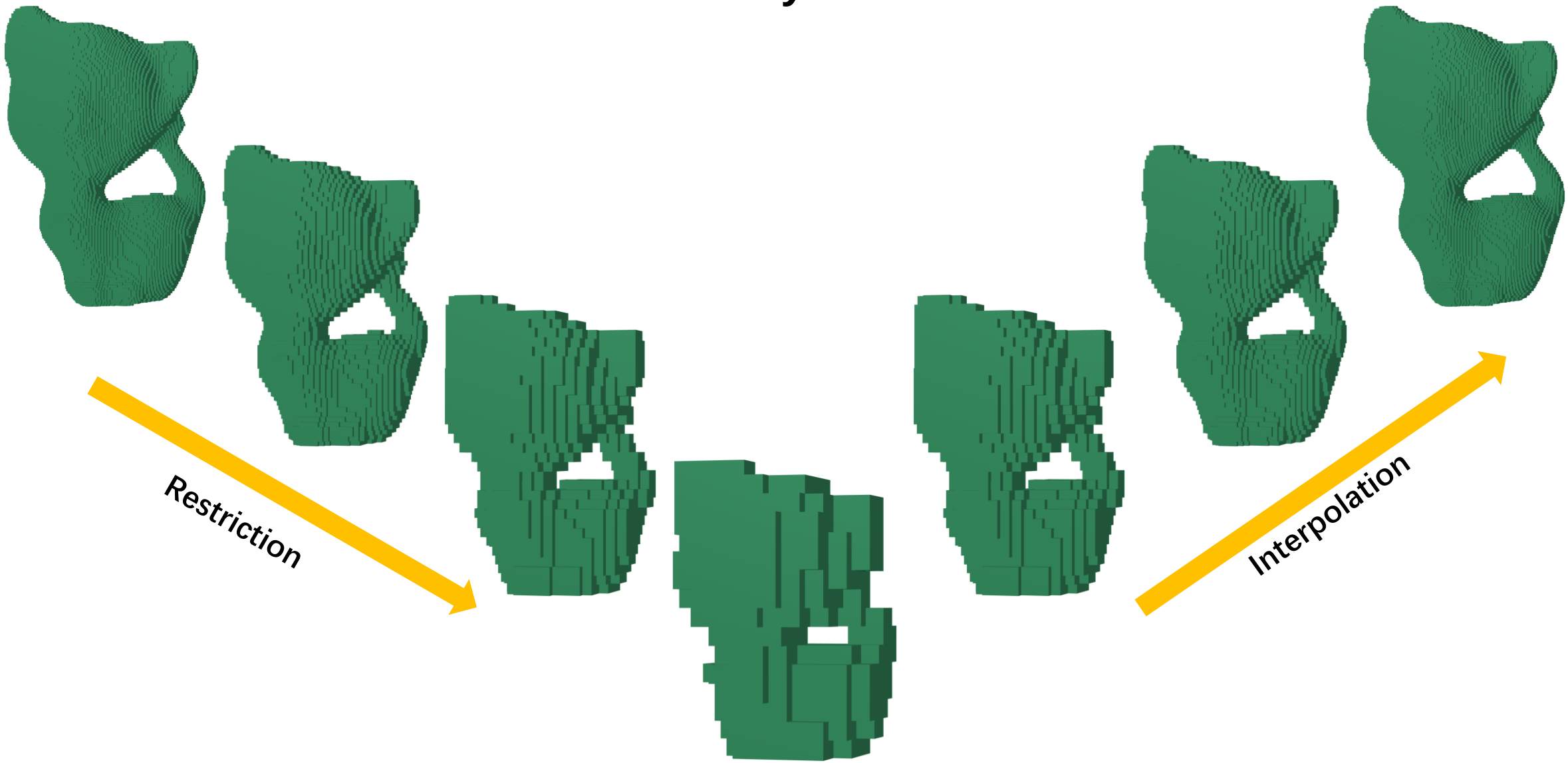
GMGS-3D
A Geometric Multigrid Solver (GMGS) for Large-scale Static Finite Element Simulation on 3D Cartesian Mesh

Design Target
Out of academic use, "GMGS-3D" is designed to generate the high-resolution data sets of vector (displacement) or 2nd-order tensor (stress) field with the NORMAL PC within an AFFORDABLE time.

Design Description
"GMGS-3D" proceeds the static Finite Element Analysis (FEA) for solid objects discretized into Cartesian mesh, where,

1) an element index based data structure is used to store the FEA stiffness matrix;
2) combined with the Jacobian smoother, a Geometric Multigrid based V-cycle is built on the Cartesian mesh;
3) the FEA equation is iteratively solved by Conjugate Gradient Method preconditioned with V-cycle;
4) besides the displacement vector field, the stress tensor field also can be computed.

V-cycle



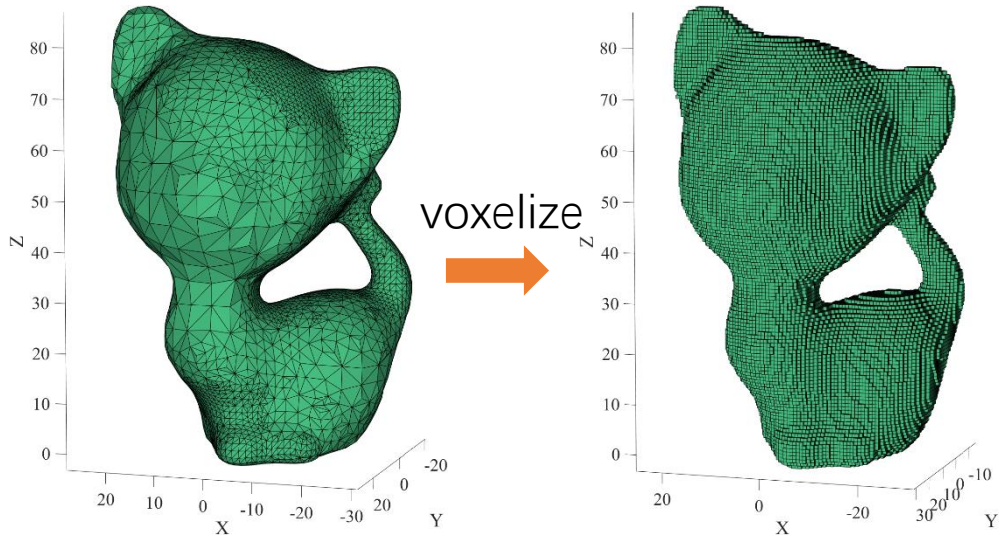
Directly Solving on Coarsest Level

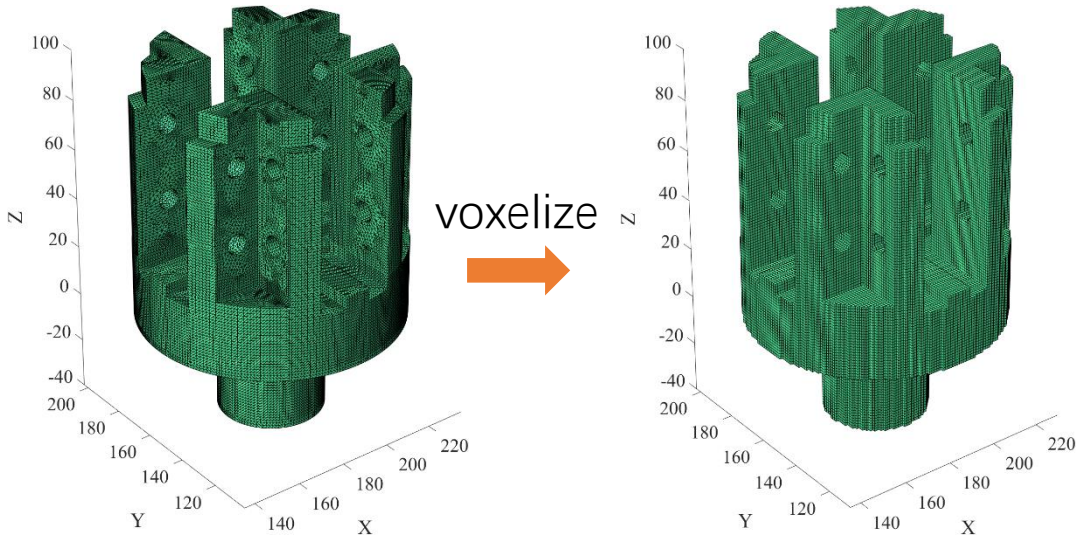
How to play around with GMGS-3D?

GMGS-3D takes the closed surface discretized with the triangular mesh (in the standard '.ply' format) as input data set, the simulation mesh is the hexahedral Cartesian mesh, which is created by voxelizing the region within the input closed surface (see below). The mesh resolution is controlled by user.

With the attached data sets "../data/kitten.ply" and "../data/parts.ply", and their corresponding boundary conditions, the script "../src/Demo.m" shows an overview of GMGS-3D in terms of its functionality.

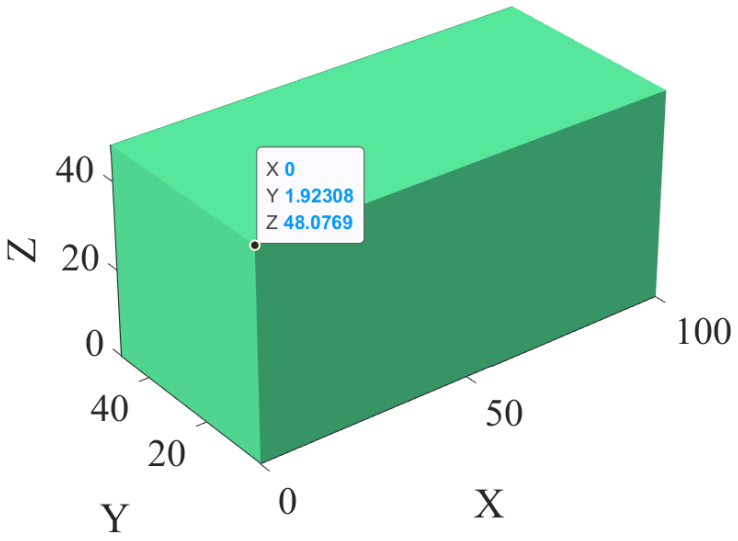
For general use, one needs to go to the script "../src/GMGS3D.m", where the specific boundary conditions need to be defined according to the specific models/requirements, which might not be very handy to the newer who don't know the data structure very well. To this end, with some examples, the following pages give guidance about how to apply for the boundary conditions, which hopefully can cover the most common use requirement.

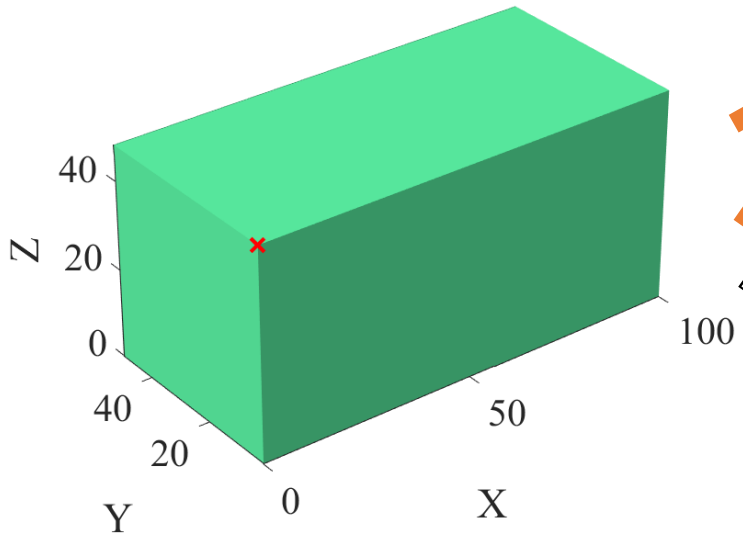




Fixing or Loading a Point

Step 1: pick up a position by cursor





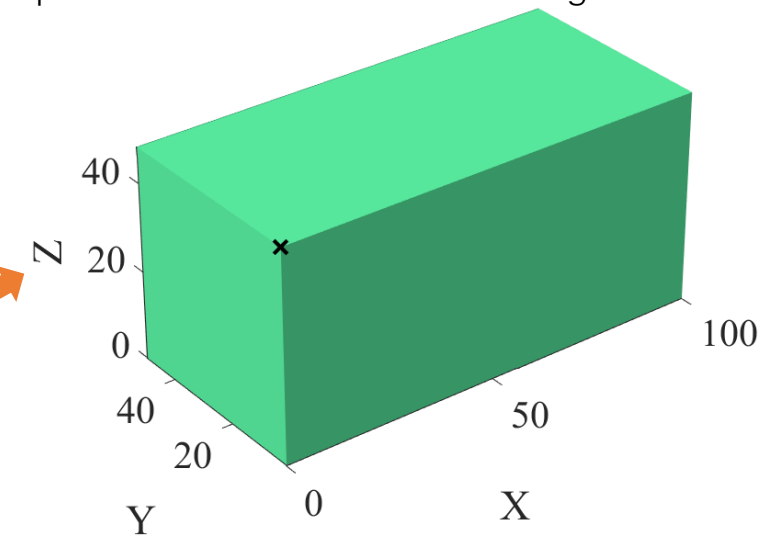
Step 2: "../src/common/PickByPoint"

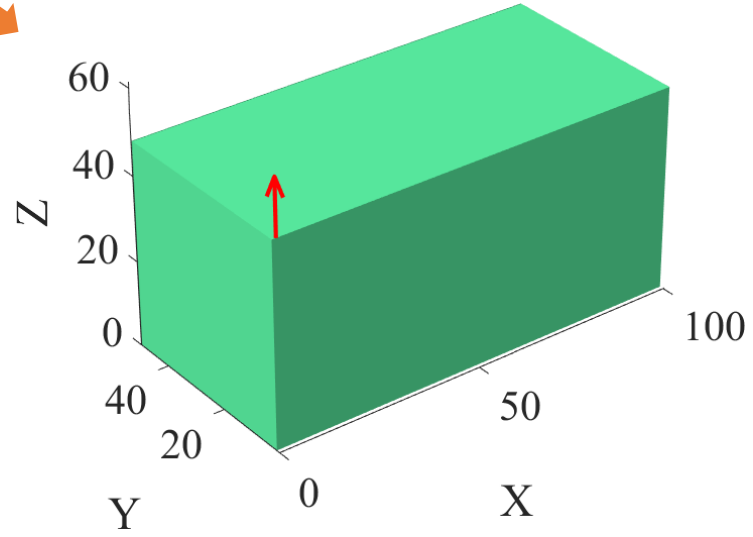
Step 3: "../src/common/InitializeFixingCond"

Fixing

or

Loading

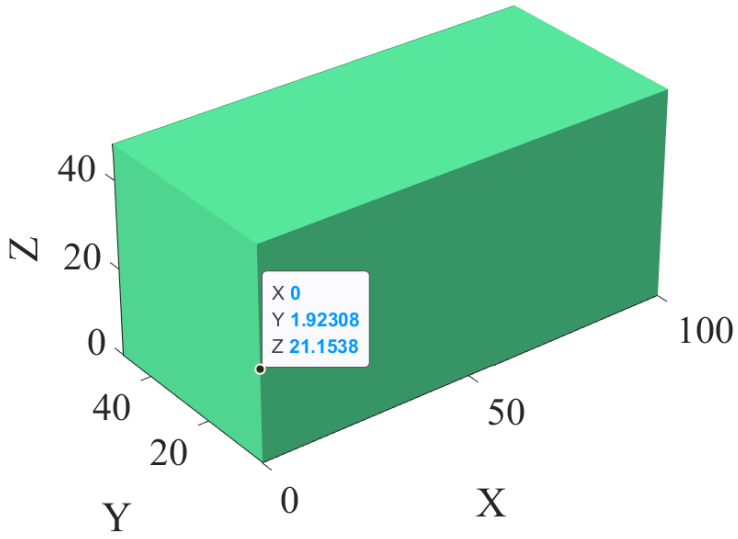


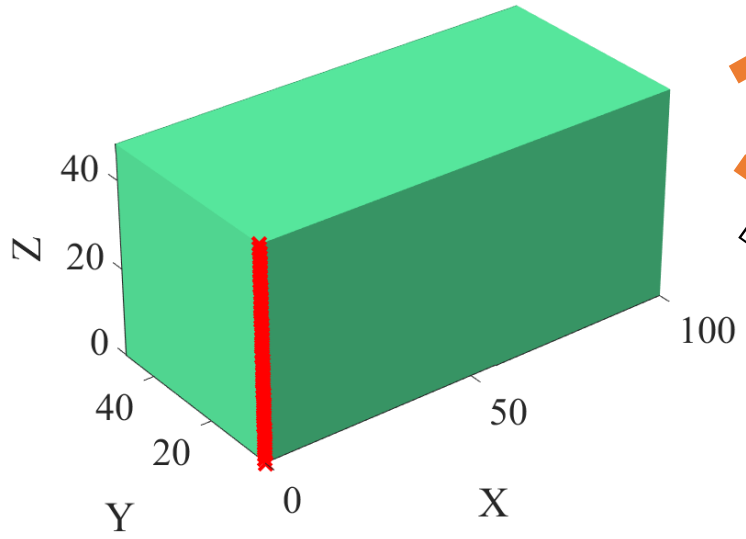


Step 3: "../src/common/InitializeLoadingCond([0 0 1])"

Fixing or Loading a Line

Step 1: pick up a position by cursor





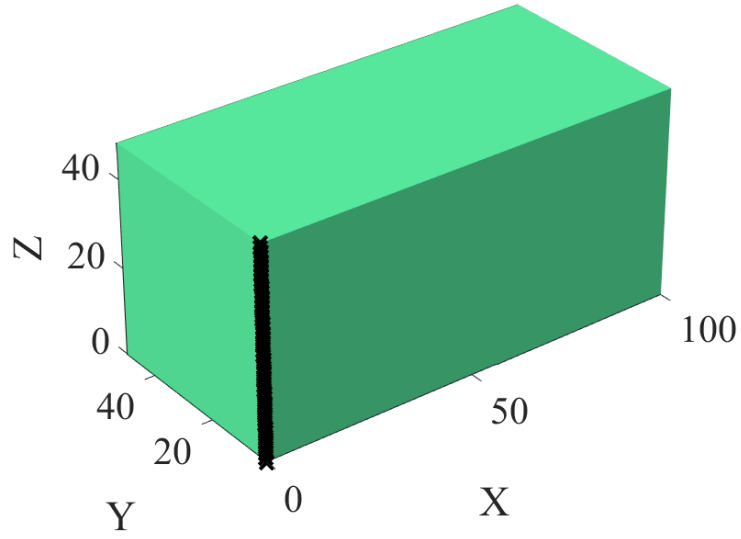
Step 2: "../src/common/PickByLine(["X", "Y"])"

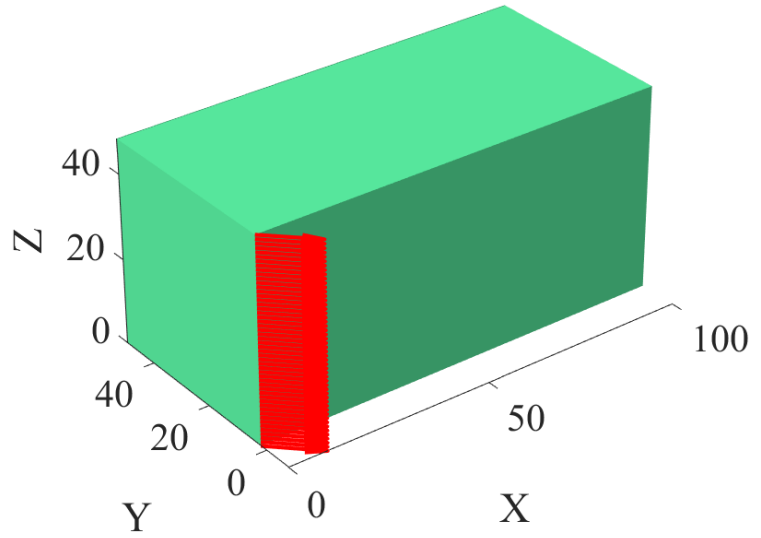
Fixing

or

Loading

Step 3: "../src/common/InitializeFixingCond"

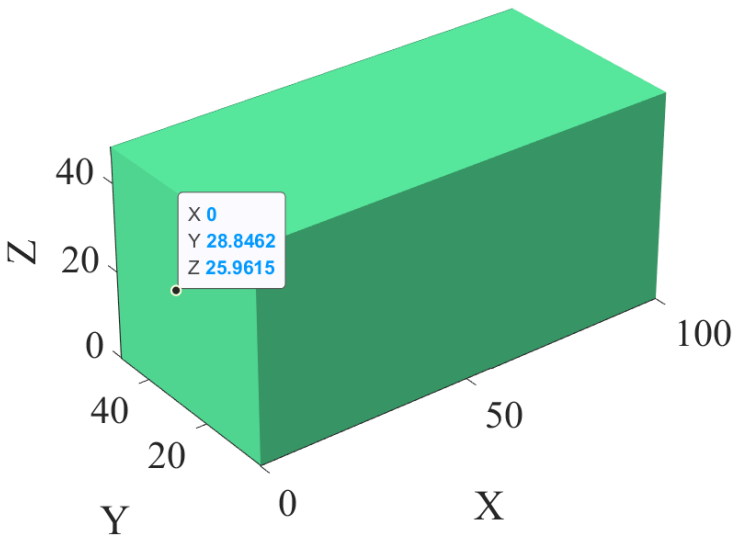


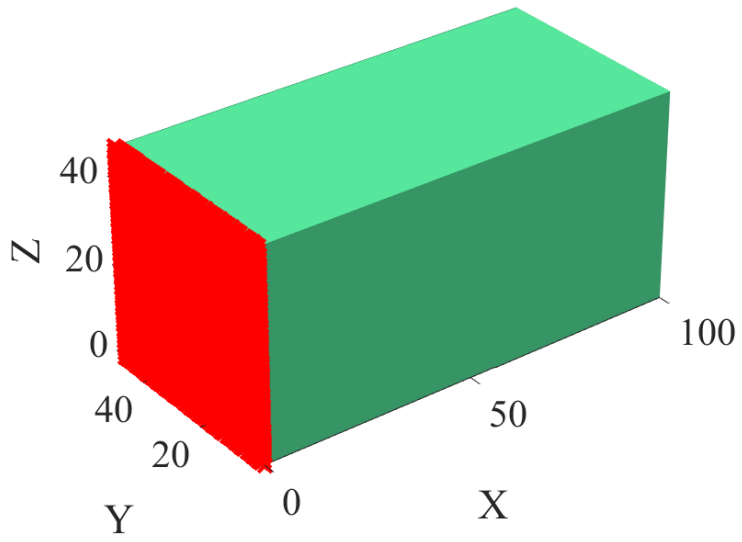


Step 3: "../src/common/InitializeLoadingCond([1 -1 0])"

Fixing or Loading a Surface

Step 1: pick up a position by cursor





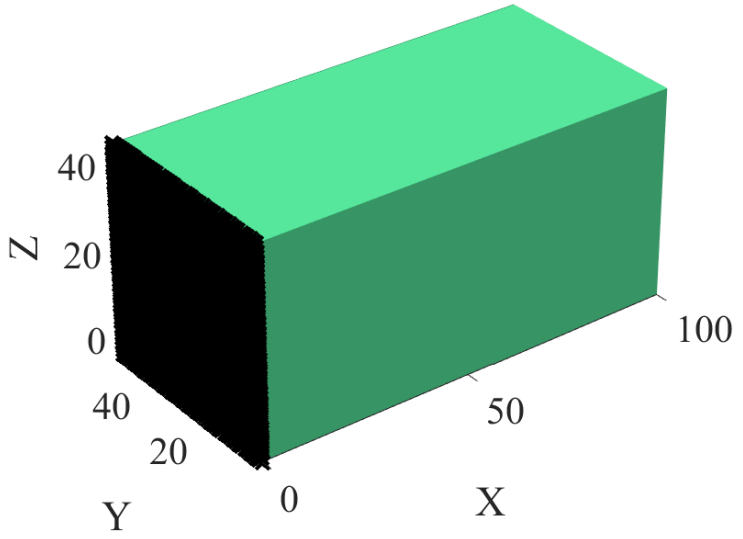
Step 2: "../src/common/PickBySurface('X')"

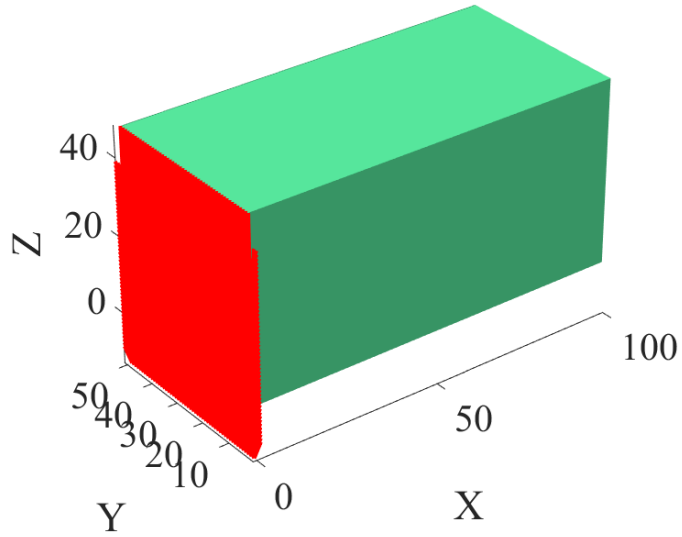
Fixing

or

Loading

Step 3: "../src/common/InitializeFixingCond"

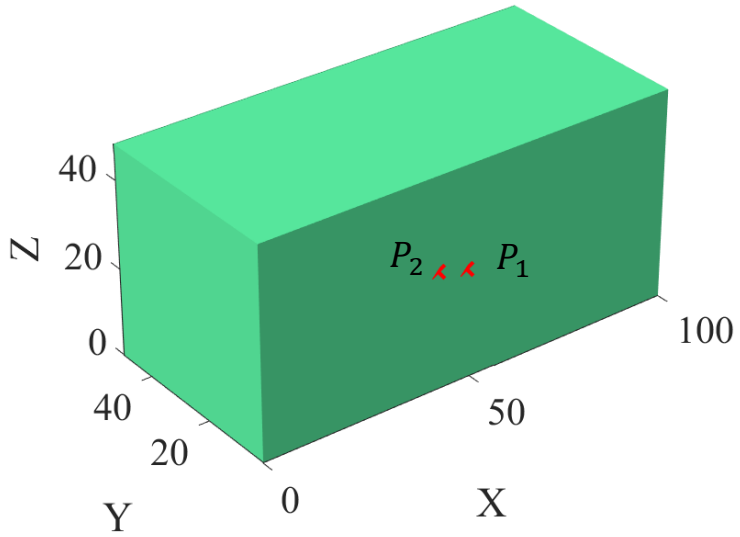


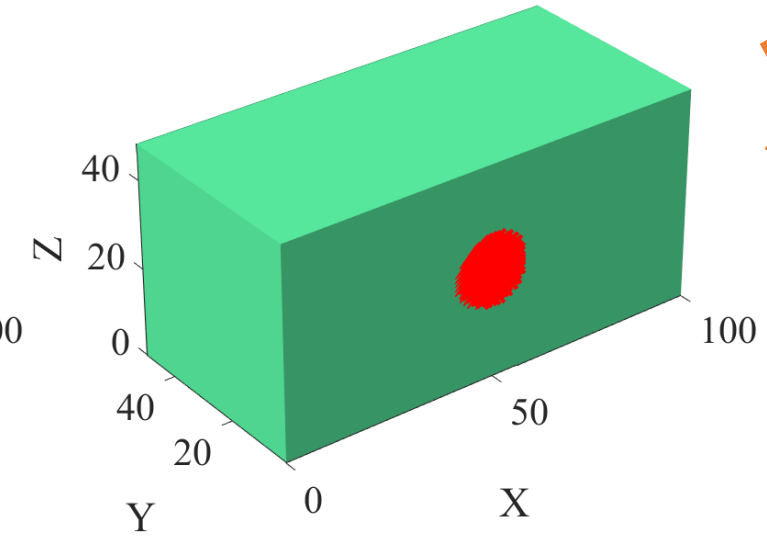


Step 3: "../src/common/InitializeLoadingCond([0 0 -1])"

Fixing or Loading a Circular Region

Step 1: sequentially pick up two points P_1 and P_2 by "../src/common/PickByPoint"





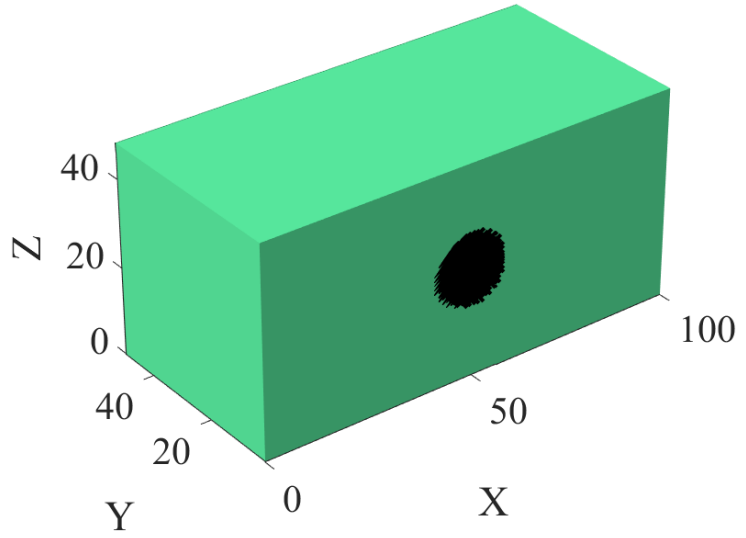
Step 2: "../src/common/PickByCircle"
(Pick up the nodes in the circular region centered at P_1 with radius $|P_1P_2|$)

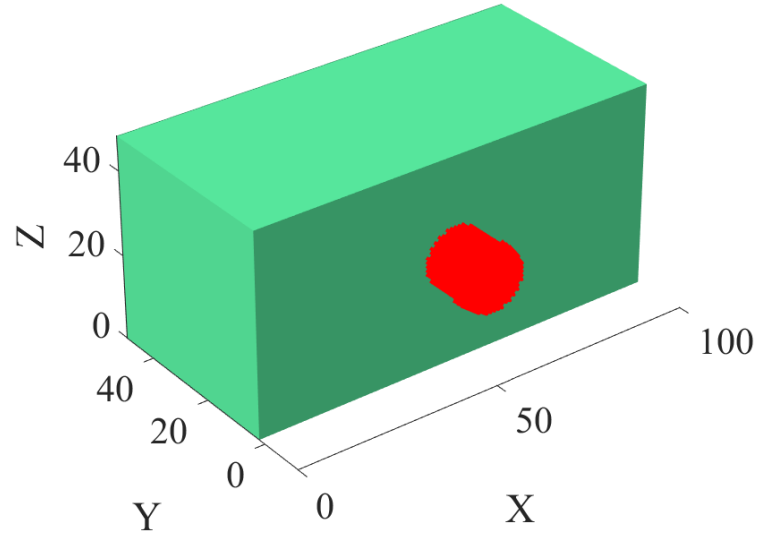
Fixing

or

Loading

Step 3: "../src/common/InitializeFixingCond"





Step 3: "../src/common/InitializeLoadingCond([0 -1 0])"