GEOMETRIC MODELING FOR CO-MODELING AND CO-SIMULATION IN COLLABORATIVE PRODUCT DEVELOPMENT

RUAN LIWEI

NATIONAL UNIVERSITY OF SINGAPORE

2004

GEOMETRIC MODELING FOR CO-MODELING AND CO-SIMULATION IN COLLABORATIVE PRODUCT DEVELOPMENT

Ruan, Liwei
(B. Eng. (Hons). Zhejiang University)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

ELECTRICAL & COMPUTER ENGINEERING DEPARTMENT

NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgement

First of all, I would like to express my appreciation to my supervisor, Dr. Zhejie Liu from the Data Storage Institute (Singapore), for his supervision and guidance, strong support and great patience during the period of my candidature. No endeavor is accomplished alone. And I would like to thank all the other members of EDM group of Data Storage Institute for their help and assistance.

I would also like to thank my parents, Ruan Yongxian & Wang Feizhi and my sister, Ruan Lishun, for giving me all kinds of supports in my life.

Summary

Due to the globalization of manufacturing and product development activities, rapid product development in distributed engineering design environments has become more prevalent. Such practices can greatly shorten the product development cycle and lower costs. Product design is an activity where collaboration is of vital importance. Therefore, Collaborative Design and Engineering has become a major issue in manufacturing industry where people from different fields of expertise and research backgrounds strongly need to co-operate. State-of-the-art computer technologies have enabled the product design process to be undertaken across distributed teams in different locations. Given this development, Distributed and Collaborative Design and Engineering Systems have been invented to support geographically separated users to conduct design activities synchronously, asynchronously and cooperatively. Meanwhile, it is a research challenge to seamlessly integrate robust three-dimensional solid geometric modeling functionalities into the distributed and collaborative design software systems to facilitate Computer Aided Design (CAD) process and prepare geometric data for numerical solutions in Computer Aided Engineering (CAE).

In this thesis, the architecture of a distributed and collaborative design and engineering system, CoCADE, is proposed to address a growing industrial concern as evidenced from the thorough survey of current geometric modeling and distributed computing technologies. Based on .NET technology, the system takes advantages of object-oriented method throughout its software development lifecycle. This architecture adopts a three-tiered client/server model. The client side provides the essential geometric modeling functionalities as well as interactive visualization tools for display or

modification of sophisticated geometric models and large-scale simulation results. The server side offers the functions of collaborative session management, multi-client communication mechanism, engineering simulation and optimization.

It is demonstrated that the CoCADE system presented in this thesis can effectively support advanced product development by integrating robust geometric modeling functions into the distributed and collaborative environment. The outcome of this research indicates that applications built on CoCADE can effectively support the distributed product design and innovation, engineering numerical solutions and engineering visualization.

Keywords: geometric modeling, distributed computing, CAD/CAE, collaborative design, scientific data presentation.

Table of Contents

List of Figures

1. Introduction

1.1 Background

1.1.1 Introduction to Geometric and Solid Modeling

Traditional geometric modeling identifies a body of techniques that can model certain classes of shape of piecewise parametric surfaces, which is followed by its development as a separate field in several industries, including automobile and aerospace [1]. Today, most of its applications stem from Computer-Aided Design (CAD)/Computer-Aided Manufacturing (CAM), robotics, computer graphics and so on [11]. Hoffmann [1] provided a view that the streams of geometric and solid modeling are converging because the former modeling contemplates building complete solid representations from surface patches and the latter strives to extend the geometric coverage. Solid modeling began in the late 1960s and 1970s. But it only gains more and more popularity and is adapted as primary technique to define 3-dimensional (3D) geometry by major commercial CAD/CAE/CAM software vendors since the 1990s.

Solid modeling is a method by which various solid objects are combined into a single 3D part design. The real advantage of a solid modeling application is how it can create the desired resulting part designs by joining, intersecting and subtracting the solid objects from one another. Recent solid modelers are based on solid objects being created by sweeping, rotating and extruding 2-dimensional wire-frame or sketch geometry. Prior to this, these solid objects were formed by a variety of primitive shapes such as a block,

sphere, cone, cylinder, etc. Today more and more solid modelers have abandoned the primitive shapes in favor of the predefined solid object library.

There are various ways of defining the shape of an object in digital form to a computer such as wire frame, boundary representation, set-theoretic, polyhedral model, and so on. Each of the representation methods has its own specific strength over the others in terms of applicability and elegance. However, in the commercial world of exact geometric modelers, virtually all CAD/CAE/CAM software products have used boundary representation (B-Rep) approach exclusively for solid modeling. In B-Rep, every solid model is defined and stored by its boundary, which consists of vertices, edges, faces and explicit boundary, and includes the topology for each face.

The two predominant commercial solid modeling engines in the market are the Parasolid® and the ACIS®. The standard file formats are .x_t and .x_b for Parasolid files and .SAT for ACIS files.

1.1.2 Introduction to Distributed and Collaborative Design

Computer Aided Engineering Design is an enabling technology that leverages on the powerful computer systems to support product design process and to shorten the product development cycle. Over the years, substantial progress has been made in the development of major computer aided engineering design systems. These systems typically include CAD, CAE, and design optimization modules. Figure 1.1 outlines the relationship between them.

CAD was first introduced by Ivan Sutherland's Sketchpad at MIT and the DAC-1 project at General Motors in the early 1960s. Different industries developed their own CAD applications, delivered on multi-user mainframes, in the 1960s and 1970s. Unlike

the bundled hardware and software systems in the 1980s, hardware and software components are separated in current CAD implementations. As a result, CAD software most often executes locally on powerful UNIX or Windows-Intel workstations in a distributed environment [2].

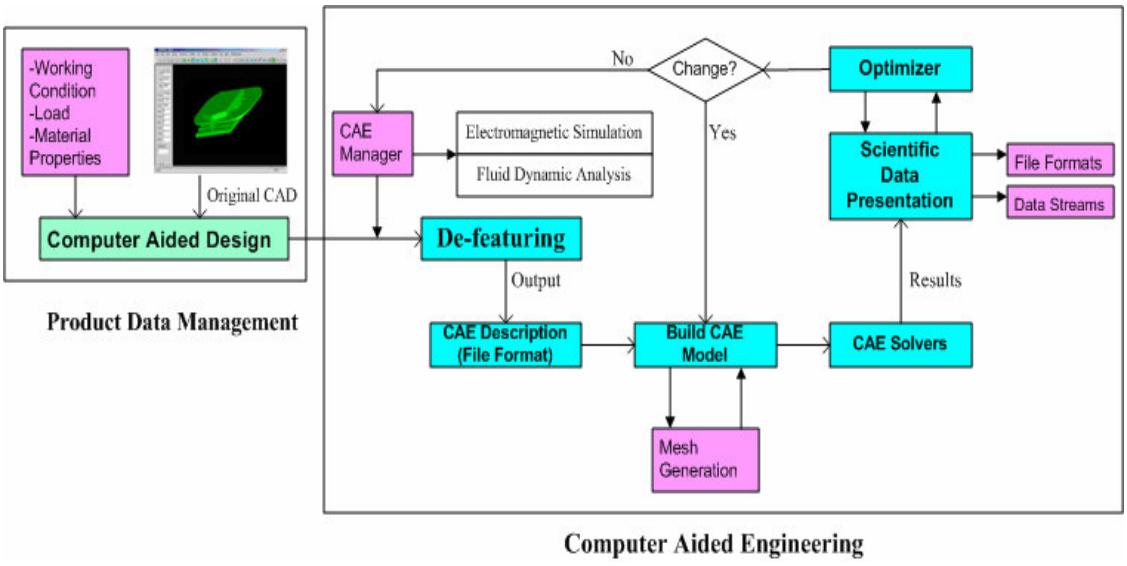


Figure 1-1 CAD/CAE Process Outline

Since 1990s, solid modeler based CAD/CAE/CAM software has been widely used for product design, simulation, analysis and manufacturing. The solid modeling technology facilitates product design engineers with a variety of modeling functions like creation of solid primitive shapes, performing of Boolean operations on solid, sheet or wire models, etc. Apart from the above modeling capabilities, solid modeling provides product manufacturers a much more complete and accurate component description than the traditional blueprint or IGES file. Recent advancement in solid modeling systems

provides the capability of input and output of solid models in different file formats, thus facilitating data exchange with major computer aided design and manufacturing systems.

CAE generally relies on 3D geometric models, which defines the shapes of real-world objects, description of component attributes and physical conditions, and uses advanced numerical techniques such as finite element method (FEM), finite difference method (FDM) or boundary element method (BEM). Over the years, CAE software developers have implemented numerous systems in which engineers are able to design, analyze, simulate and visualize the performance of even the most complicated products. A variety of CAE software such as ANSYS®, ADAMS® and ANSOFT® have enabled manufacturing companies to greatly shorten the product design and development cycle, improve product quality and lower costs.

One of the typical characters in CAE is the extremely large amount of result data generated during geometric modeling and product analysis stages. How to efficiently handle the result data with the currently limited network bandwidth and computer performance has become of imperative concern for CAE solution researchers. In this situation, data streaming is one of the efficient techniques used to distribute visualization task onto heterogeneous computer systems and thus to reduce network load.

In conclusion, major manufacturing companies in fields are increasingly investing in distributed CAD, CAE or CAM systems so as to remain competitive in the global market.

1.2 Main Issues in Distributed Systems

1.2.1 The Client-Server Model

The client-server model is an approach to organize distributed systems. In this model, a client requests a service from a server by sending it a request while a server implements a specific service. Communication between a client and a server can be implemented by a simple connectionless or reliable connection-oriented protocol. This client-server communication is shown in Figure 1.2.

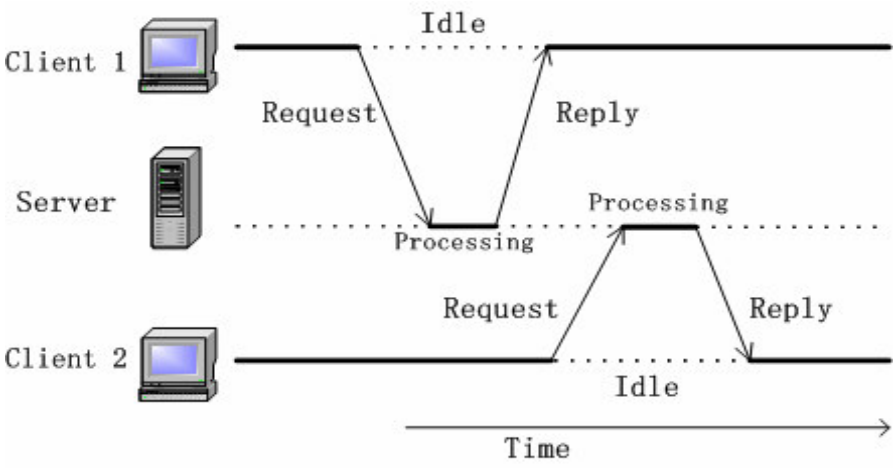


Figure 1-2 Communication between the Clients and a Server

Together, the two groups form a complete computing system with a distinction of responsibilities. The client is responsible for directly interfacing with the user, such as modeling and display management. The server typically contains the core applications to provide client-server communication, coordinate different design sessions and assure data consistency. From the perspective of physically distributing client-server application across several computers, there are two types of computers: a client computer containing

the programs that implement user-interface and a server computer containing the programs that implement the processing applications and database. However, modern multi-tiered client-server architectures often distribute the programs in the processing applications across different computers, leading to what is referred to as physically *two-tiered architecture*. A server may sometimes need to act as a client, leading to physically *three-tiered architecture*. Finally, when the clients are distributed as well, we may have the case where there is no server at all. In such a case, it is often referred to as a *peer-to-peer architecture*.

When applying the distributed model and technology in the CAD/CAE systems, we can have three major types of distributed CAD/CAE systems that are of particular interest to researchers.

1. Two-tiered distributed CAD/CAE systems

In a typical two-tiered distributed CAE/CAE system, we often make a distinction between only two kinds of machines: clients and servers. The programs in the application logic can be distributed in the client or the server, or across both machines. One possible arrangement is to have only the data presentation on the client machine and the applications in the server having remote control over the presentation of data, as shown in Figure 1.3. In this scenario, Remote Procedure Call (RPC) provides an effective mechanism for the client to communicate with the applications residing at the server.

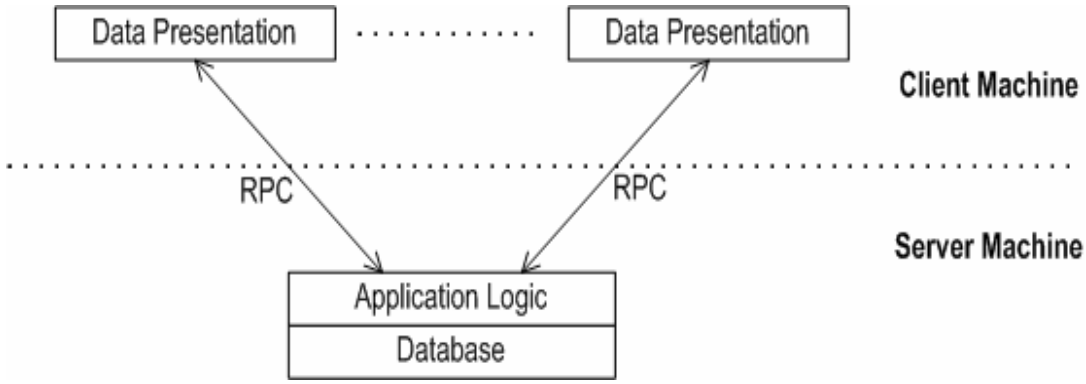


Figure 1-3 Typical Two-tiered Client-Server Architecture

During a RPC process, the client stub usually makes a call by bundling the parameters into a message and requesting that the message be sent to the server. The server stub in the server machine unpacks the parameters from the message and pushes them into a local stack. Then it calls the server procedures to perform its work and obtain the results. Finally, the retuned results are sent back to the caller. To define the interfaces in the RPC procedures, a special interface definition language (IDL) is adopted to map these interfaces to the programming language used in developing the applications.

Continuing along the above client-server organization, we may also move the application logic to run on the client machines. This arrangement is particular popular in the case where the client machine is a workstation, connected through a network to a distributed file system or database. The server is in charge of all the operation on files or database entries while the client's local disk may contain part of the data. For example, a client browsing the Web can accumulate a huge data cache of most recent accessed Web pages [2].

2. Three-tiered distributed CAD/CAE systems

In the previous section on two-tiered client-server architecture, we distinguish between only clients and servers. However, the server may sometimes play a role as a client, as shown in Figure 1.4, leading to a three-tiered architecture. A typical example of where this model is used is in transaction processing. In this case, a separate transaction monitoring process coordinates all transactions across possibly different data servers [2]. The application logic is separated from the data presentation layer and database layer. The programs in the application logic layer usually reside on a separate server but they may additionally be partly distributed across the client and server machines, making this model more flexible than the two-tiered one. Furthermore, decoupling the application logic from the data allows data from multiple sources to be used in a single transaction. Major features of the three-tiered architecture include multiple operating systems, one or more programming languages, local and remote databases, inter-program communications and message routing etc.

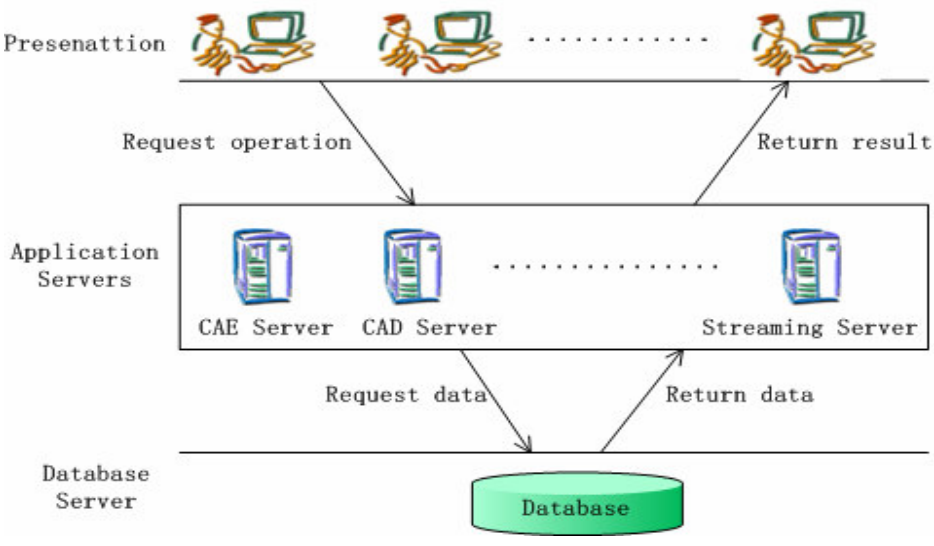


Figure 1-4 Three-tiered Client-Server Architecture

3. Peer-to-peer distributed CAD/CAE systems

Although less apparent, clients can be distributed as well, as shown in Figure 1.5. For simple collaborative applications, we may even have the case where there is no server at all. For example, a client seeks contact with another client, after which both clients launch the same application for starting a session. A third client may contact either one of the two, and subsequently launch the same application software [2].

The recent wide spread use of peer-to-peer applications such as SETI, Napster, and Gnutella indicates that there are many potential benefits to fully distributed peer-to-peer systems. The characteristic of peer-to-peer distributed system is that every client in the network is an autonomous system. And two applications can communicate via sockets to collaborate over network, which usually adopts TCP socket programming paradigm. One of the shortcomings of TCP socket programming is that programmers must describe the complicated socket connection process, after which a procedure can be called to set up connection, listen for and accept connections, bind to available port number, and so on.

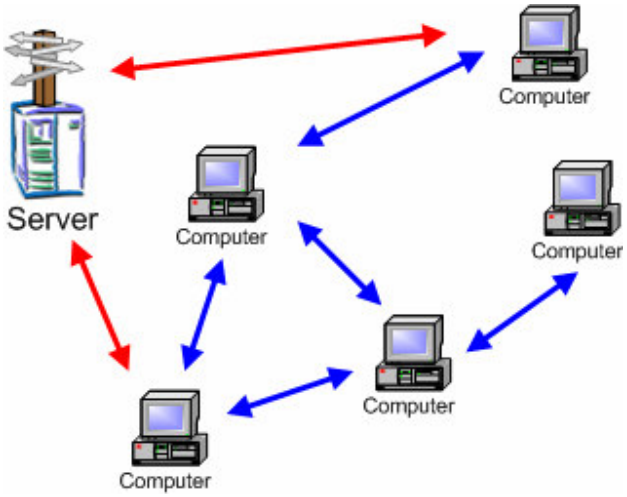


Figure 1-5 Peer-to-peer Distributed Architecture

1.2.2 Distributed Objects and .NET Remoting Technology

1. Distributed Objects

A key characteristic of an object is its separation between data (often called the "state") and operations (often called the "methods") on those data. The process can manipulate the state of an object by invoking an object's methods, which is made possible through an object's interface. When the notion of an object is applied in distributed computing, distributed objects are formed to facilitate clients' access of services and resources. The separation between an interface and its object allows us to deploy an interface at one machine while its object is placed at another machine, leading to what is referred to as a remote object. The state itself can be physically distributed across multiple machines as well [2]. In such an environment, distribution transparency is achieved by remote access of objects because the distributed objects can be virtually anywhere on the network. When a client requests access to a remote object's methods, a proxy (similar to the stub in RPC) implementing the interface is loaded into the client's address space first to marshal the request into messages. The skeleton (similar to the server stub) un-marshals the incoming messages to invoke the requested methods in the object at the server and passes the reply message to the client's proxy.

As a first example, we take a look at one of the major distributed object paradigms – CORBA. CORBA is the acronym of Common Object Request Broker Architecture. This distributed object-based system and its specifications have been designed by a nonprofit organization called Object Management Group (OMG). One of the major goals of this system is to provide a mechanism so as to solve many of the interoperability problems between various applications resided across the network.

CORBA uses the remote-object model that we discussed previously. CORBA is a typical example illustrating that making a distributed systems is a somewhat difficult task.

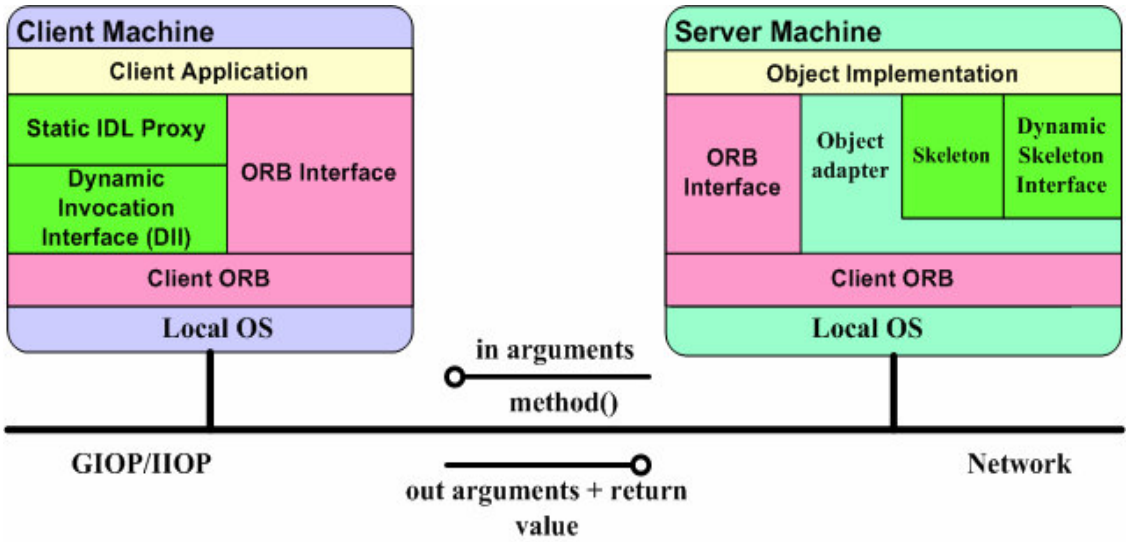


Figure 1-6 General Architecture of CORBA [2]

As shown in Figure 1.6 [2], the Object Request Broker (ORB) is the foundation of any CORBA distributed systems. It acts as a low-level object bus that is responsible for the communication between remote server objects and their distributed clients, ensuring that an object invocation is sent to server and its reply returned to the client. Internet Inter-ORB Protocol (IIOP) is the implementation of General Inter-ORB Protocol (GIOP) on top of TCP, which enables interoperability between clients and object servers from different CORBA system manufacturers. Interface Definition Language (IDL) specifies the objects and their services. To request a service, a client sends an invocation request message comprising an object reference, the method's name and all input parameters to CORBA object server. A CORBA server object interacts with ORB either through the ORB interface or Object Adapter. When the ORB in the object server receives the

request, it looks for the implementation of the requested method and passes the parameters to it. Finally, a reply message containing the returned values and output parameters is sent back to the client.

Another paradigm of distributed object-based system is the Distributed Common Object Model (DCOM) growing out from Microsoft's Common Object Model (COM), which is the core technology in Windows operating systems.

It is sometimes hard to make a distinction between COM, COM+ and DCOM. COM+ can be viewed as a superset of COM consisting of various services that were previously offered as add-ons to COM. In particular, COM+ includes facilities for a server that can efficiently handle a large number of objects. DCOM can be viewed as an extension to the architecture of COM. However, the features supporting a process to communicate with components placed on another machine are often the same for both COM and DCOM. DCOM also adopts the remote-object model as most of other object-based systems. Both synchronous and asynchronous communications are supported in DCOM, which means that a client invoking an object may not necessarily be blocked until a reply is received. In contrast to CORBA's language-specific interfaces, each server object in DCOM supports and implements multiple interfaces in the language-independent binary form, which allows these server components to be written in a variety of programming language like Java, C++, Visual Basic, and Delphi. Meanwhile, Microsoft Interface Definition Language (MSIDL) can generate these interfaces each implementing different functions of the object.

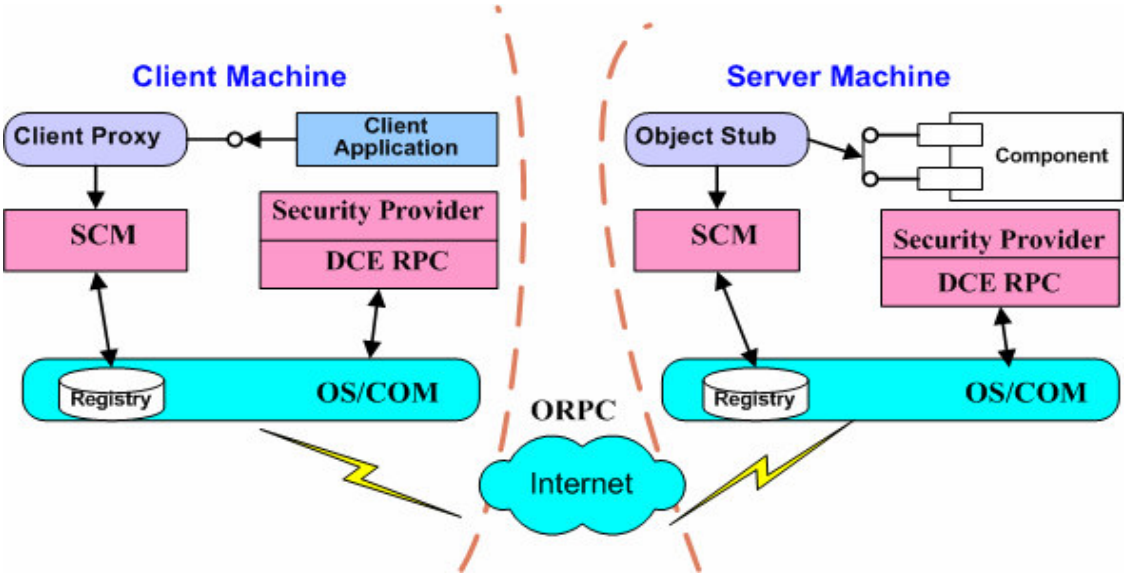


Figure 1-7 General Organization of DCOM

As shown in Figure 1.7, the communication in DCOM is supported through the mechanism of Object Remote Procedure Call (ORPC). On the client side, a process sets up a binding to a remote object by accessing the Service Control Manager (SCM), which is responsible for activating objects, and the registry. The client proxy is responsible for marshaling (or un-marshaling) the invocation and sending a request to (or receiving a reply from) the remote object. The client object can invoke the exposed methods of the server object by the interface pointer as if the object is resided in local address space. On the server side, the object stub is responsible for processing the invocation from the client. To instantiate a new object with a Class Identifier (CLSID) for a client, the SCM receives this CLSID and looks it up in its local registry to find the relevant file so as to load the class object. To facilitate this process in the server, DCOM provides just-in-time (JIT) activation to efficiently activate and destroy objects.

2. NET Remoting Technology

Coming with many powerful yet simple features, the .NET Remoting is a new framework developed by Microsoft to replace DCOM discussed in the previous section. This rich and extensible framework enables objects residing in different application domains, in different processes or in different machines to interact with each other seamlessly, which is made possible and supported through its runtime and programming model.

The .NET Remoting framework supports the functions such as object activation, lifetime management, distributed identities as well as communication channels. There are two main object activation models, client-activated objects model and server-activated objects model, to manage the lifetime of remote objects under the support of .NET framework. In the case of a client-activated objects model, a lease-based lifetime manager controls these objects to ensure that the objects are garbage collected when its lease expires. In the case of a server-activated objects model, the programmer can choose "single call" or "singleton" to manage the lifetime of the objects.

Figure 1.8 shows an example of a client calling a remote object where the called methods are implemented. In order to invoke the methods of the remote object running inside a different machine or process, the client creates an object proxy called TransparentProxy. For the client, this Object Proxy looks like a local object with the public methods. When the methods of the remote object are called, an IMessage with the call parameters will be created and forwarded to a RealProxy class. The message is serialized into a stream of bytes using a Serialization Formatter and then is sent into a Client Channel that is responsible for communicating with the server.

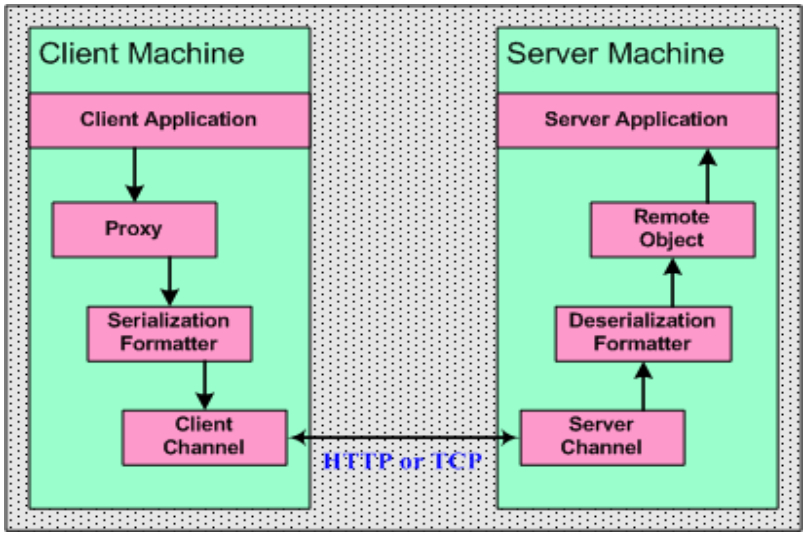


Figure 1-8 An Example of Calling a Remote Object in .NET Remoting Framework

The client channel communicates with the server channel to transfer the message across the network. After receiving the message, the sever channel passes it to the De-serialization Formatter for de-serialization so that the called methods can be sent to the remote object. The remote object must be implemented in a class derived from System.MarshalByRefObject that has methods to manage lifetime service, and it is only confined to the application domain where it is created.

The two main channels used to transport messages to and from remote objects are HTTP channel and TCP channel. HTTP uses the Simple Object Access Protocol (SOAP) to communicate with remote objects. In this case, all messages are changed into XML and serialized by SOAP formatter first and then a required SOAP is added to the message stream. Finally, the stream is transferred to the target server by HTTP protocol. Similary in the situation of transporting a message using TCP channel, a binary formatter is used

to serialize all messages to a binary stream and transferred to target server by TCP protocol.

In conclusion, the .NET framework provides an almost perfect paradigm that meets the requirements of most distributed applications. It provides a powerful, extensible and language-independent framework to develop robust and scalable distributed systems. This is also why .NET remoting is chosen in this research to support and implement the distributed functionalities in CoCADE.

1.2.3 Data Streaming for Distributed Collaborative CAE

In most cases, the raw engineering design data from large 3D solid models and engineering simulation results are relatively large. Traditionally, the data could not be displayed before the entire data sets are received. In a distributed design environment, especially in a real-time collaborative design process, it creates a bottleneck for efficiently and effectively sharing geometric information of large 3D solid models.

Streaming technology is one of the key technologies for efficient transmission and visualization of high volume data. Although this technology has already being applied to audio and video transmission, it is still a research challenge to develop a strategy for the effective transmission and visualization of engineering data. Engel et al. [4] introduced a web-based visualization approach to enable the remote control of a visualization application by image streaming and CORBA (Common Object Request Broker Architecture) techniques. This approach is only applicable in some basic visualization functions. It shows limitations, e.g. level of user interactivity, multi-user capabilities, and rendering synchronization. The use of CORBA technique also exposes security problem

because it requires the configuration of network firewall. Mahovsky et al. [5] have developed a Java-based system for real-time distributed visualization by using a cluster of conventional PCs. This system adopts an extensible and object-oriented model that accommodates a variety of rendering algorithms and data sets. However, collaborative visualization function for multiple users to operate on a shared data set is not supported in this system. The system is not feasible to be deployed on the Internet so as to provide remote real-time visualization facilities.

In this thesis, the CoCADE framework is a distributed system that supports real-time collaborative visualization of complex engineering geometric models and simulation results. Multiple users can conduct engineering design activities such as real-time co-modeling, simultaneous 3D view and mark-up, text-chatting etc. Data streaming technology is utilized in the scenario where data sets are to be re-ordered according to the clients' perspective before transmitted. In addition, simplified mesh approximation of the geometry of each individual part may also need to be created and downloaded in a hierarchical order, therefore, each part can be delivered incrementally. An open standard stream-able format for exchanging of visualization data between disparate applications, HOOPS Stream File, is employed in this system and will be extended to support varied custom types of engineering data.

1.3 Main Modules in CAE Systems

The CAD/CAE process of a product usually involves intensive interactions between different groups of designers. Current existing computer-based systems have been greatly enhanced to support such interaction by facilitating the iteration process between

computer aided design and engineering analysis and vice versa. The major design activity iteration process is shown in Figure 1.9.

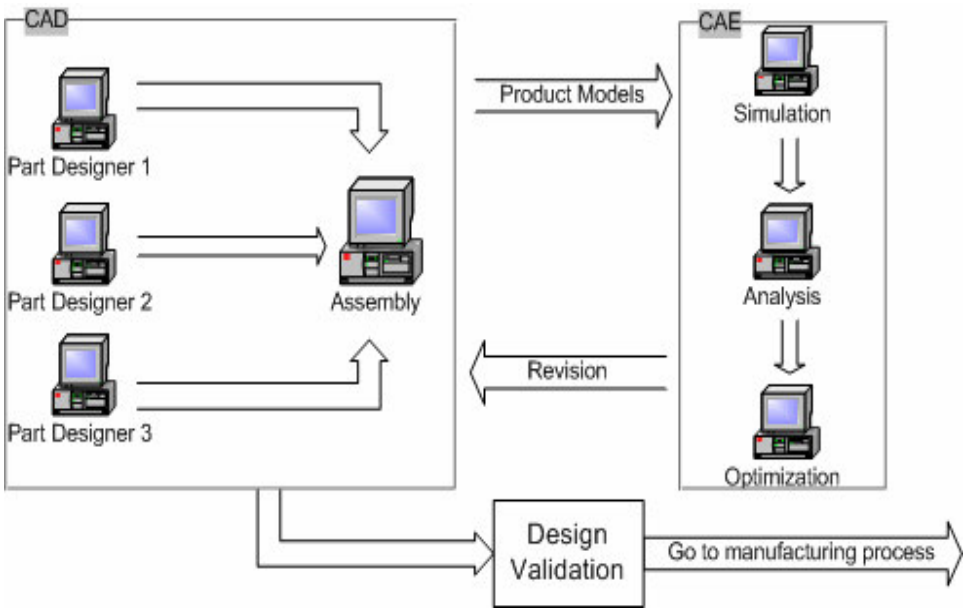


Figure 1-9 Product Design Process

As discussed in the above, the model design and engineering analysis of a product is an iteration process involving different modules of CAD and CAE systems. First of all, the part designers use the computer aided design systems to construct the geometric design models. After being assigned physical attributes, these part models are assembled to form a complete product design model in order to test its performance in CAE stage (Figure 1.9). Subsequently, the product design models are sent to CAE engineers for realistically simulating the functioning of the design models in real life. The CAE engineers analyze, simulate and visualize the performance of the design models. Based on the simulation feedbacks, some optimization approaches may be applied to achieve better product performance in a holistic way. If the simulation results are satisfactory, the

validated product design models will be passed for manufacturing, otherwise revisions will be communicated to the CAD engineers to modify the design models.

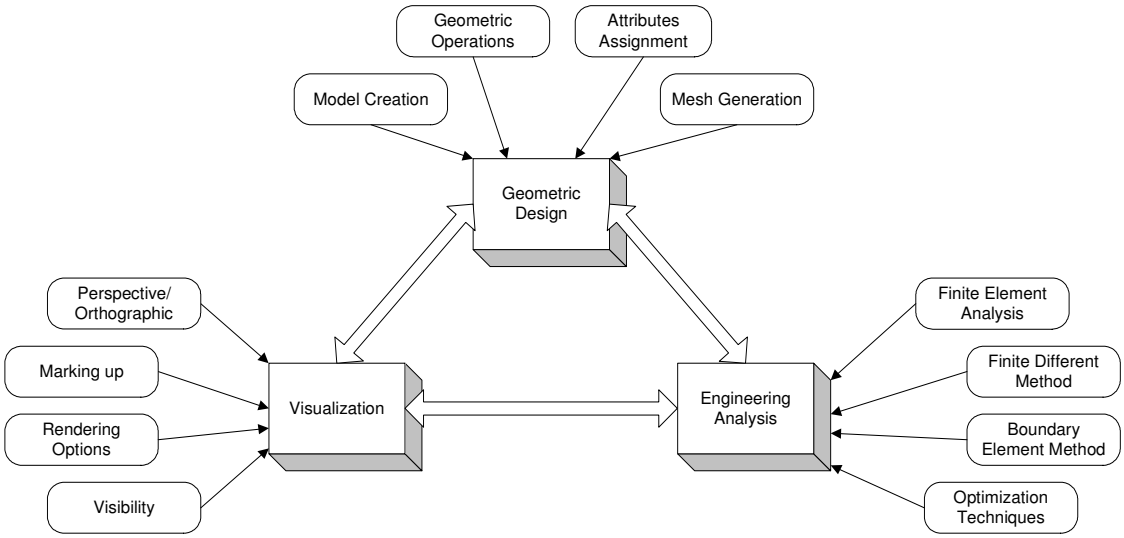


Figure 1-10 Different Modules in CAD/CAE Systems

As illustrated in Figure 1.10, there are three major modules, namely geometric design module, engineering analysis module and visualization module, in a typical CAD or CAE system.

1.3.1 Geometric Design Module

The geometric design module usually provides the designers with a graphical user-friendly interface where they can have easy and quick access to the geometric data. It means the user interface should not be only suitable for seasoned designers, but also for inexperienced designers who are beginners.

The common attributes in the geometric design module might include 1) capabilities to allow more and more geometric operations to be performed on the design

model, 2) support for data filtering of huge amount of information, 3) view, print and interrogate capabilities for design models in a variety of file formats, 4) portability across major operating systems such as UNIX and Windows [3]. To meet these requirements, the classes used to represent a geometric object in typical CAE applications usually comprise the geometric data and topology information. These data information must be able to be saved to and restored from a save file such as ACIS' SAT file. The geometric data generated by this module can be saved as a variety of file formats such as SAT, IGES, HSF, VRML, etc.

One of the important steps in geometric design process is mesh generation so as to prepare the geometric model for engineering simulation and analysis. Mesh generation is a process that breaks up a physical domain into smaller sub-domains (elements) in order to facilitate the numerical solution of a partial differential equation [6]. The surface or 2D domain of the geometric model can be subdivided into triangular, quadrilateral or hexagonal elements while the 3D volume can be subdivided into tetrahedral or hexahedra elements. In recent decades, finite element methods are increasingly used as principle numerical solutions to simulate industrial engineering designs and physical processes. The accuracy of finite elements method depends heavily on the number of the elements. Generally, the finer the elements, the better the simulation. However, one should realize that the computation time for numerical solution also increases dramatically when the number of elements increases. So one should be acutely aware of the balance between the number of elements and computation time.

1.3.2 Engineering Analysis Module

The engineering analysis module has been improved substantially in recent years to incorporate various engineering tools for more efficient and realistic simulation of the product performance. Appropriate computational means such as finite element methods (FEM) and boundary element method (BEM) are used to facilitate the simulation process. FEM required that the input data be 2D or 3D mesh comprising numerous structured or unstructured mesh elements such as triangle, tetrahedron, hexahedra, quadrilateral, etc.

1.3.3 Visualization Module

Advances in the development of CAE visualization module provide designers with visualization interface to view in high resolution the large amount of result data from different viewing modes such as the orthographical and perspective views. This module is usually coupled to the raw data source. In common commercial post process applications, markup tools and rendering modes are available for designers to manipulate simulation results. Designers will further post process the designed model if the performance is not satisfactory according to industrial standards.

1.4 Voronoi Diagrams or Tessellations

The Voronoi Diagrams or tessellation is a method to construct randomly sized and shaped subdivisions or "cells" (polygons) of space [7]. Its application in this thesis is to emulate the realistic physical microstructure of perpendicular recording media. Voronoi Diagrams are part of an encompassing field of Computational Geometry, which emerged from the field of algorithms design and analysis in the late 1970s and has grown into a

successful research discipline. The Voronoi Diagrams and its graph theory dual, Delaunay Tessellation or Triangulation, and the numerical solution of partial differential equations have recently become the major applications of computational geometry.

With the advent of more powerful and efficient computer technology (hardware and algorithms), Voronoi Diagrams have been widely used in various research fields, particularly in the area of micro-magnetic modeling and simulation of magnetic recording media. How to build a mathematical model to accurately describe the physical characteristics of magnetic recording media is very essential in micro-magnetic media simulation.

In recent years, much attention [8][9][10] has been attracted to the study of microstructure of the magnetic recording media, which composes of grains that vary in both shape and size. However, these approaches are mostly based on uniform hexagonal grains shapes. Therefore, the effect of irregular grains shapes, which result from variations in the media fabrication process, is not taken into account. In this model, irregular grain shapes are created to emulate the realistic physical microstructure of perpendicular recording media.

1.4.1 Brief History and Applications

As a great antiquity, the formal Voronoi diagram method can be historically traced back and credited to mathematicians around the turn of the century, G. L. Dirichlet (1850) and G. F. Voronoi (1908). Descartes in his *The Principles of Philosophy (1644)* used Voronoi-like diagrams to show a solar system consisting of vortices, as shown in Figure 1.11. A star is at the center and a convex region of influence surrounds it.

During the late 19th and early 20th centuries, Voronoi diagrams had been re-invented many times. As a result many synonyms have been used to describe basically the same techniques such as Direchlet domain, proximal polygon, S cell and tile. Okabe and John D. Rogers et al report that, during the early 1970's, many algorithms had been developed to construct Voronoi diagrams in two and three dimensions primarily stimulated by the developments in the computer science fields. The concept of Voronoi diagrams and the associated algorithms, computer implementation, and applications have proliferated over the next twenty years. Its use to construct the structures of different materials continues to help understanding of complex distributed behavior of the materials. What is of interest here is the recent application of Voronoi Diagrams to the random arrangement of recording media grains and the application of Voronoi Diagrams to automatic mesh generation in numerical solution of partial differential equations (Taniguchi, Arakawa, and Kobayashi 1991; LaBarre 1992).

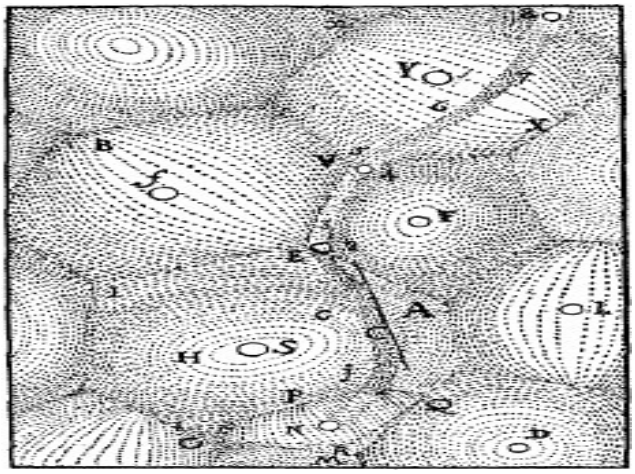


Figure 1-11 A Solar System Consisting of Vortices

1.4.2 Theory and Basic Properties

The concept of Voronoi Diagrams is simple and can be easily defined. A reduced introduction is given here taken with reference to [11]. Firstly, the distance between two points *p* and *q* is denoted dist (*p, q*). In the plane, dist (*p, q*) is given by:

$$dist(p,q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Let $P := \{p_1, p_2, ..., p_n\}$ be a set of *n* distinct points in the plane; these points are the cells (or sites). The Voronoi diagram is defined as the subdivision of *P* into *n* cells, one for each site in *P*. A point *q* belongs to a cell p_i (corresponding to site p_i) if and only if $dist(q, p_i) < dist(q, p_j)$ for each $p_j \in P, j \neq i$. The Voronoi diagram *P* is denoted *Vor(P)*. The cell of *Vor(P)* that corresponds to a site p_i is denoted $V(p_i)$. In other words, $V(p_i)$, as a subset of *Vor(P)*, is the locus of points *q* such that each point in $V(p_i)$ is nearer to p_i than is any point not in $V(p_i)$ (Brostow, Dussalt, and Fox 1978). Secondly, the structure of a single cell is described as follows. For two points *p* and *q* in the plane, its bisector is defined as the perpendicular bisector of the line segment $\overline{pq}$. This bisector splits the plane into two half-planes. The open half-plane that contains *p* is denoted $h(p,q)$ and the open half-plane that contains *q* is denoted $h(q,p)$. Notice that $r \in h(p,q)$ if and only if $dist(r,p) < dist(r,q)$. Therefore, $V(p_i)$ can be defined as:

$$V(pi) = \mathrm{I}_{1 \le j \le n,\, j \neq i} h(p_i, p_j)$$

Thus, $V(p_i)$ is the intersection of *n-1* half-planes and an open convex polygonal region bounded by at most *n-1* vertices and at most *n-1* edges, as illustrated in Figure 1.12.

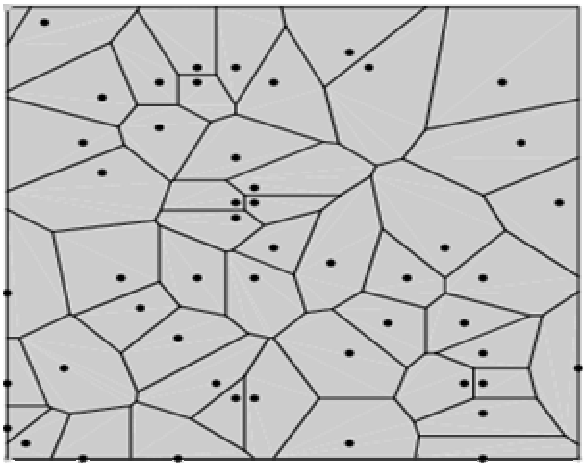


Figure 1-12 Voronoi Diagrams as a Planar Subdivision with Straight Edges

A generalized Voronoi diagram is defined as follows.

Definition 1. *The Generalized Voronoi Diagram in d dimensional Euclidean space, E^d, is a set of unique convex regions or cells called Voronoi Polyhedrons (V_i). These polyhedrons contain only one nucleus (P_i) (not necessarily centered within the polyhedron) of a set of nuclei distributed within a larger space. The polygons under discussion bound or enclose a subset of space closer to a specific nucleus (P_i) than to any other nucleus (P_j). More formally:*

$$V(P_i) := \{x \in E : d(x, P_i) \leq d(x, P_j),\ j = 1,2,..n\}$$

In this thesis, the 2D planar case is of primary concern. Some of the Voronoi diagram properties are listed below [12][13][14].

Property 1. The Voronoi Diagram is a unique tessellation of the set of distinct points of *P* where

$$P = \{p_1, p_2, ..., p_n\} \subset E^2 (2 \leq n \leq \infty)$$

Property 2. Let P be a set of *n* point sites in the plane. If all the sites are collinear then *Vor(P)* consists of *n-1* parallel lines. Otherwise, *Vor(P)* is connected and its edges are either segments or half-planes.

Property 3. For $n \geq 3$, the number of vertices in the Voronoi diagram of a set of *n* point sites in the plane is at most *2n-5* and the number of edges is at most *3n-6*.

Property 4. The average number of Voronoi edges per Voronoi Polygon does not exceed six. The maximum number of Voronoi vertices in a 2D Voronoi diagram is *2n-5*.

1.5 Objective of This Study

Distributed and Collaborative Design and Engineering has become a major industrial issue. From the previous discussion, it can be seen that a substantial amount of research work has been done in the field of distributed and collaborative CAD/CAE systems based on CORBA, DCOM or Java RMI technologies. Most of these systems are suitable for deployment only within the Intranet. For example, DCOM has its limit when the system is deployed over the Internet because it tends to communicate over a range of ports that are typically blocked by the network firewall. The .NET Remoting technology from Microsoft has been tailored to solve this problem. However, how to integrate robust 3D geometric modeling functionalities into a distributed and collaborative environment so as to better prepare the geometric models for engineering simulation and analysis remains a critical problem to be solved.

Therefore, this thesis proposes a three-tier architecture called CoCADE (Collaborative Computer Aided Design and Engineering) to facilitate the development of

an open distributed and collaborative CAD/CAE system based on .NET Remoting technology. Robust 3D geometric modeling functionalities are integrated into the system to allow users to conveniently browse and manipulate a product model as well as process the model for numerical solution by leveraging on powerful CAD stations. Necessary design coordination mechanisms are also provided to guarantee an effective collaborative session without generating conflicts. Meanwhile, scientific data presentation tools with rich features are provided to give intuitive and vivid visualization of engineering simulation results. The general structure of CoCADE is shown in Figure 1.13.

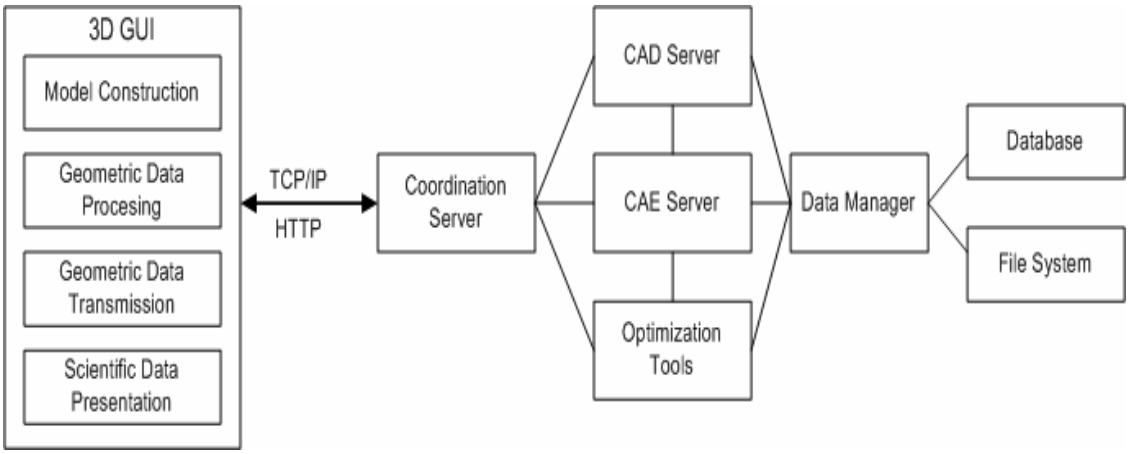


Figure 1-13 General Architecture of CoCADE

The fat client side consists of several modules such as Model Construction, Geometric Data Processing, Geometric Data Transmission and Scientific Data Presentation. The server side comprises five components. The coordination Server is responsible for managing design sessions and facilitating client-server communication. The CAD Server is used for geometric model construction and the CAE Server for CAE tasks. The Data Manager handles database and file system transactions to ensure data consistency.

1.6 Dissertation Outline

A brief dissertation outline is given as follows. Chapter 1 introduces the research background related to co-design and co-simulation in collaborative product development and some key technologies used to build distributed systems. The three-tiered software architecture of Collaborative Computer Aided Design and Engineering system (CoCADE) is discussed in Chapter 2. In Chapter 3, the detailed implementation of geometric modeling functionalities to support co-modeling and co-simulation in distributed design is presented. Chapter 4 focuses on the implementation of collaboration functions in CoCADE. In Chapter 5, a case study based on CoCADE is given to demonstrate its feasibility and effectiveness. Finally, conclusions and future work are discussed in Chapter 6.

2. CoCADE: A Framework for Collaborative Modeling and Simulation in Product Development

2.1 System Architecture of Collaborative CAE Framework

A framework for distributed and collaborative computer aided design and engineering, CoCADE, employs a three-tiered Client/Server architecture as shown in Figure 2.1. Different modules of a distributed system spread across the network with care taken in maintaining network load balance and maximizing the utilization of the client machine's performance. Applications are divided between client side software and server side software. The fat client method is adopted while developing the client software to offer full local design and interactive functions efficiently. HTTP, TCP/IP and FTP protocols are supported to fulfill the communication between client and server.

It is aimed to offer the following facilities for distributed and collaborative design and engineering: (1) an integrated CAD/CAE environment for multiple users to browse and operate on a shared solid model over the Intranet/Internet synchronously, (2) effective geometric data processing functionalities to facilitate numerical solutions for engineering simulation, (3) graphical visualization tools for large-scale engineering simulation results, (4) an infrastructure to enable users to access workflow services.

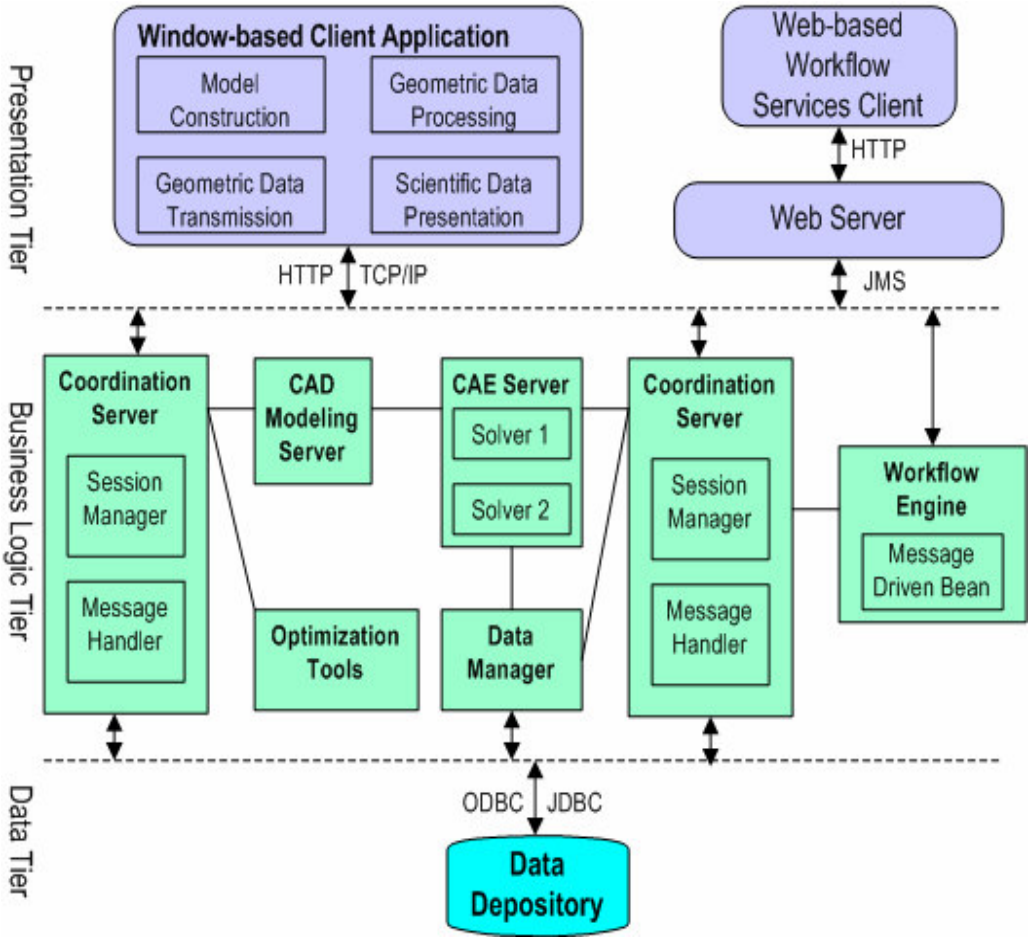


Figure 2-1 Software Architecture for Distributed and Collaborative Design and Engineering

As shown in Figure 2.1, the three-tiered software system can be deployed on either Intranet or Internet. The presentation tier is usually a stand-alone Window-based application with rich CAD/CAE /Collaboration functions. In addition, a thin Web-based workflow client that is embedded into a browser can be used to enable the users to access workflow services. The workflow client comes with workflow editing functions. The

business logic tier, which consists of multiple functional modules, is the central coordination part of CoCADE. The workflow engine as the core of the workflow model is responsible for explaining and executing the messages from Web-based client. The workflow can communicate with the other modules of the server side via Coordination server to keep itself updated about the collaborative sessions activities. The messages transferred between them can be in XML format. The data tier is a data depository used to provide persistent storage and maintain product data consistency.

2.2 User Case View of CoCADE System

From an application user's point of view, a qualified CoCADE system must meet the following general requirements:

- The users should be provided with a friendly GUI with 3D modeling, mouse-clicking and dialog-based tools.
- Major industry-recognized CAD/CAE file formats can be loaded into the GUI for viewing and editing. General geometric modeling and operation functions should be available.
- An effective user management mechanism should be devised to grant team members different privileges according to their roles in the team.
- A coordination mechanism should be devised to ensure that collaboration between members would not result in conflict. Shared data consistency should be guaranteed.
- Collaboration tools such as text-chat or audio/video tools should be available to ease communication process among team members.

- Any changes such as marking up or highlighting made by a team member on a shared model should be reflected on the screen to other members.

To better describe these domain-dependent requirements, the schematic diagram is used to show the use case for the online session as in Figure 2.2. A team of designers with different industrial expertise usually performs the product development work together. Therefore, the CoCADE system must provide an effective mechanism to manage the team member's privilege according to their roles and positions in the design team. From the perspective of a team manager, there are four role types of members who have access to the application; they are the leader or initiator, CAD member, CAE member and workflow planner. The leader or initiator is the only one who has the privilege to create/terminate a collaboration session. Meanwhile, the leader can approve or reject the other members' application to join or leave the session. CAD and CAE members are granted the status according to their expertise so they can use the CAD or CAE functions with minimum external support. Any member who has the privilege to collaboration functions can discuss with each other throughout the whole design session. The workflow planner plans and harmonizes the whole product development process. All the CAE or CAE members must follow the pre-defined workflow and perform product design activities accordingly.

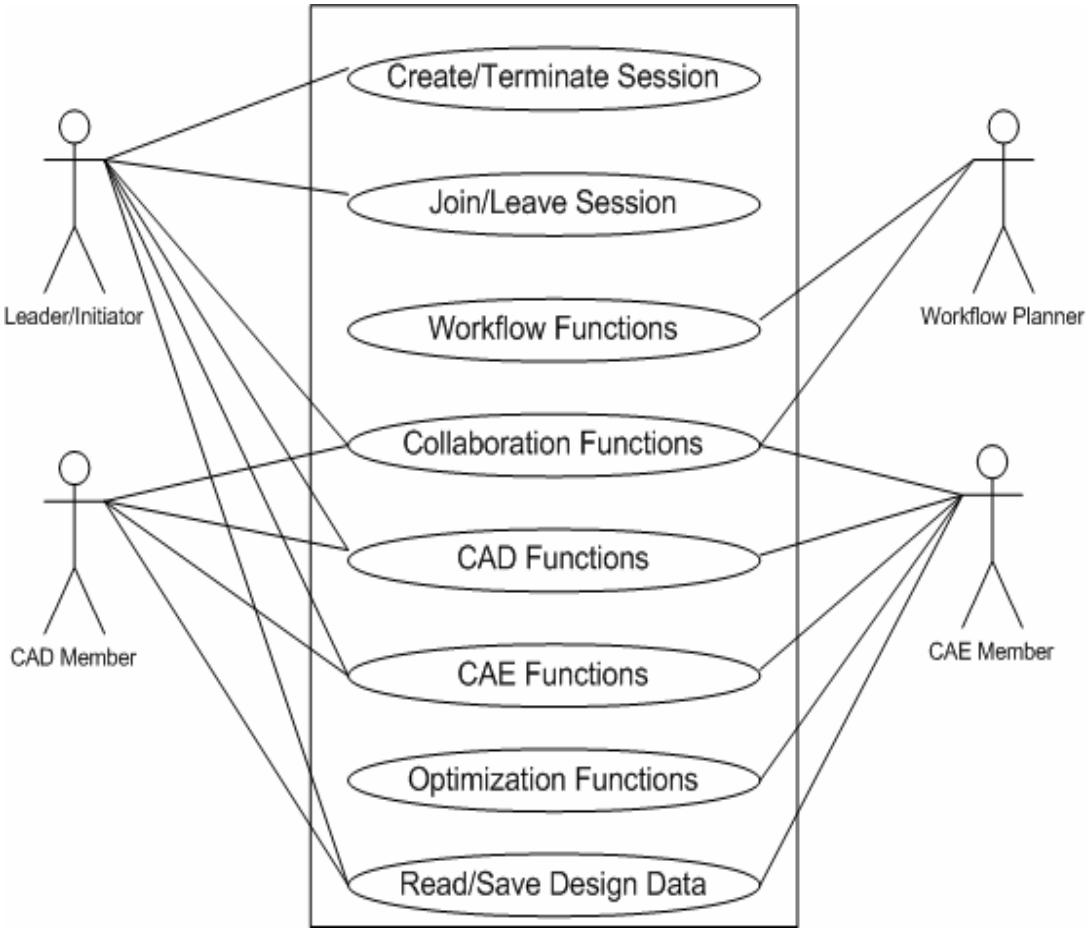


Figure 2-2 Users Case Diagram of CoCADE System

2.3 Data Streaming View of CoCADE System

As the development of product design advances, the amount of raw product design data such as large 3D solid models and engineering simulation result, is increasing speedily. Visualization of the product data provides benefits in helping detect and resolve problem early in the product development cycle, thus saving time and cost. Traditionally, these

raw data could not be visualized before the entire data sets are entirely received. In a distributed design environment like CoCADE, especially in a real-time collaborative design process, it creates a bottleneck for updating effectively so as to keep the clients' information consistent with the server.

Streaming technology is one of the key technologies for efficient transmission and visualization of high volume data. Although this technology has already being applied to audio and video transmission, it is still a research challenge to develop a strategy for the effective transmission and visualization of engineering data. Engel et al. introduced a web-based visualization approach to enable the remote control of a visualization application by image streaming and CORBA (Common Object Request Broker Architecture) techniques [2]. This approach is only applicable in some basic visualization functions. It shows limitations, e.g. level of user interactivity, multi-user capabilities, rendering synchronization. The use of CORBA technique also exposes security problem because it requires the configuration of network firewall. Mahovsky et al. have developed a Java-based system for real-time distributed visualization by using a cluster of conventional PCs [3]. This system adopts an extensible and object-oriented model that accommodates a variety of rendering algorithms and data sets. However, collaborative visualization function for multiple users to operate on a shared data set is not supported in this system. The system is not feasible to be deployed on the Internet so as to provide remote real-time visualization facilities.

In this research, we describe a sub-system of CoCADE that supports real-time collaborative visualization of complex engineering geometric models and simulation results. Multiple users can conduct engineering design activities such as real-time co-

modeling, simultaneous 3D view and mark-up, text-chatting etc. Data streaming technology is utilized in the scenario where data sets are to be re-ordered according to the clients' perspective before transmitted. In addition, simplified mesh approximation of the geometry of each individual part may also need to be created and downloaded in a hierarchical order, therefore, each part can be delivered incrementally. An open standard stream-able format for exchanging of visualization data between disparate applications, HSF, is employed in this platform and will be extended to support varied custom types of engineering data.

From the perspective of visualization data flow, Figure 2.3 shows the architectural modules of CoCADE as a real-time collaborative design system, which includes engineering simulation module, data streaming server module and interactive visualization module. The system is developed on the basis of the object-oriented design concept.

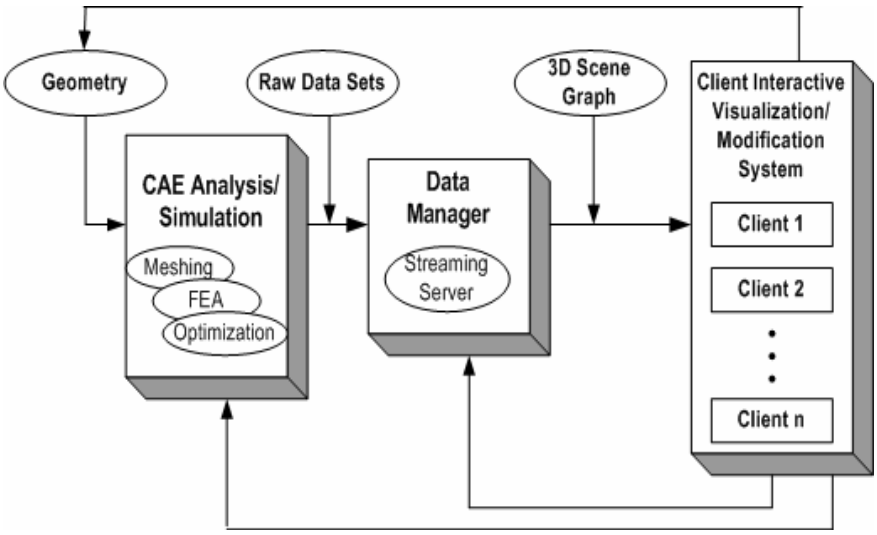


Figure 2-3 Data Streaming View of CoCADE System

Firstly, the CAE simulation module takes in input data the geometric models with various attributes such as material properties, center of gravity etc. The CAE simulation module uses appropriate computational means like FEM or BEM to carry out the engineering simulation process. As a result, large-scale raw data sets are generated during this phase. Sequentially, these raw data sets are post processed by efficient data extraction algorithms in the streaming server. The streaming server module delivers highly elegant 3D scene graphs in streaming file formats (HSF Format) to represent the characteristics of simulation results. Finally, the 3D scene graphs are sequentially streamed to the distributed real-time interactive visualization system and displayed to clients.

2.4 Static Class View of CoCADE System

The UML static class view of CoCADE system is shown in Figure 2.4. On the client side, the MFC Document/View application architecture is adopted which includes CoCADEApp, CoCADEDoc, CoCADEView and CoCADEFrm (CoCADEChildFrm). The other classes developed to accomplish functional modules are centered on the classes.

The CoCADEDoc class provides the basic functionalities for user to access geometric data. Standard operations are supported, such as creating a geometric model, loading and saving. The file can be loaded and saved locally on client machine or remotely on server machine, depending on whether the file resides on client or server machine. The above mentioned functions are implemented in the methods of OnOpenDocument(), OnNewDocument() and OnSaveDocument().

The CoCADEView class associated with CoCADEDoc class is the interface through which the user can interact with the geometric data. The fundamental CAD/CAE visualization and operation tools like zoom, rotation, wire-frame, markup, Boolean operation and sweeping etc, are all contained in this view class. In addition, this class is characterized by providing rich features for three-dimensional visualization of large-scale CAE simulation result data sets. For example, the popular tetrahedral mesh suitable for numerical solution can be displayed and evaluated before sending to the FEM solver. After simulation is carried out, simulation results can be displayed, for example, by colored arrows that represent 3D vectors at the nodes in the spalce of the solution domain. Another issue of concern while developing distributed system is how to keep the volume of geometric data transmitted across network to the minimum level and, in the meantime, give user a satisfactory visual effect of geometric data. Data streaming technology discussed in the next chapter is used to solve this problem.

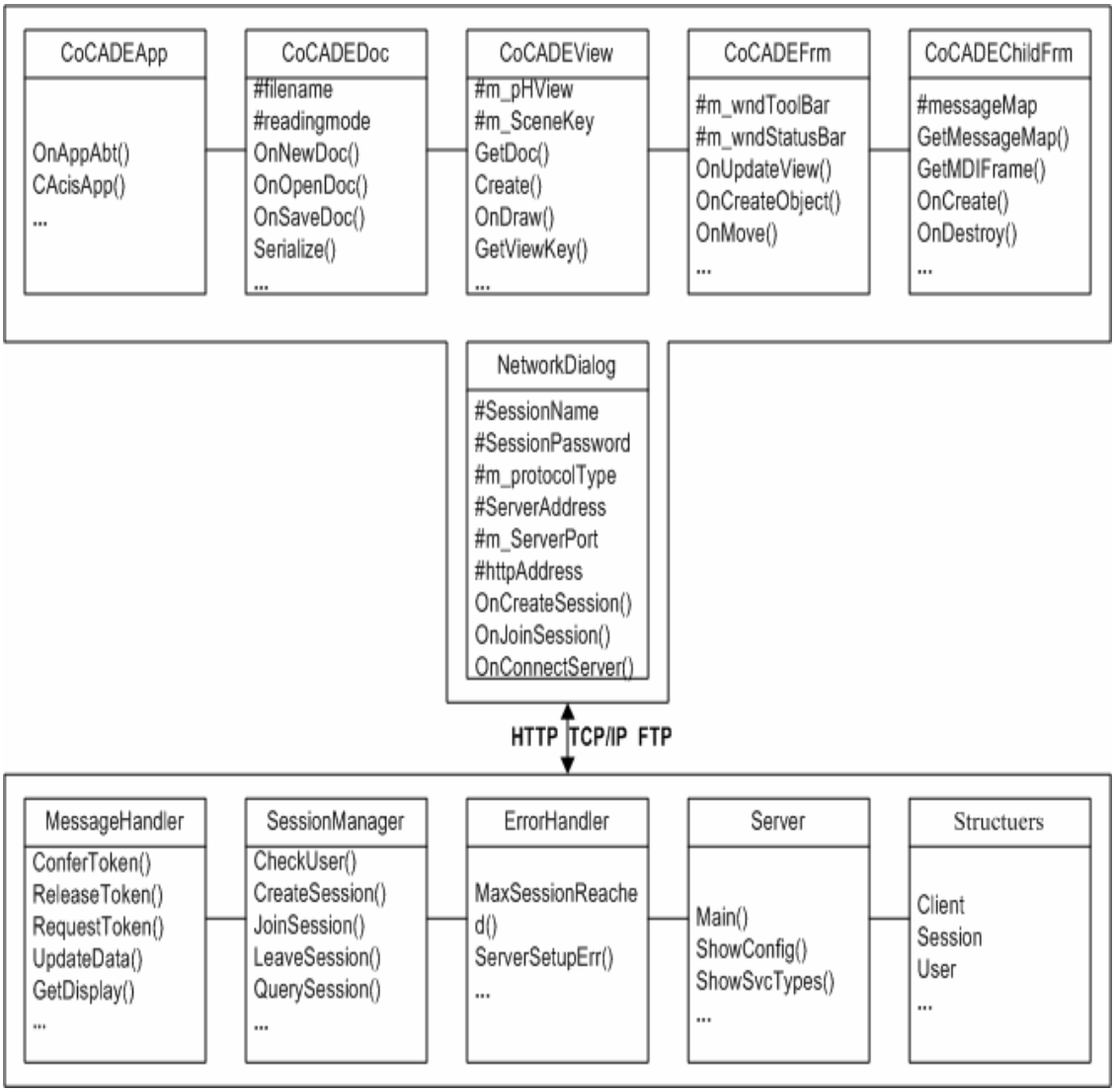


Figure 2-4 Static Class View of CoCADE System

The class NetworkDialog encapsulates all the response functions that a user may need to communicate with a server. The method OnCreateSession() is used by the leader/initiator to initialize a collaboration session. After that, other team members can

join the session by providing #SessionName, #SessionPassword and other information like server address or URL address, and invoking the method OnJoinSession() and OnConnectServer().

In a distributed and collaborative design system, the design process highly depends on the effective communication among the multiple clients as well as between the clients and the server. On the server side, MessageHandler is responsible for managing the collaborative design process between clients. This class provides the method of ConferToken() to grant a team member a token so that he will have the privilege to modify the geometric data in a collaborative session. Only one member in a design team can hold the token in a design session. This is to eliminate the possible conflict of concurrent multiple modifications and ensure data consistency. A team member can be forced to release the token to other members by the leader/initiator. Also, when a shared geometric data is modified, the change must be immediately broadcast to all other members in the same team. This task is accomplished via the method of UpdateData().

Another issue of concern is the session management in the collaborative design process. The collaborative activities can occur synchronously or asynchronously, where the team members contribute to a product design session with their expertise. In the asynchronous scenario, the designer may carry out different sessions simultaneously depending on whether or not these sessions are independent on each other. In the synchronous scenario, some team members can carry out the same design task collaboratively in real-time mode. SessionManager class is devised to handle both the synchronous and asynchronous collaborative sessions.

The ErrorHandler class is used to safeguard the design sessions according to computer resources such as memory, CPU usage etc. Each session has a pre-defined maximum number of members who can collaborate simultaneously. Error message would be returned if this max number were reached. This is to ensure the design session would not be suspended due to overuse of computer resources.

3. Client Interface and Geometry Processing Module

3.1 Overview

The client side in CoCADE, called CoClient, is presented as a Graphical User Interface (GUI) for users to input data and output results. It is designed as a multi-window interface that supports a variety of geometric modeling functionalities. Moreover, different clients in different locations can conduct real-time collaborative design activities like co-modeling, co-viewing, co-simulation, and text-chat discussion. After constructing geometric models and assigning geometrical parameters and properties to them, the geometric models are further processed to generate irregular grain shapes to emulate magnetic recording media models. The goal is to prepare the geometric models suitable for engineering numerical solution. The grain generation algorithm is based on the concept of the Voronoi algorithm.

This chapter first discusses the graphical user interface and interactive functionalities in CoClient, followed by study of data collaboration for geometric model construction and discretization of the design model for engineering numerical solution. Finally, visualization of engineering simulation results is given.

3.2 Design Details of Graphical User Interface

In CoCADE, the client module is designed as a multi-window user interface that supports various modeling functionalities such as creation of solid primitive models, viewing and

marking up of solid/mesh parts, Boolean operations, file formats conversion, engineering simulation result display and so on.

The “fat client” mode is employed in the client-side programming to provide powerful functionalities for CAD/CAE usage so the client-side program can be used as a stand-alone tool for product design when the user is working in asynchronous mode. The web-based approach often leads to restriction on client side functionality and may compromise efficiency. As a result, the CoClient is deployed as a stand-alone MFC application in an EXE file to fully utilize client computer resources and reduce network load, while the workflow editor is deployed as a browser-based application downloaded from server-side because it is a data intensive application as compared to computation intensive applications and a thin client approach is well suited. This deployment scheme, however, requires the installation of software modules on the client-side before it can be used.

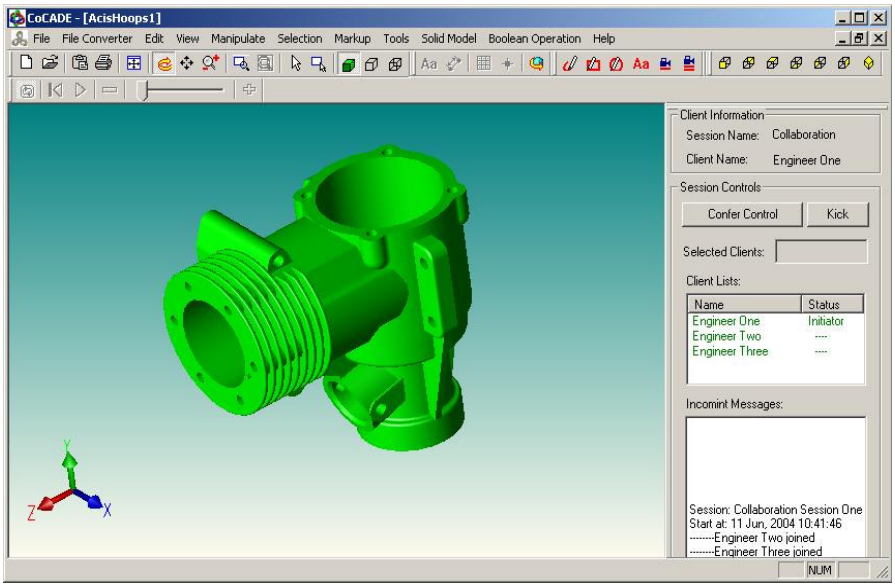


Figure 3-1 Snapshot of CoCADE Client Interface in a Collaborative Design Session

Figure 3-1 is a snapshot of the client interface in a collaborative design session. On the right hand side of the interface, the collaboration information is shown, which includes session and client names, client status (leader or team member), and real-time network connection messages. As shown in Figure 3-2, the web-based workflow client is developed to help coordinate the product design process. The workflow planner predefines the design and simulation sequence by using workflow charts and diagrams. These charts and diagrams are saved in XML files on the server side and can be downloaded to the client's browser. This is to ensure that all the members in a same team will follow the pre-defined design process and be kept updated about the design process immediately once a design task is finished.

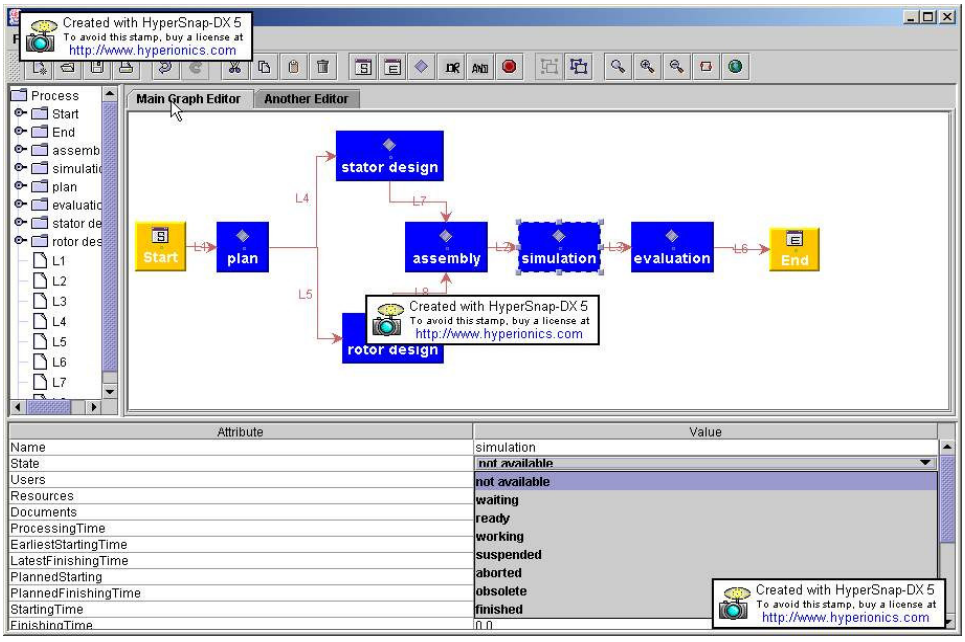


Figure 3-2 Snapshot of CoCADE Web-based Workflow Client

3.2.1 Interactive Functionalities Supported

In CoClient, extensive interaction functionalities for CAD modeling and CAE analysis are supported. This interaction module resides on the client side. It takes advantage of the ample functions typically provided in a Window-based environment. A full range of interactive functionalities is implemented to facilitate the modeling process, e.g. mouse clicking, dialog-based, and keyboard input etc.

The classical concept of Object-Oriented (OO) analysis is used to effectively design and implement different functions in the interaction module. Each modeling function is encapsulated in a class. The class controls the behaviors of mouse click and keyboard input and invokes the relevant method in the class to respond. For example, Figure 3-3 illustrates the sequence of respective response to the event of mouse clicks when a user creates a block and a cylinder and then intersects both primitives by invoking the Boolean Operation method.

The block and cylinder creation requires user to specify two and three vertex coordinates respectively. For the block, they are the coordinates of two ends of its diagonal. For the cylinder, they are the two coordinates for two ends of its circle diameter and the coordinate for its height. To create a block and a cylinder with mouse clicks, a user first clicks the mouse to specify the first position of each primitive. After the first position is specified, the geometric toolkit provides a “rubber band” tool to draw a virtual frame of the block or the cylinder circle until the user specifies the second position. Then the virtual frame disappears. For the cylinder, the virtual frame disappears when coordinates of the third point is defined.

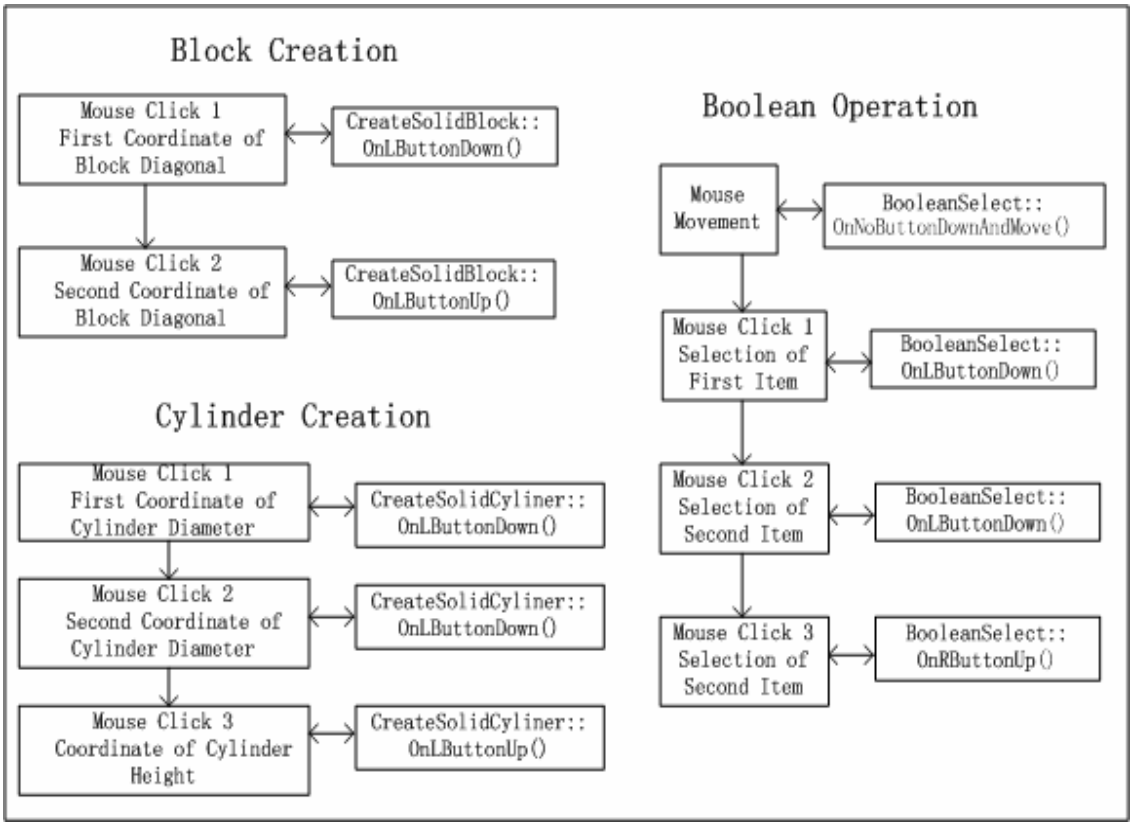


Figure 3-3 Sequential Responses to Mouse Click Events

Each primitive is treated as an object in CoClient. Before carrying out a Boolean operation, the user needs to select the first object called "tool" object and the second object called "blank" object. The "blank" object will be returned as the resulting object of the Boolean operation. The following code shows the details of how the Boolean operation is implemented.

```
int HOpBooleanSelect::OnRButtonUp(HEventInfo &event)
{
    BooleanOperation m_pBO=GetBooleanOperation();
    if ( m_pBO!= none)  //m_pBO is the pointer to selected objects
```

```
    {
HSelectionSet * m_Selection=(HSelectionSet *) m_pView->GetSelection();

        int iSelCount=m_Selection->GetSize(); //number of selected items
    if (iSelCount<=1)
        {
                m_Selection->DeSelectAll();
                SetBooleanOperation(none);
    }
    else{
                ENTITY *        tool=0;
                ENTITY *        blank=0;
                ENTITY *        result=0;
                outcome o;
                BOOL_TYPE bp;
                HC_KEY hc_tool, hc_blank;

                int iBlank=1;

                hc_tool=m_Selection->GetAt(0);
                tool=HA_Compute_Entity_Pointer(hc_tool,BODY_TYPE);
    ((HSolidModel *)m_pView->GetModel())->DeleteAcisEntity(tool);

                o=api_initialize_booleans();
                assert(o.ok());
           switch (m_pBO)
        {
                        case  unite:

                                bp=UNION;
                                break;

                        case  subtract:
```

```
                                    bp=SUBTRACTION;
                                    break;

                            case  intersect:

                                    bp=INTERSECTION;
                                    break;
                }
                while(iSelCount!=iBlank){
                        hc_blank=m_Selection->GetAt(iBlank);
                        blank=HA_Compute_Entity_Pointer(hc_blank);
    ((HSolidModel *)m_pView->GetModel())->DeleteAcisEntity(blank);
    //Carry out the operation by using API provided by ACIS kernel
        o=api_boolean((BODY*)tool,(BODY*)blank,bp,
        NDBOOL_KEEP_NEITHER,(BODY*&)result);
                                assert(o.ok());
                                tool=blank;
                                blank=0;
                                iBlank++;
                        }
                        HC_Open_Segment_By_Key(m_pView->GetModel()-
>GetModelKey());
// Put each entity in a separate segment so that we can easily apply
// transformations to separately created entities

                        HC_Open_Segment("");
                        HA_Render_Entity((ENTITY*)tool);
    ((HSolidModel *)m_pView->GetModel())->AddAcisEntity(tool);
                        HC_Update_Display();

                        o=api_terminate_booleans();
                        assert(o.ok());

                        HC_Close_Segment();
```

```
                    m_Selection->DeSelectAll();
                }
            }
            SetBooleanOperation(none);
            return(HOP_READY);
        }
```

3.2.2 Implementation Class Hierarchy

As CoClient is mainly designed to run on Microsoft's Windows platform, it naturally adopts the Document/View application architecture of Microsoft Foundation Classes (MFC) and HOOPS/ACIS interface for MFC as development tools. HOOPS toolkit provides an application wizard to create the application skeleton of CoCADE and connect this skeleton to MFC. Figure 3-4 illustrates the implementation class hierarchy of CoCADE.

In CoClient, the concepts of object, class, polymorphism and inheritance from OO analysis are used as main development approaches to build the system. The advantage of the object-oriented design includes early identification of system requirements and application of a consistent approach throughout the entire software development life cycle. It also facilitates reusability and maintenance of the software source codes.

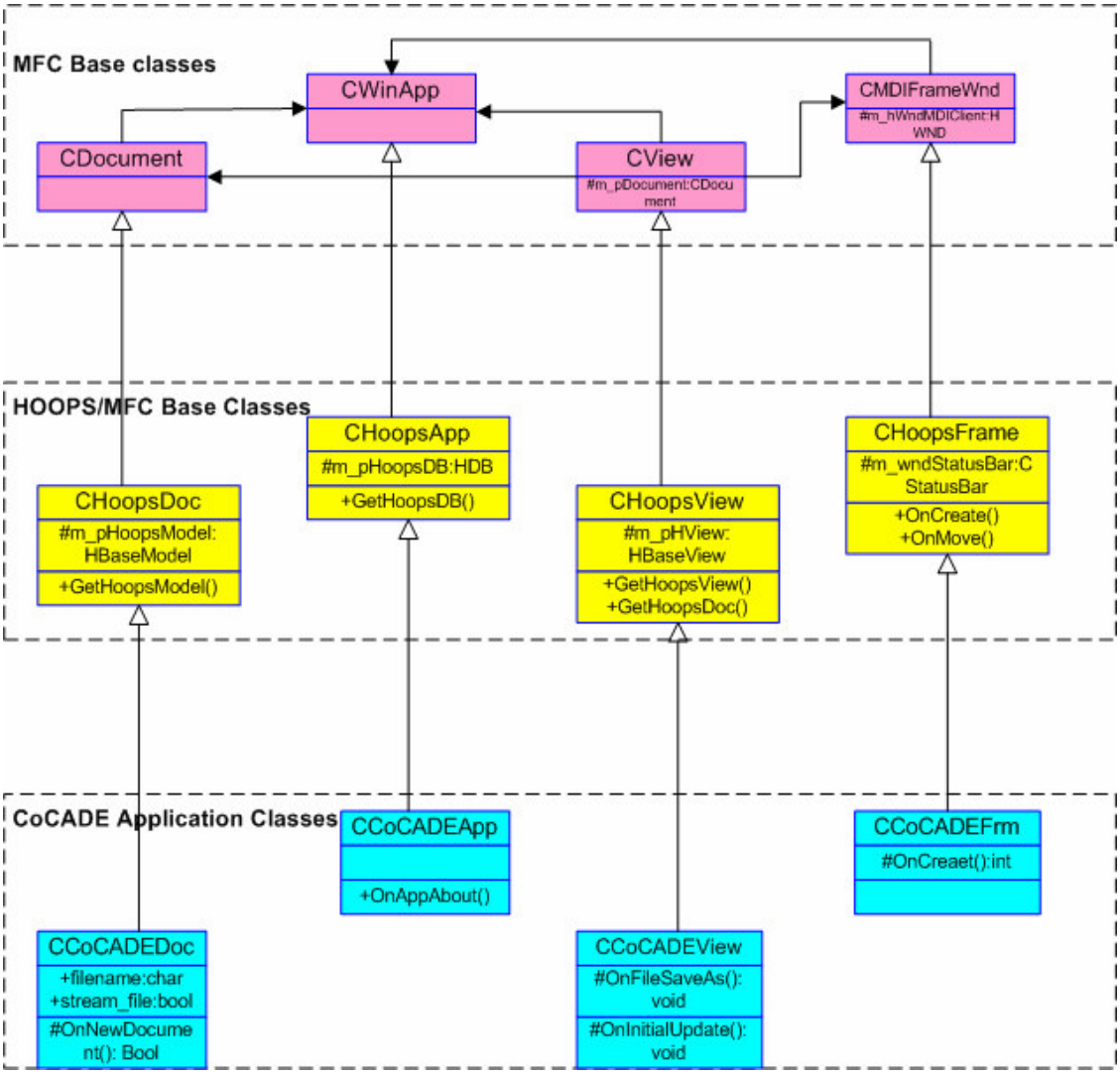


Figure 3-4 Implementation Class Hierarchy of CoCADE

3.3 Data Collaboration

3.3.1 Geometric Data Structure

In CoCADE, the ACIS geometric kernel is used to construct design models. This geometric kernel uses Boundary Representations (B-rep) data structure to describe models of arbitrary complexity. An entity is the most basic ACIS object that is implemented in C++ class ENTITY. All other geometric and higher model objects are derived from the ENTITY class.

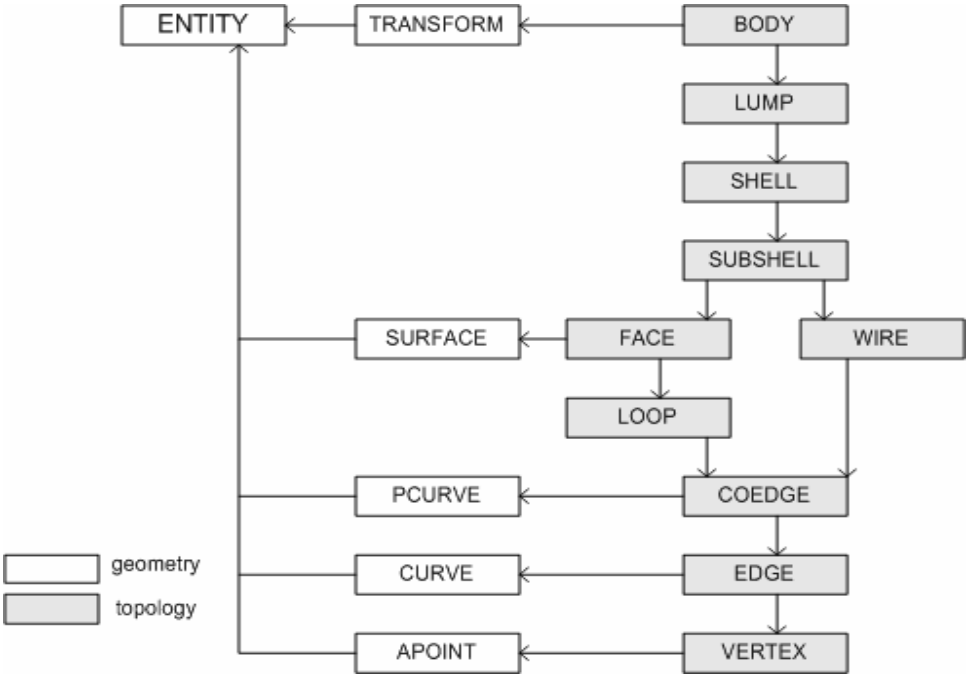


Figure 3-5 Model Object Decomposition Data Structure

Figure 3-5 shows the model object decomposition structure. Different entity classes like the solid, sheet, wire and mixed classes can be decomposed into the above classes by following the structure from top to bottom. There are two types of class: geometry class and topology class. The topology class contains pointers to the corresponding geometry class. These classes provide the data and methods that are tailored to support the operations of a solid modeler. The arrows show the hierarchical relationship between different classes and the data structure traversal direction to search a specific entity. All these geometric data are assembled into a central class called ENTITY_LIST that has variable length. The contents of ENTITY_LIST are hashed so that they can be located relatively quickly.

3.3.2 Geometric Data Transmission

Product development process requires people from different fields of expertise and research backgrounds to collaborate. The design model construction or modification operations incurred by a team member must be displayable on the other members' screens. Therefore, an effective geometric data transmission mechanism should be devised to meet this requirement.

As discussed in the previous chapter, there are two types of clients in a collaborative design session: leader/initiator and team member. There can be only one leader/initiator who creates the design session and leads the design activities while there can be multiple team members (CAD members or CAE members) who undertake the design activities. A token will be circulated around team members in the online collaboration session. Only the one who holds the token has access to modify the

original data. The member who holds the token is not necessarily the same member who holds the original data.

The leader/initiator designates one team member to hold the original copy of design data in ACIS format. Each of the other members has a local copy of the design data for visualization in HSF format. The creation or modification of a design model must be applied on the original data. The design data of other members' sites are updated immediately after a modification takes place. Figure 3-6 shows the flowchart of creating a solid cylinder in a stand-alone mode or collaboration mode.

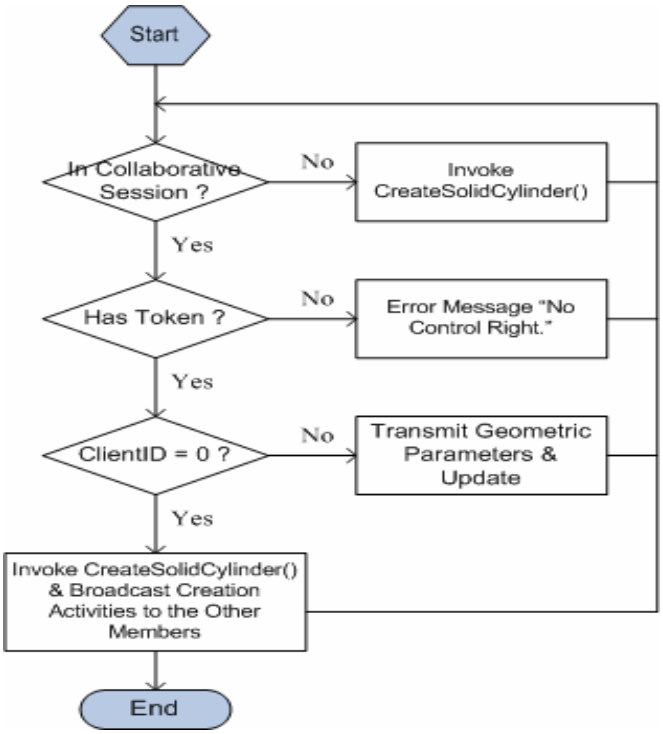


Figure 3-6 An Example of Collaborative Solid Model Creation

The program decides whether the client is in collaboration mode according to the variable m_pClientDlgBar. If the client is not in a collaborative session, then the method CreateSolidCylinder() will be simply invoked to create the model locally without

transmitting geometric data to the other clients. Otherwise, if the client is in a collaborative session and holding a token, the program will look at its ClientID variable. If the ClientID is not zero, which means the client holding the token does not have the original design model residing in its machine, the input geometric parameters will be transmitted as instructions to the client who holds the original design model. The results will be broadcast to the other clients to update their models. If the ClientID is zero, which means the client has the original design model residing in its machine, it is not necessary to transmit the geometric parameters as commands to the other clients. The modification will be directly applied on the original data residing in its machine. Only the result information is transmitted to the other clients for visualization. The transmitting message is tagged with either "O_OPERATION_CREATE_CYLINDER" or "V_VIEW_CREATE_CYLINDER" to indicate whether it is a modification operation or a visualization update operation. For the client where the original data resides, the entity_list_class.add() method adds an entity to the list. The list can be saved as .SAT or .SAB files.

The following codes illustrate how the creation of a solid cylinder model is implemented.

```
void CoCADEView::OnSolidmodelSolidcylinder()
{
CCoCADSolidCylinder * m_CreateSolidCylinderDlg=new CCoCADSolidCylinder();

if (m_CreateSolidCylinderDlg->DoModal() == IDOK){
if (m_CreateSolidCylinderDlg->cyl_r==0 || m_CreateSolidCylinderDlg->cyl_h ==0){
                    AfxMessageBox("Please specify the radius or height!");
                    return;
        }
```

```
    char operationData[1024];
    sprintf(operationData,"%f %f %f %f %f", m_CreateSolidCylinderDlg->cyl_x,
            m_CreateSolidCylinderDlg->cyl_y, m_CreateSolidCylinderDlg->cyl_z,
            m_CreateSolidCylinderDlg->cyl_r, m_CreateSolidCylinderDlg->cyl_h);

            CString tempString=operationData;

            if (m_pClientDlgBar){
                    if (m_pClientDlgBar->m_hasToken){
                            if (m_pClientDlgBar->m_pNetInfor->clientID!=0){
    Cstring operationString="O_OPERATION_CREATE_CYLINDER " +tempString;
            m_pClientDlgBar->UpdateDisplayInformation(operationString,true);
                                    return;
        }
                    }else{
                    AfxMessageBox("You don't own control right currently.\n");
                            return;
        }
            }
            CreateSolidCylinder(operationData,tempString.GetLength());
    }
}

void CoCADEView::CreateSolidCylinder(const char *in_data,int data_length)
{...
    ENTITY_LIST entity_list_class;
    BODY* cyl_body;
   memcpy(data, in_data, data_length);    data[data_length] = '\0';
   sscanf(data, "%f %f %f %f %f", &btm_ctr.x, &btm_ctr.y, &btm_ctr.z,
       &cyl_radius, &cyl_height);
...
    outcome o;
    o=api_solid_cylinder_cone (bottom_center,top_center, cyl_radius, cyl_radius,
cyl_radius,NULL, cyl_body);
```

```
    assert(o.ok());

    HA_Render_Entity((ENTITY*)cyl_body);
    ((HCoCADEModel *)m_pHView->GetModel())->AddAcisEntity(cyl_body);

    entity_list_class.add(cyl_body);
    ...
    if (m_pClientDlgBar)
            if (m_pClientDlgBar->m_pNetInfor->clientID==0){
...
HUtility::URIencode("cylinders", (int) strlen("cylinders"), encoded_segment_name);

sprintf(message, "V_VIEW_CREATE_CYLINDER %s %f %f %f %f %f %d %f %f %f
%f %f %f",   encoded_segment_name, btm_ctr.x, btm_ctr.y, btm_ctr.z,
              cyl_radius, cyl_height, num_sides, pos.x, pos.y, pos.z, up.x, up.y,
up.z);

            CString messageString=message;
            m_pClientDlgBar->UpdateDisplayInformation(messageString,false);
...
}
}
```

The implementation codes for creating a geometric model from message are shown in the next page. This method is invoked on the clients who do not possess the token/control right over the design model. Unlike the original solid models constructed by ACIS APIs, the models residing in these non-token holders' machines are constructed by HOOPS APIs and for viewing only. The HOOPS API HC_Insert_Cylinder() is used to insert a cylinder that is specified with two points and a radius plus an option string to specify which of the end caps are to be drawn. The advantage here is that only a few

geometric parameters are transmitted as text message over network instead of transmitting the whole files, which often leads to heavy network load and low-efficient rendering.

```
void CoCADEView::CreateCylinderFromMessage(const char *in_data, unsigned int
data_length)
{
        char data[8192];
        HPoint cyl_center, pos, up;
        int num_sides;
        float cyl_radius, cyl_height;
...
   memcpy(data, in_data, data_length);
   data[data_length] = '\0';

   sscanf(data, "%s %f %f %f %f %f %d %f %f %f %f %f %f",
encoded_segment_name,
&cyl_center.x, &cyl_center.y, &cyl_center.z, &cyl_radius, &cyl_height, &num_sides,
&pos.x, &pos.y, &pos.z, &up.x, &up.y, &up.z);

   unsigned long n;
   HUtility::URIdecode((const char *)encoded_segment_name, segment, &n);
...
    HPoint top, bottom;
        bottom = cyl_center;
    top.x=bottom.x; top.y = bottom.y + cyl_height; top.z = bottom.z;

    HC_Insert_Cylinder (&bottom, &top, cyl_radius, "both");
...
}
```

3.4 Geometry Processing

3.4.1 Overview

This section discusses the processing of the geometric models constructed by CoClient. Processing geometric models for finite element analysis is a major factor in the time required to develop and optimize a design [15]. More specifically, the term of geometry processing in this thesis refers to two processes: Voronoi Diagrams generation and Tetrahedral mesh generation. The Voronoi Diagram generation with a given set of seeds is implemented in combination with the tetrahedral mesh generation scheme that follows in order to achieve adaptive mesh generation. Furthermore, in this particular research, the Voronoi diagram generation is also used to create the cells modeling the grain structure of magnetic recording media.

Figure 3-7 shows the different stages starting from solid model construction to mesh generation with output file from each stage. From the top of Figure 3-7, the models produced by the solid modeling module can be either sent directly to generate mesh or to generate seeds. The seeds generation is a prerequisite for Voronoi diagrams generation. Following the seeds generation is the Voronoi diagrams generation that outputs .VOR files. The geometrical parameters and properties will be then attached to .VOR files. This is followed by passing the VOR files to the tetrahedra mesh generation module, which outputs the .MESH files. Finally, the mesh files are converted into .INP file format. INP files is the required file format for the finite element analysis module in CoCADE.

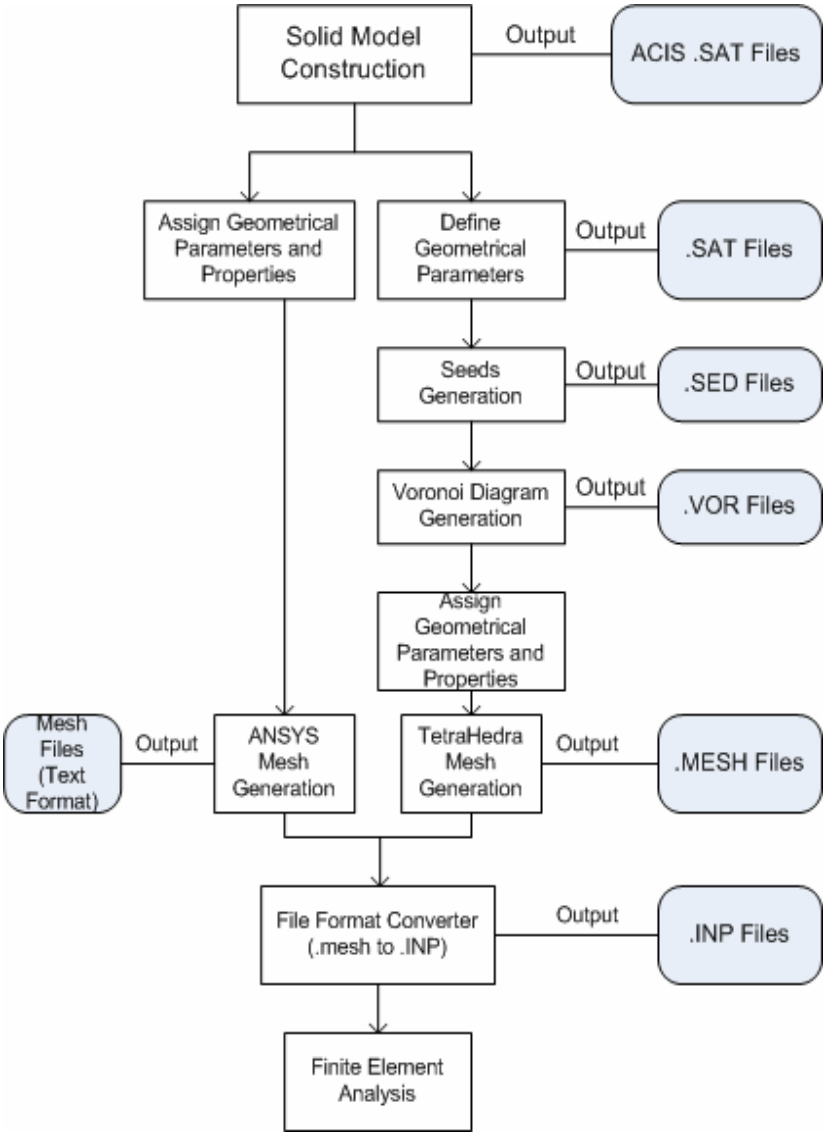


Figure 3-7 Different Stages in the Process of Geometric Processing

3.4.2 Definition of Data

The definition of data includes four types of file formats: (1) definition of seeds generation results (.SED), (2) definition of Voronoi diagrams results (.VOR), (3)

definition of mesh generation results (.MESH) and (4) definition of file format for Finite Element Analysis (.INP).

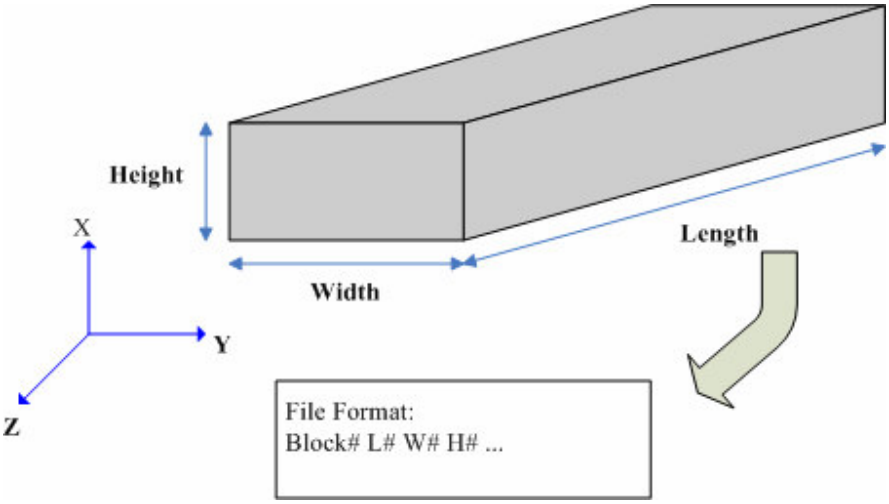


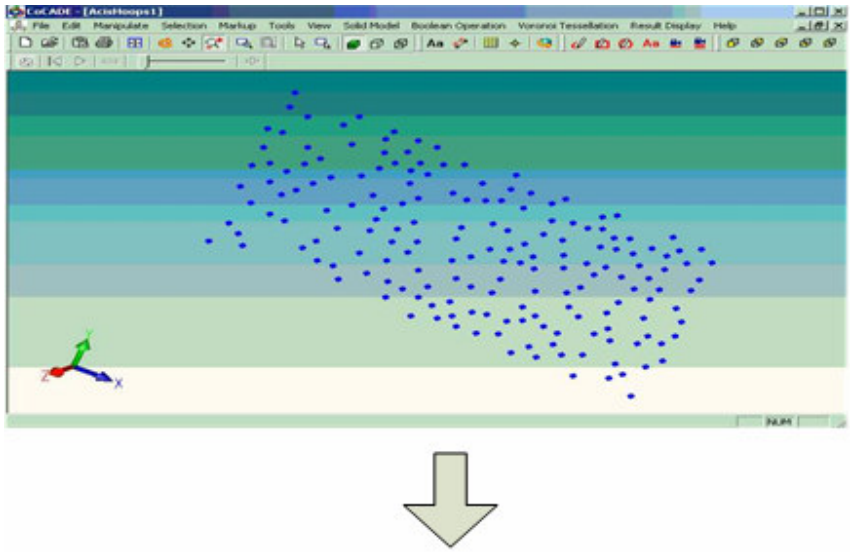
Figure 3-8 An example of Geometrical Parameters Definition

1. Definition of seeds generation results

A geometric object can be defined with geometrical parameters, which can be extracted from the solid model and can be stored in a text file when necessary. Figure 3-8 shows an example of defining a block with parameters and its associated file. The "#" symbol represents the number or the value.

Seeds generation is the first step in the process of Voronoi diagrams generation. The grains of the Voronoi diagrams are generated from the seeds, which are defined in a two-dimensional plane. For example, the plane can be the top surface of the block defined above. Care must be taken to ensure there is a minimum distance between any two seeds so that the seeds will not overlap with each other. The values of the distance

between any two seeds comply with Gaussian distribution. Figure 3-9 shows the file format of seeds generation results which can be saved in a text file with suffix "*.sed".



Seeds# coordinate-x# coordinate-y# average-size# min-dis#

1	34.040101	31.008759	6	2
2	51.480087	17.189001	6	2
3	5.402509	10.192694	6	2
4	9.607715	2.198309	6	2
5	33.760796	20.600238	6	2
6	58.140446	16.012207	6	2
7	24.852321	24.571062	6	2

...

Figure 3-9 An Example of Seeds File Format

2. Definition of Voronoi diagrams results

Given the seeds, the Voronoi diagram will be computed according to the definition described in Section 1.4.2 and output to a text file with suffix "*.vor". The file format is shown in Figure 3-10.

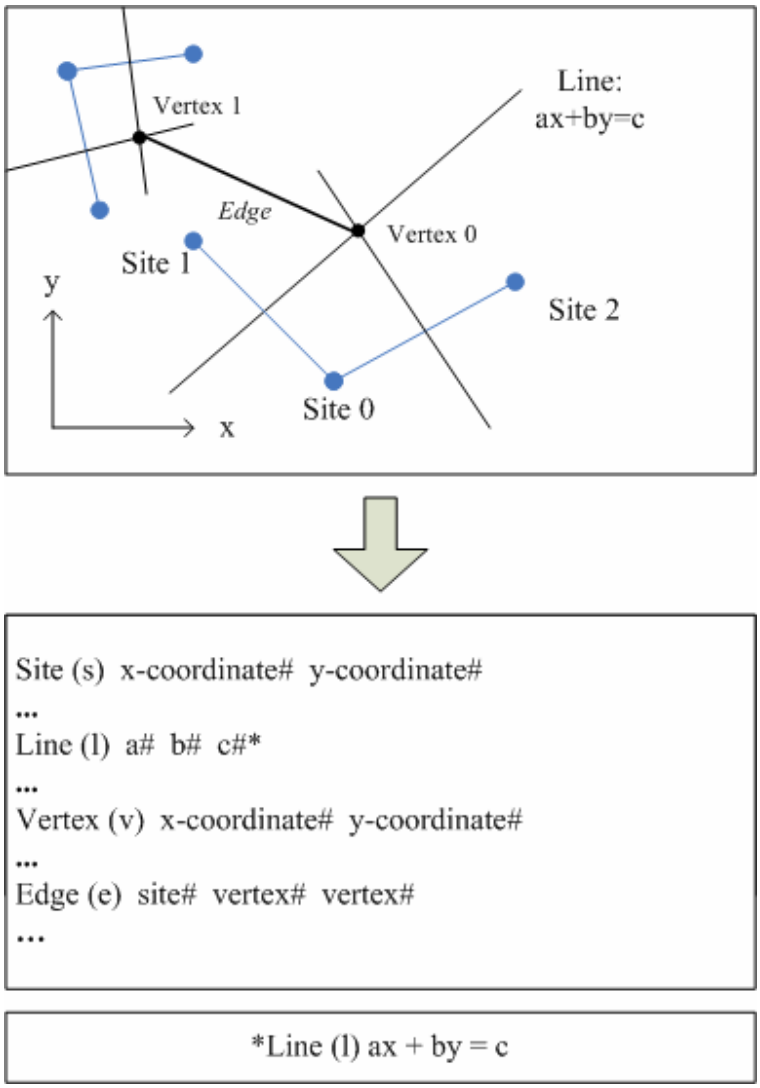


Figure 3-10 An Example of Voronoi Diagrams File Format

There are four components in a Voronoi diagrams file. The first component is the coordinate information of sites, which correspond to the seeds information. Each seed is equivalent to a site in the Voronoi diagrams. The second component is the perpendicular bisector of the line segment between the two sites. As shown in the figure, the bisecting line is an output in the form of a line formula. The third component is the vertex. As

shown in the diagram, the vertex is where the two bisectors meet. The final component is the edge representing the line segment by connecting two vertexes. As is known, each site which is numbered in the Voronoi diagram may have several edges surrounding it. The site number, to which the edge belongs, is given as the first value of the component, followed by two vertex numbers as two ends of the edge.

3. Definition of mesh generation results

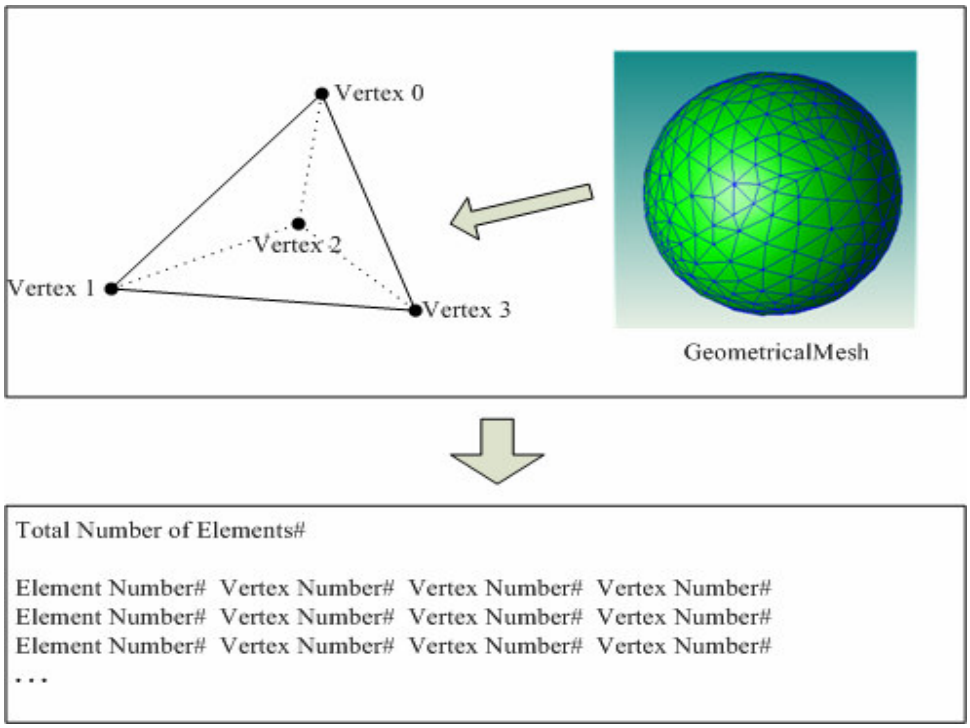
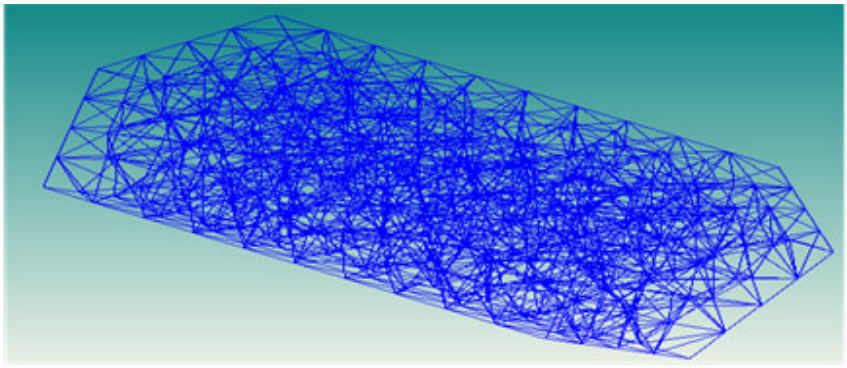


Figure 3-11 An Example of Mesh Generation File Format

The file format for the generated mesh is relatively simple as shown in Figure 3-11. Each row in the file represents an element by specifying its element number and the four vertexes forming the tetrahedral element. The file is output as a text file with the suffix "*.mesh".

4. Definition of file format for Finite Element Analysis

The file format for Finite Element Analysis (FEA) is given as an ASCII file with the file extension of "*.inp". This file format can also be used to represent the FEA results. Figure 3-12 shows the file format of "*.inp".



Total Number of Nodes# Total Number of Elements# Number of Data at Each Node# Number of Data at Each Element# Number of Data for This Model#

Node# x-coordinate# y-coordinate# z-coordinate#
Node# x-coordinate# y-coordinate# z-coordinate#
…
…
Element# Material# Type of Element# Connectivity of Nodes of the Element#
Element# Material# Type of Element# Connectivity of Nodes of the Element#
…
…
Number of Data Components at Each Node# Number of Structures# Number of Structures# …
…
…

Figure 3-12 Input File Format for FEA Analysis

One of the features of INP file format is that it is a time-dependent (multi-step) file format. Relevant simulation results from a certain stage can be attached to the trail of

the file. The geometric model's material properties can also be reflected in the file by specifying the material values.

3.4.3 Voronoi Diagrams

1. Seed generation

As discussed in the previous section, the Voronoi diagrams are based on randomly generated seeds. Figure 3-13 illustrates a Voronoi diagram for 10 randomly selected sites in a square [16]. The computation of the seeds is a Poisson point process.

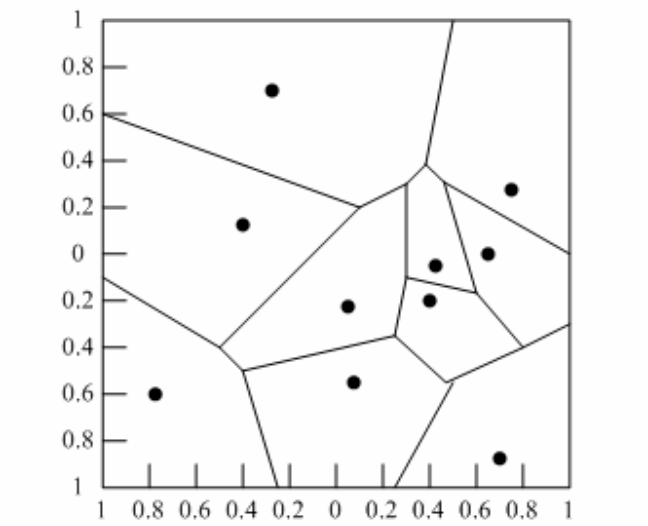


Figure 3-13 Voronoi Diagrams for 10 Randomly Selected Sites in a Square

The detailed implementation codes are as given in Figure 3.17.

```
void CSolidHoopsView::OnVoronoitessellationSeadsgeneration()
{
        double Ml, Mw, Gs, Md; //media length, width, grain size, mini-distance;
```

```
        int n, i, j, m, rseed = 11000; //generate for random()
        FILE *outs;
    if ((outs = fopen("Seeds.sed", "wt")) == NULL) {
            fprintf(stderr, "Cannot open output file.\n");
            return ; }
. . .
    n = Ml*Mw/Gs;
        // initialize the position of all seeds
    double S[1000][2];
        for (i=0;i<n;i++){
            for (j=0;j<2;j++){  S[i][j]=0.0; }
        }
    // generate seeds by random()
        srand(rseed);
    for (i=0;i<n;i++)
    {       S[i][0] = Ml * rand() / RAND_MAX;
        S[i][1] = Mw * rand() / RAND_MAX;
        if ( i > 0){  m = 1;  // sign for regenerate a seed
                    for ( j = 0; j < i; j++)
                    {       if ( sqrt((S[i][0]-S[j][0])*(S[i][0]-S[j][0])+(S[i][1]-
S[j][1])*(S[i][1]-S[j][1]))<=Md ){  m=m+1; }
                    }
            if ( m>1 ){ i=i-1; }
            else{   fprintf(outs,"%d %f %f\n",i,S[i][0],S[i][1]);   }
    }
     }
}
```

2. Computation of Voronoi diagrams

There are many algorithms and their variations that are proposed to construct Voronoi diagrams. One of the classifications by Fortune [17] divides these into three categories – incremental algorithms, divide-and-conquer algorithms and sweep-line algorithms. The algorithm used in this study is the sweep-line algorithm adopted from [11].

The sweep-line algorithm is efficient and popular. The algorithm computes the Voronoi diagrams by sweeping the plane. When the sweep-line sweeps the plane, it encounters the site (or seeds) for each grain before intersecting the geometric area. The geometric area here refers the regions around the site (or seeds). Before the sweeping line starts sweeping, the seeds are sorted in ascending order according to their values of y-coordinate and x-coordinate.

A detailed discussion of the algorithm can be found in literature, e.g. textbook on computational geometry books. Here, only the overall structure of the sweep-line algorithm is given in the following code.

Input. A set $P := \{p_1, ..., p_n\}$ of point sites in the plane

Output. The Voronoi diagram Vor (P) given inside a bounding box in a doubly connected edge list D.

1. Initialize the event queue Q with all site events, initialize an empty status structure T and an empty doubly connected edge list D.
2. **while** Q is not empty
3. **do** Remove the event with smallest y-coordinate from Q.
4. **if** the event is a site event, occurring at site p_i

5. **then** HANDLESITEEVENT(p_i)

6. **else** HANDLECIRCLEEVENT(Y), where Y is the leaf of T representing the arc that will disappear.

…

Refer to the Appendix B about the detailed process of this algorithm. The algorithm runs in *O(nlogn)* time and it uses *O(n)* storage. The main part of the implementation codes for the above algorithm, which is adapted from Steven Fortune's [18] program, is given in the following code.

```
voronoi(triangulate, nextsite)
{…
newsite = ( *nextsite )();
while(1)
{
    if(!PQempty()) newintstar = PQ_min();
    if (newsite != (struct Site *)NULL  && (PQempty()
            || newsite -> coord.y < newintstar.y
            || (newsite->coord.y == newintstar.y && newsite->coord.x <
newintstar.x)))
    {// new site is smallest
            out_site(newsite);
            lbnd = ELleftbnd(&(newsite->coord));
            rbnd = ELright(lbnd);
            bot = rightreg(lbnd);
            e = bisect(bot, newsite);
            bisector = HEcreate(e, le);
            ELinsert(lbnd, bisector);
            if ((p = intersect(lbnd, bisector)) != (struct Site *) NULL)
```

```
        {
                PQdelete(lbnd);
                PQinsert(lbnd, p, dist(p,newsite));
        };
        lbnd = bisector;
        bisector = HEcreate(e, re);
        ELinsert(lbnd, bisector);
        if ((p = intersect(bisector, rbnd)) != (struct Site *) NULL)
        {
                PQinsert(bisector, p, dist(p,newsite));
        };
        newsite = (*nextsite)();
}else if (!PQempty())   // intersection is smallest
{
        lbnd = PQextractmin();
        llbnd = ELleft(lbnd);
        rbnd = ELright(lbnd);
        rrbnd = ELright(rbnd);
        bot = leftreg(lbnd);
        top = rightreg(rbnd);
        out_triple(bot, top, rightreg(lbnd));
        v = lbnd->vertex;
        makevertex(v);
        endpoint(lbnd->ELedge,lbnd->ELpm,v);
        endpoint(rbnd->ELedge,rbnd->ELpm,v);
        ELdelete(lbnd);
        PQdelete(rbnd);
        ELdelete(rbnd);
        pm = le;
        if (bot->coord.y > top->coord.y)
        {
        temp = bot; bot = top; top = temp; pm = re;}
        e = bisect(bot, top);
        bisector = HEcreate(e, pm);
```

```
            ELinsert(llbnd, bisector);
            endpoint(e, re-pm, v);
            deref(v);
            if((p = intersect(llbnd, bisector)) != (struct Site *) NULL)
            {
                    PQdelete(llbnd);
                    PQinsert(llbnd, p, dist(p,bot));
            };
            if ((p = intersect(bisector, rrbnd)) != (struct Site *) NULL)
            {       PQinsert(bisector, p, dist(p,bot));     };
    }else break;
};
for(lbnd=ELright(ELleftend); lbnd != ELrightend; lbnd=ELright(lbnd))
    {
            e = lbnd -> ELedge;
            out_ep(e);
};
}
```

3.4.4 File Format Converting

After computing the mesh for the solid models that are constructed by ACIS modeling engine and saved in SAT file format, the mesh files may be passed to the file format converter to create the file accessible by commercial FEM software tool to perform simulation tasks. The file format converter can also be used to convert the mesh generated by the commercial software package ANSYS to INP file format as the input file for FEA simulation.

3.5 Visualization of Simulation Results

3.5.1 Data Streaming for Co-Modeling and Co-Simulation

The product development activities could result in transmission of massive engineering geometric and simulation data. Despite the continuous improvements in computing hardware performance and network bandwidth, the visualization and analysis of these large-scale data sets remains a challenging task for researchers. In this study, a client-server software architecture is presented to support real-time collaborative design and simulation tasks for product development purposes. The 3D objects generated by high-performance server are transmitted to visualization clients using a highly compressed file format, HOOPS Stream Files (HSF). Based on data streaming technology, the system supports 3D interactive exploration of the complicated geometric models and simulation results. Synchronous and asynchronous forms of collaboration such as real-time co-modeling, simultaneous view and mark-up and CAE analysis interpretation are also supported.

As discussed in the previous chapter, the implementation of CoCADE consists of three major components: client, server and data depository. The client interface provides functionalities such as 3D visualization, communication and interaction. The multiple clients are run on conventional computers while the server is run on some high-performance computer to fit the computing resource demand of engineering simulation. The server supports collaborative design coordination, session control, 3D simulation data generation and streaming data delivering. A file system is used as data depository for geometric models (in SAT or HSF format) and simulation results. Figure 3-14 shows

the implementation structure of data streaming for collaborative modeling and simulation.

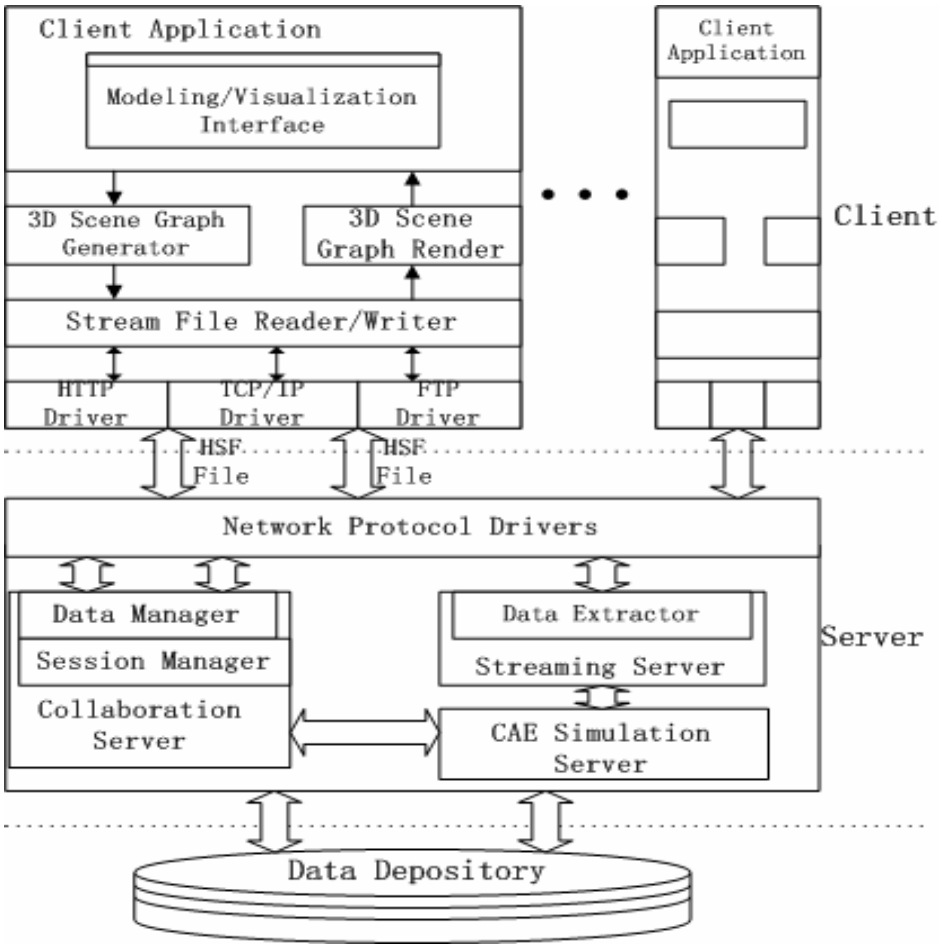


Figure 3-14 Implementation Structure of Data Streaming for Co-Modeling and Co-Simulation

1. HSF data structure and 3D streaming data creation

In the data structure of HSF, the geometric data and properties are stored as a hierarchical tree. The root of the tree is placed at the top. Each node in the tree is called a "segment" containing geometry and attributes. The segment can have its own sub-

segments. When exported, the HSF files are organized into three main sections, including file header, data block and file termination. It is a highly compact file format for data to be efficiently transmitted over Internet/intranet. Figure 3-15 shows the comparison between HSF and other mainstream graphical formats.

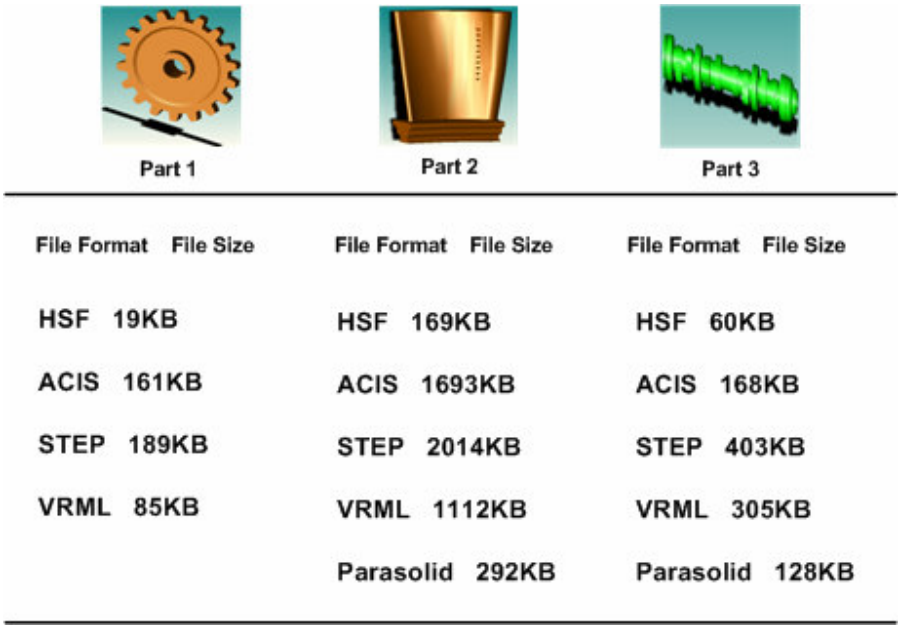


Figure 3-15 Comparisons Between HSF and other Mainstream Graphical Formats

2. Streaming Process

The process of 3D scene graph streaming consists of two important steps: server export phase and client import phase.

During a server export phase, the 3D streaming server generates data buffers that contain the encoded, highly compressed, binary HSF files, which represent the 3D scene graph objects or customized user data. These streaming files can be stored locally or sent to a remote client as part of a message. Both HTTP and TCP/IP connections are supported in the system. Similarly, during a client import phase, sequential 3D scene

graph, which is to be parsed and inserted into appropriate data structures, are retrieved from data buffers or remote server.

3. Connection Establishment and Session Control

Remote distributed clients send request commands to server using TCP/IP protocol in intranet or HTTP protocol in Internet. The server validates the client request before granting a connection to be established.

After validation, a design session is set up as an environment for collaborative design and 3D visualization. In the system, a Centralized Session Management mechanism (CSM) [19] is developed to ensure effective session control. Streaming data transfer is bound by a session. The efficiency of large-amount of data transfer depends on available bandwidth and client PCs performance.

4. Collaborative Visualization and Co-modeling

After the connection is established, the 3D scene graph is retrieved from server and is displayed as soon as the clients are receiving it. It allows the clients to quickly obtain the visual feedback and interact with the 3D objects while it is still being transmitted. A public data depository is deployed to store all versions of data. Only one client in a design session owns the initial data for editing and analysis. A simplified copy of design data is passed to each client in current session for collaborative 3D visualization. Only the client who holds the control token has access to modify the design data.

3.5.2 Visualization of Simulation Results

The goal of the visualization module in CoCADE is to provide a powerful CAE pre-processing and post-processing tool, which embraces fast and high quality graphics, user friendly interface and smart functionality. The post-processing tool supports various functions for the three-dimensional and two-dimensional visualization of results from various solvers.

The supported simulation results may include: node and element results, vector values, mesh generation results, Voronoi diagram results, and other formats of complex FEA simulation results. In this section, the implementation of visualization of simulation results is discussed with emphasis on the use of data streaming technology in CoCADE to support co-visualization.

1. Data structure of simulation results

The scientific representation of the simulation results requires an effective data structure to read in the data and store it in the memory. Of concern are the node class and the element class because the element is the basic unit in FEA process and the node is used to construct the element. Moreover, computation results are usually reflected on the nodes of the product design such as vectors starting from the nodes to show the magnetic flux distribution. Therefore, two classes are designed to hold the information of simulation results, as shown in the source code in the next page.

```
struct element_info{
                    int ele_index;
                    int material_num;
                    char ele_type[8];
                    int ele_connectivity[4]; //Each element is formed by 4 nodes
};


struct node_info{
                    float node_coordinate[3];//each node has X, Y, Z coordinate

                    float M_coor[3];

                    float divM;
                    float u1;
                    float u2;
                    float u;

                    float H_coor[3];
                    float Hex_coor[3];
};
```

2. Dynamic memory allocation

The memory allocated to hold the information of simulation results can be extremely huge that improper handling of memory allocation could cause the program to crash. Therefore, the memory management is critical.

There are two types of dynamic memory allocation widely used throughout the implementation of visualization of simulation results: dynamic memory allocation of a certain class (e.g., the node class or the element class) and dynamic memory allocation of two-dimensional arrays. For the dynamic memory allocation of a certain class, the C++

STL *vector* class is used. Any dynamic class that is explicitly allocated should be explicitly de-allocated. The advantage of this class is that it automatically cleans up the memory it needed to allocate its own data.

The relatively more difficult part is how to dynamically allocate two-dimensional arrays that hold the node information such as their coordinates. The conventional C++ static memory allocation fails to solve this problem. The static method requires the programmer to decide the amount of memory for the array in programming time instead of run time. The pre-allocated memory has its maximum limit depending on different platforms. Therefore, to handle this problem, a class called *Dynamic2DArray* is specially devised. Another reason could be that many APIs of the commercial visualization toolkits required the node information to be passed by reference in a two-dimensional array. The implementation of class *Dynamic2DArray* is shown in the code below.

```
#pragma once
#ifndef DYNAMIC_2D_ARRAY_H_HEADER_GUARD_
#define DYNAMIC_2D_ARRAY_H_HEADER_GUARD_
#include <cstdlib>
#include <vector>
template <class T>

class Dynamic2DArray
{
private:
    Dynamic2DArray& operator=(const Dynamic2DArray&);
    const int m_row;
    const int m_col;
    T* m_data;
public:
    Dynamic2DArray(void);
```

```
    const T* DataPtr() const { return m_data; };

        Dynamic2DArray(int row, int col) : m_row(row),
                                           m_col(col),
                                m_data((row!=0&&col!=0)?new T[row*col]:NULL){};

        Dynamic2DArray(const Dynamic2DArray&src) : m_row(src.m_row),
                                                   m_col(src.m_col),
m_data((src.m_row!=0&&src.m_col!=0)?new T[src.m_row*src.m_col]:NULL)
    {
         for(int r=0; r<m_row; ++r)
            for(int c=0;c<m_col;++c)
                    (*this)[r][c] = src[r][c]
    };
    ~Dynamic2DArray(){ if(m_data) delete []m_data; }

    inline T* operator[](int i) {
        return (m_data + (m_col*i));
        }
    inline T const*const operator[](int i) const
         {
        return (m_data + (m_col*i));
        }
};

#endif //!DYNAMIC_2D_ARRAY_H_HEADER_GUARD_
```

3.6 Summary

In this chapter, the graphical user interface and geometry processing in CoCADE are discussed. The user interface is characterized by various features to support modeling functionalities and simulation results visualization. After geometric model construction and assignment of geometrical parameters and properties, the design models are further processed to generate irregular grain shapes to emulate magnetic recording media. The goal is to prepare the geometric models suitable for engineering numerical solution by FEA.

4. Case Study Using CoCADE

This chapter presents the case study using CoCADE to demonstrate the use of CoCADE framework for co-modeling and co-simulation in collaborative product development. In the first section, a brief introduction of the product development process is given. Secondly, a discussion of the collaborative modeling functions in CoCADE is given. Finally, a case study about the design and analysis of magnetic recording media is presented.

4.1 Introduction to Product Development

Product development is usually a complicated and iterative process of developing new products to meet customers' needs. This process may involve a group of designers from multiple disciplines and different locations. The task of developing high-tech products is difficult, time-consuming and costly. They often evolve over time through countless hours of research, analysis, design studies, engineering and prototyping efforts, and finally, testing, modification, and re-testing until the design is perfected.

The new product development process may vary among companies. Regardless of organizational differences, a methodical development effort with well-defined product specifications and project goals is a must. Figure 4.1 illustrates a generic product development process.

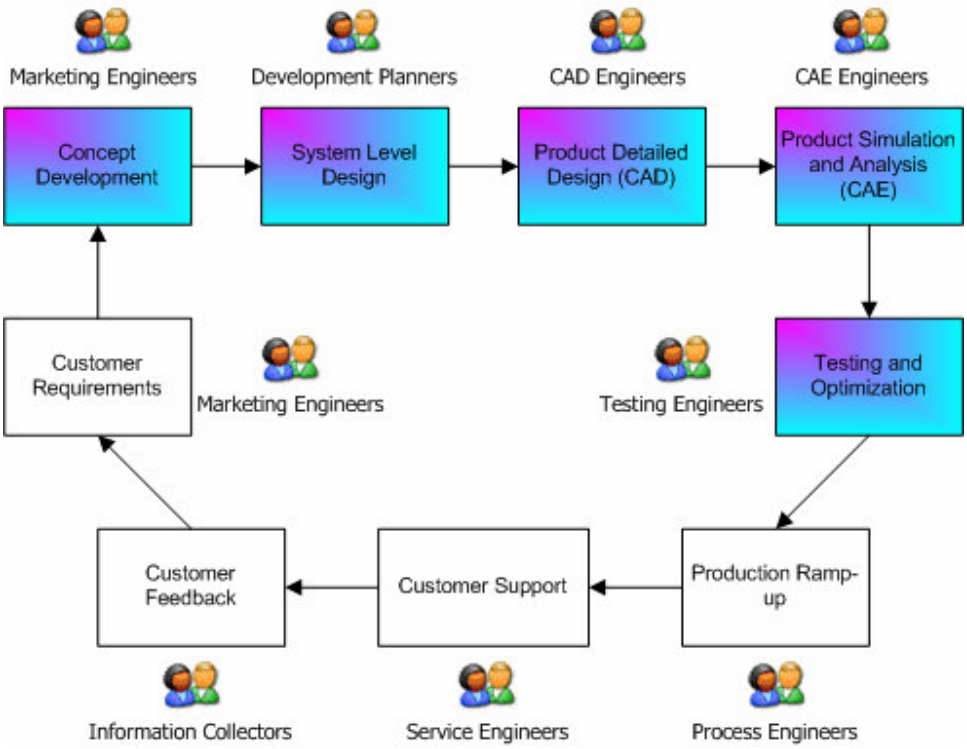


Figure 4-1 A Generic Product Development Process

The above figure shows the generic product development process with each block representing a stage. The highlighted blocks are of particular concern to this study. Good concept development is crucial. This stage provides the foundation for the development effort. In the next stage, the system-level design is to be carried out to define primary functional sub-systems and how these systems are arranged to work as a unit. In the detailed product design (CAD) stage, the digitized form of the product is created, followed by the simulation and analysis (CAE) stage where the product is assigned various properties, and simulation activities carried out to assess the product's performance. Before the product is sent for manufacturing, it is tested and refinement applied if necessary.

From the above discussion, it is obvious that solving the product design specification conflicts and meeting the customers' requirements in the early design stage is important, resulting in the reduction of the development cycle and saving production costs.

4.2 Collaborative Modeling Function in CoCADE

This section illustrates a scenario where designers from different locations and research backgrounds conduct geometric modeling activities in a distributed and collaborative modeling environment built with CoCADE. When conducting product co-modeling activities in such an environment, the following requirements should be met:

(1) The system must have a 3D graphical user interface that supports geometric model creation and basic solid modeling operation like Boolean operations and sweeping, which are considered in the development of CoCADE. The interface should support either mouse click or dialog-based operation or both. Currently, the popular geometric modeling engines include ACIS, ParaSolid and CASCADE. This study uses ACIS as its geometric engine, which is programmed in C++ language.

(2) The system must have a 3D visualization system that supports geometric model visualization and object selection. In CoCADE, these features are available. Meanwhile, the display functions include orbiting, zooming, and panning. Both select by click and by window are supported in the selection function. The selection level includes vertex, edge, face and body. Rich rendering modes such

as wire-frame, Gouraud shaded mode and hidden line mode are all supported in the visualization system.

(3) The clients must be able to store the model's geometry and topology in industry-recognized standard file formats so that these models can be exchanged with other commercial CAD software packages under heterogeneous platforms. In CoCADE, the standard file formats like SAT, HSF, SAB are supported. The Hoops development toolkit provides the bridge between mapping of SAT file and HSF file format, facilitating the file transmitting process.

(4) Each client must be able to collaborate with the others. The modeling operation by a client must be able to be reflected on the screen of the other clients immediately. In CoCADE, a master client is used to coordinate the collaborative session and central session control mechanism is devised to ensure that no conflicts between clients occur.

Based on the above requirements, a scenario where designers from different locations and backgrounds conduct a part modeling activity is developed and illustrated in Figure 4.2. In this scenario, the designers can conduct collaborative modeling within the Local Area Network (LAN) or across the Internet. In the case of collaborating within LAN, TCP/IP protocol is used to boost communication efficiency. In the case of collaborating across Internet, the HTTP protocol is used.

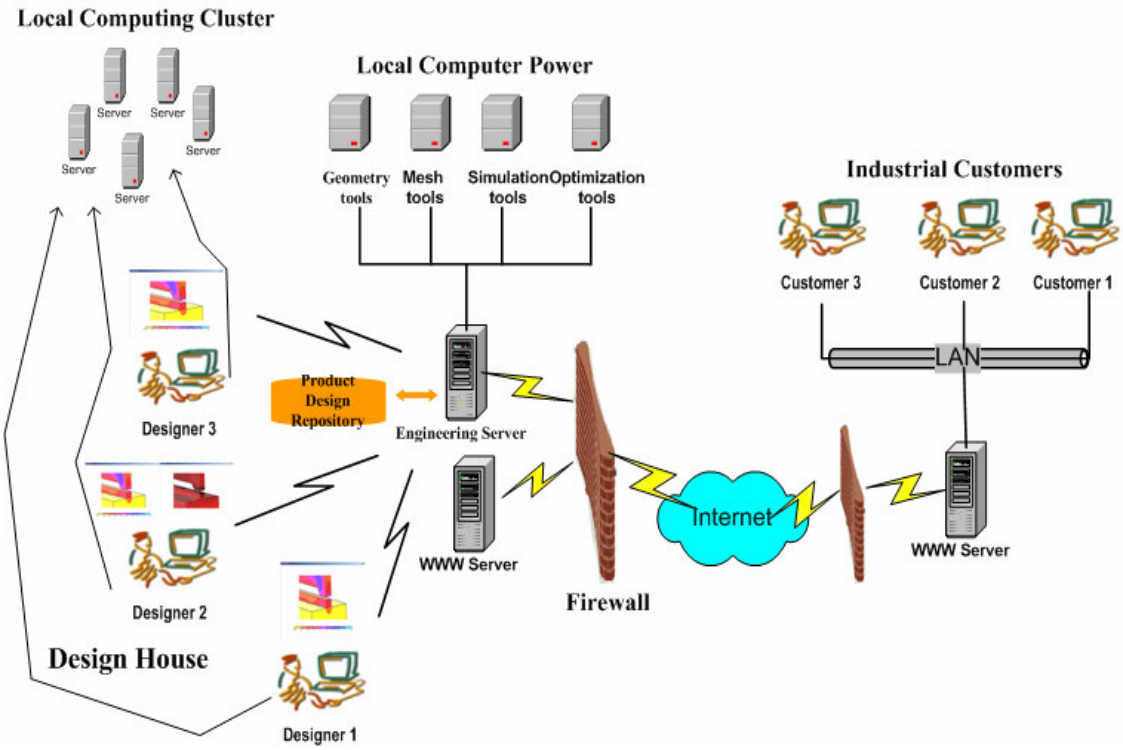


Figure 4-2 The General Diagram of the Co-Modeling and Co-Simulation Environment

In particular, a Boolean operation during the modeling process is highlighted in the following diagram to show the typical geometric modeling procedure in a collaborative modeling environment. Figure 4.3 is an example of subtraction operation being carried out in a design process. The design first creates a solid hub model leveraging on the primitives creation functionalities supported in CoCADE. In the next stage, a cylinder (highlighted in yellow color) is added to the hub model. To carry out the subtraction operation, the designer must select a "blank" (the hub) object and a "tool" object (the cylinder). The "blank" object will be returned as the operation result.

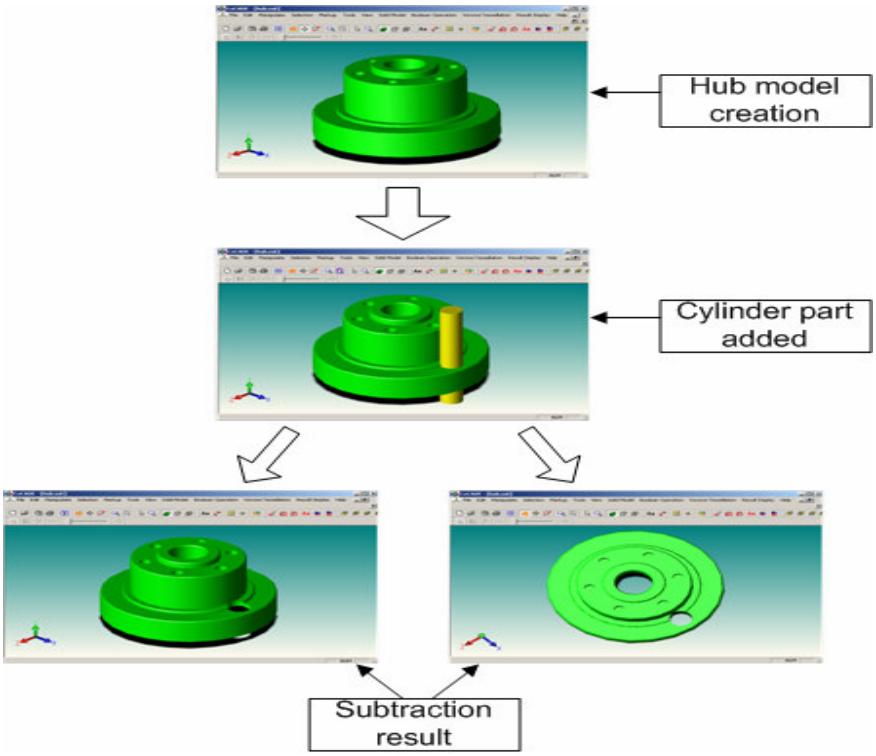


Figure 4-3 Boolean Operation Example One: Subtraction

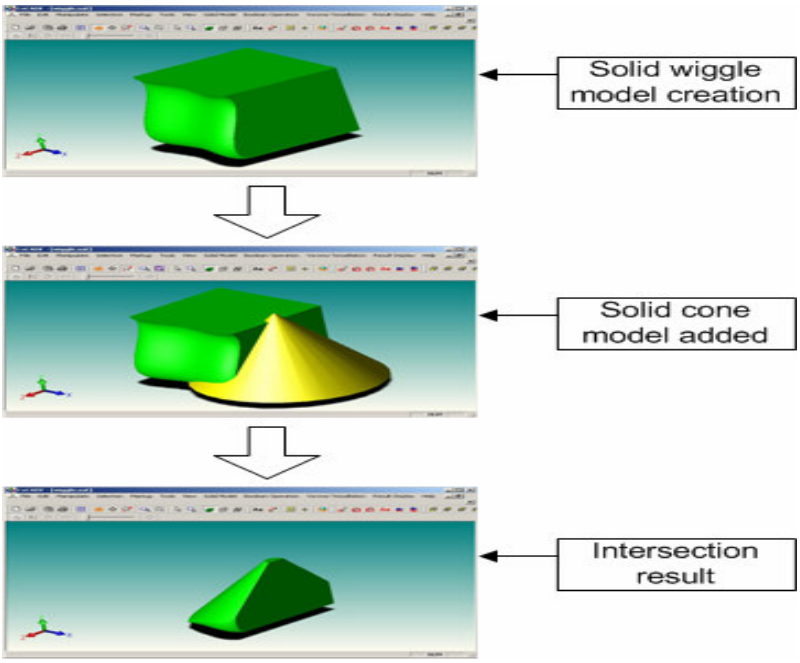


Figure 4-4 Boolean Operation Example Two: Intersection

Another example is shown in Figure 4.4. Here the Boolean operation is intersection. The procedure is similar with the previous subtraction operation. Care must be taken when selecting the object because the selection sequence may influence the final returned result, depending on which operation is executed.

4.3 Design and Analysis of Magnetic Recording Media

4.3.1 Introduction

It is very essential to consider the realistic physical microstructure when building the mathematical model for accurately predicting the magnetic properties of recording media. The Voronoi model [20]-[25] is used in this research to generate grain size distribution as it adequately represents the grain structure of the magnetic media. The grain will build up at a finite number of initial nucleation sites randomly and grows until the two neighboring grains touch each other.

4.3.2 Design and Processing of Magnetic Recording Media

Much research into the structural details of the magnetic patterns which strore information in magnetic recording systems has been based on micro-magnetic models of magnetic recording media [26]-[29]. Some extremely irregular and unrealistic shapes may be generated based on a uniform triangular, hexagonal or square seed lattice. The finite element models of the grain structure obtained form Voronoi construction and subsequent meshing of the polyhedral region are proposed to yield a realistic microstructure of recording media [30].

As shown in Figure 4.5, the process begins from a randomly located seed points from which the grains are assumed to grow with constant velocity in each direction. Thereafter, the grains are surrounded by the Voronoi cells (Figure 4.5 b). According to the algorithm of Voronoi construction, the Voronoi cell of seed point *i* contains all points of space which are closer to seed point *i* than to any other seed point. It is expected to subdivide the media model into cubic cells and choose one seed point within each cell at random to avoid highly irregular shaped grains. Once the polyhedral grain structure is obtained, the grains are further meshed into finite elements using tetrahedral elements [12].

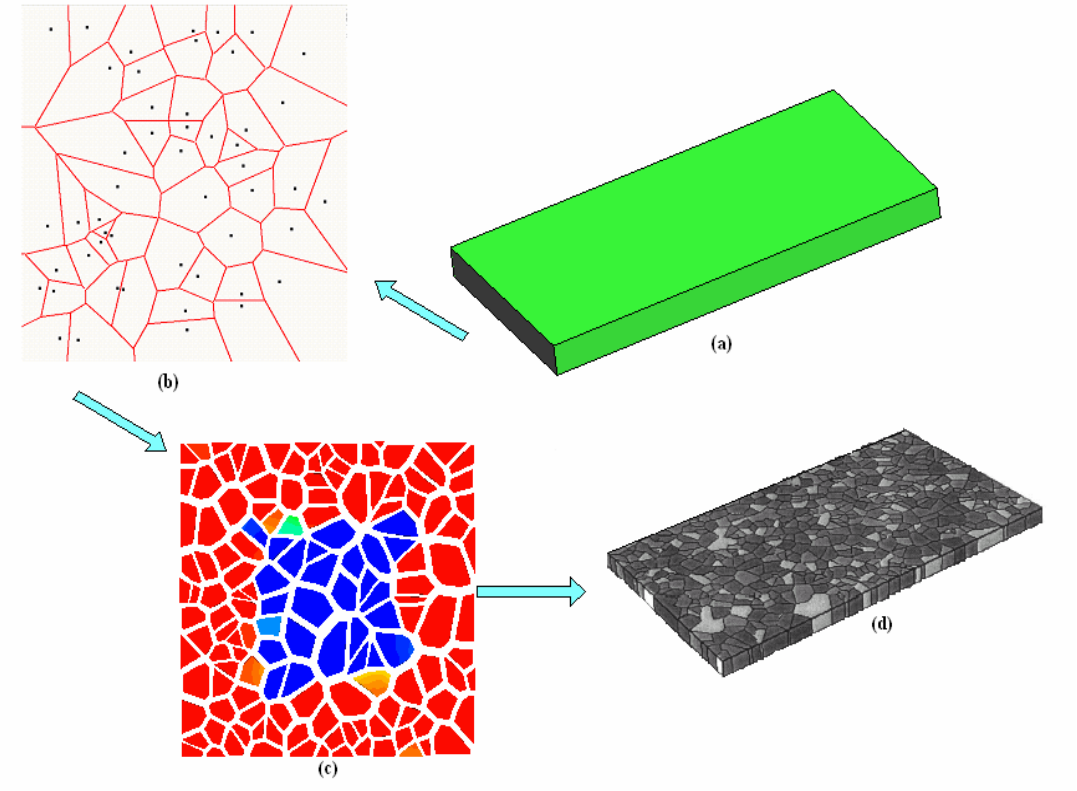


Figure 4-5 Magnetic Recording Media Finite Element Model with Voronoi Algorithm

For our case study, firstly a block representing the magnetic recording media is constructed. Thereafter, one of the surfaces of the block is extracted from the block. This surface is then sent to compute seeds. The result of seeds computation is shown in Figure 4.6 (b). Finally, the 2D Voronoi diagrams of the media surface are to be computed from the seeds and then extruded in a certain direction to construct the final media model. Figure 4.6 shows an example of the case study before the Voronoi diagrams are extruded in a certain direction.

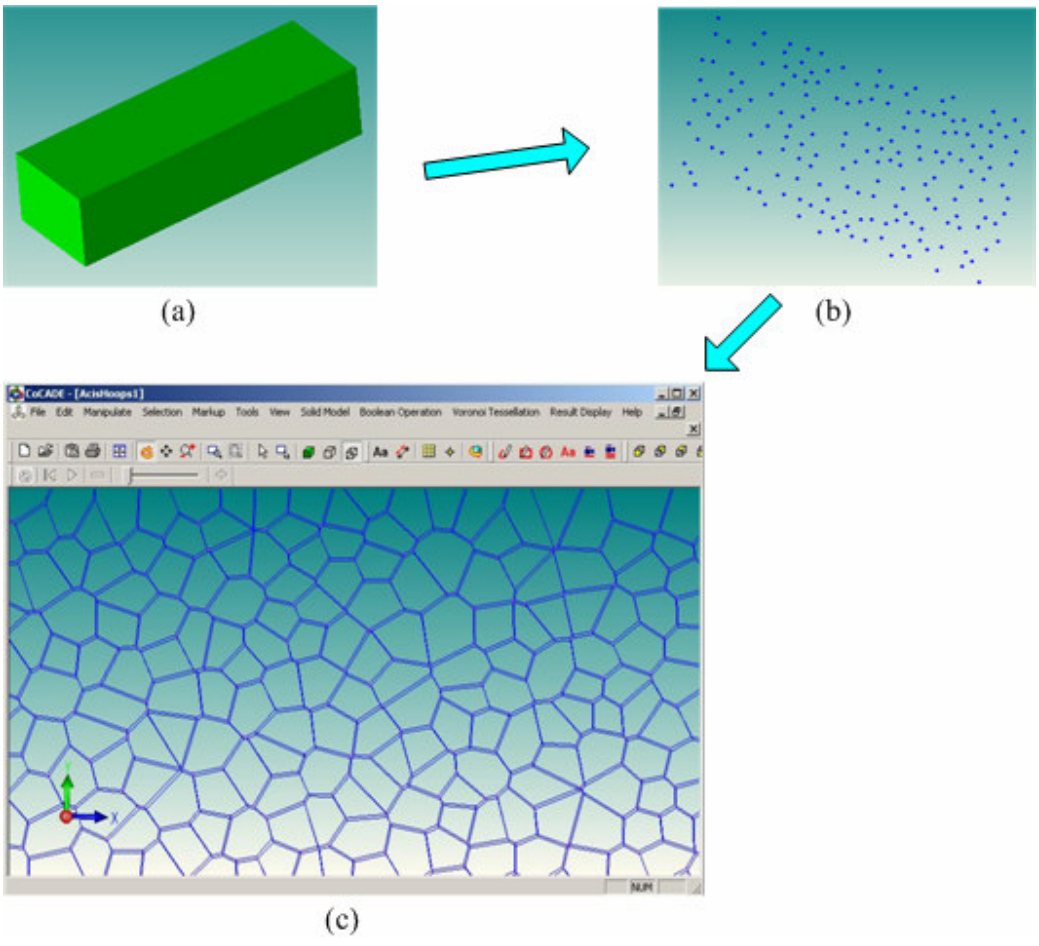


Figure 4-6 An Example of Magnetic Recoding Media Design and Processing

Figure 4.7 shows the result of extruding the Voronoi diagrams in a certain direction of the space. Each of the grains can be viewed as a prism.

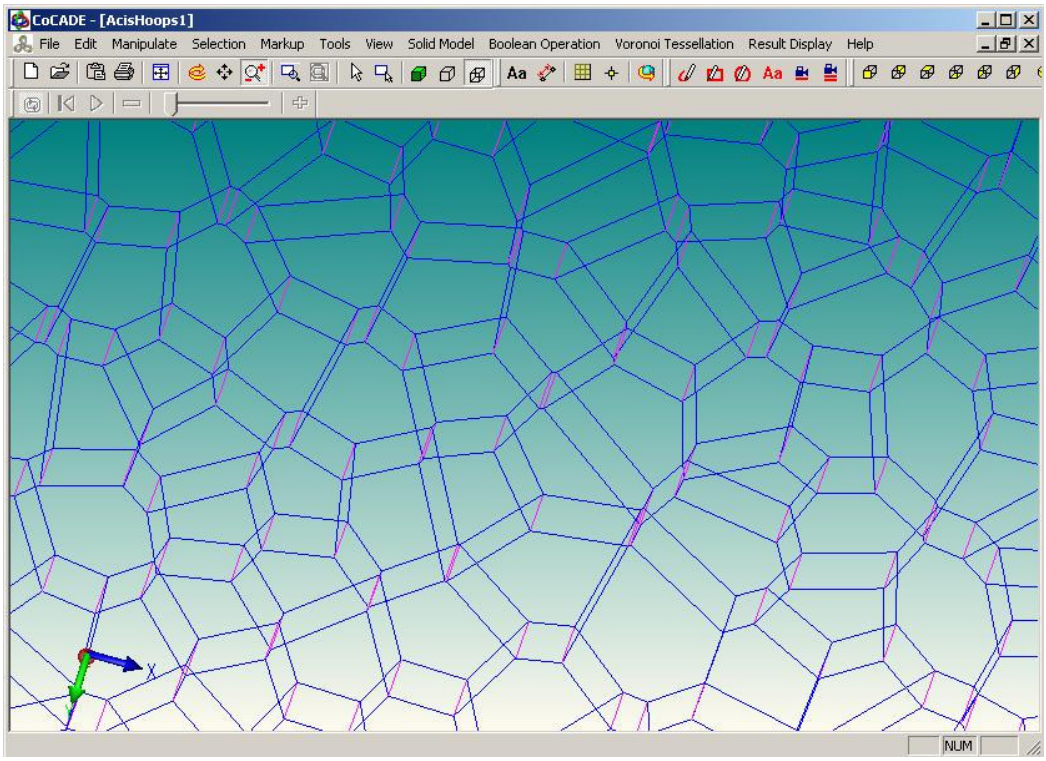


Figure 4-7 An Example of Extrduing the Voronoi Diagrams in a Certain Direction

After each Voronoi diagram is built, the magnetic recording media model is sent for mesh generation. Figure 4.8 shows the mesh generation result of a single hexahedral-shape grain. The elements in the meshed grain are tetrahedral structures.

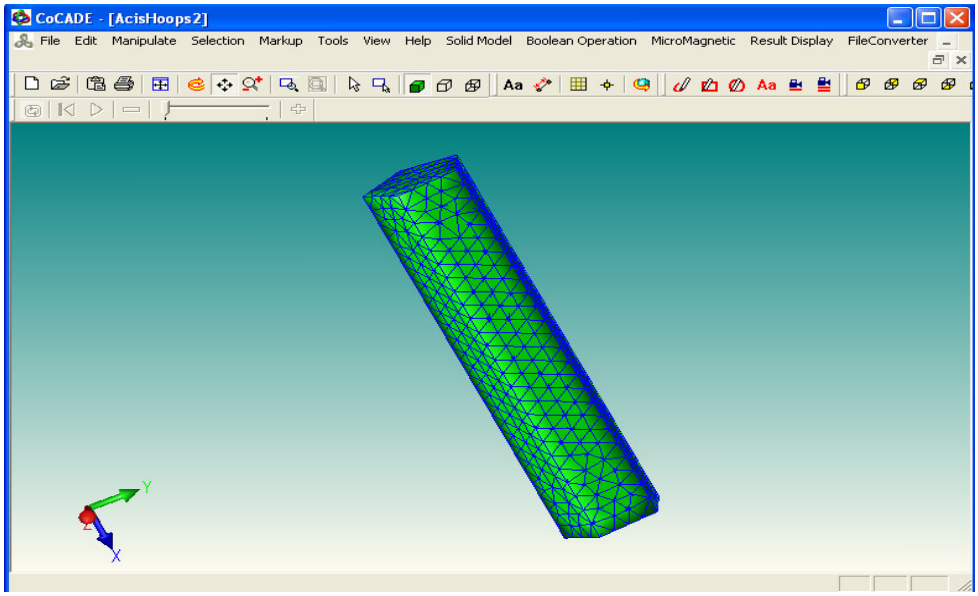


Figure 4-8 An Example of Tetrahedral Mesh Generation

4.3.3 Visualization and Analysis of Simulation Results

Till now, the model of magnetic recoding media is ready to be fed into simulation solvers. The simulation task can be accomplished by leveraging on commercial CAE software tools using Finite Element Analysis (FEA). A detailed discussion of the FEA method is beyond the scope of this study. The simulation results are presented in a 3D viewer to the designers for visualization and analysis. As shown in Figure 4.9, the vectors show the direction of the magnetization in the internal part of the grain.

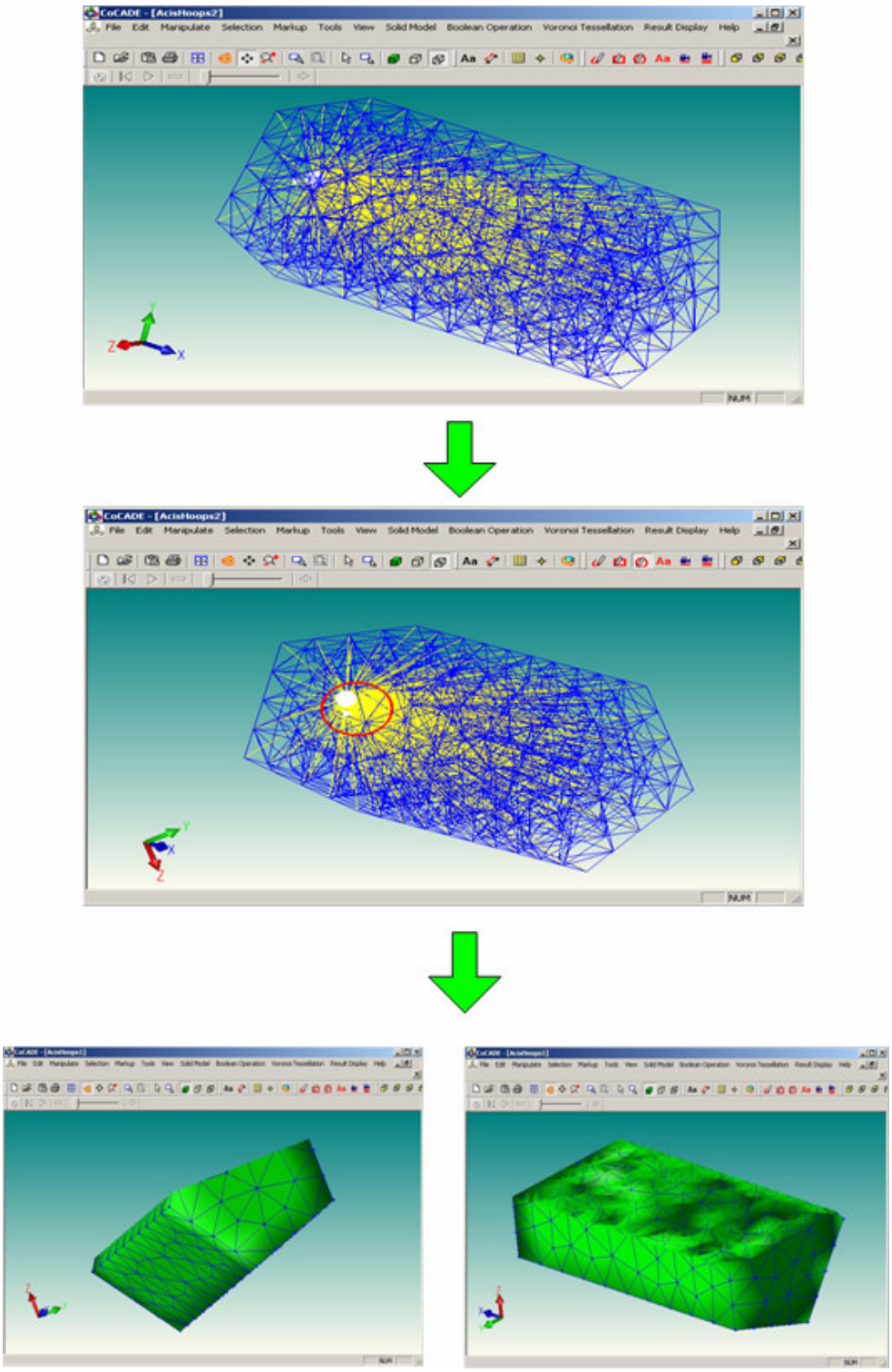


Figure 4-9 Visualization of Simulation Results

Figure 4.10 shows another example of visualization of simulation results where the designer can cut into the interior part of the model and analyze the results. The implementation of a cutting plane tool allows the user to take a cut-away view of the product. As such the detailed distribution of the flux vectors inside the model can be visualized.

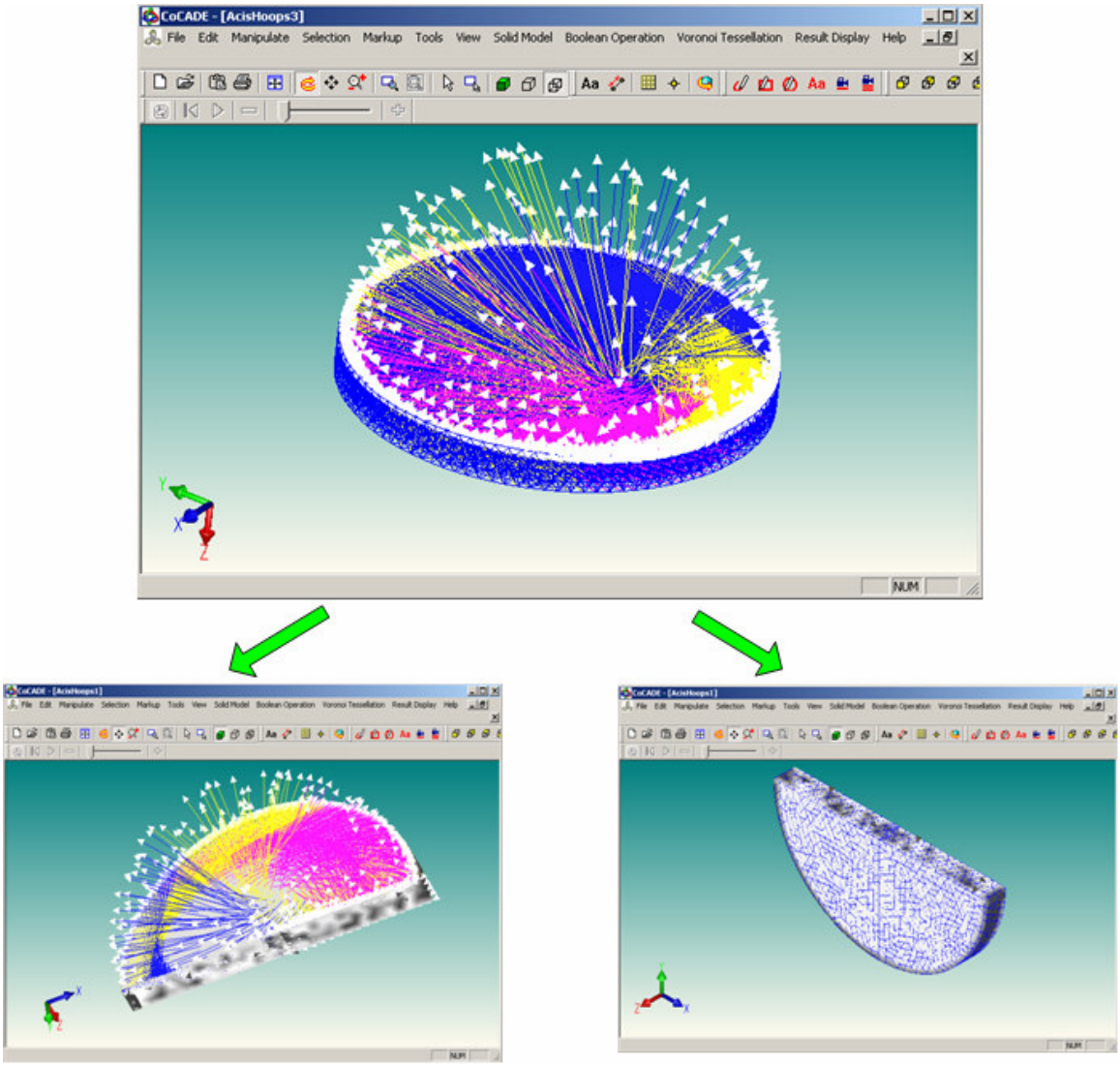


Figure 4-10 An Example of Simulation Results Visualization with Cutting Plane

5. Conclusions

The state-of-the-art innovation in computer technologies has enabled product development activities to be undertaken across distributed teams in various organizations. A team of engineers with different expertise and backgrounds collaborates to achieve a common goal: the development of a product that can be used and sold. In this context the collaboration must be organized to allow a better communication among designers [31]. Given this development, collaborative modeling and simulation systems are emerging to support geographically separated engineers to conduct design activities synchronously, asynchronously and cooperatively.

In this thesis, a framework called CoCADE is introduced to support collaborative product development activities among geographically dispersed organizations. The architecture of CoCADE has been described from different perspectives such as user case view, data streaming view and static class view. This architecture adopts a three-tiered client-server structure. The client side provides the essential geometric modeling functionalities as well as interactive visualization tools for analysis of sophisticated geometric models and large-scale simulation results. The server side offers the functions of collaborative session management, multi-client communication mechanism, engineering simulation and optimization.

The functionalities of the client interface as well as the methodology of the geometry processing have been fully discussed and implemented. The proposed

distribution of modeling functionalities is reasonably well balanced while leveraging on the powerful CAD workstation. The powerful features of geometric processing and simulation results visualization characterize the client side software while the server side offers the functionalities of session management, simulation and data coordination.

The effectiveness of the system is demonstrated through the development of two applications: a collaborative modeling application and a CAE application concerned with design and analysis of magnetic recording media.

In conclusion, this research focuses on geometric modeling in distributed environment to support collaborative CAD and CAE activities. It is a research challenge to implement a collaborative product development system based on .NET technology [34] for distributed computing and HOOPS streaming technology for 3D scientific data presentation. This research attempts to meet such technological challenge. Moreover, this research has special reference to the product development process related to magnetic recording devices. The design and realization of such product generally involve the effective collaboration among key component manufacturers in data storage industry. The implementation of Voronoi diagram generation emulates the realistic irregular grain structure of magnetic recoding media, avoiding the shortcoming of unphysical anisotropic properties yielded by regular microstructure. Finally, the visualization of large-scale simulation results using data streaming technology provides a useful tool for engineering analysis and optimization.

Effective methods were presented in this research to further develop co-modeling and co-simulation in collaborative product development.

References

[1] Hoffmann, C.M. Geometric and Solid Modeling. Morgan Kaufmann Publishers Inc, 1989.

[2] Tanenbaum, A.S. and M.V. Steen. Distributed Systems, Principles and Paradigms. pp. 494-542, Prentice Hall, 2002.

[3] Regli, W.C. Internet-enabled Computer Aided Design. IEEE Internet Computing1(1), pp. 39 – 50, 1997.

[4] Sommer, K.O., C. Ernst and T. Ertl. Remote 3D Visualization using Image-Streaming Techniques. International Symposium on Intelligent Multimedia and Distance Education, 1999.

[5] Mahovsky, J. and L. Benedicenti. An Architecture for Java-based Real-time Distributed Visualization. IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 4, 2003.

[6] Oven, S. http://www.andrew.cmu.edu/user/sowen/mintro.html. Mesh Research Corner.

[7] Rogers, J.D. Modeling HFM Contractors for Systems with Interfacial Reactions using Voronoi Tessellations: Fundamental Rigorous Models and Simpler. Dissertation, pp.387, 1996.

[8] Kanai, Y. and S. H. Charap. Simulation of Magnetic Aftereffect in Particulate Recording Media. IEEE Trans. Magn., Vol. 27, No. 6, pp.4972-4974, 1990.

[9] Alex, M. and D. Wachenschwanz. Thermal Effects and Recording Performance at High Recording Densities. IEEE Trans. Magn., in press.

[10] Sharrock, M.P. Time-Dependent Magnetic Phenomena and Particle-Size Effects in Recording Media. IEEE Trans. Magn., Vol. 26, No. 1, pp. 193.

[11] Berg, M. de, M. Van Kreveld, M. Overmars and O. Schwarzkopf. Comptutational Geometry: Algorithms and Applications, 2nd edition. pp. 2, 147-162, Springer, 2000.

[12] LaBarre, R.E. Computational Geometry Techniques for 2D and 3D Unstructured Mesh Generation with Application to the Solution of Divergence from Partial Differential Equations. Ph. D. dissertation, University of Connecticut, 1992.

[13] Okabe, A., B. Boots and K. Sugihara. Spatial Tessellations Concepts and Applications of Voronoi Diagram. John Wiley & Sons.

[14] Preparata, F.P. and M.I. Shamos. Computational Geometry: An Introduction, New York: Springer-Verlag.

[15] Armstrong, C.G., R.M. McKeag, H. Ou and M.A. Price. Geometric Processing for Analysis. Geometric Modeling and Processing, 10 – 12 April, 2000, Hong Kong, China.

[16] Gunzburger, M. Centroidal Voronoi Tessellations. CCS-2/CNLS Seminar, Los Alamos National Lab, 14 May, 2003.

[17] Fortune, S. A Sweep-line Algorithm for Voronoi Diagrams. Algorithmica 2, pp. 153 – 174.

[18] S. Fortune, http://cm.bell-labs.com/who/sjf/

[19] Sun, D.W., L.W. Ruan, Z.J. Liu, J.M. Zhao, W.F. Lu and X.G. Ming. Concurrency in a Distributed Collaborative CAD/CAE Environment. The

11th ISPE International Conference on Concurrent Engineering: Research and Applications, Jul. 2004.

[20] Zhen, J., M. Doerner and H.N. Bertram. A Growth Model for Polycrystalline Thin Film Media: Comparison of Bicrystal and Unicrystal Grain Structures. IEEE Trans. Magn., Vol. 36, pp. 2294-2296, 2000.

[21] Miles, J.J., M. Wdowin, J. Oakley and B.K. Middleton. The effect of cluster size on thin film media noise. IEEE Trans. Magn., Vol. 31, pp.1013-1023, 1995.

[22] Porter, D.G. et al. Irregular Grain Structure in Micromagnetic Simulation. Journal of Applied Physics, Vol. 79, pp. 4695-4697, 1996.

[23] Xue, J. and R.H. Victoria. Micromagnetic Calculation for Superlattice Magnetic Recording Media. Journal of Applied Physics, Vol. 87, pp. 6361-6363, 2000.

[24] Walmsley, N.S. et al. Comparison of the Effects of Regular and Irregular Physical Structure on the Magnetic Microstructure of Longitudinal Thin Films. J. Magn. Magn. Mater., Vol. 170, pp. 81-94, 1997.

[25] Schrefl, T., J. Filder, and Kronmüller. Remanence and Coercivity in Isotropic Manocrystalline Permanent Magnets. Phys. Rev. B., Vol. 49, pp. 6100-6110, 1994.

[26] Hughes, G.F. J. Appl. Phys. pp. 54, 5306, 1983.

[27] Zhu, J.G. and H.N. Bertram. J. Appl. Phys. pp. 63, 3248, 1998.

[28] Mansuripur, M. and R. Giles. Comput. Phys. pp. 4, 291, 1991.

[29] Miles, J.J. and B.K. Middleton. J. Magn. Magn. Mater. pp. 95, 99, 1991.

[30] Chantrell, R.W., J. Fidler, T. Schrefl, and M. Wongsam. Micromagnetics: Finite Element Approach. Encyclopedia of Materials: Science and Technology, pp. 5651-5661, 2001.

[31] Porter, D.G., E. Glavinas, P. Dhagat, J.A. O'Sullivan, R. S. Indeck and M. W. Muller. Irregular Grain Structure in Micromagnetic Simulation. Journal of Applied Physics, 1996.

[32] HOOPS Help Document, Tech Soft America.

[33] ACIS Help Document, Spatial Inc. http://www.spatial.com.

[34] Microsoft .NET Developer Center, http://msdn.microsoft.com/netframework/.

Appendix A. Recent Publications

[1] X. K. Gao, Q. H. Liu, L. W. Ruan, and Z. J. Liu, "Sensitivity Analysis and Design Optimization for Head Interconnect", Asia-Pacific Magnetic Recording Conference 2004 (APMRC '04), HD02, 16 - 19 August, 2004, Seoul, Korea.

[2] L. W. Ruan, D. W. Sun, H. H. Long, Z. J. Liu, "Data Streaming for Real-Time Collaborative Design", International Conference on Scientific and Engineering Computation (IC-SEC), 30 June – 02 July, 2004, Singapore.

[3] D. W. Sun, L. W. Ruan, Z. J. Liu, J. M. Zhao, W. F. Lu and X. G. Ming, "Concurrency in a Distributed Collaborative CAD/CAE Environment", the 11th ISPE International Conference on Concurrent Engineering: Research and Applications, 26 – 30 July, 2004, Beijing, P. R. China.

[4] D. W. Sun, X. H. Xiong, L. W. Ruan, Z. J. Liu, J. M. Zhao, Y. S. Wong, "Workflow-driven Collaborative Session Management in Product Lifecycle Management", to be presented on the International Engineering Management Conference (IEMC 2004), 18 – 21 October, 2004, Singapore.

Appendix B. Overall Structure of Sweep Line Algorithm for Voronoi Diagrams[11]

Algorithm VORONOIDIAGRAM(P)

Input. A set $P := \{p_1, ..., p_n\}$ of point sites in the plane

Output. The Voronoi diagram Vor (P) given inside a bounding box in a doubly connected edge list D.

1. Initialize the event queue Q with all site events, initialize an empty status structure T and an empty doubly connected edge list D.
2. **while** Q is not empty
3. **do** Remove the event with smallest y-coordinate from Q.
4. **if** the event is a site event, occurring at site p_i
5. **then** HANDLESITEEVENT(p_i)
6. **else** HANDLECIRCLEEVENT(Y), where Y is the leaf of T representing the arc that will disappear.
7. The internal nodes still present in T correspond to the half-infinite edges of the Voronoi diagram. Compute a bounding box that contains all vertices of the Voronoi diagram in its interior, and attach the half-infinite edges to the bounding bsx by updating the doubly connected edge list appropriately.
8. Traverse the half-edges of the doubly connected edge list to add the cell records and the pointers to and from them.

The procedures to handle the events are defined as follows.

HNADLESITEEVENT(pi)

1. If T is empty, insert pi into it (so that T consists of a single leaf storing pi) and return. Otherwise, continue with steps 2-5.
2. Search in T for the arc α vertically above pi. If the leaf representing α has a pointer to a circle event in Q, then this circle event is a false alarm and it must be deleted from Q.
3. Replace the leaf of T that represents α with a sub-tree having three leaves. The middle leaf stores the new site pi and the other two leaves store the site pj that was originally stored with α. Store the tuples <pi, pj> and <pj, pi> representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on T if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating V(pi) and V(pj), which will be traced out by the two new breakpoints.
5. Check the triple of consecutives arcs where the new arc for pi is the left arc to see if the breakpoints converge. If so, insert the circle event into Q and add pointers between the node in T and the node in Q. Do the same for the triple where the new arc is the new arc is the right arc.

HANDLECIRCLEEVENT(Y)

1. Delete the leaf Y that represents the disappearing arc α from T. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on T if necessary. Delete all circle events involving α from Q; these

can be found using the pointers from the predecessor and the successor of Y in T. (The circle event where α is the middle arc is currently being handled, and has already been deleted from Q.)

2. Add the center of the circle causing the event as a vertex record to the doubly connected edge list D storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoints of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end that the vertex.
3. Check the new triple of consecutive arcs that has the former lef neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into Q, and set pointers between the new circle event in Q and the corresponding leaf of T. Do the same for the triple where the former right neighbor is the middle arc.