A CAD-centric Approach to CFD Analysis With Discrete Features

Matthew L. King [1], Matthew J. Fisher [2] and C. Greg Jensen [3]

[1]Altair Engineering, matt@altair.com
[2]Brigham Young University, matt.fisher@byu.edu
[3]Brigham Young University, cjensen@byu.edu

ABSTRACT

CAD models from conceptual design often follow the "over-the-wall" approach for downstream analyses such as FEA, CFD, heat transfer, and vibrations. The CAD-centric approach consists of using the CAD model as a source of data for downstream applications such as mesh generation, and CFD setup. The CAD model used in the CAD-centric approach contains the geometry to be analyzed and all non-geometric data required to mesh and solve the CFD model in the form of attributes. A special class of topology change, the discrete feature problem, is encountered when an array of features instances change in number. A method is proposed, developed and reported on that automates the CAD to CFD process, including fluid domain creation, while addressing the discrete feature problem that can occur during preliminary design.

Keywords: CAD-centric, CFD analysis, discrete features.

1. INTRODUCTION

Engineering tools are constantly being revised, enhanced and extended to help improve the engineering and design artifacts, increase productivity, and reduce costs. Today, software tools such as parametric CAD systems, finite element analysis (FEA), and computation fluid dynamics (CFD) have been developed to meet these objectives. Incorporating these tools into product development has been a cost-effective way to increase productivity and improve designs [5].

CAD models from conceptual design often follow the "over-the-wall" approach (see Fig. 1.) for downstream analyses such as FEA and CFD [3]. The over-the-wall approach consists of four domain experts namely: the designer, airsolid designer, mesh expert, and CFD expert. When the designer has proposed a design it is transferred to the airsolid designer to create the fluid domain, hereafter referred to as an airsolid, then to the mesh expert to create the mesh, and finally the CFD expert to run the CFD analysis. These experts frequently use CAD neutral formats as the model moves from one domain to the next and even duplicate efforts. This over-the-wall approach creates opportunity for error or design escapes that cost a company large amounts of time and money. Since the CAD-to-CFD process is time consuming, CFD has played a limited role in conceptual design, especially where complex models are involved. Utilizing a CAD-centric approach (see Fig. 2.) will greatly reduce the CAD-to-CFD cycle time and facilitate for optimization. CAD-centric denotes that all geometry and non-geometric parameters required to perform the CAD to CFD process are contained within the CAD model and created within the CAD domain.

Samareh [4] states that in an ideal environment, an engineer would use a parametric model to effortlessly evaluate variant models. In essence this would mean that all the inputs necessary to perform FEA would be given in the CAD domain. For this ideal environment to be realized, Samareh [4] suggests that geometry modeling and mesh generation tools must be automated, provide consistent geometry to all disciplines, be parametric and fit within the product development cycle times. Bailey [2] said it best when he stated that "the new approach is to have geometry central, or common, to all the processes and to use geometry as a design integrator." The method presented in this paper attempts to satisfy the requirements of this ideal environment.

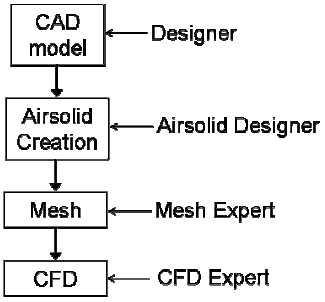

Fig. 1. The over-the-wall approach.

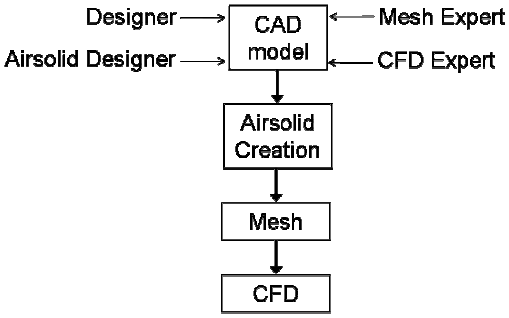

Fig. 2. The CAD-centric approach.

This paper reports on research aimed at automating the CAD-to-CFD design process from the CAD software to CFD setup while addressing the discrete feature problem. Presently we are focused on the geometry of a single component or part and not the geometry of an assembly, however many of the principles discussed here could easily be applied to assemblies. The meshing presented in this paper is the result of the autoemeshing and tetrameshing contained within HyperMesh. In addition we also used the smoothing algorithm's within HyperMesh to improve the element quality. We present results to show that we have solved the discrete feature problem and have created a CAD-centric approach to CFD analyses.

Varying the number of feature instances in an array within the CAD model, such as spokes on a wheel, vanes in a flow passage, or pins on a cooling surface is a special class of topology change. This creates the problem of introducing new entities and having to create or delete attributes associated to the feature. Throughout this article the aforementioned problem will be referred to as the *discrete feature problem*. While it is a relatively minor change in today's parametric CAD systems, the CAD-centric model must be reworked. Reworking the model would include tagging the new entities that were created, recreating the airsolid, and generating a mesh and analysis model. This can become time intensive and delay time to market. One of our objectives was to develop a method for automatically reworking the model and applying attributes when a discrete feature change occurs.

2. METHOD

To accomplish our objectives required the development and coding of the following six steps. Each of these steps will be discussed in more detail.

1. Develop a method that allows for automation of the conceptual design process involving CAD parametric component model, parametric airsolid creation, mesh generation, and CFD setup.
2. Solve the discrete feature problem within the CAD parametric component model.
3. Creation of the airsolid.
4. Semi-automatic mesh and preprocessing of the airsolid.
5. Solving the CFD model.
6. Iterate, by varying the parameters of the CAD component model.

2.1 Concept Generation: Parametric Models

Concept generation and their 2D and 3D parametric models can either be automated (programmatic) or performed manually (interactively). In this paper we discuss (use) both the automatic and manual concept generation approaches. An automatic approach was used for a cylinder study. It is recommended that a programmatic approach be developed if many instances of a similar concept need to be analyzed. In the case of the cylinders the programmatic approach will reduce concept generation time and facilitate automatic airsolid creation. One of the requirements of this method is that the component(s) involved must be parametric and a parametric CAD system must be used since it enables quick design changes that often occur in concept generation. The parametric model is created by a skilled designer by creating key parameters and relations that allow for reuse of the model. The parametric model can be time consuming since care must be taken to ensure that parameter changes result in a valid design. Once the parametric model is created an attribute management system was needed to associate attributes to the model for downstream applications, thus making it CAD-centric.

An attribute management system (AMS) simply assists the designer in applying, deleting, and editing global and associative attributes of the model. An object-oriented data structure and an intuitive interface are needed to create an AMS. The organization of this data structure is explained in section 3.2

2.2 The Discrete Feature Problem

This section discusses a general method that resolves the discrete feature problem. Section 3.3 discusses how this general method was developed using the API of a commercial CAD system. To solve the discrete feature problem a program must be built that follows these steps:

1. Search all features of the model for a discrete feature.
2. Find parent feature instance of the discrete feature.
3. Copy all attributes from parent feature instance.
4. Collect children feature instances.
5. Apply attributes to children feature instances.

2.3 Airsolid Creation

This section describes some generic methods that should be used as part of creating an airsolid. A programmatic approach to creating a parametric airsolid is recommended if many instances of a similar concept are used. We will discuss a duct containing varying cylinder configurations as a case study for this paper. Steps for creating a valid airsolid include:

1. Wave (link) component(s) resulting from concept generation.
2. Unite all waved components if necessary.
3. Create initial airsolid representing the limits of the fluid domain without detail from concept design.
4. Subtract concept generation geometry from initial airsolid resulting in the final airsolid.

Steps 1, 2, and 4 should be applied to arbitrary geometry, but creating the initial airsolid is specific to individual models. A method cannot be developed to create the initial airsolid for an arbitrary geometry because entities from concept generation are used and vary between concepts. Fig. 3., shows an exhaust manifold and its corresponding airsolid. The airsolid was created by sweeping edges from the inlets along a guide curve. These edges may change dimensionally or they may undergo topological changes. Fig. 4., shows how a 2D airsolid or *airsurface* was created for a car. First, a surface (A) large enough to enclose the profile of a car and model everything from the surface effects to the ambient was created. Then the profile (B) was created and subtracted resulting in airsurface (C).

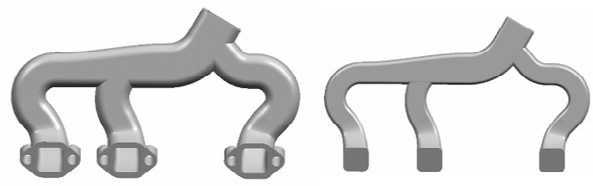

Fig. 3. Exhaust manifold and its airsolid.

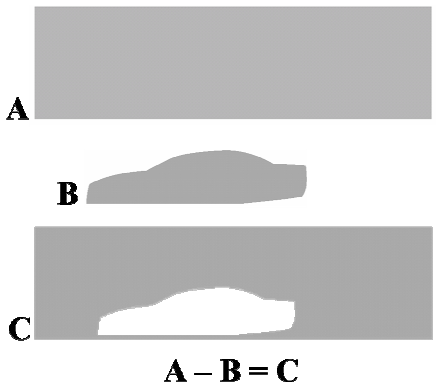

Fig. 4. 2D Process for creating an airsurface for a car.

Even though creating an airsolid is unique to each case, some elements of airsolid creation should be consistent regardless of the geometry. Those elements include the use of WAVE technology, automation, and attribute mapping techniques. The essence of WAVE is the ability to relate geometry between files by creating associative copies of geometry. The key to WAVE is that the parent assembly or component serves as a reference and all changes to the waved geometry will have no effect on the original geometry from which it was waved. Airsolid creation is specific to CFD analysis so the majority of this section is directed toward CFD analysis, with the exception of the discussion on waving the geometry that results from concept generation.

The AMS, as discussed in Section 2.1, is initially used to create and associate attributes to each component used in concept generation. A method was developed that maps all attributes from each component to the corresponding entities of the final airsolid. This allows the designer to do the majority of the attribution on the component level rather than at the airsolid level.

When a component is waved all attributes are copied from the component to the waved geometry. After all of the waved geometry has been united, each entity has still retained the original attributes associated to them. It is not until new entities have been created that attributes must be applied. When the initial airsolid is created, none of the entities have attributes associated to them. Thus, this necessitates that attributes be mapped onto the airsolid. Attributes are mapped onto the airsolid with the use of the subtraction Boolean operation. After all the components are united, a final subtraction operation with the united components and the initial airsolid occurs. This results in the final airsolid with attributes mapped from the united components.

2.4 Semi-automatic Mesh Generation and CFD Setup

Once the final airsolid has been created and the attribution process is complete as just described, the designer has a CAD-centric airsolid model. The CAD-centric model has all of the information necessary to generate a mesh and setup the analysis model. This is achieved by creating scripts, hereafter referred to as command files, within the CAD system for the mesh, and analysis software. The program within the CAD system that creates the command files will be referred to as the Command File Generator (CFG). Once the command files are created by interrogating the CAD-centric model (details in section 3.5) they are executed in the respective software, either interactively or in a batch mode. The command files will generate the mesh, smooth the mesh, apply boundary conditions, and set up solution criteria for the analysis. If the command file is executed from the user interface, then the designer may still interactively interrogate the mesh or analysis model and make changes. This allows the designer the ability to manually override any mesh or analysis parameters that were specified in the AMS. Thus this method is referred to as semi-automatic.

The mesh and analysis command file is created by using the CAD system's API. The resulting command files are a list of commands written for the mesh and analysis software. The APIs of the mesh and analysis software is typically a macro type language. A benefit of the macro API is that it is easy to learn and create from the CAD API. The CAD-centric model contains all information needed to create the command files, so the CFG interrogates the model for all global and associative attributes, and processes them into a command file for both the mesh and analysis software.

This is accomplished by creating an algorithm within the CAD API that will search all global and associative attributes specific to the mesh and analysis software.

2.5 Solving

When the meshing and CFD setup is complete by executing the command files for the meshing and CFD software the input deck is submitted to the solver. This input file can be prepared to run on a single workstation, a carpet cluster or a supercomputer. We have experimented with all of these scenarios and have determined that it is an easy set of commands that will allow the job to run on a single workstation or be farmed out to multiple processors.

2.6 Iteration

Because this is being proposed as a preliminary design tool and methodology it is necessary to allow for quick iterations within components or assemblies design space; the design goal is the determination of an optimal set of defining parameters. In our research, we have developed our own optimization loop as well using commercially available software like FIPER and Teamcenter Engineering. It has been our experience that developing our own or customizing a commercial tool leads to the most robust iteration of a CAD-centric application.

3. DEVELOPMENT

A programmatic parametric model had to be built in order to automate the conceptual design process that includes preliminary design, assembly, and airsolid creation. The application for developing the concepts and resulting airsolid applies the development of the case study; however, the methods used have application to many other industries including: automotive, aerospace, and ship building. The models and rules are part of a design tool that is being built as a custom application by utilizing C++ object oriented programming and the Unigraphics application program interface (API). This design tool will be referred to as the Preliminary Design, Assembly, and Airsolid creator, or PDAA.

3.1 Concept Generation

A programmatic approach for concept generation was selected for the cylinder region. PDAA is the tool resulting from the programmatic approach for the cylinder region. PDAA has many capabilities, and since it is still in development, its functionality increases daily. Individual components were created interactively and used by PDAA. Components were defeatured by using an airsolid flag that is controlled by PDAA.

3.2 Attribute Management System

The Attribute Management System, AMS, was developed by utilizing an object-oriented data structure in C++, the Unigraphics UG/OPEN API toolkit, and the Unigraphics User Interface Styler (UIStyler). The AMS is used to assign attributes and address the discrete feature problem at the component level. After the airsolid is created the AMS is used again to apply attributes to the few newly created face entities.

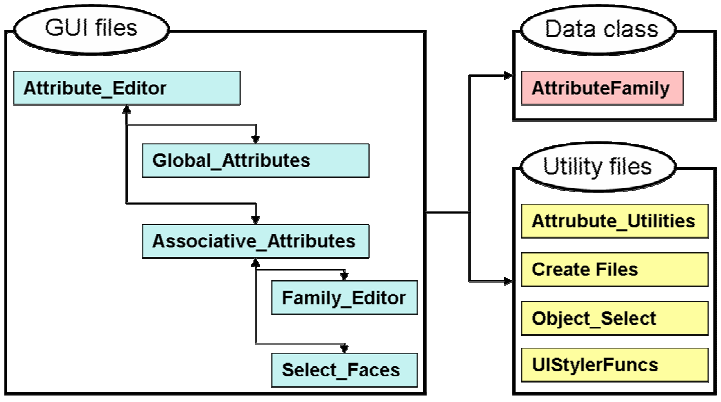

Fig. 5. Organization of AMS program.

The organization of the AMS data structure consists of graphical user interface (GUI) files, a data structure class, and utility functions. Fig. 5., shows how these files are organized. The Attribute_Editor GUI file is the master GUI and is explained in Fig. 6. When the master GUI opens, the constructor interrogates the currently active part file for attributes

and fills the data structure. If no attributes exist then they are created through the GUIs. The only data class that exists in the AMS is the AttributeFamily class. The AttributeFamily class consists of data including a name, boundary condition, boundary condition value (if appropriate), element size, and a color. The AttributeFamily class also consists of functions that will set and retrieve the data, write the family data as attributes to the part file, and associate attributes to face entities. The information for all the families is contained in a global vector that can be accessed anywhere in the AMS program.

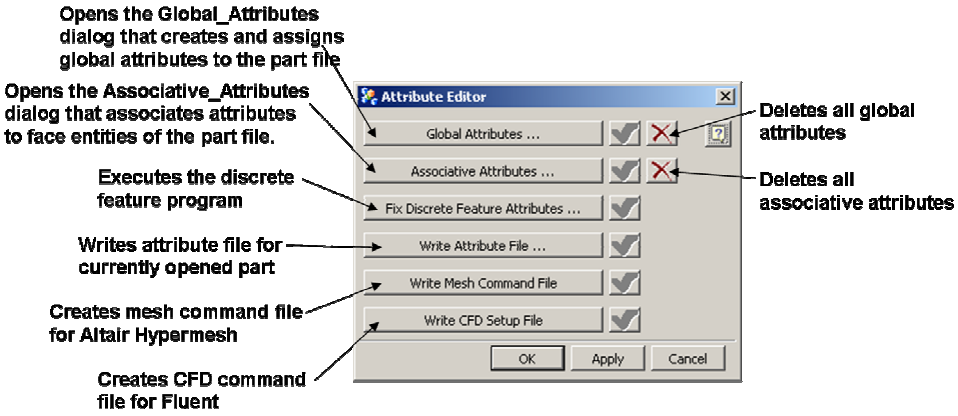

Fig. 6. Attribute editor dialog.

AMS consists of dialogs that contain pull downs and entry fields to gather information to be stored in the CAD file in the form of attributes. The AMS dialogs contain many options, but do not contain all possible choices. For example, the AMS allows the designer to select between three different element types among hundreds. The purpose of the AMS is to create a proof of concept analysis tool that creates and edits CAD-centric models.
The global attributes dialog assigns the global attributes and includes the ability to copy attributes from other part files. This functionality is very useful when dealing with multiple components, because global attributes only have to be manually defined in one component and then they can be copied to the other components.

The second step for creating a CAD-centric model using the AMS is to create families. Since the family element size is a multiple of the global element size, the global attributes should be assigned before creating families. A check was created that ensure global attribute exists in the part that contains the element size. If no global element size was found then a warning message is displayed. The third step is to apply the family name to face entities of the model. The developed AMS assigns attributes to face entities only, but the functionality could be extended to point and edge entities. Assigning attributes to only the face entities made creating the mesh command file much easier with Altair HyperMesh, see section 3.5. Currently, the AMS copies the element density of the face entity to its edges. In the case of an edge that is shared between two faces of different densities, the smaller of the two is applied. A function was built that prints all the global and associative attributes to a file. This file is used to check all of the current attributes that exist in the part file. The first section of the file lists global attributes including all family information. The last section of the attribute file lists the HyperMesh face identification number and the family name associated with it.

After the attribution process is complete the discrete feature problem is addressed, if necessary, and the command files are generated. This AMS was developed specifically for Unigraphics, Altair HyperMesh, and Fluent; however this method could also be applied to any CAD and CAE software that has an API.

3.3 The Discrete Feature Program

The method to solve the discrete feature problem, described in section 2.2, was successfully implemented using C++ and the Unigraphics UG/OPEN API. The discrete feature program has the ability to address discrete features that are User Defined Features or UDFs. A programmatic approach that is specific to the case study was used which incorporates the steps to address the discrete feature problem as mentioned in section 2.2. The first step to solving the discrete feature problem is to search the features of the model for a discrete feature. Pseudo code for finding a discrete feature is shown below.

```
loop through all features in the model
       if feature type is a circular or rectangular array
              then address_discrete_feature(feature)
       else
              Do nothing
end loop
```

If an array exists then the discrete feature problem must be addressed. If one does not exist then nothing will happen. The discrete features will have a parent feature instance and in most cases children feature instances. Fig. 7., shows the case study with multiple instances of cylinders as the discrete features that are found by the AMS.

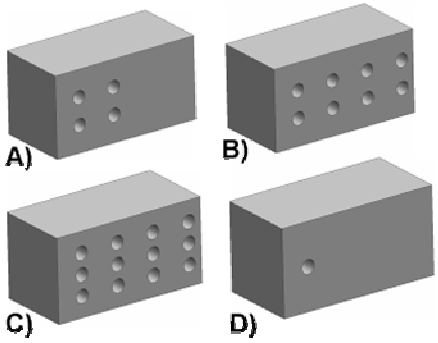

Fig. 7. Instances of cylinder case study.

Next, the parent feature instance is found. Pseudo code for this process is shown below.

```
parent_features_array = parent_features(discrete feature)
loop through parent_features_array
       feature = parent_features_array[i]
       ask_feature_type(feature)
       if feature type is an instance
              exit loop
       else
              do nothing
end loop
```

Finally, all of the attributes are copied from the parent feature instance. Copying the attributes consists of asking for all of the attributes associated to the edges and face entities of the parent feature instance. All of the attributes will be stored to an array to be applied to the children instances.

The fourth step is to collect all of the children feature instances. The fifth step is to apply the attributes to all of the children feature instances. Pseudo code for finding the children instances and applying the copied attributes is shown below:

```
children_features_array = parent_features(discrete feature)
loop through children_features_array
       feature = children_features_array[i]
       apply copied attributes to feature
       end loop
```

This method works for most geometry with only a few exceptions. One difficulty is that success of the method is not guaranteed if the children feature instances go through a topology change. For example, if one of the children feature instances was altered in a way where the hole consisted of a different number of entities than the parent, then the attributes may or may not be copied to the correct face. A test is performed in the AMS that checks the number of entities of the children feature instances against the number of entities of the parent feature instance. If the number of entities does not match then it is likely that this problem has occurred. A warning is issued to the user, the children feature instance in question is highlighted, and the program continues to run. This way the designer can still benefit from the program, but should manually inspect the features involved to make sure attributes have been copied

correctly. Another issue with the method described for solving the discrete feature problem is that it does not work when the discrete features involve user-defined features. To improve the described method to work with user-defined features, additional checks where added.

User-defined features (UDFs) allow the designer the ability to extend the range and power of the built-in geometric features of the CAD system. UDFs allow the designer to create his or her own feature that will automate commonly used design elements. The UDF is a "set" of CAD features, and is represented as only one feature in the feature tree. This "set" of features is what makes the above discrete feature method difficult to implement. Step two (find parent feature instance of the discrete feature) and step three (copy all attributes from the parent feature instance to child instances) of the above method had to be altered slightly. Since the UDF is actually a "set" of features, the program would crash when trying to find the one parent feature instance. To remedy the problem a check was done to see if the parent feature instance of the discrete feature was a UDF. If it was a UDF then all attributes were gathered from all features that make up the "set." Pseudo code for checking if the parent feature instance is a UDF and copying all attributes from the parent UDF set is shown below.

```
parent_features_array = ask_parent_features_of(discrete features)
loop through parent_features_array
       ask_feature_type(parent_features_array[i])
       if feature type is an instance and UDF then
              parent_udf_feature = parent_features_array[i]
              exit loop
       else
       do nothing
end loop
array_of_attributes
array_of_features = ask_all_features(parent_udf_feature)
loop through each feature in array_of_features
       array_attributes = add attributes from current feature
end loop
```

3.4 Airsolid Creation

A programmatic approach was used to create an airsolid for the cylinders region. The ability to create the airsolid for the cylinders region is part of the functionality of PDAA. Creation of airsolids in multiple regions of an assembly does not exist as a part of PDAA and will be developed in the future. Again, a programmatic approach was used that is specific to the case study incorporating the steps that address the airsolid creation methodology as mentioned in section 2.3.

3.5 Command File Generator

The CAD-centric model contains all information needed to create the command files. The CFG interrogates the model for all global and associative attributes, and processes them into a command file for both the mesh and analysis software. This is accomplished by creating an algorithm within the CAD API that will search all global and associative attributes specific to the mesh and analysis software.

4. RESULTS

PDAA, AMS and the discrete feature program were created as discussed in section 2. With these two software tools the objectives, as mentioned in section 2 of this paper, were successfully accomplished. The cylinders concept contains discrete features in the form of the number of cylinders in the air passage. Changing the number of cylinders is accomplished by simply changing the parameter within the component, running the discrete feature program, and updating the airsolid. With this geometry the process consumes about one minute. Mesh results are shown in Fig. 8., and Tab. 1. Initially the mesh was not adequate and changes had to be made to the global and family element sizes, and some faces were re-associated to an alternate family with a different element size. These changes were all done within the AMS and the mesh was recreated by re-executing the updated mesh command file.

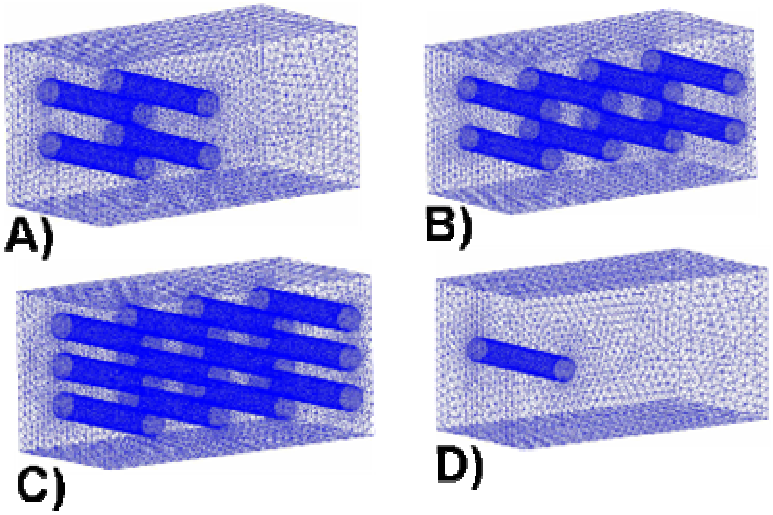

Fig. 8. Cylinders concept meshes.

Concept	*Number of Elements*	*Max Aspect Ratio*	*Max Skew*
A)	89,702	4.09	60.25
B)	370,999	4.30	64.46
C)	231,062	4.10	64.21
D)	32,639	3.62	62.53

Tab. 1. Mesh results.

As you can see in the mesh, the discrete feature program did execute successfully and map all of the attributes for each cylinder. The mesh command file was created and passed successfully allowing the creation of an adequate mesh. This whole process was accomplished in approximately four minutes for the cylinder case. With the CFD model fully configured, the analysis was executed with the results shown in Fig. 9. The AMS proved to be a user-friendly method for creating a CAD-centric model and in addressing the discrete feature problem for the proof case.

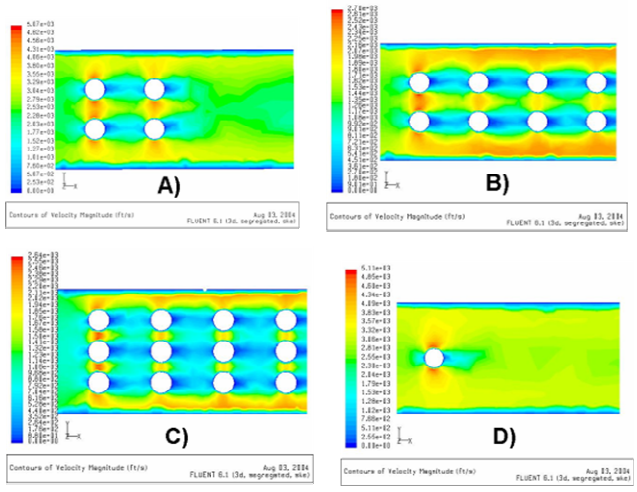

Fig. 9. Contours of velocity magnitude for cylinder case study.

5. CONCLUSIONS

We were able to prove with this research the following:

- A solid model containing attributes can be used as a master model for all downstream applications,
- That reliable automation of the conceptual design, airsolid creation, mesh generation, and CFD setup for complex 3D CAD models can be performed,
- Large amounts of time can be saved by automating the CAD-to-CFD process,
- A special class of topology change called the discrete feature problem can be easily and quickly solved,
- The proposed method can be incorporated on commercially available CAE software through a combination of user interaction and API programming.

It was found that a CAD-centric model with discrete features can be created and used for downstream applications. Furthermore, the time required to generate a mesh and analysis model is drastically reduced compared to the over-the-wall method. The method developed in this work will help industry to shorten the design cycle time where analysis is involved. The method also allows for analysis to become part of concept generation and opens the door for optimization to be used in the conceptual design stages.

Presently, we are researching how to apply these principles to the analysis of full assemblies, specifically a full gas turbine engine. Methods are needed to address the problem of inter-part relations and topology changes. Because of the time constraint associated with creating a mesh through an entire engine, global parameters need to be adjusted so that optimization routines can refine the mesh parameters on individual sets of airfoils while still maintaining the integrity of the flow analysis. Research has been done related to storing these parameters in a database[1] and we are looking into using user-defined objects to facilitate automatic updating of parts when parameters are altered. With these tools we can greatly improve the efficiency of assembly analysis.

6. REFERENCES

[1] Baker, T., *Attribution Standardization for Integrated Concurrent Engineering*, M.S. Thesis, Brigham Young University, 2005.

[2] Bailey, M. W., Rohinton, K., Irani, P. M., Finningan, P. J. and Rohl, K.B., (2000), Integrated multidisciplinary design, *NASA HPCC/CAS Workshop/NASA Ames Research Center*, 15-17 February 2000.

[3] Hogge, D., *Integrating Commercial CAx Software to Perform Multidisciplinary Design Optimization*, M.S. Thesis, Brigham Young University, 2002.

[4] Samareh, J. A., Status and Future of Geometry Modeling and Grid Generation for Design and Optimization, *Journal of Aircraft,* Vol. 36, 1999, pp 97-104.

[5] Srinivasan, H., Jensen, C. G., Kopper, F. C., Staubach, J. B. and Pack, D. R., Assembly Parametrics for Multidisciplinary Design Optimization. *8th International Society for Productivity Enhancement Conference on Concurrent Engineering, Anaheim, CA*, 2001, pp 158-165.