API FOR GRID GENERATION OVER TOPOLOGICAL MODELS

S. Gopalsamy, Douglas H. Ross, Alan M. Shih

Enabling Technology Laboratory, Mechanical Engineering Department
University of Alabama at Birmingham, Birmingham, AL., U.S.A
{sgopals, dhross, ashih}@uab.edu

ABSTRACT

Topological representations are being used to define geometric models suitable for grid generation and grid generation tools are being developed that work directly on topological entities. While there are standards for topological representation, there is no standard for Application Programming Interface (API) for grid generation over topological models. This paper describes one such API. It discusses an API for generating various types of grids - edge grids and structured/ unstructured face and volume grids - in the context of a topological representation scheme used in the Geometry and Grid Toolkit (GGTK) developed at University of Alabama at Birmingham (UAB). As an application of the API, this paper proposes an XML schema for specifying desired grids for a topological model. Using XML as a mode of input, one can then develop a web component for grid generation from a grid generator that supports the API, which can be used by remote users to obtain required grids.

Keywords: topological representation, grid generation, grid specification, web component for grid generation

1. INTRODUCTION

Geometric modeling and grid generation are required pre-processing steps for computational simulation of many engineering processes such as Computational Fluid Dynamics (CFD), and Computational Structural Mechanics (CSM). Traditionally, in a typical application, a geometric model will be obtained using a CAD system and grid over the model will be generated using a separate grid generator. It is known that the data transfer between CAD system and grid generator is error-prone and a great deal of time is spent to fix the errors and make the geometric model suitable for grid generation [1,2]. One remedy suggested in [1] is to develop unified geometric modeling and grid generation tools so that both can be done in the same system. The unified approach will supply grid generators with CAD tools that are readily available for all geometry processing, including watertight representation, while avoiding the error-prone data transfer. Accordingly, people are developing such unified systems [3, 4, 5, 6] and developing grid generation tools over them [6, 7, 8]. One common thing that is being adopted in such systems is the use of topological representation [9, 10] in order to define watertight geometric models that are suitable for grid generation.

While there are standards for topological representation, such as STEP-42 [10], there are no standards that specify the following two categories of Application Programming Interface (API) functions:

1. Geometric API functions, in terms of topological entities, needed by grid generators
2. Grid generation API functions, again in terms of topological entities, needed by users to generate various types of grids – structured, unstructured, and generalized face and volume grids - over topological models.

Such standard API functions will help to build standard grid generation tools/ components, which can be widely used, instead of rebuilding for various cases of underlying geometry representation. Development of standard API functions for grid generation has also been proposed by the Unstructured Grid Consortium [11]. While the API functions of this paper are defined over specific data-structures of a topological representation scheme, those of [11] are abstract API without specific data-structures.

This paper discusses the two categories of API functions mentioned above in the context of the topological representation scheme of Chew et al [4] used in GGTK [6]. GGTK is a Geometry and Grid Toolkit being developed

and maintained at the University of Alabama at Birmingham. It is developed as a comprehensive software library consisting of functions for geometric modeling, topological representation, and grid generation. Figure 1 shows various components and functionalities of GGTK.

The rest of the paper goes as follows: Section 2 briefly explains the topological representation scheme and gives a list of API functions needed by grid generators. Section 3 defines grid entities – edge grids, face grids, and volume grids, and gives API functions for generating various types of grids by various methods. They include structured and unstructured grid types and Transfinite Interpolation (TFI), Delaunay and advancing front methods. Section 4 proposes an XML schema to specify grids over a part of or whole model based on the grid generation API functions. Finally section 5 gives conclusions and future directions.

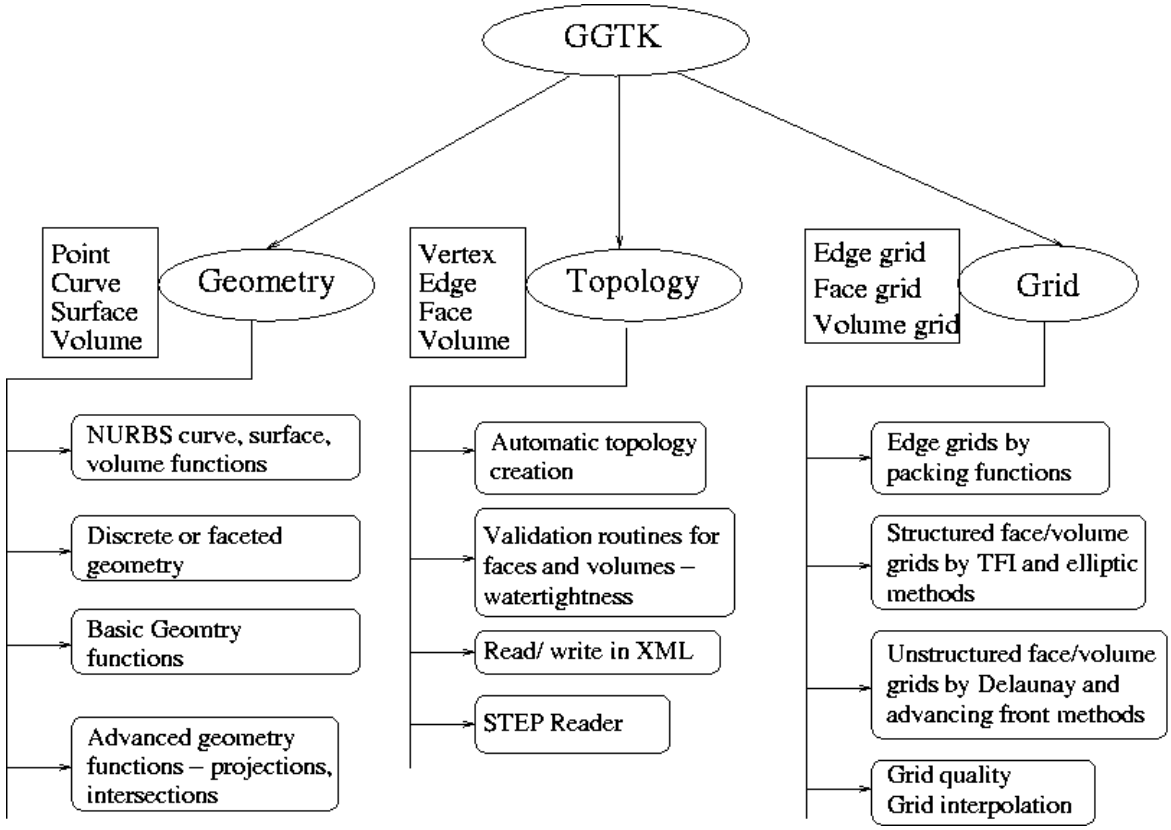

Figure 1: Components, entities and various functionalities of GGTK

2. TOPOLOGICAL REPRESENTATION SCHEME

This paper uses the topological representation scheme given in [4]. The scheme is based on *boundary representation* or *brep*. In a brep format, regions/volumes of three-dimensional space are described by the faces that bound them. Faces, in turn, are described by their underlying surfaces and boundary edges that bound them. Edges are described by their underlying curves and end vertices. Finally, vertices are defined by points. Brief descriptions of user level entities and API functions of the topological representation scheme are given below.

2.1. Topological Entities

In the scheme, a geometric object is called *GeoModel*. It is composed of entities called *GeoEntities* of dimensions 0, 1, 2, or 3 corresponding to vertices, edges, faces, and volumes. Each entity has an ID string. Each entity (except a vertex) is described by its boundaries, which are lower-dimensional entities. For example, a face shaped like a square holds the ID strings of four edges surrounding it (Figure 2). A *mapping function* embeds an entity from one dimension into a higher dimension. In the scheme, each time an entity *A* is used as a boundary for a higher-dimensional entity *B*, there is an accompanying mapping function to map *A* into the parametric domain of *B*. Each edge of the face in Figure 2, when used as a boundary of the face, is accompanied by a function from an interval of R^1 to R^2 that gives the position of the edge in the parametric domain of the face. This face, with an

accompanying mapping function from a region of R^2 to R^3, can then appear as the boundary of a volume.

The approach described in the previous paragraph is common to almost all brep formats that have been proposed in literature including STEP [10]. A distinguishing feature of the approach in [4] is that three-dimensional positions are not stored directly with the entities. For example, the position of an edge in three-dimensional space is not stored with the edge; instead, the edge's position is obtained by composing the mapping function embedding the edge into the parametric domain of a face (from R^1 to R^2) with the mapping function embedding the face into R^3. See Figure 2. Similarly, the three-dimensional position of a vertex is determined by composing three maps - a trivial parametric map from R^0 to R^1, a parametric map from R^1 to R^2, and a parametric map from R^2 to R^3.

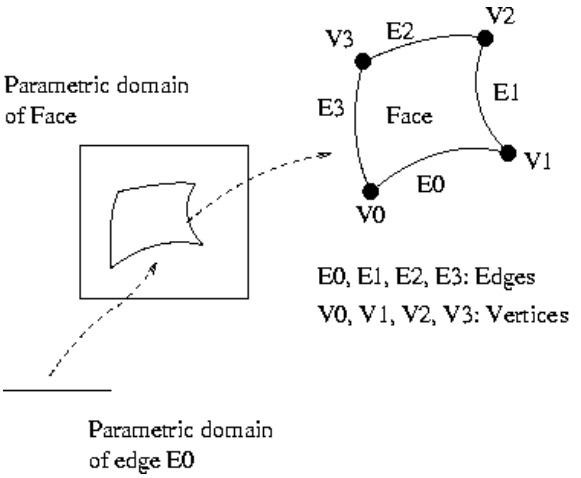

Figure 2: Representation of a face shaped like a square

The edge described in the previous paragraph can be mapped into R^3 in more than one way, since there are likely to be different faces that share that edge as a boundary. It is required that both mappings are equal up to a tolerance value [4]. Therefore, each entity of dimensions 0, 1 or 2 also stores a tolerance to indicate how much error (in the 2-norm) to expect when comparing two different embeddings of the entity into R^3.

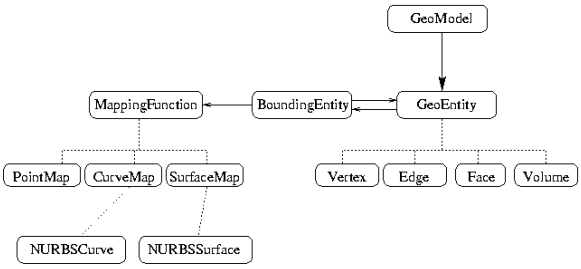

Figure 3. Topological entities

Figure 3 shows the main topological entities and mapping functions. In principle one could allow many classes of mapping functions, but the only type currently supported in GGTK is NURBS.

2.2. API Functions

The topology module contains API functions needed by grid generators. These functions are defined in terms of the topological entities – GeoModel and GeoEntity, thereby hiding the geometric details from the grid generators. Some of the API functions needed by surface grid generators mentioned in the next section are given below.

GeoModel Functions:

- vector<GeoEntity*> parts(int dimension)

Returns a list of all entities of the given dimension in a GeoModel.

- int checkWatertight(GeoEntity* volume, vector <GeoEntity*> wrongEdges)

Checks if a volume is watertight by looking at the edges of the volume. Note that a model can have more than one volume. So checking for watertightness is done for individual volumes.

- int checkFaceLoops(GeoEntity* face)

Checks if the boundary edges of a face form proper loops in the parametric domain of the face.

GeoEntity Functions:

- int orientationOf(GeoEntity* subentity)

Returns the orientation of subentity with respect to GeoEntity. This is needed to distinguish "inside" from "outside" of GeoEntity. Orientation is defined only for subentities whose dimensions are one less than that of the GeoEntity.

- int parameterDomain(double &xmin, double &xmax, double &ymin, double &ymax)

Computes the parameter domain of GeoEntity.

- vector<GeoEntity*> parts(int dimension)

Returns a list of all subparts or super parts of given dimension of the GeoEntity..

- GPoint coordinates(GeoEntity* subentity, GPoint coords)

Returns the coordinates in the parameter space of GeoEntity that correspond to given coords in the parameter space of subentity.

- double tangentBound(GeoEntity* edge, double a, double b)

Returns a bound on the tangent angles for a portion of edge. Parameters a and b are coordinates in the parameter space of edge. The angles are computed in the parameter space of GeoEntity. The value returned is the largest angle between a tangent along the portion of edge and the vector from a to

b in the parameter space of GeoEntity. This can be an upper bound rather than exact.

- double normalBound(GeoEntity* face, Point3d vector, double u0, double v0, double radius)

Returns a bound on the normal angles for a portion of a face. The surface portion desired is that within a sphere of specified radius about a center in the 3D space of GeoEntity. Parameters u0, v0 are parametric domain coordinates of the center. The value returned is the largest angle between a normal on the portion of surface and the specified vector. This can be an upper bound rather than exact.

- Point3d projectPointOnSurface(Point3d P, double u0, double v0, double &up, double &vp)

Computes the parametric coordinates (up, vp) of the projection of a given 3D point P on to the surface of a face. The computation is done starting with given initial parametric coordinates (u0, v0). The function returns the point of projection.

3. GRID ENTITIES AND API FOR GRID GENERATION

Grid entities are GEdgeGrid, GFaceGrid, and GVolumeGrid. These are defined over GeoEntities of dimensions 1, 2, and 3 respectively. Face and volume grids are further classified in to structured, unstructured, and generalized grids giving rise to the following derived entities: GStructFaceGrid, GUnstructFaceGrid, GGeneralizedFaceGrid, GStructVolumeGrid, GUnstructVolumeGrid, and GGeneralizedVolumeGrid (see Figure 4). Apart from these, there is also an entity called GModelGrid that contains all the grids related to a single topological model. Descriptions of these grid entities and API for generation are given in the following subsections.

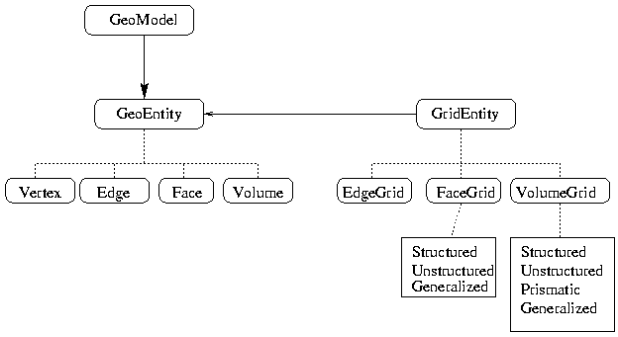

Figure 4. Grid entities and their link to GeoEntity

3.1. Edge Grid

An edge grid is obtained by computing points on the edge according to specified distribution. The distribution can be uniform or defined by what are known as packing functions. The packing can be at either end or both ends or at some in-between point of an edge. Table 1 shows some of the commonly used [14] packing functions and their parameters.

Packing type	Description	Parameters
Epack1L	Elliptic packing at start	n, ds1
Epack1R	Elliptic packing at end	n, ds2
Hpack1L	Hyperbolic packing at start	n, ds1
Hpack1R	Hyperbolic packing at end	n, ds2
Hpack2	Hyperbolic packing at both start and end	n, ds1, ds2
Hpack3	Hyperbolic packing at in-between point	n, ratio, ds3

Table 1: Various types of packing functions

The packing parameters n, ds1, ds2, ds3, and ratio mentioned in Table 1 denote the following:

n: Number of grid points
ds1, ds2, ds3: grid size at starting, end, and in-between point
ratio: ratio of the distances of the in-between point from start and end points

API Functions: GGTK provides the following GEdgeGrid class constructors as API functions for edge grid generation:

- GEdgeGrid(GeoEntity* edge, int npts)
- GEdgeGrid(GeoEntity* edge, double ds)
- GEdgeGrid(GeoEntity* edge, char* packType, int n, double ds1, double ds2, double ratio, double ds3)

The first two constructors generate uniform grids specified by number of points or by grid spacing. The third constructor generates a grid as per packing parameters.

3.2. Face Grid

This section gives descriptions of grid generation methods and API functions to generate structured and unstructured face grids. It is assumed that edge grids have been generated over all edges of a face before generating the face grid. The face grid will be consistent with the edge grids over its boundary edges.

3.2.1. Structured Face Grid

A structured face grid is defined by a rectangular array of grid points of size ni by nj. It can be generated over a face under the following conditions:

- The face has no holes

- It has four vertices v0, v1, v2, v3, designated as corner vertices
- *e0, e1, e2, e3* are boundary segments between the vertices (Figure 5) such that each of them is an edge or union of two or more edges
- Opposite pairs of segments (e0, e2) and (e1, e3) have same number of grid points – number of grid points on e0 and e2 is ni and that on e2 and e3 is nj.

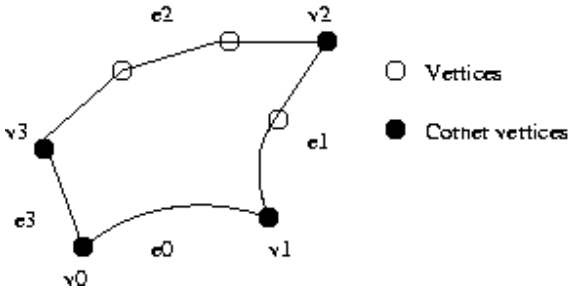

Figure 5. Four boundary segments for a structured face grid

API functions: GGTK has the following two constructors for structured grid generation. In both of them, the grid is generated by transfinite interpolation in the parametric domain.

- GStructFaceGrid (GeoEntity* face, GModelGrid* modelgrid)

This constructor assumes that the face has exactly four edges e0, e1, e2, and e3 forming a loop in that order and that opposite pairs (e0, e2) and (e1, e3) have same number of grid points.

- GStructFaceGrid(GeoEntity* face, GModelGrid* modelGrid, GeoEntity *v0, GeoEntity *v1, GeoEntity *v2, GeoEntity *v3)

This constructor is for a face with four or more edges. v0, v1, v2, v3 are corner vertices satisfying the conditions mentioned in the definition of a structured face grid.

3.2.2. Unstructured Face Grid

An unstructured face grid is a discretization of a face into triangles. The unstructured face grid class, *GUnstructFaceGrid* contains the following:

- Number of grid points, number of triangles
- Array of grid points storing their 3D coordinates and 2d parametric values
- For each triangle, three indices in the grid points array indicating the three vertices of the triangle.

Unstructured face grids can be generated by any surface grid generation method. The most common methods of surface grid generation are the advancing front method and Delaunay triangulation. These can be computed either in the parametric domain or directly in the physical (3D) space. It is to be noted that, in most of the cases, grids computed in the parametric domain will not be of good quality and can be used only for display purposes. Quality grids can be obtained if computed directly in 3D space.

API functions: Two API functions for unstructured face grid generation corresponding to two different methods are given below. The first one does the computation in parametric domain and the other one does it in 3D.

- GUnstructFaceGrid* delaunayInParametricDomain(GeoEntity* face, double size, GModelGrid* modelgrid)

This function computes an unstructured face grid by first computing a 2d unstructured grid in the parametric domain and then mapping it on to the surface in 3D. It preserves the edge grid points in the boundary of the face. The 2D grid is obtained using Shewchuk's *triangle* routine [15], which computes a Delaunay triangulation. While the triangle routine takes many parameters to control the triangulation, this routine passes only the maximum area bound parameter specified by the input parameter, "size".

- GUnstructFaceGrid* advancingFrontMethodOverParametricSurface(GeoEntity* face, GModelGrid* modelgrid, int maxNumTriangles)

This function computes an unstructured face grid by *advancing front method* directly in 3D, which is similar to the one given in [16]. New points are computed in physical space in order to ensure quality triangles. The new points are projected onto the underlying surface so that all the grid points lie on the surface.

- *Background grid*: Advancing front method uses what is known as background grid to obtain required triangle sizes in the interior of the surface. In this routine, a background grid is computed internally over the parametric domain using initial boundary grid points. Required triangle sizes at these grid points are estimated from the average boundary edge grid sizes at these points.
- *Maximum number of triangles*: This is used as one of the terminating conditions in advancing front method. This is specified by the input parameter "maxNumTriangles".

Delaunay surface mesh component: Using the API functions listed in Section 2.2, Chew [8, 12] has developed a surface mesh component on top of GGTK. It takes a geometric model file in XML and generates surface grid for the whole model. Grid size and shape requirement are provided by the following input parameters:

- maxRatio: This is an upper bound for the ratio of triangle circumcircle radius over shortest edge. The smallest accepted value is 1 which implies an angle-bound of 30 degrees. Larger numbers allow smaller angles to appear in the resulting mesh.
- maxDisplacement: The displacement of a triangle is a rough measure of how closely the triangle matches the

curvature of the local surface. It is the distance between the actual surface and the plane of the triangle.

- maxSize: This is an upper bound for the length of an element's longest edge.

Remark: The above-mentioned Delaunay mesh component has been developed outside GGTK. It uses only the topological entities and API functions of GGTK. It does not use the grid entities. Hence, instead of an API function, the current interface is through files. One provides a model file and gets back a mesh file in a specific file format.

3.3. Volume Grid

3.3.1. Structured Volume Grid

A structured volume grid is defined by a three dimensional array of points defined over a volume. ni, nj, nk denote the number of points in three directions and v000, v100, v110, v010, v001, v101, v111, v011 denote the eight corner vertices of the volume.

API Functions:

- GStructVolumeGrid(GeoEntity* vol, GModelGrid* modelgrid, GeoEntity* v000, GeoEntity* v100, GeoEntity* v110, GeoEntity* v010, GeoEntity* v001, GeoEntity* v101, GeoEntity* v111, GeoEntity* v011)
- GStructVolumeGrid(GeoEntity* vol, GModelGrid* modelgrid, vector<GStructFaceGrid*> facegrids)

The above two routines generate structured volume grids over a given volume, provided the volume is bounded by six faces having structured face grids defined over them and the face grids have compatible number of points along i, j, k directions. The structured volume grid is obtained by 3D transfinite interpolation.

For the first routine, the corner vertices are passed as input parameters in a specific order. The routine will search the modelgrid and obtain the six face grids. For the second routine, the six face grids are passed in an input vector containing the face grids in the following order: i-min, i-max, j-min, j-max, k-min, k-max.

3.3.2. Unstructured Volume Grid

Unstructured volume grid over a topological volume is defined by a collection of grid points and grid elements, which are tetrahedrons. The two most popular methods for unstructured volume grid generation are the advancing front and Delaunay triangulation. Usually, the input to a tetrahedral volume grid generator is a surface grid defined over the boundary faces of the volume and some parameters to indicate required size and shape of grid elements. Some grid generators, such as the advancing front method in [17], will generate volume grids that conform to input surface grids and will not require interface to the geometric models. Others, such as the Delaunay meshing in [18], may not conform to input surface grids and may need interface to insert additional points on the surface to ensure high quality meshes.

API Function: Currently GGTK has the following API constructor for unstructured volume grid generation.

- GUnstructVolumeGrid(GUnstructFaceGrid* surfaceGrid)

This constructor takes an unstructured surface grid as input and generates an unstructured volume grid by advancing front method that conforms to the input grid. The input surface grid is required to be watertight and oriented.

3.4. Model Grid

A model grid contains all the grids related to a single topological model. Specifically, it has lists of edge, face, and volume grids.

GModelGrid Functions: GGTK provides the following API functions to generate grids over a whole model using the GEdgeGrid and GFaceGrid API functions/ constructors defined previously.

- createAllEdgeGrids(int npts)

Creates all edge grids with uniform packing for given number of points.

- createAllEdgeGrids(double ds)

Creates all edge grids with uniform packing for given grid spacing.

- createAllStructuredFaceGrids()

Creates all structured face grids, assuming that all faces are four sided and the edge grids satisfy the requirements of structured face grids.

- createAllFaceGrids(double size)

Creates all unstructured face grids by Delaunay triangulation in parametric domains.

- createAllFaceGridsbyAFM()

Creates all unstructured face grids by advancing front method.

These functions generate grids using a few globally applied input parameters. If one wants to control the grid over individual edges or faces, then one can use the grid specification scheme given in the following section.

4. GRID SPECIFICATION

This section explains a scheme to specify required grids over some or all parts of a model and mentions how this scheme can be used to develop web components for grid generation.

4.1. XML Scheme

The scheme is defined in XML (Extensible Markup Language). The actual schema file (gridSpec.xsd) is given

in the Appendix. The main elements of the schema are *GridSpecification, edgeGridSpec, faceGridSpec,* and *volumeGridSpec*. Explanations of these elements and their attributes are given below.

GridSpecification element: It consists of a sequence of elements edgeGridSpec, faceGridSpec, and volumeGridSpec. It has two attributes GridSpecId and ModelId. The GridSpecId attribute stores a unique name for the grid specification. The ModelId attribute stores the name of the geometric model over which the grid is to be generated.

EdgeGridSpec element: It is used to specify an edge grid. Its attributes are edge name, edge grid packing type, and packing parameters n, ds1, ds2, ds3, and ratio as defined in Section 3.1.

FaceGridSpec element: It is used to specify a face grid. Its attributes are face name, face grid type, face grid method, and required face grid parameters as defined in Section 3.2.

VolumeGridSpec element: It is used to specify a volume grid. Its attributes are volume name, volume grid type, volume grid method, and required volume grid parameters as defined in Section 3.3.

4.2. Scheme Example

Figure 6 shows grid over a single element injector (SEI) model specified by the above XML scheme. The model has 42 vertices, 70 edges, and 34 faces. Given below is a sample specification over four edges and three faces of the model. Note that structured and unstructured grids are specified over different parts of the same model.

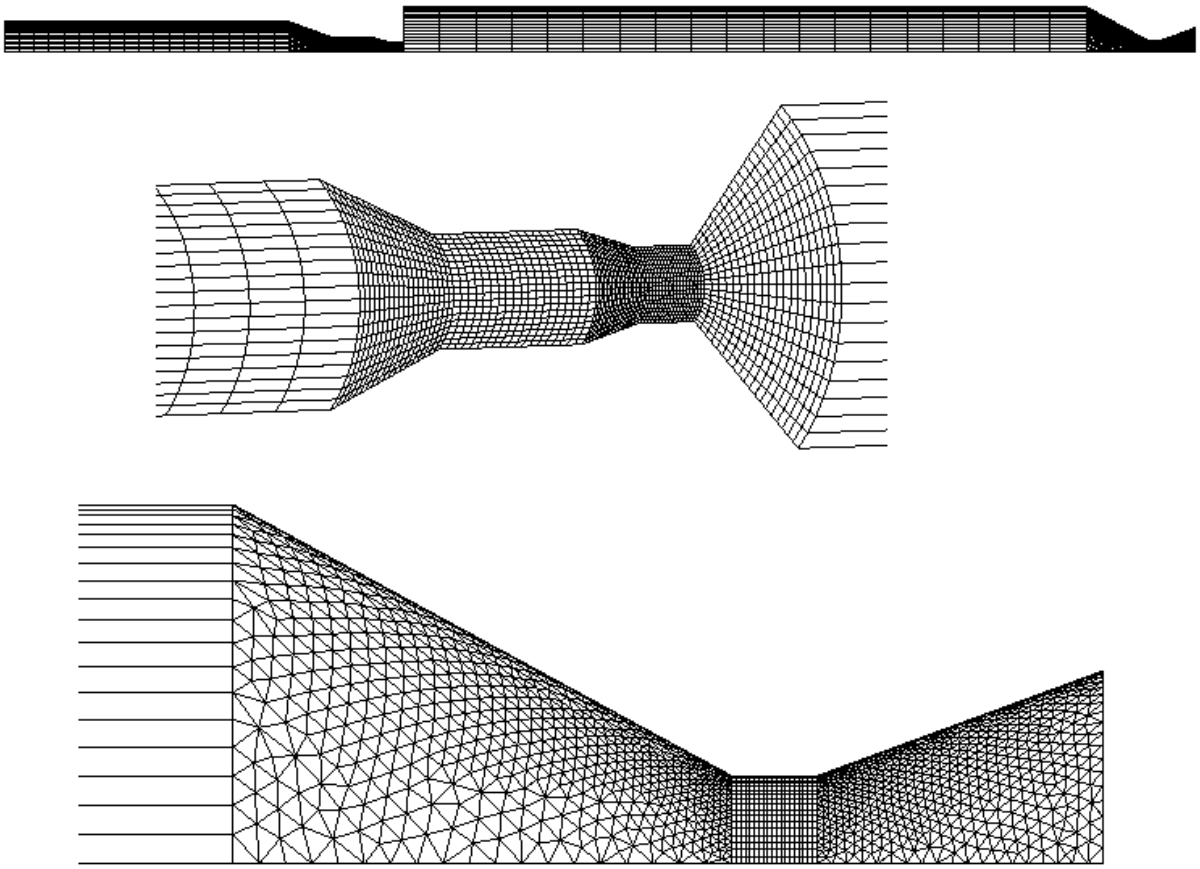

Figure 6: Grids over various faces of SEI

Sample grid specification:

```
<?xml version = "1.0" ?>
<GridSpecification
        GridSpecID = "SEI_GridSpec"
        ModelID = "SingleElementInjector"
<edgeGridSpec edgeName = "ED0"
        packType = "UPACK" npts = "20"/>
<edgeGridSpec edgeName = "ED1"
        packType = "UPACK" npts = "20"/>
<edgeGridSpec edgeName = "ED2"
        packType = "UPACK" ds = "0.001"/>
<edgeGridSpec edgeName = "ED3"
        packType = "UPACK" npts = "20"/>
<faceGridSpec faceName = "NozzleFace_0"
        gridType = "Unstructured"
        method = "3D_Surf_AdvancingFront"
        maxNTriangles = "10000"/>
<faceGridSpec faceName = "NozzleFace_1"
        gridType = "Structured"
        method = "2D_TFI"/>
<faceGridSpec faceName = "NozzleFace_2"
        gridType = "Unstructured"
        method = "3D_Surf_AdvancingFront"
        maxNTriangles = "10000"/>
</GridSpecification>
```

4.3. Web component for grid generation

Using the above XML scheme for grid specification, one can develop a web component for grid generation that takes in a model file and a grid specification file, and returns a grid file in a specific file format. The component can be used by remote users to obtain required grids

The following things are needed to develop such a web component:

1. Model generator in GGTK format: It can either provide mechanisms to create models in GGTK format or it can provide tools to convert models in IGES or STEP format generated using other CAD systems.

   GGTK has many API functions for model creation [5, 6] and currently STEP to GGTK converter functions are being developed. A MiniCAD system is also being developed, which is based on GGTK and provides Graphical User Interface (GUI) for model creation, editing and grid generation.

2. Grid generator that supports the grid API functions: GGTK already has the API functions mentioned in Section 3. The grid generator can use these functions or can have its own functions like Delaunay surface mesher of Chew mentioned in Section 3.2.2.

3. Web interface: This is to provide an interface to remote users through the Internet. Since the grid generation component needs only the transfer of files – model file, grid specification file, and grid file, the interface can be easily created using web tools like OSOAP [19].

5. CONCLUSIONS AND FUTURE DIRECTIONS

The paper has shown a framework of geometric and grid API functions that can be provided in an integrated CAD/Grid system using topological representation to define geometric models. It can be used to develop standard grid generation tools/ components to generate structured/ unstructured face/ volume grids over topological models by TFI/ Delaunay/ advancing front methods.

The framework has been presented in terms of the topological scheme used in GGTK, which currently supports curves and surfaces defined by NURBS. The topological scheme is being extended to include other kinds of surfaces such as faceted and subdivision surfaces. Then the framework will be extended to provide grid generation tools for models defined by those kinds of surfaces. It also can be extended to include generalized grids and other grid generation methods.

The current grid generation API assumes that there is a one to one correspondence between topological entities and grid entities. An extension is being investigated to create grid entities that can exist within or span topological entities of the same or higher dimension. For example, currently a grid edge exists on a complete topological edge. The extension would allow a grid edge to exist as part of an edge or span edges or exist in a face, several faces, or in a volume. The data structure of the extended grid entities would maintain its own boundary representation. Using the underlying topological model it will be possible to create a watertight grid model because of the grid's topological structure and the fact that the topological model provides a unique solution for any position on the model. One may have to address continuity problems that may exist where grid edges, faces, and volumes cross topological-boundaries of the model.

ACKNOWLEDGEMENTS

This research is supported in part by NASA URETI Program and NSF ITR Adaptive Software Project. The authors would like to thank Dr. Paul Chew of Cornell University for specifying the topological API functions needed for his surface grid generator and for fruitful discussions on developing the APIs. They thank Yasushi Ito for providing unstructured volume grid generation tool in GGTK.

APPENDIX

XML Schema for Grid Specification

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"

<annotation> <documentation xml:lang="en">
   Grid specification for a geometric model defined in
terms of the topological representation scheme of the
Cornell-UAB-MSU Adaptive Software Project.
```

```
    </documentation>
</annotation>

<element name="GridSpecification"
       type="CU_UAB_ASP:GridSpecificationType">
</element>

<complexType name="GridSpecificationType">
  <sequence minOccurs="0" maxOccurs="unbounded">
    <choice>
      <element name = "edgeGridSpec"
            type = "CU_UAB_ASP:EdgeGridSpecType"/>
      <element name = "faceGridSpec"
            type = "CU_UAB_ASP:FaceGridSpecType"/>
      <element name = "volumeGridSpec"
            type = "CU_UAB_ASP:VolumeGridSpecType"/>
    </choice>
  </sequence>
  <attribute name = "GridSpecID" type = "NMTOKEN" use
= "required"/>
  <attribute name = "ModelID" type = "NMTOKEN" use =
"required"/>
</complexType>

<complexType name = "EdgeGridSpecType">
  <attribute name = "edgeName" type = "string" use =
"required"/>
  <attribute name = "packType" type =
"CU_UAB_ASP:EdgeGridPackType" use = "required"/>
  <attribute name = "npts" type = "positiveInteger" use=
"optional"/>
  <attribute name = "ds" type = "double" use="optional"/>
  <attribute name = "ds1" type = "double" use="optional"/>
  <attribute name = "ds2" type="double" use="optional"/>
  <attribute name = "ratio" type="double" use="optional"/>
  <attribute name = "ds3" type = "double" use="optional"/>
</complexType>

<simpleType name = "EdgeGridPackType">
  <restriction base = "string">
    <enumeration value = "UPACK"/>
    <enumeration value = "EPACK1"/>
    <enumeration value = "EPACK2"/>
    <enumeration value = "HPACK1"/>
    <enumeration value = "HPACK2"/>
    <enumeration value = "HPACK3"/>
  </restriction>
</simpleType>

<complexType name = "FaceGridSpecType">
  <attribute name = "faceName" type = "string" use =
"required"/>
  <attribute name = "gridType" type =
"CU_UAB_ASP:FaceGridType" use = "required"/>
  <attribute name = "method" type =
"CU_UAB_ASP:FaceGridMethodType"
          use = "required"/>
  <attribute name = "maxSize" type = "double" use =
"optional"/>
  <attribute name = "maxNTriangles" type = "integer" use =
"optional"/>
</complexType>

<simpleType name = "FaceGridType">
  <restriction base = "string">
    <enumeration value = "Structured"/>
    <enumeration value = "Unstructured"/>
  </restriction>
</simpleType>
<simpleType name = "FaceGridMethodType">
  <restriction base = "string">
    <enumeration value = "2D_TFI"/>
    <enumeration value = "2D_using_CMU_triangle"/>
    <enumeration value = "3D_Surf_AdvancingFront"/>
    <enumeration value = "3D_Surf_Delaunay"/>
  </restriction>
</simpleType>

<complexType name = "VolumeGridSpecType">
  <sequence minOccurs="0" maxOccurs="1">
    <choice>
      <element name = "eightVertices"
            type = "CU_UAB_ASP:EightStringsType"/>
      <element name = "sixFaces"
           type = "CU_UAB_ASP:SixStringsType"/>
    </choice>
  </sequence>
  <attribute name = "volumeName" type = "string" use =
"required"/>
  <attribute name = "gridType" type =
"CU_UAB_ASP:VolumeGridType" use = "required"/>
  <attribute name = "method" type =
"CU_UAB_ASP:VolumeGridMethodType"
          use = "required"/>
</complexType>

<simpleType name = "VolumeGridType">
  <restriction base = "string">
    <enumeration value = "Structured"/>
    <enumeration value = "Unstructured"/>
  </restriction>
</simpleType>
<simpleType name = "VolumeGridMethodType">
  <restriction base = "string">
    <enumeration value = "TFI"/>
             <enumeration value = "AdvancingFront"/>
             <enumeration value = "Delaunay"/>
  </restriction>
</simpleType>

<simpleType name = "stringList">
  <list itemType = "string"/>
</simpleType>
<simpleType name = "EightStringsType">
  <restriction base = "CU_UAB_ASP:stringList">
    <length value = "8"/>
  </restriction>
</simpleType>
<simpleType name = "SixStringsType">
  <restriction base = "CU_UAB_ASP:stringList">
    <length value = "6"/>
  </restriction>
</simpleType>
</schema>
```

REFERENCES

[1] Rida T. Farouki, "Closing the Gap Between CAD Model and Downstream Application", *SIAM News Online*, http://www.siam.org/siamnews/06-99/cadmodel.htm, Vol. 32, 1999.

[2] Y. Zheng, N. P. Weatherill, O. Hassan, "Topology abstraction of surface models for three-dimensional grid generation", *Engineering with Computers,* vol. 17, 28-38, 2001.

[3] R. Haimes and G.J.Follen, "Computational Analysis Programming Interface", *Proceedings of the 6th International Conference on Numerical Grid Generation in Computational Field Simulations*. July, 1998.

[4] L. Paul Chew, Stephen Vavasis, S. Gopalsamy, TzuYi Yu, Bharat Soni, "A Concise Representation of Geometry Suitable for Mesh Generation", *Proceedings of 11th International Meshing Roundtable*, Ithaca, September 2002.

[5] S. Gopalsamy, TzuYi Yu, "A Geometry Engine for CAD/Grid Integration", AIAA 2003-800, 41st Aerospace Sciences Meeting & Exhibit, January 2003, Reno, Nevada.

[6] Geometry and Grid Toolkit (GGTK) documentation, http://www.eng.uab.edu/me/ETLab/Software/GGTK/manual/index.html

[7] M. J. Aftosmis, M. Delanaye, R. Haimes, "Automatic Generation of CFD-Ready Surface Triangulations from CAD Geometry", AIAA Paper 99-0776, January 1999.

[8] L. P. Chew, P. Stodghill, "Interface for the surface mesher", *Project communication, ITR-ASP Project*, Cornell University, 2003. http://www.asp.cornell.edu/project.

[9] K.J.Weiler, "Topological Structures for Geometric Modeling", *Ph.D. Thesis*, Rensselaer Polytechnic Institute, 1986.

[10] STEP: International Standard ISO 10303-42. Industrial automation systems and integration - Product data representation and exchange - Part 42: *Integrated generic resource: Geometric and topological representation*, 2000.

[11] "Unstructured Grid Consortium – Programmers Reference", *UGC standards document*, Version 1.0, 2002, http://www.pointwise.com/ugc/

[12] L. P. Chew, "Guaranteed-quality mesh generation for curved surfaces." *Proceedings of the ninth symposium on computational geometry*, pp. 274–280. ACM Press, 1993.

[13] L. P. Chew, "API functions needed for Delaunay surface grid generation over a topological model", *Project communication, ITR-ASP Project*, Cornell University, 2002.

[14] J. F. Thompson, B. K. Soni, N. P. Weatherill (Eds), "*Handbook of Grid Generation*", CRC Press, 1999.

[15] J. R. Shewchuk, "Triangle – A two dimensional quality mesh generator and Delaunay triangulator", http://www-2.cs.cmu.edu/~quake/triangle.html.

[16] J. R. Tristano, S. J. Owen, S. A. Canann, "Advancing front surface mesh generation in parametric space using a Riemannian surface definition", *Proceedings of 7th International Meshing Roundtable*, Dearborn, Michigan, 1998.

[17] J. B. Cavalcante Neto, P. A. Wawrzynek, M. T. M. Carvalho, L. F. Martha, A. R. Ingraffea, "An algorithm for three-dimensional mesh generation for arbitrary regions with cracks", *Engineering with Computers,* vol. 17, 75-91, 2001.

[18] P. L. Chew, "Guaranteed-Quality Delaunay Meshing in 3D", *Proceedings of the 13th ACM Symposium on Computational Geometry*, pp. 391–393, ACM Press, 1997.

[19] P. Stodghill, "O'Caml Tools for Exposing Legacy Applications as Web Services", the file "osoap-0.9.3.tar.gz" at www.asp.cornell.edu/osoap/sources/.