Final Project. Geometric multigrid solver using a hierarchical mesh

Mathias Schmidt

CU Boulder's Center for Aerospace Structures develops its own Finite Element Optimization code, “Moris”. This code should be able to perform Multiphysics Finite Element Analysis. One existing functionality of this code is it, to create a hierarchical mesh. The degrees of freedom are expressed by B-Spline basis. The benefit of this hierarchical structure is that the mesh can be refined according to an arbitrary criterion.

Theory hierarchical refined B-spline mesh

The finite element method is often based on Lagrange based meshes. Therefore, a given field is discretized by Lagrange interpolation functions. The degrees of freedoms (dof) are located at the nodes of the element. In areas with high field gradients a higher resolution is advantageous and thus, a refined mesh can be beneficial. B-Spline meshes are an alternative to Lagrange based meshes. B-spline meshes often need less degrees if freedom for the same accuracy than Lagrange meshes. Furthermore, they can create smooth spatial field derivatives of order $p-1$, where p is the chosen polynomial degree of interpolation of an field. Therefore, CU Boulder's Center for Aerospace Structures developed an adaptively refining hierarchical B-spline mesh generator.

A field $\phi(x)$ is discretized with respect to a given mesh. The interpolation is given through

$$\phi\left(x\right) \approx N_i(x)\widetilde{\phi_i}$$

where $N_i(x)$ are piecewise defined polynomials, in this case B-splines, and $\widetilde{\phi_i}$ are control points that determine the weight. B-splines of the order p can be represented by $p+2$ bases of span per spatial dimension. These smaller b-splines shall be called children. For a one dimensional spline one obtains

$$N^n = \sum_{k=0}^{p+1} w_k N_k^{n+1}$$

where n stands for the refinement level. The weights w_k can be determined by

$$w_k = \frac{1}{2^p} \begin{pmatrix} p+1 \\ k \end{pmatrix}$$

Figure 1. Shows such a refinement where a B-spline is represented through four weighted children B-splines. The presented example is constructed with quadratic B-splines.

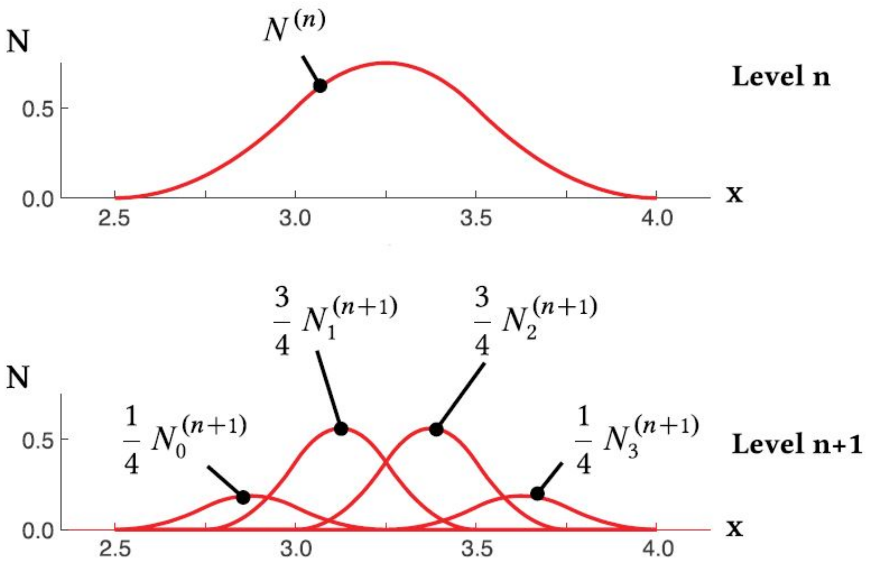


Figure 0.1: Quadratic B-spline represented through sum of smaller B-splines

Figure 2. shows a domain discretized with B-splines. Two refinement steps are done, replacing coarse B-splines by their children representations.

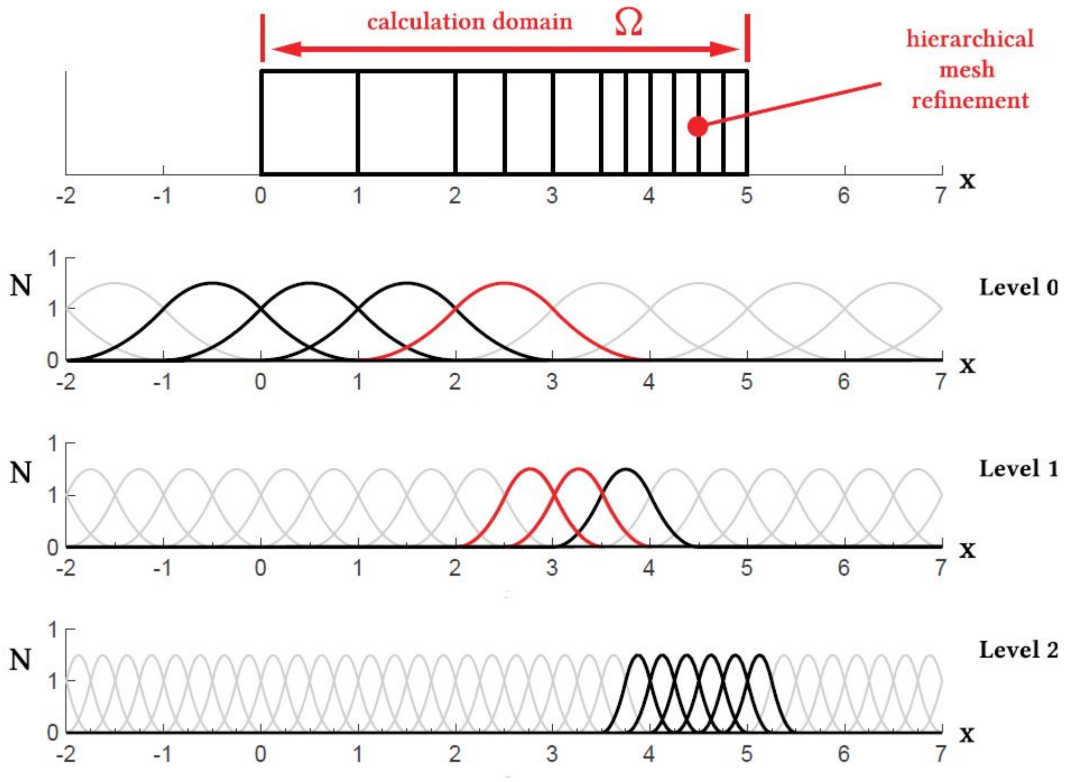


Figure 0.2: Hirarchical refined B-spline mesh

Mapping between B-spline and Lagrange mesh

A field can be discretized on a nodal basis on the mesh. A possible example for a nodal field is a signed distance field. For testing purpose, such a signed distance field will be used later on for verification of the implementation. Using a least squares projection, field data on the B-spline mesh can be mapped to a Lagrange mesh. When the B-Spline mesh is used to solve a finite element problem, the concept of element extraction plays an important role. The basic idea of element extraction is to project the B-spline degrees of freedom into a Lagrange based space, so that they can be interpreted by a Lagrange element. The Lagrange interpolation requires the field values $\hat{\phi}_j$ at the supporting Lagrange nodes $\hat{x_j}$. A transformation matrix T_{lk} can be used to transform by evaluating the B-spline interpolation functions N_i at the Lagrange nodes.

$$\hat{\phi}_j = T_{lk}\widetilde{\phi_k}$$

Code implementation

Hierarchical mesh refinement(HMR) implementation The idea of this project is to exploit the hierarchical nature of the B-spline basis to create prolongation and restriction operators which can then be used to create a geometric multigrid preconditioner. Therefore, the hierarchical mesh generator must be modified in a way that it calculates the projection stencils corresponding to the pairs of fine and coarse B-spline basis. This is done in the appended file cl_HMR_Bspline_Mesh_Base.cpp. The function BSpline_Mesh_Base::calculate_child_stencil() calculates a non-truncated stencil for two and three dimensions. This stencil is adjusted for every parent basis and can be requested using BSpline_Mesh_Base::get_children_ind_for_basis() and BSpline_Mesh_Base::get_children_weights_for_parent(). The first function returns the children indices. The second returns the corresponding weights.

Model Solver Interface (MSI) implementation The Model Solver Interface serves as the interface between the finite element model and the nonlinear or time solver. This module enables building and solving the following system of nonlinear equations:

$$F_i\left(u_k\left(a_j\right)\right) = 0$$

where F_i is the residual equation, a_j are independent abstract variables and u_k are dependent physical variables. It is assumed that the residuals F_i are functions of physical variables, i.e. physical degrees of freedom (pdofs), which in turn depend on abstract degrees of freedom (adofs). The pdofs are defined by a type, such as temperature or displacement in x-direction, and a vector of time step indices. Adofs are derived from pdofs using information provided byt the transformation matrices presented above. Each adof is linked to one type and time step ID.

To create a multigrid preconditioner, the model solver interface has to provide a mapping between the different degrees of freedom on the multigrid levels, for each dof type and time. This is done in the file cl_MSI_Multigrid.cpp.

Linear Solver multigrid implementation In order to use PETSc, wrappers are written around the PETSc matric, vector, and the linear solver. This was done to fit PETSc into the existing framework. In addition, the interpolation operators are assembles in cl_DLA_Geometric_Multigrid.cpp. The interpolation operators are assembled using the weights provided by the hierarchical mesh and the mapping provided by the model solver interface. The operators are provided to PETSc's PCMG module to build a multigrid preconditioner. This is done in the file cl_DLA_Linear_Solver_PETSc.cpp.

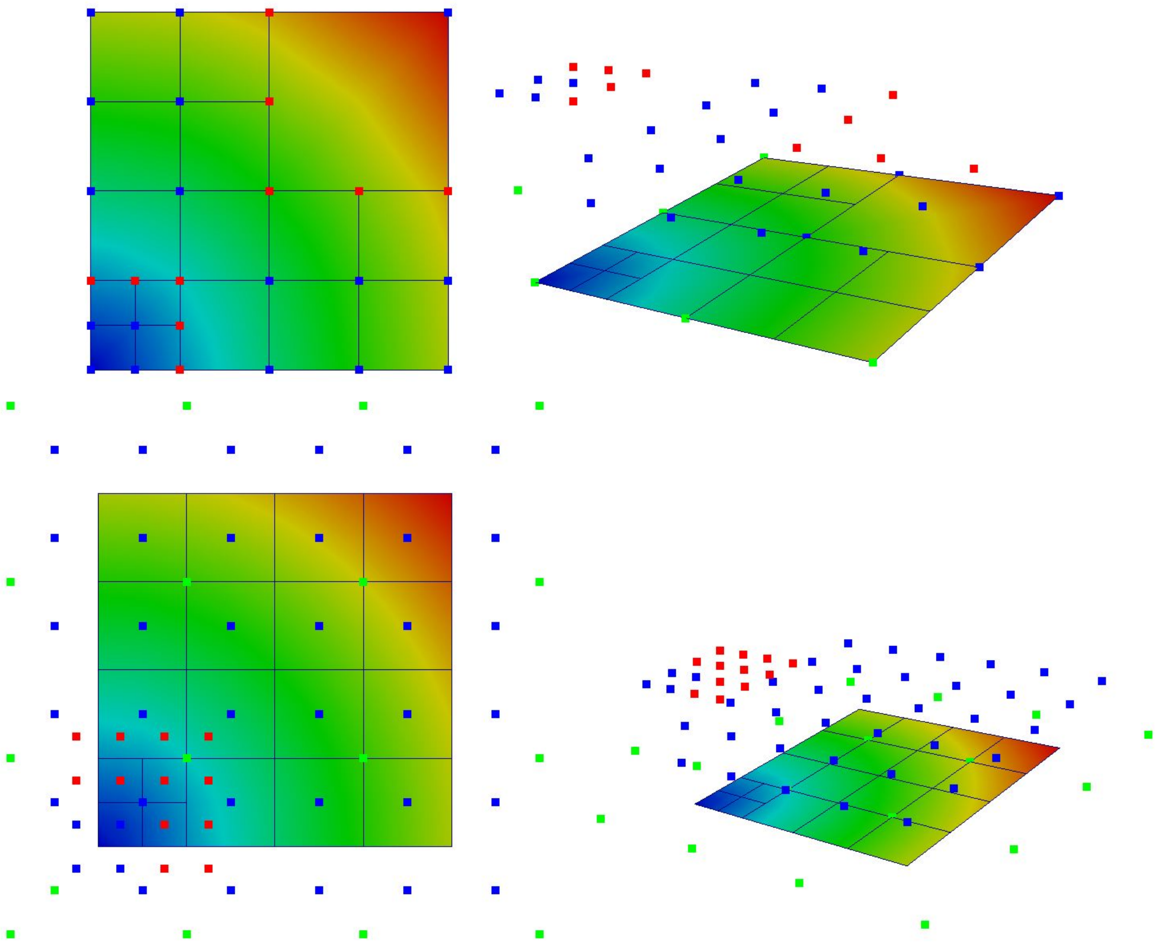
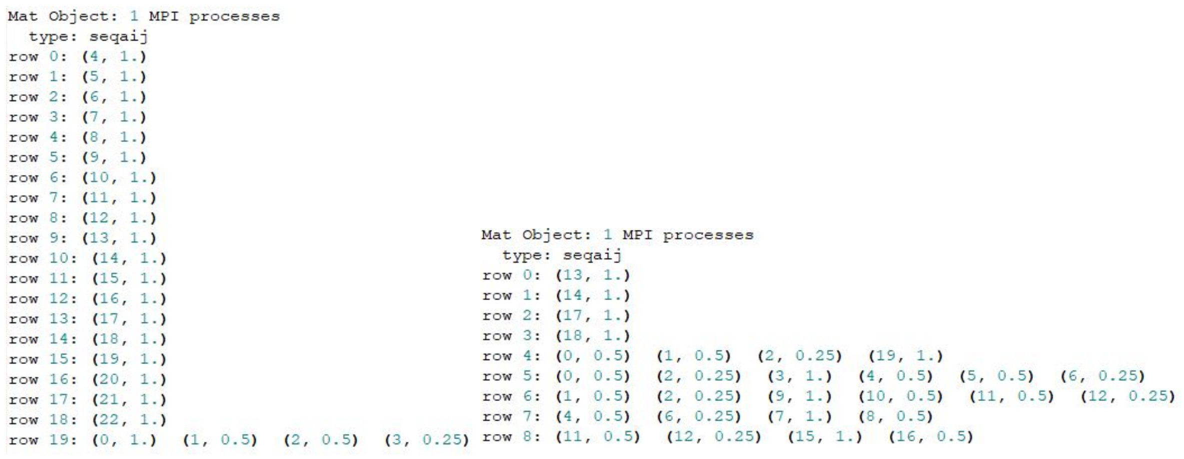


Figure 0.3: Top: Linear B-spline basis. Bottom: quadratic B-spline Basis. Linear and quadratic basis are presented in a view from above and from an angle. The B-spline basis in different levels are warped to present the levels.

Evaluation and Results

For debugging purposes two simple hierarchical meshes are created. The first mesh operates on linear B-splines, the second one on quadratic B-splines. In both meshes the bottom left element is refined twice. The meshes and the corresponding B-spline basis are presented in Figure 3. The top row shows linear B-spline basis. The bottom row quadratic B-spline basis. In the figure to the right the B-spline basis on different levels are warped to sow the levels. Red basis are inactive. Blue ones are active on the fines level. Green ones are inactive on the fines level but will be used when a coarser level is reached during the multigrid process. The presented mesh on the plane is the corresponding Lagrange mesh. The different refinement patterns between linear and quadratic plot are based on a refinement buffer criterion which increases with increasing B-spline order. The shown meshes have two coarser multigrid meshes.



```
Mat Object: 1 MPI processes
  type: seqaij
row 0: (4, 1.)
row 1: (5, 1.)
row 2: (6, 1.)
row 3: (7, 1.)
row 4: (8, 1.)
row 5: (9, 1.)
row 6: (10, 1.)
row 7: (11, 1.)
row 8: (12, 1.)
row 9: (13, 1.)
row 10: (14, 1.)
row 11: (15, 1.)
row 12: (16, 1.)
row 13: (17, 1.)
row 14: (18, 1.)
row 15: (19, 1.)
row 16: (20, 1.)
row 17: (21, 1.)
row 18: (22, 1.)
row 19: (0, 1.)  (1, 0.5)  (2, 0.5)  (3, 0.25)
```

```
Mat Object: 1 MPI processes
  type: seqaij
row 0: (13, 1.)
row 1: (14, 1.)
row 2: (17, 1.)
row 3: (18, 1.)
row 4: (0, 0.5)  (1, 0.5)  (2, 0.25)  (19, 1.)
row 5: (0, 0.5)  (2, 0.25)  (3, 1.)  (4, 0.5)  (5, 0.5)  (6, 0.25)
row 6: (1, 0.5)  (2, 0.25)  (9, 1.)  (10, 0.5)  (11, 0.5)  (12, 0.25)
row 7: (4, 0.5)  (6, 0.25)  (7, 1.)  (8, 0.5)
row 8: (11, 0.5)  (12, 0.25)  (15, 1.)  (16, 0.5)
```

Figure 0.4: Multigrid operators for the linear case. Left: First coarsening step. Right: Second coarsening step

The created prolongation or restriction operators for these B-Spline meshes can be printed to screen and compared to the operator created by hand. This was done and the following two operators were found for the linear case. The found operators correspond the expected result.

The same was done for the operatores based on quadratic B-splined. The sparse matrices correspond the expected results again.

Multigrid Solver

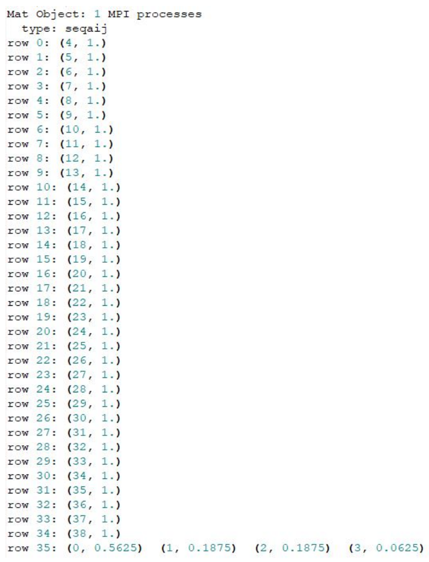
The effectiveness of the multigrid preconditioner is compared to an ILU preconditioner. Therefore, the following settings for the ILU preconditioner and the multigrid preconditioner were chosen. As a solver a flexible gmres is used.

ILU-Preconditioner

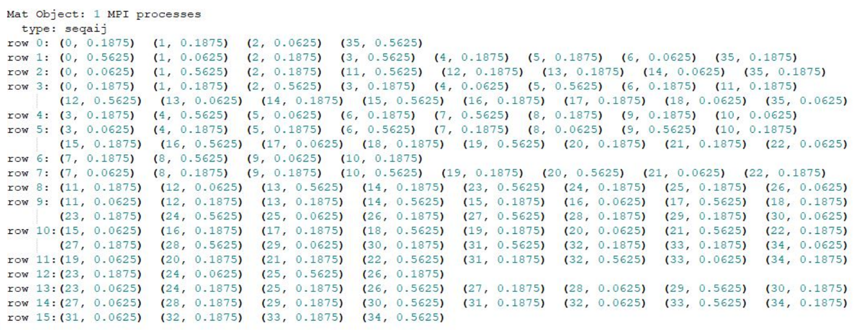
- Level of fill: 0
- Drop tolerance: 10^{-6}

Multigrid Preconditioner

- Two coarser multigrid level
- Multigrid form: Multiplicative
- The coarser grids are computed from the fines grid using the Galerkin process
- A preconditioned Richardson iterative method is used to solve on the coarsest level
- As a smoother down and up the V-cycle one iteration of a GMRES is used with a jacobi preconditioner



```
Mat Object: 1 MPI processes
  type: seqaij
row 0: (4, 1.)
row 1: (5, 1.)
row 2: (6, 1.)
row 3: (7, 1.)
row 4: (8, 1.)
row 5: (9, 1.)
row 6: (10, 1.)
row 7: (11, 1.)
row 8: (12, 1.)
row 9: (13, 1.)
row 10: (14, 1.)
row 11: (15, 1.)
row 12: (16, 1.)
row 13: (17, 1.)
row 14: (18, 1.)
row 15: (19, 1.)
row 16: (20, 1.)
row 17: (21, 1.)
row 18: (22, 1.)
row 19: (23, 1.)
row 20: (24, 1.)
row 21: (25, 1.)
row 22: (26, 1.)
row 23: (27, 1.)
row 24: (28, 1.)
row 25: (29, 1.)
row 26: (30, 1.)
row 27: (31, 1.)
row 28: (32, 1.)
row 29: (33, 1.)
row 30: (34, 1.)
row 31: (35, 1.)
row 32: (36, 1.)
row 33: (37, 1.)
row 34: (38, 1.)
row 35: (0, 0.5625)  (1, 0.1875)  (2, 0.1875)  (3, 0.0625)
```



```
Mat Object: 1 MPI processes
  type: seqaij
row 0: (0, 0.1875)  (1, 0.1875)  (2, 0.0625)  (35, 0.5625)
row 1: (0, 0.5625)  (1, 0.0625)  (2, 0.1875)  (3, 0.5625)  (4, 0.1875)  (5, 0.1875)  (6, 0.0625)  (35, 0.1875)
row 2: (0, 0.0625)  (1, 0.5625)  (2, 0.1875)  (11, 0.5625)  (12, 0.1875)  (13, 0.1875)  (14, 0.0625)  (35, 0.1875)
row 3: (0, 0.1875)  (1, 0.1875)  (2, 0.5625)  (3, 0.1875)  (4, 0.0625)  (5, 0.5625)  (6, 0.1875)  (11, 0.1875)
       (12, 0.5625)  (13, 0.0625)  (14, 0.1875)  (15, 0.5625)  (16, 0.1875)  (17, 0.1875)  (18, 0.0625)  (35, 0.0625)
row 4: (3, 0.1875)  (4, 0.5625)  (5, 0.0625)  (6, 0.1875)  (7, 0.5625)  (8, 0.1875)  (9, 0.1875)  (10, 0.0625)
row 5: (3, 0.0625)  (4, 0.1875)  (5, 0.1875)  (6, 0.5625)  (7, 0.1875)  (8, 0.0625)  (9, 0.5625)  (10, 0.1875)
       (15, 0.1875)  (16, 0.5625)  (17, 0.0625)  (18, 0.1875)  (19, 0.5625)  (20, 0.1875)  (21, 0.1875)  (22, 0.0625)
row 6: (7, 0.1875)  (8, 0.5625)  (9, 0.0625)  (10, 0.1875)
row 7: (7, 0.0625)  (8, 0.1875)  (9, 0.1875)  (10, 0.5625)  (19, 0.1875)  (20, 0.5625)  (21, 0.0625)  (22, 0.1875)
row 8: (11, 0.1875)  (12, 0.0625)  (13, 0.5625)  (14, 0.1875)  (23, 0.5625)  (24, 0.1875)  (25, 0.1875)  (26, 0.0625)
row 9: (11, 0.0625)  (12, 0.1875)  (13, 0.1875)  (14, 0.5625)  (15, 0.1875)  (16, 0.0625)  (17, 0.5625)  (18, 0.1875)
       (23, 0.1875)  (24, 0.5625)  (25, 0.0625)  (26, 0.1875)  (27, 0.5625)  (28, 0.1875)  (29, 0.1875)  (30, 0.0625)
row 10:(15, 0.0625)  (16, 0.1875)  (17, 0.1875)  (18, 0.5625)  (19, 0.1875)  (20, 0.0625)  (21, 0.5625)  (22, 0.1875)
       (27, 0.1875)  (28, 0.5625)  (29, 0.0625)  (30, 0.1875)  (31, 0.5625)  (32, 0.1875)  (33, 0.1875)  (34, 0.0625)
row 11:(19, 0.0625)  (20, 0.1875)  (21, 0.1875)  (22, 0.5625)  (31, 0.1875)  (32, 0.5625)  (33, 0.0625)  (34, 0.1875)
row 12:(23, 0.1875)  (24, 0.0625)  (25, 0.5625)  (26, 0.1875)
row 13:(23, 0.0625)  (24, 0.1875)  (25, 0.1875)  (26, 0.5625)  (27, 0.1875)  (28, 0.0625)  (29, 0.5625)  (30, 0.1875)
row 14:(27, 0.0625)  (28, 0.1875)  (29, 0.1875)  (30, 0.5625)  (31, 0.1875)  (32, 0.0625)  (33, 0.5625)  (34, 0.1875)
row 15:(31, 0.0625)  (32, 0.1875)  (33, 0.1875)  (34, 0.5625)
```

Figure 0.5: Multigrid operators for the quadratic case. Left: First coarsening step. Right: Second coarsening step

A two-dimensional problem and a three-dimensional problem are used. The experiment was repeated while the underlying tensor grid was refined for both the two- and three-dimensional problem.

The finest two-dimensional problem is presented in Figure 6. The underlying tensor grid has 32x32 elements. After refining 19782 elements are existent.

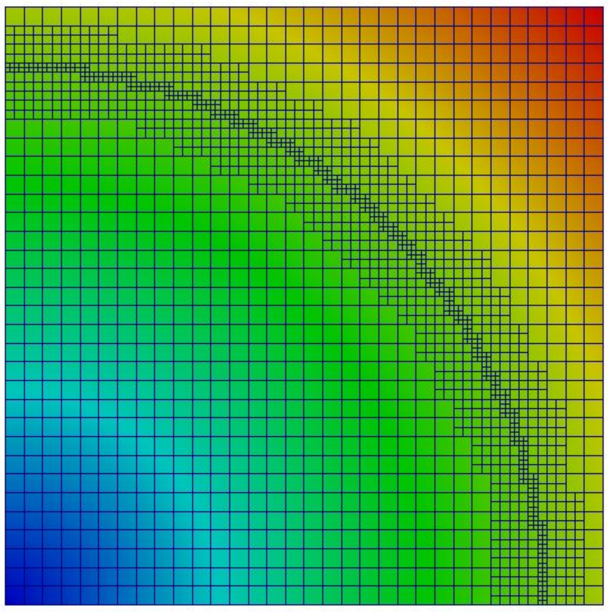


Figure 0.6: 2D-example

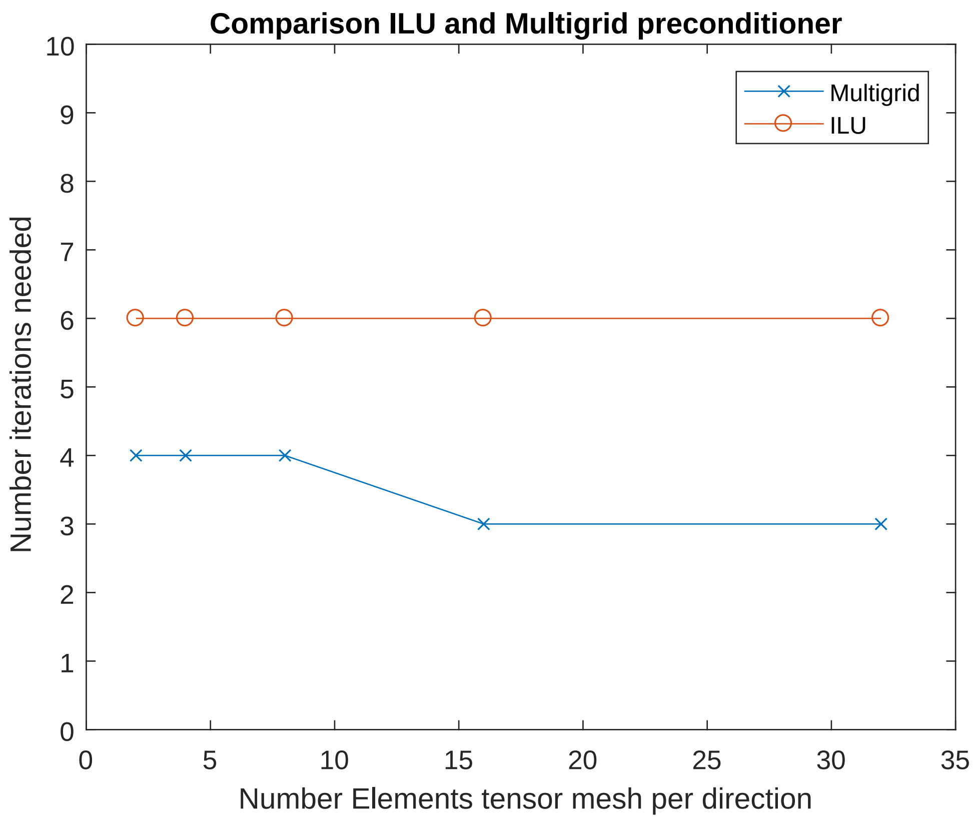


Figure 0.7: Comparison number of iterations needed for two dimensional problem

For the presented solver settings, the comparison of the multigrid preconditioner and the ILU shows for the solver with the multigrid preconditioner needs 3 to 4 iterations to solve the problem while the solver with the ILU preconditioner need 6 iterations. The number of iterations needed stay nearly constant while increasing the mesh size

The same analysis as for the two-dimensional problem was done for the three dimensional problem shown in Figure 8. In Figure 9 a plot is presented which compares the number of iterations needed to solve the three-dimensional problem with either an ILU or a multigrid preconditioner. Results are presented for different mesh refinements. For the presented solver settings, the comparison of the multigrid preconditioner and the ILU shows for the solver with the multigrid preconditioner needs 4 to 6 iterations to solve the problem while the solver with the ILU preconditioner need 6 to 7 iterations. The number of iterations needed stay nearly constant while increasing the mesh size.

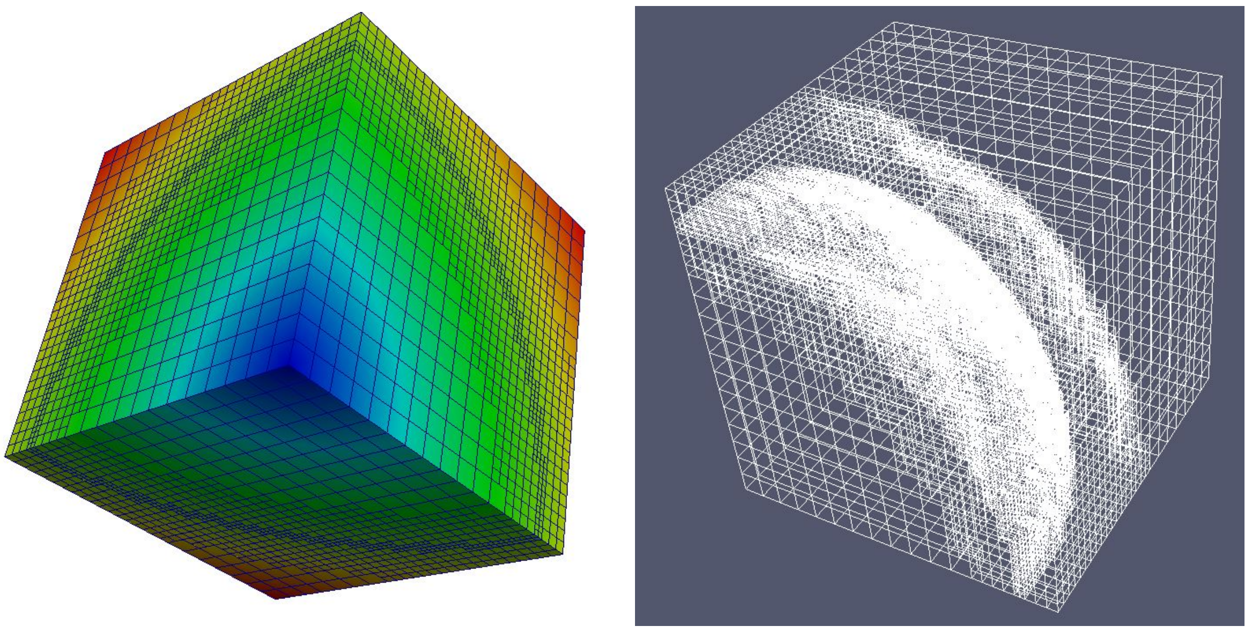


Figure 0.8: 3D-example

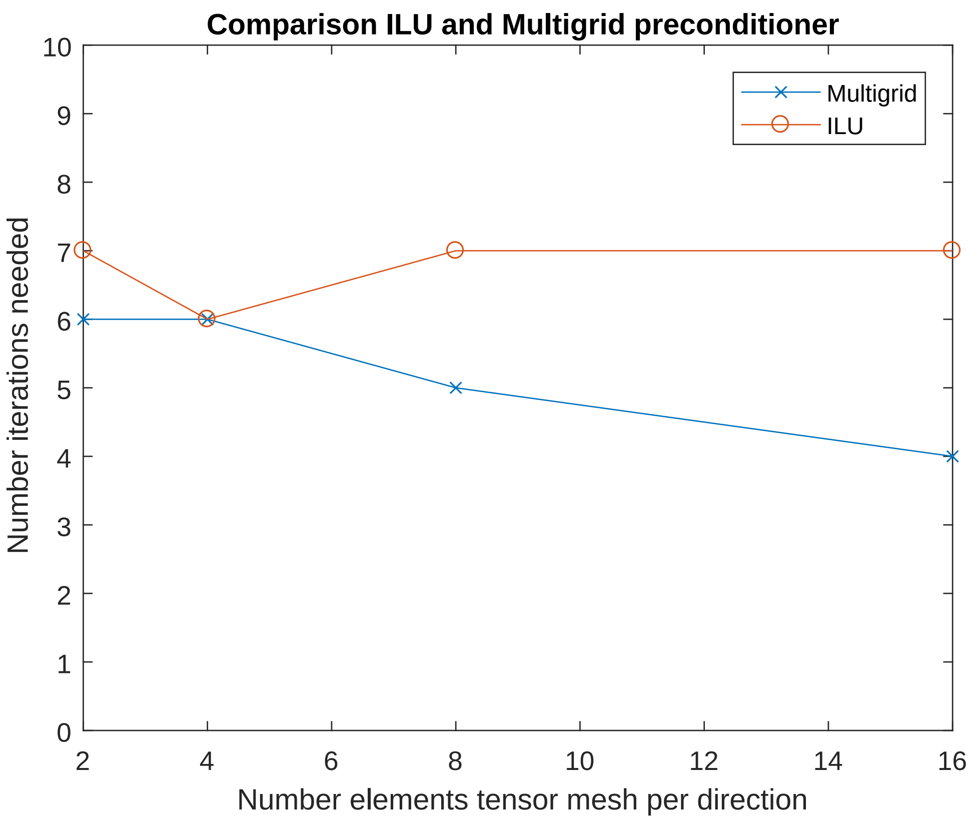


Figure 0.9: Comparison number of iterations needed for two dimensional problem