# Analysis Model Visualization and Graphical Analysis Attribute Specification System

**Robert M. O'Bara, Mark W. Beall and Mark S. Shephard**

**Scientific Computation Research Center**
**Rensselaer Polytechnic Institute, Troy NY, 12180**

**Abstract**
A visual environment for defining and manipulating engineering analysis information has been developed. This environment: (i) allows queries and modifications of the topology and geometry that defines a geometric model obtained from various geometric modeling systems, (ii) abstracts the modeler's functionality needed to associate analysis information, (iii) provides a hierarchical attribute association model, and (iv) gives a graphical user interface to both the geometric modeler abstraction and attribute management system. Finally graphical issues relating to performance, portability and flexibility of different workstation environments are discussed.

## Section 1 Introduction

Today there exist many commercial modelers that are used to define the geometric domains of engineered products. These packages can be classified into the following two groups:

1. A set of library routines which allow a programmer to construct a model and make inquires. Examples of such libraries are SHAPES [1], Parasolid [2], and ACIS [3].

2. A modeling environment that provides an interactive graphical interface for model construction and an underlying set of geometric modeling routines. Examples of such environments are Unigraphics [4] & CATIA [5]

The programming, user interfaces, and functionality of these modelers can vary greatly from system to system. This variation in the interfaces makes it difficult for both users and programmers to be able to switch between different modeling systems as needed. There has been work done to construct PDES/STEP [6], which is a standard common data description that is modeler independent. However, access to the geometric data is not sufficient for more advanced analysis frameworks which need to modify the geometry of the model for analysis, append analysis attribute information, and generate numerical analysis discretizations. An abstraction of a geometric model and allows the interrogation, analysis specification, and modification of the model would provide an analyst or programmer a tool which is independent of any particular modeler, thus making the transition from one modeler to the other relatively simple. A modeler independent graphical user interface is also desirable to allow users to perform the operations required for model modification and attribute specification. Without such an abstraction the interface between modelers and analysis packages that use them can end up looking like figure 1, where each package has its own interface and much work must be done to port the system to a different modeler. The abstraction, presented in this paper, simplifies the interface to the model and the analysis atrributes as shown in figure 2.

The definition of an engineering analysis problem consists of the geometric domain and the "analysis attributes", consisting of loads, boundary conditions, material properties, and initial conditions. The analysis attributes are best related to the model by associating them with the topological entities in the model [7]. This can be done outside of the modeling package with the information being stored in a database system [8][9]. However, in some cases, proper specification of the analysis attributes may require additions and/or modifications of the geometric model. Since many of the models are three dimensional, these modifications will also need to be viewed in 3D along with the visualization of the attribute information itself. In addition, the attributes typically have relations among themselves which need to be viewed and maintained.

This paper describes a modeler independent interface being implemented to address these issues. In designing the interface to the modeler, an object-oriented approach is used that allows the tight integration of both data, that represents the topological entities, and functionality, such as inquiries and modification operations. The resulting abstraction is then mapped to specific modelers. The abstraction is also used in associating analysis attributes as well as designing graphical interfaces.

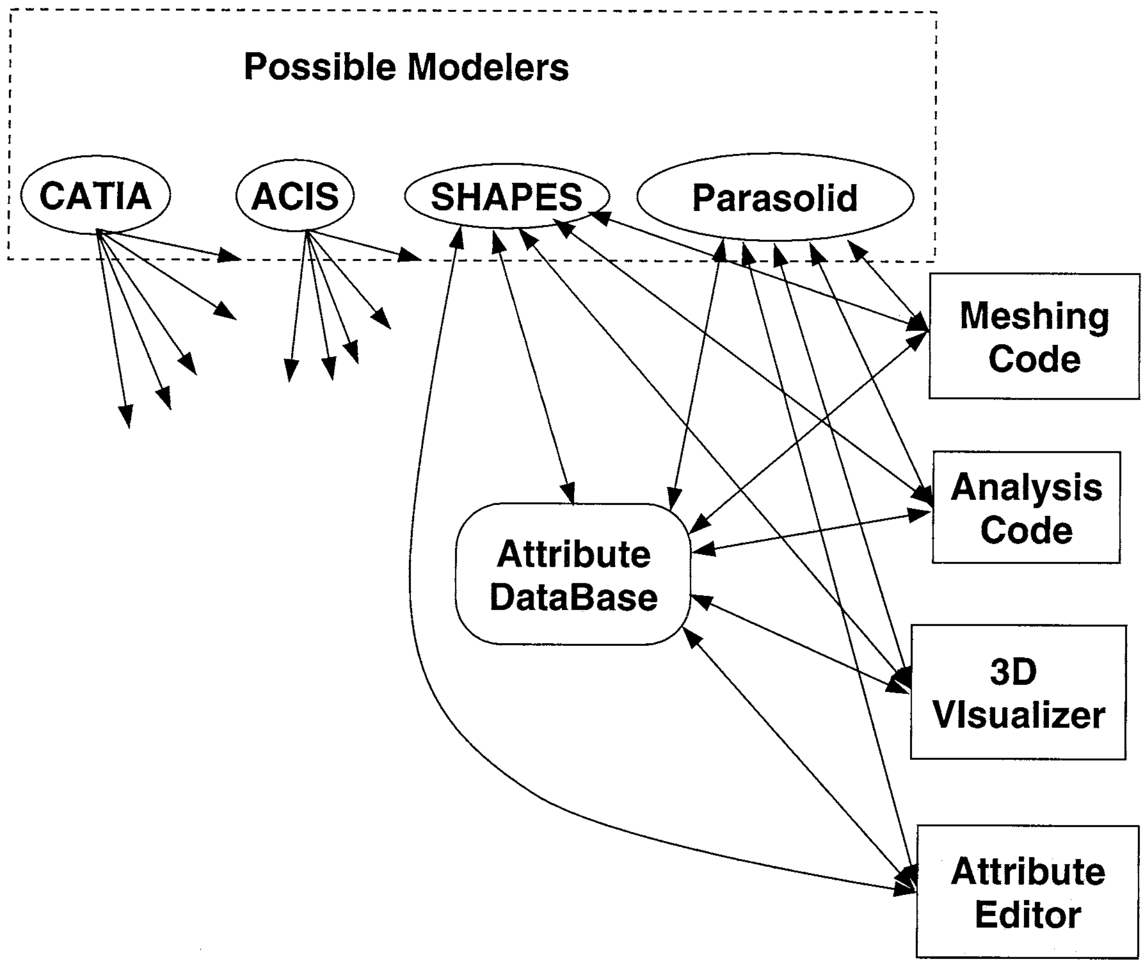


Figure 1. Combinatorics problem resulting from using different modelers and separate attribute databases. Each package has its own interface to each modeler.

By using an object-oriented approach, an unified system that represents the various levels of geometric models as well as the attribute information has been developed. This abstraction, called the Attributed Geometric Model, is used by the analysis process, as well as by visual interfaces (see figure 2).

Two user interfaces that work with the Attributed Geometric Model are discussed. The first is the Model Graphical Interface which is a modeler independent user interface that allows interaction with a geometric model. The interface provides basic 3D displaying controls such as lighting, color selection, and filtering out unwanted geometry.

The second is the Attribute Graphical Interface which allows attribute specification/modification through an intuitive graphical interface in which the user enters attributes via a set of visual widgets. These widgets provide initial syntax checking and can be specialized to provide semantic checking. The current implementation uses 2D widgets for attribute specification. With the integration of the 3D model visualizer, attributes can be visualized

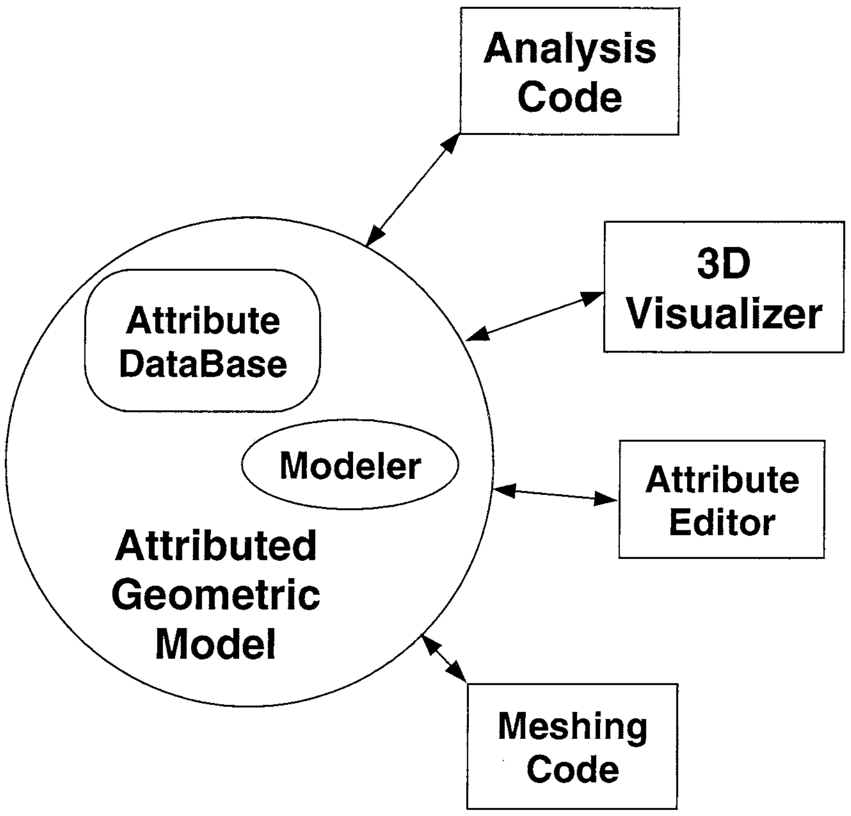


Figure 2. Unified object-oriented approach that provides a common interface to both geometric and analysis information and eliminates the combinatorics problem.

relative to the geometric model. This includes visualizing the auxiliary geometry that maybe associated with the attribute as well as the attribute itself.

The hierarchical relationship between attributes also needs to be visualized in order to allow a user to modify association between collections of attributes. The current design is to view the representation as a graph which represents the relations between attributes. Modifications to the attribute relations can be made by directly modifying this graph-based representation.

The remainder of the paper describes the design of the complete system which includes the abstractions of both the geometric model and analysis attributes, as well as, their graphical user interfaces. Issues regarding hardware and software environments are also presented.

## Section 2 System Design

The overall system is broken down into four parts: the geometric model abstraction, the attribute abstraction, the model graphical interface, and the attribute graphical interface. The first two parts are abstractions used to hide the implementation details of a particular modeler and attributing system, and to provide a consistent programming interface. The second two parts are user interfaces, that build on the abstractions, to allow users to interactively view and modify model and attribute information.

## Object-Oriented Model Abstraction

The object-oriented abstraction presents an unified view of an attributed geometric model which allows a programmer to modify geometry/topology, and attribute information. In addition the implementation of the abstraction for a given modeler may also increase the functionality of the modeler. An example of this is enhancing a 2–manifold modeler to be able to represent non-manifold models.

In order to abstract the Attributed Geometric Model, both the geometric model, which is maintained by the modelling environment, and the attribute database, which may be integrated with the modeler or implemented by a set of external routines, must be abstracted as well as their association with each other. The abstraction used for the modeler must be very general to be able to encompass the functionality of any modeler that it is implemented for. For this reason an abstraction based on the Radial-Edge Data Structure[10][11] is used. This representation has been shown to be complete and sufficient for the representation of general non-manifold models.

For a complete description of the Radial-Edge Data Structure see references [10] and [11]. In brief terms it is best described as a topological hierarchy consisting of regions (three dimensional entities that are bounded by shells), shells (sets of faces that define a closed surface), faces (two dimensional entities that are bounded by loops, loops (sets of edges that form closed curves), edges (one dimensional entities that are bounded by vertices) and vertices (zero dimensional entities). For the remainder of this paper the term "topent" will be used to refer to any one of these topological entities.

The actual abstraction is done in terms of the topents that make up the model. There are objects that represent the regions, shells, faces, loops, edges and vertices. There is also an object that represents the model which acts as a container for the topent objects. The objects for the topents and the model are the entire public interface for the model abstraction and completely hide the modeler for which the interface has been implemented, as shown in Fig. 3.

The interface for the model and the topents is given below. These operators, especially the geometric query operators, reflect a bias to the operators needed for automatic mesh generation, as this is one of the first areas that the modeler abstraction is being used. The list of operators is expanded as needed to fit other application areas.

1. Model Operators

   a. Query

      - Get_top_level() — returns a list of topents that represents the top level topology (topents that are not connected to a higher dimension topent).
      - Get_all_{topent_type} — returns a list of all topents in the model of that type. For example, Get_all_vertices().
      - Get(topent_name) — returns the topent(s) that have the name "topent_name".

- Get_number_of_{topent_type} — returns the number of topents in the model of a given type.

b. Modification

- Add() — add a topent to the model
- Remove() — remove a topent from the model

2. Topent Topological Operators

a. Query

- Sub() — returns a list of topents that are used in the definition of that particular topent. For example, shell1.sub() returns a list of faces that form shell1.
- Sup() — creates a list of all topents that the given topent is used in the definition of. For example, vertex1.sup() returns a list of edges that use vertex1.
- Get_type() — returns the type of the topent.
- Adjacent() — checks if one topent is adjacent to another, i.e. face1.adjacent(edge1) checks if edge1 is being used by face1.

b. Modification

- Attach() — attach one topent to another. For example shell1.attach(face1) adds face1 to the list of faces that defines shell1.
- Detach() — detach one topent from another.

3. Topent Geometric Operators

a. Query

- Point evaluation function — Since all topents are associated with parameterized geometry, the must be some way of calculating point evaluations. For example, edge1(t0) would return a point that corresponds to edge1's parametric function evaluated at t0.
- Closest_point() — Given a point in space, find the closest point on the given topent.
- Normal() — Returns the normal space of a topent at a given point.
- Tangent() — Returns the tangent space of a topent at a given point.
- Tolerance() — Returns the modeler tolerance associated with a particular topent.
- Range() — Returns the range of the topents parametric space.
- Find_intersection() — Finds the points of intersections on a topent of the topent and a geometric construct such as a line or plane. Only finds the location of the intersections does not modify the model.

- Geomerty() — returns a geometric representation of the topent suitable for visualization purposes.

b. Modification — This is only a partial list.

- Create() — create topents
- Delete() — delete topent
- Intersect() — return the intersection of two topents
- Union() — return the union of two topents
- Split() — split topent with another topent (intersect the two topents and separate the target topent at the intersection)
- Imprint() — imprint topent on another topent (project one topent onto another of equal or higher order and split at the boundary of the projection)
- Merge() — merge two topents into a single topent

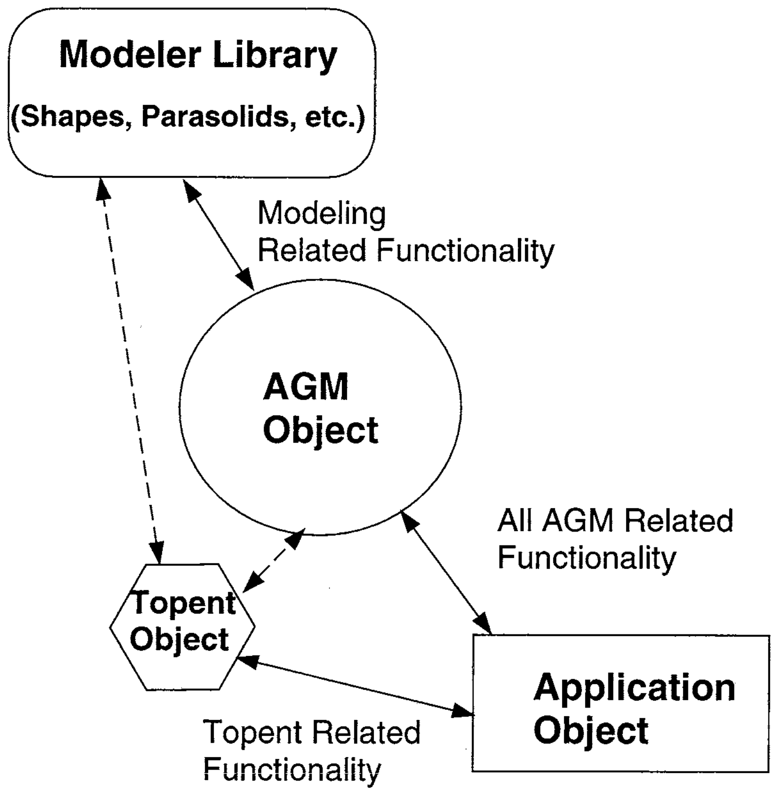


Figure 3. Approach to the Attributed Geometric Model / modeler relationship problem

**Implementation** The amount of work that needs to be done to implement the interface for any given modeler depends greatly on the capabilities of the modeler and on whether is it necessary to expand those capabilities. For a modeler that is capable of representing nonmanifold models, the job is much more straightforward than for one that is not capable of such a representation assuming one of the goals is to be able to represent nonmanifold models using that modeler. Our initial implementation uses Shapes from XOX Corp. [1] as the modeler since it matches up well to the abstraction that has been selected for the modeler. In this case it is often just a matter of matching up application programming interface calls in Shapes to the corresponding Attributed Geometric Model routines.

The next modeler that has been interfaced to the Attributed Geometric Model is Parasolid from Shape Data [2]. The current version of Parasolid is not capable of representing nonmanifold models. However, Parasolid has been used as a nonmanifold modeler by utilizing an interface on top of Parasolid that keeps a Radial Edge representation of the topology of the model that is independent of the representation stored by Parasolid itself.

## Analysis Attribute Model Abstraction

An analysis attribute is any information in addition to the geometric model that is needed to specify a particular problem for analysis [8]. Many attributes, such as loads and material properties, are tensorial in nature. Other attributes may be best described using a character string. Every model entity may have one or more attributes associated with it. Using this association of the attributes with the geometric model and information which gives the classification of the finite element mesh with respect to the geometric model (what entity of the model a mesh entity is associated with), it is possible to determine which attributes apply to what entities in the finite element mesh. This methodology has been found to be very powerful when dealing with an adaptive finite element environment.

In addition to the association of the attributes to model entities, attribute information is also grouped into a "part of" hierarchy consisting of the following classifications:

1. Case — a collection of one or more Groups, Sets or Attributes
2. Group — a collection of one or more Sets or Attributes
3. Set — a collection of Attributes of the same type
4. Attribute

Attribute groupings form acyclic directed graphs with cases at the root and individual attributes at the leaves of the graph. In addition, the arcs in the graph may contain a multiplier that is applied to all descendent nodes connected by that arc (Fig. 4).

The information stored in an attribute consists of a tensor, where each component may be an arbitrary function, a list of associated topents, and an unique identifier. The tensor may also have symmetry properties that reduce the number of independent components.

The proper evaluation of an attribute may require access to the definition of the geometry of the topent that it is associated with. An example of this would be a load that is defined to be normal to the surfaces on which it is applied. To evaluate the vector that represents this load requires finding the normal to the surface at each point it is to be evaluated at.

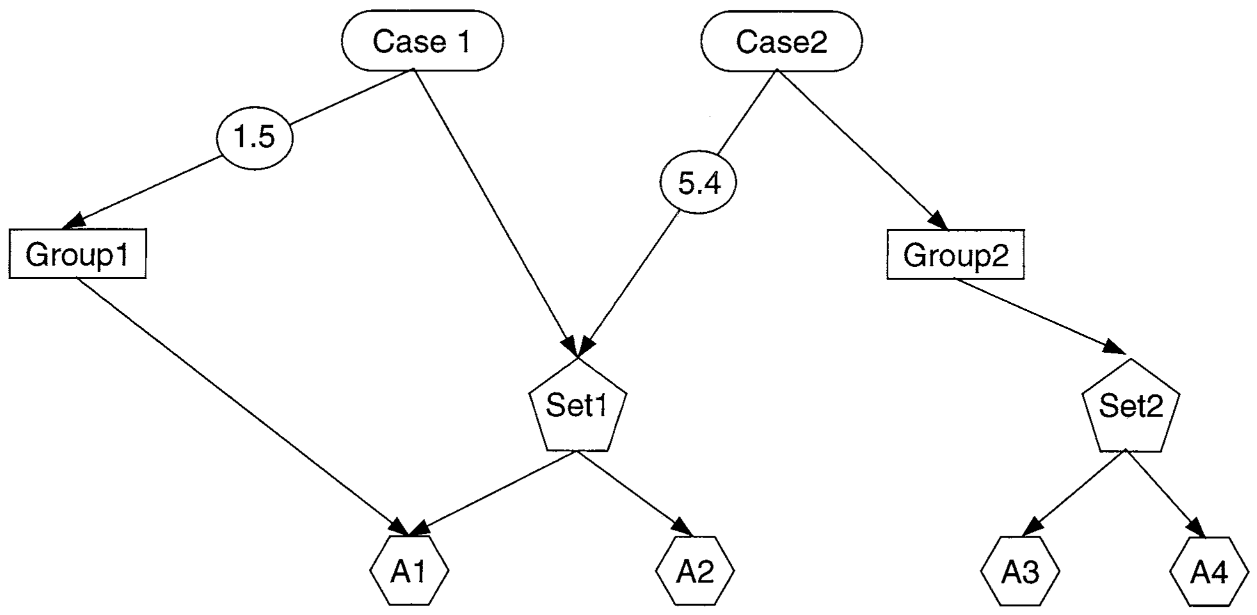


Figure 4. Sample Attribute directed acyclic graph

In some cases it may be necessary to modify the topology of the geometric model to properly reflect the application of an attribute. An example of this situation is when an essential boundary condition is applied over a portion of a topent. In this case, to properly analyze the situation it is necessary that the boundaries of the elements that descretize the topent properly reflect the boundary of the essential boundary condition. The only way to ensure this is to split the topent along the boundary of where the attribute is applied.

Both of the cases above require access to the modelers functionality to properly deal with analysis attributes. It is easy to see how a generic interface to the modelers functionality reduces the effort of implementing such a functionality with multiple modelers.

The abstraction of the attribute database is relatively straight forward. Attribute entities (or atents), such as cases, groups, sets, and attributes, have interface calls which include:

1. Add(atent, multiplier) — adds an atent and zero or more multipliers to a grouping. For example case1.add(attribute1, mult1)
2. Remove(atent) — removes an atent from a grouping
3. Get_children() — get the children atents along with their multipliers.
4. Get_parents() — get the parent atents along with their multipliers.
5. Add(topent) — adds a topent to an attribute
6. Remove(topent) — removes a topent from an attribute
7. Evaluate() —returns the evaluated tensor which describes that attribute at a particular location

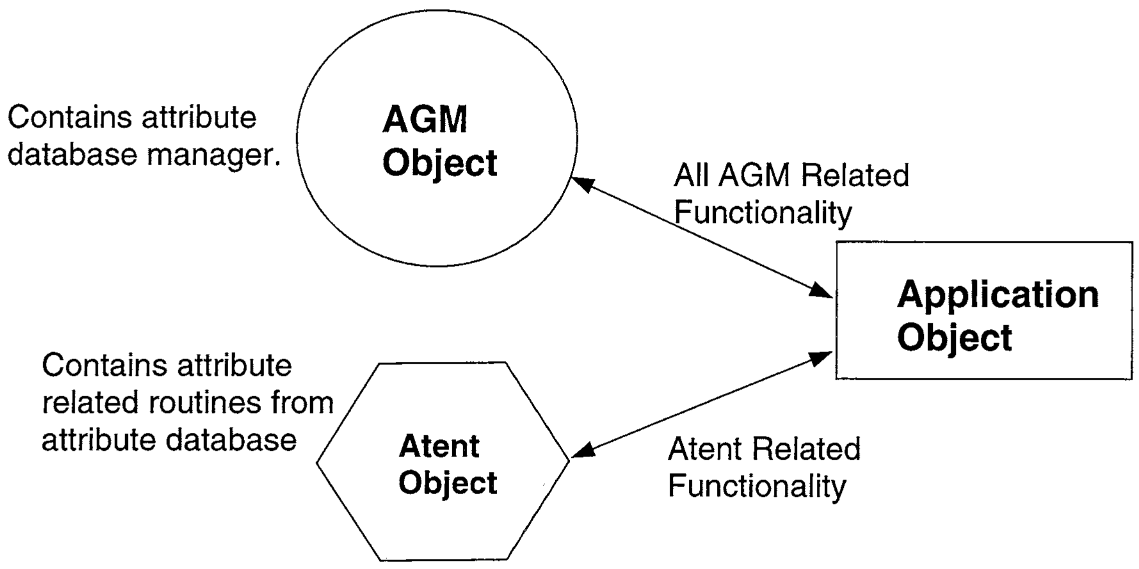

Contains attribute database manager.
AGM Object
All AGM Related Functionality
Application Object
Contains attribute related routines from attribute database
Atent Object
Atent Related Functionality

Figure 5. Approach to the Attributed Geometric Model / attribute relationship problem

## Model Graphical Interface

Some modelers are only accessible via a routine library and do not provide direct visual feedback to the user. Other environments provide graphical interfaces; however, they tend not to provide a uniform "look & feel" across different modelers. By providing a "modeler independent" 3D graphical user interface, users are able to view and interact with models without the need to learn multiple systems. In addition, when the system is "integrated" with the Attribute Graphical Interface, discussed in the next section, users are able to select topological entities in 3D and inspect the associated attributes as well as select attributes and examine the associated topological elements.

The Model Graphical Interface provides a means of spatially viewing the geometric representation of the model. Based on the degree of 3D graphics acceleration available on the workstation, the types of renderings produced by the Model Graphical Interface include the following:

1. Wireframe
2. Gouraud shaded surfaces [12]
3. Texture-mapped surfaces [13]

Texture mapping is a rendering technique that visualizes additional information associated with surface geometry and is useful when visualizing attribute information. For example, a pressure distribution can be use to texture a surface.

In addition to producing smooth shaded, hidden surface/hidden line images, the Model Graphical Interface also provides a more realistic visualization of the model in terms of depth perception by providing stereo viewing. Stereoscopic viewing involves rendering two different images which correspond to the different views seen by the right and left eye of the viewer [14]. These images are presented to the viewer using a device that allows each eye to see the image created for it. The device currently being used is a liquid shutter system that alternately "black-out" one of the eyes, and is sychronized with the display's refresh rate. The speed at which this done is fast enough to produce images with no perceived flicker. The two images are then fused by the viewer's cognitive system into a spatially perceived 3D scene. When used in conjunction with a head tracking system, the viewer has the illusion of a solid 3D object suspended in space.

The Model Graphical Interface can render surfaces, curves, and points. The mechanism used to extract a topent's geometry to be visualized is via the Geometry() member function. Traditionally in the case of curve and surface geometry, the modeler would generate a discretized first order approximation (polylines and polygons). Current graphics libraries often contain higher order primitives such as non-uniform rational B-Spline (NURBS) curves and surfaces [15], as well as, quadrilateral and triangular meshes [16][17][18][19]. The Model Graphical Interface has been designed to make use of these higher-order primitives by passing them directly to the graphics engine when appropriate. These primitives allow the application to exactly specify (or at least better approximate) the geometry. In addition, the use of these higher-level primitives can dramatically improve the time required to visualize the geometry. For example, to transform a bi-cubic surface requires the transformation of only 16 control points instead of 100 triangles typically used to approximate it for display by polygons. In addition, some of these primitives are accelerated in hardware. For example, almost all 3D accelerators are very efficient at processing triangle strips, and some of the latest accelerators such as SUN Microsystems' ZX [20] can process NURBS curves and surfaces in hardware.

In terms of the user interface, the Model Graphical Interface provides the following:

1. The ability to interactively change a viewer's position and orientation in space as well as allowing multiple views. The mechanisms of specifying the view include:
    a. A virtual trackball to specify the orientation of the viewer [21].
    b. Use of a 6D tracking system that can determine the position and orientation of the user's head [22].
    c. Selecting specific topents and tell the system to "look at them" [23].
2. Control over several light sources to provide better spatial perception as well as obtaining a better feel for the shape of the geometry.
3. The ability to specify color / optical (such as shininess) attributes to different topents, including back-facing attributes.
4. The ability to control transparency of topents. This could is useful when dealing with model that have internal structure such as regions.
5. The ability to hide different topents in order to view internal geometric structure.
6. In addition to the original model geometry, the Model Graphical Interface is able to display the following representations:

   a. Augmented geometry that results from the application of different cases of analysis attributes.
   b. Simplified or idealized geometry

7. Control topent selection based on the following criteria:

   a. Spatially
   b. Name of the topent
   c. Type of the topent
   d. Association with another topent.

8. An interface to model modification functions such as solid modeling intersection and union operators.

9. A graph-based representation of the model's topology and allow selection / modification operations by interacting with the graph.

10. A textual representation of the model using 2D window-based widgets.

The Model Graphical Interface provides the user with backface functionality in order to better distinguish the orientation of faces and shells. In terms of a polygon, the backface refers to the side of the polygon in which the surface normal points away from the viewer. By allowing the backface to have different properties from the frontface (the side whose surface normal points towards the viewer), the user can determine relative orientation. It can also be used to determine inconsistencies in the specification of the geometry. In addition, the user can remove all pieces of geometry that are currently backfacing. This functionality is referred to as backface culling [16] and in the case of viewing closed surface objects, this can increase system performance dramatically.

Backface Culling is one way of reducing the load placed on the graphics subsystem and can be very important when dealing with lower-end systems. In order to further reduce graphics complexity the Model Graphical Interface provides alternative representations of the geometric model such as a graph-based representation of the model's topology as well as a textual representation. In addition to being less graphics intensive, the textual form can be more intuitive in terms of modifying model information. For example, it may be much easier to type in a vertex's coordinates than it is to try to precisely pick it with the mouse.

Besides efficiency, another important issue is how topents can be selected by the user. The most intuitive method is spatially by either selecting a particular topent or by defining a region in space and thereby selecting all topents that lie within that region. Since topents are spatially connected, it may be very difficult to select a particular topent without accidently choosing it's spatially neighbor. For example, how does a user select a vertex without choosing the edge or face that is connected to it. The solution used in the Model Graphical Interface is to provide filtering in the selection process. Topents can be filtered out based on their name, type, or association with another topent. For example, a user can specify that all topents associated with the named "Region1" are not selectable. In addition to spatial selection, topents can be selected based on their name. This mechanism has been extended so that users can specify text patterns which can include "wildcards" in order to select several topents at the same time. A user can also select topents based on topological associativity. For example, a user can select all vertices that are associated with a particular face. Finally, a user can select topents via their association with attribute information by using the Attribute Graphical Interface.

**Examples of Model Visualization** The first commercial modeler that has been integrated into the Attributed Geometric Model Abstraction has been the SHAPES modeler from XOX Corporation[1]. Figure 6 shows a surface rendering of a SHAPES model of an oil platform that was being viewed via the Model Graphical Interface. Figure 7 was the result of hiding all of the faces of the model. This figure shows all of the edge contours of the model as well as the model vertices. Figure 8 shows a surface rendering of a Parasolid model of an electrical part from PDA.

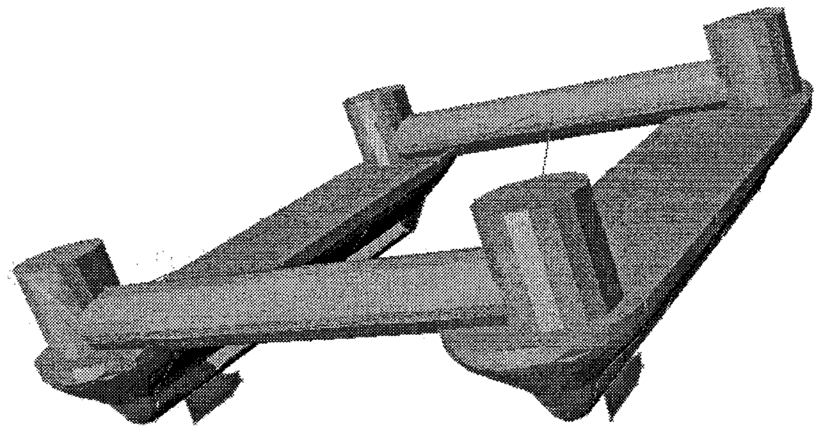


Figure 6. Model Graphical Interface display of a model of an oil platform using the SHAPES geometric modeler. Note the nonmanifold edges that represent the guy wires and the non-manifold faces that represent the rudders. Model courtesy of XOX Corp.

To show the flexibility of both the model abstraction and the graphical interface, a finite element modeler, which was developed by the Scientific Computation Research Center at RPI, was also abstracted and visualized by the interface. Figures 9 and 10 show two different views of a mechanical part. The first figure shows all of the faces and vertices in the model, while the second figure shows the edges of the model. Figure 11 shows a zoomed in view of the part's faces and edges.

One of the important features of the interface is the ease of which it can be customized. In the case of the finite element modeler, functionality was added to view the octree representation in addition to the model itself, as shown in figure 12. As in the case of viewing topents, the display of the octree can be controlled interactively. In figure 13 only the octree structure is being viewed . As previously mentioned, the ability to control the display of the model is an important feature of the interface, as shown in figure 14. This figure shows the same view as figure 11, but also displays the octree model.

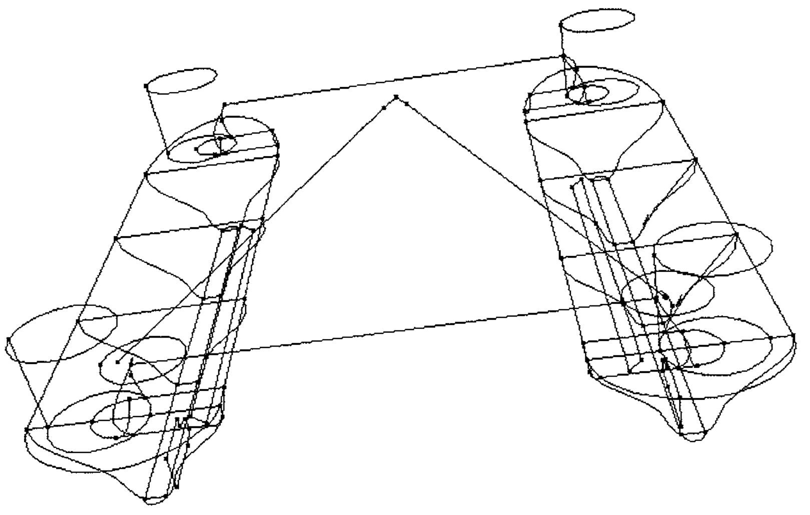


Figure 7. Model Graphical Interface display of only the edges and vertices of the oil platform model

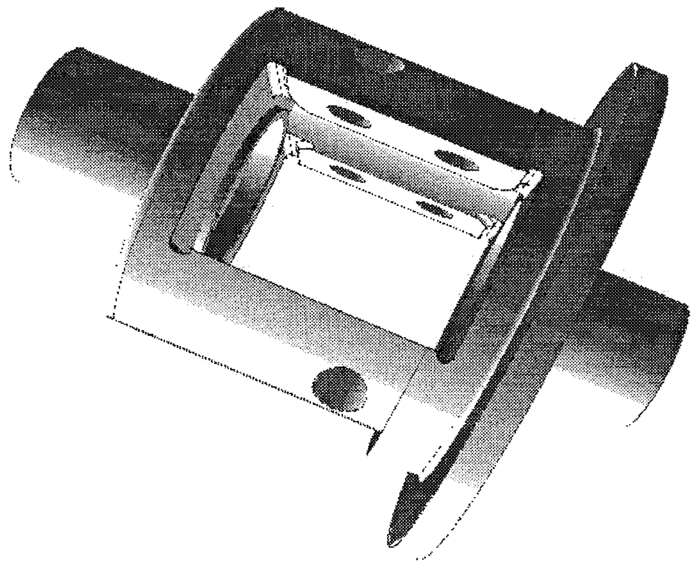


Figure 8. Model Graphical Interface display of an electrical part using the Parasolid geometric modeler. Model supplied by PDA

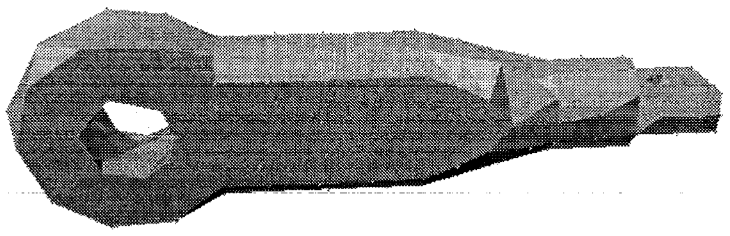


Figure 9. Finite element mesh showing faces

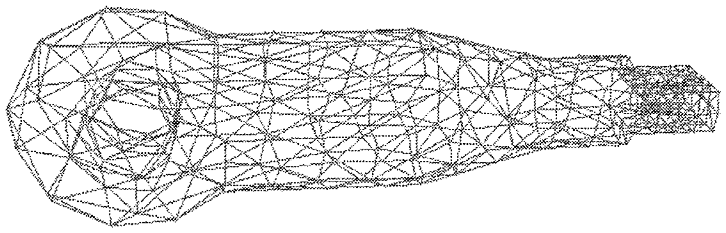


Figure 10. Finite element mesh showing edges

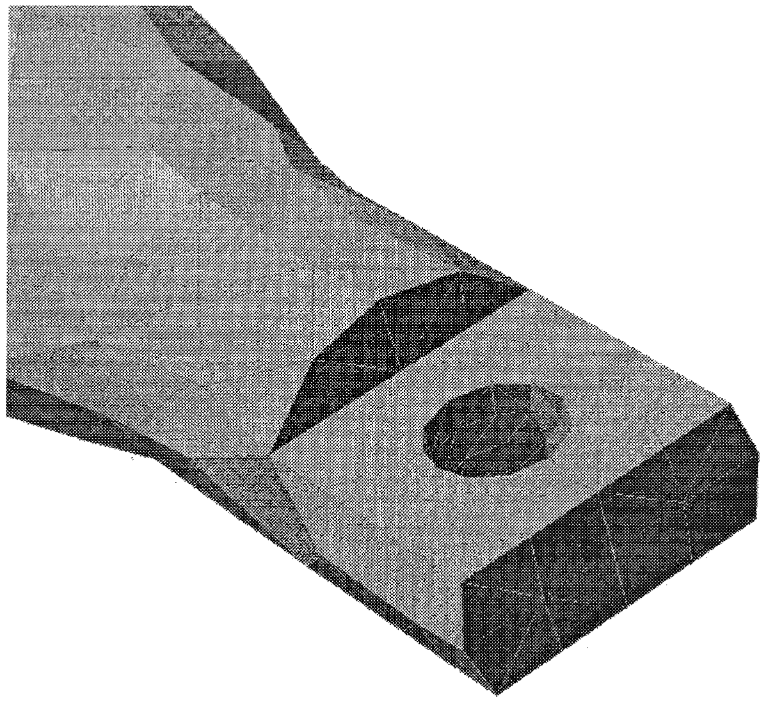


Figure 11. A magnified view of the part

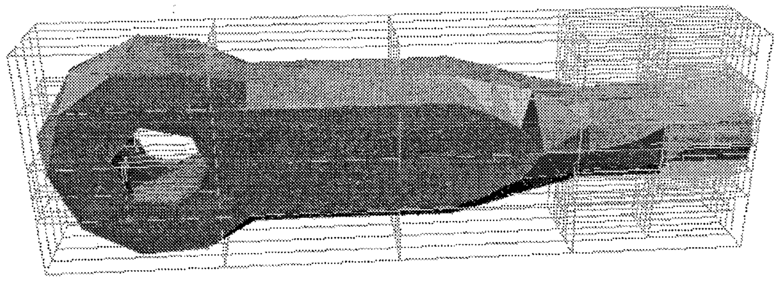


Figure 12. Finite element mesh showing faces, edges, and its associated octree.

Another important feature that was added in the finite element version of the interface was the ability to the execute the interface from a program in order to use it as a debugging tool for automatic mesh generation code. Figure 15 shows the control panel to the debugger version.

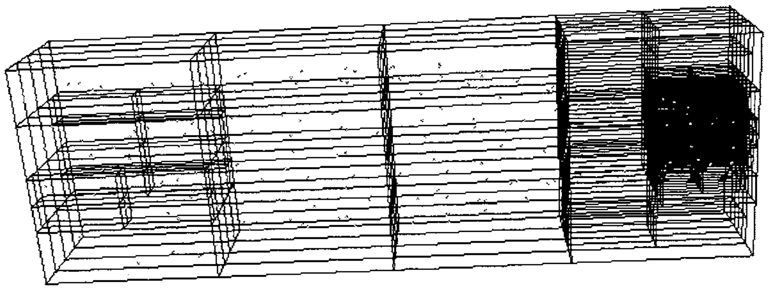


Figure 13. Display of octree used in creation of previous finite element meshes

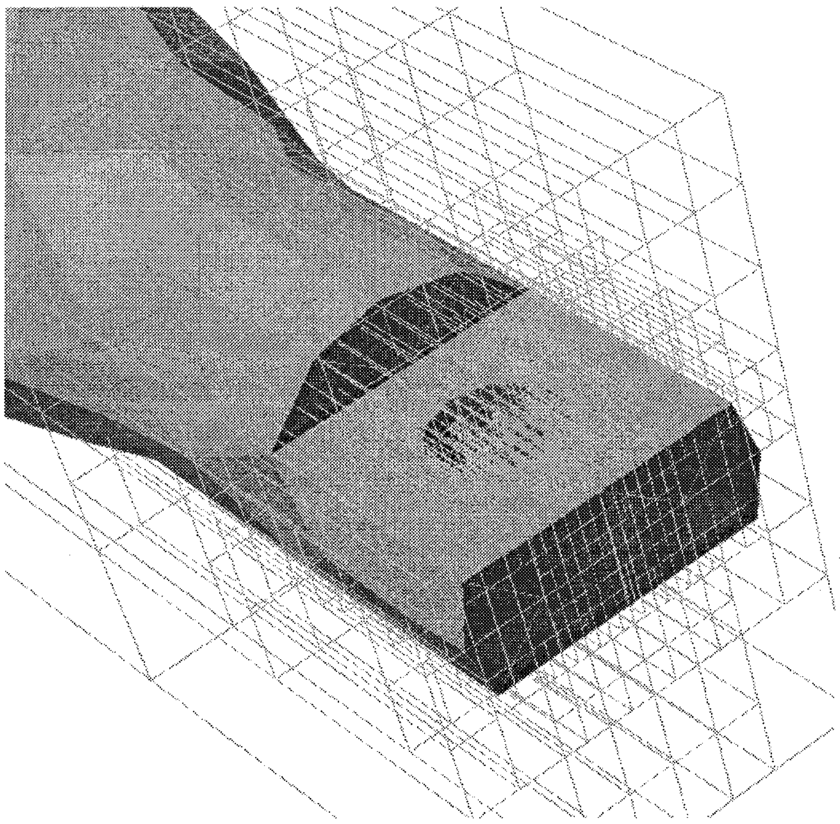
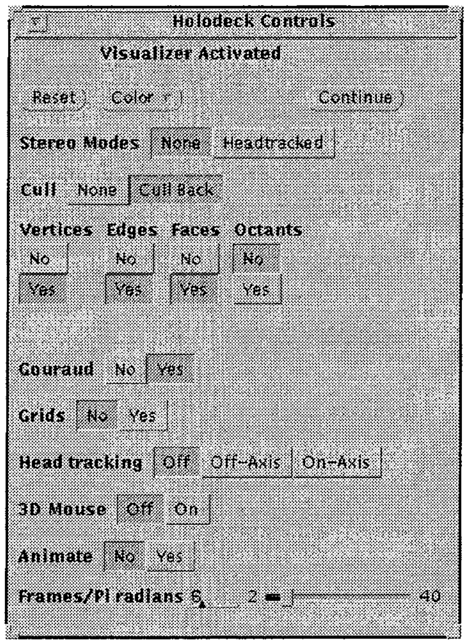


Figure 14. A magnified view of the part and it's associated octree.

The continue button, located in the upper right of the panel, allows the developer to transfer control back to the program while still viewing the model.

## Analysis Attribute Graphical Interface

The Attribute Graphical Interface serves as a graphical interface used to define, view, and modify analysis attributes and their association with topents in the geometric model. The Attribute Graphical Interface also addresses the clustering of attributes into the hierarchical structure. Unlike the Model Graphical Interface, the Attribute Graphical Interface involves more use of 2D visualization techniques such as icons and text since the information tends to be



Stereo Modes [✓ None] Headtracked
Cull None [✓ Cull Back]
Vertices No [✓ Yes]
Edges No [✓ Yes]
Faces No [✓ Yes]
Octants [✓ No] Yes
Gouraud No [✓ Yes]
Grids [✓ No] Yes
Head tracking [✓ Off] Off-Axis On-Axis
3D Mouse [✓ Off] On
Animate [✓ No] Yes
Frames/PI radians 6 2 40

Figure 15. Mesh Debugger Version of the Model Graphical Interface's Main Control Panel (for high performance 3D platforms)

more mathematical or relation oriented. However, the attribute/topent relationship as well as physical ramifications of the attribute do have spatial components that need to be visualized in 3D. In those cases, there is less distinction between the Model Graphical Interface and Attribute Graphical Interface.

As with the Model Graphical Interface, one of the most important issues is controlling the selection process. In the case of attributes, one approach to doing this is via a list, see figure 16. Since the number of choices may be very large, filtering mechanisms had been added to help structure the information. For example, the choice list can be filtered based on the type of attribute entity (atent) or the specific type of an attribute. While building attribute hierarchies, atents will not always be "properly associated", which refers to atents which do not have a case as an ancestor and/or an attribute that is associated with a topent as a descendant. The Attribute Graphical Interface can provide a list of such entities as well as what association is missing. An atent can also be selected by entering it's name or a pattern which then selects all atents that match. Another way in which atents can be selected is via a graph representation of the hierarchy with the nodes representing the atent and arcs showing the associativity (see figure 17). This form is very useful in changing atent associations. An atent can also be selected via its relationship with another atent (see figure 18). Finally, an attribute can be selected by it's association with a topent that has been selected via the Model Graphical Interface.

In addition to modifying an atent's association via a graph representation, a user can change the hierarchy via a selection panel as shown in figures 19 and 20. In this method, a user is

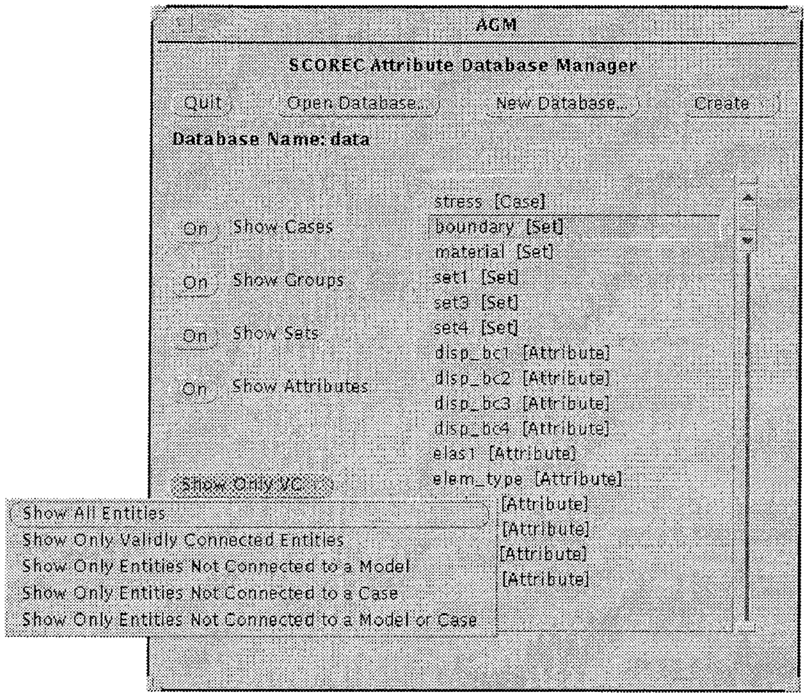


Figure 16. Attribute Graphical Interface's Main Attribute Manager Panel. The buttons on the left side which control filtering based on type as well as association.

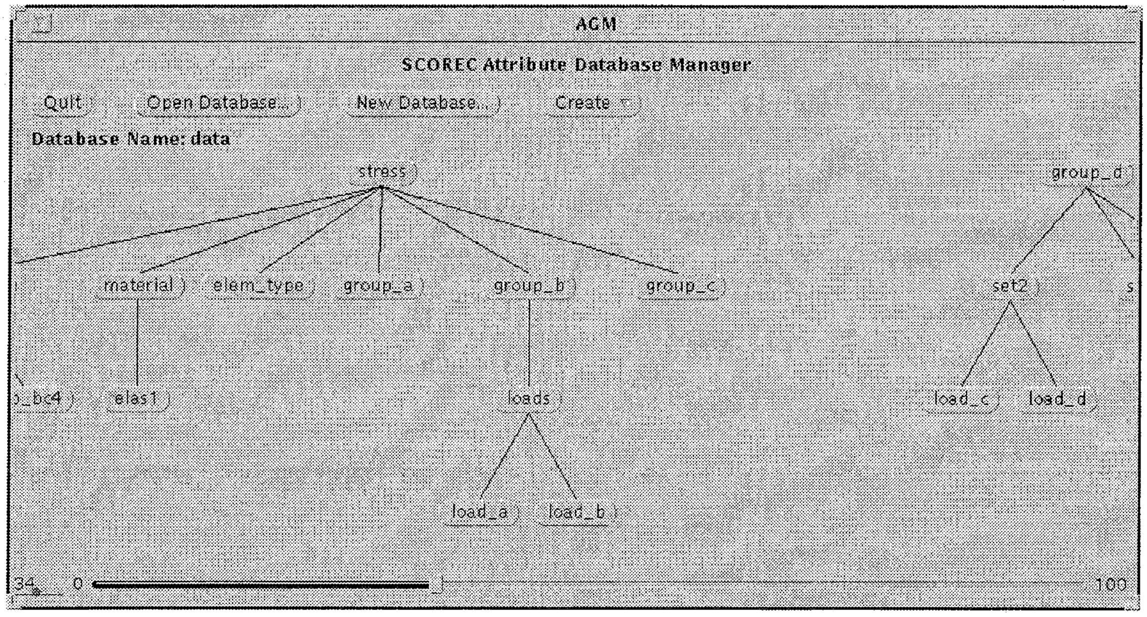
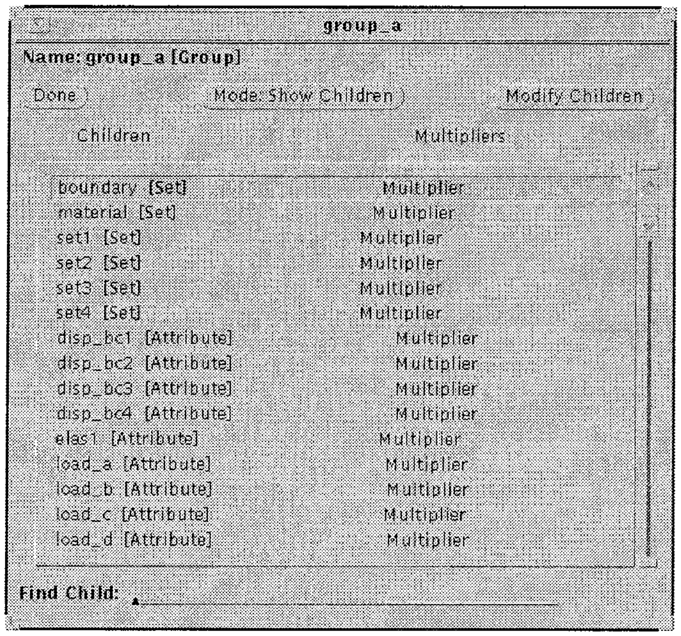


Figure 17. A graphical representation of an attribute hierarchy show all of the associations between attributes. A user can modify associations by changing the arcs between attributes.

presented with a list of possible children for a specific atent, as well as it's current children list. The user can then modify the multiplier values between the associations.



group_a

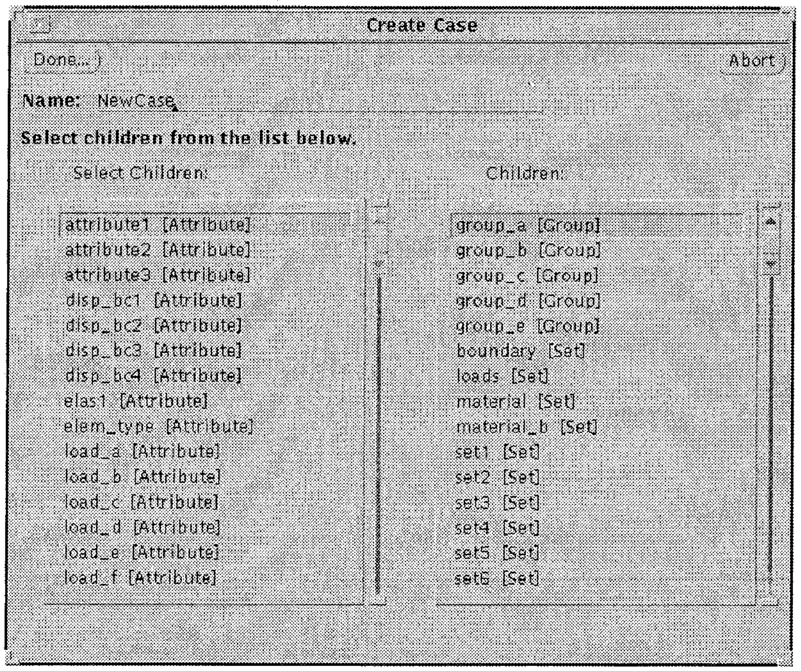
Name: group_a [Group]

Done ) Mode: Show Children ) Modify Children )

| Children | Multipliers |
|---|---|
| boundary [Set] | Multiplier |
| material [Set] | Multiplier |
| set1 [Set] | Multiplier |
| set2 [Set] | Multiplier |
| set3 [Set] | Multiplier |
| set4 [Set] | Multiplier |
| disp_bc1 [Attribute] | Multiplier |
| disp_bc2 [Attribute] | Multiplier |
| disp_bc3 [Attribute] | Multiplier |
| disp_bc4 [Attribute] | Multiplier |
| elas1 [Attribute] | Multiplier |
| load_a [Attribute] | Multiplier |
| load_b [Attribute] | Multiplier |
| load_c [Attribute] | Multiplier |
| load_d [Attribute] | Multiplier |

Find Child:

Figure 18. Attribute Graphical Interface's Attribute Collector Panel (In this example, the collector is a Attribute Group)



Create Case

Done... ) Abort )

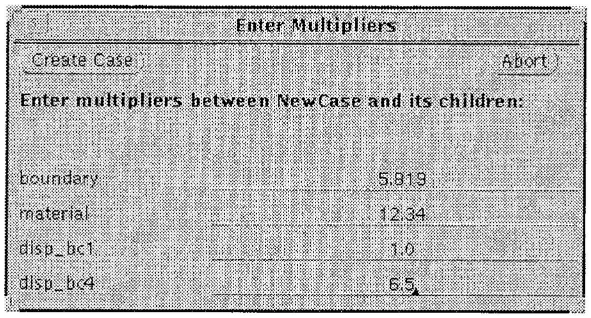
Name: NewCase

Select children from the list below.

| Select Children: | Children: |
|---|---|
| attribute1 [Attribute] | group_a [Group] |
| attribute2 [Attribute] | group_b [Group] |
| attribute3 [Attribute] | group_c [Group] |
| disp_bc1 [Attribute] | group_d [Group] |
| disp_bc2 [Attribute] | group_e [Group] |
| disp_bc3 [Attribute] | boundary [Set] |
| disp_bc4 [Attribute] | loads [Set] |
| elas1 [Attribute] | material [Set] |
| elem_type [Attribute] | material_b [Set] |
| load_a [Attribute] | set1 [Set] |
| load_b [Attribute] | set2 [Set] |
| load_c [Attribute] | set3 [Set] |
| load_d [Attribute] | set4 [Set] |
| load_e [Attribute] | set5 [Set] |
| load_f [Attribute] | set6 [Set] |

Figure 19. Attribute Graphical Interface's Attribute Collector Panel for Editing Children

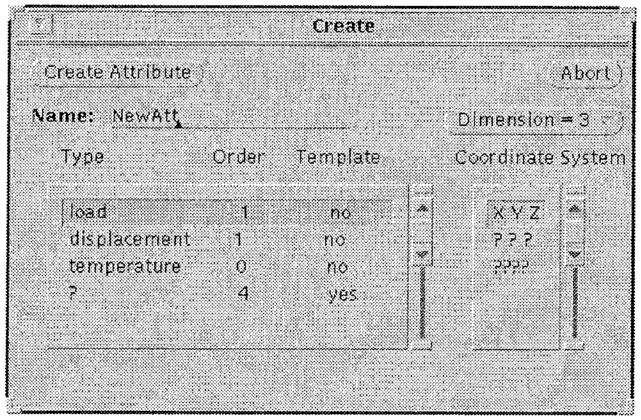
Figures 21 and 22 show similar panels for the creation and modification of attributes. Since most of the analysis information is tensorial in nature, the user is presented with a spreadsheet like interface for entering the distribution function for each element in the tensor. This method can be very tedious when dealing with tensors whose order is greater than 2. For example,



Enter multipliers between NewCase and its children:

| | |
|---|---|
| boundary | 5.819 |
| material | 12.34 |
| disp_bc1 | 1.0 |
| disp_bc4 | 6.5 |

Figure 20. Attribute Graphical Interface's Multiplier Editing Panel

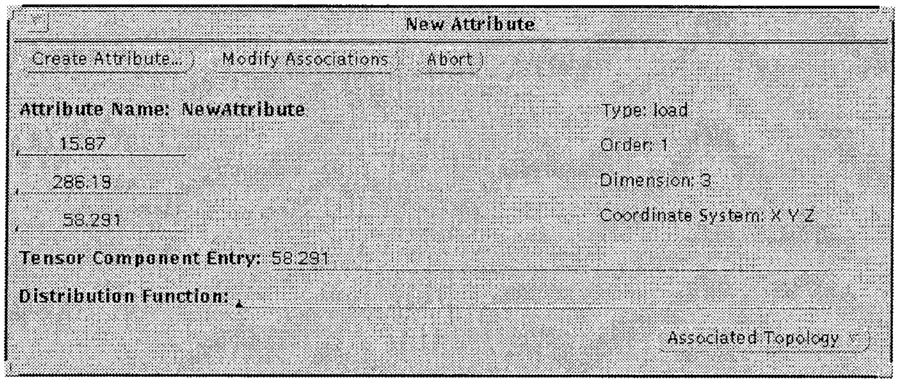


Name: NewAtt

| Type | Order | Template |
|---|---|---|
| load | 1 | no |
| displacement | 1 | no |
| temperature | 0 | no |
| ? | 4 | yes |

| Coordinate System |
|---|
| X Y Z |
| ? ? ? |
| ???? |

Figure 21. Attribute Graphical Interface's New Attribute Panel



Attribute Name: NewAttribute

15.87

286.19

58.291

Type: load

Order: 1

Dimension: 3

Coordinate System: X Y Z

Tensor Component Entry: 58.291

Distribution Function:

Figure 22. Attribute Graphical Interface's Attribute Editing Panel

a compliance tensor is 4th order which results in 81 entries (assuming than the space is 3 dimensional). Fortunately many tensors can be completely defined by a reduced set of parameters due to symmetry. For example, liner isotropic material, which is a 4th order tensor, can be completely defined by specifying the Young's Modulus and Poisson's Ratio. The Attribute Graphical Interface allows designers to add these specialized types of attributes that can result in simplified attribute specification as well as providing semantic checking such as specifying bounds on an entry's value.

In addition to textual and 2D iconic representations of attribute information, the Attribute Graphical Interface, in conjunction with the Model Graphical Interface, can visualize the information in the same space as the geometric model. For example, 3D first order tensors, such as forces, can be visualized using 3D arrow glyphs, while scalar fields such as temperature can be texture mapped onto the surface of the model.

### Integrating the Analysis Process

By having the modeler and attribute abstraction provided by the Attributed Geometric Model available it is more straight forward to provide adaptive procedures access to the geometry based problem definition information required for properly updating the discrete models as they are adaptively enriched. Specifying attributes on a model in the manner described here, as opposed to specifying them on a finite element mesh, allows everything after the specification of the problem to be solved, to obtaining the solution to a prespecified accuracy to be automated. Steps such as creation of an analysis model, where the geometry may be an idealization of the actual geometry, generation of a finite element mesh, running an analysis, and refinement of both the mesh and the model idealizations based on error estimates, can all be integrated together and be done automatically.

## Section 3 Hardware/Software Environments

The previous sections have discussed the design and functionality of the abstraction of the Attributed Geometric Model, as well as the graphical interfaces, that allow users to visually interact with it. Since the graphical interfaces are designed to be interactive, the graphics capabilities of the workstation are extremely important and may force the deactivation of certain functionality. For example, not all workstations are capable of stereoscopic displays. This section discusses the types of hardware platforms that the graphical interfaces are design to run on as well as trade-offs in terms of functionality. In addition, the types of graphics environments that were available for implementing the graphical interfaces are addressed.

### Hardware Environments

The Model Graphical Interface and Attribute Graphical Interface are designed to run on a variety of UNIX platforms independent of their 3D graphics capabilities, which include rendering speed, and special graphics functionalities such as transparency and texture mapping. Though 3D graphics acceleration is preferable it is not a requirement. As a result of using a "2D" system, some of the more advance visual capabilities (such as interactively viewing smooth shaded geometry) may not be available. The reason for supporting the 2D platforms is due to the number of these types of workstations that exist in the engineering environment. As the cost of 3D accelerators continues to drop and PEX-stations (graphics terminals that have hardware to support PEX [19], a 3D extension to the X windowing system) become more available, it is expected that within the next couple years, the typical workstation will have basic 3D support such as built in hidden line/hidden surface support in hardware. The major issue that results from support a range of graphics workstations is maintaining a level of system response time that will not frustrate the user. In order to do this, the interfaces need to determine the type of 3D acceleration that is available and customize themselves in the following manner:

1. Remove functionality that is not supported on that specific platform. For example, stereo viewing requires a head tracking system, stereo glasses, etc... If they do not exist on the workstation, then all of the controls pertaining to it are removed or deactivated.
2. Changing the default behavior. For example, on systems that do not have any 3D accelerator, the geometry is displayed using a wireframe model without hidden line / hidden surfacing enabled. The behavior can also change based on the model complexity. For example, on a 3D system that rendering at 30,000 triangles/sec., the default behavior maybe smooth shaded geometry for model that have < 1,000 planar faces (assuming an update rate of 30 frames/sec) and wireframed geometry for more complicated models.

The breakdown of features on various platforms is as follows:

1. "2D" Systems — no 3D accelerator (such as SPARC/GX)
   a. Ability to enter/view attribute information via Attribute Graphical Interface.
   b. Ability to view interactively wireframe representation of geometry and select topents spatially from the wireframe and text labels.
   c. Allow the user to view static smooth surface representation.
   d. Ability to "overlay" analysis information onto surface geometry.
2. Basic 3D Systems — graphics accelerators which support hidden line/hidden surface and have performance < 80,000 triangles/sec (such as SPARC/GS). The interfaces on these system support all functionality supported in "2D" systems; however, the default is to view smooth shaded geometry. The main restrictions on these systems are:
   a. The lack of special features such as hardware supported transparency.
   b. The limit to the model complexity that can be viewed as smooth shaded geometry (< 4,000 faces).
3. High Performance 3D Systems — graphics accelerators which support advance functionality such as transparency and anti-aliasing [16][13] with a performance <= 600,000 triangles/sec (such as SPARC/ZX and Indigo2 Extreme)
   a. All functionality supported in Basic 3D Systems
   b. Ability to view smooth surface representations of complex geometry (<= 30,000 triangles).
   c. Advanced viewing operations such as making topents transparent in order to see internal detail.

4. Advanced 3D Systems — graphics accelerators which support texture mapping and have performance > 600,000 triangles/sec (such as Onyx Reality Engine2 [24])
   a. All functionality supported in High Performance 3D Systems
   b. Ability to deal with models which are represented by more than 30,000 triangles.
   c. Ability to use more advanced visualization techniques for overlaying analysis information onto the geometry.

## Software Environments

The Model Graphical Interface and Attribute Graphical Interface were designed to be platform independent in terms of the brand of workstation that can be used with the interfaces. This is primarily a software environment issue concerning the graphics libraries that the interfaces use.

**2D Software Environments** The Attribute Graphical Interface is implemented using X11 in order that it will run both distributively and on the most number of platforms; however, a decision had to be made regarding which library to use. The following is a list of libraries that are generally available:

1. XLIB [25] — A set of C routines that directly manage the X11 protocol
2. MOTIF 1.2 Toolkit [26] — A C-based library that implements the MOTIF look & feel via a set of X widgets
3. TCL/TK [27] — a language developed by the University of California that can be integrated into an application by source code modification.
4. Interviews [28] — a C++ toolkit developed by Stanford
5. Fresco [29] — A C++ interface that is included with the current release of X11 (X11R6).
6. NextStep (or OpenStep) [16] — a C++ interface designed by Next Corporation

In addition to the above, the MOTIF 2.0 Toolkit which will include a C++ interface will be available in the fourth quarter of 1994.

The major problem with using XLIB, MOTIF 1.2, or TCL is that these are C or C-like systems that are designed to work well with structure-based designs but not necessarily with designs using object-oriented languages such as C++. Interviews is a relatively old interface that is being used in the design of Fresco and will be probably replaced by the new interface. Interviews also has its own look and feel in terms of user interaction. The problem with Fresco is that it currently lacks certain critical functionality such as menus. NextStep is currently only available on a small number of platforms but will soon be available on several platforms including SUNs (aka OpenStep).

In the long term the software will support the MOTIF look & feel in order to be compliant to the de facto standard as well as using a true C++ library in order to facilitate code development.

Therefore, the Attribute Graphical Interface will be eventually implemented in either MOTIF 2.0, Fresco, or possibly OpenStep; however, in order to have a working prototype, the initial Attribute Graphical Interface is implemented in an interface library that was developed at RPI called the Modular Interface Library Kit (MILK). MILK is a C++ interface toolkit built on top of XView, which currently supports the OPENLOOK [30] look and feel, and currently runs on SUN, IBM, and SGI platforms.

**3D Software Environments** In terms of available 3D graphics libraries that are currently available, the list includes:

1. PHIGS[17] — ANSI/ISO Standard that includes a C interface
2. PEXLIB[19] — A set of C routines that directly manage the PEX protocol (3D extensions to X)
3. GL[18] — a proprietary C library developed by SGI
4. Inventor[31] — a proprietary C++ library developed by SGI
5. XGL[32] — a proprietary C library developed by SUN.

The problem with using C-based routines for the 3D development are the same as those for the 2D. One of the major requirements for the library is that it be dynamically extensible and allow a designer to be able to add new primitives and new functionality. For example, the Model Graphical Interface requires graphical primitives that can be associated with a topent's geometry that is produced via the Geometry member function call. The only commercial C++ library that is currently available is Inventor which currently only runs on SGI.

The researchers at RPI have developed a C++ library called BAGEL which is written on top of GL and XGL and runs on SUN, SGI, and IBM platforms. The current prototype of the Model Graphical Interface is implemented using the BAGEL library. It should be possible to port the library's device driver to the PEXlib platform and thus be able to develop on any PEX-based machine such as the HP. In addition, the library should be ported to the new OPENGL library developed by SGI.

## Section 4 Closing Remarks

A modeler independent abstraction that encompasses the necessary functionality for querying and manipulating a geometric model has been developed. Using this abstraction, and an abstraction for the specification of analysis attributes, a system has been developed that allows programmers to access model and attribute information in a consistent and intuitive manner. The initial implementation of the system has been done with the commercial modeler Shapes from the XOX Corporation. In addition, graphical user interfaces have been implemented to allow visualization of model and specification of analysis attribute information. The same system was used to develop an interactive visual tool to view finite element meshes for the purpose of debugging automatic mesh generators. The graphical interfaces were designed to be platform independent and to be usable on workstations with a large range of graphics performance.

Further work is being done to integrate other modelers into the system. This process is very straightforward due to the object-oriented abstraction selected for the model.

## Section 5 Acknowledgments

The authors gratefully acknowledge the support of ARPA/ONR under grant N00014-92-J-1779, "Mechanism-Based Design of Composite Materials", and the National Science Foundation under grant ASC-93-18184, "Understanding Human Joint Mechanics through Advanced Computational Model".

## Section 6 References

[1] XOX Corporation, Two Appletree Square, Suite 334, Minneapolis, Minnesota 55425. *SHAPES Reference Manual, Release 2.0.5*, July 20, 1993.

[2] Shape Data Limited, Parker's House, 46 Regent Street, Cambridge CB2 1DB England. *PARASOLID v4.0 Programming Reference Manual*, August 1991.

[3] Spatial Technology, Inc., 2425 25th St., Building A, Boulder, Colorado. *ACIS Interface Guide and ACIS API Guide*, December 1992.

[4] Electronic Data Systems Corp., Unigraphics Division, 13736 Riverport Drive, Maryland Heights, MO 63043. *Unigraphics User Manual, Version 10.2*, December 1993.

[5] Dassault Systemes, IBM Corporation, 11601 Wilshire Boulevard, LA, CA. *CATIA Solids Geometry - Geometry Interface Reference Manual; CATIA Solids Geometry - Mathematical Subroutine Package Reference Manual; and CATIA Base - Mathematical Subroutine Package Reference Manual*, 1988. Pub. Nums. SH50-0059-0; SH50-0058-0; SH50-0089-0.

[6] International Standards Organization. Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles. Technical Report ISO/DIS 10303-1, U.S. Product Data Association, 1993.

[7] M.S. Shephard and P.M. Finnigan. Toward automatic model generation. In A.K. Noor and J.T. Oden, editors, *State-of-the-Art Surveys on Computational Mechanics*, pages 335–366. ASME, 1989.

[8] M.S. Shephard. The specification of physical attribute information for engineering analysis. *Engineering with Computers*, 4:145–155, 1988.

[9] V. Wong. Qualification and management of analysis attriubtes with application to multi-procedural analyses for multichip modules. Masters thesis, Scientific Computation Research Center, Report 23-1994, Rensselaer Polytechnic Institute, Troy, New York, June 1994.

[10] K. J. Weiler. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In M. J. Wozny, H. W. McLaughlin, and J. L. Encarnacao, editors, *Geometricfor CAD Applications*, pages 3–36. North Holland, 1988.

[11] K. J. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Design Research Center, Rensselaer Polytechnic Institute, Troy, NY, May 1986.

[12] H. Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, 6:623–629, 1971.

[13] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques, Theory and Practice*. Addison Wesley, 1992.

[14] L.F. Hodges. Tutorial: Time-multiplexed stereoscopic computer graphics. *IEEE Computer Graphics and Applications*, 12(2):20–30, 1992.

[15] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide*. Academic Press, third edition, 1993.

[16] J.D. Foley, A. Van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics - Principles and Practices*. The Systems Programming Series. Addison-Wesley, second edition, 1990.

[17] PHIGS+ Committee. PHIGS+ Functional Description, Revision 3.0. *Computer Graphics*, 22(3):125–218, 1988.

[18] Silicon Graphics Computer Systems. *Graphics Library Programming Guide*.

[19] T. Gaskins. *PEXlib Programming Manual*. O'Reilly and Associates Inc., 1992.

[20] M.F. Deering and S.R. Nelson. Leo: A system for cost effective 3d shaded graphics. In *Proceedings of SIGGRAPH 93*, pages 101–108. Addison-Wesley, 1993.

[21] J. Hultquist. A virtual trackball. In A.S. Glassner, editor, *Computer Graphics Gems*, pages 462–463. 1993.

[22] Logitech Inc. *2D/6D Mouse Technical Manual*.

[23] M. Gleicher and Witkin A. Through-the-lens camera control. In *Proceedings of SIGGRAPH 92*, pages 331–340. Addison-Wesley, 1992.

[24] K. Akeley. Reality engine graphics. In *Proceedings of SIGGRAPH 93*, pages 109–142. Addison-Wesley, 1993.

[25] A. Nye. *Xlib Programming Manual*, volume 1. O'Reilly & Associates, Inc., second edition, 1990.

[26] D. Young. *X Window Systems Programming and Applications With Xt OSF/MOTIF Edition*. Prentice Hall, 1990.

[27] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[28] *InterViews Reference Manual*. Leland Stanford Junior University, 1991.

[29] X Consortium Working Group Draft. *Fresco Sample Implementation Reference Manual*, 0.7 edition.

[30] Sun Microsystems, Inc. *OPEN LOOK - Graphical User Interface Application Style Guidelines*. Addison-Wesley, 1990.

[31] J. Wernecke. *The Inventor Mentor*. Addison-Wesley, 1994.

[32] Sun Microsystems, Inc. XGL Graphics Library - Technical White Paper, 1990.