COMPUTING ABOUT PHYSICAL OBJECTS

Chanderjit Bajaj
Wayne R. Dyksen
Christoph M. Hoffmann
Elias N. Houstis
John T. Korb
John R. Rice

CSD-TR-696
July 1987

COMPUTING ABOUT PHYSICAL OBJECTS

Chanderjit Bajaj
Christoph M. Hoffmann
Elias N. Houstis
John T. Korb
John R. Rice

Computer Sciences Department
Purdue University
CSD-TR-696
July 1, 1987

Abstract

This report describes the technical aspects of the project *Computing about Physical Objects*. This project has received equipment and infrastructure support for five years from the National Science Foundation.

Contents

1. COMPUTING ABOUT PHYSICAL OBJECTS

Many questions about physical objects become approachable through computation. Questions such as: How do objects respond to heating? When subjected to forces and torques, how do they move? What happens to them under stress and strain? Several research projects at Purdue develop tools for posing and answering these and similar questions on the computer. Linking these projects are many common problems: For example, how to create, manipulate and archive suitable models of objects; how to structure algorithms to analyze and simulate physical processes; how to exploit and coordinate suitable hardware configurations to cope with the resource requirements of these computations. We are beginning a coordinated research effort to solve these common problems. Future scientific systems must integrate design, manipulation and inspection of objects with simulation of their physical behavior. Joint efforts are needed on many levels.

The design, manipulation and control of physical objects involves complex computational models that are tightly coupled. To create and use such models requires an environment that hides intricate details, that harnesses tremendous complexities, and that is supported by powerful computers and graphics. We will develop tools that integrate design and inspection of computational models of physical objects with simulation and analysis of their behavior and interaction. These tools create and manipulate the geometric shapes, simulate the physical processes, help control the power of multicomputers, and provide a natural, uncluttered environment for the users.

1.1. Overview

We are developing the tools for computing with models of physical objects. In the course of this work, we face problems of how to represent objects by suitable models, how to manipulate and edit these models, how to analyze and simulate the behavior of modeled objects. To see the varied nature of the work, and to gain a first impression of how the projects interact, consider designing a small water cooled piston engine diagram in Figure 1.1.

Imagine we are at a point where the piston and its linkage to the crank shaft have been designed. Now we wish to design the block such that the engine is kept cool, strong, and light. So, the exterior geometry of the block must be designed, the shape and location of the water cooling lines must be determined, and a suitable material for the block must be selected.

Having chosen the shapes, assisted by a geometric modeling interface, the user must solve a system of partial differential equations (PDEs) to analyze the heat flow and stresses associated with this geometry and choice of block material. He may wish to simulate running the engine at different speeds which changes the strength of the heat source and the stresses on the moving parts. Systems of ordinary differential equations (ODEs) describe the acceleration and constraint forces on the piston linkage. With access to sufficient computing power, a user can quickly explore a wide range of shapes and materials preliminary to a refined, optimized final design.

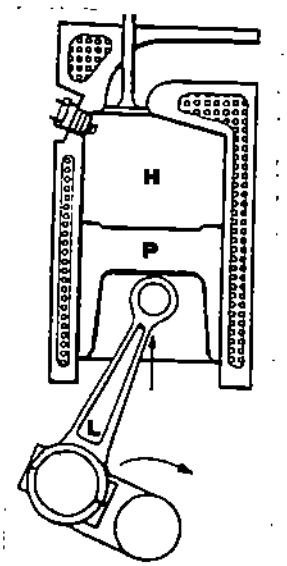


Figure 1.1 Cross section of a piston engine. The source of heat and force is at *H*, the coolant within the block is shown with bubbles. The housing of the linkage *L* to the piston *P* is not shown.

What is involved in creating and simulating this scenario? First, complex object models must be created and coordinated. The geometric modeling project provides tools for this. The models are created through a user interface and require much automatic design support from the system. For example, the geometric design of piston and linkage requires a sophisticated solid modeling system. The ODEs describing the dynamic behavior of the piston linkage can then be derived automatically by the system for the geometry and material composition of the links. Project Newton builds such a system. PDEs model the physical behavior such as heat flow, stresses and strains. They are identified and numerical methods are selected that are well suited to the problem's nature and the desired accuracy. The Mathematical Software Systems project does this. The user should be allowed to specify a sacrifice some accuracy for the sake of speed or economy, the system can then allocate the computation among the available resources to achieve this objective - without detailed intervention by the user. The parallel processing project provides tools for this.

Our research projects are involved in building complex interactive systems. Although the projects differ, many problem domains are shared and the problems of communicating with the user are similar. Geometric modeling is one common focus, as is the coordination of different domain-specific models so that object behavior can be studied comprehensively. All aspects of this work require very high powered workstations and sophisticated graphics backed up by supercomputer power. Because of the scope of our effort, all projects use a very high level approach to software development and integration that must be backed up by powerful computing resources.

1.2. The Central Projects

The relationship of the principal projects is shown in Figure 1.2. We summarize each here.

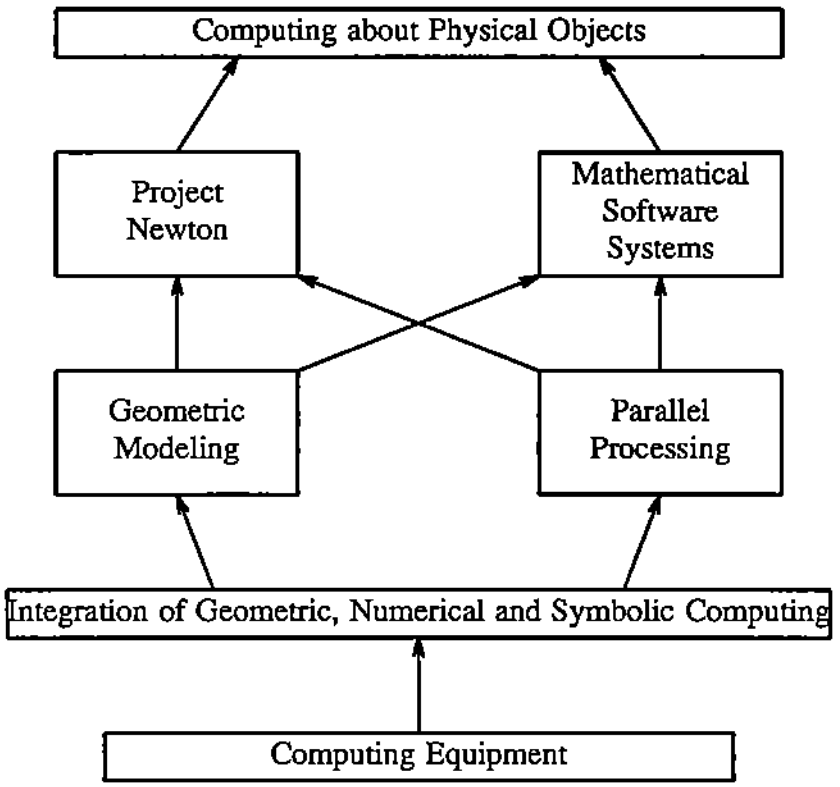


Figure 1.2. The relationship of the central projects.

1.2.A. Project Newton

The goal of this research is to develop a highly modularized and extensible system that duplicates the precise behavior of physical objects from their models. The work is part of a consortium effort involving groups both at Cornell and Purdue University, and should have a major impact on computer-aided design and manufacturing.

Objects represented in this system are complex, ultimately involving thousands of equations and constraining relationships. Objects are composed from many components and are modeled in a spectrum of domains including their geometry, dynamics, and controlled behavior. The project is implementing a first prototype in which the motion equations are derived from geometric models and are integrated numerically. Work is underway that aims at expanding the physical coverage of modeling, developing flexible user interfaces, and adding more intelligence to the automatic capabilities assisting the user in defining and manipulating objects and assemblies. Future work includes an effort to distribute the system over a network of specializing processors, including parallel machines.

1.2.B. Geometric Modeling

Geometric and solid modeling has reached a plateau that cannot be elevated unless a number of basic computational problems in mathematics are solved efficiently. This project focuses on these problems. Its results, as they mature, will impact the other projects significantly.

The research is engaged in extending the geometric coverage of solid modelers to algebraic surfaces of arbitrary degree. Here, questions such as effective parameterization, implicitization, and singularity resolution handling are addressed. Efficient computational procedures are being developed that will eliminate traditional bottlenecks that presently make modeling operations so expensive. The work also explores questions of representing surfaces succinctly, of approximately representing complicated surfaces by simpler ones, of approximating naturally occurring surfaces by mathematically defined ones, and of automatically generating standardized surface areas. It seeks to merge results from algebra, geometry, topology, and approximation theory into effective tools for a higher level of performance in geometric modeling.

1.2.C. Mathematical Software Systems

A high level mathematical software system targeted toward elliptic partial differential equations is already operational. Its principal components are an interactive, mathematically based, graphically oriented interface and a broad range of problem solving modules. This 100,000+ lines of code system is built using various software tools so that it can readily evolve and be enhanced. It is an ideal vehicle to test approaches to future systems for computations about physical objects.

The new work will greatly enhance the quality and performance of the user interface. An expert system project is under way to provide users sophisticated guidance in using tools and problem solving modules. Significant expansion in the domain of applicability will be based on the following work on the algorithmic infrastructure: 1. The rectangular 3D geometry will be extended to general domains, 2. The single domain, single equation operation will be replaced by a multiple domain, multiple equation operation, 3. Time dependent and nonlinear problems will be allowed, 4. More extensive domain mapping and grid adaptation capabilities will be incorporated.

These extensions involve a large number of specific projects, some very algorithmic in nature, e.g., interfacing algebraic geometry with numerical methods and 3D numerical methods. We will rely heavily on a high level software integration approach to gain many of the desired capabilities. For example, a structural engineering software package, an expert system, and MACSYMA will all be integrated, appearing to the user as though they were really part of one system.

1.2.D. Parallel Processing

Computing with physical models requires enormous computing power. This power will come mostly from massive parallelism. Our work concentrates on three of the many aspects of this problem area. First, we will be heavily involved in the algorithmic infrastructure both for numeric and geometric computation. Beyond parallelizing existing numerical methods, we will deeply explore new domain decomposition techniques as a means of effectively using hundreds or thousands of processors for realistic applications. A key ingredient of this work is to study how the numerical methods interact with the geometry of physical objects. Ultimately, this use of parallelism will result in reaching true interactive performance for complex geometric computations.

Second, we will study how to create an intelligent system for resource allocation without significant user input or intervention. We will consider both tightly coupled

computing environments where algorithm specific, synchronous approaches seem to be most promising, and loosely coupled environments where we will concentrate on very general, heuristic approaches to resource allocation. In addition, our intelligent system will handle the closely related problems of scheduling the computation in a heterogeneous computing environment.

Finally, we will monitor performance issues continually. We give high priority to providing reliable data and estimates for assessing the effective utilization that can be obtained from massively parallel machines with thousands of processor nodes.

1.3. Coordinated Research Foci

Our research projects share two central thrusts. We must develop

(1) *A high level computational interpretation of geometric, symbolic, and numerical knowledge.*

(2) *Implementation tools for future application systems and their underlying scientific theories.*

Beyond these common concerns, a unified framework must be identified in which to combine the tools developed for different application domains, thereby laying the foundations for a comprehensive system for representing, manipulating, analyzing, and simulating models of physical objects. This system must be designed to allow ready incorporation of different application domains, as well as easy upgrading of components that become superceded by alternative components based on superior techniques.

The comprehensive computational treatment of physical systems even of modest size requires complex models and large computing resources. We cannot burden the user with specifying these models in all detail and explicitly controlling the underlying computers. What is needed is a concise and effective user interface. Geometric, textual, and mathematical information must be integrated into an intuitive and efficient language in which even the nonspecialist can make himself understood and can absorb the information revealed by the system. Huge computing power must be brought to bear and needs to be harnessed with as little explicit user control as possible. In a coordinated effort, we will develop such interfaces. A valuable pool of initial concepts is provided by the experience of the individual projects. Our development work will be further enriched by contributions of other researchers in our department, and should greatly benefit from the diversity of the domains we consider.

Geometric models are an integral part both of Project Newton and of the Mathematical Software Systems project. They constitute a common focal point because they provide a natural medium for channeling the interaction with other model domains. In the Geometric Modeling project we push the available techniques and develop novel tools to be incorporated by the other projects.

Devising a system for computing with realistic physical models is a software project of very large scale. To lay its foundations, we must approach the task at a high enough level, backed up by sophisticated development environments and a competent infrastructure of technical staff. We approximate the needed resources by three layers of hardware:

1. graphics and Lisp workstations,
2. super minicomputers with high bandwidth graphic connections,
3. supercomputer power through a massively parallel machine.

The first layer of workstations takes care of user interfaces and software development environments. It also provides specialized processing such as for expert systems and symbolic mathematical computations. The second layer is needed for computations with moderately complex physical models. Together with the first layer, it approximates the next generation of work stations to be expected on the market in about five years. High bandwidth graphic interfaces are needed for sophisticated animations whose data exchange rates exceed the Ethernet bandwidth.

Larger experiments require parallel processing and supercomputer power. For instance, real time animation of turbulent fluid flow exceeds the computational resources on the second layer. Consequently, we plan to acquire a massively parallel machine. Since our computations are specialized, we do not need to address all issues that arise in parallel computation. The Parallel Processing project will lead the effort of developing application-specific techniques for effectively utilizing parallel machines and for distributing computations over the layers. Together, the three layers provide a perfect paradigm for the next generation of supercomputer configurations and their utilization.

2. PROJECT EQUIPMENT

2.1 Current Departmental Facilities

The main experimental research facility in the Department of Computer Science consists of a DEC VAX 8600 and three VAX 11/780's. The existing research facilities are listed in Tables 2.1 and 2.2 and placed in three groups: generally available, available to our projects, dedicated to other projects.

TABLE 2.1: Computer Systems

Qty	Vendor	Description	Group
1	DEC	VAX 8600 Computer System	*General*
1	DEC	VAX 11/780 Computer System	
1	FLEX	Flex32 Multiprocessor with 7 processors	*Our Projects*
1	Ridge	Ridge32 RISC Architecture Computer System	
6	TEK	Workstations	
2	Symblics	Lisp Machines	
2	DEC	VAX 11/780 Computer Systems	*Other Projects*
27	SUN	Workstations with file servers	
10	DEC and Intel	DEC LSI 11 and Intel 8086 microcomputer systems	

TABLE 2.2: Network Services

Network	Services
ARPANET	Numerous international services, including file transfer, mail, remote login, etc.
Ethernet	Full communication between all departmental machines.
ProNET	Full communication between most departmental VAXes, including a fiber optic link to facilities provided by the Purdue University Computer Center.

The I/O equipment available includes a Megatek graphics processing system, Tektronix 4115 color graphics terminal, three Printronix line printers, 100+ conventional CRT terminals (Wyse and ADDS), and 6 QMS, HP and Apple laser printers. In addition, the

department has access to facilities provided by the Purdue University Computing Center, including a Cyber 205 and an IBM 3083.

2.2 Description of Project Equipment

The needed facilities should be able to simulate the workstation environment of the 1990's, provide support for the integration of graphics, symbolic and numerical computation, and provide support for research on algorithms and software in a massively parallel environment. We plan to acquire the equipment listed in Table 2.3. Table 2.4 gives the schedule of acquiring the equipment.

TABLE 2.3: Summary of Equipment

Qty	Description
2	Alliant FX/1, extra disks
2	Graphics terminals and I/O equipment (Rastertech one/380)
10	Color Graphics Workstations
6	Lisp machines
1	Multiprocessor system I (NCUBE/seven with 32 processors)
1	Multiprocessor system II (NCUBE/ten with 256 processors)
1	NCUBE/Supergraphics Network Equipment

2.3 Rationale For Equipment

The design, manipulation and control of tightly connected computational models requires a computational environment that

(a) supports high level interfaces,

(b) supports the integration of graphics, symbolic and numerical computing,

(c) provides computing power to support realistic simulation of 3D models and time varying interactions.

These correspond to three levels of hardware, the first needs powerful workstations with sophisticated graphics, the second needs local "cycle servers" for Lisp, Fortran, geometry, etc., and the third needs supercomputer backup. We discuss the hardware/software requirements of each project and then point out their common requirements.

TABLE 2.4: Schedule for acquiring the new equipment

Qty	Description
	1987/88
1	Alliant FX/1
1	Graphics terminal and I/O equipment
3	Lisp machines
1	Multiprocessor System I (NCUBE/seven with 32 processors)
	Network equipment (ProNET ring)
	1988/89
3	Scientific Workstations
1	Lisp machine
1	Multiprocessor system II (NCUBE/ten with 256 processors)
1	Supergraphics system attached to NCUBE
	Network equipment upgrades
1	Alliant FX/1 disk upgrade
	1989/90
1	Alliant FX/1
1	Graphics terminal and I/O equipment
1	Scientific Workstation
	Network equipment upgrades
	1990/91
3	Scientific Workstations
1	Lisp machine
	Network equipment upgrades
	1991/92
3	Scientific Workstations
1	Lisp machine
	Network equipment upgrades

2.3.A Project Newton

Project Newton is at a stage where large programs are developed and must be interfaced with existing Fortran code for scientific calculation. It already needs the first two levels of hardware support. Symbolics Lisp machines are a good choice because of the software environment that provides a Fortran interface to the Alliant, the many sophisticated development tools, and the TCP/IP interface to the ethernet. Also of importance is the company's continued commitment to its product line and the clever solutions to garbage collection and floating point handling. The Cornell group is also working on these machines, so software can be exchanged with a minimum of effort. We have found no competitor in overall performance given the project's specific needs.

By year 3, Project Newton will have matured to the point where we can attempt to distribute larger simulations across a network of machines. We envisage an initial partition where the geometry modeling and animation is done locally on a graphics workstation

connected to a powerful cycle server such as the FX/1. Since the major traffic is exchange of state variables and their update, there will be a moderate data traffic on the connection. If real time animation is to be achieved for more complex scenes, high bandwidth interfaces must be used to connect the machines. For thousand state variable exchanges at 30 times a second, an effective speed of 1.2Mb/sec must be sustained.

As Project Newton matures, we plan on two steps. First, we will move it to workstations designed to "deliver" its capabilities rather than to support system development. These will be workstations or small Lisp machines. Second, we will be ready to attempt ambitious, realistic applications that require supercomputer power for both geometric and numerical processing, this will be the NCUBE.

2.3.B Geometric Modelling

The key requirement in geometric modeling is to support the basic research into algebraic and numerical geometry. It is tedious and impractical to draw 3D objects adequately by hand. So this project needs high resolution, excellent color facilities to provide the visualizations. This is provided by the hardware on the first level that supports high level interfaces. As the algorithm development advances, more complex situations and motion studies will use the "cycle servers" on the second -or even the third- level as well as the more sophisticated graphics on the Alliants or NCUBE.

2.2.C Mathematical Software Systems

The Mathematical Software Systems project has already matured to where it needs facilities on all three levels. We visualize interactive model generation and graphical evaluations of results done on a workstation with excellent graphics, these computations are backed by the Alliant for modest problems and the NCUBE for more ambitious ones. Time varying applications will use the powerful graphics directly connected to the Alliant and NCUBE. The expert system will be run using a Lisp machine as a "cycle server".

2.3.D Parallel Processing

A major objective of this project is the development and evaluation of PDE algorithms on massively parallel systems. Currently the NCUBE (256 processors) is the most cost effective choice for a system with more than 128 processors. This facility will be used to identify how effectively kernel PDE and geometric applications can be sped up.

Parallel programming requires the partition of each application and its allocation to hardware resources. We are developing tools and interfaces to automatically solve this problem at load time. High power graphics workstations (Iris 3030) are required to support these interfaces. We also plan to explore a distributed mode of PDE computation. In order to schedule such computation, an intelligent scheduler will be developed and incorporated into PDE-expert system using the Lisp machines.

3. PROJECT NEWTON

The goal of Project Newton is a highly modularized, extensible simulation system in which the precise behavior of physical objects is duplicated from their models. There is an interface to an external programming language in which the controlled behavior of certain objects can be programmed. The research contributing to this goal includes work on a definitional facility in which complex objects are modeled in a multiplicity of domains, each capturing a different category of physical characteristics and behavior. It includes development of an analytic facility that is to simulate physical behavior from models. And it includes a rule-based component inspecting solutions for exceptional events. Since such events entail altered behavior, their presence requires editing object models on the fly. Presently, the scope of the investigation is limited to simulating Newtonian mechanics of possibly jointed rigid bodies from their geometric model. We include at this time modeling of impact and friction, as well as of simple control systems actuating some of the components. As the system matures, extending its coverage will be explored.

3.1. Introduction

Model driven programming, where code for the automatic assembly of objects is derived from a data base description of the assembly has long been a dream of researchers in robotics. A first step towards this goal is a model driven simulation system. Such a system should integrate the following subsystems:

1. A solid modeler for defining the parts to be manufactured and the geometric features of the environment in which they are to be assembled.
2. A simulation system knowledgeable about the laws of physics valid in the environment simulated, as well as the physical characteristics of parts.
3. An off-line planning language for specifying the objects to be assembled from parts, and expressing how to manipulate them, preferably in a generic and functional style.
4. A report generator capable of producing animation, graphical rendering of key events, or summaries of selected properties during simulation according to user specification.

Implementing each of these major components is a daunting task in itself. It is therefore crucial to achieve a system design minimally dependent on the specific internals of the major components, for then these components can make use of existing software packages, and can also be used in other contexts. Moreover, they may be modified in response to advances in knowledge or equipment, and can be exchanged or extended with minimal impact on the rest of the system. By working out clean interfaces between the components and creating suitable layers of abstraction this goal can be achieved.

Since the system must have the capability to simulate the motion of objects under external forces, it can be used to verify many aspects of off-line robot programming such as gripping or moving mechanical parts. However, such a simulation system, driven from a geometrical model, would have much wider applicability. It could be used to construct electronic prototypes and to verify aspects of a design such as the removability of each

board in a computer frame for servicing or the proper unfolding of an antenna on a satellite in space. The potential for payoffs of such a system are enormous. It would allow design changes much later in the development cycle since revalidation only involves rerunning the validating algorithms. Furthermore, devices designed and developed to operate in unusual conditions not easily approximated in the laboratory such as the deployment of an antenna in space that will not support its weight under gravity can easily be prototyped and checked electronically.

Previous work on simulation is extensive. However, most of the related work is either limited to a specific domain such as modeling the control loop of a simplified robot manipulator, Dubowski and Kornbluh (1984), or fails to include the dynamic aspects of the system, Latombe et al. (1984). Moreover, none of these simulations has the capacity to respond to events necessitating a change of the quantitative model during the simulation. This capacity is crucial when realistic simulation behavior is desired.

An exception is STEAMER, Hollan et al. (1984), a simulation system modeling turbine propulsion systems for training engineers. STEAMER attempts to separate different levels of abstractions and incorporates some information hiding. Its chief limitations are the specificity of the modeled domain that seems to tolerate no extension to new modeling domains, and the restriction on possible automated support for abstracting complex objects. What is needed is to incorporate certain aspects of each of these works into a single system in which the simulation is driven from the system description itself, along with a program globally directing the progression of events.

3.2. Approach

In collaboration with the robotics group at Cornell under John Hopcroft, we are presently implementing an electronic prototyping system and its components by studying the following situations:

1. A chain of hinged links is moved by prescribing a fixed trajectory for the top link, modeling an idealized control and actuation system. The other links move according to the induced forces on them, with or without additionally assuming gravity.
2. A block is placed on top of an assembly of blocks that are not fastened to each other. Specific possibilities include: the block to be placed collides with some of the blocks in the assembly and knocks them over, or the block is placed in an unstable configuration and the assembly topples, etc.
3. A simplified model of a human figure steps up on a curb. The model of the figure contains a sensory based feedback model of balancing and stepping strategies. The feedback model may compensate for stepping short or too high.

These examples develop different aspects that must be integrated, namely kinematic constraints, dynamic editing of interrelated objects and impact, and the incorporation of sophisticated control and actuation schemes. Example (1) has been demonstrated at the 1986 Artificial Intelligence Convention.

In all cases, the sequence of activities is as follows:

1. Complex objects have been defined in a language that is implemented by accessing definitional facilities of modeling subsystems for specific domains (geometry, dynamics, control systems, etc.).
2. The simulation scenario is initialized, possibly with the help of a graphical interaction language. Moreover, a program expressing the various motion plans and manipulator activities has been written and interfaced with the simulation.
3. The scenario is simulated observing the physical laws in effect and interpreting the program directing the manipulator and other *active* agents in the simulation.

We describe how the major system components are structured and interact.

3.3. The Modeling Subsystems

Every object to be simulated is modeled in a number of domains. Presently, we model the geometry, the dynamic behavior, and the controlled behavior of objects. We foresee the need of modeling objects in different domains as well. For example, while at present the impact of two objects is abstractly summarized by the notions of impulse and coefficients of restitution, it may be desirable at a future time to extend the simulation into a domain where elastic deformations and internal stresses of objects are modeled and simulated. Consequently, the system design has to permit a basic flexibility and extensibility.

An object is modeled by a definition language that is implemented by accessing a number of separate subsystems, each in charge of modeling a specific domain. These subsystems are at present the geometric modeler, the dynamic modeler, and the control modeler. The subsystems are unaware of the fact that the model constructed by them is coordinated on the next higher level with models from the other domain. This ensures greater flexibility when adding new domains of modeling.

Geometric Modeling

Geometric models are built from primitive shapes and are combined in a CSG-like manner into rigid bodies as outlined in Hoffmann and Hopcroft (1985). Briefly, shapes are named and combined by the usual CSG operations of regularized union, intersection, and difference. In addition, *features* on shapes, i.e., surfaces, space curves and points, can be defined and referenced by a sophisticated naming scheme. The advantage here is that it permits the use of definitional operations that are intuitive rather than relying on the usual tedious and error-prone coordinate calculations. For example, the *attach* primitive joins two shapes by mating congruent features on the two component shapes.

Geometric coverage has been limited so that the technical difficulties that must be addressed, e.g., volume and inertia calculations, do not distract from the central problem of organizing the system in the right way. At a later time the geometric coverage will be enlarged.

Automatic capabilities are implemented to compute volume and volumetric moments of inertia, i.e., to determine shape-dependent information needed by the other modeling systems. Each resulting object is represented relative to a body-specific local coordinate system. On the next higher level this coordinate system provides the means for connecting the geometric model to the models in other domains, of the same object.

Dynamic Modeling

The dynamic behavior of a primitive object is captured in its *dynamic model*. Each object is modeled by a local coordinate frame and a set of state variables and equations describing the dynamic response of the object under external forces and torques, irrespective of whether these forces act over an extended period of time or are impulsive. The state variables are position, orientation, velocity, mass and inertia. For nonimpulsive forces F and torques T, the vectorial equations of motion (more precisely, the equation schemata) are $m\ddot{r}=F$ and $J\dot{\theta}+\dot{\theta}\times(J\dot{\theta})=T$, where J is the inertia tensor. Similar schemata describe instantaneous velocity changes due to impulsive forces and torques, e.g., Wittenburg (1977).

A primitive object is a single rigid body. Other primitive objects could be added to the modeling system, provided they can be viewed as local coordinate frames with a dynamic behavior that can be expressed by the appropriate equations of motion. Although unexplored as yet, this facility for adding primitives is ultimately a vehicle for abstracting the dynamic behavior of more complex mechanisms: For instance, a pocket watch being manipulated by a gripper can be abstracted as a rigid body since its internal dynamic behavior is likely of no interest to the gripper motions. Thus, if we were to describe both the internal working as well as the abstraction of the watch, then the simulation of time keeping could be made independent from the simulation of the gripping device moving the watch. Such partitioning may play a major role in eventual parallel versions of the system.

Control Models and Sensing

Control loops and sensors are modeled abstractly as functions depending on the state variables and their rate of change, and on geometric properties. They may have additional state variables of their own. For example, the control loop for a motorized arm approaching an object may express a relationship between a distance sensing variable, a voltage regulator state variable, and a torque variable. At the primitive modeling level, the variables have not been bound to the corresponding constructs in the other models involved. The binding is effected at the object modeling level.

Control models also provide a natural interface with an external programming system. More sophisticated schemes for intervention during the simulation and actuating selected components can be expressed as control models some of whose state variables are communicated to and modified by an external program that has been appropriately synchronized.

Object Definition Level

So far, no provisions have been described to construct complex objects with moving parts. These provisions are made at the higher level which we call *definition level*. To begin with, primitive objects are defined by accessing the lower level and defining the relevant models. The different domains are then coordinated by relating the intrinsic coordinate frames on the one hand, and by binding the appropriate state variables on the other.

Complex objects are constructed from primitive objects by a sequence of coordinated composition operations of the constituent primitive models. These compositions are drawn from a set of primitive operations that, roughly speaking, connect objects by joints or geometric adjacency, collectively called *hinges*. This concept of hinge is as in Wittenburg

(1977).

For the geometric model, composition implies merely a geometric relationship between the coordinate systems of the constituent primitive components. For the dynamic model, components are related by *kinematic constraints* that express, intuitively speaking, the hinge geometry equivalently by constraints on the state variables, their derivatives, and new *constraint forces*.

There are cases in which the imposed constraints have no direct geometric counterpart, i.e., there need not be a functionally complete geometric model realizing the kinematic constraint imposed. For example, we might wish to join two objects by a fictitious pin hinge that would have to interpenetrate a third object were it to be realized directly. Designing such virtual hinges later "implemented" by a physical linkage of different geometry is common practice in machine design and analysis.

Due to the fact that the component parts move relative to each other, the dynamic model of a complex object is described as the union of the models of its constituent parts, plus the set of constraint equations. Since hinges connect components, the resulting model is a graph whose vertices are the dynamic models of the primitive components, and whose edges correspond to the constraints imposed when placing the hinge. Since constraint forces transmitted by these hinges may have to be known, for instance in the case of modeling compliant motion, corresponding state variables are associated with the graph edges. In the resulting model of the complex object, the constraint equations, along with the equations of motion result in ordinary differential equations that form a system of linear equations whose unknowns are the changes to the state variables and constraint forces. The coefficients of these equations are (not necessarily linear) functions of the state variables and the external forces and torques applied to the object. The system may need conditioning to make it better suited to simulation, e.g., Hoffmann and Hopcroft (1987), and is solved numerically.

Virtually all kinematic hinge compositions can be reduced to a sequence of applying a single composition primitive that constrains a point of an object 1 to coincide with a surface of an object 2. This is a significant fact, for it increases the ease of extending the modeling coverage to new domains: The basic composition repertoire of the definition level that must include cylindrical, revolute, prismatic joints, and others, can be defined without awareness of the different modeling domains. Only the single composition primitive needs to know the domains available.

3.4. Initialization and Simulation

Having defined the relevant objects as classes, *instances* are created under direction of a simulation language, thereby initializing the simulation. Of course, the class definitions are given interactively or retrieved from a data base or both. It is now also necessary to relate the body specific coordinate systems to a world coordinate system. In addition, global rules such as "all objects are subject to gravity," have to be declared. A suitable language for this is currently being developed, Hoffmann, Hopcroft and Whitesides (1986).

Simulation then commences with interpreting the command language directing the various control loops and sensor models, along with simulating the physical forces in effect. The simulator accesses the model instances to simulate a time step. For the geometric model, this includes an interference test that ascertains that no two objects

interpenetrate. For the dynamic model, all external forces must be determined, and from them the new values of the state variables are obtained. Finally, the input variables for the control and sensing models are determined, and from them the relevant control outputs are found.

Before the models are updated, the results must be interpreted for exceptional conditions. These exceptional conditions include the following:

1. Two objects geometrically interpenetrate or come into contact.
2. A constraint force that should be positive vanishes or becomes negative.
3. The dynamic model may be indeterminate.

In response, an event handler must *edit* the models: Two objects coming into contact implies impact. Using the impact behavior of the dynamic model, the effects of the impact are instantaneous changes of certain velocities. If the contact persists (inelastic impact or due to friction), a new constraint force pair between the objects in contact must be modeled as a composition with a hinge that sustains pressure but breaks under tension.

A vanishing or negative constraint force at a *pressure-only* hinge signals that the contacting bodies separate. In response, the dynamic model is updated by deleting the corresponding graph edge and updating the equations of motion.

It is well-known that when a rigid body is supported at more than two points the corresponding set of constraint forces is indeterminate. This is a model deficiency of Newtonian mechanics. Different methods exist to handle the situation. One possibility is to model infinitesimal penetration of the contacting bodies and determine rebounding forces. Other possibilities include modeling internal stress and elasticity characteristics, i.e., adding a new modeling domain. The system is designed such that different methods of handling this situation can be modeled and simulated.

3.5. Interference Tests

We need to ascertain rapidly whether two objects interfere in 3-space. A direct approach is computationally too expensive, so the interference test is done hierarchically. We are presently exploring techniques from computational geometry to design a suitable polyhedral approximation sequence that is automatically derived by the system. Other possibilities include a spherical approximation sequence exploiting the fact that spheres have perfect rotational symmetry and that intersections among spheres can be found quickly, Hopcroft, Schwartz and Sharir (1983).

In principle, the test should be structured as follows: On the top level, the object is a single simple shape. Next, the object is the union of a number of smaller shapes enclosing the object more precisely. Finally, the geometric model of the object is used. Note that it is usually not necessary to intersect complete models. Only the spaces of intersection in the next higher hierarchical level have to be examined, and many complex operations such as the assembly of a model of the intersected volume are not needed.

Movable subassemblies may be known not to intersect. For instance, the wheels of a cart do not touch the cart body. For this reason, certain components may be identified as nonintersecting pairs. For complex objects, this information is stored as a graph and is

used to reject certain intersections as noncritical.

3.6. Immediate and Long Term Plans

We are completing a first version of the system designed as described in Hoffmann and Hopcroft (1987). Primitive shapes are presently limited to cuboids, and at the time of writing friction has not yet been incorporated. Extending the geometric modeling capabilities could have been done easily, but had been postponed in favor of developing the global structure of the system more thoroughly. The work is conducted in cooperation with the robotics group at Cornell and includes sharing software written in Common Lisp.

After the initial development phase, the project will branch out. On the one hand, detail issues will be explored, e.g., how to speed up the interference test. This work, and the associated software development can be done on smaller Lisp machines. On the other hand, we will explore effective ways to parallelize the system. Initially, we plan a conservative step as follows: The geometric model and the animation facility resides in a graphics work station, that also contains the user interface. For simple wire frame animations, these workstation have already sufficient power to locally update the screen in real time. The dynamics and control model reside on a cycle server with powerful floating point capabilities. This server itself could be exploiting parallelism, and here we can take advantage of the results of the parallel processing project.

In this system partition, only state variables are routinely exchanged. Occasionally, e.g., upon impact, geometric information is communicated, but this will happen relatively infrequently. Assuming each state variable requires on the order of 4 bytes and that the screen is updated 30 times a second, each primitive object with its 6 state variables generates a data traffic of approximately 5.8kb/sec, assuming that no constraint forces must be communicated. Thus a simulation with more than 174 primitive objects will exceed Ethernet capacity. We expect that by the second half of the granting period further parallelization will become mandatory. At that time, we will explore ways to effectively distribute the system over the NCUBE, thereby moving into a position where complex experiments can be conducted with interactive performance. This involves further parallelization, perhaps partitioning by locality, and will apply the insights of the Parallel Processing project.

We also plan to extend the coverage of the system, once a well developed prototype has been implemented. Here we consider extending both the capability of the individual submodeling systems, e.g., adding more sophisticated geometric models to our repertoire, as well as extending the physical coverage by adding new domains of modeling and simulation, e.g., capabilities for vibration, elastic deformation, heat flow, etc. Such extensions very naturally foster strong interactions among the central research programs described in this proposal, and serve as integrating forces to the research and the needed infrastructure.

3.7. References

Ambler, A., (1984), "Robotics and Solid Modeling: A Discussion of Requirements Robotic Applications put on Solid Modeling Systems". *Robotics Research*, 2^{nd} Intl. Symp., Kyoto, 1984, MIT Press, 1985, 361--367

Dubowski, S. and R. Kornbluh, (1984), "On the Development of High Performance Adaptive Control Algorithms for Robotics", *Robotics Research*, 2^{nd} Intl. Symp.,

Kyoto, 1984, MIT Press, 1985, 119--126.

Fishwick, P.A., (1986), "Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction", Ph.D. Dissertation, Electr. Eng., University of Pennsylvania

Hoffmann, C. and J. Hopcroft, (1985), "Automatic Surface Generation in Computer Aided Design", *The Visual Computer 1*, 92--100.

Hoffmann, C., J. Hopcroft, and S. Whitesides, (1986), "Constraint Based Placement of Geometric Models", in preparation.

Hoffmann, C. and J. Hopcroft, (1987), "Simulation of Physical Systems from Geometric Models", to appear, special issue of *IEEE J. of Robotics and Automation*, 1987.

Hollan, J.D., E. L. Hutchins, and L. M. Weitzman, (1984), "STEAMER: An Interactive Inspectable Simulation-Based Training System", *AI Magazine* 5, 15--28.

Hopcroft, J., J. Schwartz, and M. Sharir, (1983), "Efficient detection of Intersections among Spheres", *Intl. J. of Robotics Res.* 2, 77--80.

Latombe, J., C. Laugier, J. Lefebvre, E. Mazer, and J. Miribel, (1984), "The LM Robot Programming System", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 377--391.

Lozano-Perez, T., (1983), "Spatial Planning: A Configuration Space Approach", *IEEE Trans. on Comp. C-32*, 108--120.

Mason, M., (1984), "Mechanics of Pushing", *Robotics Research*, 2nd Intl. Symp., Kyoto, 1984, MIT Press, 1985, 421--428.

Papadopoulos, J., (1986), "Incremental Deformation of an Irregular Assembly of Particles in Compressive Contact", Ph. D. Dissertation, Dept. of Mech. Engr., Cornell University

Peshkin, M., and A. Sanderson, (1985), "The Motion of a Pushed, Sliding Object; Part I: Sliding Friction", *Tech. Rept.* CMU-RI-TR-85-18, Robotics Institute, Carnegie-Mellon Univ.

Wesley, M.A., T. Lozano-Perez, L. I. Liberman, M. A. Lavin, and D. D. Grossman, (1980), "A Geometrical Modeling System for Automated Mechanical Assembly", *IBM J. of Res. and Devel. 24*, 1980, 64--74.

Wittenburg, J., (1977), *Dynamics of Systems of Rigid Bodies*, B. G. Teubner, Stuttgart, W. Germany, 1977, 244p.

4. GEOMETRIC MODELING

The project seeks to develop new techniques for representing and manipulating the shape and structure of physical objects. On the one hand, it explores fundamental mathematical properties of curved surfaces and develops new techniques to analyze them with efficient methods. On the other, it develops the techniques for assisting the user with automatic derivations and experiments with linguistic concepts by which to direct this assistance.

4.1. Introduction

Geometric Modeling is the use of a collection of techniques to describe the shape and structure of physical objects or dynamic processes. Its power stems from the ability to use efficient algorithms for describing computer models of complex shaped objects as arrangements (e.g., boolean operations of union, intersection and difference) of simpler ones. Research in geometric modeling requires a deep understanding of mathematical facts and seeks to develop efficient tools that assist the naive user in a spatially intuitive manner. A good geometric modeling system finds applications in engineering design, analysis and simulation. Designed with sufficient generality, it can also be used in motion planning Bajaj and Kim (1987d). Both Project Newton and the Mathematical Software project will use the advances made by the Geometric Modeling project.

Many representations have been developed for modeling the shape of physical objects in computers. A popular representation is the *wire frame* model, e.g., Foley and Van Dam (1983). Though it allows modeling complex objects with modest computational resources, it is an ambiguous representation not completely defining the solid. Another representation is through volume decomposition by *octrees*, Meagher (1982). It is primarily used in image processing. Octree models cannot be rotated easily, since the volume decomposition must respect the principal coordinate directions. It is also difficult to support finite element methods with an octree representation. More general purpose solid modelers use one of two representation methods, Requicha and Voelcker (1980). The first method, called *constructive solid geometry* (CSG), represents an object by combining primitive solids using Boolean operations. Primitive solids usually include cuboids, spheres, cylinders, tori, and cones, but could be extended to more complex shapes. The modeled solid is stored as a tree whose leaves are primitive solids and whose interior nodes are operations on them, typically (regularized) union, intersection, and difference, as well as translation and rotation. For such a tree it is difficult to determine whether the represented solid is empty and whether two solids are equal. The second method, called *boundary representation*, represents a solid by the topological structure of its vertices, edges and faces, along with their geometric description. In this representation, it is difficult to determine whether two solids are equal unless faces are defined in a canonical way.

We are investigating a spectrum of problems furthering the ability to represent and manipulate geometric models Bajaj and Kim (1987e, f), Bajaj, Hoffmann and Hopcroft (1987), Bajaj, Lui and Wu (1987). Using a boundary representation scheme has the advantage of direct applicability of the results to Project Newton, to path planning research, and to solving partial differential equations (PDE) used by the Mathematical Software Systems project. We describe now some of the research problems that need to be solved in this effort.

4.2. User Interfaces

The design of an effective user interface is a neglected, but important problem area in geometric modeling. All too often the user has to penetrate complicated conventions for describing the building blocks from which to construct objects, and has to master complex and error-prone spatial transformations in positioning objects.

It is widely accepted that programming effort is related to the correct structuring of the problem to be solved and the algorithms for solving it. Likewise, the design of physical objects and systems of physical objects must be well structured. Even a simple object such as the gate valve shown in the picture below requires defining a variety of component shapes and interrelating them by position, orientation, and dimension. The problem quickly magnifies when *generic* designs are done, which must be archived in a data base for future use in deriving design variants.

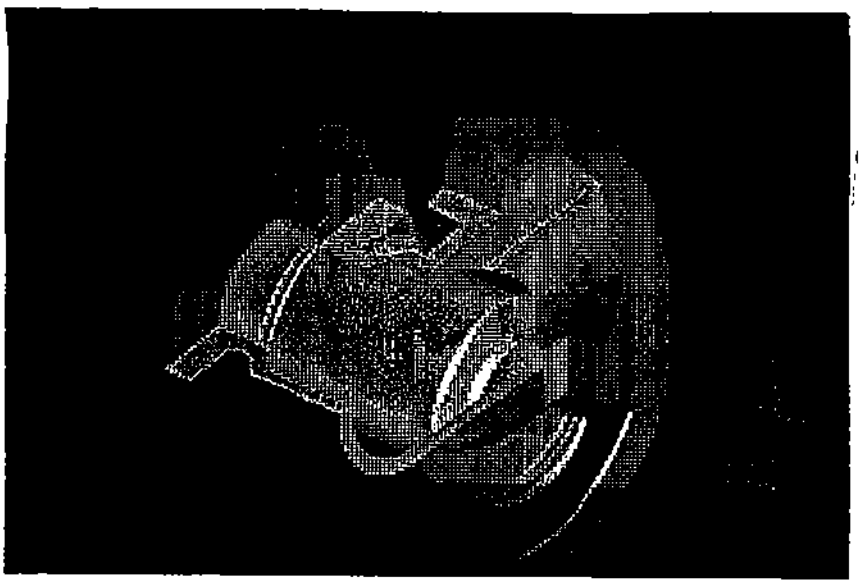


Figure 4.1. Gate Valve: All blending surfaces can be derived automatically, Hoffmann and Hopcroft (1985).

While an interactive, graphical component is very important, the specification language must also include a complete textual component for documentation and archiving purposes. In Hoffmann and Hopcroft (1985) we have given a preliminary textual language design based on considerations we now describe. Various elements of our design have also been advocated by others, e.g., Poppelstone et al. (1980).

Generic design requires the ability to dimension shapes symbolically and express complex relationships between object features. One way to achieve this goal is to syntactically define shape procedures with parameters, as well as express numerical relationships between parameters by equations. As an example, consider the generic definition of *pipe* of Hoffmann and Hopcroft (1985).

pipe(radius, thickness, length) := $(cyl1 - cyl2) \cap H1 \cap H2$ **where**

begin
cyl1 := ycylinder(radius + thickness);
cyl2 := ycylinder(radius);
H1 := {y ≥ 0};
H2 := {y ≤ length};
top := (cyl1 − cyl2) ∩ {y = length};
bottom := (cyl1 − cyl2) ∩ {y = 0};
outside := cyl1.surface ∩ H1 ∩ H2;
top.in_edge := top ∩ cyl2;
bottom.in_edge := bottom ∩ cyl2
end

Then *pipe(3,1,7)* would refer to a pipe of (inside) radius 3, wall thickness 1, and length 7. Moreover, pipes with a wall thickness functionally dependent on the radius may be defined; e.g.,

pipe2(radius,length):= pipe(radius, radius/10, length).

When two shapes are combined, the intrinsic coordinate frames in which the two shapes are defined must be related. Traditionally, one must "move" one of the shapes relative to the other's coordinate frame, e.g., Brown (1982). This is intuitive only in the simplest cases, i.e., as long as no rotations are required, but in general this style of specification becomes difficult and error-prone. We therefore make provisions to name *features* as body-specific frames of reference. For example, in the pipe definition above, only the names *cyl1, cyl2, H1,* and *H2* are part of the definition the shape. The remaining names define features. For example, *pipe.bottom.in_edge* refers to the edge of intersection of the pipe's inside with the "bottom" end of the pipe. In conjunction with a syntactic schema for inheriting features of substructures, we can define and implement intuitive language constructs like *attach* that combine objects by mating congruent features.

A rather difficult topic at present is how one best expresses functional dependency, for very little systematic knowledge about it seems to exist. Yet a clear understanding of functional dependency is a key to developing higher level modeling languages. For example, in the design of objects for the purpose of simulation, it is clear that certain shape parameters may be dependent on overall functional characteristics: In modeling a robot arm, the intended pay load influences both the material as well as the structural aspects of the links. Ideally, one would like to make a generic design and derive from it specific instances suited to the application demands. While the knowledge to formulate and exploit such dependencies to the fullest may not yet exist, it is not unreasonable to seek means of expressing such dependency where recognized. In this regard, the modeling project will benefit considerably from a coordinated effort with Project Newton.

4.3. Automatic User Assistance

A sophisticated user interface hides the mathematical complexity of many modeling steps. To support this information hiding and to expose the essential information needed to control complex steps easily and effectively, automatic facilities must be developed that implement the interface language.

Automatic Surface Derivation

Virtually all objects in engineering design possess *blending surfaces*, that is, surfaces of rounded appearance that smoothly connect two primary surfaces; e.g., Rossignac and Requicha (1984). These blending surfaces either round otherwise sharp edges, or else connect similar surfaces in a standardized manner. Frequently, blending surfaces are only approximately specified and within limits their precise shape is not crucial.

Blending surfaces of an object may be difficult to describe. For example, consider the picture of the valve shown above in Figure 4.1. Here the blending surfaces are shown in blue. Deriving them explicitly is difficult because of the tangency requirement at the seams, and significant mathematical knowledge is required to obtain them directly. This is unacceptable to the user.

In Hoffmann and Hopcroft (1985), we have introduced a simple method for blending that combines great simplicity of derivation with intuitiveness and flexibility. Given two arbitrary algebraic surfaces, the method delivers a blending surface joining the two that is of low algebraic degree. In fact, in Hoffmann and Hopcroft (1986) we show that for quadrics in general the method delivers *all* low-degree blending surfaces. Complex corners can be smoothly rounded by exploiting generic base configurations, as explained in Hoffmann and Hopcroft (1987b). A few cases are illustrated in Figure 4.2 below.

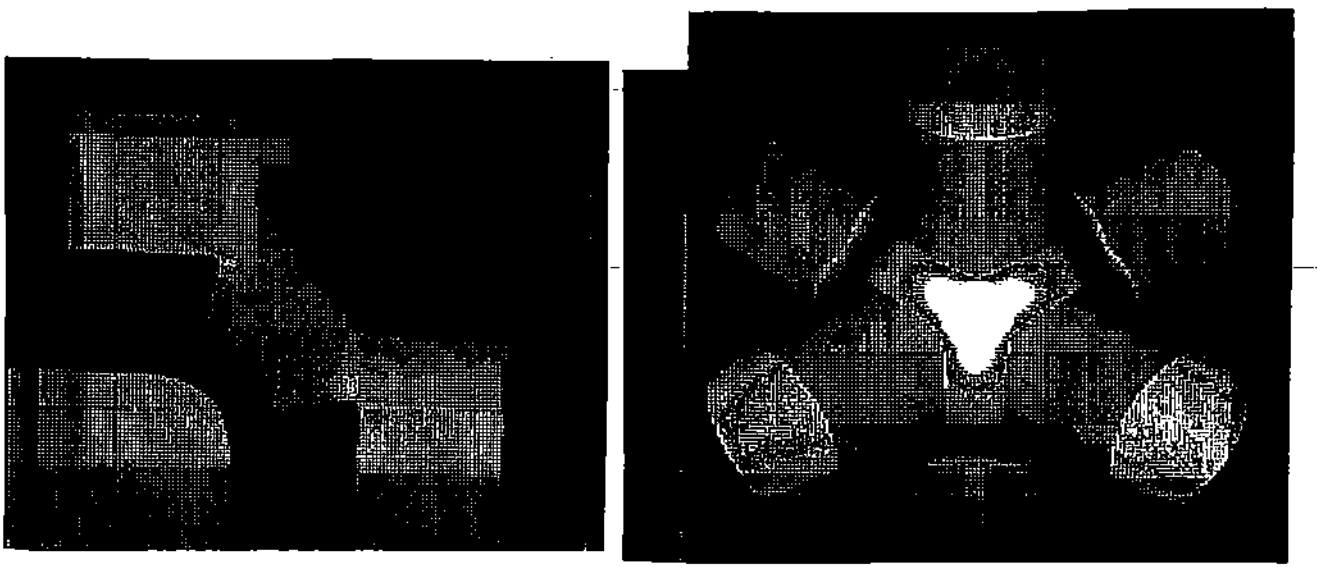


Figure 4.2. Examples of blending complex corners using the algebraic method of Hoffmann and Hopcroft (1987b).

We have extensively experimented with the mathematics of blending surfaces. Two major tools proved invaluable: symbolic algebraic computation, giving us access to explore different, often complex surface derivations, and graphical rendering of these surfaces, allowing us to inspect their behavior visually. A number of theoretical results, e.g., the completeness theorem for our method, are an outgrowth of this experimentation during which it was observed that all quartic surfaces we ever found for blending two quadrics

were derivable by our method. [We later found the analytic proof of this observation in Hoffmann and Hopcroft (1986)]. Now that our method for surface derivation has matured, we can incorporate it into solid modeling by means of simple primitives. In Hoffmann and Hopcroft (1985) we have made a first attempt at defining such a language primitive, i.e., the *smooth_attach* primitive. Like *attach, smooth_attach* combines two objects by mating features on them. It also adds a blending surface of approximate radius of curvature. This curvature radius is specified by the user who thinks of placing a surface of approximately circular cross section with given radius.

Many problems in blending remain that require substantial experimentation. For example, we have a flexible method for combining blending surfaces at corners without degree penalty. This involves constructing certain base configurations, and reducing the general problem to them. At this time, we can handle vertices incident to three edges. The systematic treatment of vertices of higher valence is presently not available unless surfaces of high algebraic degree are accepted. Other situations such as blending across tangential surface patches also require further research. In order to attack these problems, the theoretical work done with the help of symbolic algebraic manipulation systems has to be combined with experimental methods that explore visually by rendering the surfaces.

Automatic Path Planning

One way of automatically planning collision free paths for a single rigid object among physical obstacles is by reducing it to planning paths for a mathematical point among "grown" obstacles in *configuration space* (*C*-space). Path planning for a point is simple since a point can be moved without restriction in any connected region of the *C*-space Bajaj (1986, 1987a). However, the construction of *C*-space obstacles proves to be difficult in general. In the case of polyhedral objects, efficient algorithms were given by Lozano-Perez (1983) and Donald (1984). In Bajaj and Kim (1987a,b,c), algorithms were developed for constructing the *C*-space obstacles for objects described by algebraic curves and surfaces. A collision free path is then found by combining straight line segments in free space with geodesic paths on obstacle surfaces.

Our algorithms for generating *C*-space obstacles can also be used to generate swept surfaces. To see this, observe that for a moving sphere the surface of a *C*-space obstacle is simply the offset surface of the original object. Although of high degree, such surfaces can be used for blending. Rossignac and Requicha (1984), Farouki (1986) discuss ways to approximate such surfaces from simpler ones.

If the movement of polyhedral objects is restricted to translations only, the *C*-space obstacles are also polyhedral. In this case, shortest collision-free paths are relatively easy to determine. In Bajaj (1985, 1986, 1987a) and Bajaj and Moh (1987), both algebraic and numeric solutions to this problem have been given. Kantabutra and Kosaraju (1986) and Kosaraju (1986) give efficient algorithms for moving multilinked arms in confined regions. The solutions to these and similar path planning problems need to be incorporated into a general system capable of generating paths automatically.

Automatic Mesh Generation

Meshes are a means of partitioning or discretizing space into small elements where relatively simple, local models and analysis are sufficient. By far the most common

instance of this occurs in solving partial differential equations (PDEs), for example the finite element method; see Thompson et al. (1984) for a survey. However, this approach has much greater generality, examples of which include piecewise polynomial approximation, domain splitting methods (sub-structuring and Schwarz splitting) for physical simulation, and multifrontal methods for large sparse problems. We believe that meshes will eventually became a tool to provide efficiency in many geometric processing applications.

A key concern in mesh generations is obtain a fineness that is compatible with the application's need. Thus, one needs a fine mesh where rapid oscillations occur and where high accuracy is needed. Mesh construction is very tedious and error prone, so automatic algorithms are essential. Recently, Ribbens (1986) has given a framework for creating meshes compatible with general criteria for fineness. He has also developed new, efficient algorithms for automatically computing well-conditioned meshes. Rice (1986), has shown that certain regular meshes (tensor products) can handle singularities without a substantial penalty in the number of elements. These algorithms provide the infrastructure for a system which automatically refines or creates meshes in response to either problem characteristics or user direction.

4.4. The Curved Surface Domain

A large part of our research effort aims at extending the types of surfaces a solid modeler can handle to include algebraic surfaces of arbitrary degree. Current geometric modeling systems disallow curved surfaces outright, e.g., Laidlaw et al. (1986), or severely limit the types of curved surfaces they can handle. For example PADL, e.g., Requicha and Voelcker (1980), can represent only planar, spherical, cylindrical, conical, and toroidal faces. Other systems use bicubic patches, Faux and Pratt (1981). The problems that must be solved in extending the geometric coverage necessitate combining results from algebra, geometry, topology, and numerical approximation theory.

Representation

No matter what surfaces are represented, basic foundational work is required that yields the insights needed to assess whether the desired operations can be carried out, based on the chosen representation scheme, and how to implement them correctly. Many modeling systems are in use today that have neglected this issue, and in consequence, they fail for certain object configurations that are frequently encountered in applied work.

As long as all faces are planar, the underlying theory is well understood. Work exists that describes such modeling systems and convincingly argues that the system is complete and correct, Paoluzzi et al. (1986). When extending the geometric coverage to curved surfaces new difficulties are encountered. Both topological as well as with geometric ambiguities can be present in the chosen representation scheme. While possible topological ambiguities are widely appreciated, Weiler (1985), possible geometric ambiguities are not. In Hoffmann and Hopcroft (1987a), we have addressed this issue, showing that the natural extension of boundary representations to the curved surface domain is geometrically ambiguous. Figure 4.3 shows side by side two different interpretations of the same boundary structure. The two interpretations differ in shape and in position. Representation schemes that avoid such ambiguities can be devised and are discussed in that paper.

Figure 4.3. Example of ambiguity, two different interpretation of the same boundary structure are shown, Hoffmann and Hopcroft (1987a).

Geometric Coverage

We are extending the geometric coverage of solid modelers to algebraic surfaces of arbitrary degree. This will enable us to design and represent more accurately the geometric models of physical objects along with the environments in which to manipulate them. Increased geometric coverage also permits the incorporation of blending surfaces into the modeling system and allows more realistic mesh generation.

Low degree surfaces (e.g. quadrics) permit efficient use of exact algebraic techniques. This is generally not the case for higher degree surfaces, and numerical techniques become necessary. For example, bicubic patches are popular in computer graphics and engineering, but are surfaces of degree 18 and so do not allow efficient exact techniques, Sederberg et. al., (1985).

When algebraic surfaces of arbitrary degree are permitted, implementing the regularized intersection of two solids in boundary representation requires the solution to a number of basic mathematical problems. Briefly, we must devise efficient methods for determining the complete intersection curve of two algebraic surfaces, for analyzing its components and singularities, and for linearly ordering intersection points along a regular segment of the curve. Fortunately, these problems can be formulated and attacked separately.

Singularity Analysis

Self intersections of curved surfaces and space curves give rise to singularities. These singularities occur frequently in practice. For example, when sweeping a shape along a space curve singular surface points are easily generated. When intersecting two objects with curved surfaces, the intersection curves may have singularities even though none of the intersecting surfaces have singular points. Occurring singularities must be determined

explicitly, for in the vicinity of a singularity most algorithms needed to implement modeling operations will fail.

Singularities can be resolved by special techniques. The desingularization theorem for plane curves says that any given algebraic plane curve can be transformed by a birational transformation (i.e. by an almost one-to-one algebraic transformation) into a curve devoid of singularities. A similar theorem also holds for surfaces, in fact over fields of any characteristic. Desingularization is useful for algebraic integration needed when computing volumetric and inertial properties of physical objects in our geometric modeler. We are currently implementing the desingularization method for algebraic plane curves given in Abhyankar (1983) to robustly trace the curves with proper connectivity, Bajaj, Hoffmann, and Hopcroft (1987).

Desingularization yields enough information so that all the singularities on the curve or the surface can be found. For curves there can only exist a finite number of point singularities and these are systematically obtained together with the number of branches of the curve at each singular point by the method of Abhyankar (1983). Surfaces may contain both isolated singular points as well as curves of singular points. Obtaining them all is more difficult, Giffiths and Harris (1978), and efficient computational methods are presently not known.

Parametric vs. Implicit Forms

The rational parametric form of representing a surface allows greater ease for transformation and shape control. The implicit form is preferred for testing whether a point is above, on, or below the surface, where above and below is determined relative to the direction of the surface normal. As both forms have their inherent advantages it becomes crucial to be able to go efficiently from one form to the other, especially when surfaces of an object are automatically generated in one of the two representations. The tool for converting the parametric representation to the implicit one is called *elimination.* One computes polynomial resultants, Griffiths and Harris (1978). Closed form solutions are known to exist for two polynomials in a single variable, (e.g. Sylvester's resultant). However no satisfactory solution is known for three or more multivariate polynomials. Taking the resultant for two polynomials at a time leads to extraneous factors that cannot be avoided. In practice, this means that the resulting implicit form describes not only the parametric surface but, in addition, other surfaces, Bajaj (1987b). For three polynomials in two variables a satisfactory solution may exist, but it is unknown at this time. Finding a solution would mean that affine parametric surfaces in three space can be efficiently implicitized. We are currently exploring if such a method can be based on the invariant method of Grace and Young (1902).

To go from the implicit to the parametric form is more difficult. For surfaces of degree higher than three no rational parametric forms exist in general, although parameterizable subclasses can be identified. For low degree curves and surfaces Abhyankar and Bajaj (1987a,b) have developed and implemented procedures for parameterizing implicit forms. The approach has been extended to parameterize planar curves of higher degree and special space curves, Abhyankar and Bajaj (1987c). These methods can be specialized to work over rational or real fields, and have been implemented in MACSYMA. Currently we are trying to obtain explicit parameterizations of special families of quartic surfaces and surfaces of higher degree which we plan to use for representing blending surfaces.

Approximate Representations

The efficiency of almost all computational methods for problems dealing with curves and surfaces depend primarily on the algebraic degree of the equation being manipulated. Using lower degree surface approximations for the higher degree surfaces generated, e.g., for complex blending surfaces, is therefore a very attractive possibility that must be explored. In such an approach to modeling, one chooses a family of low degree rational algebraic surfaces that give sufficient flexibility in controlling shape so as to enable close approximations of high degree surfaces. Choosing a good family of approximating algebraic surfaces requires extensive experimentation and good graphics tools. It also requires research to find efficient computational methods for obtaining close approximations by rational parametric patches. Rational parametric surfaces represent a wider class of algebraic surfaces than those represented by polynomial parametric patches, Griffiths and Harris (1978). Most of the work in the past has focussed on the approximation of functions, Rice (1969) or the approximation of curves and surfaces by polynomial parametric patches, e.g., Barnhill (1977).

4.5. Immediate and Long Term Work

At this time we are completing the implementation of a polyhedral solid modeling system, Hoffmann et al. (1986). This system will be extended to model algebraic surfaces of arbitrary degree. Both the global structure as well as the interfaces among the internal components have been designed so as to reduce this extension to a number of well-defined, specific components, each addressing one of the problems discussed above. A similarly conceived rendering algorithm already exists. Extending it to render the larger class of algebraic surface objects requires components based on the same mathematical problems, so both programs will be extended at the same time.

4.6 References

Abhyankar, S. S., (1983), Desingularization of plane curves. In *Proc. Symp. in Pure Mathematics*, 40, 1-45.

Abhyankar, S. S., and C. Bajaj, (1987a), "Automatic rational parameterization of curves and surfaces I: Conics and conicoids", *Computer Aided Design*, 19,1, 11-14.

Abhyankar, S. S., and C. Bajaj, (1987b), "Automatic rational parameterization of curves and surfaces II: Cubics and cubicoids", *Computer Aided Design*, to appear.

Abhyankar, S. S., and C. Bajaj, (1987c), "Automatic parameterization of rational curves and surfaces III: Algebraic plane curves", CSD-TR-619, Computer Science, Purdue University.

Atallah, M., and C. Bajaj, (1987), "Efficient algorithms for common transversals", *Information Processing Letters*, 25, 2, 87-91.

Bajaj, C., (1985), "The Algebraic complexity of shortest paths in polyhedral spaces". In *Proc. 23rd Annual Allerton Conference on Communication, Control and Computing*, Univ. of Illinois, 510-517.

Bajaj, C., (1986), "An efficient parallel solution for shortest paths in 3-dimensions". In *Proc. 1986 IEEE International Conference on Robotics and Automation*, San Fransisco, 1897-1900.

Bajaj, C., (1987a), "Exact and approximate shortest path planning". In *Path Planning*, R.

Franklin, ed., SIAM, to appear.

Bajaj, C., (1987b), "On algorithmic implicitization of rational algebraic curves and surfaces", CSD-TR-681, Computer Science, Purdue University.

Bajaj, C. and M. Kim, (1987a), "Generation of configuration space obstacles I: The case of a moving sphere", *IEEE J. of Robotics and Automation*, to appear.

Bajaj, C. and M. Kim, (1987b), "Generation of configuration space obstacles II: The case of moving algebraic surfaces", CSD-TR-586, Computer Science, Purdue University.

Bajaj, C. and M. Kim, (1987c), "Generation of configuration space obstacles III: The case of moving algebraic curves", *Algorithmica*, to appear.

Bajaj, C., and M. Kim, (1987d), "Compliant motion planning with geometric models", *Proc. of 3rd ACM Symposium on Computation Geometry*, 171-180.

Bajaj,C., and M. Kim, (1987e), "Convex decomposition of objects bounded by algebraic curves", CSD-TR-677, Computer Science, Purdue University.

Bajaj, C., and M. Kim, (1987f), " Convex hull of objects bounded by algebraic curves", CSD-TR-697, Computer Science, Purdue University.

Bajaj, C., C. Hoffmann and J. Hopcroft, (1987), "Tracing algebraic curves: Plane curves", CSD-TR-637, Computer Science, Purdue University.

Bajaj, C., C. Liu, and M. Wu, (1987), "A face area evaluation algorithm for solids in CSG representation", CSD-TR-682, Computer Science, Purdue University.

Bajaj, C., and T. Moh, (1987), "Generalized unfoldings for shortest paths", *Intl. J. of Robotics Research*, to appear.

Brown, C.M., (1982), "PADL-2: A technical summary", *IEEE Comp. Graphics and Applications* 2, 69-84.

Donald, B., (1984), "Motion planning with six degrees of freedom", A.I. Tech Report 791, MIT.

Faux, I., M., Pratt, (1981), "Computational geometry for design and manufacture", Ellis Harwood, Chichester.

Farouki, R, (1986), "The approximation of non-degenerate offset surfaces", *Computer Aided Geometric Design*, **3**, 15-43.

Foley, J., A. Van Dam, (1983), *Fundamentals of interactive computer graphics*, Addison-Wesley, Reading, MA.

Griffiths, P., J. Harris, (1978), *Principles of algebraic geometry*, Wiley-Interscience Series.

Grace, W., A. Young, (1902), *The algebra of invariants*, Cambridge University Press.

Hoffmann, C., J. Hopcroft, (1985), "Automatic surface generation in computer aided design", *The Visual Computer*, **1**, 92-100.

Hoffmann, C., J. Hopcroft, (1986), "Quadratic blending surfaces", *Comp. Aided Design* **18**, 301-306.

Hoffmann, C., J. Hopcroft, (1987a), "Geometric ambiguities in boundary representations", *Comp. Aided Design* **19**, 3, 141-147.

Hoffmann, C., J. Hopcroft, (1987b), "The potential method for blending surfaces and corners", *Geometric Modeling*, G. Farin, ed., SIAM, 347-366.

Hoffmann, C., J. Hopcroft, (1987), "Simulation of physical systems from geometric models", special issue, *IEEE J. of Robotics and Automation*.

Hoffmann, C., J. Hopcroft, and M. Karasick, (1986), "Boolean operations on boundary representations of polyhedral objects", in preparation.

Kantabutra, V. and R. Kosaraju, (1986), "New algorithms for multilinked robot arms",

Journal of Computer and System Sciences, **32**, 136-153.

Kosaraju, R., (1986), "Problems on points distributed around a circle", (in preparation).

Laidlaw D., W. Trumbore, J. Hughes, (1986), "Constructive solid geometry for polyhedral objects", *Proc. of the ACM SIG-GRAPH'86 Conference*, Dallas, TX, 161-170.

Lozano-Perez, T., (1983), "Spatial planning: A configuration space approach", *IEEE Trans. on Computers*, **C-32**, 108-120.

Lozano-Perez, T. and M. Wesley, (1979), "An algorithm for planning collision free paths among polyhedral obstacles", *CACM*, **22**, 10, 560-570.

Middleditch, A., and A. Sears, (1985), "Blend surfaces for set-theoretic volume modeling systems", *Comp. Graphics*, **19:3**, 161-170.

Meagher, D., (1982), "Geometric modeling using octree encoding", *Computer Graphics and Image Processing*, **19**, 2, 129-147.

Paoluzzi, A., M. Ramella, and A. Santarelli, (1986), "Un modellatori geometrico su rappresentazioni triango-alate", Tech. Rept. 13.86, Dept. of Inform. and Systems, University of Rome.

Poppelstone, R., A. Ambler, and I. Bellos, (1980), "An interpreter for a language describing assemblies", *Artificial Intelligence*, **14**, 79-107.

Requicha, A., (1980), "Representations for Rigid Solids: Theory, methods, and systems", *ACM Comp. Surveys*, **12**, 437-464.

Requicha, A., and H. Voelcker, (1983), "Solid modeling: Current status and research directions", *IEEE Comp. Graphics and Appl.*, 25-37.

Rockwood, A., and J. Owen, (1986), "Blending surfaces in solid geometric modeling", *Geometric Modeling*, G. Farin, ed., SIAM.

Rossignac, J., and A. Requicha, (1984), "Constant-radius blending in solid modeling", *Comp in Mech. Engin.* **3**, 65-73.

Ribbens, C., (1986), *Domain mappings: A tool for the development of vector algorithms for numerical solutions of partial differential equations*, Ph.D. Thesis, Purdue University.

Rice, J., (1969), *The approximation of functions, vol. II - advanced topics*, Addison-Wesley.

Rice, J., (1986), "Adaptive tensor product grids for singular problems", *Algorithms for the Approximation of Functions and Data*, J. Mason, ed., Oxford University Press.

Sederberg, T., D. Anderson, and R. Goldman, (1985), "Implicit representation of parametric curves and surfaces", *Computer Vision, Graphics and Image Processing*, **28**, 72-84.

Thompson, J., Z. Warsi, and C. Mastin, (1985), *Numerical Grid Generation*, North Holland.

5. MATHEMATICAL SOFTWARE SYSTEMS

We first describe the current status and approach of this work. The mathematical software group (primarily Wayne Dyksen, Elias Houstis and John Rice) has developed one of the most advanced mathematical software systems, ELLPACK. We plan in the next five years to produce a much more sophisticated system with a greatly enlarged domain of applicability. ELLPACK is described and documented in the book [Rice and Boisvert, 1985]. It consists of (1) a very high level, mathematical like language for elliptic partial differential equations (PDEs), (2) an integrated library of 60+ problem solving modules and (3) a set of software tools to provide easy extensibility, portability and system evolution. It is embedded in a system for the scientific evaluation of the performance of PDE software [Boisvert, Houstis and Rice, 1979]. ELLPACK consists of 120,000+ lines of code and is being used at about 100 sites worldwide. Its current status is summarized in the report [Rice et. al., 1986]. Overviews of PDE software systems are given by [Machura and Sweet, 1980] and [Boisvert et. al., 1985], see also [Boisvert et. al., 1986]. Many of the characteristics of ELLPACK will be apparent from the discussions given below, the future evolution of ELLPACK is discussed in some detail in [Rice et. al., 1986] and [Rice, 1986].

Instead of describing ELLPACK in more detail, we present our approach to the creation and evolution of mathematics software systems. We find computational systems for PDEs an excellent vehicle for studying mathematical software systems in general. Further, we know the problem domain well, the group has written perhaps 50 papers on methods for solving PDEs and PDEs are central to a large variety of major scientific fields. As Figure 5.1 illustrates, the problem area touches all the major components found in mathematical software systems. The box "algorithm mapping" refers to the problem of mapping a computational structure (program) onto an architecture.

A useful alternative is our view of the future structure of mathematical software systems given in Figure 5.2. At the top is the user and at the bottom is an enormously powerful and varied computing environment. A central task facing computer science is to give the user full benefit of the underlying computing power. The top layers will have very large, sophisticated software systems running on powerful, graphically oriented workstations to provide access to supercomputers. The workstations will provide a large body of generic software for user interaction, graphical interfaces, text processing. Each major application area will have an expert system to aid in formulating the problem to be solved, choosing the methods to solve and analyzing the results.

Underneath this is another software layer consisting of application specific software (modules to solve particular classes of problems, to present results in particular ways, etc.) and somewhat generic mathematical software: geometric software to define and manipulate physical objects, numerical software to solve equations or do data analysis, symbolic software to handle algebraic systems or logic programming. We will specifically focus on very high software integration of a "loosely coupled" nature. This is best defined by an example, our new "PDE Solving System" could have MACSYMA running to do algebraic manipulations or differentiations, Lisp running to support the expert system, ELLPACK running to solve a PDE, SAS running to do data analysis or provide graphical summaries, I-DEAS running to allow input of complex physical structures. We do not intend to "get inside" any of these systems but rather make them communicate. There are many obvious inefficiencies in this approach, but we expect to compensate by having high powered

workstations. Once the feasibility and desirability of a particular configuration is determined, one can then turn to optimizing the system interfaces to improve performance.

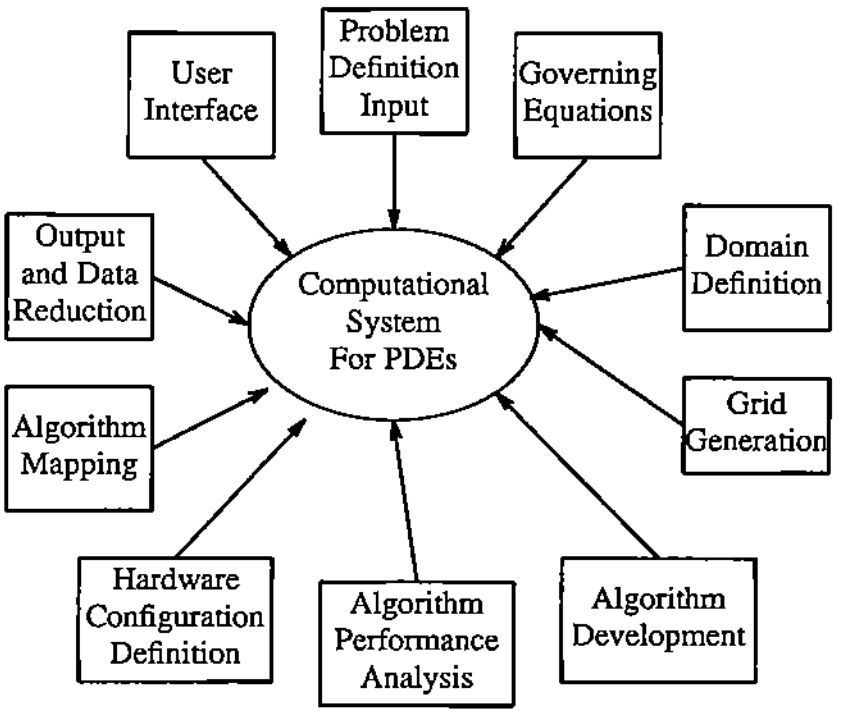


Figure 5.1. Schematic illustration of the facets of scientific computing involved in a system for PDEs.

5.1. High Level Environments for Scientific Computing

ELLPACK is a very high level language for solving a large class of elliptic problems: second order, linear elliptic PDEs in two and three dimensions with Dirichlet, Neuman, mixed or periodic boundary conditions. For example, the simple elliptic problem

$$\begin{aligned} -\nabla^2 u - 20\pi^2 u &= 0 && (x, y) \in (0, 1) \times (0, 1) \\ u &= 0 && x = 0, 1, \quad y = 0 \\ u_y &= 4\pi \sin(2\pi x) && y = 1 \end{aligned}$$

can be solved by the ELLPACK program shown in Figure 5.3.

An ELLPACK program consists of several *segments*. The elliptic problem is defined by the *equation* and *boundary* segments; these segments are declarations to ELLPACK and as such are not "executed". The remaining segments are executed from top to bottom. Figure 5.4 shows further examples of equation and boundary segments.

ELLPACK contains four basic types of problem solving *modules*. *Discretization modules* discretize the continuous problem by generating a system of linear equations. *Indexing modules* are used to order the linear system which is then solved by a *solution*

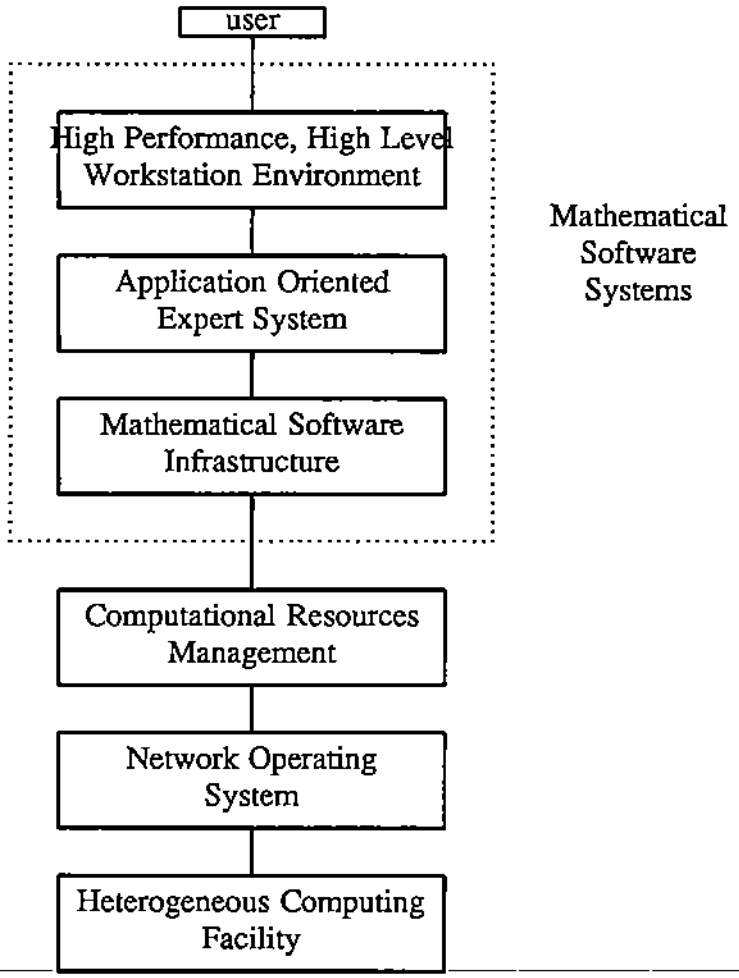


Figure 5.2. Schematic of the future organization of scientific computing. The mathematical software systems area is indicated by the dotted box.

module. *Triple modules* incorporate all three of the above steps into one module.

Interactive ELLPACK is an extension of ELLPACK. Menus of traditional ELLPACK statements can be constructed using the newly added *menu* segment. Interactive ELLPACK uses both color graphics output and graphics input to provide a sophisticated user interface.

The segments in a traditional ELLPACK program are executed sequentially, from top to bottom. In order to allow the user to specify several different methods or procedures that might be used in solving a problem, and to interactively choose from them at run time, we added a new *menu* segment to ELLPACK:

```
equation.       - uxx - uyy - (20*pi**2)u = 0

boundary.       u = 0                       on x = 0
                                            on x = 1
                                            on y = 0
                uy = 4*pi*sin(2*pi*x)       on y = 1

grid.           17 x points % 17 y points
discretization. 5 point star
indexing.       as is
solution.       linpack band
output.         max(u) % plot(u) % max(residual) % plot(residual)
end.
```

Figure 5.3. Sample ELLPACK program for solving the Helmholtz equation by ordinary finite differences and Gauss elimination.

```
menu.   '<menu name>'
        <menu item>
            .
            .
        <menu item>
```

The title for the menu is given by <menu name>. One or more <menu item>'s follow, specifying the choices to be listed in the menu. Each <menu item> is of the form:

'[<key>] : [<label>]' <item definition>

where <key> is an optional key (the user enters this to select the item at run time), and <label> is an optional name for this item. The default <key> is an integer such that the items in each menu are numbered sequentially. The default <label> is a meaningful string from <item definition>. The <item definition> may include one or more of the following ELLPACK segments: *grid, discretization, indexing, solution, triple, output, procedure,* or *fortran.* An <item definition> may extend over several lines, as long as segment names within menus do not appear in column one. Since segment names which appear outside of a *menu* segment must begin in column one, the assumption is that a *menu* segment continues until another segment name occurs beginning in column one. Every menu automatically contains three standard items: *continue* to the next menu, *return* to the previous menu, and *quit* from the Interactive ELLPACK session. In a UNIX environment, the user can escape to the shell by ''!command''. Three examples which illustrate most of the features of the menu segment are given in Figure 5.5.

In a windowing environment, <menu name> is used as the window title, and the <key>'s are used to construct pop-up menus; in this case, the default keys 1, 2,... should be avoided and meaningful ones supplied.

Sample Equations Segments

```
uxx + uyy + (1.+sin(pi*x))ux - u = f(x,y)

(p(x,y)ux)x + (p(x,y)uy)y - q(x,y)u = f(x,y)

uxx + uyy + uzz = f(x,y,z)
```

Sample Rectangular Boundary Segment

```
periodic                on x = 0
                        on x = 1
u = 0                   on y = 0
uy + 2u = g(x)          on y = 1
```

Sample Nonrectangular Boundary Segment

```
u = 0       on line 0.2, 0.0
              to 0.0, 0.4
              to 0.0, 0.6
              to 0.2, 1.0

uy = 0      on line 0.2, 1.0
              to 0.6, 1.0

a(x,y)ux + b(x,y)uy = 0 on x=1+0.4*cos(t), &
                           y=1+0.4*sin(t)  &
                        for t = pi to 3*pi/2

ux = 0      on line 1.0, 0.6
              to 1.0, 0.4

a(x,y)ux + b(x,y)uy = 0 on x=1+0.4*cos(t), &
                           y =  0.4*sin(t)  &
                        for t = pi/2 to pi

uy = 0      on line 0.6, 0.0
              to 0.2, 0.0
```

Figure 5.4. Examples of ELLPACK equation and boundary segments used to describe second order, linear elliptic PDEs in two and three dimensions with Dirichlet, Neuman, mixed or periodic boundary conditions.

```
menu.     'Discretization'
          '5ps:finite differences'              dis.  5 point star
          'hbc:hermite bicubic collocation'     dis.  interior collocation
          'hod:high order finite differences'   dis.  hodie

****************************************
*              discretization          *
****************************************

5ps : finite differences
hbc : hermite bicubic collocation
hod : high order finite differences
c   : continue
r   : return
q   : quit

menu.     'Solution'
          ':'    sol.   band ge
          ':'    sol.   linpack spd band
          ':'    sol.   sor

****************************************
*              solution                *
****************************************

1 : bandge
2 : linpackspdband
3 : sor
c : continue
r : return
q : quit

menu.     'Output'
          ':maximums'        out.    max(true)
                                     max(error)
                                     max(resid u)
          ':table u'         out.    table(u)

****************************************
*              output                  *
****************************************

1 : maximums
2 : table u
c : continue
r : return
q : quit
```

Figure 5.5. Examples of Interactive ELLPACK menu segments. Each menu segment is followed by the corresponding menu that would be produced at run time in a non-windowing environment.

ELLPACK contains a number of *output* modules which produce graphics output. If *function* is a FORTRAN function, then *plot(function)* produces a two dimensional contour plot (level curves), and *plot3d(function)* gives a three dimensional rendering of *function*. A plot of the domain (perhaps nonrectangular) along with the current grid can be obtained by *plot domain*. In standard ELLPACK a small set of plotting primitives are called to produce the plots. These routines are system dependent, but examples using well-known plotting packages such as CALCOMP or DISSPLA are provided. Most installations then provide some way of sending these plot files to a number of output devices.

Interactive ELLPACK gives the user the ability to interactively view and manipulate multiple ELLPACK plots. The interface depends on the terminal type, which is specified using the Interactive ELLPACK option *terminal*; currently, terminal can be any one of *dumb*, *tek4105*, *tek4107*, *tek4115*, or *ridge*. If the terminal type is *dumb*, graphics output is merely written to a file in the standard ELLPACK way. On a graphics terminal, the interface consists of either a fixed number of static views or a variable number of dynamic views (i.e., windows), each of which is identified by number.

On the Tektronix terminals, color graphics output is written to predefined graphics views. The number of colors and views is terminal specific. Figure 5.6 shows a complete Interactive ELLPACK program; the color pictures of Figure 5.7 (which is unlabeled) illustrates the Interactive ELLPACK interface on a Tektronix 4115.

The Interactive ELLPACK screen has ten fixed views and the resolution is good enough for all the numbers to be read easily. The second color picture is an enlargement of view 10 of the first picture, this can be displayed instantly using the ev command seen at the lower left (the text view) of the first picture. Each view is a rectangular area typically occupying some subregion of the terminal's graphic surface. Each graphic view is logically equivalent to a copy of the entire graphics surface; hence, the high level software does not need to know about the size or location of individual views. The terminals themselves translate and scale the graphics output to fit in a given view. One view is chosen as the default output view. Graphics output is stored in so-called segments which are stored in memory local to the terminal. These segments can be manipulated locally (i.e., with only a few bytes of communication from the host). Graphic output can be saved in a file to be viewed during subsequent sessions, or to be printed on a color printer. These terminals support a separate surface for dialog (non-graphics) output. This dialog area covers (transparently or opaquely) some number of views. It can be made invisible (and visible) from the keyboard so that all graphics views may be exposed.

On the Ridge display (a bit-mapped device), graphics output is displayed in windows which can be manipulated with a mouse. Figure 5.8 contains examples of an Interactive ELLPACK session on a Ridge display. One window is used for dialog; its shape and size change depending on the active menu. Menu items are selected from pop-up menus with the mouse. Graphics output is usually placed into a newly opened window, although it can be directed to any existing window. Once opened, the windows are managed by the Ridge window manager

A number of new output modules have been added to ELLPACK to implement the interactive graphics interface of Interactive ELLPACK. For example, *move view* moves the contents of one view to another. If a menu contains the item

'mv : move view' output. move view

then after entering "mv", the user will be prompted for "From view?" and "To view?". Recall that views are identified by number. Parameters for these interactive output modules can also be specified directly on the command line as in "mv 3 7". A complete list of these modules is given in Table 5.1.

Table 5.1

Interactive ELLPACK output modules for manipulating the contents of views

Module	Parameters	Effect
move view	*view1 view2*	Moves the contents of *view1* to *view2*.
copy view	*view1 view2*	Copies the contents of *view1* to *view2*.
select view	*view*	Selects *view* as the active view.
delete view	*view*	Deletes the contents of *view*.
enlarge view	*view*	Enlarges the contents of *view* so that it fills the screen.
plot grid	*view*	Plots the current grid in *view*. The plot is not part of the retained segment in the view.
overlay grid	*view*	Plots the current grid in *view*. The plot is added to the retained segment in the view.
put plot	*view file*	Puts the plot displayed in *view* into *file*.
get plot	*file view*	Gets a plot from *file* and displays it in *view*.

The ELLPACK *grid* segment allows the user to specify either a uniform grid as in

```
grid.      5 x points
           5 y points
```

or a nonuniform grid as in

```
grid.      5 x points  0.0, 0.2, 0.5, 0.8, 1.0
           5 y points  0.0, 0.2, 0.5, 0.8, 1.0
```

In the nonuniform case, it is often desirable to place the grid lines with respect to some known function such as the right side of the PDE, a residual, a trial solution or an estimate of the error. This is done both computationally and visually.

Interactive ELLPACK contains a new grid segment *interactive* which allows the user to interactively construct a grid. In a non-graphic environment, the user is simply prompted for the appropriate grid information. On graphics terminals, the grid is constructed via a graphics input device. The interactive grid module displays the domain with the current grid in the default graphics view, and prints the following menu:

```
**************************************************************************
*              INTERACTIVE ELLPACK PROGRAM                               *
**************************************************************************
options.
                max x points = 33 $ max y points = 33
                interpolation = splines
                terminal = tek4115

equation.  uxx + uyy + (20*pi**2)u = 0

boundary.  u = 0                        on x = 0
                                        on x = 1
                                        on y = 0
           uy = 4*pi*sin(2*pi*x)        on y = 1

menu.      'Solution Menu'
           'ig : interactive grid'                  grid.  interactive
           '5p : ordinary finite diffs'             disc.  5 point star
                                                    solu.  band ge
           'co : hermite collocation'               disc.  hermite collocation
                                                    solu.  band ge
           'hh : high order diffs/linpack band'     disc.  hodie helmholtz
                                                    solu.  linpack band
           'hf : high order diffs/fft'              trip.  hodie fft

menu.      'Output Menu'
           'pt  : plot true'                        out.   plot(true)
           'pt3 : plot true 3d'                     out.   plot3d(true)
           'pu  : plot u'                           out.   plot(u)
           'pu3 : plot u 3d'                        out.   plot3d(u)
           'pa  : plot abserr = abs(error)'         out.   plot(abserr)
           'pa3 : plot abserr 3d'                   out.   plot3d(abserr)
           'pr  : plot residual'                    out.   plot(residu)
           'mv  : move plot from view to view'      out.   move view
           'cv  : copy plot from view to view'      out.   copy view
           'dv  : delete plot from view'            out.   delete view
           'ev  : enlarge views'                    out.   enlarge views
           'pp  : put function plot to  a file'     out.   put plot
           'gp  : get function plot from a file'    out.   get plot
           'pg  : plot grid'                        out.   plot grid
           'og  : overlay grid'                     out.   overlay grid
           'mx  : max error'                        out.   max(error)

subprograms.
      function true(x,y)
      common  / clrvgl /  rlepsg, rlepsm, pi
      true = sin(2*pi*x) * sin(4*pi*y)
      return
      end
      function abserr(x,y)
      abserr = abs(error(x,y))
      return
      end
end.
```

Figure 5.6. Interactive ELLPACK program. Two menus are specified in this example: one with a set of problem solving modules (*grid*, *discretization*, *solution*, *triple*), and one with a choice of *output* modules.

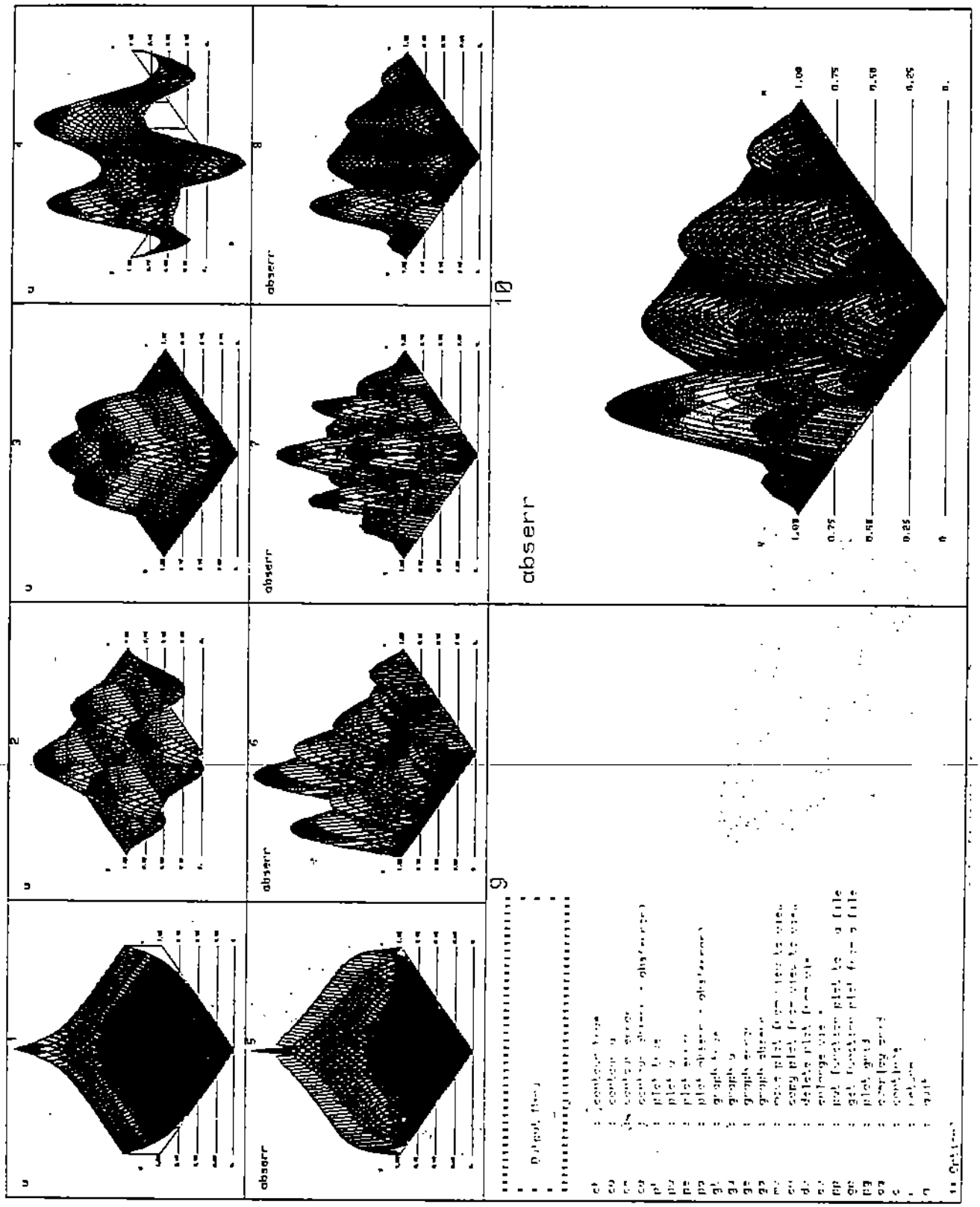


Figure 5.7.

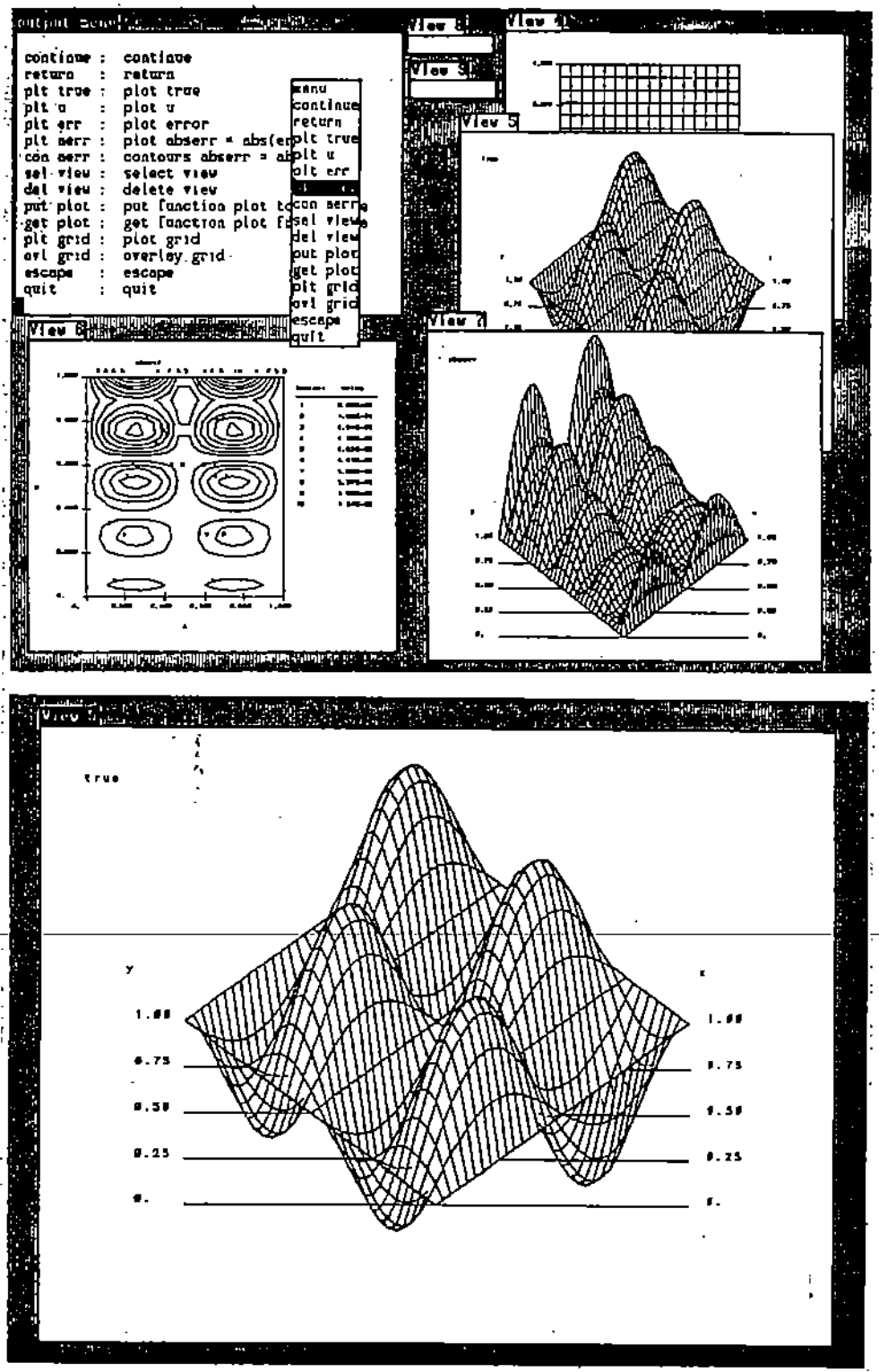


Figure 5.8. Interactive ELLPACK session on a Ridge display which supports windowing (top); individual views may be enlarged (bottom).

```
**************************************
*                                    *
*    Interactive Grid Module Commands    *
*                                    *
**************************************

c  : clear the grid and the screen
E  : enlarge view
e  : undo enlarge view
g  : get a grid from a file
h  : help
i  : input a value for a grid line
m : make the grid uniform in x or y
n  : print the number of grid lines
o  : restore original grid
p  : put a grid into a file
q  : quit
r  : redraw the screen
U : user defined via usrgrd
u  : uniform grid in x or y
v  : print the value of the nearest grid lines
x  : add    an x grid line
X : delete an x grid line
y  : add    a  y grid line
Y : delete a  y grid line
```

location of the grid line to be added or deleted is specified by a cross hair cursor which is positioned using either a joystick, thumbs wheels, or a mouse depending on the type of terminal. Typically, the grid is constructed over top of a plot of some function of interest. If a command requires input, the user is prompted for it in the dialog area. The command menu can be displayed by typing "h".

5.2. Expert System Technology Applied to Scientific Computing

For any nontrivial scientific computing problem, the selection of the "best" solution algorithm is difficult for the average nonexpert. Inferior familiar algorithms are often selected over superior unfamiliar ones in order to insure confidence in the computed results. As a step toward solving this problem, we are applying the use of Artificial Intelligence (AI) techniques to make powerful scientific computing techniques usable by nonexperts. We are performing this investigation by actually designing, implementing and testing an expert system in scientific computing. As a case study, we have begun work on *Elliptic-Expert*, an expert system for solving elliptic partial differential equations (PDEs).

Although ELLPACK contains vast "raw" PDE solving power, its potential as a problem solver is often not realized since it takes an "elliptic expert" to make full use of ELLPACK's problem solving powers. For a given elliptic problem, ELLPACK provides 1147 distinct solution paths. For the nonexpert user, choosing a valid path is difficult while

choosing an "optimal" path is nearly impossible. Most similar very high level systems share analogous drawbacks.

Elliptic-Expert is an extension of Interactive ELLPACK which advises the user in the selection of the "best" solution path (algorithm) for solving an elliptic problem. The interaction between the Elliptic-Expert system and the user includes the following areas:

1. Elliptic problem definition
   - Options
   - Equation
   - Boundary
   - Hole
   - Subprograms
2. Objectives
   - Accuracy requirements
   - Resource constraints
3. Metaknowledge
   - Elliptic problem properties
   - Solution properties
   - Reasoning strategies
4. Solution method
   - Grid selection
   - Discretization or triple module
   - Indexing/Solution module
5. Output specifications

The architecture of Elliptic-Expert is shown in Figure 5.9. Many tools and techniques exist for building expert systems [Hayes-Roth, *et al.*, 1984]. They assume the existence of a knowledge-base and a set of inference rules for the given problem domain. We already have a knowledge-base of performance of methods in over 10,000 elliptic problem solutions. A prime research effort for us is that of using our knowledge-base of the performance data plus an inference engine to create an expert system for elliptic problems.

The knowledge-base contains facts about the following three main areas:

1. Discretization or Triple Modules
   - Applicability to elliptic problem
   - Effect of grid on discrete problem
   - Convergence properties
   - Discrete problem properties
   - Resource requirements
   - Relative ranking from performance data
2. Solution Modules

- Effect of and/or need for indexing
- Applicability to discretization
- Resource requirements
- Relative ranking from performance data

3. Module Documentation

The knowledge-base also contains rules for selecting the best ELLPACK elliptic solver for a particular problem. Examples of knowledge-base rules are

1. If there is limited information about the problem and its solution, then use a robust method.
2. If the coefficient of u is much larger than those of u_{xx} and u_{yy}, then the solution may exhibit boundary layer behavior.
3. If the right side of the PDE is oscillatory, then the solution is oscillatory.
4. If the elliptic problem is the LaPlacian with Dirichlet boundary conditions, the domain is rectangular and the solution is smooth, then use Fishpak-Helmholtz.

Elliptic-Expert's inference engine has two modes. In *heuristic mode* the solution method is selected only on the basis of *a priori* knowledge; e.g., symbolic analysis of the elliptic problem and past performance knowledge. In *algorithmic mode*, dynamic strategies based on both *a priori* and *a posteriori* knowledge are used; e.g.,

1. side calculations to study the behavior of coefficients, forcing function and boundary data,
2. trial, low accuracy, cheap solutions,
3. trial solutions using the 2 or 3 "most promising" methods, and
4. additional input/advice from the user after presentation of the initial results.

The Acquisition System is used to gain knowledge-base facts and inference rules for the Knowledge Processor. This knowledge comes from three sources: the elliptic expert, the Expertise System as it learns from its actions, and the ELLPACK Performance Evaluation System [Boisvert, Rice & Houstis, 1979], [Rice, Houstis & Dyksen, 1980].

The System Supervisor gives the problem definition to ELLPACK analyzer which returns the *problem interface*—that is, the encoded elliptic problem. This interface is then passes to the Expertise System along with requests for advice and/or explanation. Using the Knowledge System, the Expertise System then makes suggestions to the user. If no performance data is available for this type of problem, the Acquisition System is requested to generate some via the Performance Evaluation System. If the problem is deemed to be interesting and different, it is added to the PDE population. After the user chooses a solution method, the System Supervisor passes the *solution interface* to the appropriate ELLPACK module which returns the solution. The Expertise System contains an explanation system to "justify" its actions.

Elliptic-Expert will serve in many roles including consulting, training and refining expert's expertise. We expect the methodology developed in building Elliptic-Expert to be applicable in other scientific problem domains.

Elliptic-Expert

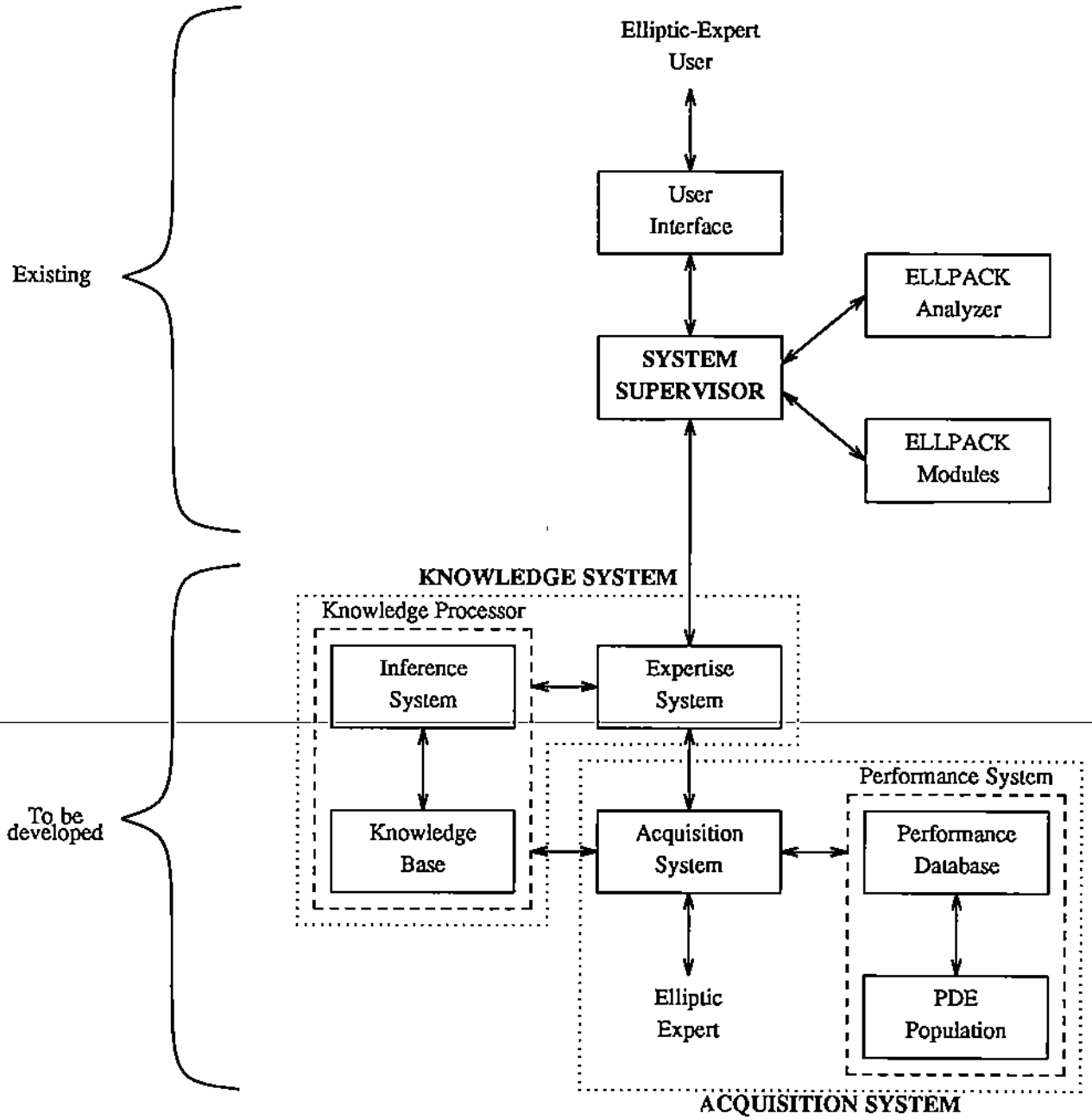


Figure 5.9. Schematic of Elliptic-Expert showing the user interface (top) and the expert system for analyzing PDE and selecting methods (bottom). Note that the Performance System exists but must be incorporated into Elliptic-Expert.

5.3. Mathematical Software Infrastructure

This area includes creating the algorithms and software for rather specific mathematical tasks (e.g., solving a certain type of PDE on a certain type of region, mapping a 2D domain with five sides onto a rectangle, approximating a set of scattered data points by a polynomial surface). There are literally hundreds of such items needed in a mature system for computing about physical objects. We have these approaches to building this infrastructure:

(1) Incorporate large sets of existing software using the high level integration technique.

(2) Create ourselves a certain number of items critical to the next generation of systems.

(3) Do without many items that are less critical, especially for an experimental system.

Note that the large size (100,000+ lines of codes) of the existing ELLPACK system is due primarily to its rich infrastructure.

Geometry Infrastructure.

We will concentrate on three areas: 3D objects, multiple domains and mappings or grids. We have produced software [Rice, 1984] for the numerical processing of general 2D objects. We now know how to modify this work to apply to 3D objects with surfaces either defined implicitly (i.e., $f(x, y, z) = 0$), explicitly (i.e, $z_i = g_i(x, y)$, $i = 1, 2, \cdots, p$) or parametrically (i.e., $x = h_1(s, t)$, $y = h_2(s, t)$, $z = h_3(s, t)$). This will provide the interface with the geometric modeling project. Knowing how is still a long way from accomplishing, only those who have implemented geometric algorithms know how difficult it is to obtain reliable software for seemingly straight forward tasks.

We will also attempt to use existing systems for 3D objects. We do not expect much trouble when the entire computation can be carried out within such a system, we do expect problems in using the 3D geometry capability separately because of complications in determining and accessing the data structures used internally.

Multiple domains are needed because so many applications involve more than one domain. It is only work to extend the existing system to do most of the geometric processing of multiple domains. There are, however, serious research challenges in handling the interactions and interfaces between domains. The geometric portion of this challenge is addressed by the geometric modeling project. We are still left with the problems of creating algorithms for handling PDEs with general interface conditions or with interfacing domains where unrelated PDEs model the physical processes. We will create some such algorithms using the collocation approach, see below.

Domain mappings are a key tool in handling general geometry. Note that gridding or mesh generations may be viewed as simply mapping a domain with a very regular mesh onto the domain of interest. An extensive and recent survey of this area is given by [Thompson et. al., 1985], we have successfully used this approach recently [Ribbens, 1986], [Rice, 1986] to handle a wide class of problem difficulties and believe that further work will be fruitful in this area.

Domain of PDE Applicability

We will extend our existing systems for PDEs in three directions: allowing time dependent problems, allowing systems of PDEs and allowing nonlinear PDEs. This effort consists of two steps. First the language and preprocessing system must be extended. We have known how to do this for many years but, until now, have not had sufficient motivation to do it. Second, we must obtain the software modules (the infrastructure) to solve the PDEs newly admitted to the domain of applicability. We want to make one point clear: **we do not claim that we will produce a PDE solving system that will handle all nonlinear, time dependent systems of PDEs on multiple domains**. This is impossible now. We do claim that we will produce a system that will handle a wide range of such problems, including many of practical interest. We have already solved such problems within our existing framework (see [Rice and Boisvert, 1985], Chapter 5). We will extend these ideas but we will depend more heavily on using other PDE solving systems and software. Of particular interest are DSS (Differential Systems Simulator) [Schiesser, 1982] and PDE PROTRAN (formerly TWODEPEP) [IMSL, 1985] which are quite general in their applicability. Smaller, but still substantial, sets of PDE solving software are the ACM algorithms: 540: PDECOL and 565 PDETWO/PSETM/GEARB by D.K. Melgaard and R.F. Sincovec. Finally, we intend to use one of the standard structural engineering systems to extend our capability to the fourth order PDEs that arise in stress and strain models.

Symbolic Systems

Symbolic computation has two rather distinct aspects: algebra (polynomials, derivatives, integrals, etc.) and logic (Lisp Prolog, Smalltalk, etc.). The applications oriented expert systems will use symbolic systems of the logic flavor. We also need the algebraic capabilities, first for the algebraic geometry computations for geometric modeling and second for symbolic methods to solve PDEs. For example, we believe that symbolic methods will be much more efficient than numerical methods for solving LaPlace's equation on general 2D or 3D domains. One expands the solution in a harmonic series (generated symbolically) and then satisfies the boundary conditions by a least squares fit (computed numerically). This old idea [Davis and Rabinowitz, 1961] has not given a fair test because of the unfortunate separation that has existed between numerical and symbolic computations. We will be able to implement this and similar methods within the environment we will create.

Numerical Algorithms

We have already mentioned that new algorithms will be needed to handle PDE interfaces between domains. We have had good success in using collocation methods (see [Houstis et. al., 1985], [Houstis et. al., 1985a], [Houstis, Vavalis and Rice, 1986], [Houstis, Christara and Rice, 1986] for recent work). This area is still at the frontier of numerical methods for PDEs (see [Birkhoff and Lynch, 1984] in Section 7.10) and we expect to make contributions here and to discover algorithms that are both reliable and reasonably general.

We will also develop algorithms tailored to 3D problems. While many 2D methods extend directly, they often have less efficiency than possible when one takes explicit advantages of the 3D nature of the problem. We will work with Robert E. Lynch in using high order finite difference methods (see [Lynch and Rice, 1978], for example) in 3D. We will also continue our one of collocation methods.

Finally, we will develop parallel algorithms for PDEs as part of the parallel processing project discussed in the next section.

5.4 References

Basden, A., "On the application of expert systems", *Int. J. Man-Machine Studies*, *19*(1983), 461—477.

Birkhoff, G. and R.E. Lynch, (1985), *Numerical solutions of elliptic problems*, SIAM Publications, Philadelphia.

Boisvert, R.F., S.E. Howe, and D.K. Kahaner, (1986), "GAMS: A framework for the management of scientific software", *ACM Trans. Math. Software*, 12, to appear.

Boisvert, R.F., S.E. Howe, and D.K. Kahaner, (1985), "GAMS: Guide to available mathematical software", Technical Report, Nat. Bur. Standards.

Boisvert, R.F., E.N. Houstis, and J.R. Rice, (1979), "A system for performance evaluation of partial differential equations software". IEEE Trans. Software Engineering, *5*, pp. 418–425.

Coombs, M.J., [ed.], *Developments in expert systems*, Academic Press, Orlando, 1984.

Davis, P.J. and P. Rabinowitz, (1961), Advances in orthonormalization computation. In *Advances in Computers*, Vol. 2 (F.Alt, ed.).

Duda, R.O., P.E. Hart, K. Konolige, and R. Reboh, (1979), "A computer-based consultant for mineral exploration", Technical Report, SRI International,

Dyksen, W.R. and C.J. Ribbens, (1986), "Interactive ELLPACK: An interactive problem solving environment for elliptic partial differential equations", CSD-TR 588, Department of Computer Sciences, Purdue University.

Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, eds., (1983), *Building Expert Systems*, Addison-Wesley, Reading, Massachusetts.

Houstis, E.N., E.A. Vavalis and J.R. Rice, "Convergence of an $O(h^4)$ cubic spline collocation method for elliptic partial differential equations", submitted.

Houstis, E.N., W.F. Mitchell, and J.R. Rice, (1985), "Collocation software for second order elliptic partial differential equations", *ACM Trans. Math. Software*, *11*, pp. 379–412.

Houstis, E.N., W.F. Mitchell, and J.R. Rice, (1985a), "Algorithm 638 GENCOL: Collocation on general domains with bicubic Hermite polynomials", *ACM Trans. Math. Software*, *11*, pp. 416–418.

Houstis, E.N., C.C. Christara and J.R. Rice, "Quadratic spline collocation methods for two point boundary value problems", submitted.

"IMSL", (1985), TWODEPEP (Two dimensional, elliptic, parabolic and eigenvalue problems), Houston.

Lynch, R.E. and J.R. Rice, (1978), "High accuracy finite difference approximation to solutions of elliptic partial differential equations", *Proc. Nat. Acad. Sci.*, *75*, pp. 2541–2544.

Machura, M. and R. Sweet, (1980), "A survey of software for partial differential equations", *ACM Trans. Math. Software*, *6*, pp. 461–488.

McDermott, J. "R1: A rule-based configurer of computer systems", *Artificial Intelligence*, *19*(1982), pp. 39–88.

Mehta, U.B. and P. Kutler, (1984), "Computational Aerodynamics and AI", NASA

Technical Memorandum 85994,

Michie, D., "Expert Systems", *Computer Journal*, *23*(1981), 369-375.

Ribbens, Calvin, J., (1986), "Domain mappings: A tool for the development of vector algorithms for numerical solutions of partial differential equations", Ph.D. thesis, Purdue University.

Rice, John, R., (1986), "Adaptive tensor product grids for singular problems". In *Algorithms for the Approximation of Functions and Data*, (J. Mason, ed.) Oxford University Press, Oxford.

Rice, J.R., W.R. Dyksen, E.N. Houstis, and C.J. Ribbens, (1986), ELLPACK status report. CSD-TR 579, (31 pages).

Rice, J.R., (1986), "ELLPACK: An evolving problem solving environment". In *Problem Solving Environments for Scientific Computing* (B. Ford, ed.) North-Holland, to appear.

Rice, J.R., and R.F. Boisvert, (1985), *Solving Elliptic Problems Using ELLPACK*, Spring Verlag.

Rice, J.R., (1984a), "Numerical computation with general two dimensional domains". *ACM Trans. Math. Software*, *10*, pp. 443–452.

Rice, J.R., (1984b), "Algorithm 624: A two dimensional domain processor". *ACM Trans. Math. Software*, *10*, 453–562.

Rice, J.R., E.N. Houstis, and W.R. Dyksen, (1981), "A population of linear, second order, elliptic partial differential equations on rectangular domains, Parts 1 and 2", *Math. Comp*, *36*, 475–484.

Schiesser, W.E., (1982), "DSS/2: Differential systems simulator, Version 2". Dept. of Computer Science, LeHigh University.

Shortliffe, E.H., (1975), *Computer-Based Medical Consultations: MYCIN*, American Elsevier, New York,

Thompson, J.F., Z.U.A. Warsi, and C.W. Mastin, (1985), *Numerical Grid Generations*, North-Holland, New York.

6. PARALLEL PROCESSING

The design and analysis of algorithms for the next generation of parallel computers requires attention to many factors including:

(i) parallelism detection, representation and exploitation.

(ii) partition and allocation of computational objects based on the "physical" structure of the application and target architecture.

(iii) performance evaluation of algorithm/architecture pairs.

In the next five to ten years we will see the wide spread use of the following computer organizations:

(i) parallel MIMD systems with hundreds of processors of *k* Million Instructions Per Second (kMIPS)

(ii) parallel MIMD systems with tens of processors of Billion Instructions Per Second (BIPS)

(iii) heterogeneous networks of supercomputers

The fundamental research project that we try to address is:

How does one create a system where parallel algorithms can be designed and specified in a reasonable time and which provides good implementation mappings to any of the three architectures above?

Our work concentrates on the following components of the fundamental problem:

A. Develop algorithms for partitioning and allocating PDE computations for the these machines.

B. Create new algorithms of an inherently parallel nature especially for physical objects modeled by PDEs.

C. Provide a software environment for designing algorithms for targeted architectures.

D. Study the performance of parallel algorithms in various computer architectures.

E. Develop a distributed PDE-Expert system for heterogeneous computing environment.

F. Provide a high level man-machine interface.

The above components of research are described in more detail in the following sections.

6.1. Performance Analysis of Algorithm/Architecture Pairs

We are interested in the performance of PDE and Geometry processing algorithms in two important classes of computer systems: BIPS (Billion Instructions Per Second) systems consisting of several very high performance processors and kMIPS (k Million Instructions Per Second) systems with hundreds of low speed processors. Several models of computation will be considered based on BIPS and kMIPS processor elements. We believe that computing environments in the 1990 time frame will contain elements of both systems.

Detailed simulation of such systems is intractable. Simple approximate models will be utilized for estimating the performance of such systems. The NCUBE will be used to verify kMIPS models for non-shared memory architectures. A simulator will be developed on the FLEX multicomputer at Purdue to simulate BIPS systems with large shared semiconductor memories.

For such systems there are no existing application programs to use as benchmarks. We propose to create a number of these using algorithms from PDE and Geometry modeling applications.

The first step in performance evaluation is the careful determinating of a set of metrics. Evaluation metrics can be classified as computational and programming (or software). Metrics for computation are related to hardware performance and software metrics try to assess the productivity and usability of various hardware/software resources. The metrics for parallel programming software are in an exploratory stage. We plan to study them in collaboration with the Software Engineering Research Center (SERC) at Purdue.

6.2. Parallel Algorithms

We propose to examine PDE and Geometry modeling applications on BIPS and kMIPS multiprocessor systems. The Purdue group in PDEs (W. Dyksen, E.N. Houstis, R.E. Lynch and J.R. Rice) have extensively studied a subclass of finite element methods known as collocation methods and a subclass of high order finite difference methods (HODIE) for elliptic PDEs. One of the features of these methods is that the corresponding discretization process is fully parallel. In [Houstis, Houstis, Rice 86] we have explored the feasibility of FLEX32 multiprocessor system for solving discrete spline collocation equations. The techniques used to partition the corresponding computations are based on the partition of data structures involved. The results obtained indicate that optimal speedup is possible for large problems. A number of techniques based on domain decomposition are currently explored for solving the discrete collocation and HODIE type models on a variety of existing architectures. In order to simulate and verify their behavior in future multiprocessor systems of BIPS and kMIPS type it is essential to have a facility like NCUBE (256 processors). We plan to place greater emphasis on local iterative methods for 2D and 3D collocation and HODIE computational models.

We also will examine more fundamental questions about parallel algorithms and explore the practical application of recent advances in our theoretical understanding of them. For example, we [Kosaraju 86, 86a] have obtained new results about algorithms for trees of processors. We can explore the performance of such algorithms on the NCUBE (a tree can be embedded in its structure easily) or consider them for use on the hypothetical multi-FLEX machine [Rice, 86].

We have recently obtained striking results [Atallah, Kosaraju 86] about simulating one network of processors by another. This is an abstract form of the problem of mapping a computational structure onto an architecture structure. We will explore the possibility of using this approach for the mapping problems.

Also, a study into the feasibility of using the NCUBE system for grid generations will be undertaken. Careful consideration of the graphical input and output will be given and the high performance NCUBE graphics system will be exploited.

6.3. Partitioning and Mapping of Large Scale Engineering and Science Systems to Parallel Machines

Many engineering and science problems involve two or more distinct subsystems (models) that are tightly coupled. For example, a computational system for studying the weather usually comes from the interactions of many models with different scales in both time and space. Other examples are provided by fluid-structure, soil-structure, thermal-structure and structure-magnetodynamic interactions. This is centered in the following four areas:

Partitioned analysis of coupled systems

The tight interaction of coupled models (subsystems) is a common characteristic of many of these problems. The interaction between the components means that the response of the overall system must be calculated concurrently. The current approach for handling most coupled problems is the 'monolithic' expansion of existing computer programs to handle more interaction effects. The availability of multiprocessor computer systems, the uncontrollable complexity and lack of flexibility of large-scale sequential software systems justifies the study of partitioning of such systems and how to distribute their processing. We intend to study and apply various partitioned analysis procedures to the problem of weather phenomenon.

Partitioned analysis of single subsystems

We are witnessing major changes in the architectures of the general use computer systems. Experience has shown that such drastic hardware changes require major changes in the way we design and write algorithms in order to achieve promised performance levels. Some of the primary goals in the design of algorithms for parallel processors are (1) minimization of the execution time, (2) minimization of the global memory access, (3) minimization of communication between processors, (4) identification of sufficient concurrency and (5) load balancing.

We propose to study partitions of elliptic and time dependent PDEs so as to satisfy the above objectives. These partitions will be based on mathematical properties of governing equations and some discretization models based on finite element and finite difference techniques. The techniques of substructuring and Schwarz splitting [Wern, 1963], [Tang, 1986] will be exploited and various iterative techniques will be analyzed for handling the interactions across the interior interfaces due to substructuring.

Mapping partitioned application to parallel machines

Supercomputers are essential for large scale scientific and engineering computations but their usage is difficult for average scientist and engineer. We propose to develop intelligent software tools which are able to analyze a concurrent application program and map it into a given system. For this discussion we assume tightly connected parallel systems. One of the allocation methodologies to be tested is the methodology of Houstis/Rice [Houstis, Houstis, Rice 86]. The precedence graph of the application is identified and the computational and communication requirements are determined. Then the application's graph is mapped to the corresponding architecture graph provided certain parameters that characterize the architecture are given. Our current implementation is able to analyze a concurrent C or Fortran application programs and generated the data to a heuristic mapping

algorithm [Houstis, Houstis, Rice 84]. Figure 6.1 depicts the application which is an instance of a collocation program for PDEs. Figure 6.2 is the precedence graph of the application generated automatically and Figure 6.3 is the mapping of the application graph to a shared memory multibus architecture. We will also explore the use of recent advances of exact algorithms for such allocation problems. Currently we are developing a knowledge base with various analytical models that characterize different architectures. We plan to advance the stage of this system and make it available as an intelligent front end to the FLEX32 and NCUBE multiprocessor systems.

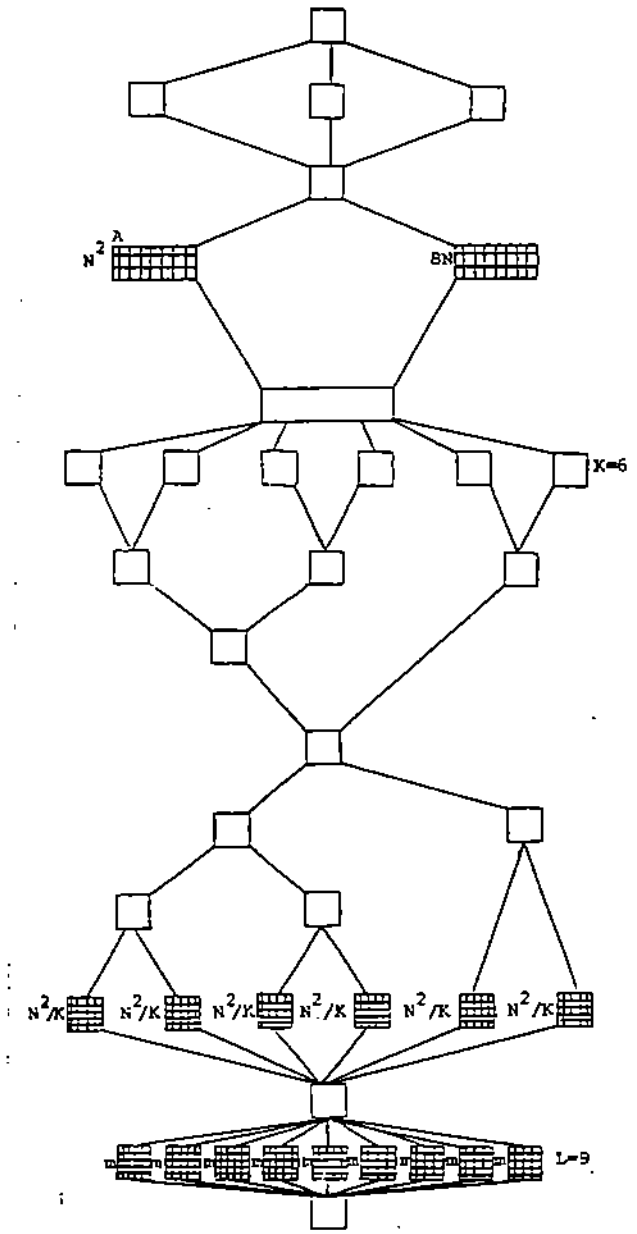
6.4. Heterogeneous Distributed Computing

Above we considered a tightly coupled homogeneous parallel machine. Now we discuss determining an optimal schedule in a heterogeneous environment. We recognize that the cost of producing optimal schedules for realistic computations is exorbitant. Our distributed computing facility will be used by three major Purdue projects to explore the use of heuristics for scheduling when optimum solutions cannot be found. These heuristic solutions can be classified by the binding time at which the heuristics are applied:

- BLAZE: Programming Environment for Multiprocessing.
  Heuristics applied at compile time assume that much is known about the algorithms and potential architectures, but that nothing is known about the input data. They attempt, for example, to recognize and exploit implicit parallelism. They assume complete control over the hardware on which the computation executes (i.e., they do not consider loads introduced by other computations).

- DE2: A Problem Solving Environment for a Scientific Application
  Heuristics applied at program load time assume that much is known about the algorithm and architecture, and that the sizes of the computation components are estimable. They schedule the computation components on the machines, attempting to minimize the processing time. The heuristics may, for example, schedule a sequence of matrix manipulation steps, perhaps with some steps performed in parallel on multiple machines. Again, the heuristics optimize only for the computation and architecture that is specified; they do not consider loads introduced by concurrently executing programs.

- TILDE: Control in a Loosely Compiled Distributed Environment.
  Heuristics applied at execution time consider the set of resources requested by computations and dynamically balance load on a set of machines. Such heuristics are closely related to the usual notions of scheduling and load balancing in a distributed operating system. Typically, execution time heuristics know little about the performance of a given algorithm, but they do understand the capabilities and loads of processors on the system. To minimize delay, tasks are assigned to those processors that (a) are suitable for the computation, and (b) have the least load among all suitable processors, where the "load" is normalized by the capacity of the machine. Note that such heuristics allow multiple independent processes to compete for processors because the scheduler makes



Text: 2000 Chars

P * [V: N*8, F: N*80, A: N*300]

V: N*8, A: P*20
A: N^2 * 25, Mat: N^2

N^2 * [F: 40, V: 28, Mat: 16x16, A: 1000]
/8N * [F: 15, V: 16, Mat: 16x16, A: 1800]

T: K^2 + 4K, A: 10K^2 + 300K

K * [Mat: N^2/Kx64N/K, V: N/K * 2,
A: $N^6/K^3/2 + N^4/K^2$]

K/2 * [Mat: $2N^2$/Kx64N/K, V: 2N/K * 2,
A: $N^6/K^3/16 + N^4/K^2$]

1 * [Mat: $2N^2$/Kx64N/K, V: 4N/K * 2,
A: $N^6/K^3/16 + N^4/K^2$]

1 * [Mat: $2N^2$/Kx64N/K, V: N * 2
A: $N^6/K^3/16 + N^4/K^2$]

2 * [Mat: $2N^2$/Kx64N/K,
V: 4N/K * 2, A: N^4/K^2]

K/2 * [Mat: $2N^2$/Kx64N/K,
V: 2N/K * 2, A: N^4/K^2]

K * [Mat: N^2/K^2x(64/N/K), V: N/K*2, A: $(N/K)^4$]
K * [Mat: J^2, F: J^2 * 120]
K * [Mat: J^2, F: J^2 * 300]

T: 81 * L^2, A: KJ^2 * 20

L * [T: m * 2, A: m * 8 + 500]

Figure 6.1. Graphical representation of a complex computation. The annotations are encoded values of the memory and computation required at each node; another set of annotations shows the communication requirements between the nodes.

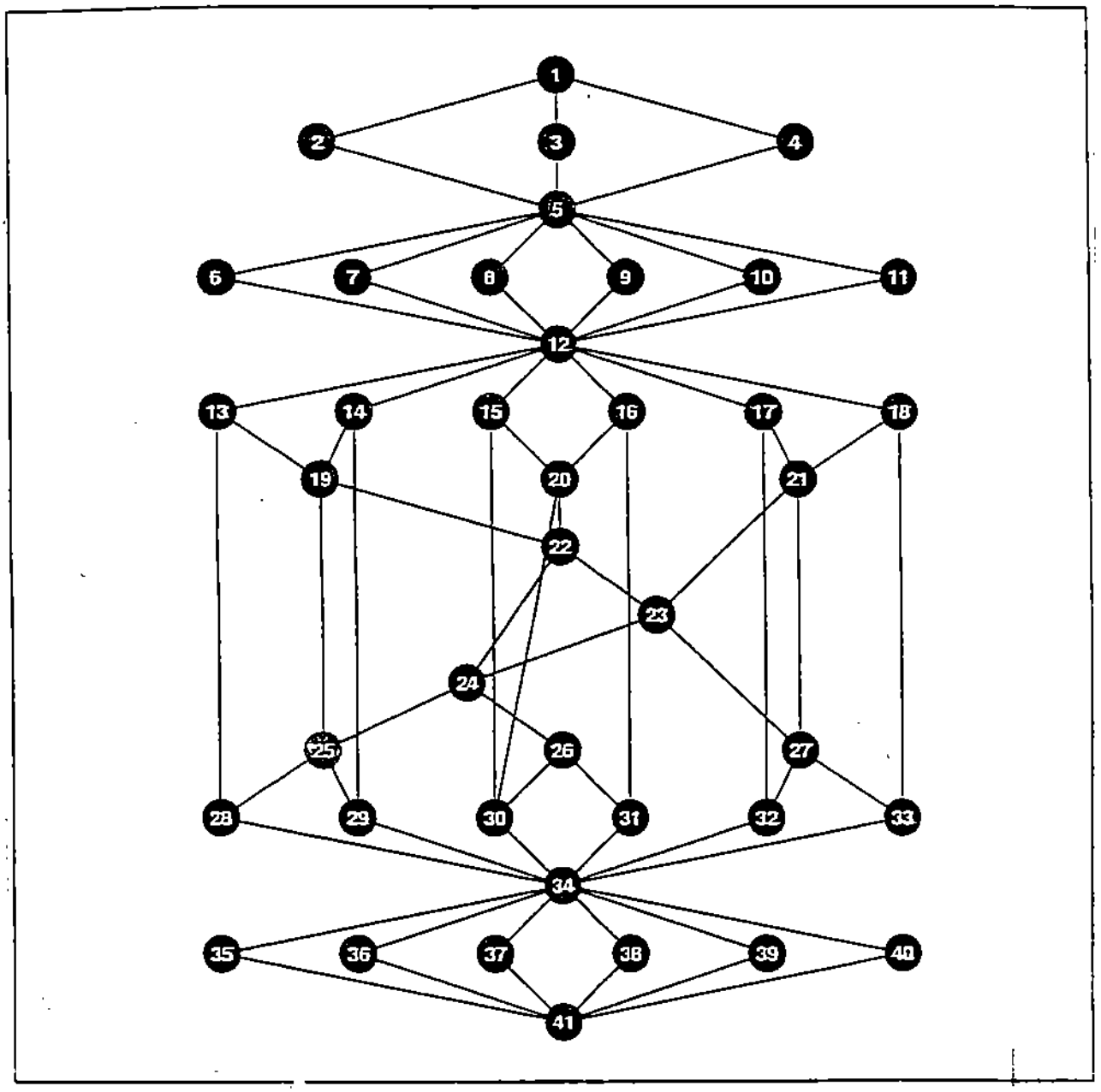


Figure 6.2. The precedence graph of application of Figure 6.1

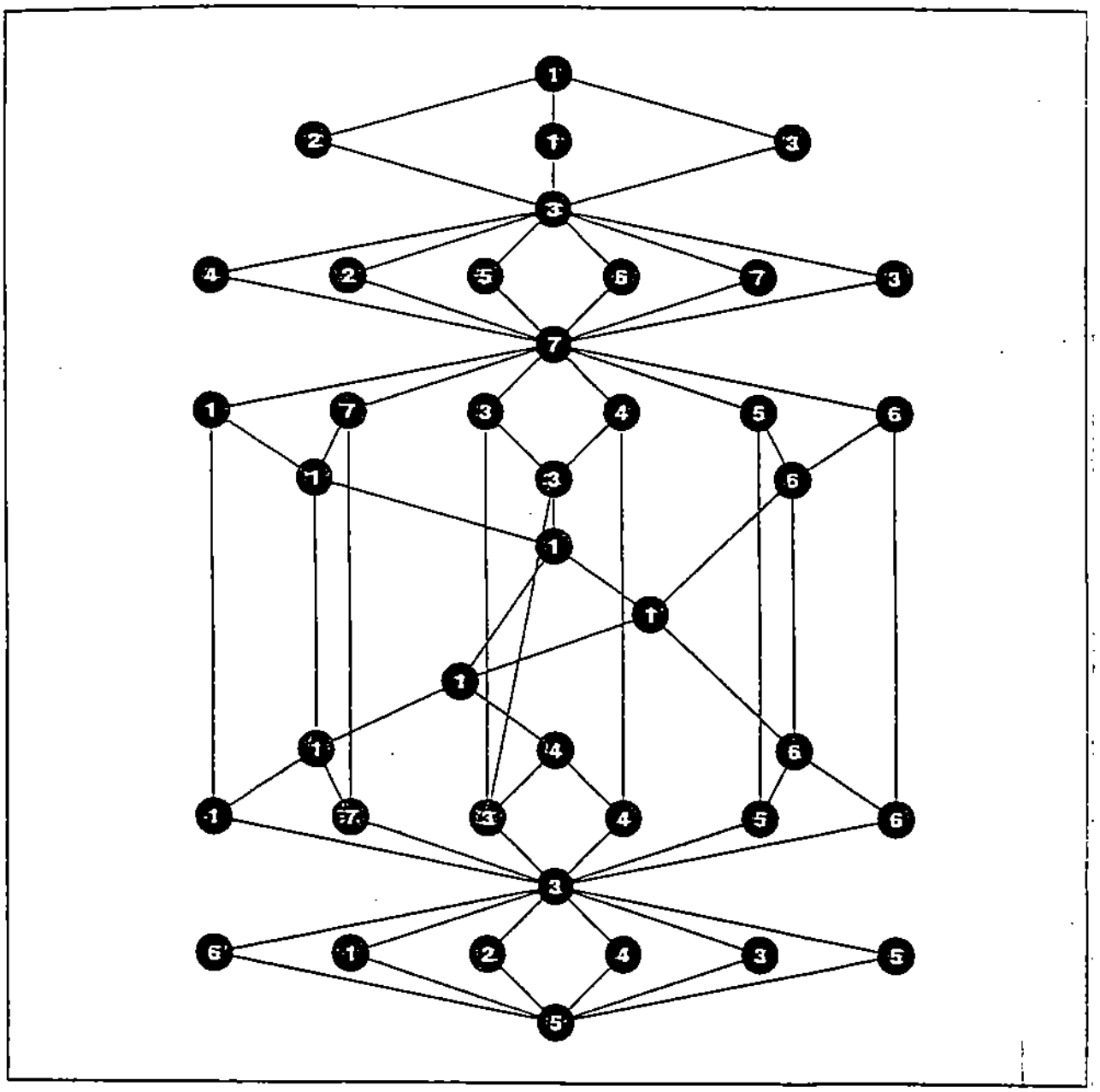


Figure 6.3. The mapping of the application graph (Fig. 6.2) to the FLEX32 system. The numbers at the node indicate the processor to execute that node's code.

dynamic decisions.

6.5. Distributed Elliptic-Expert (DE2): A Problem Solving Environment (PSE) for Elliptic PDEs.

This project is a direct continuation of the Elliptic Expert project described briefly in Section 5.2. Figure 6.4 shows the expansion of the DE2 system from the Elliptic-Expert system shown in Figure 5.9. Our distributed computing facility attaches to the system supervisor of Figure 5.9. Without further generalities, we describe our approach to key problems: construction of an *architecture compiler* for a problem solving environment.

At the end of its analysis, the PSE knows a great deal about the computation; its size, its input/output, the software modules to be used, etc., as well as the heterogeneous network facility. Assume that this information is represented in a quantified form such as annotated graphs. One then has to map the computation graph into the architecture graph subject to all the constraints on size, sequencing and user specifications.

Once a machine is selected (e.g., a multiprocessor) for a computational module further restructuring is feasible. The PSE has not yet created the load module for execution, so it can restructure the module to fit the architecture better. An obvious, but very effective, tactic is as follows: we have a submodule that can do K things, we have N^2 of these things to do and P processors. We create a load module with P copies of the submodule, each assigned to one processor and each doing N^2/P of the things. Figure 6.1 shows a realistic example of an annotated graph for a DE2 application. The shaded boxes represent replicated modules that can be grouped in any convenient way.

Our operational plan is as follows. We work with relatively coarse grain structures so the graphs are of reasonable size. We translate all this information into a mathematical optimization problem; it will be mostly a linear program with perhaps a few nonlinear constraints and a nonlinear objective function. We apply a fast heuristic algorithm to obtain a good - probably not optimal - solution of the optimization problem. This solution is then used to create the load module for the PSE run.

One can visualize from this description of DE2 that there are many specific technical problems to be solved. To keep the presentation brief, we list only eight of these.

Technical Problems:

1. *Obtain the module structure and resource use data.* We want more "coarse" structure and data that one obtains from a complete analysis of a program [Rice 84]. We plan on software tools to help extract information and simplify programs (e.g., count the variables declared, reduce statements to arithmetic counts, ignore "small" loops or branches). It is premature to automate this completely and we also will use human interaction. Most submodules will reduce to one annotated node in the graph.

2. *Parallelization of problem solving modules.* We [Houstis, Houstis Rice 84], [Rice 85], and many others [Takuhashi 82] have studied recasting methods so they are easily adapted to parallel computation. We have already parallelized some of the DE2 modules. We find that automatic approaches give helpful information but that human

analysis is essential for really good results.

3. *Obtain the machine structure and resource capacities.* We will obtain this information by hand for our network and machines.
4. *Build the resource allocation problem.* A particular run combines a fair number of modules whose separate structure is known. These must be combined into one resource allocation problem. We will automate this process.
5. *Model the communications nonlinearities.* In a multiprocessor or multimachine environment the communication capacities behave nonlinearly as saturation is neared. Since this is the interesting range of operation, this nonlinearity must be modeled. We will measure this behavior ourselves for the FLEX32 and for intermachine communication on the heterogeneous network.
6. *Solve the resource allocation problem.* The normal case will be to have only a few hundred constraints and a nonlinear objective function. The problem is too big to use time consuming standard optimization techniques, yet it is not a huge problem. We [Houstis, Houstis, Rice 84] and others, [Gylus Edwards 76] have developed fast heuristic methods for these problems. Our initial experience is that the work to obtain a "good" solution heuristically grows linearly with the size of the problem. Our heuristic algorithm will be developed further to make it robust and more widely applicable.
7. *Architecture compiler: Dynamic software reconfiguration and program transformations.* Out long term goal is not only to map the computations onto the architecture, but to dynamically reconfigure the software to fit better on the architecture. We will operate at the procedure level (submodules).
8. *Develop performance estimators.* In selecting machines we need to be able to make a prior; estimates of routines based on the representations we have of the computations and machines [Rice 82]. Performance prediction on multiprocessors is currently one of the leading open questions in high performance computations; we place high priority on this problem and believe that the information we gather is the proper basis for progress. Extensive experimental work is planned to calibrate these estimators.

Distributed PDE-Expert

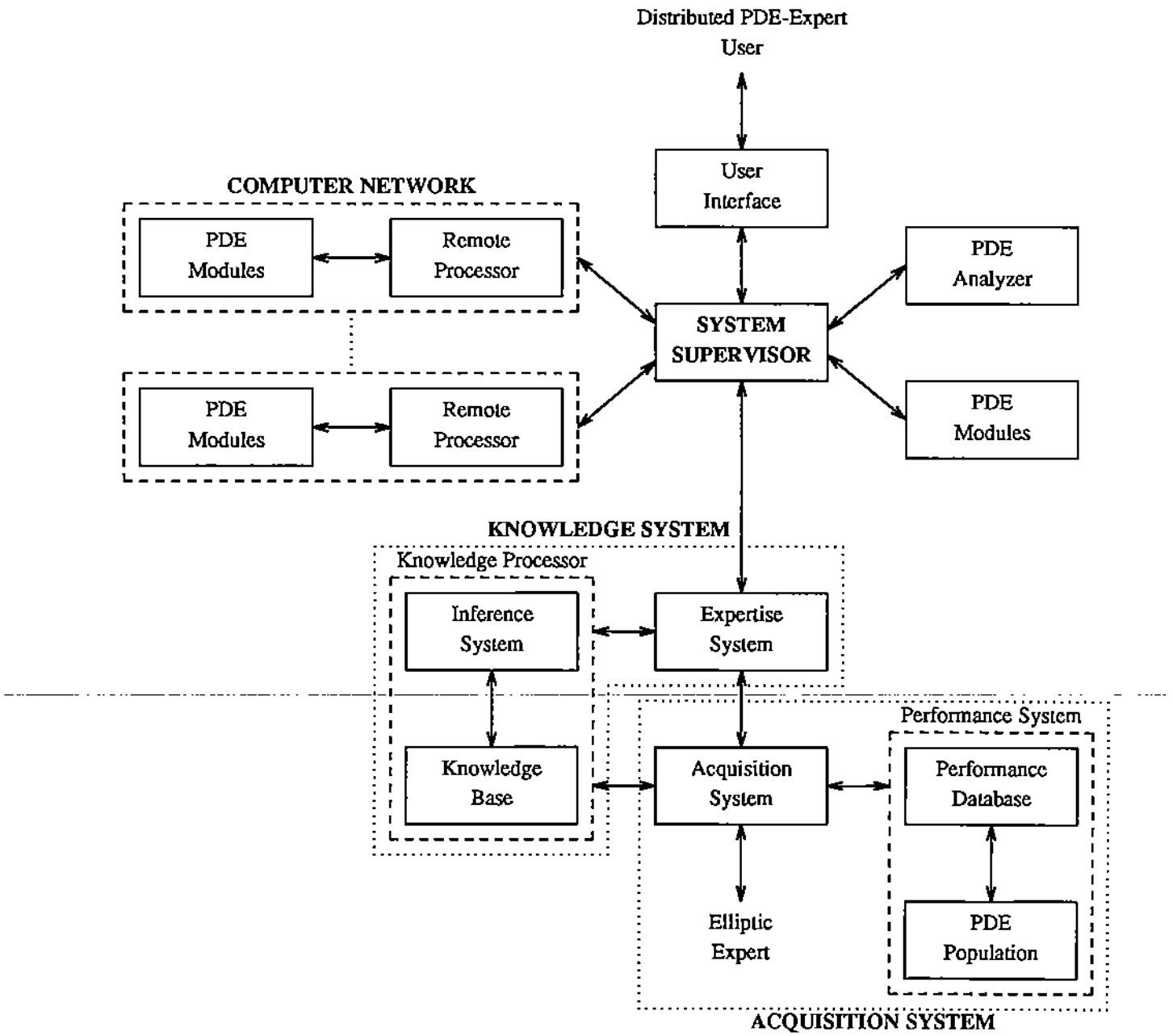


Figure 6.4. Schematic of DE2 showing the user interface (top right), the expert system for analyzing PDEs and selecting methods (bottom) and the connection to the computer network.

Distributed Hardware Facility

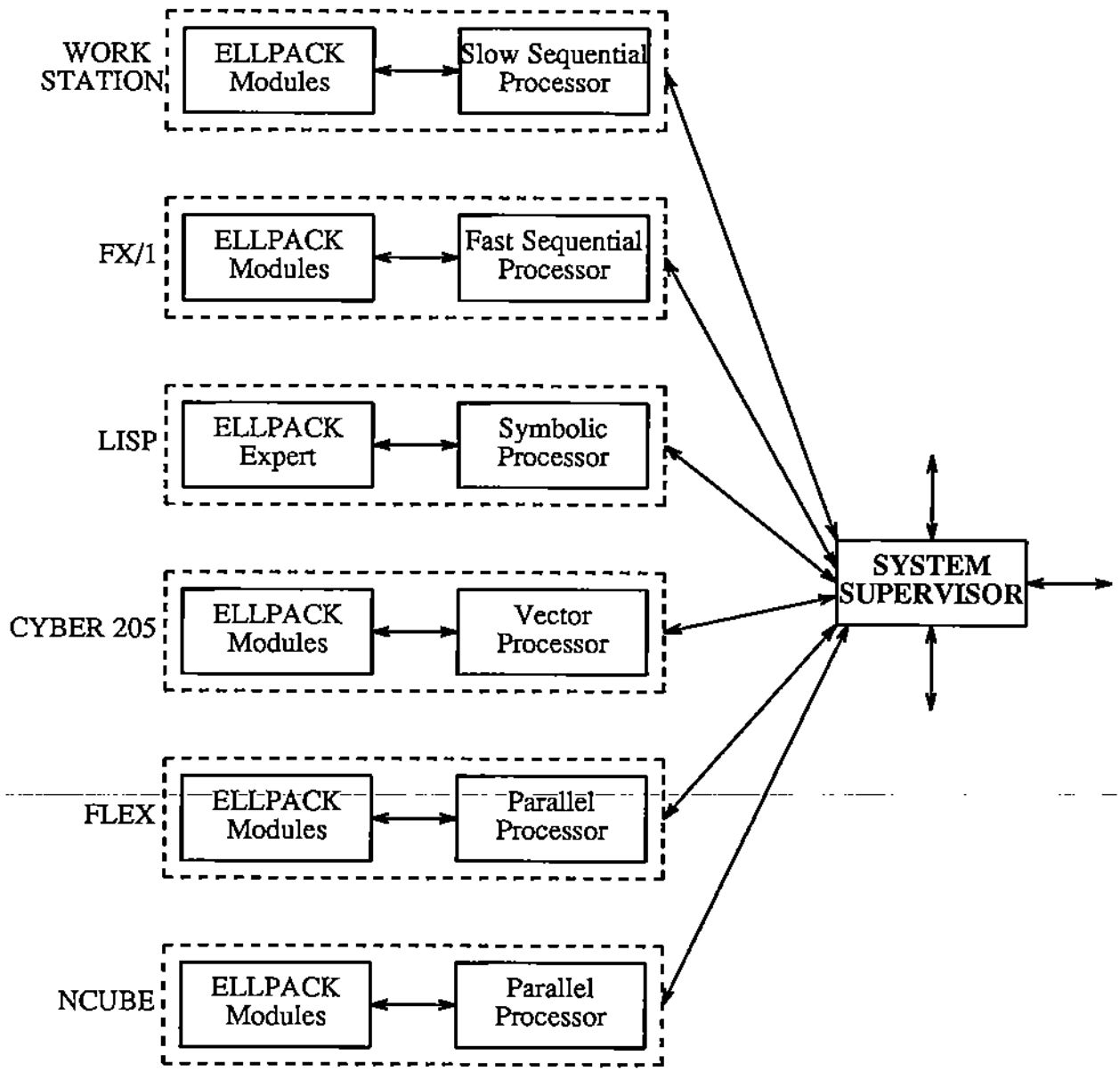


Figure 6.5. Schematic of the distributed computer facility as attached to the PDE-Expert system. The second expert system will analyze the computational requirements, help select appropriate machines and map the computation onto them.

6.6. References

Atallah, M.J. and S.R. Kosaraju, (1986), "Optimal simulation between mesh-connected arrays of processors". preliminary report.

Berman, F., M. Goodrich, C. Koelbel, W.J. Robinson, III, and K. Showell, (1985), "Prep-p: A mapping preprocessor for chip computers," *Proc. Inter. Conf. Parallel Processing*, 731–733.

Berman, F. and L. Snyder, (1984), "On mapping parallel algorithms in parallel architectures," *Proc. Internat. Conf. Parallel Processing*, 307–309.

Birkhoff, G. and R.E. Lynch, (1984), *Numerical solution of elliptic problems*, SIAM, Philadelphia.

Boisvert, R.F., E.N. Houstis and J.R. Rice, (1979), "A system for performance evaluation of partial differential equations software," *IEEE Trans. Software Eng.*, **19**, pp. 418–425.

Comer, D.E., J.T. Korb, T.P. Murtagh, W.F. Tichy, (1985), "The TILDE Project," Dept. of Computer Science Technical Report CSD-TR-500, also appeared in the Proceedings of the Workshop on Operating Systems in Computer Networks, ACM and IBM, Zurich, Switzerland.

Dyksen, W.R., R.E. Lynch, J.R. Rice and E.N. Houstis, (1984), "The performance of the collocation and Galerkin methods with Hermite bi-cubics," *SIAM J. Numer. Anal.*, **21**, pp. 695–715.

Gannon, D. and J. Von Rosendale, (1984), "On the communication complexity of parallel numerical algorithms", *IEEE Trans. Computer*, Vol. C-33, No. 12, 1180–1194.

Gylus, V., and D. Edwards, (1976), "Optimal partitioning of workload for distributed systems," *Proc. Compcon*, pp. 353–357.

Houstis, E.N., E.A. Vavalis and J.R. Rice, "Convergence of an $O(h^4)$ cubic spline collocation method for elliptic partial differential equations", submitted.

Houstis, C.E., E.N. Houstis and J.R. Rice, "Performance analysis of future multiprocessing systems", to be submitted.

Houstis, E.N., C.C. Christara and J.R. Rice, "Quadratic spline collocation methods for two point boundary value problems", submitted.

Houstis, C.E., E.N. Houstis and J.R. Rice, (1987), "Partitioning PDE computations: Methods and performance evaluations", *Journal of Parallel Computing*.

Houstis, E.N., M.F. Mitchell and J.R. Rice, (1985), "Collocation software for second order elliptic PDE's," *ACM Trans. Math. Software*, **11**, 379–412.

Houstis, C.E., E.N. Houstis and J.R. Rice, (1984), "Partitioning and allocation of PDE computations in distributed systems," in PDE *Software: Modules Interfaces and Systems* (B. Engquist, ed.) North-Holland, 67–85.

Houstis, E.N., M.A. Vavalis and J.R. Rice, (1984), "Spline-collocation methods for elliptic PDE's." In *Advances in Computer Methods for Partial Differential Equations V* (R. Steplman, Ed.), IMACS, pp. 191–194.

Houstis, E.N. and J.R. Rice, (1982), "High order methods for elliptic partial differential equations with singularities," *Inter. J. Numer. Meth. Eng.*, **18**, 737–754.

Irodoatou-Ellina, M. and E.N. Houstis, "An $O(h^6)$ quintic spline collocation method for fourth order two point boundary value problems", submitted.

Kleinrock, Leonard, (1985), "Distributed systems," *Communications ACM*, 28, 1200–1213.

Kosaraju, S.R., (1986), "Efficient algorithms for selection on a tree of processors",

preliminary report.

Kosaraju, S.R., (1986a), "Pipelining characteristics of a tree of processors", preliminary report.

Mehrotra, P. and J.R. Van Rosendale, "*The BLAZE Language: A Parallel Language for Scientific Programming,*" ICASE Report No. 85–29, ICASE, Hampton, VA. 23665 (to appear in Journal of Parallel Computing).

Norton, Alan and G.F. Pfister, (1985), "A methodology for predicting multiprocessor performance," *Proc. Internat. Conf. Parallel Processing*, 772–778.

Rice, J.R., (1986), "Multi-Flex machines: Preliminary report", CSD-TR-612, Computer Science, Purdue University.

Rice, J.R., (1985), "Problems to test parallel and vector languages". CSD-TR-516, Department of Computer Science, Purdue University.

Rice, J.R., (1984), "Software parts for elliptic PDE software". In *PDE Software: Modules, Interfaces and Systems*, (Engquist and Smedsaas. Eds), North-Holland, 123–134.

Rice J.R., (1982), "Machine and compiler effects on the performance of elliptic PDE software". In *Proc. IMACS 10th World Congress*, 1, IMACS, 446–448.

Tichy, W.F. and Z. Ruan, (1984), "Towards a Distributed File System," CSD-TR-480, Computer Science, Purdue University, West Lafayette, IN, also appeared in the *Winter USENIX Conference.*

Takuhashi, Y., (1982), "Partitioning and allocations in parallel computations of partial differential equations", In *Proc. 10th IMACS World Congress*, 1, IMACS, pp. 311–313.

Tang, W.P., (1986), "Schwarz splitting, a model for parallel computations", Ph.D. thesis, Stanford University.

Towsley, D., (1983), "An approximate analysis of a multiprocessor", IBM Research Report RC120 (43663).

Werner, A., (1963), "Anwendungen and fehlerabschatzungen fuer das alternierende verfahren von". H.A. Schwarz. *ZAMM*, **43**, pp. 55–61.

Williams, E.A., (1983), "Assigning processes to processors in distributed systems," *Proc. Internat. Conf. Parallel Processing*, 404–406.