CONSTRUCTING A VOLUME GEOMETRY MAP FOR HEXAHEDRA WITH CURVED BOUNDARY GEOMETRIES

Sandro Elsweijer[*†] Johannes Holke[†] Jan Kleinert[†] Dirk Reith[*]

January 14, 2022

ABSTRACT

In (dynamic) adaptive mesh refinement (AMR), a given input mesh is refined and coarsened during the computation to optimally adapt the resolution of the computational mesh to specific requirements. The input mesh is often the output of a mesh generator and provides information on the geometry of the domain. It is desired to keep its resolution as coarse as possible in order to benefit from the AMR mesh hierarchy and efficient mesh indexing algorithms. We present a novel approach to equip the coarse mesh with high-order geometry data and evaluate this geometry on the fine mesh elements in order to ensure geometric accuracy of the refined mesh elements, even for coarse input meshes. To this end, we construct a volume geometry map for hexahedral cells from given curved boundary geometry data and discuss our implementation in a state-of-the-art AMR library.

Keywords: adaptive mesh refinement, computational geometry, hexahedron, high-order meshes, mesh generation

1. INTRODUCTION

Tree-based adaptive mesh refinement (AMR) [1,2] starts with an initial unstructured coarse mesh C as input. The purpose of this mesh is to model the topology and geometry of the computational domain Ω. Each cell of the coarse mesh is then refined further into subelements up to a desired resolution according to given refinement rules. The resulting fine mesh is then used to numerically solve the problem at hand (e.g., a time-dependent partial differential equation (PDE)). During the computation, the fine mesh resolution may change dynamically in order to optimally adapt to the specific problem.

Structured, recursive refinement methods lead to a refinement tree in each coarse cell. Utilizing the tree structure drastically increases the performance of the mesh handling in terms of runtime, memory and parallel scalability compared to using fully unstructured meshes [1,3].
In general, the coarser the input mesh, the larger the benefit from the structured refinement. But the coarse mesh carries all information on the geometry of the domain and fewer coarse cells may result in a loss of information and hence accuracy of the geometry.

When the geometrical position of a fine mesh element or a point inside such an element is required for the computation, we compute it using the available geometry information of the coarse mesh. Since the fine mesh is constructed during the computation and can change arbitrarily, we do not know the position of the elements and possible points beforehand. Therefore, we require a general rule to evaluate the geometry at any point inside the coarse mesh reference element.

We present an approach to increase the geometric accuracy of the fine mesh. We do this by storing information from the original geometry at the coarse mesh level and developing an evaluation scheme to calculate correct geometric coordinates for any arbitrary point in the mesh; see Figure 1 for a visualization.
Thus, for each hexahedron coarse mesh cell C, we construct a function

$$G_C\colon [0,1]^3 \to \mathbb{R}^3 \tag{1}$$

mapping any point inside a unit cube reference element to its geometric position in the computational domain. This function should take all available information on the geometry of the coarse cell into account. Commonly used geometry

[*]Bonn-Rhein-Sieg University of Applied Sciences, Institute of Technology, Resource and Energy-Efficient Engineering, Sankt Augustin, Germany

[†]German Aerospace Center (DLR), Institute for Software Technology, Cologne, Germany, johannes.holke@dlr.de, corresponding author

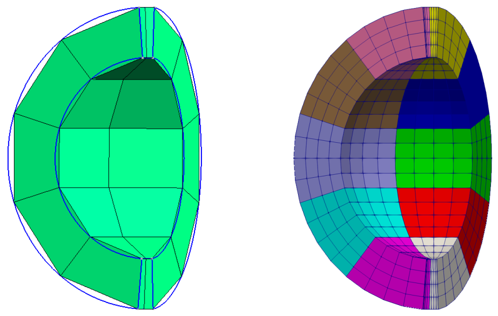


Figure 1: A coarse mesh (left, green) is used as input for adaptive mesh refinement (right, uniformly refined twice). We enhance the coarse mesh with the original geometry information (left, blue) to be able to arbitrarily adapt the fine mesh while maintaining geometrical accuracy.

models that serve as input for mesh generators use boundary representations of surfaces and curves of the geometry.

We describe the volume map with given arbitrary parameterized curve and/or surface data and present an implementation in the `t8code` AMR library [4] using the Open CASCADE Technology (OCCT) library for geometry evaluation [5]. In a first step, we concentrate on hexahedral mesh elements. However, we believe a generalization to tetrahedra, prisms and pyramids is possible, which will be part of future research.

Our approach differs from common approaches on high-order mesh generation such as [6, 7] and the references therein, where during pre-processing new points are introduced into existing meshes and projected onto curved boundaries to produce finer resolved input meshes. In contrast, our geometry evaluation happens during the simulation on any arbitrary position inside the coarse cells and does not require the solution of PDEs or the minimization of energy functionals. Its application is not restricted to tree-based AMR and could be used in all adaptive mesh settings where coarse cells are refined. A similar approach to ours uses radial basis functions [8].

2. FROM BOUNDARY GEOMETRY TO VOLUME GEOMETRY EVALUATION

In general, a parameterized curve γ is a map

$$\begin{array}{rcl} \gamma\colon \Omega_\gamma := [u_{\min}, u_{\max}] & \to & \mathbb{R}^3, \\ u & \mapsto & \gamma(u) \end{array} \tag{2}$$

and a parameterized surface σ, a map

$$\begin{array}{rcl} \sigma\colon \Omega_\sigma := [u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] & \to & \mathbb{R}^3, \\ (u,v) & \mapsto & \sigma(u,v). \end{array} \tag{3}$$

In our setting, a geometry is composed of vertices, (parameterized) curves and (parameterized) surfaces. In a first step, this geometry is meshed with an initial coarse mesh using a mesh generator. For each edge and face of a coarse mesh cell, we assume that the mesh generator can provide us with the

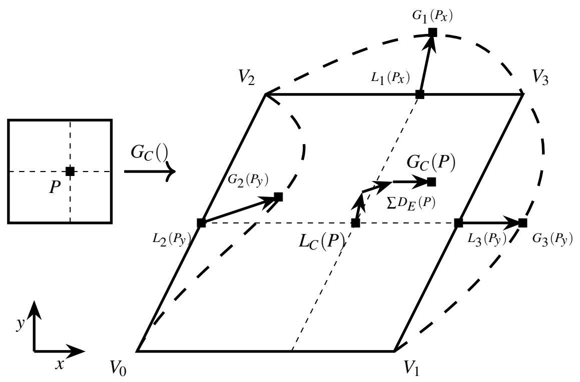


Figure 2: Mapping of the unit square to the actual, geometry-deformed cell volume. The map is based on linear interpolations and the displacements of linked surfaces and linked edges; see (5). The 3D case is analogous.

information of a parameterized curve or surface that they lie on[1]. We denote these as *linked edges* and *linked faces* of the coarse mesh cell. The edges of a linked face automatically carry the face's geometry (restricted to that edge). Hence, if a face is linked, we do not require its corresponding edges to be linked, too. Note that multiple coarse mesh cells may link to the same parametrized curve or surface.

For each linked face F (and linked edge E) of a coarse mesh cell C – interpreted as the unit cube – we obtain an associated parameterized surface σ_F (parameterized curve γ_E) and a geometry map

$$\begin{array}{rcccl} G_F\colon [0,1]^2 & \hookrightarrow & \Omega_{\sigma_F} & \to & \mathbb{R}^3, \\ (x,y) & \mapsto & (u,v) & \mapsto & \sigma_F(u,v), \end{array} \tag{4}$$

(or G_E in 1D). The first inclusion map is a linear embedding of the unit square (interval) into the parameter space Ω_{σ_F} (Ω_{σ_E}). It is, hence, completely determined by the values of the corners of the unit square (interval). We call these values the u,v parameters of the nodes on the geometric entity and can obtain their values from the input mesh; see Section 2.

The task at hand is to construct the volume map G_C (1) of a coarse mesh cell. Provided the vertex coordinates and the maps G_F, G_E for each linked face and edge are known.
Our solution (as depicted in Figure 2 for the 2D case) is the following: For a point $P = (P_x, P_y, P_z) \in [0,1]^3$ in the reference coarse mesh cell, we split $G_C(P)$ into the sum of the (tri-)linear interpolation $L_C(P)$ from the vertex coordinates (as if no geometry information was present) and the remainder, which are the displacements $D_E(P)$ and $D_F(P)$ of the edge/face geometry interpolation from the linear interpolation

$$G_C(P) = L_C(P) + \sum_{\substack{\text{linked} \\ \text{edges } E}} D_E(P) + \sum_{\substack{\text{linked} \\ \text{faces } F}} D_F(P). \tag{5}$$

[1]This assumption is reasonable for most common mesh generators and holds in particular for Gmsh [9] which we use in our implementation.

We can now compute the terms on the right-hand side of (5) to get $G_C(P)$. Here, $L_C(P)$ can be calculated from the vertex coordinates of the cell.

To compute the displacement of P for a face F (of the unit cube reference cell), we first determine the orthogonal projection $\pi_F(P)$ of P onto F. We then evaluate the surface geometry map G_F (4) at the projected point, and subtract the linear interpolation $L_F(\pi_F(P))$.

Since the influence of the geometry of the face should decrease with the distance of P from it, and should be zero on the opposite face, we scale the result with 1 minus the distance of P from the surface. This equals P_i resp. $1-P_i$, where i is the orthogonal coordinate to the face F. For example for the face $\mathscr{F}$ determined by $x=0$ we scale with $1-P_x$ and the total displacement calculates to

$$D_{\mathscr{F}}(P) = \left(G_{\mathscr{F}}(P_y,P_z) - L_{\mathscr{F}}(P_y,P_z)\right)(1-P_x). \tag{6}$$

For the face determined by $x=1$ we scale with P_x.

The computation of the displacement of a linked edge follows the same scheme. For the final scaling we need to take the two orthogonal directions of the edge into account, and scale with the product of P_i or $1-P_i$ and P_j or $1-P_j$. With this scaling we ensure that the influence decreases the further P is away from the edge and that it is zero on any face that is not adjacent to the edge. For example the edge $\mathscr{E}$ determined by $x=0$ and $z=1$ is scaled with $(1-P_x)P_z$ and the total displacement is

$$D_{\mathscr{E}}(P) = \left(G_{\mathscr{E}}(P_y) - L_{\mathscr{E}}(P_y)\right)(1-P_x)P_z. \tag{7}$$

Summarizing, we compute the volume geometry evaluation of a coarse cell by adding up the linear interpolation and the displacements for each of the cell's linked edges and faces.

Coupling with mesh generators

We use the open source mesh generator `Gmsh` [9] for generating input meshes. In its parametric mesh format, each geometric entity (vertex, curve, surface or volume of the meshed geometry) gets its own label and each coarse mesh node is labeled according to the geometric entity on which it is located. Moreover, the u,v parameters of each node that lies on a geometric entity are stored. Thus, there is no need to manually recombine the nodes with their geometries and re-calculate the parameters, as it would be mandatory for non-parametric meshes.

However, this information is not sufficient to enable the geometry-based AMR. A node can only be stored with one geometric entity at once (the entity it was generated on by `Gmsh`). This means some nodes get stored with the geometric entity of a vertex, some with that of a curve or surface, and the rest is not parametric.

Moreover, a node on a geometric vertex lies on all curves and surfaces connected to this vertex as well. This can cause that a quadrilateral face can have nodes on geometries with different dimensions (e.g. on a vertex, two different curves and a surface). This is a challenge, because our geometrical description from section 2 interpolates between the parameters of each node of the face. But in this case the nodes on a vertex have no parameters, the nodes on curves have an u-parameter and only the nodes on a surface have an u- and v-parameter. We obtain the missing parameters using the OCCT library.

3. EXPERIMENTS

Our first experiment is the meshing and refining of the NACA 6412 profile. We use `Gmsh` to generate the initial geometry in the OCCT brep format and then mesh it. After reading the mesh and the brep file in `t8code`, we use `t8code`'s `Adapt` algorithm [3] to refine the mesh along the wing boundary. The coordinates of the new elements are computed using (5) and map exactly to the initial NACA geometry; see Figure 3.

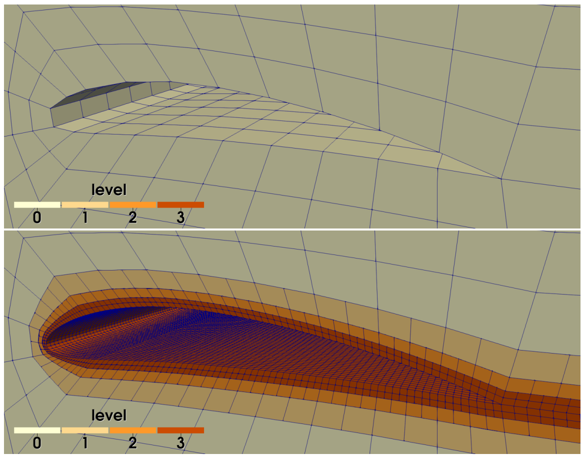
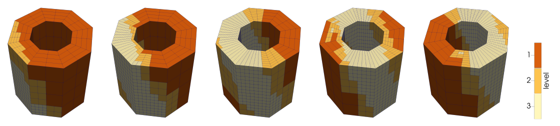


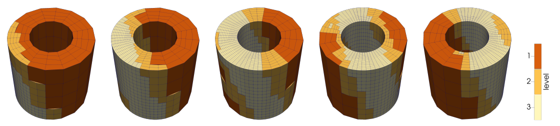
Figure 3: Top: Coarse input mesh of a NACA 6412 profile. Bottom: Adaptive mesh refined at the boundary with its coordinates evaluated according to the curved geometry using (5).

In the second experiment, we compare our curved geometry evaluation of a coarse input mesh with a linear geometry evaluation of the same input mesh. Our test geometry is a hollow cylinder that we mesh with different coarse mesh resolutions. The initial coarse mesh c_0 consists of four cells, which we divide into eight subcells recursively to obtain the coarse meshes c_m with 4×8^m cells.

We now compare the geometry linked versions of these coarse meshes with their non-linked (pure linear) versions. As a test scenario, we advance a plane through the cylinders and refine all elements in a certain proximity to the plane by three adaptive refinement levels. This resembles real world simulations, where we would use the adaptive refinement to track important features in the mesh, such as an interface between two fluids or a shock wave. To get a consistent amount of elements, we will also refine the coarse mesh by ℓ levels so that $m+\ell=6$. The plane advances in five time steps. An example of this is shown in Fig. 4



(a) Linear cell volume description



(b) Geometry-based cell volume description

Figure 4: A plane of refined elements (in beige) is moved through the cylinder meshes c_1. One mesh (top) uses the linear cell volume description, while the other (bottom) uses the new geometry-based cell volume description.

We measure the time spent on the algorithms for load-balancing, mesh adaptation, 2:1 balance and ghost layer creation [3] and the runtimes of the geometry evaluations using 14 MPI ranks in parallel. The results and more details on the measurements can be found in [10].

As can be expected, the runtime of the curved geometry evaluation using (5) is more expensive than the evaluation of the linear geometry, but remains reasonably bounded – at most 6.6 times for c_0. This factor decreases to 2.9 for c_3 and 2.4 for c_6, because there are proportionally more coarse mesh cells with no geometry linked to them.

The adapt algorithm is the only one of the tested AMR algorithms which uses the cell volume description. It does so to decide whether an element is close to the refinement plane or not. We chose this criterion deliberately to get and observe the influence of the geometry evaluation. As expected, only the adapt algorithm's runtime is affected by the new coarse mesh cell volume description and increased by a factor of 2.8 for c_0, 1.9 for c_3 and 1.6 for c_6. This is significantly less than the increase of the geometry evaluation runtime alone.

4. CONCLUSION

We have successfully constructed a volume geometry map for hexahedral cells given curved boundary information and demonstrated its usability in adaptive mesh refinement with the library `t8code`. With this technique, information such as element volumes, face areas, or positions of quadrature points in high order meshes can be calculated correctly with respect to the actual geometry.

Further research will include an integration of the techniques into simulation use cases with in-depth performance and accuracy evaluations. Additionally, we strive to extend the method to the remaining standard 3D element shapes tetrahedra, prisms and pyramids.

Our results are promising that a significant reduction in the coarse mesh size for tree-based AMR is possible even for complex geometries. Existing AMR simulation codes can benefit from an increased geometrical resolution and a decreased coarse mesh overhead. Additionally, our technique opens up memory efficient and scalable tree-based AMR for use cases with complex geometries that are traditionally dominated by classical pure unstructured meshes.

References

[1] Burstedde C., Wilcox L.C., Ghattas O. "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees." *SIAM Journal on Scientific Computing*, vol. 33, no. 3, 1103–1133, 2011

[2] Burstedde C., Holke J. "A tetrahedral space-filling curve for nonconforming adaptive meshes." *SIAM Journal on Scientific Computing*, vol. 38, C471–C503, 2016

[3] Holke J. *Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2018

[4] Holke J., Burstedde C., Elsweijer S., et al. "t8code: Parallel AMR on hybrid non-conforming meshes.", 2022. URL `https://github.com/holke/t8code/releases/tag/IMR2022`

[5] "Open CASCADE Technology." URL `https://www.opencascade.com/open-cascade-technology/`

[6] Shephard M.S., Flaherty J.E., Jansen K.E., Li X., Luo X., Chevaugeon N., Remacle J.F., Beall M.W., O'Bara R.M. "Adaptive mesh generation for curved domains." *Applied Numerical Mathematics*, vol. 52, no. 2, 251–271, 2005

[7] Mengaldo G., Moxey D., Turner M., Moura R.C., Jassim A., Taylor M., Peiró J., Sherwin S.J. "Industry-Relevant Implicit Large-Eddy Simulation of a High-Performance Road Car via Spectral/hp Element Methods." *CoRR*, vol. abs/2009.10178, 2020

[8] Krais N., Beck A., Bolemann T., Frank H., Flad D., Gassner G., Hindenlang F., Hoffmann M., Kuhn T., Sonntag M., Munz C.D. "FLEXI: A high order discontinuous Galerkin framework for hyperbolic–parabolic conservation laws." *Computers & Mathematics with Applications*, vol. 81, 186–219, 2021

[9] Geuzaine C., Remacle J.F. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities." *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, 1309–1331, 2009

[10] Elsweijer S. "Curved Domain Adaptive Mesh Refinement with Hexahedra." Tech. rep., Hochschule Bonn-Rhein-Sieg, Jul. 2021. URL `https://elib.dlr.de/143537/`