A Framework for Mesh-Geometry Associativity during Mesh Adaptation

Nicholas J. Wyman[1]
Pointwise, Fort Worth, TX 76104, USA

Michael A. Park[2]
NASA Langley Research Center, Hampton, VA 23681, USA

Patrick A. Baker[3] and John R. Chawner[4]
Pointwise, Fort Worth, TX 76104, USA

A framework has been developed for describing how a computational mesh is associated to the geometry model it discretizes. The target application of this framework is surface mesh adaptation in a CFD flow solver. The framework, called MeshLink, consists of two components. First, a schema has been defined for describing the one-to-many associativity of a surface mesh to the geometry model entities to which it is attached. Second, a high-level library provides a kernel-agnostic wrapper for providing the necessary geometry queries to an application (e.g., mesher, flow solver). Both the schema and library are provided freely and openly. MeshLink's ability to support solution mesh adaptation on linear and curved meshes for a high-order flow solution is demonstrated on several test cases relevant to the aerospace and automotive industries.

I. Nomenclature

API	=	application programmer interface
BC	=	boundary condition
B-Rep	=	boundary representation
CAD	=	computer-aided design
CFD	=	computational fluid dynamics
gid	=	geometry identification
gref	=	geometry reference
NURBS	=	nonuniform rational B-Spline
U,V	=	parametric coordinates
VC	=	volume condition
X,Y,Z	=	Cartesian coordinates

II. Introduction

The CFD 2030 Vision Study [1] identified an "inadequate linkage with CAD" as a pacing item in the advancement of applied CFD. Beyond the usual geometry model issues of interoperability and suitability for simulation, the Study noted that CFD flow solvers need to evaluate the geometry model (e.g., compute curvature, project mesh points onto the model) and to perform those evaluations in a tightly-coupled, low-overhead manner within a massively parallel computational environment. An example of this need is mesh adaptation during which surface mesh points constrained to the geometry model must be moved, deleted, or new points inserted.

[1] Director, Applied Research, AIAA Associate Fellow
[2] Research Scientist, NASA Langley Research Center, AIAA Associate Fellow
[3] Director, Product Development, AIAA Senior Member
[4] President, AIAA Associate Fellow

While mesh adaptation is the target application for this work, any downstream consumer of a mesh that must modify the mesh could benefit from a more "adequate linkage with CAD." Therefore, the manner in which mesh adaptation is to be performed is not a primary subject of this work. The essential challenge addressed herein is how the downstream mesh consumer can modify the mesh while ensuring it remains attached to the geometry model.

Evaluation of geometry models, in particular those comprised of NURBS in a B-Rep topology, is a solved problem. CFD flow solver authors have access to a plethora of proprietary (e.g., Parasolid [2], Geode [3]) and open-source (Open CASCADE [4], EGADSlite [5]) geometry kernels to project points and evaluate curvature for performing simulation computations. Therefore, delivering a new geometry kernel is not a primary subject of this work. Here we focus on providing the critical linkage between the mesh and geometry kernel to support the CFD 2030 Vision.

Prior work by the authors [6] concluded that just providing geometry kernel software to CFD flow solver developers is not sufficient to robustly meet their needs for mesh-geometry adherence. What is lacking is a description of the associativity between the mesh and the geometry model that can be exchanged between applications (e.g., initial mesher, flow solver, adaptive mesh mechanics). A flow solver typically limits its mesh associativity queries to boundary conditions (BC) (e.g., wall, symmetry plane, outflow) and volume conditions (VC) (e.g., fluid, porous media, solid). Even if one were to assume that all wall BC mesh points are on the geometry model, there is no data to identify exactly where on the geometry model those points belong (i.e., on which specific geometry entity). To make matters more complicated, some surface mesh points are actually constrained to curve entities (often intersection curves) versus surfaces. This additional complication arises from the fact that the computation of intersection curves is a tolerance-based fit of discrete points. Because this tolerance can be on the order of the local mesh edge length (especially when refining the mesh during adaptation), knowing whether mesh points adhere to the intersection curve or one or the other of its parent surfaces is critical to robust use of the geometry model.

In addition to computational robustness, a concise description of mesh-geometry associativity should improve computational efficiency when mesh adaptation is deployed in a massively parallel environment. When a CFD application is deployed across multiple compute cores, it is likely that only a minority of the cores will contain geometry-constrained surface meshes. Knowledge of which compute nodes require access to specific geometry model entities will help achieve the goal of low-overhead geometry computations.

Therefore, in addition to classification by BC, flow solvers need to access data that classifies the mesh by geometry model associativity. The following sections define this associativity, describe a schema for documenting the associativity, introduce a high-level API for implementing the framework, and demonstrate the utility of this framework. Collectively, this framework is known as MeshLink.

III. Defining Mesh-Geometry Associativity

The term geometry association is used to denote the relationship between a mesh element (including its nodes and edges) and the supporting computational geometry model used to constrain the location of the mesh element in Cartesian space. The geometry association provides a link between the mesh element identifier defined by the mesh data file and the computational geometry entity identifier defined in the geometry data file.

MeshLink provides access to a number of geometry kernels. The examples in this paper use Pointwise's Geode kernel. Project Geode is a response to the NASA CFD Vision 2030 Study's identification of the lack of CFD software access to geometry. Geode is currently available to a select group of partners. Geode is not intended to compete with or replace the many geometry kernels – both proprietary and open-source – currently available in the market today. The Geode kernel is tuned to the needs of engineering simulation in the sense that its capabilities and performance are biased toward evaluations and inverse evaluations versus providing a breadth of geometry creation functionality. Its heritage is the full geometry modeling kernel, which forms the basis of the Pointwise® [7] meshing software's geometry-related capabilities as well as some of its meshing techniques.

The MeshLink Library defines a GeometryGroup (CamelCase geometry terms defined below) object that provides a unique integer identifier for the computational geometry entity(s) associated with a mesh element. The Mesh Element object stores a geometry reference integer matching the unique identifier of the associated Geometry Group object. The Geometry Group object and reference identifier scheme allows mesh associativity both to a single geometry entity and to multiple geometry entities using a common API. Furthermore, the unique GeometryGroup identifier allows for efficient representation of repeated geometry association for multiple mesh entities, which is a common occurrence.

At the mesh element level, the following terminology is used:

MeshPoint - A zero-dimensional mesh element, typically defined in the mesh data file and referenced by the index to the point in that file. Geometry associativity may be optionally stored for each MeshPoint element by prescribing

the appropriate Geometry Group reference integer. Geometry associativity defined on the MeshPoint element supersedes any geometry associativity defined on a parent mesh topology container.

MeshEdge - A one-dimensional mesh element, typically referenced by the two end point indices. Geometry associativity may be optionally stored for each MeshEdge element by prescribing the appropriate Geometry Group reference integer. Geometry associativity defined on the Mesh Edge element supersedes any geometry associativity defined on a parent mesh topology container.

MeshFace - A two-dimensional mesh element, typically triangular or quadrilateral and referenced by the face vertex indices. Geometry associativity may be optionally stored for each MeshFace element by prescribing the appropriate GeometryGroup reference integer. Geometry associativity defined on the MeshFace element supersedes any geometry associativity defined on a parent mesh topology container.

MeshString - The MeshString topology level is a container for MeshEdge elements. A Mesh String is typically used to associate a collection of MeshEdges to one or more computational geometry curves, as shown in Figure 1, by prescribing the appropriate Geometry Group reference integer. The MeshString abstraction allows contained MeshEdges to use the MeshString geometry association without requiring an explicit geometry association for each MeshEdge. A MeshString is considered the lowest (most specific) topology level within a MeshModel.

MeshSheet - The MeshSheet topology level is a container for MeshFace elements. A MeshSheet is typically used to associate a collection of Mesh Faces to one or more computational geometry surfaces, as shown in Figure 1. The Mesh Sheet abstraction allows contained MeshFaces to use the MeshSheet geometry association without requiring an explicit geometry association for each MeshFace.

MeshModel - The MeshModel topology level is a container for MeshString and MeshSheet topology and corresponds to a mesh volume region in the third-party simulation software system. The primary purpose of the Mesh Model topology level is to provide organization of the MeshString and MeshSheet topology levels corresponding to the mesh volume. This is essential when working with multiple mesh volumes, for example, in an overset mesh simulation.

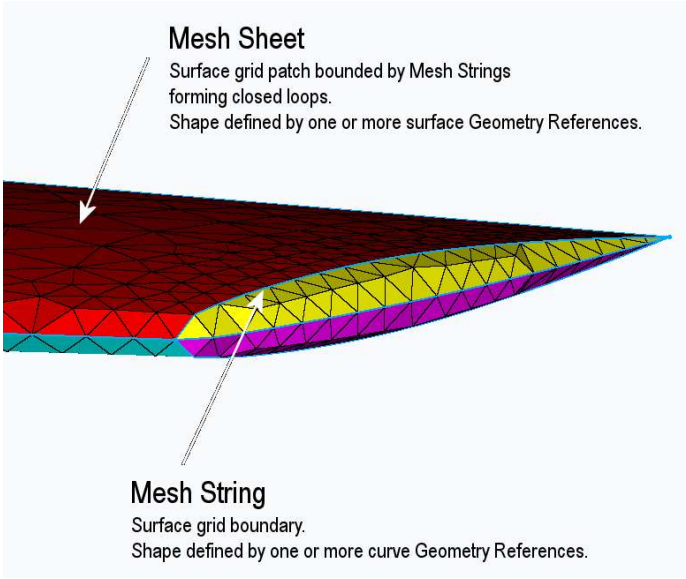

Figure 1: MeshString and MeshSheet topology objects contain MeshEdge and MeshFace elements.

IV. The MeshLink Schema and File

An open, neutral file format, termed the MeshLink file, is designed and documented for the transfer of mesh-geometry associativity data. The MeshLink file associates geometry entities contained in the geometry data file with mesh elements contained in the mesh data file. The three files taken together (mesh, geometry, and MeshLink) represent the entire geometry and mesh configuration. The MeshLink file contains paths to the geometry and mesh data files and, therefore, could be considered the configuration's master file.

The MeshLink file format is XML and compliant with the *XML Schema Definition* (XSD) standard [8 and 9]. By following XML standards, the resulting MeshLink file is compatible with several open source, freely available XML

parsing and schema-validating software packages. Furthermore, since the XML file may be validated against the MeshLink XSD, consistency can be ensured across a variety of implementations while still allowing flexibility of use for any given application. The MeshLink file's XML hierarchy is illustrated in Figure 2.

The schema was designed to allow maximum flexibility for a given application. For example, the content (text node) of a MeshSheet (collection of surface-like mesh elements) is not rigidly defined to allow an application the freedom to define their own structure. That is, only the framework and hierarchy is supplied by the schema, without specific requirements on the mesh structure.

The MeshLink file defines geometry associativity in a mesh-centric manner, that being the most natural manner of using the associativity data in a simulation framework. Geometry entities are given a unique global identifying integer termed the *gid* (geometry reference ID). Mesh elements reference their associated geometry by specifying the *gref* attribute – an integer matching the geometry entity's *gid*.

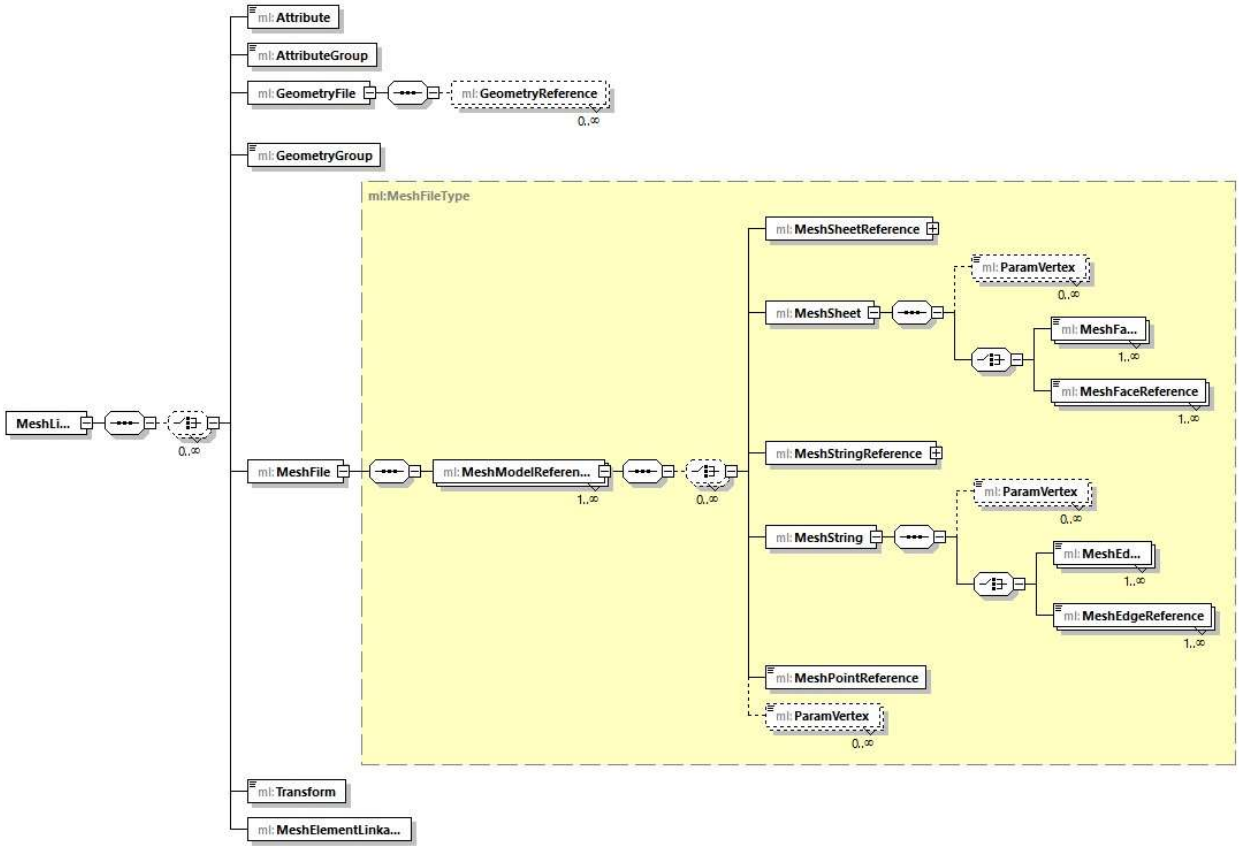

Figure 2: MeshLink XML Schema Hierarchy.

The elements of the MeshLink file are described below.

MeshLink - The mandatory outer-most container element in the MeshLink file.

Attribute - Allows definition of an arbitrary, application-defined attribute that can be linked to any element within the MeshLink file. Each Attribute must have a unique attribute ID (*attid*) among all Attribute and AttributeGroup elements within a single MeshLink definition. The text node (i.e., body) of an Attribute element describes the attribute's value in any suitable manner in the context of the application.

AttributeGroup - A mechanism for aggregation of Attribute elements. MeshLink elements can only reference a single Attribute or AttributeGroup by *attid*, so this provides a flexible way to assign multiple, arbitrarily-grouped Attribute elements. The text node of an AttributeGroup must be a list of unique *attids* of any Attribute or AttributeGroup already defined in the MeshLink file. Attribute and AttributeGroup are assigned to most MeshLink elements through an optional *aref* attribute (the attribute reference ID).

GeometryFile - A GeometryFile defines a set of tagged (e.g., named or enumerated) geometric elements, represented as GeometryReference elements, which are typically surfaces (or surface-like entities) and curves,

obtained from the external resource indicated by the filename attribute. Multiple GeometryFile elements are allowed, but all geometry reference IDs (*gid*) must be unique within a single MeshLink definition.

GeometryReference - A GeometryReference element maps a single named geometric entity (surface, surface-like, or curve) from the enclosing geometry file via the ref attribute to a unique numeric geometry reference ID (*gid*). These gids are later used to link mesh elements to geometry.

GeometryGroup - A GeometryGroup is a mechanism for grouping GeometryReference elements in order to associate a single mesh element (MeshSheet, MeshString) to multiple GeometryReference elements. A GeometryGroup must have a *gid* that is unique among all GeometryReference and GeometryGroup elements in the MeshLink file. The text node of a GeometryGroup must be a list of unique *gids* of any GeometryReference or GeometryGroup already defined in the MeshLink file.

MeshFile - A MeshFile element defines a set of mesh elements and mesh topology, obtained from the external resource indicated by the filename attribute. A mesh element can be linked to a single *gid* of a GeometryReference or GeometryGroup through its *gref* attribute. Mesh elements can be defined explicitly or referenced from the MeshFile by any mechanism suitable to the application.

MeshModelReference - A MeshModelReference provides a context for mesh elements that can be linked to geometry. A MeshModelReference typically refers to a contiguous mesh region (a block). A MeshModelReference is mapped to the mesh region in the mesh file in an application-specific way through the *ref* attribute.

MeshSheet - A MeshSheet element provides a way to define a set of surface cells to be linked to geometry. A MeshSheet can be defined by a collection of MeshFace and/or MeshFaceReference elements, or in an application-defined manner in its text node. A MeshSheet is typically linked to a surface or surface-like GeometryReference or GeometryGroup through the *gref* attribute. A MeshSheet must reside in the context of a MeshModelReference element.

MeshFace - The text node of a MeshFace element defines one or more mesh surface cells in an application-defined manner. Like a MeshSheet, a single MeshFace can be linked to geometry through its *gref* attribute, which may override the linked geometry of an enclosing MeshSheet depending on the application. The type of element (linear triangle, quadratic quadrilateral, etc.) is indicated in its *etype* attribute. A MeshFace must reside within the context of a MeshSheet element.

MeshFaceReference - The text node of a MeshFaceReference provides a way to reference a single surface cell from the MeshFile in an application-defined manner. Like a MeshFace, a MeshFaceReference may also be linked to geometry through the *gref* attribute, and must also indicate its element type through the *etype* attribute. A MeshFaceReference must reside within the context of a MeshSheet element.

MeshString - A MeshString element defines a set of mesh edges in an application-defined manner. A MeshString can be defined by a collection of MeshEdge or or MeshEdgeReference elements, or in an application-defined manner in its text node. Similar to MeshSheet and MeshFace, a single MeshString can be linked to geometry through its *gref* attribute. A MeshString must reside in the context of a MeshModelReference element.

MeshStringReference - The text node of a MeshStringReference provides a way to reference a set of edges or other one-dimensional boundary-like entity or entities from the MeshFile in an application-defined manner. Like a MeshString, a MeshStringReference may be linked to geometry through the *gref* attribute. A MeshStringReference must reside in the context of a MeshModelReference element.

MeshEdge - The text node of a MeshEdge element defines one or more mesh surface cells in an application-defined manner. Like a MeshString, a single MeshEdge can be linked to geometry through its *gref* attribute, which may override the linked geometry of an enclosing MeshString depending on the application. The type of element (linear, quadratic, etc.) is indicated in its *etype* attribute. A MeshEdge must reside within the context of a MeshString element.

MeshEdgeReference - The text node of a MeshEdgeReference provides a way to reference a single edge from the MeshFile in an application-defined manner. Like a MeshEdge, a MeshEdgeReference may also be linked to geometry through the *gref* attribute, and must also indicate its element type through the *etype* attribute. A MeshEdgeReference must reside within the context of a MeshString element.

Transform - The text node of a Transform element provides a way to define an arbitrary geometric transformation operator in an application-defined way. For example, a Cartesian translation/rotation transformation matrix could be defined to specify a periodic linkage.

MeshElementLinkage - A MeshElementLinkage element provides a way to link two related mesh elements using the sourceEntityRef and targetEntityRef attributes with an optional transform using the *xref* attribute in an application-defined manner. The typical usage of MeshElementLinkage is to specify periodic MeshSheet and MeshString relationships.

In addition to the descriptions above, MeshModel, MeshSheet, and MeshString elements may contain ParamVertex elements. A ParamVertex element links a mesh point to a geometry element and provides parametric coordinate information in an application-specific manner in its text node. For example, the content of a coordinate in the parameter space of a surface element could be represented as (U,V) coordinates, where a curvilinear coordinate might be expressed as an arc-length parameter. The mesh vertex is referenced from the MeshFile through the *vref* attribute (the application-defined vertex reference string), also in an application-specific manner. For example, this could be a reference to a point index within the mesh file or mesh model.

ParamVertex elements are typically used when parametric geometry association is desirable. For example, an application may choose to perform mesh adaptation by using parametric (1D or 2D) interpolation, rather than model space (3D) interpolation. In that usage, the MeshLink ParamVertex information is provided to, and used by, the application to calculate new mesh point locations parametrically, and the MeshLink API is used to calculate the new location in model space using the geometry kernel.

The MeshLink file is an ASCII XML text file which ensures portability between hardware platforms. The XML standard allows comment strings making the file human readable, allowing an application to provide context for debugging purposes. Since the text nodes (body) of the MeshSheet, MeshString, MeshFace and MeshEdge elements can be defined by the application, compact data structures and special encoding (e.g., base64) can be used to improve the efficiency of the file storage.

The MeshLink schema and reference MeshLink API implementation are available on GitHub [10] along with a number of example XML files.

V. The MeshLink API

The MeshLink API is a high-level application programming interface that abstracts much of the details of evaluating geometry models from the CFD flow solver developer. The API is written in C++ with wrappers for C and Fortran. A Python interface is under development. The API is not a kernel itself. Instead, the API wraps any geometry kernel with less than a dozen, commonly used functions listed here.

- Read the geometry model file
- Read the MeshLink XML file
- Project a mesh point onto the geometry model
- Return the (X,Y,Z) coordinates of a projected mesh point
- Return the (U,V) coordinates of a projected mesh point
- Return the geometry model entity ID of a projected mesh point
- Evaluate the (X,Y,Z) coordinates for given (U,V) coordinates
- Evaluate the radius of curvature of the geometry model at given (U,V) coordinates

The API also wraps any XML parser that an application chooses. Our implementation includes a fully wrapped XML parser with schema (XSD) validation. The MeshLink API has been designed to be streamlined for common analysis tasks and does not require an intimate knowledge of geometry model topology.

To provide context for the demonstrations to follow, pseudocode for a typical application to instantiate a new Mesh Associativity database and populate it with associativity data from the MeshLink XML file is shown below.

```
meshAssoc = new MeshAssociativity;
// parser semantics are provided by the application
MeshLinkXMLParser parser;
parser.readMeshAssociativity(mlFile, meshAssoc);
```

Pseudocode to instantiate a new Geometry Kernel wrapper and populate it with geometry entities from linked geometry files in the Mesh Link XML file is shown below.

```
// API-compliant geometry kernel implementation is provided
// by the application
MyGeometryKernel kernel;
kernel.loadGeometry(meshAssoc);
```

Once the application has loaded the MeshLink associativity and instantiated an API-compliant GeometryKernel, the API is used to query mesh elements and their corresponding geometry references, and then evaluate or project new or updated coordinates as the application requires. For example, the most anticipated use for the API involves point projection (i.e., inverse evaluation) to geometry associated with a MeshFace (face refinement), MeshEdge (edge splitting) or higher level topology such as a MeshString or MeshSheet (point smoothing). In determining the geometry associativity for a MeshEdge or MeshFace, we can leverage the fact that determining the associativity at the lowest (most specific) topology level is most desirable thereby making the pseudocode generic as shown below.

```
// Locate a MeshEdge by point indices at the lowest topological level
targetEdge = meshModel.findLowestTopoEdgeByInds(ind1, ind2);

// Determine Geometry Group association for the target edge
geomRef = targetEdge.getGref();

// Locate a triangle MeshFace by point indices at the
// lowest topological level
targetFace = meshModel.findLowestTopoFaceByInds(ind1,ind2,ind3);

// Determine Geometry Group association for the target face
geomRef = targetFace.getGref();
```

Regardless of how the geometry associativity is determined, the following pseudocode for projecting a point to the associated geometry becomes generic as well.

```
kernel.projectPoint(geomRef, xyz, projData);
kernel.getProjectionXYZ(projData, projectedXYZ);
```

VI. Geometry Access in an HPC Environment

The MeshLink schema allows the geometry model to be efficiently accessed within an HPC environment. For example, the largest mesh examples in this paper used 400 cores in a distributed memory implementation. Given the MeshLink file and the partitioning of the mesh across the compute nodes, one can easily determine which geometry entities are required on each compute node for adaptation. Also, the MeshLink schema can be easily streamed within an HPC environment so that communication need not be done using file transfer.

During a bootstrap phase, the *refine* mesh adaptation tool [11], caches the one-to-many MeshLink mesh-geometry associativity from the MeshLink API. The cached mesh-geometry associativity is stored at each vertex where a vertex will have zero or more sheet, string, or point associations [12]. The vertex associativity information is written in a Gamma Mesh File (meshb) format record [13] to retain the mesh and mesh-geometry associativity in a single file. In parallel, r*efine* reads the meshb records in parallel and performs a load balance and migration of mesh and associativity information. *refine* instantiates a Geode and a MeshLink on each MPI rank and instructs these instances to individually read the required data (in XML and NMB formats) directly from disk. At higher core counts, each instance directly reading geometry data may become a bottleneck. An extension to the MeshLink/Geode via API could be made to cache and return the geometry data in an opaque binary block in the same manner as the mesh-geometry associativity meshb record.

VII. Demonstrations

All of the demonstrations use the reference implementation of MeshLink API and the Geode geometric modeling kernel. They also use the Apache Xerces XML parser [14]. The cases labeled with the FUN3D (Fully Unstructured Navier Stokes 3D) finite-volume CFD flow solver [15] include the Spalart-Allmaras (SA) turbulence model [16], *refine* mesh adaptation tool, MeshLink, and Geode.

A. ONERA M-6 Wing – Mesh Refinement to Geometry

A simple demonstration of MeshLink involves refinement of a mesh on the ONERA M-6 wing [17]. Given the geometry model shown in Figure 3 and the coarse mesh shown in Figure 4, simple geometric resolution requirements are used in a process designed to mimic adaptation to a flow solution.

The tool chain used in this demonstration begins with a coarse mesh generated on the geometry model by using Pointwise. The mesh was exported from Pointwise with the geometry model in Geode's NMB format and MeshLink data in an XML file.

Refinement goals for this mesh were strictly geometric.

- Limit mesh edge lengths to circular arc subtension of 20 degrees based on the associated geometry radius of curvature.
- Set the minimum mesh edge length to 0.5% of wing chord.
- Set the maximum mesh cell aspect ratio to 20.
- Set the minimum included angle in a mesh cell to 5 degrees.

The resulting refined surface mesh is shown in Figure 5.

The example demonstrates retrieval of geometry associativity for a MeshEdge and evaluation of geometry curvature at the edge midpoint to determine the refinement metric. Mesh refinement is accomplished by a simple edge-splitting operator which illustrates modification of the mesh-geometry associativity information through the addition of new MeshPoints and new child MeshEdges, and MeshFaces. Additionally, removal of parent MeshEdge and MeshFace geometry associativity information is demonstrated.

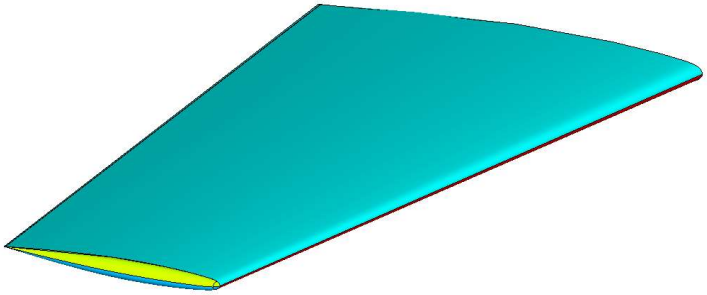

Figure 3: Geometry model of the ONERA M-6 wing. Colors are used to delineate the various entities in the model.

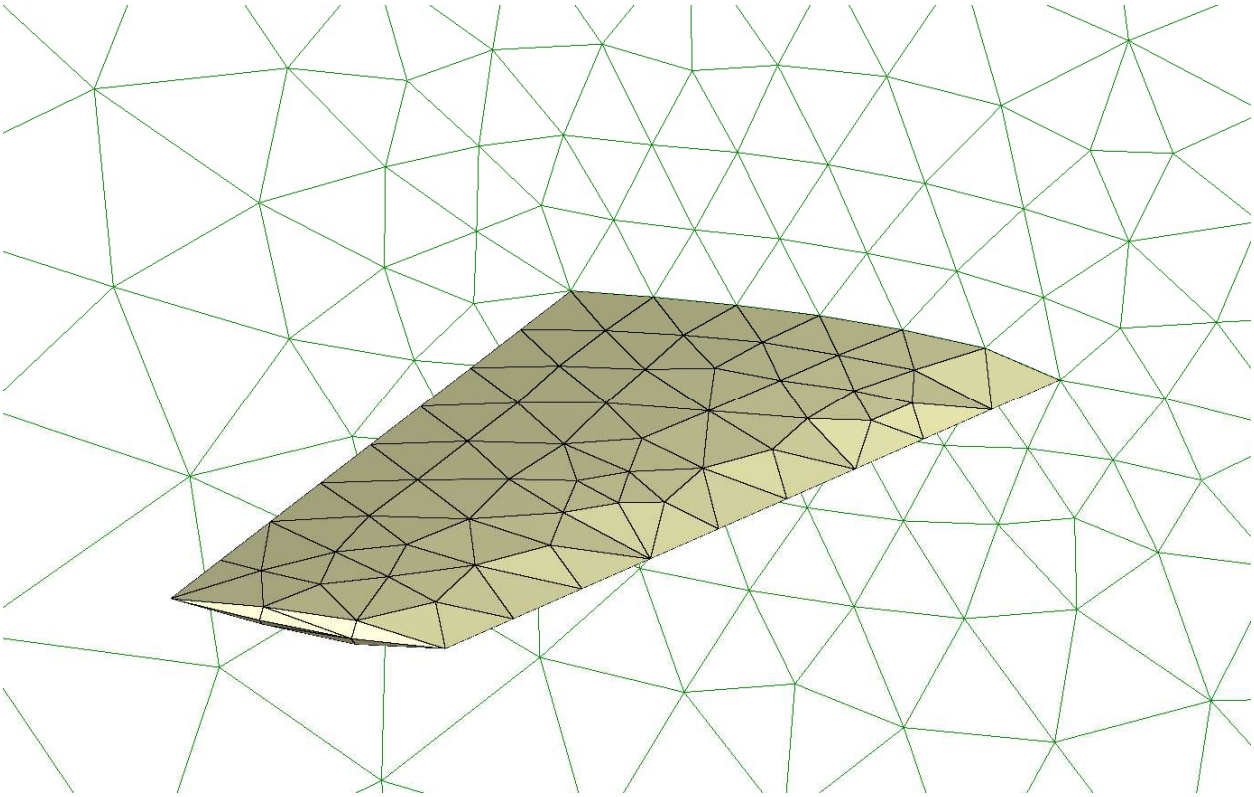

Figure 4: The coarse initial mesh for the ONERA M-6 wing demonstration case.

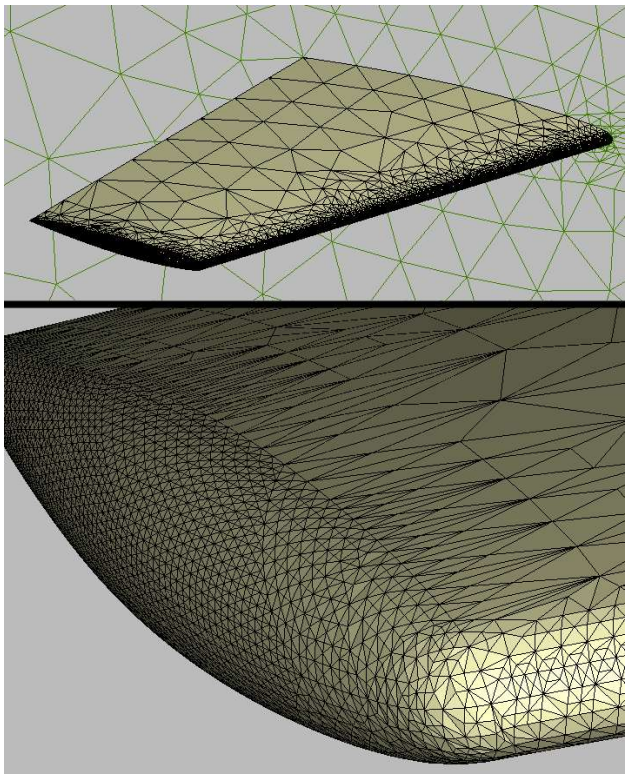

Figure 5: The result of adapting a coarse mesh for the ONERA-M6 mesh to simple geometric goals while maintaining adherence to the geometry model using MeshLink.

B. ONERA M-6 Wing - Mixed-Order h-p Mesh Adaptation

MeshLink was used to handle the geometry model adherence when adapting an ONERA M-6 mesh to a flow solution computed using the SU2 Multiphysics Simulation and Design Software [18] as shown in Figure 6.

In this example, the flow solution is processed to locate the pressure discontinuity across the lambda shock. The shock region is designated for h-refinement (using linear elements) via source point clouds in the Pointwise meshing software by using the process described in Ref. [19]. Elsewhere, relative solution interpolation error is used to define a target element polynomial order Q. The Pointwise hp_CurveMesh software [20] performs elevation to the target polynomial order, while maintaining linear elements at the shock location. The MeshLink library and geometry associativity information are used in the hp_CurveMesh application for insertion of high-order nodes on the geometry and for maintaining geometry adherence during point smoothing.

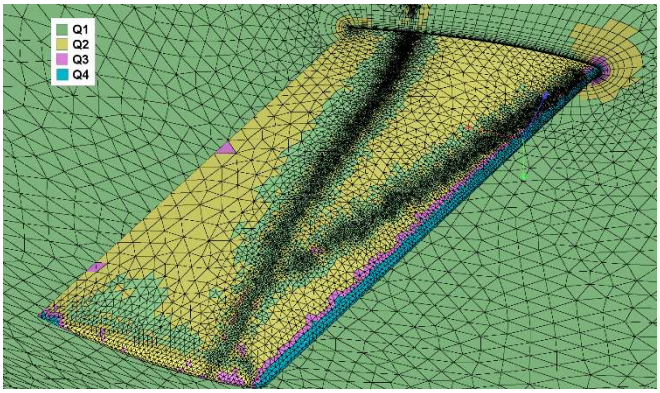

Figure 6: This mesh for the ONERA M-6 was applied hp-adaptation to a flow solution. The mesh cells are colored by their polynomial degree Q.

C. ONERA M-6 Wing - Linear Anisotropic Adaptation

The mesh in Figure 7 is adapted to surface curvature by using the process described in Ref. [12]. The one-to-many association feature of Meshlink eliminates the constraint of a thin trailing edge face that is present in the geometry model because it is grouped with the larger adjacent face. This coarse surface curvature constrained mesh was used to launch the CFD computation and mesh adaptation to the flow solution. Boundary layer resolution is absent on the initial mesh as illustrated by the symmetry plane mesh.

The flow conditions are 0.84 Mach, 3.06° angle of attack, 14.6M unit Reynolds number per root chord, and 300° Kelvin. Fifty cycles of mesh adaptation were performed with increasing mesh resolution. The final mesh has 9.8 million vertices while the starting mesh had 8.7 thousand vertices. New string and sheet vertices are constrained via MeshLink API inverse evaluations. Mesh smoothing is also performed with inverse evaluations. The upper surface lambda shock feature is observed in the final, adapted surface mesh (Figure 8). Adaptation of the mesh to the thin boundary layer, wake, and shock wave can be seen imprinted on the upper surface and symmetry plane (Figure 9).

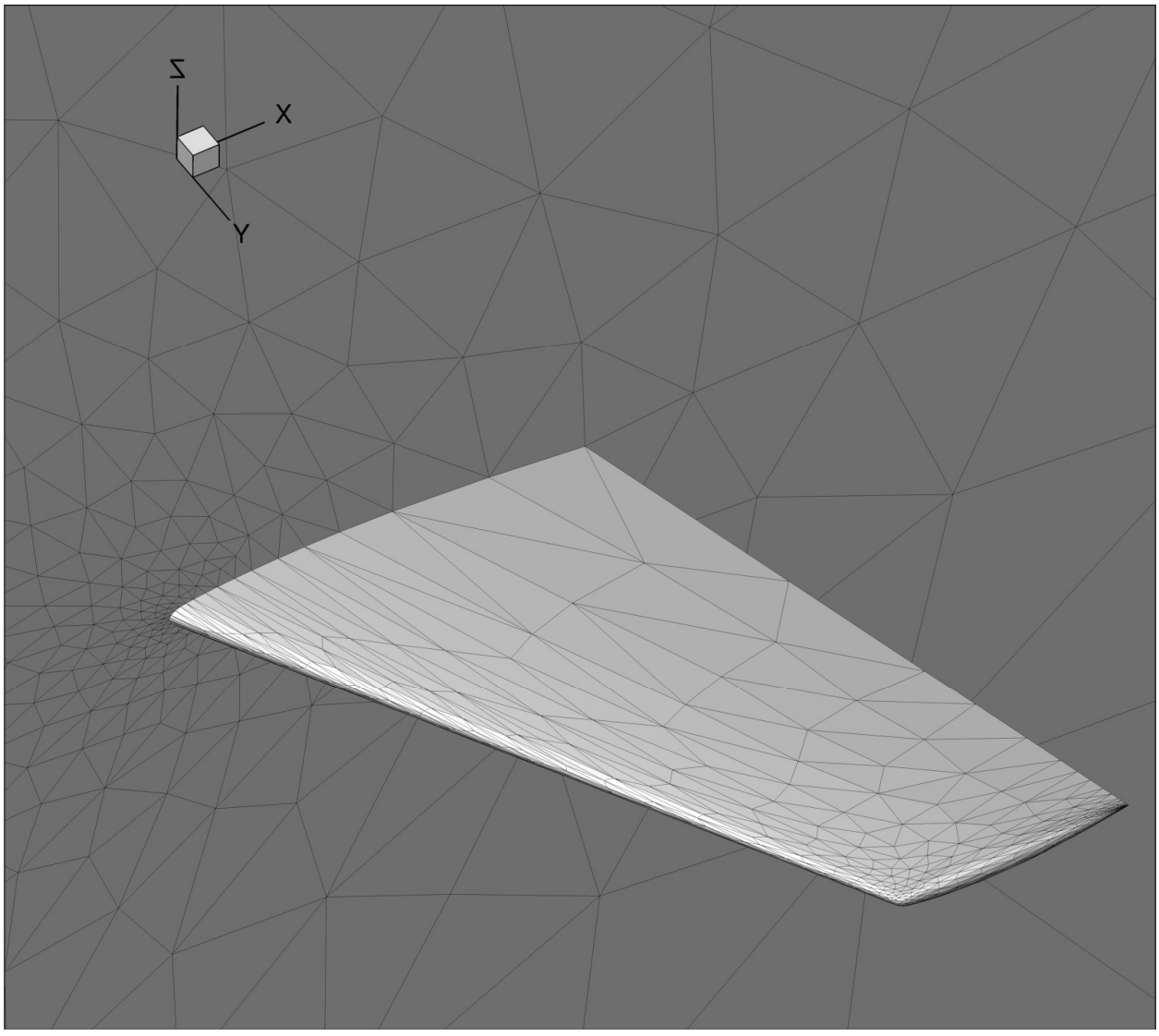

Figure 7: This mesh for the ONERA M-6 wing served as the starting point for mesh adaptation to a flow solution. It was adapted to the geometry model's surface curvature by the *refine*-MeshLink tool chain.

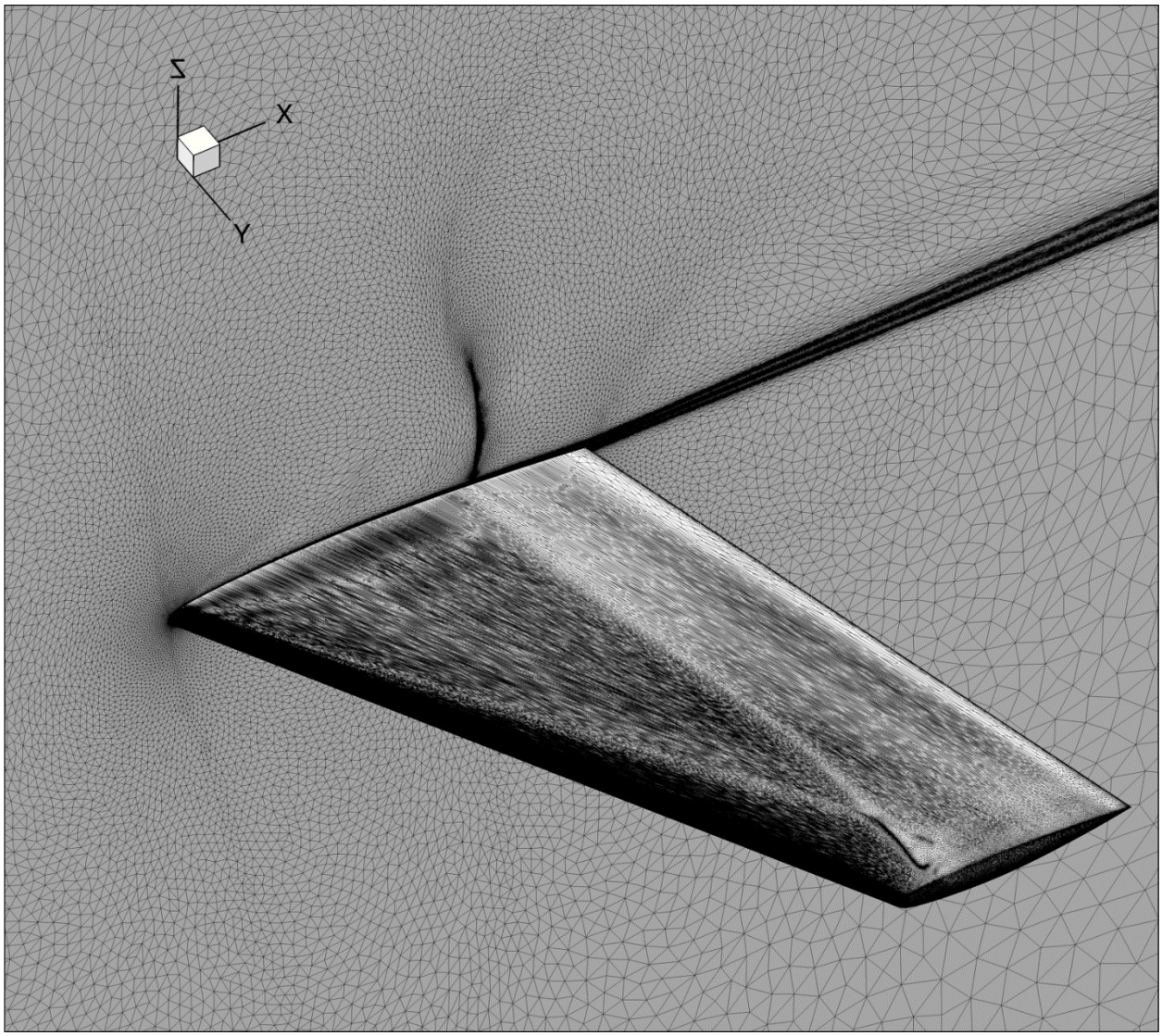

Figure 8: This mesh for the ONERA M-6 Mesh has been adapted by the FUN3D tool chain to the CFD solution.

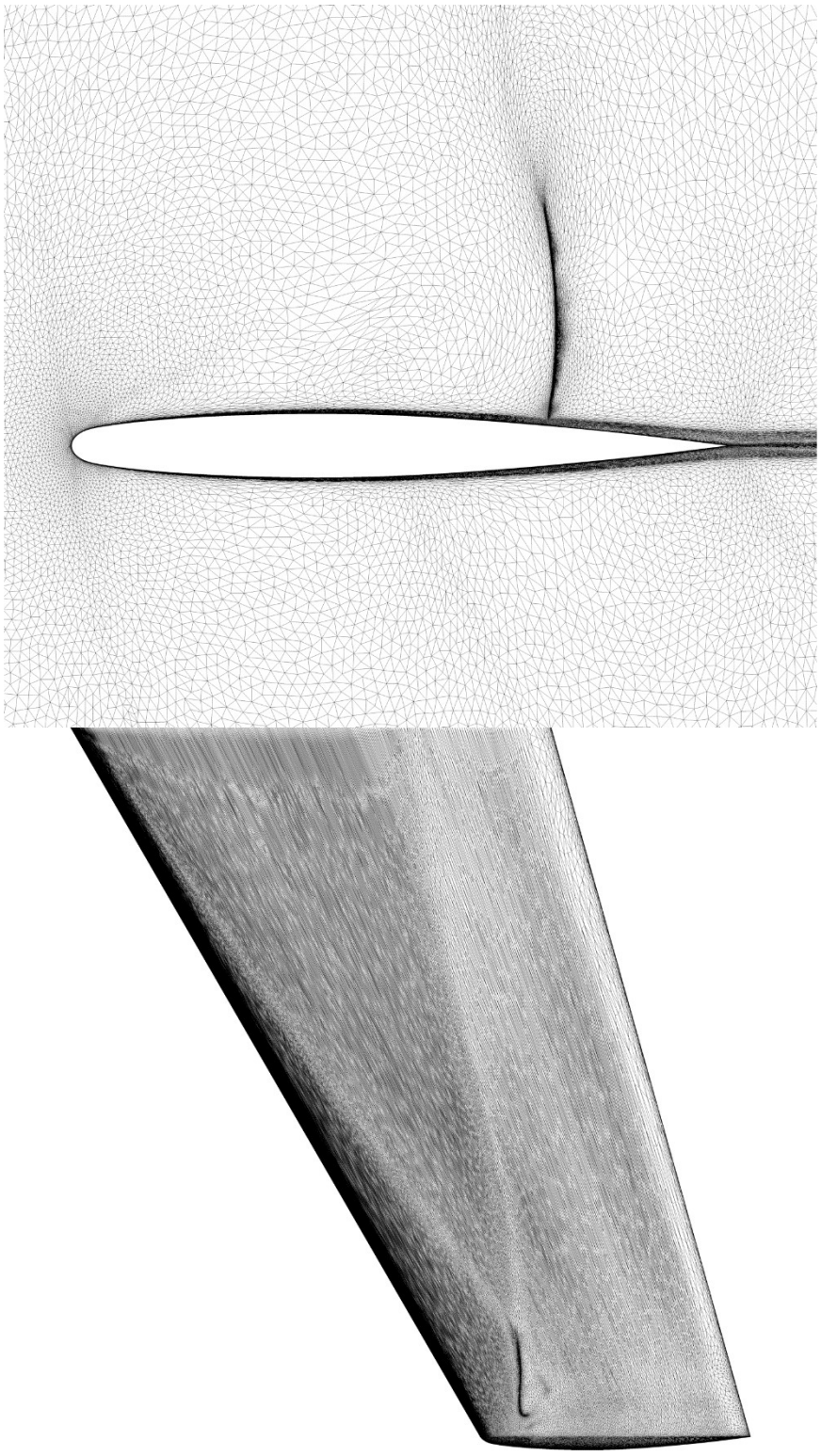

Figure 9: The symmetry plane (top) and upper surface (bottom) meshes for the ONERA M-6 illustrated the results of anisotropic adaptation.

D. DLR-F6 - Linear Anisotropic Adaptation

The FUN3D tool chain is used to adapt the DLR-F6 [21]. The flow conditions are 0.75 Mach, 0° angle of attack, 12.2k unit Reynolds number per mm, and 300° Kelvin. The initial mesh (not shown) is provided by Pointwise without boundary layer refinement. A coarse, curvature resolving, inviscid mesh is adapted from the Pointwise mesh by using the process described in Ref. [12]. After 37 cycles of flow solution and mesh adaptation, the surface pressure coefficient is shown in Figure 10 with details of the surface mesh and symmetry plane mesh. The thin boundary layer is shown surrounding the nose on the symmetry plane. Details of the nacelle trailing edge and horizontal tail attachment plane in the tail are captured with fine anisotropic surface meshes.

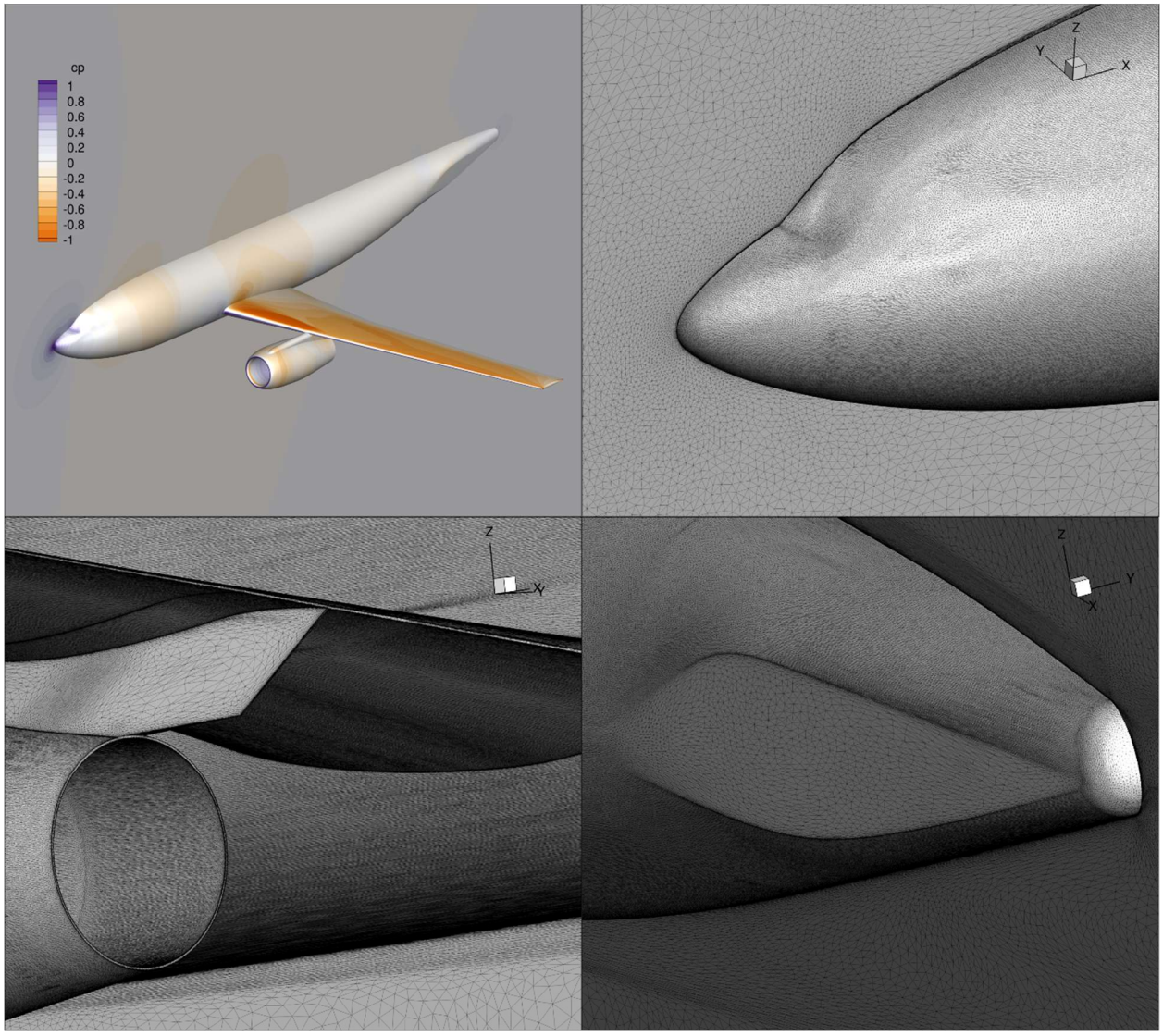

Figure 10: A mesh for the DLR-F6 Mesh was adapted by the FUN3D tool chain. Shown clockwise from upper left are pressure coefficient, nose, tail, and nacelle trailing edge.

A detailed image of the DLR-F6's tail is shown in Figure 11 to illustrate the tolerances used to organize the geometry model's surfaces into quilts (groups of trimmed surfaces in the geometry model to be meshed with a single surface mesh). The dot product of the discrete surface triangle normal (mesh) and the underlying geometry model surface is shown in color with spheres indicating the affected vertices.

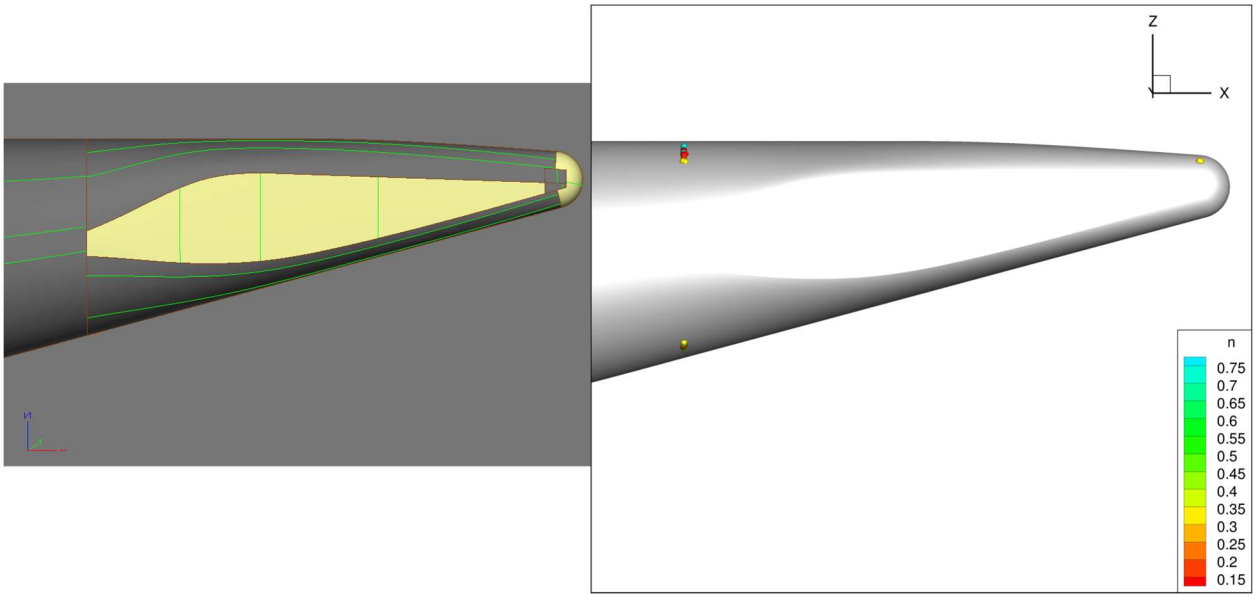

Figure 11: A quilt (left) was used to group the geometry model surfaces on the DLR-F6 tail (boundaries shown in green) for meshing as a single surface. The colored dots (right) indicate where the normal vectors of the mesh and geometry model differ substantially.

The color scale is trimmed above 0.75 in Figure 12 (right), where the discrete triangle provides an accurate representation of the underlying surface normal. There is a step in height between the underlying surfaces that compose the one-to-many collection of faces into a sheet. The height of this step (shown in the surface-tangent view in the right subfigure) is on the same order of magnitude as the viscous normal spacing of the first volume cell adjacent to this surface mesh. The underlying step exploits a weakness in the *refine* mesh adaptation tool and research is underway to accommodate boundary representation tolerances larger than required viscous normal spacing, which is common in practical mechanical CAD models that are focused on required manufacturing tolerances, not required analysis tolerances. In this example, *refine* is creating poorly shaped cells in the region of this step that may adversely impact the convergence of the flow solver. Accommodating boundary representation tolerances was identified as an important research topic in [22].

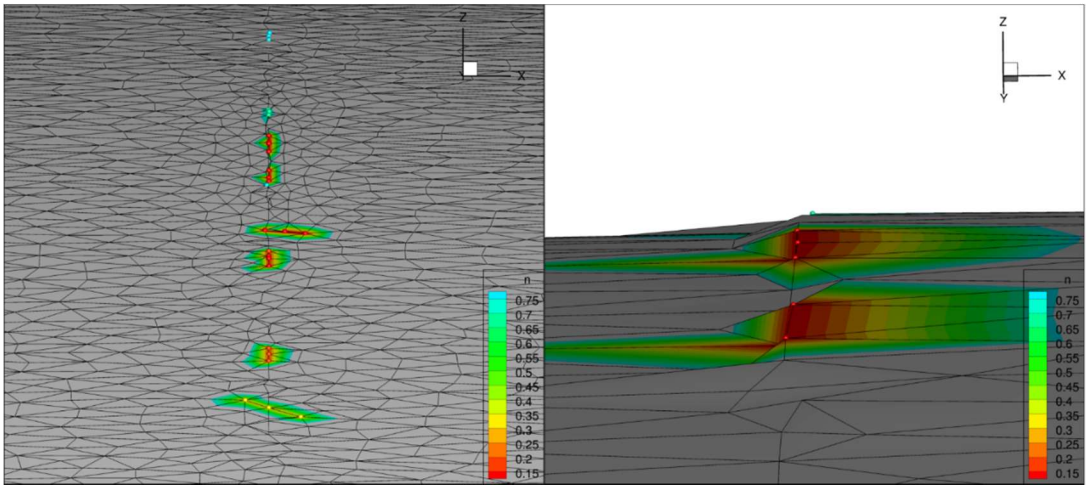

Figure 12: Detailed view of the mesh on the DLR-F6 tail with normal vector deviations between the mesh and geometry model shown in color (left). The view on the right shows a step in the geometry model due to the model's boundary representation tolerance that the mesh is resolving.

E. DrivAer Sedan - Linear Anisotropic Mesh Adaptation

The FUN3D tool chain is used to adapt the DrivAer [23]. The flow conditions are 0.116 Mach, 0° angle of attack, 2.5k unit Reynolds number per mm, and 273° Kelvin. FUN3D has the ability of specifying wall velocity to model a moving ground plane and spinning wheels, but the iterative convergence of the residual with nonzero velocity boundary conditions diverged. This example uses zero velocity on all solid walls (car body, wheels, and ground plane). The initial mesh shown in Figure 13 uses the procedure from Ref. [12] to satisfy a surface curvature meshing constraint.

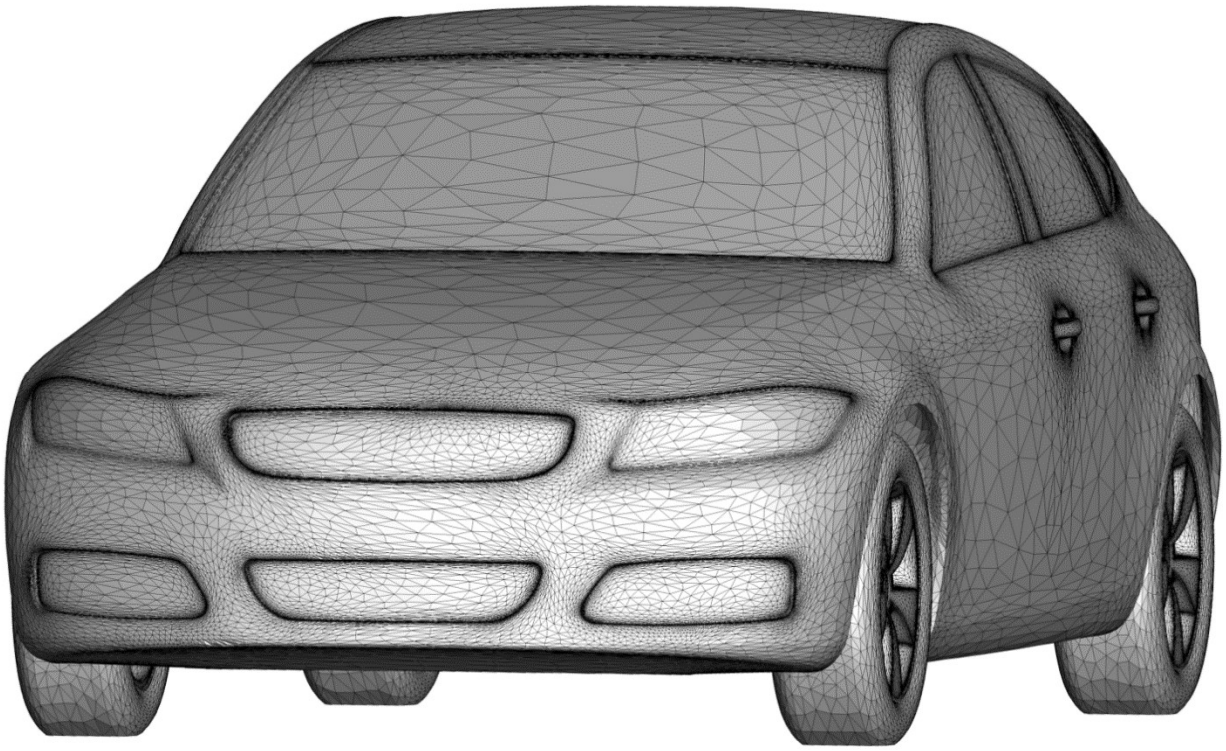

Figure 13: Initial surface mesh of the DrivAer adapted to surface curvature.

The adapted surface mesh and volume mesh intersected with the symmetry plane are shown in Figure 14. Both the right and left halves of the sedan are modeled, but the left half is not visible behind the intersected symmetry plane. Coefficient of pressure and mesh are shown on the right half of the sedan, ground plane, and symmetry plane in Figure 14. The boundary layer on the roof of the vehicle separates and the imprint of a shear layer is seen in the symmetry plane intersected wake mesh. The imprint of the front and rear tire wakes is seen in the ground plane mesh.

The underlying DrivAer geometry model has large boundary representation tolerances where the spokes of the wheel meet the hub. The interpretation of MeshLink-Geode inverse evaluation results by *refine* had to include the accommodation of these large boundary representation tolerances. While the underlying surfaces and curves could be modified to improve boundary representation tolerances, the current level of boundary representation tolerances is very typical. Improved mitigation strategies are being developed in *refine* to reduce or eliminate manual intervention during the import of CAD models. This MeshLink one-to-many representation has discontinuous slopes where multiple faces construct a sheet. The normal deviation limit introduced in the DLR-F6 discussion seems to accommodate the discontinuous slope, but more development in *refine* may be necessary to push the adaptive mesh to finer resolution of this model.

Figure 14: FUN3D adapted surface and volume mesh intersected with the symmetry plane and coefficient of pressure.

VIII. Conclusion

MeshLink is an on-going project to define and deliver mesh-geometry associativity. MeshLink consists of a schema and an XML implementation that defines how mesh elements are associated to specific geometry entities in a geometry model. MeshLink also includes a high-level API that simplifies flow solver authors' geometry evaluations in a kernel-agnostic manner. Both the schema and API are freely available from Ref. [10].

Examples are provided to show the application enabled when mesh-geometry association information is provided and persisted. These examples include mesh adaptation to control geometric deviation of the discrete mesh and continuous geometry model, mesh curving for h-p mesh adaptation, and solution-driven anisotropic mesh adaptation. The wing, wing-body, and sedan models varied in complexity. Initial results are very promising and have enabled the identification of research investments for the applications. These investments include mitigating large boundary representation tolerances and handling height and slope discontinuities internal to one-to-many geometry collections. Interactions between the applications, MeshLink, and Geode have improved all tools and helped to harden these toolchains to typical CAD construction artifacts.

The development of the MeshLink API and example applications contribute to the goals of the CFD 2030 Vision Study by directly addressing the concern of "inadequate linkage with CAD". While some of the applications described in this paper need additional development to meet the robustness intent of the Vision Study (particularly for imported mechanical CAD models), the possibilities of direct linkage with CAD is clearly shown. MeshLink will likely enable applications not considered in this paper. Accommodating the typical features present in imported geometry models (e.g., quilting inconvenient topology, accepting relatively large boundary representation tolerances) will contribute to the automation intent laid out in the CFD 2030 Vision Study.

Acknowledgments

The work described herein was funded partially by the U.S. Air Force (Computational Geometry Kernel Support, Contract Number: FA9101-18-P-0042). The work of the second author was partially supported by the Transformational Tools and Technologies (TTT) Project of the NASA Transformative Aeronautics Concepts Program (TACP) under the Aeronautics Research Mission Directorate. Images used herein were made using the Pointwise and Tecplot 360 EX® software. Product names used herein are for reference only and do not constitute an endorsement. All registered and unregistered trademarks used herein are property of their respective owners.

References

1. Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., & Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," NASA CR-2014-218178, March 2014.
2. "Parasolid," URL: https://www.plm.automation.siemens.com/global/en/products/plm-components/parasolid.html, Siemens Digital Industries Software, [retrieved 16 Oct 2020].
3. "Project Geode: Geometry for Simulation," URL: https://www.pointwise.com/geode/, Pointwise, [retrieved 16 October 2020].
4. "Open CASCADE Technology," URL: https://www.opencascade.com/open-cascade-technology/, Open CASCADE, [retrieved 16 October 2020].
5. Haimes, R. and Dannenhoffer, III, J. F., "EGADSlite: A Lightweight Geometry Kernel for HPC," AIAA Paper 2018–1401, 2018.
6. Chawner, J., & Wyman, N., "Computational Geometry and Mesh Associativity for CFD Software: MeshLink," 28th International Meshing Roundtable, October, 2019.
7. Pointwise, CFD mesh generation software, Version 18, URL: https://www.pointwise.com/pointwise/, Pointwise, [retrieved 16 October 2020].
8. "W3C XML Schema Definition Language (XSD) 1.1. Part 1: Structures," URL: https://www.w3.org/TR/xmlschema11-1/, April 2012, [retrieved 16 October 2020].
9. "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes," URL: https://www.w3.org/TR/xmlschema11-2/, April 2012. [retrieved 16 October 2020].
10. MeshLink API, URL: https://www.pointwise.com/meshlink/doc, [retrieved 16 October 2020].
11. Park, M. A., "Anisotropic Output-Based Adaptation with Tetrahedral Cut Cells for Compressible Flows," Ph.D. thesis, Massachusetts Institute of Technology, Sept. 2008. doi:1721.1/46363.
12. Park, M. A., Kleb, B., Jones, W.T., Krakos, J.A., Michal, T., Loseille, A., Haimes, R., & Dannenhoffer, J.F. III, "Geometry Modeling for Unstructured Mesh Adaptation," AIAA paper no. 2019-2946, June 2019.
13. libMeshb, URL: https://github.com/LoicMarechal/libMeshb, [retrieved 16 October 2020].
14. "The Apache Xerces Project," URL: https://xerces.apache.org/, January 2020, [retrieved 16 October 2020].
15. Biedron, R. T., Carlson, J.-R., Derlaga, J. M., Gnoffo, P. A., Hammond, D. P., Jones, W. T., Kleb, B., Lee-Rausch, E. M., Nielsen, E. J., Park, M. A., Rumsey, C. L., Thomas, J. L., Thompson, K. B., and Wood, W. A., "FUN3D Manual: 13.6," NASA TM-2019-220416, Langley Research Center, Oct. 2019.
16. Spalart, P. R., Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," La Recherche Aerospatiale, Vol. 1, 1994, pp. 5-12.
17. Mayeur, J., Dumont, A., Destarac, D., & Gleize,V., "RANS Simulations on TMR 3D Test Cases with the ONERA elsA Flow Solver," AIAA Paper 2016–1357, 2016.
18. SU2, CFD flow solver software, Version 7.0.07, URL: https://su2code.github.io/, SU2 Foundation, [retrieved 16 October 2020].
19. Wyman, N. J., Galpin, P., Hansen, T., and Scheuerer, G., "Robust, Efficient and Accurate Mesh Adaptation for Turbomachinery CFD Simulations," AIAA paper no. 2020-3688, August 2020.
20. Karman, Steve L. Jr., "Mixed-Order Curving for Viscous Meshes," AIAA paper 2019-3317, June 2019.
21. Brodersen, O., and Sturumer, A., "Drag Prediction of Engine-Airframe Interference Effects using Unstructured Navier-Stokes Calculations," AIAA Paper 2001-2414, 200.
22. Park, M. A., Krakos, J. A., Michal, T., Loseille, A., & Alonso, J. J., "Unstructured Grid Adaptation: Status, Potential Impacts, and Recommended Investments Toward CFD Vision 2030," AIAA paper no. 2016-3323, June 2016.
23. Heft, A. I., T. Indinger, T., & Adams, N. A., "Investigation of Unsteady Flow Structures in the Wake of a Realistic Generic Car Model," AIAA Paper 2011-3669, 2011.