Journal of Mathematics in Industry
a SpringerOpen Journal

RESEARCH **Open Access**

Fast 3D solvers for interactive computational mechanics

Stefan Gavranovic[1,2], Zain Hassan[2,3], Lukas Failer[4] and Dirk Hartmann[2*]

*Correspondence:
hartmann.dirk@siemens.com
[2]Simulation and Test Solutions, Siemens Industry Software GmbH, Otto-Hahn-Ring 6, Munich, 81739, Germany
Full list of author information is available at the end of the article

Abstract
While interactive simulations have been mostly limited to Computer Graphics applications, new generations of Graphics Processing Units (GPUs) allow the realization of industrial-grade interactive 3D physics simulations. By combining an immersed boundary method with efficient GPU-based MINRES and CG solvers using a GPU-based geometric multigrid preconditioner, we demonstrate a fast industrial 3D computational mechanics solver. The various implementation aspects - specifically how they differ from similar concepts used in the Computer Graphics community - are discussed in detail. The proposed concept opens up new classes of industrial simulation applications allowing a democratization beyond today's expert users, from designer centric simulation to operational and service decisions based on 3D simulations. To support this, we provide various benchmark cases including a real-world study of a simulation-based service decision for a damaged gear-box mount.

Keywords: Immersed Boundary Method; Geometric Multigrid Preconditioner; MINRES; CG; GPU Computing; Interactive Simulation; Computational Mechanics

1 Introduction

Computer Aided Engineering (CAE) and simulation are a success story in industrial engineering. Today, hardly any product is produced without being simulated beforehand. Thereby, Computational Mechanics is one of the most important application fields [1]. Simulation is a strategic priority for many industrial corporations [2] with an ambition to extend its adoption among more users [3] (democratization of simulation) as well as along the life cycle [4] (Digital Twins). A main challenge in this endeavor is the usability and end-to-end simulation time [5]. Here, major aspects are not only the actual solver time but also the time required for geometry preparation and meshing of complex geometries.

At the same time, the Computer Graphics community has been for decades at the forefront of developing techniques aimed at accelerating animations and physics engines for video games and animations [6, 7], c.f. [8] for an example in the field of Computational Mechanics. These endeavors have given rise to innovative algorithms and methodologies, specifically exploiting Graphics Processing Units (GPUs), with high computational efficiency and real-time responsiveness without the need for complex meshing.

This paper has the objective of bridging the gap between Computer Graphics and Computational Mechanics, harnessing well-established concepts and practical solutions from the former and enhancing them with established Computational Mechanics techniques ensuring the required accuracy for industrial solutions. The ambition is to create simulation solutions that are exceptionally fast and performant while maintaining a level of accuracy that is comparable to classical Computational Mechanics approaches. By combining the strengths of these two distinct domains, we seek to provide a new paradigm for simulations that can be applied to a wide array of engineering domains demanding quick prediction but not being addressed by today's simulation technology. That is, democratizing simulation beyond today's expert users. These include applications from the use of simulation in early lifecycle phases, e.g., allowing designers to exploit interactive simulations, as well as in later stages, e.g., allowing simulation based optimization of operations and maintenance.

1.1 Contributions

In pursuit of the aforementioned research objective, this study presents several noteworthy contributions.

Immersed Boundary Method (IBM) Integration: To ensure an accurate representation of essential boundary conditions within a voxel-based discretization, we have integrated the IBM. The well-known Marching Cubes algorithm has been adapted for surface triangulation and numerical integration. Furthermore, to fully exploit the parallel processing capabilities of GPUs, the IBM has been effectively parallelized.

Fast Linear Algebra Solver Implementation: A Geometric Multigrid Method (GMG) has been adapted to work as a preconditioner to the Conjugate Gradient (CG) and the Minimum Residual (MINRES) solvers. The entire solver has been intricately parallelized to harness the computational power of GPUs using the CUDA [9] programming framework.

Robustness Analysis: Extensive convergence analysis has been conducted for both the Finite Element Method (FEM) and the linear algebra solver across various practical use cases. These comprehensive analyses provide empirical evidence of the method's robustness and its ability to consistently deliver accurate results in diverse scenarios.

Collectively, these contributions form a robust and efficient computational framework for Computational Mechanics applications tailored to address complex problems requiring accurate boundary representations and high-performance linear algebra solvers.

2 Related work

2.1 Immersed boundary method

In classical FEM, the finite element (FE) space is a subspace of the solution space. If this FE space approximates the solution space, then by construction the approximate solution converges to the true solution. However, a FE space that is not a subspace of the solution space can be advantageous in some cases. For example, an FE mesh that does not conform to the boundary of the domain can be chosen arbitrarily to simplify meshing or to assist the solution process. Various types of such non-conforming FEM have been presented and are referred to by different names, such as unfitted FEM, fictitious domain FEM, Finite Cell Method, Cartesian FEM, cutFEM, and Fixed Grid FEM. The common underlying idea in all these methods is that the problem domain is embedded in a background mesh, hence they are collectively referred to as IBM [10]. The Generalized-FEM [11] and the Extended-FEM [12] can also be loosely categorized into the IBM family.

In most implementations of IBM, Dirichlet boundary conditions have to be imposed weakly since the immersion of the problem domain in a non-conforming mesh makes strong enforcement infeasible. Along with the usual penalty and Lagrange methods of weak enforcement, the Nitsche method [13] is widely used e.g. [14, 15]. For an extensive discussion on Nitsche method, we refer to the work of Juntunen et al. [16].

IBM including the Nitsche method suffer from small cut instability. Burman et al. [17] have introduced boundary value correction for the stabilization of elements at the boundary which ensures the optimal condition number of the resulting linear system of equations. Related to this is the work of Hansbo et al. [18] in which a stabilization for control of jumps in derivatives in the vicinity of boundaries is added. The stabilization leads to what is called the ghost penalty. Moreover, high-order elements are used in the background mesh.

Parvizian et al. [19] have also used higher-order test functions in IBM for 2D elasticity in a method that they called the Finite Cell Method (FCM). The method has been further extended to 3D linear elasticity [20]. The core idea is the approximation of extended variables (in the extended domain) using high-order test functions and performing *hp*-refinement for optimal order convergence of smooth problems. The stiffness of fictitious material outside the boundaries is set to an extremely low value. For the integration of cut-cells, a uniform grid of sub-cells or a grid of non-uniform sub-cells based on a quad-tree approach is suggested.

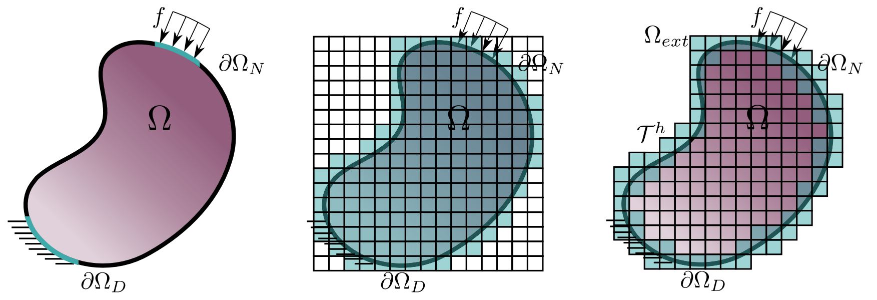
A simpler approach for integrating the cut-cells is developed in Fixed Grid FEM by García-Ruíz et al. for 2D [21] and 3D elasticity [22]. The FE domain is represented with a fixed Cartesian grid and the element stiffness matrix of a cut element is obtained as a factor of a reference element stiffness matrix. Daneshmand et al. [23] extended this method to include static and dynamic analysis of 2D and 3D elastic solids. The further introduced a volume integration scheme based on Gaussian quadrature and the characteristic function of the original domain embedded in the extended domain. Some other notable works in this field are FEM based on hierarchic h-refined Cartesian meshes [24] and immersed-FEM [25].

2.2 Fast solvers

In 2011, Dick et al. [8] introduced a GPU-based simulation of elastic bodies utilizing hexahedral meshes. Following this, the employment of tetrahedral meshes with efficient GPU data structures was demonstrated by Allard et al. [26] and further explored by Weber et al. [27]. This work was subsequently extended to incorporate higher-order elements as discussed in [28].

Large-scale efficient multigrid-based solvers for massively parallel architectures have been another focal point of research. Ljungkvist and Kronbichler have made significant contributions with development of a GPU-based matrix-free multigrid method tailored for solving 2D and 3D Poisson problems [29, 30]. This methodology was subsequently adapted for addressing linear elasticity problems on CPUs in [31] and has been integrated into the deal.II library [32]. More recently, these techniques have been further refined to support locally refined meshes [33], enhancing their applicability and performance.

Recently, Jomo et al. [34] took the advantage of the Finite Cell Method's hierarchic structure to build a multigrid preconditioned CG solver for solution of very large systems on a massively parallel distributed memory machines. However, the parallelization is not realized for more readily available GPUs.



(a) The problem domain and its boundaries.

(b) Discretized bounding box and all active elements (shaded blue).

(c) The FE mesh $\mathcal{T}^h$ defining the extended domain Ω_{ext}. Inside and cut elements are shaded differently.

Figure 1 The problem domain and its embedding in voxel mesh

3 Methods

3.1 Voxel-based immersed boundary method

3.1.1 Discretization

Unlike conforming FEM, in voxel-based IBM the geometry discretization requires a separate representation since the hexahedral mesh, also known as voxel grid, approximates the extended domain instead of the geometry. The geometry is represented using a signed distance function discretized over the background mesh using the highly performant OpenVDB library [35].

Let Ω be the problem domain with Dirichlet and Neumann boundary conditions applied at $\partial\Omega_D$ and $\partial\Omega_N$ respectively (Fig. 1a). To use a geometric multigrid algorithm a hierarchy of meshes needs to be constructed. The FE mesh, i.e. the finest in the grid hierarchy, is generated by meshing the bounding box of the geometry (Fig. 1b) and removing the elements that do not overlap with the geometry. The extend of the resulting mesh $\mathcal{T}^h$ defines the extended domain Ω_{ext} (Fig. 1c), in which the geometry Ω is embedded. The geometry is represented by the signed distance function ϕ residing on the vertices of the mesh $\mathcal{T}^h$. The coarser grids are created using a modified form of vertex-centered coarsening. Thereby, the coarse elements at the edges may have nodes that do not correspond to any node in the finer grid. Here, the coherency between different levels of multigrid is kept by ensuring that every element in a fine grid has a corresponding element in the next coarser grid.

Thus, a grid $\{\mathcal{T}^{ih}\}$ in the multigrid hierarchy consists of elements e^{ih} having the side length ih, where $i = 2^0, 2^1, 2^2, \dots, 2^m$ and h is the side length of an element e^h in the finest grid $\{\mathcal{T}^h\}$. Elements in any grid $\{\mathcal{T}^{ih}\}$ can be classified into two sets of inside elements $\{e_I^{ih}\}$ and cut elements $\{e_C^{ih}\}$ defined as:

$$\{e_I^{ih}\} = \{e \in \{\mathcal{T}^{ih}\} | e \cap \Omega \neq \emptyset \text{ and } e \cap \partial\Omega = \emptyset\}, \tag{1}$$

$$\{e_C^{ih}\} = \{e \in \{\mathcal{T}^{ih}\} | e \cap \partial\Omega \neq \emptyset\}, \tag{2}$$

where $\partial\Omega$ represents the geometric boundary, and it holds $\{e_I^{ih}\} \cap \{e_C^{ih}\} = \emptyset$ as well as $\{e_I^{ih}\} \cup \{e_C^{ih}\} = \Omega_{\text{ext}}$.

3.1.2 Discretized linear elasticity equations

Following a classical computational approach for linear elasticity [36], weak form as obtained by multiplying the strong form by virtual displacements $\delta\boldsymbol{u}$ and integrating over the whole physical domain can be written as:

$$\int_\Omega \boldsymbol{\sigma}(\delta\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u})\,\mathrm{d}x = \int_\Omega \delta\boldsymbol{u}\cdot\boldsymbol{b}\,\mathrm{d}x + \int_{\partial\Omega_N} \delta\boldsymbol{u}\cdot\hat{\boldsymbol{t}}\,\mathrm{d}x + \int_{\partial\Omega_D} \delta\boldsymbol{u}\cdot(\boldsymbol{\sigma}(\boldsymbol{u})\cdot\boldsymbol{n})\,\mathrm{d}x, \tag{3}$$

where $\boldsymbol{u}$ denotes displacement, $\boldsymbol{\sigma}(\boldsymbol{u})$ is the Cauchy stress, $\boldsymbol{\varepsilon}(\boldsymbol{u})$ is the linear strain, $\boldsymbol{n}$ represents surface normal vector, $\boldsymbol{b}$ is the body force, $\hat{\boldsymbol{t}}$ is the traction vector acting on the surface $\partial\Omega_N$, and $\partial\Omega_D$ is the boundary where Dirichlet boundary condition is imposed. The stress $\boldsymbol{\sigma}(\boldsymbol{u})$ and strain $\boldsymbol{\varepsilon}(\boldsymbol{u})$ are related via a material law, which in our case is the simple linear elastic, i.e. $\boldsymbol{\sigma}(\boldsymbol{u}) = \boldsymbol{C} : \boldsymbol{\varepsilon}(\boldsymbol{u})$, where $\boldsymbol{C}$ is the elasticity tensor relating strain to stress. The last term of the right hand side in (3) vanishes in boundary conforming FEM by choosing the FE space of virtual displacement such that $\delta\boldsymbol{u} = 0$ on $\delta\Omega_D$, while the FE space of displacements ensures $\boldsymbol{u} = \hat{\boldsymbol{u}}$ on $\delta\Omega_D$ [37], where $\hat{\boldsymbol{u}}$ is the prescribed displacement.

In IBM, the enforcement of Dirichlet boundary conditions by prescribing the displacement on nodes which coincide with the Dirichlet boundary is not feasible. An alternate way is to modify the weak formulation such that Dirichlet boundary conditions are satisfied weakly. Using the symmetric Nitsche method [37], the modified weak formulation is described as:

$$\begin{aligned}\int_\Omega \boldsymbol{\sigma}(\delta\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{u})\,\mathrm{d}x + \int_{\partial\Omega_D} \delta\boldsymbol{u}\cdot\beta\cdot\boldsymbol{u}\,\mathrm{d}x - \int_{\partial\Omega_D} (\boldsymbol{\sigma}(\delta\boldsymbol{u})\cdot\boldsymbol{n})\cdot\boldsymbol{u}\,\mathrm{d}x - \int_{\partial\Omega_D} \delta\boldsymbol{u}\cdot(\boldsymbol{\sigma}(\boldsymbol{u})\cdot\boldsymbol{n})\,\mathrm{d}x \\ = \int_\Omega \delta\boldsymbol{u}\cdot\boldsymbol{b}\,\mathrm{d}x + \int_{\partial\Omega_N} \delta\boldsymbol{u}\cdot\hat{\boldsymbol{t}}\,\mathrm{d}x + \int_{\partial\Omega_D} \delta\boldsymbol{u}\cdot\beta\cdot\hat{\boldsymbol{u}}\,\mathrm{d}x - \int_{\partial\Omega_D} (\boldsymbol{\sigma}(\delta\boldsymbol{u})\cdot\boldsymbol{n})\cdot\hat{\boldsymbol{u}}\,\mathrm{d}x,\end{aligned} \tag{4}$$

where β is stabilization parameter, $\boldsymbol{\sigma}(\delta\boldsymbol{u}) = \boldsymbol{C} : \boldsymbol{\varepsilon}(\delta\boldsymbol{u})$, and the additional terms compared to Equation (3) cancel each other.

The volume integrals involved in the computation of the element stiffness matrix of inside elements $\{e_I^h\}$ is evaluated using the standard second-order Gaussian quadrature. For integrals of the cut elements $\{e_C^h\}$ volume integrals are computed using the Marching Cubes algorithm [38]. Likewise, the surface integrals in Equation (4) are computed using the same algorithm, which also conveniently provides the surface normal vectors n at the surface integration points (see Sect. 3.1.3). After FE discretization and integration, the system of equations to be solved is of the form:

$$(\mathbf{K} + \beta\mathbf{M} - \mathbf{G} - \mathbf{G}^T)\mathbf{u} = \mathbf{f} + \beta\mathbf{m} - \mathbf{g}, \tag{5}$$

$$\hat{\mathbf{K}}\mathbf{u} = \hat{\mathbf{f}}. \tag{6}$$

Here the matrix $\mathbf{K}$ represents global stiffness matrix $\mathbf{M}$, which is the same as the penalty matrix, and stabilizes the formulation while $-\mathbf{G} - \mathbf{G}^T$ ensures the symmetry as described in [37]. However, unlike the penalty method, the stabilization parameter β does not need to be very large to ensure good satisfaction of the constraint. Here, β is chosen as:

$$\beta = 10^2 \cdot \frac{E}{h}, \tag{7}$$

where E is the Young's modulus, and h is side length of the element. For more details on the effect of the Nitsche stabilization parameter the reader is referred to [37].

3.1.3 Marching cubes-based cut element treatment

Surface integrals As described in Sect. 3.1.1, the geometry in this work is represented using a signed distance function stored at the nodes of the hexahedral elements. Using the Marching Cubes algorithm for triangulating the iso-surface is quite convenient in this case. The Marching Cubes algorithm, widely used in the Computer Graphics community, works by dividing the 3D scalar field into a grid of "cubes", and then determining how the iso-surface intersects each cube. The surface intersection is identified using the vertex values of the cube to look up the particular intersection case in a table.[1] Using this information, it computes the surface triangles, and the corresponding surface normal vectors, before "marching" on to the next cube. The reader is referred to the original paper of Lorensen and Cline [38], and the implementation of Bourke [39] for more details.

The resulting triangles are assigned with integration points using the standard second-order Gaussian quadrature for triangles. The normal vectors as provided by the Marching Cubes algorithm are useful for the integration of the Nitsche term in weak enforcement of boundary conditions as seen in Equation (4) and for computation of the volume fraction of cut elements.

Volume integrals The integral involved in the computation of the element stiffness matrix of inside elements $\{e_I^h\}$ is evaluated using the standard second-order Gaussian quadrature. Thereby uniform hexahedral mesh elements allow to compute the element stiffness matrix only once and it is denoted by $\mathbf{K}_0$. For the cut elements, belonging to the set $\{e_C^h\}$, the volume integrand is however discontinuous since these elements lie on the boundary of the geometry. The discontinuous integral can be written using a characteristic function χ as,

$$\mathbf{K}_c = \int_{\Omega^e} \mathbf{B}^T \chi(\boldsymbol{x}) \mathbf{C} \mathbf{B} \, \mathrm{dx}, \tag{8}$$

where $\mathbf{B}$ is the linear strain matrix, $\mathbf{C}$ is the elasticity matrix, and the characteristic function χ is defined as

$$\chi(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \Omega, \\ \alpha & \text{if } \boldsymbol{x} \in \Omega_{\text{ext}} \setminus \Omega. \end{cases} \tag{9}$$

Here, $0 < \alpha \ll 1$ is a very small constant that represents the relative stiffness of the material in the empty portion of the cut element. Its value in this work is chosen to be 10^{-3}. The smaller α the more accurate will be the model, however it may lead to very high condition number and numerical instability. Setting α close to 1 will make the problem well-behaved albeit highly inaccurate.

[1] Given that there are eight vertices of a cube with two states each (in or out), there are $2^8 = 256$ intersection cases. However, using the different symmetries, they are reduced to only 15 unique cases.

In this work, we assume that the discontinuous bi-material distribution in a cut element can be approximated by a constant distribution of a fictitious material as done in [21, 22]:

$$\mathbf{K}_c = \left(\nu_f + \left(1 - \nu_f\right)\alpha\right) \int_{\Omega^e} \mathbf{B}^T \mathbf{C} \mathbf{B} \,\mathrm{dx} = \left(\nu_f + \left(1 - \nu_f\right)\alpha\right) \mathbf{K}_0, \tag{10}$$

where ν_f is the volume fraction of the cut element, i.e. $\mathbf{K}_c$ is simply obtained by scaling the stiffness matrix $\mathbf{K}_0$.

In Equation (10), the volume fraction can be computed as:

$$\nu_f = \frac{\int_{\Omega^{e^+}} 1 \,\mathrm{dx}}{\int_{\Omega^e} 1 \,\mathrm{dx}}, \tag{11}$$

where Ω^{e^+} represents the portion of the element e that lies inside Ω. The integral in the nominator is still an integral over a discontinuous volume. Following the work of Wang [40], the volume enclosed by the iso-surface is computed using the divergence theorem, which converts the discontinuous volume integration into the surface integration:

$$\int_{\Omega^{e^+}} 1 \,\mathrm{dx} = \frac{1}{3} \int_{\partial\Omega^{e^+}} \boldsymbol{n} \cdot \boldsymbol{x} \,\mathrm{dx} = \frac{1}{3} \sum_{i \in T} \int_{\partial\Omega_i} \boldsymbol{n} \cdot \boldsymbol{x} \,\mathrm{dx}, \tag{12}$$

where T is a set of surface triangles that include the iso-surface triangles and the triangles on the cube surface to form an enclosed volume (enclosing the intersection of the geometry and the element), $\boldsymbol{n}$ is the surface normal vector and $\boldsymbol{x}$ is the location of the integration point (c.f. Figure 2).

The advantage of this method is that the matrix assembly does not require element wise integration as it uses the precomputed stiffness matrix $\mathbf{K}_0$ and scales it accordingly, hence requiring minimal computations. Having a small memory footprint, stiffness matrix $\mathbf{K}_0$ can be stored in constant GPU memory, which allows extremely fast memory access. However, the simplification of just scaling the element matrix does not take into account the geometry of the boundary and the orientation of the cut and distributes the stiffness equally to all nodes of the element. This implies that vertically, horizontally, and diagonally cut elements have the same element stiffness matrix if the volume fraction ν_f is the same.

In Sect. 4 we compare the scaled boundary element approach above with unscaled boundary element approach as adopted typically in the Computer Graphics community like in [8]. The latter applies no scaling factor to the boundary elements, i.e., one always chooses $\nu_f = 1$ for intersected elements. In the subsequent sections, we refer to the approach where $\nu_f = 1$ as the unscaled method and the approach where ν_f is defined according to Equation (11) as the scaled method.

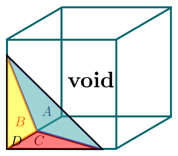


Figure 2 The surface triangle *A* (blue) and three face triangles *B* (Yellow), *C* (Red), and *D* (Black) that form volume (enclosing the intersection of the geometry and element) for computation of volume fraction

3.2 GPU-based iterative solvers

Although discretization is defined on a per-element basis, to effectively leverage the massive parallelism of GPU streaming multiprocessors for iterative solvers, transitioning to a vertex-based data distribution is more advantageous. Consequently, all operations are redefined using vertex-specific stencils to optimize performance and fully exploit the computational capabilities of GPUs. Thanks to the structure of the mesh, as mentioned in the Sect. 3.1.3, the element stiffness matrix $\mathbf{K}_0$ is precomputed and stored in the fast constant GPU memory. The matrix $\mathbf{K}_0$ and the volume fraction (11) are then used to construct vertex stencils for all vertices in the mesh $\mathcal{T}^h$. These stencils are subsequently stored in the GPU memory as per [8, 41].

Matrices $\mathbf{M}$, $\mathbf{G}$, and $\mathbf{G}^T$ in Equation (5), arising from the cut elements $\{e^h_C\}$ are contingent on the configuration of the cuts and thus cannot be precomputed. Instead, these matrices are computed on the CPU[2] for each individual cut element in the set $\{e^h_C\}$, their sum is then used to construct corresponding stencils $\mathcal{Q}_v$ for vertices v in $\{e^h_C\}$. These stencils are then copied to the GPU memory, where they are summed in order to obtain final vertex stencil $\mathcal{K}_v = \mathcal{Q}_v + \mathcal{S}_v$ that corresponds to the left hand side of Equation (5).

We implement the standard MINRES [42] and CG [43] solvers (e.g. see [44]) using CUDA [9] to be used along with our GMG preconditioner $\mathcal{P}_{\mathrm{MG}}$ as defined in [41] and detailed below. We will refer to these as MINRES-$\mathcal{P}_{\mathrm{MG}}$ and CG-$\mathcal{P}_{\mathrm{MG}}$ respectively. Compared to GMG implementation [41] as a primary solver, our MINRES-$\mathcal{P}_{\mathrm{MG}}$ or CG-$\mathcal{P}_{\mathrm{MG}}$ implementation requires only one additional matrix-vector multiplication per iteration, which does not present large computational overhead. Additionally, there is a minor increase in memory usage due to the need to store intermediate arrays. The outcomes of these improvements, aiming to help robustness and computational efficiency, are detailed in Sect. 4.2.

GMG solvers stand out as highly efficient tools for solving partial differential equations, achieving an optimal computational complexity of $\mathcal{O}(N)$ where N is number of degrees of freedom of a system. The core concept behind the GMG approach is the utilization of a series of nested grids, each with varying resolution. This multi-tiered strategy smoothes out errors at their respective scales, allowing for a comprehensive error reduction across the full spectrum of frequencies [45, 46]. To leverage the structured discretization scheme and maximize GPU performance, we employ a GMG-based preconditioner operating on vertex stencils $\mathcal{K}_v$.

The majority of the multigrid implementation is detailed in the works [8, 41], covering its implementation aspects. The multigrid solver employs a standard V-cycle with two pre-smoothing and two post-smoothing steps, using the multicolor Successive Over-Relaxation (SOR) smoother. The coarsest grid size in the multigrid hierarchy is predetermined using heuristics to ensure adequate geometric detail, typically comprising 10^4 nodes in our studies. At the coarsest level, a direct solver, oneMKL PARDISO [47], is utilized on the CPU. Due to the relatively small problem size on the coarsest grid, memory transfers between the CPU and GPU are negligible in the overall execution time. The primary components of the multigrid algorithm, namely smoothing, residual computation, restriction, and prolongation, are implemented as separate CUDA kernels.

[2]Due to the different cut configurations that produce numerous branching conditions, implementing this on a GPU would be challenging. Therefore, cut cell computation is handled on the CPU, which is better suited for such tasks.

4 Results

In the following, we illustrate the effectiveness of the proposed approach through several industrial examples. Our analysis concentrates on the h-convergence of the solver and the convergence of the multigrid preconditioned Conjugate Gradient (CG-$\mathcal{P}_{\text{MG}}$) and Minimum Residual (MINRES-$\mathcal{P}_{\text{MG}}$) solvers on increasingly complex geometries. Finally, we present a practical industrial example to demonstrate how this fast solver can be effectively used.

All results were obtained on a workstation equipped with an Intel i9 13900KF CPU, 32 GB of DDR5 RAM, and an NVIDIA RTX 4090 GPU with the NVIDIA 555.x graphics driver. Our codebase has been built using GCC 11.4 and NVCC 12.3 compilers.

4.1 *h*-convergence analysis

In our study, we perform h-convergence analysis on two distinct examples: a bracket and a perforated plate. Reference values, denoted with *, for convergence comparisons are obtained using the Simcenter 3D (SC3D) structural solver [48] with tetrahedral elements having 4 degrees of freedom (C3D4). The Simcenter 3D solver, with its mesh conforming capability, offers better geometry discretization compared to voxel-based methods but is computationally more expensive. To simplify the comparison of mesh conforming and non-conforming results, we examine the error in several key quantities of physical interest, including:

- relative error in the mean displacement value $\|\mathbf{u}\|_2$:
  $$\varepsilon_u = \frac{|\|\mathbf{u}\|_2 - \|\mathbf{u}^*\|_2|}{\|\mathbf{u}^*\|_2};$$
- relative error in the maximal displacement value $\|\mathbf{u}\|_\infty$:
  $$\varepsilon_{u_{\max}} = \frac{|\|\mathbf{u}\|_\infty - \|\mathbf{u}^*\|_\infty|}{\|\mathbf{u}^*\|_\infty};$$
- relative error in the maximal Von Mises stress value $\|\boldsymbol{\sigma}_{\text{VM}}\|_\infty$:
  $$\varepsilon_{\sigma_{\max}} = \frac{\left|\|\boldsymbol{\sigma}_{\text{VM}}\|_\infty - \|\boldsymbol{\sigma}^*_{\text{VM}}\|_\infty\right|}{\|\boldsymbol{\sigma}^*_{\text{VM}}\|_\infty};$$
- relative error in the external work (Energy) value $W_{\text{ext}}(\mathbf{u})$, which is defined and computed in discrete form as:
  $$\varepsilon_{W_{\text{ext}}} = \frac{|W_{\text{ext}}(\mathbf{u}) - W_{\text{ext}}(\mathbf{u}^*)|}{W_{\text{ext}}(\mathbf{u}^*)} = \frac{\left|\mathbf{f}^T\mathbf{u} - \mathbf{f}^{*T}\mathbf{u}^*\right|}{\mathbf{f}^{*T}\mathbf{u}^*}.$$

Furthermore, we use the number of degrees of freedom rather than the discretization size in the following convergence analysis as it provides more information about problem scale. While these relationships can be straightforwardly established in the case of a voxel grid, they are not applicable in the context of the Simcenter structural solver. The latter does not work with an hierarchical refinement, but rather requires re-meshing completely when the size of the mesh is changed as in most state of the art industrial solvers. This also explains the "non-optimal" convergence rates observed in the following studies for the Simcenter structural solver.

Simple bracket example In this study, we examine a simple steel bracket (Fig. 3), with material properties $E = 2.069 \cdot 10^5$ MPa, and $\nu = 0.288$. The bracket is fixed at one end and subjected to a 5000 N downwards force at the other end. We observe convergence in the mean displacement value in both approaches: unscaled and scaled boundary presented in Fig. 4. When employing the scaled boundary approach, we achieve accuracy comparable to the mesh conforming Simcenter 3D reference solution. This similarity in accuracy and

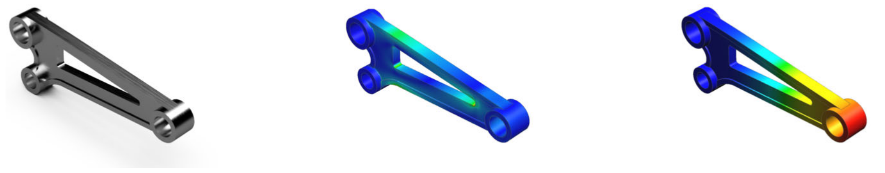


Figure 3 Simple bracket geometry, stress field and displacement field respectively

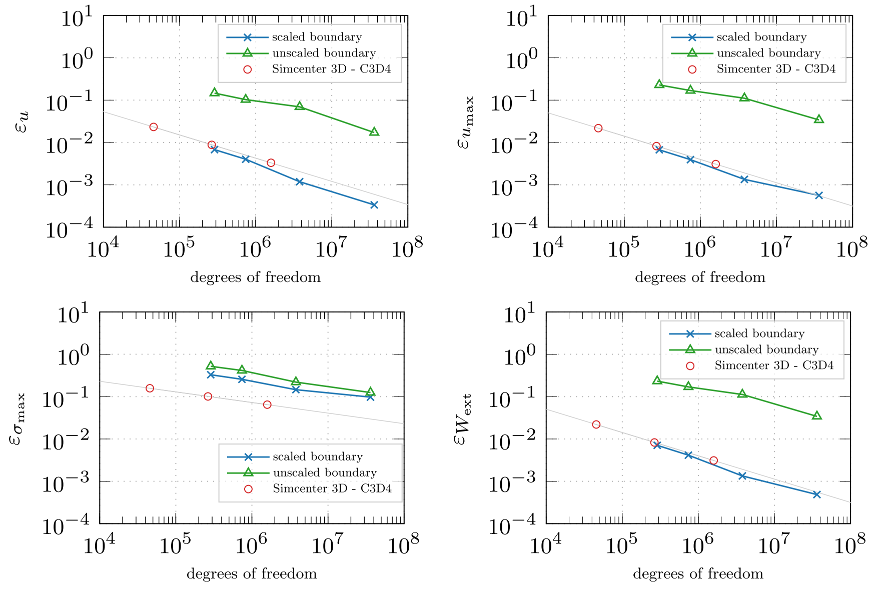


Figure 4 Bracket *h*-convergence plots comparing the error in various quantities of interest between the presented non-conforming method and the results from a classic conforming mesh method in Simcenter 3D using C3D4 elements

convergence rate also extends to the relative error in maximum displacement $\varepsilon_{u_{\max}}$ and external work $\varepsilon_{W_{\text{ext}}}$.

However, the relative error in maximum stress $\varepsilon_{\sigma_{\max}}$ is consistently higher than in the reference solution for both approaches. This is primarily due to the challenges in accurately recovering stress from the voxel space to the surface of the geometry. The most significant error contributions are found in elements with minor cuts, where stress values cannot be reliably calculated.

Perforated plate example In our analysis of the perforated steel plate example, with material properties $E = 2.069 \cdot 10^5$ MPa, and $\nu = 0.288$, the plate is fixed at one end and subjected to a tensile load of 5000 N at the other end (Fig. 5), convergence is assessed over grids with varying degrees of freedom (Fig. 6).

Similar to the results of the bracket example (Fig. 4), in Fig. 6 we observe convergence for both unscaled and scaled boundary approaches as defined in Sect. 3.1.3. However, the relative error in maximum stress $\varepsilon_{\sigma_{\max}}$ is consistently higher than the reference, due to reasons previously discussed. Interestingly, for the unscaled approach, the error in maximum stress $\varepsilon_{\sigma_{\max}}$ occasionally drops below that of the scaled approach. This can be due to the fact that for certain element size h, the mesh is better aligned to geometry produc-

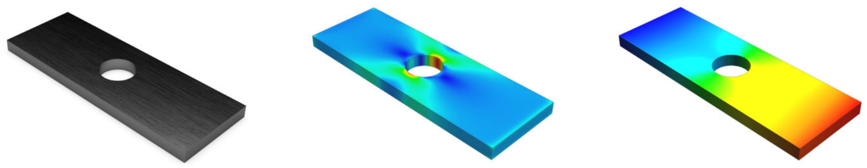


Figure 5 Perforated plate geometry, stress field and displacement field respectively

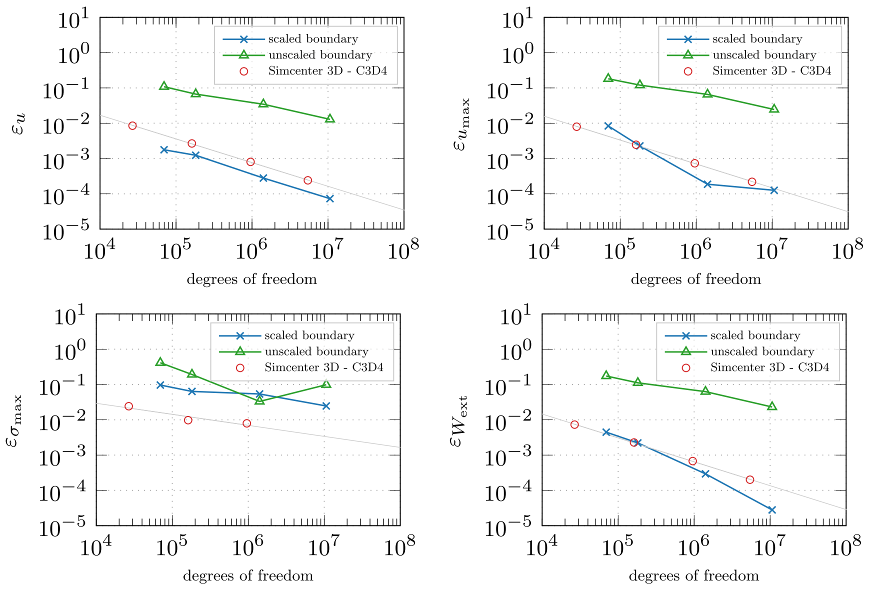


Figure 6 Perforated plate *h*-convergence plots comparing the error in various quantities of interest between the presented non-conforming method and the results from a classic conforming mesh method in Simcenter 3D using C3D4 elements

ing better stress estimates. It is important to note here the overall convergence behavior, which shows that the usage of boundary scaling consistently results in lower errors in the maximal stress value.

4.2 Iterative solver analysis

In this section, we present a convergence analysis of implemented iterative solvers. We compare the accuracy of the GMG preconditioned Conjugate Gradient (CG-$\mathcal{P}_{\text{MG}}$) and Minimum Residual (MINRES-$\mathcal{P}_{\text{MG}}$) solvers. In addition to the simple geometries above, we consider two complex geometries from typical industrial applications, an airplane landing gear and a car rim (depicted in the corresponding plots). Thereby, our analysis focuses on two types of errors.

On the one hand, we compute the relative residual $\mathbf{r}_k$ at each iteration k and define the following error analyzing the residual in reference to the value in the first step:

$$\varepsilon_{\text{solver}}(\mathbf{r}) = \frac{||\mathbf{r}_k||_M}{||\mathbf{r}_0||_M}, \tag{13}$$

where M represents the norm type. For the CG-$\mathcal{P}_{\text{MG}}$ solver the standard l^2-Norm is used and for the MG-$\mathcal{P}_{\text{MG}}$ solver the norm is defined via the preconditioner as described in [44]. As we always start the iteration with zero initial guess, the residual $||\mathbf{r}_0||_M$ is equivalent to the norm of the right hand side $||\mathbf{f}||_M$. On the other hand, we compute the relative error of the solution $\mathbf{u}_k$ at iteration k compared to an exact discrete solution $\mathbf{u}^*$ of the linear system which was computed using a direct solver:

$$\varepsilon^*_{\text{solver}}(\mathbf{u}) = \frac{||\mathbf{u}_k - \mathbf{u}^*||_2}{||\mathbf{u}^*||_2}. \tag{14}$$

The error $\varepsilon^*_{\text{solver}}$ enables us to monitor the evolution of the "true" error throughout the iterations. The iterative solver should be ideally stopped if the relative error in the solution $\varepsilon^*_{\text{solver}}(\mathbf{u})$ falls below a specific threshold to avoid unnecessary high compute times. However, the relative error in the solution $\varepsilon^*_{\text{solver}}(\mathbf{u})$ is usually not known a priori. It thus cannot serve as a stopping criterion for either solver. Instead, at each iteration, we compute the relative residual error $\varepsilon_{\text{solver}}(\mathbf{r})$, which is then employed as the stopping criterion. When comparing $\varepsilon_{\text{solver}}(\mathbf{r})$ for both solvers, as shown in Figs. 7 and 8, it is evident that the residual error estimate of CG-$\mathcal{P}_{\text{MG}}$ is more conservative than that of MINRES-$\mathcal{P}_{\text{MG}}$. Consequently, to achieve the target accuracy in terms of the relative residual, CG-$\mathcal{P}_{\text{MG}}$ necessitates a greater number of iterations. In the context of democratizing simulation

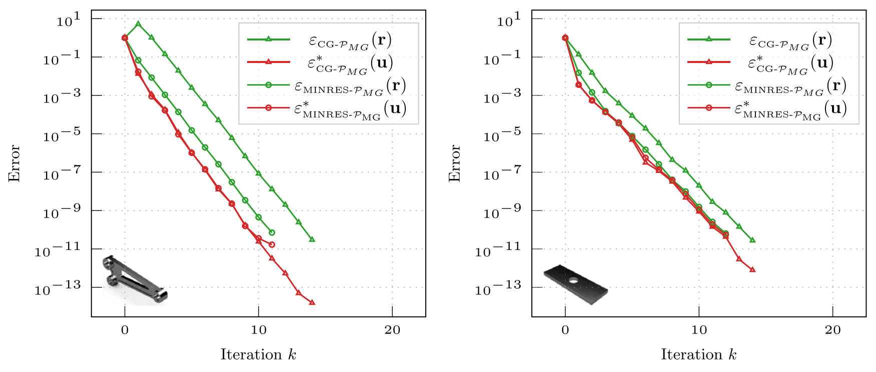


Figure 7 Iterative solver convergence rates for bracket and perforated plate examples

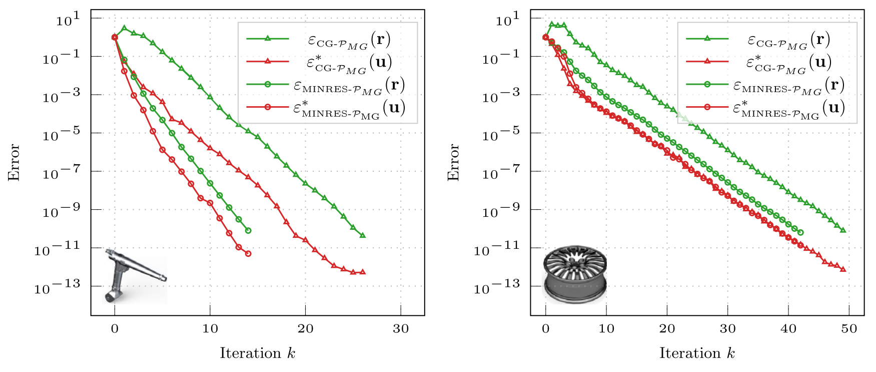


Figure 8 Iterative solver convergence rates for landing gear and rim examples

tools, choosing right parameters for stopping criterion presents a significant challenge to non-expert users. Tolerances need to be predetermined to allow any user to successfully run the simulations.

Furthermore, it is noteworthy that both the MINRES-$\mathcal{P}_{\text{MG}}$ and CG-$\mathcal{P}_{\text{MG}}$ solvers exhibit nearly identical convergence rates with respect to the exact error as defined in Equation (14).

4.3 Performance tests

For these examples, the stopping criteria for both CG-$\mathcal{P}_{\text{MG}}$ and the MINRES-$\mathcal{P}_{\text{MG}}$ solver was $\varepsilon_{\text{solver}}(\mathbf{r}) \leq 10^{-5}$. This threshold was heuristically determined, as it generally suffices for most industrial use cases.

In both examples, as illustrated in Fig. 9, we observe linear scaling with respect to an increase in the number of degrees of freedom. The maximal problem sizes solved on the RTX 4090 GPU with 24 GB of memory were $1.7 \cdot 10^7$ degrees of freedom in 1.2 seconds for the plate example and $1.5 \cdot 10^7$ degrees of freedom in 1.1 seconds for the bracket example. It can be observed that for more geometrically complex examples, the MINRES-$\mathcal{P}_{\text{MG}}$ trend line lies below that of CG-$\mathcal{P}_{\text{MG}}$, indicating better performance in practice. This can be attributed to the fact that CG-$\mathcal{P}_{\text{MG}}$ requires more iterations to meet the previously mentioned stopping criterion. Conversely, for simpler geometries, as shown in Fig. 7, when the relative residual errors are comparable for both CG-$\mathcal{P}_{\text{MG}}$ and MINRES-$\mathcal{P}_{\text{MG}}$, CG-$\mathcal{P}_{\text{MG}}$ performs slightly better due to its lower number of compute operations per iteration. This analysis indicates that for an intermediate number of degrees of freedom, the MINRES-$\mathcal{P}_{\text{MG}}$ solver can be well-suited for interactive 3D Digital Twin simulations.

4.4 Interactive computational mechanics

Fast 3D Computational Mechanics solvers open up numerous possibilities for interactive feedback loops - from designer centric simulation applications to simulation-based decision support in operation and service. In the following, we investigate the potential for an operation and service centric use case. For those types of use cases many existing industrial interactive simulation applications leverage model order reduction technologies [49]. However, their application is often restricted to scenarios that can be precisely parameterized. The fast 3D solver we introduce in this paper is considerably more adaptable. It allows for example to identify the implications of defects in mechanical structures, e.g., caused by manufacturing errors or operational damage.

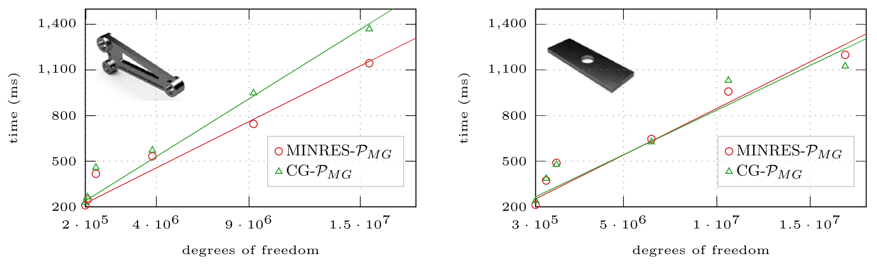
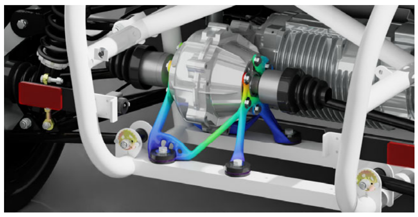
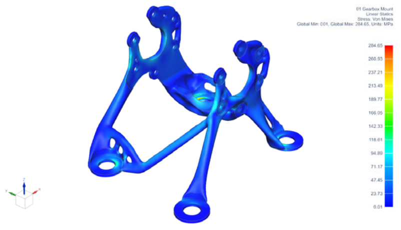
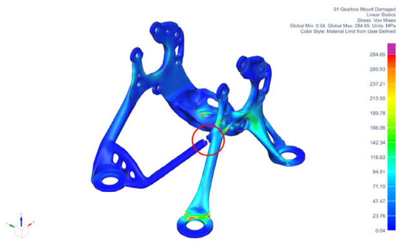


Figure 9 MINRES-$\mathcal{P}_{MG}$ and CG-$\mathcal{P}_{MG}$ performance scaling with respect to number of degrees of freedom for bracket and perforated plate examples. Trend line in red corresponds to the MINRES-$\mathcal{P}_{MG}$, and in green to the CG-$\mathcal{P}_{MG}$ solver


(a) Deformation of the part without defect


(b) Stress distribution in the part without defect with a maximum Von Mises stress of 283.65 MPa.


(c) Stress distribution in the part with defect with a maximum Von Mises stress of 668.04 MPa.

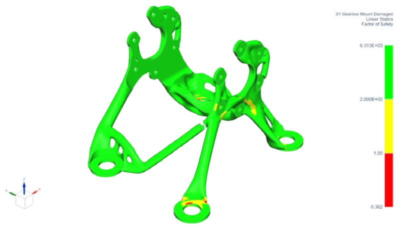

(d) Factor of safety in the part with defect.

Figure 10 Comparative analysis of stress distribution in gearbox mount example: Evaluating the impact of defects on mechanical integrity and factor of safety

Table 1 Gearbox mount simulation time

# elements	#DoFs	time (s)
$3.3 \cdot 10^5$	$1.2 \cdot 10^6$	3.54
$1.2 \cdot 10^6$	$4.2 \cdot 10^6$	14.30

Let's consider a damaged gearbox mount of an electric vehicle as a real-life scenario. With no spare parts available and the vehicle being essential, a rapid assessment is crucial to determine if the car can still be operated safely. Our approach begins with a 3D scan of the damaged part, creating an STL model. We then run a simulation using our solver, quickly assessing the resulting stresses and the extent of the damage as shown in Fig. 10. In this instance, moderate fidelity simulation takes approximately 3 seconds to complete (Table 1) and reports a maximum stress that is more than double the norm, suggesting that operating the car would be unsafe.

However, the technology itself allows for many more opportunities. For example, designers can perform interactive simulations early in the design phase - tweaking and refining designs in real-time to achieve an optimal performance design. In fact, the optimization of the gearbox mount's topology was achieved using this same technology [41].

5 Conclusion

This work has presented an interactive 3D Computational Mechanics solver which scales well up to a high number of degrees of freedom. By exploiting GPU computing, this solver is capable of handling tens of millions of degrees of freedom within seconds.

We addressed key challenges associated with the IBM when used with piecewise continuous C^0 basis functions, such as inadequate surface and volume approximation, and difficulties in stress recovery. A detailed analysis of the h-convergence of our method was conducted, focusing on multiple quantities of interest. These were compared against a well-established industrial mesh conforming solver [48]. Our findings indicate that the scaled boundary element approach not only outperforms the unscaled version - used frequently in the Computer Graphics community - but also matches the convergence order of an industrial state-of-the-art tetrahedral mesh conforming solver. We also implemented and evaluated various iterative solvers, concluding that the multigrid preconditioned MINRES–$\mathcal{P}_{\mathrm{MG}}$ solver offers better accuracy and the fastest convergence in relation to the established stopping criteria.

Finally, through a real industrial example, we demonstrated the practical applicability of our approach. We showed that our solver can rapidly assess the structural integrity of parts, provided corresponding geometric models are available.

Acknowledgements
We would like to thank the anonymous reviewers for their very detailed and diligent feedback, which has allowed us to improve the paper significantly.

Author contributions
SG/DH: conceptualization and methodology; SG/ ZH/LF: software development; SG/ZH/DH: writing - original draft preparation; SG/ZH/LF/DH: writing - review and editing; All authors read and approved the final manuscript.

Funding
The work leading to this publication has been supported by the VLAIO O&O project HBC.2020.3241 "Analysis for Design Engineers" (A4D), coordinated by Siemens Digital Industries Software (Leuven, Belgium) and involving KU Leuven (Leuven, Belgium) as research partner. The EU Recovery and Resilience Facility (RRF) is gratefully acknowledged for the financial support. Gefinancierd door de Europese Unie NextGenerationEU
Additionally, the project was funded by Siemens Digital Industries Software, Germany and Belgium, as well as Siemens AG, Germany.

Data availability
Data will be made available on reasonable request.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Computation, Information and Technology, Technical University of Munich, Arcisstraße 21, Munich, 80333, Germany. [2]Simulation and Test Solutions, Siemens Industry Software GmbH, Otto-Hahn-Ring 6, Munich, 81739, Germany. [3]Institute of Structural Analysis, Technical University of Braunschweig, Universitätsplatz 2, Braunschweig, 38106, Germany. [4]Simulation and Test Solutions, Siemens Industry Software N.V., Interleuvenlaan 68, Leuven, 3001, Belgium.

Received: 1 December 2023 Accepted: 4 September 2024 Published online: 16 September 2024

References
1. Newton P. In: The NAFEMS simulation capability survey 2015. NAFEMS - vendor advisory board. Glasgow. 2016. https://books.google.at/books?id=4kGOnQAACAAJ.
2. Ragani AF, Stein P, Keene R, Symington I. The state of simulation, prototyping and validation. A NAFEMS and McKinsey whitepaper. 2023. https://www.mckinsey.com/capabilities/operations/our-insights/unveiling-the-next-frontier-of-engineering-simulation.
3. Hanna R, Weinhold I. The democratization of CFD. Whitepaper, mentor graphics whitepaper. 2017. https://revolutioninsimulation.org/wp-content/uploads/2020/06/THE-DEMOCRATIZATION-OF-CFD-K-Hanna-I-Weinhold.pdf.
4. Hartmann D, Auweraer H. Digital twins. In: Cruz M, Parés C, Quintela P, editors. Progress in industrial mathematics: success stories. Cham: Springer; 2021. p. 3–17. https://doi.org/10.1007/978-3-030-61844-5_10.
5. Engineering.com. The State of Simulation, Prototyping and Validation. Research Report. 2022. https://image.engineering.com/239575/articles/Apr_2022/Dassault_State_of_Simulation_Prototyping_and_Validation.pdf.
6. Nealen A, Müller M, Keiser R, Boxerman E, Carlson M. Physically based deformable models in computer graphics. In: Computer graphics forum. vol. 25. New York: Wiley; 2006. p. 809–36.

7. NVIDIA. PhysX SDK. 2024. https://developer.nvidia.com/physx-sdk.
8. Dick C, Georgii J, Westermann R. A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. Simul Model Pract Theory. 2011;19(2):801–16. https://doi.org/10.1016/j.simpat.2010.11.005.
9. NVIDIA Corporation. NVIDIA CUDA Toolkit Documentation. https://developer.nvidia.com/cuda-toolkit. Accessed: 2023. (Year of Release).
10. Verzicco R. Immersed boundary methods: historical perspective and future outlook. Annu Rev Fluid Mech. 2023;55:129–55. https://doi.org/10.1146/annurev-fluid-120720-0221290.
11. Strouboulis T, Copps K, Babuška I. The generalized finite element method. Comput Methods Appl Mech Eng. 2001;190(32):4081–193. https://doi.org/10.1016/S0045-7825(01)00188-8.
12. Chessa J, Smolinski P, Belytschko T. The extended finite element method (XFEM) for solidification problems. Int J Numer Methods Biomed Eng. 2002;53(8):1959–77. https://doi.org/10.1002/nme.386.
13. Nitsche JA. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. Abh Math Semin Univ Hamb. 1971;36:9–15. https://doi.org/10.1007/BF029959040.
14. Hansbo A, Hansbo P. An unfitted finite element method, based on Nitsche's method, for elliptic interface problems. Comput Methods Appl Mech Eng. 2002;191(47–48):5537–52. https://doi.org/10.1016/S0045-7825(02)00524-80.
15. Alborghetti Londero A. A cut-cell implementation of the finite element method in deal.II. PhD thesis. Uppsala, Sweden: Uppsala Universitet; 2015.
16. Juntunen M, Stenberg R. Nitsche's method for general boundary conditions. Math Comput. 2009;78(267):1353–74. https://doi.org/10.1090/S0025-5718-08-02183-2.
17. Burman E, Hansbo P, Larson MG. A cut finite element method with boundary value correction. Math Comput. 2015;87:633–57. https://doi.org/10.1090/mcom/3240.
18. Hansbo P, Larson MG, Larsson K. Cut finite element methods for linear elasticity problems. 2017. ArXiv e-prints. arXiv:1703.04377 [math.nA].
19. Parvizian J, Düster A, Rank E. Finite cell method. Comput Mech. 2007;41(1):121–33. https://doi.org/10.1007/s00466-007-0173-y.
20. Düster A, Parvizian J, Yang Z, Rank E. The finite cell method for three-dimensional problems of solid mechanics. Comput Methods Appl Mech Eng. 2008;197(45):3768–82. https://doi.org/10.1016/j.cma.2008.02.036.
21. García-Ruíz MJ, Steven GP. Fixed grid finite elements in elasticity problems. Eng Comput. 1999;16(2):145–64. https://doi.org/10.1108/02644409910257430.
22. García MJ, Ruiz OE, Henao MA. Fixed grid finite element analysis for 3d strucutral problems. Int J Comput Methods. 2011;02(4):569–86. https://doi.org/10.1142/S0219876205000582.
23. Daneshmand F, Kazemzadeh-Parsi MJ. Static and dynamic analysis of 2d and 3d elastic solids using the modified FGFEM. Finite Elem Anal Des. 2009;45(11):755–65. https://doi.org/10.1016/j.finel.2009.06.003.
24. Nadal E, Rødenas JJ, Albelda J, Tur M, Tarancøn JE, Fuenmayor FJ. Efficient finite element methodology based on Cartesian grids: application to structural shape optimization. Abstr Appl Anal. 2013;**2013**. https://doi.org/10.1155/2013/953786.
25. Lin T, Lin Y, Sun W-W, Wang Z. Immersed finite element methods for 4th order differential equations. J Comput Appl Math. 2011;235(13):3953–64. https://doi.org/10.1016/j.cam.2011.01.041. Engineering and Computational Mathematics: a Special Issue of the International Conference on Engineering and Computational Mathematics, 27–29 May 2009.
26. Allard J, Courtecuisse H, Faure F. Chap. 21 - implicit fem solver on gpu for interactive deformation simulation. In: Hwu W-mW, editor. GPU computing gems jade edition. Applications of GPU computing series. Boston: Morgan Kaufmann; 2012. p. 281–94. https://doi.org/10.1016/B978-0-12-385963-1.00021-6.
27. Weber D, Bender J, Schnoes M, Stork A, Fellner D. Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. Comput Graph Forum. 2013;32(1):16–26. https://doi.org/10.1111/j.1467-8659.2012.03227.x.
28. Weber D, Mueller-Roemer J, Altenhofen C, Stork A, Fellner DW. A p-multigrid algorithm using cubic finite elements for efficient deformation simulation. In: VRIPHYS. 2014. p. 49–58.
29. Ljungkvist K, Kronbichler M. Multigrid for matrix-free finite element computations on graphics processors. 2017.
30. Kronbichler M, Ljungkvist K. Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Trans Parallel Comput. 2019;**6**(1). https://doi.org/10.1145/3322813.
31. Clevenger TC, Heister T, Kanschat G, Kronbichler M. A flexible, parallel, adaptive geometric multigrid method for FEM. ACM Trans Math Softw. 2021;47(1):7–1727.
32. Bangerth W, Hartmann R, Kanschat G. deal.ii—a general-purpose object-oriented finite element library. ACM Trans Math Softw. 2007;33(4):24. https://doi.org/10.1145/1268776.1268779.
33. Munch P, Heister T, Saavedra LP, Kronbichler M. Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. ACM Trans Parallel Comput. 2023;10(1):3–1338.
34. Jomo J, Oztoprak O, de Prenter F, Zander N, Kollmannsberger S, Rank E. Hierarchical multigrid approaches for the finite cell method on uniform and multi-level hp-refined grids. Comput Methods Appl Mech Eng. 2021;386:114075. https://doi.org/10.1016/j.cma.2021.114075.
35. Museth K. VDB: high-resolution sparse volumes with dynamic topology. ACM Trans Graph. 2013;32(3):1–22. https://doi.org/10.1145/2487228.2487235.
36. Zander N, Kollmannsberger S, Ruess M, Yosibash Z, Rank E. The finite cell method for linear thermoelasticity. Comput Math Appl. 2012;64(11):3527–41. https://doi.org/10.1016/j.camwa.2012.09.002.
37. Saberi S, Meschke G, Vogel A. The influence of Nitsche stabilization on geometric multigrid for the finite cell method. 2023. arXiv:2305.02161.
38. Lorensen WE, Cline HE. Marching cubes: a high resolution 3d surface construction algorithm. In: Proceedings of the 14th annual conference on computer graphics and interactive techniques. 1987.
39. Bourke P. Polygonising a Scalar Field. 1994. http://paulbourke.net/geometry/polygonise.
40. Wang S. 3D volume calculation for the marching cubes algorithm in Cartesian coordinates. 2013.

41. Gavranovic S, Hartmann D, Wever U. Topology optimization using GPGPU. In: Advances in evolutionary and deterministic methods for design, optimization and control in engineering and sciences. 2019. p. 553–66. https://doi.org/10.1007/978-3-319-89988-6_33.
42. Paige CC, Saunders MA. Solution of sparse indefinite systems of linear equations. SIAM J Numer Anal. 1975;12(4):617–29. https://doi.org/10.1137/0712047.
43. Hestenes MR, Stiefel E. Methods of conjugate gradients for solving linear systems. J Res Natl Bur Stand. 1952;49(6):409–36.
44. Elman HC, Silvester DJ, Wathen AJ. Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. 2nd ed. Numerical mathematics and scientific computation. Oxford: OUP Oxford; 2005. https://doi.org/10.1093/acprof:oso/9780199678792.001.0001.
45. Hackbusch W. Multi-grid methods and applications. Springer series in computational mathematics. vol. 4. Berlin: Springer; 1985. https://doi.org/10.1007/978-3-662-02427-0.
46. Briggs WL. A multigrid tutorial. 2nd ed. Miscellaneous Bks. Philadelphia: SIAM; 1987.
47. Intel Corporation. Intel® oneAPI Math Kernel Library (oneMKL). 2024. https://www.intel.com/content/www/us/en/docs/onemkl/developer-reference-c/2024-1/onemkl-pardiso-parallel-direct-sparse-solver-iface.html.
48. Siemens Digital Industries Software. Simcenter 3D. Siemens. Software application for simulation and analysis. 2023. https://www.sw.siemens.com/en-US/simcenter/3d-simulation/.
49. Hartmann D, Herz M, Paffrath M, Rommes J, Tamarozzi T, Auweraer HV, Wever U. In: Benner P, Grivet-Talocia S, Quarteroni A, Rozza G, Schilders W, Silveira LM, editors. Model order reduction and digital twins. Berlin: de Gruyter; 2021. p. 379–430. https://doi.org/10.1515/9783110499001-012.

Publisher's Note