

# GRIDEX – AN INTEGRATED GRID GENERATION PACKAGE FOR CFD

William T. Jones\*  
NASA Langley Research Center  
Hampton, VA 23681-2199  
w.t.jones@larc.nasa.gov

## ABSTRACT

An integrated tool developed for the construction of unstructured numerical grids about complex configurations is presented. The tool is highly integrated and employs the native underlying geometry modeling kernel used to define the target domain for providing topological and geometric access used by the grid generation procedures. This access greatly reduces the overall process time required to generate grids for complex models. In addition, the tool is based on an underlying framework that enables the integration of new grid generation technology as it becomes available. The GridEx package presented herein is under development at the NASA Langley Research Center as part of the Fast Adaptive Aerospace Tools initiative.

## INTRODUCTION

The Fast Adaptive Aerospace Tools (FAAST) initiative at the NASA Langley Research Center is targeted at developing fast adaptive methods for the analysis and design of complex aerospace configurations in all speed regimes<sup>1</sup>. One of the major components of this process is the initial grid generation for complex configurations and the adaptation of those grids to specified global error tolerances. Key to the automation of the grid generation and adaptation procedures is the use of unstructured grid generation techniques and the incorporation of Computer Aided Design (CAD) models.

At the core of the FAAST research is the goal of streamlining the overall numerical simulation process. Unstructured grid generation techniques provide a means of generating high quality discretizations for complex domains in a nearly automated fashion<sup>2-5</sup>. In addition, unstructured grids facilitate localized

refinement, coarsening, and topology changes such as those resulting from solution based adaptation. With these advantages in mind, development of highly automated grid generation and adaptation tools tends to focus on unstructured techniques. However, unstructured techniques alone are not sufficient to optimize the overall analysis process.

With the ever increasing desire to expand the role of numerical simulation in the overall production process, it is necessary to leverage the existing tools and capabilities used in other areas during the design of a new product. The use of production CAD models as the basis for the simulation is one such technique. As the targets of simulation become increasingly more complex, lower fidelity descriptions no longer suffice. With increasing complexity comes the growing desire to reuse the CAD descriptions created during other phases of production. It is therefore desirable to link the grid generation and solution based adaptation directly to the subject CAD model. It is typical to exchange CAD data via standardized file formats such as the Initial Graphics Exchange Specification (IGES) or the Standard for the Exchange of Product Model Data (STEP). However, it must be realized that often the CAD model has been designed for manufacture rather than for analysis. For example, some geometric features, irrelevant to the analysis stage, may need to be suppressed or eliminated from the model. Unfortunately, this potential need for adjustment to the model makes standardized data exchange mechanisms less desirable. Instead it is most beneficial to interface directly to the originating modeling kernel used to construct the subject part. The latter method also provides the ability to leverage the model design intent (master model access, feature suppression, etc.). In practice however, NASA must contend with a variety of diverse commercial CAD systems as part of its technology assessment and problem solving role and therefore cannot tailor its analysis capability to a given system. We require compatibility internally within NASA as well as with our partners in industry and academia. Vendor independent access to numerical geometry is therefore a requirement for the tools and techniques developed as part of the current work.

---

\*Computer Engineer, Data Analysis and Imaging Branch

Copyright © 2003 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for Governmental Purposes. All other rights are reserved by the copyright owner.

Though the field of unstructured grid generation has produced a number of tools capable of creating high quality grids for complex configurations, the field is far from mature. As such, one of the design goals for the current work was to allow for the extension of the resulting application to track technological advances in the field. To satisfy this goal, an Application Programming Interface (API) for unstructured grid generation was developed as the basis for the GridEx package<sup>6</sup>. The API provides a generic software interface to relevant geometry, grid metric, and meshing component routines. The use of the API serves to insulate the application from the specific algorithmic details. Therefore, the implementation of a given software interface can be modified to accommodate technological improvements without impacting the overall application. In addition, multiple implementations for the interfaces can be supplied to the application offering the ability to do side-by-side comparisons of different algorithms, and combinations thereof, within a given session.

What follows is a description of the development and operation of GridEx along with examples of unstructured grids suitable for use in high Reynolds number viscous Computational Fluid Dynamics (CFD) simulations.

## **DESIGN AND DEVELOPMENT**

The design of the GridEx application was centered on usability and extensibility. The package is a product of the FAAST element of the Airframe Systems Concept to Test (ASCoT) and 3<sup>rd</sup> Generation Reusable Launch Vehicles (RLV) programs of the NASA Langley Research Center. One of the key components to the FAAST effort is the rapid generation and adaptation<sup>10</sup> of numerical grids directly from CAD models. The grid generation and adaptation capabilities are required to be independent of the originating CAD system, thereby providing support across the multitude of available systems. In addition, the capabilities must be implemented in a manner which facilitates the infusion of new techniques that result from FAAST research efforts.

To meet these needs, the GridEx application is based on a framework that is implemented by a suite of libraries tailored to the field of numerical mesh generation. The framework is also used to tag the entities of the CAD model with application and analysis specific information. As such, the grid is inherently associated to the defining geometry from the point of creation. Downstream processes in the analysis have the

advantage of this association when they access the geometry or the tagging data. The result is a history of the analysis that is centered on the geometry model of interest. For example, a mesh adaptation package can obtain not only the original grid, but also the underlying geometry to which it is associated. Node movement or the addition of new nodes required by the adaptation can be accurately computed using the actual geometry. Similarly, the geometry can be tagged with the adapted mesh for later use. The framework also utilizes an unstructured grid generation API providing the ability to seamlessly incorporate enhancements in unstructured meshing techniques for tailored use by the FAAST team. The API facilitates the inclusion of emerging technological advances in unstructured methods over the life of the FAAST effort and beyond.

The unstructured grid generation API is defined with a set of functions specific to the task at hand. These include the discretization of 1-dimensional edge, 2-dimensional face, and 3-dimensional volume entities. The API also provides a generic method of determining spacing constraints to be imposed on the resulting mesh. This method provides three principal spacing values and directions for any location in the computational domain. Also, included in the API are logistical functions to provide: a unique identifier for the implementing algorithm; capabilities of the algorithm (edge, face, volume, etc.); and the ability to communicate meshing progress metrics. Listing the capabilities of an implementation provides the opportunity to satisfy by other means any functionality not provided by the given algorithm. If an algorithm only supports a portion of the API, only that portion need be implemented and the capabilities list is reported accordingly. Implementation of missing capability will be provided by other supporting algorithms if possible. Access to the geometrical information of the target model is also provided by an API which will be discussed in a subsequent section.

The API definition is currently restricted to triangular surface and tetrahedral volume mesh generation. However, the definition is independent of any specific algorithm and may be implemented in any manner consistent with the functional definition. For example, the meshing functions may be implemented with Delaunay or advancing front techniques; the spacing constraints may come from a background grid/source analogy, from analytic functions, or from some solution derived quantity. Regardless of the implementation, data is exchanged through the API making possible the transparent substitution of data construction methods.

The implementation of the unstructured grid generation API functions used by GridEx is provided via Dynamic Shared Objects (DSOs) alternatively referred to as Dynamically Linked Libraries (DLLs). One or more DSOs can be loaded at execution providing runtime customization of the capabilities made available to the user by the GridEx package. The user controls the list of available shared objects via an environment variable that is processed at runtime and used to load and assess the specified libraries. As part of the assessment process, the Graphical User Interface (GUI) is dynamically updated and the DSO list is traversed in order to resolve any missing capabilities. If the selected DSO lacks a particular capability, the first object that provides the missing functionality is used to resolve it. An example would be a DSO that implements an algorithm that only provides a volume meshing capability. In this case when the lacking DSO is selected, the list is traversed to locate, from the remaining DSOs, an object that provides edge and face meshing. That implementation is then used as the default algorithm for edge and face meshing when and if needed by the “volume only” algorithm. Similarly, the first available implementation is used as a default for missing DSOs when restarting a session. In the latter case, if a previous session depended on a DSO not currently loaded, the “default” is used and an appropriate warning is posted to the user.

The API definition is detailed in Reference 6, but is included here for completeness. It should be noted that the implementations of the API used by GridEx are assumed to be written in the C programming language. Therefore, all of the descriptions to follow assume the C language. In practice it is possible to support existing algorithm implementations written in other languages such as FORTRAN77 and FORTRAN90. This is accomplished by assembling a DSO consisting of C implementations of the API which call routines written in other languages. This places the burden of mixed language programming on the DSO developer, but keeps the calling sequence consistent for GridEx. The unstructured grid API follows as:

```
UG_MeshEdge(v,e,b,u,m,c,p)
  int    v      - Target region
  int    e      - Target edge entity
  double b[3]   - Bounding points
  double u[2]   - Bounding parameters
  int    *m     - Number of computed nodes
  double *c     - Output physical coords
  double *p     - Output parametric coords

  Purpose: Discretize an edge entity
```

```
UG_MeshTriFace(v,f,n,x,u,nb,b,ns,s,m,ne,e,c,p)
  int    v      - Target region
  int    f      - Target face entity
  int    n      - Number of fixed nodes
  double *x     - Fixed node coords
  double *u     - Fixed node parameters
  int    nb     - Number of boundary segs
  int    *b     - Oriented Boundary segs
  int    ns     - Number of Interior segs
  int    *s     - Interior segments
  int    *m     - Number of computed nodes
  double *ne    - Num of computed elements
  int    **e    - Output element defs
  double **c    - Output physical coords
  double **p    - Output parametric cords

  Purpose: Triangulation of a face entity
```

```
UG_MeshTetVolume(v,n,x,l,t,nf,f,ns,s,m,ne,e,c)
  int    v      - Target region
  int    n      - Number of fixed nodes
  double *x     - Fixed node coords
  int    *l     - Number of shell elements
  int    **t    - Shell elements
  int    nf     - Number of interior faces
  int    *f     - Interior triangular faces
  int    ns     - Number of interior segs
  int    *s     - Interior segments
  int    *m     - Number of computed nodes
  double *ne    - Num of computed elements
  double **e    - Output element defs
  double **c    - Output physical cords

  Purpose: Volume tetrahedralization
```

```
UG_GetSpacing(x,y,z,s,dir)
  double *x     - Target X coordinate
  double *y     - Target Y coordinate
  double *z     - Target Z coordinate
  double s[3]   - Edge lengths
  double dir[9] - Principal directions

  Purpose: Determine spacing constraints given a
           target location in the domain
```

Note: Arguments representing the target coordinate for UG\_GetSpacing() are passed by reference to facilitate a call from FORTRAN code.

Unless otherwise stated, all functions return an integer value of 0 on success and 1 on failure. All arrays are 1-dimensional. Coordinate arrays are ordered such that the stride of the array is the space dimension (i.e. for a Cartesian array, ordering is  $x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_n, y_n, z_n$ ). Likewise, element arrays are listed with a stride

equal to the number of nodes (i.e. triangle array  $n_{1,1}$ ,  $n_{1,2}$ ,  $n_{1,3}$ ,  $n_{2,1}$ ,  $n_{2,2}$ ,  $n_{2,3}$ , ...,  $n_{n,1}$ ,  $n_{n,2}$ ,  $n_{n,3}$ ). Allocation of arrays is handled by the API implementation and it is expected that GridEx can freely release these memory resources as necessary.

The following functions must be implemented to provide logistical information about the DSO. Each DSO is assigned a unique identity represented as an integer. DSO developers are encouraged, but not required, to verify the uniqueness of their respective identity with the GridEx developers. While failure to do so may prevent interoperability with other DSOs, it will not prevent the use of a DSO within GridEx. Failure to provide a unique identity however could potentially encounter collisions with other production and development DSOs. Such collisions may or may not affect development and should be noted by DSO developers.

```
int UG_Identity(void)
    Returns an unique integer Id of the
    algorithm/DSO

int UG_WhatProvides(void)
    Returns an packed integer that details
    the functionality provided by the DSO
```

The progress function described earlier is provided by the GridEx package and may be used by an API implementation to transfer information to and from GridEx. When called the API will supply a progress metric (percent complete) which will be displayed by the GridEx progress bar. GridEx will return a value of zero if the caller should continue. A returned value of 1 signals an interrupt whereby the caller is expected to perform local clean up followed by an error return.

```
int UG_Progress(metric)
    double *metric - Current progress metric
```

Again, the argument is passed by reference to facilitate calls from FORTRAN routines.

The above functions can be implemented and used to build custom DSOs by a GridEx user. This allows the user a great deal of flexibility to expand upon the methods of operation provided by the standard GridEx distribution. It also allows easy incorporation of technological breakthroughs into the production environment. As opposed to a fixed procedure, GridEx provides a matrix of capabilities with combinations defined by the supporting DSOs selected at runtime.

## GEOMETRY ACCESS

The current work employs the Computational Analysis Programming Interface (CAPrI)<sup>7,8</sup> as the basis for geometry access. CAPrI is a CAD-vendor neutral API providing the reduced set of solid modeling operations common to computational analysis. It accesses computational solid geometry related information directly from the kernel of the originating CAD system. The CAPrI API offers a layer of abstraction from the specific methods of a given CAD kernel's API while ultimately utilizing the original system used to create the subject geometry. Applications derived from CAPrI, however, are shielded by the API from the specifics of the underlying modeling kernel. They automatically may employ any of the of supported CAD systems without modification of the application itself. Support for additional modeling systems is provided to all derivative applications by the creation of a new CAPrI driver for the desired modeling kernel. This level of abstraction is in direct alignment with the development of the GridEx application. The single GridEx source code will support, transparent to the user, any and all of the CAD kernels supported by the CAPrI API. The development of new CAPrI drivers will allow GridEx to effortlessly track the changes, enhancements, and developments of the CAD industry.

The CAPrI API provides operations that are common across the supported systems and provides for interrogation, data tagging, and the creation of solid primitives. Since it is actually implemented with the native modeling kernel of the subject part, CAPrI also provides access to the master model of the part allowing for feature suppression, parameter modification, regeneration, etc. CAPrI operations are restricted to manifold solid geometry, such as that defined by most modern CAD systems, and as such provides a closed topological description of the domain of interest. CAPrI also provides a closed tessellation<sup>9</sup> of the subject part that may be used to ensure physical consistency of the model. Therefore, inherent in the design of CAPrI, all of the geometric and topological information required for intelligently automated unstructured grid generation is available for use by derivative applications.

An additional layer of abstraction is used by the unstructured grid generation API to encapsulate the geometry operations such that they may be replaced or enhanced as directed by future needs. One such change might be to support geometry-only definitions. A primary example of such a definition is that of legacy IGES data. This type of data would require

combination with a separate description of the topology for use in automated grid generation and as such does not fit into the current design goals of the CAPrI API.

The following documents a portion of the abstraction layer to the CAPrI API. Only the primary methods typically required by the methods of the unstructured grid generation API are included. Input and Output of geometry as well as other topological and logistical operations are handled internally by GridEx and are not documented here for the sake of brevity.

```
CADGeom_LengthOfEdge(v, e, ts, te, l)
  int    v    - Target region
  int    e    - Target edge entity
  double ts   - Starting parameter
  double te   - Ending parameter
  double *l   - Output length
```

Purpose: Determine bounded length of Edge

```
CADGeom_PointOnEdge(v, e, t, pt, d, d1, d2)
  int    v    - Target region
  int    e    - Target edge entity
  double t    - Target parameter
  double *pt  - Output coordinate
  int    d    - Derivative flag
  double *d1  - Output 1st derivative
  double *d2  - Output 2nd derivative
```

Purpose: Evaluate a parameter on the Edge

```
CADGeom_NearestOnEdge(v, e, po, t, pt)
  int    v    - Target region
  int    e    - Target edge entity
  double *po  - Target coordinate
  double *t   - In/Output Edge parameter
  double *pt  - Output Edge coord
```

Purpose: Snap a point to the Edge

```
CADGeom_PointOnFace(v, f, uv, pt, d, du1, dv1,
                    duv, du2, dv2)
  int    v    - Target region
  int    f    - Target face entity
  double *uv  - Target UV parameters
  double *pt  - Output coordinates
  int    d    - Derivative flag
  double *du1 - Output 1st U derivative
  double *dv1 - Output 1st V derivative
  double *duv - Output mixed derivative
  double *du2 - Output 2nd U derivative
  double *dv2 - Output 2nd V derivative
```

Purpose: Evaluate a point on the Face

```
CADGeom_NearestOnFace(v, f, po, uv, pt)
  int    v    - Target region
  int    f    - Target edge entity
  double *po  - Target coordinates
  double *uv  - In/Output Edge parameter
  double *pt  - Output Edge coord
```

Purpose: Snap a point to the Face

```
CADGeom_NormalToFace(v, f, uv, pt, n)
  int    v    - Target region
  int    f    - Target face entity
  double *uv  - Target parameters
  double *pt  - Output coordinates
  double *n   - Output normal
```

Purpose: Determine normal of a Face

Coordinates are ordered X, Y, Z and surface parameters are ordered U, V. Where applicable a flag is specified to control derivative calculation as follows: 0 – no derivative; 1 – first derivative; 2 – first, second, and mixed derivatives. Snap methods require an initial parameter estimate as input.

## DATA PERSISTENCE

Primarily, each function listed above is a simple wrapper of the equivalent CAPrI function. However, the GridEx framework makes internal use of an application I/O mechanism provided by CAPrI to save analysis data in a “Geometry Centric” fashion.

Currently CAPrI saves a separate file in addition to the native CAD part file. This file contains auxiliary information needed by CAPrI. The data is stored separately so as to prevent modification of the part that would hinder loading of that file back into the respective CAD system.

The CAPrI I/O mechanism provides the GridEx software developers the ability to augment the auxiliary file with application specific information. Information local to a specific region is differentiated from information global to the application (i.e. all regions). A simple method of registering applications with CAPrI is supplemented with callback functions that are invoked by each CAPrI save operation. With each invocation, the user supplied callback function is provided: the target region identifier; the target application name; and a file pointer to which the developer can safely write data. For reading application data, a function is provided to return a file pointer that is positioned appropriately to read the

specified application data. The number of bytes written is also provided as a checksum. A function is also available to return the number and names of the applications that have stored data within a given CAPRI auxiliary file. It is the responsibility of GridEx to ensure the integrity of the read and write operations.

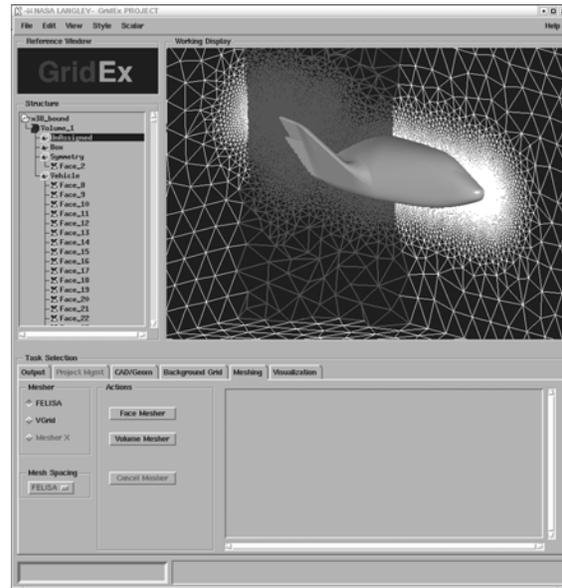
The GridEx framework internally uses this mechanism to save application data for session restarts. As a consequence, all data gathered and generated by GridEx is available to downstream applications. This method of tagging the geometry with analysis data maintains a geometry based coupling between the various disciplines within the overall design/analysis process.

### **GRIDEX OPERATION**

Within GridEx, the user may interactively: define the domain(s) of interest surrounding the subject geometry; impose grid metric constraints to govern the distribution of discrete grid points and the resulting element quality; individually or collectively generate surface grids for the constituent faces of the solid model; generate volume grids for the domain(s) of interest; and visualize the results. Each phase of the grid generation procedure is organized on a task oriented tabbed form located on the GUI as shown in Figure 1. The GUI also includes a 3D view of the problem space that can be manipulated interactively. A model tree is provided for the hierarchical organization of the problem to include boundary condition definition for output to the analysis software. Additional operational functionality is provided by means of a standard application menu bar.

In addition to the API basis of the underlying framework used to construct GridEx, one of the unique characteristics of the tool is the interaction between grid metric constraint specification and grid generation. The tool allows the user to specify grid metric constraints via the method of choice by setting appropriate parameters on the tabbed form. At any time during the specification, the user may elect to view the localized impact of the constraints on one or more selected surface meshes. The inspection requires the grid to be generated for the subject face(s) and any of the respective component edges. The grid generation is limited to those entities specified by the user and is thus very efficient. This process however may result in inconsistencies in the surface grid as faces fall out of sync with their neighbors. To allow for this iterative flexibility, a mechanism for automated consistency is built into the application via the support framework. The method is summarized as follows. The framework

maintains timestamps on the constraints and on each individual component grid (edge, face, and volume). As such the grids are aware of their state relative to the constraint specifications. As a given face grid is generated, the constituent edges of the face are assembled into the bounding discretization. This process includes a check of the current state of each edge relative to the metric constraints. If an edge mesh



**Figure 1 - GridEx Application**

is out of date, it is discretized with the current constraints before assembly. This operation has the added benefit that edges common to multiple faces are only updated once. Likewise, prior to volume grid generation all component grids are verified against the current state of the constraints. Component grids that are consistent with the constraints remain unchanged. Inconsistent component grids are automatically updated and assembled for use in the volume grid computation. This capability greatly reduces the time required to specify the desired metric constraints and provides flexibility and automatic consistency to the user.

The grid generation methods of GridEx are integrated into the tool through the aforementioned DSOs. This allows the user to iterate through the metric specification process while obtaining visual feedback on the impact to the desired grid with respect to the changing constraints. The integration results in additional time reduction for the overall process.

GridEx provides the ability to define separate grid metric specifications from multiple algorithms. These

can then be individually used in conjunction with the available meshing algorithms to generate unstructured tetrahedral meshes directly from the CAD definition. The manifold solid model access provided by the CAPrI interface allows for automated topology extraction that is used to drive the grid generation procedure. The user is responsible for specifying grid metric constraints prior to meshing and has the ability to iteratively generate grids on individual geometric entities while assessing the local impact of the constraints on the quality of the resulting mesh as stated above.

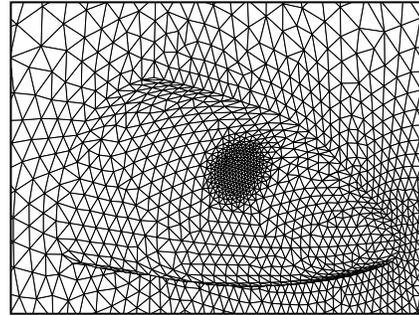
GridEx currently supplies the user with grid metric constraints using the algorithms found in FELISA<sup>3</sup> and VGRID<sup>11</sup>. These algorithms may be used seamlessly and interchangeably between both the FELISA<sup>3</sup> and VGRID<sup>2</sup> surface meshing implementations that are currently provided as DSOs with distribution. These methods can of course be augmented by user supplied DSOs. Supplied meshing implementations only support isotropic surface grid generation however work is in progress to support anisotropic stretching of surface and volume elements. Both meshing implementations have been refactored into a modular set of API conforming libraries for use within the framework. The refactoring also included the use of the API to decouple the meshing algorithms from the underlying geometry and grid metric specification. As a result the framework now provides the stated matrix of capabilities to the GridEx user. The supplied surface meshing matrix is populated as shown in Table 1 and is expected to be expanded in the future to support additional techniques as required.

	FELISA Spacing	VGRID Spacing
FELISA Mesh	X	X
VGRID Mesh	X	X

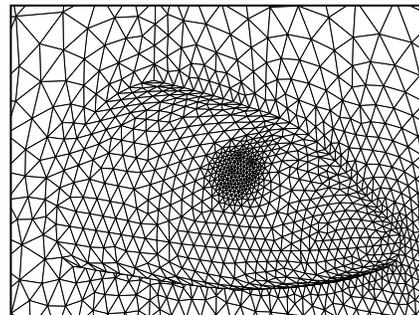
**Table 1 – Surface Grid Generation Capability**

In addition to the surface meshing matrix, volume meshing capabilities supplied add support for the AFLR3<sup>4</sup> viscous volume generator.

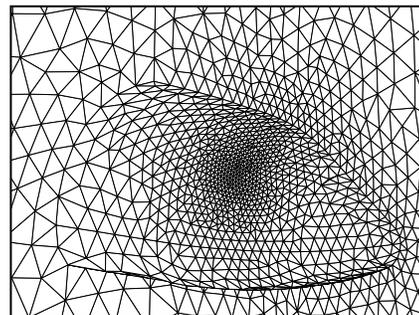
Side-by-side comparison of results from multiple metric/meshing algorithms is available within the same GridEx session. Visualization of grid data is provided in the interactive 3D window. Grid inspection is aided via flooded contour plotting of predefined grid quality measures. These contours can be viewed for any



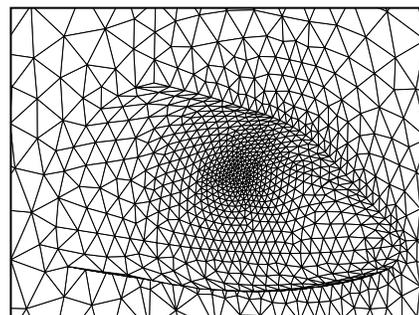
**a) FELISA Mesh from FELISA Background Grid**



**b) VGRID Mesh from FELISA Background Grid**



**c) FELISA Mesh from VGRID Background Grid**



**d) VGRID Mesh from VGRID Background Grid**

**Figure 2 - Surface Grid Generation Capability**

collection of boundary surfaces and/or “crinkle cuts” through the volume grid.

Examples of the flexibility afforded the user by the supplied capabilities is demonstrated in Figure 2. The figure uses surface grids on the nose of a hypersonic vehicle to represent each of the four scenarios supplied for surface grid generation within GridEx. Figure 2a shows the result of a surface grid computed using the FELISA meshing algorithm and a traditional FELISA background grid. Figure 2b shows the same geometry meshed with the VGRID surface meshing algorithm and same FELISA background grid. The background grid used here consists of a line source that extends along the longitudinal axis of the vehicle and a single point source, centered in the clustered region of the figures. The point source was added for the purpose of demonstration. The point source was defined with a constant spacing distance 6 times that of the desired edge length defined for the source. The edge length doubling distance was specified as 10 times the source edge length. These parameters are detailed in the FELISA User's Guide<sup>3</sup>. Figures 2c and 2d show the same progression of meshing algorithm but with a VGRID background grid controlling grid clustering. Similarly, a line source is defined along the longitudinal axis and a point source is defined at the center of the clustered region. The effect of the point source with the VGRID background grid decays smoothly with increasing distance based on a user specified intensity value. Again, the details of the background grid definition are found in the literature<sup>11</sup>.

When comparing surface grids generated with different meshing algorithms but the same background grid in Figure 2, only subtle differences are noted. This is reasonable as both grids adhere to the same metric constraints defined by the background grid. However, it is possible that other cases may yield more drastic differences. Fortunately, the decoupled unstructured grid generation API provides the flexibility to choose the appropriate combination best suited for a particular problem at little or no cost. The reader is reminded that, as this figure demonstrates, future metric specification schemes can be added to the application with no impact to existing meshing algorithms.

A recent enhancement of the GridEx application comes from access to the viscous volume grid generation capabilities of the AFLR3 software<sup>4</sup>. AFLR3 is a standalone volume grid generation package based on the advancing front local reconnection algorithm and is capable of generating inviscid as well as viscous volume grids from an existing boundary triangulation. The tool has the ability to generate fully tetrahedral or mixed pentahedral boundary layer grids.

The API basis of the GridEx application facilitated the integration of AFLR3 resulting in a total time to integrate of less than 12 hours. Modification consisted of creating an API conforming wrapper routine used to invoke the standalone AFLR3 executable. The purpose of the wrapper was to: create a disk file defining the boundary elements obtained through the API along with the associated boundary conditions; generate a script to control AFLR3 execution, also as a disk file; invoke a system call to execute the script; and finally import of the AFLR3 volume grid from the disk file generated by the execution. No refactoring of AFLR3 was possible for inclusion into the current work. As such, use of the API for other algorithms, namely grid metric constraint calculation, could not be accommodated within AFLR3 itself. The definition of grid metric constraints for the volume grid is confined to that set by the AFLR3 application. The schemes available are based on interpolation, with various forms of decay, of the grid metrics derived from the initial boundary triangulation.

Boundary triangulations for AFLR3 are generated using any of the available surface meshing techniques within GridEx and therefore are geometry conforming. In general, no new surface nodes are introduced as part of the volume grid generation. Element connectivity, however, may be altered as a result of local reconnection. Surface element connectivity is updated as part of the volume grid import process. However, viscous cell growth is allowed on planar symmetry surfaces. For these faces, new nodes and connectivity will be generated and both must be updated as part of the import process.

### CUSTOMIZATION

As stated above, the use of an unstructured grid generation API allows the user to customize GridEx operation through the development of meshing "plug-ins". This section serves to describe this process.

In order to facilitate the construction of a user defined DSO, a C language header file with basic definitions is supplied as part of the standard GridEx distribution. This header file lists: the known algorithm identities; masks used for defining and evaluating algorithm capabilities; return code definitions; and the definition of the API interfaces listed above. It is the DSO developer's responsibility to implement the desired interface methods with the algorithm of interest.

All DSO libraries used by GridEx must implement the `UG_Identity()` and the `UG_WhatProvides()` interfaces. The `UG_Identity()` interface should

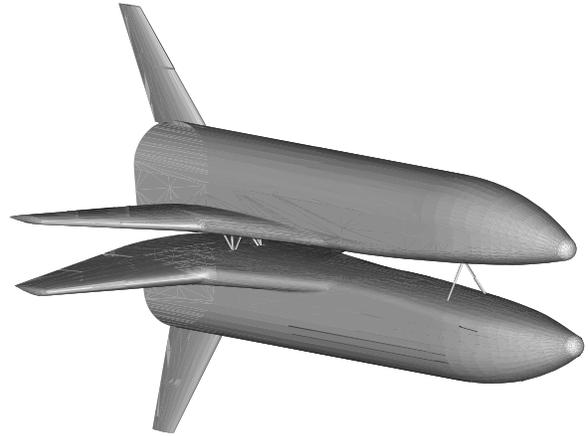
simply return an integer representing the unique identity of the algorithm. During development, the identity is somewhat arbitrary. However, for proper interoperability with the supplied DSOs, it should not duplicate any of the “acknowledged” meshing libraries contained in the header file. Also, as a best practice, the new identity should be communicated to the GridEx developers for inclusion as an “acknowledged” algorithm thereby avoiding any potential future conflicts with other DSOs. To reiterate, it is not necessary to implement all phases of the meshing process with a given algorithm. To provide a means of communicating the specific capabilities of the DSO to GridEx, `UG_WhatProvides()` returns a constant comprised of the bitwise combination of the capability masks supplied in the header. This information will be used by GridEx to substitute the “default” algorithm for the missing functionality as described earlier.

The remaining responsibility of the DSO developer is to supply the desired functionality (as defined by the capabilities mask) by implementing the appropriate interface(s) followed by the building of a shared object library. Use of other API functions is encouraged in order to provide the greatest flexibility to the user. For example use of the `UG_GetSpacing()` interface is favored over a proprietary constraint mechanism in that it affords the user selection of any of the available constraint algorithms. Likewise, the case is made for geometry access and even meshing of entities lower in the solid modeling hierarchy. Details required to build the shared object library are operating system dependent and are therefore not covered here. The reader is referred to the appropriate documentation for the target operating system for a description of that procedure.

Inclusion of the newly created DSO is then conveyed to GridEx by setting an environment variable that defines which DSOs should be used by the GridEx session.

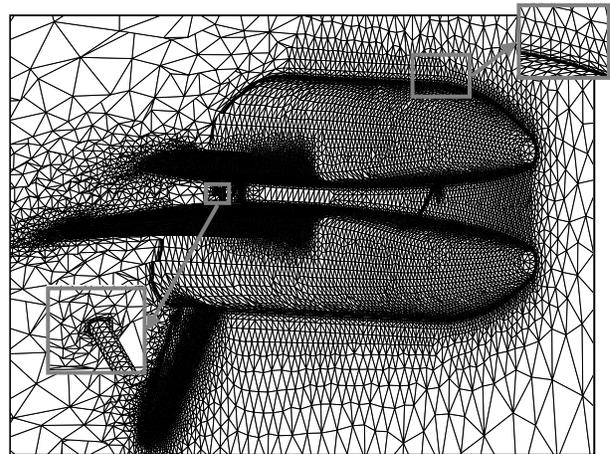
### EXAMPLES

As a demonstration of GridEx capability we now consider the application of the tool to two complex test cases. The first is that of the Langley Glide Back Booster (LGBB) shown in Figure 3. Geometric complexities included in this model involve the struts used to connect the two vehicles as well as cavities along the wing trailing edge that represent gaps between flap surfaces. An example AFLR3 viscous volume grid is shown in Figure 4. The insets show the smooth transition from the semi-structured viscous layers to the inviscid region of the grid. Not shown are



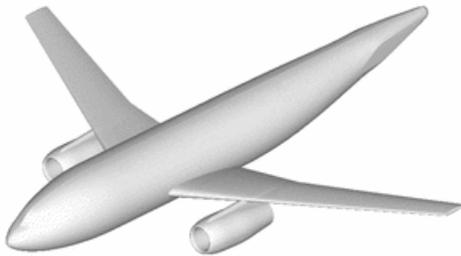
**Figure 3 - Sample Geometry for use with AFLR3**

the cavities representing the juncture of flap geometry. Similar grid quality was obtained in these regions. The boundary triangulation generated for this example was created using the FELISA surface grid generator and a FELISA background grid. The geometry was obtained in the form of a solid model from the Unigraphics commercial CAD system and processed using the associated CAPrI driver. The computational domain was defined by a Boolean subtraction of the geometry from a “Box” solid primitive created within GridEx and assumed half plane symmetry. This operation was carried out within GridEx. The resulting model was defined topologically with 412 edge and 134 face entities. This topological information was automatically extracted during the grid generation procedure. The entire process required less than 4 hours to complete following receipt of the geometry definition.

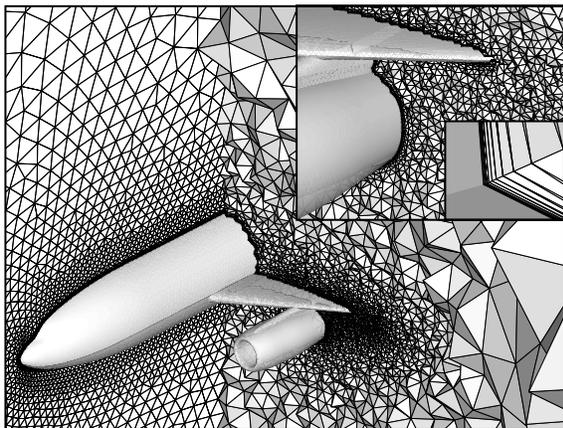


**Figure 4 – Sample Viscous Grid from AFLR3**

Figure 5 shows geometry provided for the 2<sup>nd</sup> AIAA CFD Drag Prediction Workshop. This geometry of the DLR-F6 transport configuration adds a pylon and nacelle to a generic wing/body geometry. The initial surface mesh and inviscid volume grid were generated in just under 1 hour following initial receipt of the geometry in the form of an IGES file. This IGES file was used to create a solid model to include the defining domain of interest using the Unigraphics CAD package and again processed with appropriate the CAPRI driver. Following the generation of the inviscid grid, an AFLR3 viscous grid was generated using the same surface triangulation requiring an additional 20 minutes to compute. This grid is shown in figure 6.



**Figure 5 – DLR-F6 Geometry from 2<sup>nd</sup> DPW**



**Figure 6 – Viscous grid for DLR-F6 Geometry**

## CONCLUSION

A new grid generation application has been presented which is capable of generating grids about complex configurations suitable for use in high Reynolds number viscous CFD. The integrated nature of the GridEx application serves to reduce overall process

involvement providing quality grids in minimal time. GridEx was designed to be extensible so as to track technological advances in the field of computational simulation. A defining component of this design is the use of Dynamic Shared Object libraries to implement the key functionality of the tool. As a result the tool is not only extensible but also customizable by the end user. The application has been described and demonstrated through the construction of complex unstructured tetrahedral volume grids.

## ACKNOWLEDGEMENTS

The author wishes to thank the members of the FFAST team of the NASA Langley Research Center for their assistance and conversations regarding the current work. I also wish to extend thanks to Bob Haines and Dr. Jaime Peraire of MIT for the exchange of ideas and generous support of their respective technologies.

## REFERENCES

- <sup>1</sup>Thomas, J. L., et al, "Opportunities for Breakthroughs in Large-Scale Computational Simulation and Design," NASA TM-2002-211747, 2002.
- <sup>2</sup>Parikh, P., Pirzadeh, S., and Löhner, R., "A Package for 3-D Unstructured Grid Generation," Finite Element Flow Solution and Flow Field Visualization, NASA CR-182090, 1990.
- <sup>3</sup>Peiro, J., Peraire, J., and Morgan, K., "FELISA System Reference Manual and User's Guide, Volume 1," University of Wales Swansea Report, CR/821/94, 1994.
- <sup>4</sup>Marcum, D. L., "Generation of Unstructured Grids for Viscous Flow Applications," AIAA Paper 95-0212, 1995.
- <sup>5</sup>Pirzadeh, S., "Progress Towards A User-Oriented Unstructured Viscous Grid Generator," AIAA Paper 96-0031, 1996.
- <sup>6</sup>Jones, W. T., "An Open Framework for Unstructured Grid Generation," AIAA Paper 2002-3192, 2002.
- <sup>7</sup>Haines, R., "Computational Analysis Programming Interface," Proceedings of the 6<sup>th</sup> International Conference on Numerical Grid Generation in Computational Field Simulations, pp. 663-672, 1998.
- <sup>8</sup>Haines, R., "CAPRI: Computational Analysis PRogramming Interface User's Guide", Massachusetts Institute of Technology, 2001.
- <sup>9</sup>Haines, R., Aftomis, M., J., "On Generating High Quality Watertight Triangulations Directly from CAD," Proceedings of the 8<sup>th</sup> International Conference on Numerical Grid Generation in Computational Field Simulations, 2002.
- <sup>10</sup>Park, M. A., "Adjoint-Based, Three-Dimensional Error Prediction and Grid Adaptation", AIAA Paper 2002-3286, 2002.
- <sup>11</sup>Pirzadeh, S., "Structured Background Grids for Generation of Unstructured Grids by Advancing-Front Method", AIAA Journal 31:2, pp. 257-265, 1993.