AN EXPERIMENTAL SYSTEM FOR

COMPUTER AIDED GEOMETRIC DESIGN

UUCS-84-008

2 NOVEMBER 1984

This report is based on the proposal submitted to DARPA in May, 1984. The time frame for performance of the funded contract is September 1984 through August 1987. The sections of the proposal which cover budget and biographical data of the senior research personnel are not included.

Richard F. Riesenfeld

Principal Investigator

Kent F. Smith

Co-Principal Investigator

1. Project Overview and Summary

The main goal of this proposed level-of-effort research project is to extend present capabilities in the area of Computer Aided Geometric Design (CAGD) and to develop custom VLSI support for some special geometric functions. Our already proven expertise in the areas of for CAGD, Very Large Scale Integration (VLSI) and programming language development will be directed toward this goal. In order to realize the gains which are discussed in the second chapter, considerable and coordinated effort will be required in all three areas. We have considered the problem and have arrived at the overall milestones which are discussed in this chapter. Subsequent chapters elaborate on the the VLSI role in and the requirements for developing custom hardware algorithms for computer geometry, and the supporting role that continued research and development of the Lisp-based Portable Symbol Manipulation System will play in both the VLSI and CAGD efforts. Each chapter presents the individual goals of the sub-phases of this project which have been carefully correlated to meet the overall goals.

1.1 Project Statement of Work

The following statements are brief synopses of the more detailed ones which are contained in the following chapters.

Geometric Modelling Statement of Work

1. Design mathematical spline representations and corresponding data structures for use internal to the Alpha_1 modeller.

2. Demonstrate the power of the unified spline approach by modelling difficult mechanical pieces in the Alpha_1 testbed. Challenging modelling candidates, like a modern truck or a personnel carrier, will be chosen to be of interest to the DoD community. Demonstration will result after creating and implementing the necessary modelling tools. The result will be a three dimensional mathematical representation.

3. Develop an exact arithmetic algorithm appropriate for use in the intersection operation. Analyze candidate algorithms for suitability with regard to specialized hardware implementation in VLSI.

4. Specify a functional architecture of the testbed Alpha_1 modeller which shows relationships between candidate algorithms and data components to allow initial determination of an architecture for a VLSI implementation. Specialized characteristics of the algorithms which are amenable to VLSI implementation techniques will be identified if they exist.

5. Develop a methodology for extracting a finite element model from the proposed spline based master geometry representation which will allow us to interface to a large body of existing sophisticated finite element analysis packages like Adina. Demonstrate the methodology on a selected testcase.

VLSI Statement of Work

1. Formalize the structured tiling integrated circuit layout methodology and the structured arithmetic tiling design methodology.

2. Develop static and dynamic PPL cell libraries using the CMOS technology. This will involve circuit design, simulation, layout, and testing. We expect to use the MOSIS 3-micron technology.

3. Design, implement and test selected CMOS PPL circuits for the special purpose geometric processors to be integrated into the Alpha_1 testbed. These circuits will be fabricated through MOSIS and will include circuits which implement portions of the intersection and subdivision algorithms used in Alpha_1.

4. Define, design, implement and test several high-speed full-custom GaAs circuits of interest in geometric modelling. This will be done in conjuction with Rockwell, using their design rules, and fabrication capabilities. Circuit simulation will be done using an elementary model developed at Rockwell.

5. Identify, design, implement (in NMOS and CMOS) and test a group of parameterized tiles (other than arithmetic tiles) which will be required in the implementation geometric processors. Since VLSI-based arithmetic computations will be a key portion of the project, a specialized arithmetic integrated circuit design capability will be developed, based on the structured arithmetic tiling design methodology. Test structures will be fabricated using the MOSIS facility.

6. Design, implement and test a special-purpose geometric processor. This may be a fixed-point (or rational) arithmetic processor. This will be done using structured tiling.

7. Develop algorithms and tools for the automatic generation of parameterized tiles and arithmetic structures. The parameterized tiles which will be considered include RAM, ROM, multiplexors, data switches, and arithmetic structures.

8. Develop a unified set of tools which, in conjunction with the existing set of PPL design tools, will be used for the design, verification and implementation of the circuits described above. These tools will be built on a relational database system and will include a hierarchical structured tiling design editor, a hierarchical, mixed-level simulator, an advanced state-machine generator, an arithmetic module generator, and test sequence generators.

Portable Symbol Manipulation System Statement of Work

Study and develop a graphical support software environment capable of efficiently supporting VLSI and CAGD applications. This environment and programming language must have the following characteristics:

1. The environment must be portable across many different workstations and be easily transported to the new workstations soon to appear. Therefore, we plan on utilizing architectural description language techniques to transport the environment to each new workstation;

2. A language interpreter for program development to aid in the rapid prototyping of the VLSI and CAGD applications;

3. Highly optimizing compiler for the incremental development of efficient production quality code.

The following outlines the steps necessary to produce the graphical support system as described above:

1. Integrate the architectural description language techniques into the current environment used for VLSI and CAGD to provide enhanced optimization techniques.

2. Expand the architectural description techniques to include code generation that will allow more flexible and efficient production code. This enhanced environment will be released to the CAGD and VLSI groups for testing.

3. Research into the integration of the architectural description techniques into the entire environment. These new techniques will allow the graphical support software environment to be transported to a new target architecture simply by writing a new machine description.

4. Research and development into techniques for the specification of data types and declarations in an environment which includes both interpretive and compiled program development.

5. Investigate the extension of the data type and declaration mechanism to provide efficient floating point and matrices as needed in VLSI and CAGD.

6. Release the advanced portable graphical support environment for testing by the VLSI and CAGD projects.

1.2 Project Deliverables and Milestones

This section briefly describes the anticipated milestones and deliverables for the proposed effort. It is understood that the milestones are approximate in nature and the actual achievement dates may not correspond with those cited in this document.

Deliverables

At the conclusion of this contract we will have developed a hardware prototype system of the selected geometric processes and demonstrate it. In order to demonstrate this part of a total geometric design system, it will be integrated into the experimental geometric modelling testbed Alpha_1 and applied to a selected model. The results of supporting and related research into computer geometry, geometric modelling, VLSI, and Portable Symbol Manipulation, as well as the integration of these areas, will be reported.

Milestones for Geometric Modelling Effort

12 Months	Analyze modelling algorithms for hardware implementation.
12 Months	Define hardware/software interface for geometry engine.
24 Months	Demonstrate an interactive, spline-based geometric editor.
24 Months	Graphically simulate a process for manufacturing prototypes using B-spline based geometric models.
24 Months	Develop an interface to an existing finite element modelling package from a B-spline based geometric model.
36 Months	Perform performance evaluations of the special-purpose VLSI processors designed for Alpha_1.

Milestones for VLSI

6 Months	Formalize the IC tiling methodology which will permit the development of special purpose processors for Alpha_1.
12 Months	Define a special arithmetic IC tiling methodology.
12 Months	Define a set of parameterized tiles for use in processing datapaths.
18 Months	PPL tiling implementations in both static and dynamic CMOS.
24 Months	Have a tested arithmetic tiling capability in NMOS and have CMOS designs ready for testing.
24 Months	Complete designs and testing of parameterized tile techniques Have the major tile generators completed.
30 Months	Complete algorithm design for parameterized tiles.
30 Months	Have a tested arithmetic tiling capability in CMOS.

30 Months — Integrate special-purpose processors (for intersection and/or subdivision) into the Alpha_1 engine.

36 Months — Complete design and testing of GaAs test structures.

36 Months — Complete design and testing of a variable-precision general arithmetic processor.

36 Months — Refine and improve the special-purpose processors for the Alpha_1 engine, if necessary.

36 Months — Complete implementation of a unified CAD system for the design of integrated circuits using structured tiling.

Milestones for Portable Symbol Manipulation Systems

12 Months — Complete research work on a peephole optimizer.

12 Months — Add declarative data types to the system.

18 Months — Architectural description driven efficient code generation.

18 Months — Add compile-time data type checking.

36 Months — Source to source transformations working.

36 Months — Exploit data typing and declarations in an optimizing compiler.

2. Proposed Research

2.1 The Role of Computer Aided Design in Industry

The phrase that best characterizes the need for Computer Aided Design of mechanical parts is "competitive edge"; the competitive edge in product design, cost, performance, and quality. Traditional design uses "engineering drawings" as the model. These are really two dimensional projections. Since there are normally only a few views given there is usually either more than one interpretation that "fits" the drawings, or no interpretation that can fit, as when the design features cannot be correlated properly. However, complete manufacturing information is needed. This means that the drawings are interpreted wherever ambigous; in many instances detailed design is actually accomplished at the manufacturing stage. If desired, "wire frame" models are made (with interpretation) and "skins" or surfaces are put on them in the analogous manner of stretching material across the frame. If it is necessary to specify information for a numerical control process, then someone must interpret the engineering drawings and generate a stream of instructions for a numerical control machine. If it is necessary to specify information to mathematical analysis packages, like a finite element analysis, a finite element mesh has to be generated from the engineering drawings, again involving interpretation to make a three dimensional approximation. This might lead to further interpretation and design specification within this finite element model. The results, however are not available to the numerical control device, or to other analysis or visualization techniques. Different processes require different models, and none talks to the others. Getting the results and information obtained from the processes into the engineering drawings sometimes never occurs, and when it does, it is never certain that the assembly line produced version will incorporate those changes, since again, there must be interpretation of the drawings on how to make the molds, machine the product, cut the metal, bend the stock, drill the holes, and the like.

A single master model of the object geometry from which can be derived, in an algorithmic and well defined manner, an N/C specification, a finite element model, a rendering model, and other output type operations would insure that all the analysis and rendering processes are seeing the same geometry. Powerful interfaces to the model and tools for building models can lead to more accurately specified models, models whose material specifications can be exactly known in advance and whose component placement can be uniquely and exactly specified. Perhaps interfaces between the model, the results of the mathematical analysis packages, and the design interface could allow the designer to modify designs quickly to incorporate stress, pressure, thermal, or aerodynamic changes as dictated by the analyses. These are all really methods of "interrogating" the model.

American industry was slow to see the need for these new methods. Japanese industry has made a national commitment to use computer aided geometric design and to integrate it with computer aided manufacturing. While current commercially available systems have few capabilities when compared to advanced research capabilities, and are primitive when compared to what is needed, there are noticeable effects from their use. In the shipbuilding and automobile industries, the Japanese have had great success, and

the Scandinavians have long seen the need for this new technology in shipbuilding. Recently, American industry has realized that they will lose their competitive edge, perhaps forever, if these ideas are not incorporated into their design and manufacturing processes.

The need in the aircraft industry is clear, and the use there is very advanced. An illustration of the potential benefits possible from using a unified computer geometry model is in the area of modelling the surface geometry of an aircraft to assess possible configurations during the crucial period of conceptual design. The capability to predict performance characteristics from a very early configuration concept would provide designers with an extraordinarily valuable early indicator of the utility and merit of a preliminary design concept. Accurate visualizations of the parts portraying cut outs or transparent surfaces could be specified to allow the designer to gain different perspectives of the object, and multiple objects could be simultaneously displayed and manipulated to check possible fits. The experimental computer aided geometric design system, Alpha_1, that we use as our testbed, has these visualization features incorporated. Preliminary interaction with designers has indicated that these features would be useful in the wider context. The model needed for aerodynamic analysis should be developed from, and consistent with the geometry model to insure that the results of the mathematical analysis will accurately predict the performance. Finally, the geometric description could be used for building interior components like struts and spars, for manufacturing stamping dies and drilling and riveting jigs, for fixturing layup of composite fiber parts like wing panels and control surfaces, and for other manufacturing tasks.

The need for computer aided geometric design and modelling is present in more traditional industries. The design of a tank requires analysis at every stage as to its weight, armor placement for minimum vulnerability with weight and other restriction on mobility, gun fire power, etc. With a unified computer geometry system, the designers could develop a preliminary model, automatically interface to analysis packages, and determine vulnerability of the model with the current armor placement. Armor could be shifted and the process reapplied. If the model is not developed in a form consistent with the analysis methods, then a great deal of time must be spent interpreting the model and deriving the computer model for analysis. Then after the analysis results are obtained a new model must be developed, and then another new computer model. If the original model automatically interfaces to, or directly supports the analysis, all the time spent building the "second" model, which could be months, is saved. Faster interaction would allow more design iterations in the same time span and perhaps less vulnerability in the final product.

2.2 Contract Problem Statement and Objectives

Although extensive effort and success has been associated with computer aided design of digital circuits, a parallel thrust has not been the case in three dimensional geometric design where it is necessary for the design and manufacturing of mechanical, not electrical, devices. It is now becoming widely recognized that our capability as a nation to achieve and execute good mechanical design is highly dependent on our computer expertise. However many necessary fundamental advances have not yet occurred in the laboratory, much less in practice.

Substantial work in computer aided geometric design (CAGD) has already been done in the Department of Computer Science at the University of Utah. Alpha_1, a novel experimental and spline-based solid modeller shows the advantages of our approach as it acts as a testbed for new research. We now propose to press forward with necessary research to develop advanced geometric modelling capabilities that are suitable for taking on genuine industrial and DoD grade problems. Coordinated research into the system organization is necessary, including both software and hardware aspects. It requires fundamental research in geometric representations and algorithms, interactive environments for computing and design, interfaces to engineering analysis procedures, special purpose arithmetic units, symbolic algebra, high-speed parallel and pipelined processing, and computer graphics.

We believe that the theory allows for a potential increase of at least five orders of magnitude in computational capability over the current state through exploiting the inherent parallelism of the problem, the use of more suitable arithmetic techniques, and taking advantage of the faster logic families that are becoming available. Looking beyond a specialized hardware system for improving the performance of the current rendering and manipulation system and beyond the presently proposed contract period, further increases in the computation power of the system could be applied to performing analysis and functional simulation computations at interactive speeds. Certainly such a system would not produce comparable performance increases for unrelated applications, since this will not be a general purpose system. However, it will demonstrate how customized architectures can be used in an integral way to solve important, computationally intensive problems.

2.2.1 Present Objectives

Within the contracting period we propose to pursue the research directions indicated above and implement resulting algorithms or systems concepts on the testbed to prove their viability as advanced concepts for design and manufacturing for a select class of objects. It could become an example of how computers can be used to broadly support the overall process of three dimensional mechanical design and manufacturing, particularly serving to combine FUNCTION with FORM in the earliest stages of conceptual design. This will enable a designer to gain some understanding of the performance of a design through the simulation of some of its critical functions at a very preliminary stage of a design.

To achieve satisfactory performance, this will require the concurrent development of specialized computational engines to support the key tasks in converting a mathematical model of an object into a set of polygonal representation optimal for the particular view which can then be passed to a conventional rendering system. A major task includes the subdivision of a surface.

2.3 Method of Approach

We will utilize the close cooperation which exists between the software and the hardware people in the department in a joint approach to this difficult and important problem area. This project will become one of the broadest spectrum approaches to the problems in computer geometry. In order to bring computer geometry in the next stage of advancement we are proposing the following advances in the area of computer aided geometric design:

* A mathematically rigorous and functionally useful description of geometric parts and assemblies must be developed for various types of analysis and subsequent manufacturing, with natural methods of specification of this geometry for the traditional designer. We will explore more general representations which lend themselves to the fundamental notion of using a single (universal) mathematical form with an associated recursive evaluation scheme. Subdivision methods exploit recursive algorithms.

* The distribution of the system across a number of sub-systems, each optimized for their particular task. The nature of the portions of the current system concerned with the transformation of the mathematical description of the object to the polygonal form, or other simple element, acceptable for rendering points to components handling intersection, surface flatness testing, subdivision, and scanline construction, operating as a pipelined system.

* The use of massively parallel computation in many of these sub-systems, replacing iteration and queuing structures. The basic nature of many of the underlying algorithms in Alpha_1 is divide-and-conquer. A surface is tested for flatness, and if it does not meet the specified criteria, it is divided into two surfaces, each of which is then tested. Such a binary-tree process rapidly reaches a point where a large number of processing elements can be well utilized.

* The use of processing elements more suitable to specific tasks than general purpose computers. In addition to customization of datapaths and functional units, new arithmetic structures will be employed to better match the requirements of the particular geometric algorithms. Standard floating point arithmetic is not particularly suitable for many geometric algorithms, yet it is all that is available on conventional computers for handling non-integer data. In some cases, the basic algorithms can be transformed from floating point arithmetic to fixed point algorithms, utilizing arithmetic units optimized for the algorithms. In other cases, neither fixed nor floating point computation is suitable. The implementations of key algorithms have been greatly complicated by the need to compensate for inexact floating point representations, resulting in a computational increase and more special cases. The use of precise rational arithmetic (where each number is stored as two integers treated as the numerator and denominator of a fraction) solves many of these problems, but requires the development of suitable algorithms and design of hardware for the efficient hardware implementation of operations

such as addition, subtraction, and comparison (which requires the finding of a common denominator).

* Finally the use of advanced VLSI technology (such as small geometry CMOS or GaAs) and asynchronous structures can produce additional speed increase, although not so great as specialized design techniques. In particular, the use of speed-independent asynchronous structures not only eliminates the difficulties of clocking very large circuits, but may result in smaller structures than for clocked implementations. For example, to achieve high performance in a clocked system, a counter requires an extensive carry generation network (whose complexity increases with the square of the number of bits in the counter). However, an asynchronous carry-completion adder can use a simple ripple carry, along with some additional circuitry to determine when the counter has reached its final value (increasing linearly with the number of bits), and still achieve good performance. This is because fully half the count operations require no carry (the least significant bit changes from 0 to 1), a quarter require only the carry across a single bit, an eighth across two bits, and so forth. Rarely will a carry propagate across a significant number of bits.

* Development of interfaces between the geometry model and massive, accepted and proven codes for Finite Element Analysis, numerically controlled manufacturing processes, and the like. That is, take the current Alpha_1 representation for geometry and tie it in with the world to test the hypothesis that these very general high level models are adequate to support the various functions that a geometric modeller should provide. In the course of this we simultaneously learn about any defects in the present approach and, when corrected, generate a system of broader utility.

* New methodologies for geometric modelling, and appropriate paradigms for their use. Just as a categorically new computer language like Lisp or APL has its typical paradigms or modes of thinking that it supports, we will have to discover and establish the corresponding paradigms for solid modelling with freeform surface based objects. These insights come out of considerable modelling experience, so we will look for objects of particular DoD interest in selecting these tasks. While the goal is always to enhance the designer's current environment, it is inevitable that this technology will redefine the designer's role just as text editors are redefining the role of a secretary.

It is estimated that the necessary speed-up of five orders of magnitude over a VAX-750 running Unix can be potentially achieved by the use of the techniques discussed above. The use of higher-speed logic (such as high-speed CMOS and/or GaAs) will account for at most one order of magnitude. An additional one and one-half to two orders of magnitude speed improvement will be gained by exploiting parallelism in key algorithms, and highly efficient arithmetic structures which are specifically tailored to these algorithms will account for another order of magnitude. The final order of magnitude speed improvement will be gained by separating the various functional components of the system (subdivision, intersection, rendering, etc.) into their own processors and pipelining

their tasks in converting the model into a final picture, thereby eliminating much of the general-purpose processor overhead involved in computations such as loop control and data alignment. Thus, these four elements should improve performance by about 10^5.

The major performance improvements will come through exploiting the inherent parallelism in many of the individual tasks, and the use of arithmetic and data handling structures more suitable to the task. For example, the replacement of the current floating-point code in the intersector (which requires elaborate routines to counter the inexact nature of floating-point when testing if two points are identical, and still can fail for some pathological cases) with rational arithmetic could produce at least an order of magnitude speed improvement. Both the subdivision and intersection tasks have a high degree of potential parallelism, with little communications necessary between the parallel tasks. For subdivision with a depth of 14, there can be parallel execution on the order of 10^6 to 10^7. The use of between 100 and 1000 simple processors, a more manageable number to implement, could give a performance improvement of two or more orders of magnitude.

The existence of specialized, high-speed hardware would also allow the use of algorithms currently rejected because of their poor performance on our existing general purpose hardware. For subdivision, a matrix multiply can be replaced by a more suitable recursive calculation. A more precise intersector can be developed.

2.4 Summary

A close collaboration of research efforts in CAGD, VLSI, and Lisp Interactive Environments is being proposed to further the area of three dimensional mechanical design. This synergism will advance the individual projects through pursuit of the overall goal of advancing the collective target area. It also has the potential for advancing the present notion of how CAD in the three dimensional mechanical area might be performed. Developing a solid computer science basis for undertaking such an area of endeavor could open up this direction for dramatic changes in the longer term future.

This project has the promise of initiating a wider recognition of the need for coming to grips with the fundamental issues of computer geometry. It is hoped that the results, both in terms of depth and scope of implications, will trigger a realization nationally that computer geometry is a vitally important area of computer science. It bears heavily on us in strategic terms as well as in economic and intellectual terms.

3. Computer Geometry and Computer Aided Design and Modelling

3.1 State-of-the-Art External to Utah

At the present, three dimensional mechanical CAD, or CAD/CAM or CAE (Computer Aided Engineering), as it is sometimes labeled, is a field of recognized importance in industry. Universities are in the midst of a far reaching swing from viewing CAD as fringe research to accepting it as bona fide computing and engineering activity. But universities cannot find knowledgeable faculty in adequate supply to meet the teaching or research demand, nor can industries and government find trained scientists in sufficient number. Consequently the field is caught in a period of rapid transition with a very limited talent pool for carrying out the urgent needs for the future.

With the aforementioned exigencies prevailing, most effort is being committed to short term development of pieces that can be used to meet the present demands for tools. Much of this is computerized two dimensional drafting tools, and their extension to two-and-a-half dimensions with "wire frame" models. Very little long range, fundamental research is underway, especially research addressing the systems integration aspects. At present there is a great lack of integration of the essential components of a CAD system largely because separate representational models are used by many of the various parts.

Most existing CAD systems are massive, rigid, unmanageable collections of programs typically written in Fortran over many years, not written and structured in a modern way and implemented flexibly to support interactive design interpretively.

At present geometric solid modellers do not allow the integrated use of freeform surfaces, so the capability for modelling curvilinear geometry is separate from systems offering true solid modelling. This is partly because the computational algorithms and tools required for freeform object modelling are still in their infancy, and many times no reasonable implementation is known. A major complicating factor is having to implement geometric algorithms which are already complex on computers with floating point arithmetic where replications of the same answers cannot be guaranteed. The same problems are present in primitive based solid modelling systems, but since the underlying primitives are less complicated than a freeform surface, the problems have less serious consequences.

3.2 Introduction to Geometric Design and Modelling Concepts

During the 1960's Ivan Sutherland and his mentor Steve Coons introduced the notion of computer modelling geometric objects in Sketchpad and the project MAC "little red book", respectively. The ideas and results detailed therein form the seminal ideas for many ensuing research activities in computer graphics and computer aided geometric design (CAGD). The fields have developed enormously since those first days, but many of the problems have only grown in complexity. What at first seemed like dazzling solutions were really windows onto greater needs and problems to be solved. The field of Computer Aided Geometric Design has historically been preoccupied with two particular problem areas:

* Representation schemes, and

* Systems and techniques which generate models.

We shall briefly describe various representation schemes that have been developed for modelling surfaces, including their advantages and limitations. For later comparson we shall discuss representation schemes for simple "primitive" modelling methods. The issues involved in embodying a model in these representations is examined. Further, we explore whether this unified representational approach might allow a unified approach to unsolved problems or to problems at present solved with many different ad hoc techniques. We propose directions for theoretical, algorithmic systems and hardware research in modelling and computer geometry issues. This entails the integration of advances in theory, software and systems engineering, and hardware to mold a unified, coherent attack on a vast multifarious problem.

3.3 Mathematical Representations

3.3.1 Introduction

If one is to use richer representations for objects beyond copious collections of surface points, one must look to representations with mathematical structure. Even the widespread polyhedral models are mathematical representations, and homogeneous coordinates with all their properties are applications of projective geometry, as was pointed out by Riesenfeld [25]. For the most part simple objects with names like cubes spheres can be represented exactly; their formulations are known. Freeform surfaces, on the other hand are in the mind of the designer, and have no concise expression, the CAGD system is supposed to furnish one. While mathematical representations can play a role of fundamental importance in striving for a unifying view of the subject, there are basic and major differences between the problems mathematics seeks to solve and its motivations, and those of researchers in computer aided geometric design using sculptured surfaces.

3.3.2 The General Approximation Problem

The Curve Approximation Problem is stated as: Given a set of N ordered points in the plane, find a function from a predetermined class that passes as near to those points as possible. Several possibilities arise. First, the problem may be unsolvable using the given predetermined set of functions with that data. Second, the number of degrees of freedom available from the set of data is too large; that is, more than one function will pass through (interpolate) all the points. Third, exactly one function will interpolate the specified points. Fourth, there will be a nonunique solution to the "as close as possible" criterion. Fifth, there will be a unique solution to the closeness criterion. The presumption may be that the data is good and "hard"--that is, the curve must pass through the data; or that the data is noisy, to a greater or lesser extent. Finally, the class of functions chosen to approximate will determine many other desirable or undesirable characteristics of the solution.

In mathematics, it is always assumed that there is an original function that the user is trying to approximate. In *ab initio* design, however, what is original is usually a rough mental sketch or idea of the shape of the curve. Rarely is an explicit mathematical form known. Moreover, most mathematical approximation theory is developed for *explicit* functions like $z = F(x,y)$. The design world mainly operates in a *parametric* context since the desired geometry and shape is coordinate system independent and may not be representable as a graph of a single valued function. A parametric function has a vector-valued form like $F(u,v) = \langle x(u,v),y(u,v),z(u,v)\rangle$.

The methods indicated below are all methods which have been used for design of sculptured surfaces. We shall look at the underlying approximation methods originally developed for explicit functions using the mathematics model of an original or primitive function that one is trying to recover.

We shall discuss curve and surface schemes based on three classes of functions.

Class1: Polynomials on the interval [a,b] of degree less than or equal to N, denoted $\mathbf{P}^N$. They may be represented in many bases, the most common of which is the power basis $a_0 + a_1x^1 + a_2x^2 + \ldots + a_Nx^N$. It is clear that these functions have N+1 degrees of freedom. If any polynomial is zero on a subinterval of [a,b], then it must be zero on the whole interval. This means that it is impossible to have local bases using only polynomials. In any basis, modifying one coefficient effects the shape of the whole curve.

Class 2: Piecewise polynomials of degree less than or equal to N-1. These functions s(x) are called *splines*. For these functions the *breakpoints* must be specified, that is, the values of x that join the articulated polynomial pieces. A breakpoint is where the function has one polynomial piece joining against another polynomial piece on the other side. At each breakpoint the continuity class C[k] must be specified. The smoothest splines can have k=N-1 continuous derivatives (implying the same polynomial on each side); the least smooth can have k=-1 continuous derivatives (a discontinuous function with finite jumps in value). The set of breakpoints together with the associated continuity classes at the breakpoints is usually specified in a nondecreasing sequence including multiplicities called the knot vector t. One can represent one of these functions by finding the coefficients of each of the polynomial pieces and storing them. These coefficients, however are not all independent since the derivative continuity conditions at breakpoints must be met. Another alternative is to use global basis definitions which incorporate the required derivative continuities in the definition of the basis functions themselves.

One such basis that has the desired continuity built into its definition and is also minimal is the B-spline basis. This basis is minimal in the sense that it is nonzero on the smallest interval possible while still meeting required continuity conditions in a nondegenerate way. Thus B-splines are local; they vanish outside a local support domain. Modifying parameters for one particular basis function affects the shape of the curve only in a small region where that particular basis function is nonzero. Other desirable attributes will be discussed later.

The name spline comes from the mechanical draftsman's spline which behaves like a thin

beam. Over all functions that are $C^{[k]}$ and pass through a prescribed set of data points, polynomial splines minimize what is often called "linearized" curvature, and have an associate minimum energy property. The term *uniform* is used when the breakpoints, or knots, are all evenly spaced. For convenience the integer are often selected as a uniform knot set. The term *nonuniform spline* is used otherwise when the breakpoint spacing is not regular.

Class 3. Piecewise exponentials are a closer solution to the problem stated above. They also have breakpoints and continuity conditions which must be specified. They allow the introduction of a "tension" parameter and are the basis for the notion of splines under tension.

3.3.2.1 Interpolation

The Planar Interpolation Problem is stated simply: Given $\{(x_i,y_i)\}_{i=0}^{n}$ find a function $y = f(x)$ such that $y_i = f(x_i)$. While simply stated, this problem has given rise to many different solutions. If the class of functions used is polynomials of degree <= n, then the choice of basis can dramatically effect the computational stability of the evaluation scheme.

Polynomial Interpolation

One set of polynomial bases called the Cardinal or Lagrange basis is of the form

$$L_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}, \quad i = 1,\ldots,n$$

This basis is constructed to have the useful property that $L_i(x_p) = \delta_{i,p}$. As can be directly verified, the interpolant has the simple solution

$$\sum_{i=1}^{n} y_i L_i(x)$$

While this is a powerful representation for proving properties, this form is not computationally tractable unless there is a family of many different problems to solve all with the same abscissa coordinates x_i in the data.

A more general solution is called Hermite Interpolation, which is appropriate when interpolation to derivative data, in addition to position information, is required. In this problem we seek a polynomial which interpolates $\{(x_i,y_i,y'_i)\}_{i=1}^{n}$. A polynomial of degree 2n-1 is needed to meet the 2n conditions arising from values for position and slope at each data location.

Generalized Hermite Interpolation allows specification of derivatives up to any order Conditions for this then are

x_i, $y_i^{(j)}$, $j = 0,1,\ldots,m_i$, $i = 1,2,\ldots,n$.

The number of linear conditions then is always $n + \sum_{i=1}^{n} m_i$.

Polynomial Splines

When the number of points to interpolate is nontrivial, the degree used for polynomial interpolation can become very high resulting in many undulations and possibly unacceptable error in computation. To compensate for this problem, many piecewise solutions have been developed, piecewise Hermite being the most popular, since it guarantees at least derivative continuity between pieces. Unfortunately, most often the derivatives are not given as part of the data and the user must figure out some way of "guestimating" something reasonable. Another approach is to impose only derivative continuity of a certain class at each breakpoint (interpolation points, usually), but not specify the values that the slopes must take on. This approach leads to spline interpolation. Advantages of spline interpolation are that the polynomial pieces are of a low specified degree and hence there are fewer undulations with less magnitude than in polynomial interpolation. The drawback is that the spline is not an analytic function, only a piecewise analytic one. Since the degree of continuity at the breaks can be specified, this is more than sufficient for most applications. Further the "complete" spline interpolant of degree 2m-1 minimizes Integral $(s^{(m)})^2$ over all functions interpolating the point data.

The spline basis exhibiting the Kronecker delta evaluation property corresponding to the Lagrange basis is called the Cardinal Spline basis. This basis, however, can not be a local basis and still meet the continuity conditions. The basis closest to being cardinal and local is the B-spline basis. Each basis function of degree n-1 (order n) is nonzero over at most n-1 spans, and on each span there are at most n nonzero basis functions. This leads to a diagonally dominant banded linear system of equations which have highly stable numerical solutions.

Generalized Splines under Tension

Sometimes is is desirable to interpolate the data and to also minimize a variational property different from a quantity related to curvature. That is, over all functions s that interpolate the data, find s such that

Integral $_a^b (s'')^2 dt + \alpha^2$ Integral $_a^b (s')^2 dt$ = minimum.

Clearly, the value of α affects the interpolant selected, because it govern the relative weighting of importance between the first derivative and the second derivative. The solutions to this problem are piecewise exponentials.

3.3.2.2 Approximation

In contrast to the interpolation problem, the approximation problem usually does not require the approximant to go through all the data points. Sometimes there are other criteria to be considered such as shape. A standard approximation scheme is the least

squares method. This is applied when there is more data than degrees of freedom in the approximating set, and when noise might be present in the data. Least squares, however is not at all a shape preserving scheme.

Polynomial Approximation

Suppose that f is a function from [0,1] to the real numbers, and let N be any positive number. The curve

$$\gamma_p(x) = \sum_{i=0}^{n} f(\frac{i}{n})\, \theta_{i,n}(x)$$

where

$$\theta_{i,n}(x) = \binom{n}{i} x^i (1-x)^{n-i}$$

is called the Bernstein approximation to the function f on [0,1]. The functions $\theta_{i,n}$ form a basis for all polynomials of degree n on [0,1]. This approximation converges to the function f as n goes to infinity, but this method was not popular in mathematics since the "rate of convergence" is very slow. This approximant, however, has important properties for design since γ_p converges to the function f and to all its continuous derivatives, as n goes to infinity. This unusually strong approximation property has been termed simultaneous convergence. Also this approximation is variation diminishing. That is, γ_p exhibits no more zeros than f; γ_p has no more undulations or wiggles in it than f. This means that unlike interpolation, the Bernstein approximant never introduces extraneous undulations into the process.

Finally, $\sum_{i=0}^{n} \theta_{i,n}(x) = 1$ for all x.

Hence γ_p is a convex combination of original function values, so its maximal extent and a convex superset containing it are known. A recursive procedure for evaluating the basis functions can be developed from the combinatorial nature of the basis functions evidenced by the binomial coefficients in the definition.

Polynomial Splines

Despite its attractive features, Bernstein approximation still suffers from the problem that it is a single global polynomial, and hence the degree goes up with the closeness of approximation. Another approach was investigated by Schoenberg in his use of the variation diminishing splines. Given a knot collection **x**, let $\{B_{i,n}\}$ define the B-spline basis of order n (degree n-1) over that knot collection. Define another sequence $\mathbf{x}^*$ by

$$x^*_i = \frac{x_{i+1} + x_{i+2} + \ldots + x_{i+n-1}}{n-1}$$

Then

$$\gamma_s(x) = \sum f(x^*_i) B_{i,n}(x)$$

is called the *variation diminishing* approximant to f. Since the B-spline basis is used with the function values, γ_s is a local, shape preserving, approximant. Since B-splines also sum to 1 at any value of x, the convex hull property still holds. As the variation diminishing name implies, this approximation has no more undulations than the original so-called primitive function it approximates. Here, as the number of knots increases, and the max spacing diminishes, the variation diminishing approximant converges to the function. Note that the degree does not increase. In 1972 deBoor and Cox independently introduced a stable recursive algorithm for computing the B-spline basis functions making them computationally tractable [5].

3.3.3 Explicit versus Parametric Representations

While these methods were all developed for explicit functions, design occurs in a parametric world, for the most part. The geometry of mechanical parts is not influenced by the particular coordinate system used for its spatial representation. The mathematical methods developed for computer modelling must adhere to the same principles. Parametric modelling accommodates this.

3.3.4 Inverting Approximation Schemes into Design Schemes

The most straightforward way to make modelling schemes out of the above interpolating and approximating schemes is to insert vector (parametric) functions in place of the scalar functions for which they were originally developed. Then, for *ab initio* design, a further twist of affairs is necessary.

Suppose M+1 ordered points in 3 space are given, $\{P_i\}_{i=0}^{M}$. The most straightforward approach to defining a curve with the data would be to interpolate the points using a parametrized version of the cardinal basis. One could then use these points to control the curve by moving them and moving the curve also. Unfortunately, this method is fraught with fundamental problems. First, since high degree interpolation frequently has undulations, so will these curves-- interpolation is not variation diminishing. Secondly, the parametrization selected for the spatial points strongly affects the final curve. The curve, in fact, can loop, retrace itself, and do other undesirable things to insure that the interpolant meets each data point at the right parametric value. Indeed, these are dangers with any interpolation method, although the problem is much less severe for spline interpolation. Straightforward interpolation gives rise to as many problems as it solves.

To convert the Bernstein approximation to the Bezier design scheme Gordon and Riesenfeld [18] showed that one must make the identification $f(i/M) = P_i$, and that

$$\gamma_D(t) = \sum_{i=0}^{M} P_i \vartheta_{i,M}(t)$$

is the Bezier curve of degree M.

Here the polygon connecting the points $\{P_i\}$ is considered to be the function that is approximated.

If one uses B-spline basis functions to generate the curve, then Gordon and Riesenfeld [19] identified $f(t^*_i)$ and P_i to obtain the B-spline curve

$$\gamma_s(x) = \Sigma\ P_i B_{i,n}(t).$$

It should be clear from this that the curves resulting from the Bezier scheme are a subset of the curves resulting from the B-spline scheme. The B-spline curve is a variation diminishing vector approximant to the polygon (it introduces no new wiggles). Its convex approximant and local properties show a faithfulness to the polygon. One can use the vertices of the polygon to control the shape of the curve, hence the name, "control polygon". Moving a vertex can provide interactive control over the curve, and since the curve "approximates" the polygon, it also provides geometric intuition as to its shape. The shape of the curve is also affected by the particular choice of parametrization. One can modify shape by retaining a constant polygon and modifying the knot vector in the parametric space, as well.

If one chooses to interpolate fixed points, then the coefficients to the basis functions will be determined as the solution of a linear system of equations. The polygon that this yields may not look pretty, but the curve will interpolate. It is important to remember that an interpolating polynomial (or spline) is unique; its shape does not depend on the particular choice of basis used to represent or compute it.

3.3.5 Making Objects

3.3.5.1 Boundary Representations

A volumetric object can be specified in one of two ways. Either as a volume by name, or by its boundary, a point being inside the solid if it is inside the boundary. Any object that is defined by freeform surfaces has a boundary model, and one must devise a method of representing surfaces which will eventually be that boundary.

Surface Representation Schemes

Research into schemes for representing shape in the computer received a central emphasis during the late 1960's to middle 1970's. Coons' work [12] and Bezier [4] were pioneering fathers in the field, each introducing techniques that bear their names.

Although the aim has been to develop representations which were applicable to computer implementation, the character of the research is highly mathematical. An extensive variety of schemes for the representation of shape in general, and sculptured surfaces in particular have been explored.

A straightforward surface generalization developed from curve techniques is the form

$$S(u,v) = \Sigma\Sigma\ P_{i,j} Q_i(u) R_j(v)$$

where $\{Q_i\}$ and $\{R_j\}$ are two collections of functions used for approximation. Each or both might be a cardinal basis polynomial or spline, or Hermite basis functions, or Bezier basis functions of different or the same degree, or B-spline basis function of different or the same order. The resulting surface is called a tensor product surface.

When the Q_i and the R_j are both cubic Hermite basis functions, then this is called the Tensor Product Coons patch. The surface is one bicubic patch with sixteen parameters. Those specify position, two first derivatives in each tangent direction, and the second cross or "twist" derivative, at each of the four corner points of the patch. The resulting surface interpolates to all of those conditions. Unfortunately, this first early method has the drawback that information is difficult to determine in an explicit fashion, and very few individuals understand exactly how the twist vectors influence the surface. The interface to this method is numbers, since the coefficients of the basis functions are not directly related to those desired numbers.

In this early period designing a sculptured surface generally entailed describing the shape of a single surface patch. A few efforts were made to provide tools for joining patches [1], so that larger surfaces could be produced, but few attempts were made to model real objects as examples.

If the Q_i are Bernstein Bezier polynomial basis functions of degree n, and the R_j are Bernstein Bezier polynomial basis functions of degree m, the resulting surface is an n-th by m-th degree polynomial patch. The coefficients of the basis functions form the "Bezier mesh", and the attractive properties of convex hull, variation diminishing and subdivision capabilities extend from the curve case.

If the Q_i and R_j are B-spline basis functions, possibly of different order with different knot vectors, then the resulting surface is a B-spline tensor product surface. It has many polynomial patches incorporated within it and the properties associated with the B-spline curve (all the Bezier properties, plus arbitrary refinement) are easily extended to surfaces. The $P_{i,j}$ form the associated B-spline mesh.

Topological Primitives

A second method of defining objects by boundary representations is the Euler operator scheme. The objects have a topological model realized in the geometry. The model is represented as a graph of faces, edges, and vertices. The faces need not be planar in theory, although most implementations only have planar polygonal faces for database and algorithmic considerations. The Euler equation in graph theory relates the number of faces, edges, and vertices of a solid (polyhedral) object to the genus of the topological graph representing that object. The Euler operators are rules for modifying the model while preserving a valid theoretic representation, that is, satisfying the appropriate Euler equation.

3.3.6 Solid Volumetric Models

In the 1970s the notion of "solid modelling" was introduced through the scheme of primitive volumetric modelling. Those who promoted this concept of solid modelling emphasized that it is important to use objects which are well specified with a clearly defined interior, boundary, and exterior. A valid model, in the solid model sense, trichotomizes space so that for each point in space one can answer the classification question, "Is this point inside, on the boundary, or outside of the model?" The objects are thought of as solid volumes of material, hence the name. Infinitely thin shells are not allowed. All objects must have some thickness, and boolean combinations of these primitive volumes, like spheres, cylinders, rectangular prisms, and ellipsoids must preserve the realizability of the model.

Some confusion surrounds the term "solid modelling", because of the coincidence that the systems that introduced this concept of "regular operations" also introduced the concept of modelling with combinations of primitive objects. The objects to be used for modelling blocks, or primitives, were selected because their mathematical representations are simple and they are valid in the preceeding sense. Simple set operations involving these primitives were analytically possible, so they were used in systems that emphasized solid modelling. Unfortunately this has resulted in confusion between the notion of being able to "trichotomize space" and the method of adding and subtracting simple named shapes.

3.4 Modelling Paradigms

3.4.0.1 Valid Object Systems

Beginning in the mid-70's, interest and emphasis turned to techniques and systems which could automatically guarantee that an object created was in some sense a "valid" object [20]. Although the definitions of validity vary slightly, and are sometimes mathematically cumbersome, a good rule of thumb is that a "valid" object is one that can be realized physically. That is, a "valid" object which is modelled in the computer could, in principle, be manufactured.

This direction, which is still the main point of interest for many researchers in the field, has led to two important types of systems: those based on combinations of solid primitives and those based on the "Euler Operators".

Solid Primitives

The solid primitives systems, e.g. PADL [29] and GMSolid [6], (sometimes called Constructive Solid Geometry, or CSG systems) operate on the basic principle that if one performs one of a small set of operations (set union, intersection, and difference) on objects which are known to be valid, then the result of the operation is a valid object. In order to achieve this, Voelcker and Requicha [21] found it necessary to modify slightly the usual notions of set operations, and introduced the elegant "regularized" set operators which are universally used in these types of systems, and which have found important

applications in other areas as well. Many commercial systems based on these operators (including Voelcker and Requicha's PADL) have appeared, and they are well-suited for a large number of mechanical design problems. A serious drawback, however, is that these systems have no capability at all for handling sculptured surfaces. Neither the theory nor the algorithms extend in any straightforward way.

Euler Operator Systems

The other basic type of system that has emerged from the emphasis on producing valid geometric models is based on the Euler operators. All realizable, or valid, objects will satisfy the general Euler equation, and the Euler operators permit only operations which insure that the underlying graph satisfies the Euler equation. This kind of validity has a graph-theoretic context and assures a different property than the above. In these systems [15], there is an interesting distinction between the "topology" and the "geometry" of the object. Moving one vertex of a cube (for example) does not change the numbers used in the basic Euler equation involving numbers of faces, edges and vertices. In this way topologically valid, but geometrically unrealizable, objects may be defined.

The Euler operator systems can incorporate sculptured surfaces to some extent [2], although often not at the same level as the polyhedra which are the basic building blocks for the system. Much of the control over validity, one of the guiding principles for the system, may be lost when sculptured surfaces are incorporated.

3.4.1 Available Techniques for Design of Sculptured Surfaces

The historical emphasis in CAGD has largely ignored the question of techniques to aid in the design of sculptured surfaces. Representation issues, of course, had to be solved first. But the later emphasis on valid models caused sculptured surface design to be largely ignored, because the problem was difficult enough just with simple components.

One author [17] describes the design problem as typically requiring *structures* formed from various *components*. The simple primitive and Euler systems provide rich *structures* through set operations, but lack rich *components*. The surface techniques provide rich *components* at the expense of rich *structures*, since surface patches traditionally can only be formed into rectangular arrays of rectangular patches.

More recently however, as the capabilities and limitations of the systems developed so far are recognized, there is renewed interest in sculptured surfaces. And, although the area has not been a primary emphasis, there is a handful of useful techniques for designing sculptured surfaces which should be reviewed before considering the current directions.

The available techniques for designing sculptured surfaces can be roughly divided into the following categories:

1. Interactive control point movement (sometimes called the "picking and poking" technique)

2. Interpolation and approximation

3. Lofting schemes

4. Extensions of boolean set operations to closed sculptured surfaces

5. Specialized schemes for particular applications

3.4.1.1 Interactive Control Point Movement

Interactive control point movement is perhaps the most widespread technique for designing sculptured surfaces at research institutions [4, 23, 8, 27]. The development of high quality graphics displays and interactive devices such as light pens, tablets and mice, led naturally to widespread use of interactive positioning of points in space as a means for defining arbitrarily shaped surfaces. The points being manipulated could be either actual surface points, through which a surface was later fit (interpolation), or points which were fairly close to the surface and would mimic the general shape of the surface they defined.

The fact that all the interactive media tend to be two-dimensional, where the design problems are inherently three-dimensional, has always posed a serious problem with this technique. Some elaborate schemes for true three-dimensional display and interaction [9] were investigated in the 70's, with most of the research abandoned within a few years of its beginning. A recent revival of interest has produced commercial three-dimensional displays, but three-dimensional input devices are not generally in use.

More serious problems exist for the industrial designer who must produce a design for a specific product. This technique for surface design offers no natural way to specify measurements or dimensions. The positioning of the control points on the graphics screen is almost always a purely aesthetic choice made by the operator. Further, the number of points which must be positioned in order to achieve the desired shape can become quite large and the whole process can be a tedious and frustrating experience.

3.4.1.2 Interpolation and Approximation

This classical field of mathematics offers a wide variety of techniques for producing a surface which fits or comes close to a given set of data. Interpolation and approximation techniques are used widely in industrial sculptured surface design, possibly because so much design is done by adjusting an existing design for a new product. Interpolation provides a convenient way of creating a model of an existing product, which can then be modified by either adjusting the original data or using other techniques.

These techniques also have drawbacks for the designer. Interpolation has a surprisingly frequent propensity to produce unwanted undulations ("wiggles") in the resulting surfaces. In fact, much of the research in interpolation has been concerned with minimizing this effect. Interpolation and approximation also typically require large amounts of carefully prepared data, again making the production of a surface a very tedious task. Finally, the level of mathematics which a designer must understand in order to produce the desired

surface is much too high. Too many parameters, especially the ones involved with minimizing wiggles, may be difficult for the designer to understand and specify reasonable values for.

3.4.1.3 Lofting

Certain industries, most notably, the aircraft industry, have made almost exclusive use of lofting for the design of their products. There are as many lofting schemes as there are groups who use them, and new lofting schemes are fairly simple to derive for specific applications which desire to use them.

Most lofting schemes do share a few basic properties. Lofting generally consists of the design of a number of planar cross-section curves which correspond to planar cuts through the object at specified locations. The lofting process produces a surface which goes through all of those cross-sections, and behaves reasonably between the sections. The degree of continuity between the sections, the particular representation used, and all the other details depend entirely upon the application and the group designing the lofting scheme.

3.4.1.4 Set Operations on Complex Operands

The ability to combine sculptured surfaces using the set operations described for the volumetric primitive systems above has been predicted by many researchers as a breakthrough for the design of sculptured surfaces. One of the most promising aspects of this area in the last year has been the production of several theses which demonstrate that this capability is indeed possible and practical [28, 7].

The powerful capabilities of set operations have been convincingly demonstrated by the solid primitive systems, and it is clear that these operations will prove invaluable in the design of sculptured surfaces when they become tractable.

3.4.1.5 Specialized Techniques

All of the methods discussed so far can be thought of as rather general techniques which are widely accepted and used in many applications. The exception is perhaps the set operations on surfaces because the results are too new to have been extensively adopted, but there is no doubt of their utility and eventual widespread use.

However, there still exist many applications for which none of these techniques exactly fits the needs of the designers. When this situation occurs, a new approach is developed which is specific to the problem at hand. There are, of course, too many of these specialized methods to discuss or even to list here.

One of the difficulties with this collection of other techniques is that they are often presented with a narrow viewpoint. One technique cannot be combined with another because each was developed for a single, often specialized, surface representation. Some of the most intriguing techniques for designing surfaces do not produce a final surface representation - the surface design operators produce pictures of surfaces, but do not

actually define a geometric model. A surprising number of these specialized approaches do not use a true surface representation at any point. Instead they define modelling techniques which operate on polyhedral representations or on dense point sampled data, even when smooth surfaces are actually desired. A few efforts have been made to collect several techniques into one toolbox, but some of these have made the crucial error of using a natural but distinct surface representation to describe each method. This means that the designer can use only one of the available tools to design his entire object, unless he can provide a way of converting among the various representations, a formidable or even impossible task.

3.5 Computer Science CAGD at Utah

The range of research activities and interests is widespread in the Computer Science CAGD research group. We have developed a CAGD testbed, the Alpha_1 system. Fundamental concepts embodied here are the notion of a unified theoretical framework and representation, so that all processes may have access to any or all geometrical aspects of the model, and a hierarchical approach to object design. We feel that the use of separate models supporting different design and analysis processes is somewhat akin to the story of the elephant and the blind men. Each blind man is allowed to touch a different part of the elephant; each "sees" a different creature. Unfortunately, in such a case the whole is not really the sum of the separate parts. By having a single flexible representation for all objects designed, we hope to have each blind man, the various design teams, at least be able to touch all parts of the elephant. In this metaphor, the ultimate goal of a CAGD system should be to give vision to the blind man.

For the separate research components, the Alpha_1 realization offers rich sources of data to test out algorithms, systems integration problems to test out system design concepts, a graphical rendering capability to test out research on the role of visualization techniques for shapes and design, and benchmarks for new modelling tools. As research becomes mature we integrate those concepts into the testbed so all the other researchers can use the new capabilities in pursuit of their individual goals.

The first testbed component was a capability for high quality computer images of freeform surfaces. In the past, computer aided freeform surface modelling used systems which supported only line drawings of the models. While providing important information, these images cannot give the designer more than gross cues about smoothness, surface normal variation, and shape. High quality raster images, on the other hand, gave many visual cues as to shape, highlights, and normal variation, but were not easily and accurately available for surfaces defined by splines. It was felt that the designers would be able to use this visualization capability within the design cycle. Response has indicated empirical approval of this concept.

The impetus for building a system arose from the development of the Oslo Algorithms [11] for computing discrete B-splines, or d-splines as they are termed locally. Subsequent to this finding, Cohen and Riesenfeld postulated that the theory of discrete splines in conjunction with the Oslo Algorithms could form the foundations of a unified and cohesive CAGD system, a system in which there is a single uniform mathematical

representation throughout and a single way of evaluating objects. This approach essentially extends previous ad hoc subdivision schemes to a much more useful level of generality and provides a sound and rigorous theoretical basis for understanding these methods. This allows the system designer to achieve an overall degree of unification derived from the homogeneity of the representational and computational scheme. The resulting systems architecture, although based on a more complex theory, becomes conceptually and practically simpler.

The testbed serves as a study in whether a unified approach to geometry can help in the specification of an integrated system. We plan to use the Alpha_1 testbed at such time as theoretical advances make it possible to study some broader, but much more complex, related alternative representations and algorithms [13].

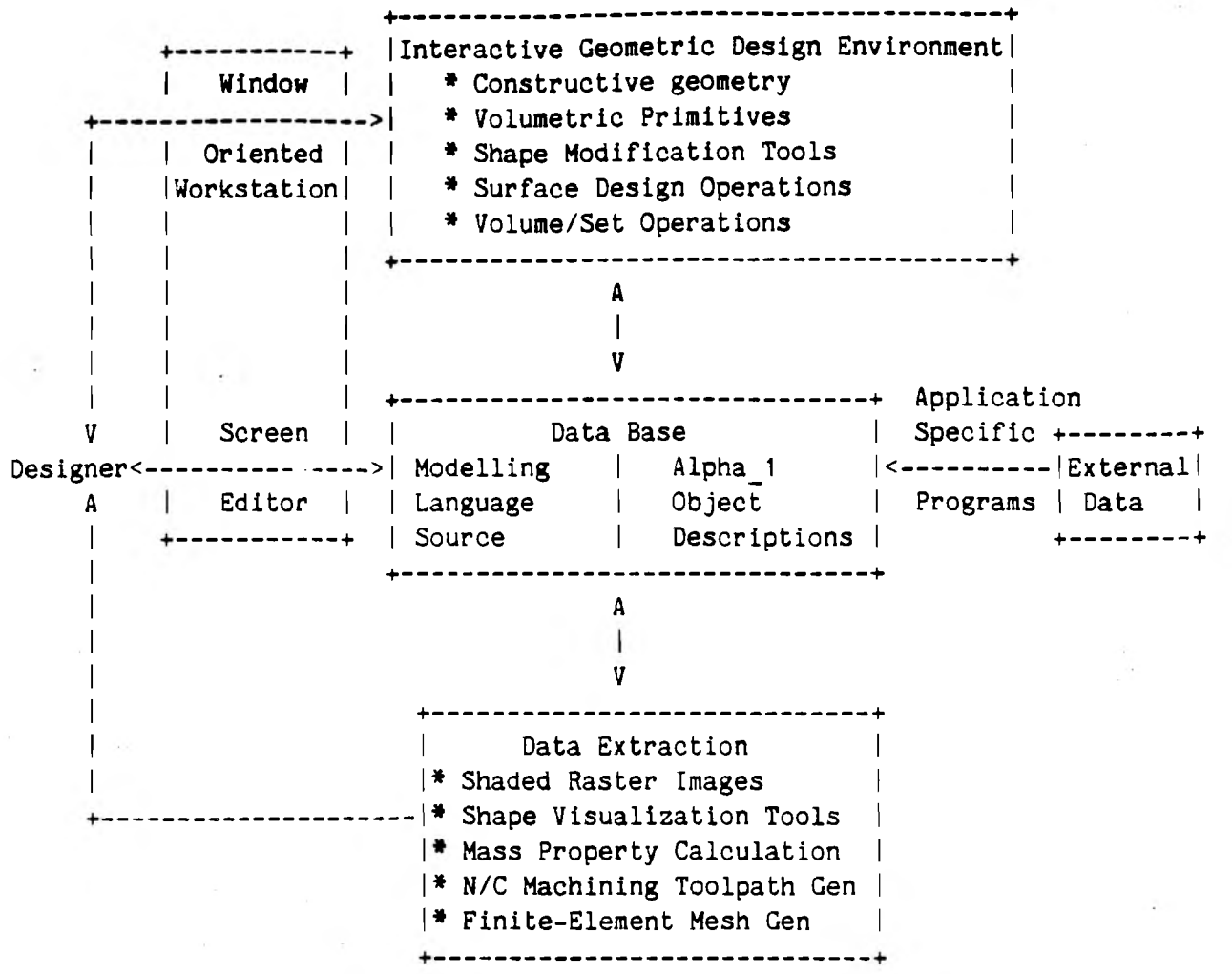
The research project proposes to continue to stay in close touch with actual, real-world problems, and has used these experiences to guide its ambitions for the kinds of support that it feels designers want and require. Trying to model them with a high degree of faithfulness to the specifications and always analyzing the instances when compromises were necessary, we have taken on several projects of substantial geometric modelling sophistication. This approach has led to research and creation of design tools and algorithms which result from a considerably enhanced understanding among the researchers of what is actually desired by designers and engineers.

Below is a section on the research issues that we pursue. To better understand them, we also must devote research effort to keeping our testbed, Alpha_1, at the leading edge of what is possible.

So far the Alpha_1 System has been developed on general purpose computing and graphics equipment. It is now felt that a sufficient understanding of the major components of CAGD systems architecture exists so that we can begin to factor out key algorithms and explore their hardware realizations. Because of the unifying homogeneous theoretical structure of Alpha_1, some critical algorithms which are heavily used can be factored out for hardware speedups. Just as realtime voice synthesizers have led to new ideas and algorithms for speech processing, when the hardware becomes functional we forsee a surge in research whose exact emphasis at present is even to predict.

3.5.1 Philosophy of the System Design and Configuration

This section describes the design philosophy underlying the Alpha_1 testbed system. The figure is a block diagram of the planned configuration; the individual components are explained in more detail below.



Alpha_1 Testbed System Configuration

The Alpha_1 system has as its core object descriptions in two forms. One should be allowed to describe an object by a program, or set of procedures in the modelling language, or by a numeric object description form. The modelling language provides for parametric, or procedural, object modelling so that an entire family of parts may be represented by a single procedure. For example, the primitives normally provided by a CSG system, such as spheres, cones, and cylinders, are defined as procedures. Evaluation of the procedural definition of an object produces a file containing a B-spline boundary representation of the object. This file can be used by the data extraction programs to, for example, produce a shaded image of the object.

The designer interacts with the data base through a window-oriented workstation with both graphics windows and text windows. He can make a change to the model source and immediately see the result of the change in a graphics window. The geometric design environment should provide many tools for the designer. He might have available

to him drafting-type operations with points, construction lines and arcs, that can be fleshed out with surfaces; volumetric primitives from which to build objects; direct manipulation of the B-spline control mesh defining the boundary of his object; shaping functions such as bend, taper and warp; and, finally, boolean set operations.

While the processes described above are fine for *ab initio* design, sometimes an object which has been produced by some other process must be represented. A good example is the airfoil of a turbine blade. The shape of this is rigidly specified by aerodynamic considerations, and is usually produced by a specialized system. Application specific programs can be written to convert such external data.

Once an object has been designed, a number of data extraction tools may be applied to it. Perhaps the most noticeable is the production of a shaded raster image of the object. In order to produce the most realistic possible image, the image rendering process has a host of options. The object may be lit with lights in several directions, its surface finish and color can be precisely specified, etc. A user-specified resolution value determines the accuracy of the picture, this may be set to a coarse value for a quick look, or to a fine value for a final image.

A number of shape visualization tools should be available to the designer. The use of various semi-realistic imaging techniques would enable him to better investigate and understand the object, as in the case of coloring an image of an object to map the values of non-geometric data such as temperature.

Another data extraction tool can calculate mass properties of the object, including volume, surface area, moments of inertia, and mass. The production of N/C machine cutter toolpaths for manufacturing an object, and the generation of meshes for finite-element analysis are other desirable data extraction tools.

3.5.2 Our Areas of Research Interest and Approaches

3.5.2.1 Long Range goals

The long term goals of the project are to treat the problems inherent in computer modelling the 3-D geometry of complex parts and the systems implications thereof, and then use this shape information to enter into the realm of predictively simulating the real-world behavior characteristics of a mechanical model. In such a scheme a model object would be imbued with knowledge about its critical functions in the world and be able to perform in reasonable accord with the rules which govern it, at least in the design, analysis, and manufacturing contexts. The medium range project goals involve speeding up a key component of the system: the process which converts the fundamental spline-based model to simple elements suitable for rendering, mass property calculation, and the like.

It is our philosophy that the most useful geometric design and modelling systems will be ones that exhibit an integrated approach which allows for many different processes, somewhat like a workbench with many different tools available. The mathematical

representation used in defining the object is the workbench for the system, and the glue that holds it together. It provides the common medium for all of the tools to work on or build up. A single flexible representation which can embody the correct geometry of the most interesting, difficult, types of mechnical parts can be used to build interfaces to mathematical analysis packages for stress, heat, pressure, aerodynamic, hydrodynamic, etc, to perhaps form an interactive loop between design and analysis so designs can quickly reflect the latest analytical results. The man machine interface, the "front end" of the design system must be able to give the designer enough flexibility to meet his design criteria for the model without flooding him with useless parameters.

The emphasis of our present and future research falls into several broad categories on which we shall elaborate.

3.5.2.2 Modelling Tools Based on a Unified Representation

At present, the nonuniform tensor product B-spline surface is used as the representional workbench. The B-spline curves and surfaces are fairly simple to work with, and very powerful in their ability to describe a wide range of shapes. Efficient computational tools now exist for manipulating and evaluating B-spline curves and surfaces.

While the Bezier and B-spline points $\{P_{i,j}\}$ and their associated meshes give a rough idea of the surface and allow manipulation by poking the points, it is rare that the designer would know at the beginning of the design cycle enough about the surface to give the initial surface enough degrees of freedom. And he certainly would not, in general, be able to position the control points in the right locations. Just as in the mind of the designer where a design is first a rough sketch which progressively becomes more refined and detailed, the computer aided sculptured design process should allow for progressive refinement and detailing as the design cycle continues.

The capability to arbitrarily add degrees of freedom to a tensor product spline was introduced by Cohen, Lyche and Riesenfeld with the Oslo Algorithm [11]. The basis for this algorithm is in the theory of discrete splines, another type of "spline" defined over a discrete domain. The duality provided by this theory between continuous and discrete splines creates a unifying theoretical basis for many operations.

The Oslo Algorithms also permit arbitrary spline subdivision, that is, the process of looking at one spline as if it were composed of two, or more, shorter (in parametrization) splines with smaller extent. Using this procedure recursively together with the convex hull property and the variation diminishing property has allowed us to generate temporary graphics databases useful as input to apply high quality computer graphics modules capable of rendering three dimensional freeform surfaces. One can hypothesize that this same type of procedure would support the generation of temporary derived models necessary for analysis procedures. It could perhaps support a feedback spiral between the analysis package and the geometric model to give very fine and high quality information where necessary, and less information where less is needed. The Oslo Algorithms are also used to support boolean operations on freeform surfaces.

While the complexity of designing with nonuniform parametric B-spline curves is greater than with simpler uniform splines, Bezier curves, or other piecewise continuous polynomial forms, it also carries much higher flexibility. The B-spline basis carries the continuity conditions, so the user does not always need to remember them. Since refinement and subdivision are inherently a nonuniform procedure they also fit within this framework.

Set operations on objects are a natural paradigm for many design realizations such as drilling a hole or hogging out material with an N/C cutter. The intersection of a sphere and a cylinder or a box and a sphere can be computed analytically, although implementationally it may not be sufficiently stable so numerical algorithms are frequently used. The intersection of two freeform surfaces is much more complex. Intersecting bi-cubic patches analytically then this results in the following:

The implicit represesentation of each patch is a trivariate implicit equation of degree 18, in other words of the form $\sum_{\substack{i,j,k\geq 0 \\ i+j+k\leq 18}} c_{i,j,k}x^iy^jz^k = 0$. The intersection is then the common solution of these two equations, but only for (x,y,z) within the parametric range, not the infinite surface. Another approach is to implicitize one and substitute the parametric equations for the other for x,y, and z. The result is an implicit bivariate polynomial in the parameters for one surface of degree 108, which could potentially have approximately 300,000 coefficients! Solving analytically for explicit ranges is clearly not feasible in any general way. Thus, geometric and numerical approximations would have to be used in any practical system. This new capability mentioned above, to use set operations with

In [28] an extended combinational algebra is proposed for partially bounded objects. These objects might be intermediate stages towards a final realizable object, or to be used in new type operations necessary to make "cut away" views or a "cookie cutter"

The concept of partially specified freeform objects with their own combinational logic is a new area of research, as is defining solid objects from partial surfaces using these extended boolean combinations to obtain a final whole. Until recently there was not any algebra for combining partially defined objects as intermediate steps to realizable objects Correct and proper paradigms for modelling with them and stable realizations of the algorithms in software and hardware are two major areas of research.

Traditionally the design of freeform surfaces has stood apart from the specification of the solid object. Hence, the making of realizable objects out of analytic definitions of tensor product B-spline surfaces requires many design paradigms; some of which are not yet invented. All of the general techniques for designing sculptured surfaces which were discussed above can be applied to surfaces represented with B-splines. Interactive control point movement has been most widely used with B-spline surfaces or their special subclass, the Bezier surfaces. Most interpolation and approximation techniques which are based on polynomials are easily adapted for B-splines, and lofting schemes can readily be derived for B-spline surfaces. Further, the recently developed set operation algorithms for general surfaces are all based on B-spline representations or on

freeform surfaces is still young, and its full utility and characteristics are not yet known.

subclasses. Potentially, a system based on B-splines can offer a designer a choice of the traditional methods for design, in addition to the newly developed set operations in a system incorporating freeform surfaces.

As noted before however, these traditional approaches, while quite powerful for many applications, fall short for other applications. Even where they can be used, they are often cumbersome and difficult for designers to understand. In order for designers of sculptured surfaces to be able to define easily and naturally their desired shapes, a new set of tools must be provided *in addition* to the existing methods. It is crucial that at any point in the design process, the designer can switch easily to the tools which are most convenient for the design of a particular portion of the object. A unified representation is required if these conditions are to be met.

An important step in providing such a flexible design system is the investigation of the various specialized approaches that have been developed to handle the areas where the traditional methods failed. This research must proceed with the view of adapting and integrating a wide variety of specialized, but flexible, tools for an environment based on the unified representation. The research must also look for totally new techniques for surface design which can be incorporated into the unified approach. Before Computer Aided Geometric Design systems for sculptured objects can come into widespread use and realize their full potential as an aid to design and manufacture, a variety of tools must be created which can be used compatibly, conveniently, and naturally by many designers within a single modelling system.

3.5.2.3 Representations

The nonuniform B-spline tensor product is the most versatile surface representation available for design. Its good attributes include the variation diminishing property, the convex hull property, refinement capability which supports a "top down" approach to design and creation of modelling tools, and a subdivision capability on which is based extended boolean operations. These properties together form the basis for the first experimental solid modelling system based on sculptured surfaces that incorporates the "simple" primitive objects.

However, there are substantial and important reasons in computer aided geometric design to seek more general, intrinsically multivariate, surface forms for some of the following reasons. Deriving from its cartesian product construction, the tensor product formulation is a mapping of a rectangular domain. Tensor product surfaces fit naturally onto rectangular patches, or "topologies" as it is called, but can become problematic when applied to other regions with three sides or five sides, for instance. While four sided patches can be fit to nonrectangular regions by collapsing the length of a side or combining a number of rectangular patches, such solutions often are inelegant and unsatisfactory. They are inelegant because they may involve degenerate sides or other special contortions that are nonsymmetric and extraneously introduced. They can be unsatisfactory because they may introduce secondary problems through degeneracies or other anomolies that require the specification of data which is not normally available as part of the original problem. In addition, tensor product surfaces have an explicit orientation and special parametric or coordinate direction associated with each

independent variable. They also require the specification of a full matrix of coefficients, regardless of what is appropriate to the particular application that gives rise to the need for the surface. It often occurs that more data is required then is convenient or reasonably available, so the user is forced into employing contrived methods to produce the remaining data that the scheme demands.

There have been many special solutions proposed for various types of special regions, especially for triangles. Such solutions may be satisfactory in some primary respects, but they are still not completely satisfactory in other respects. Some of the available solutions, which exist only in explicit form $z = F(x,y)$, are not generalized to parametric form $S(u,v) = (x(u,v),y(u,v),z(u,v))$. They also constitute a special surface element that is likely to be different from the rest of the surface form. Further, here too it frequently occurs that more data is required than is available, so the user must contrive methods to produce it. The use of a differently defined element in the middle of a homogeneous region can cause some global implementation problems because it requires special treatment by other processes that operate on the surface representation.

Properties of the Ideal Multivariate Representation

Clearly, the capability to represent intrinsically tensor product and nontensor product regions in a way faithful to the geometry is most desirable. However, beyond the desire to attain elegance, there is a clear advantage in a CAGD system to have a homogeneous representation on which all other processes can operate in a consistent and straightforward manner. There are implementation advantages to having a single surface form that perhaps specializes to the common tensor product, but that introduces a single uniform representation throughout the system for all surfaces.

For computational stability and efficiency and for interactive design utility, a surface basis element should have nonnegative values over a locally supported region, if possible. The B-spline basis is such a set of basis functions in the univariate case: a higher dimensional analog would be very attractive. A convex hull property that occurs when the basis functions sum to one and are nonnegative aids considerably in the task of developing various set operations like surface intersection. The convex hull property can be used to simplify hierarchical divide-and-conquer schemes by allowing the implementor to perform a super test for intersection of convex hulls, and then subdivide on an positive answer indicating a possibility of an actual surface-surface intersection. Such sets of functions might have a shape preserving approximation property, an attribute which has proved to be desirable when it is present in a mathematical representation applied to interactive design. The space that is spanned should be a useful space of functions like a piecewise polynonial space of a certain continuity class which forms a generalization of a univariate spline. As in the univariate case, the lower the degree the more attractive the set for CAGD. A fast and stable computational algorithm must be available to place any potential multivariate scheme into candidacy. A subdivision algorithm has come to be an important implementation approach, therefore such an algorithm would probably have to be developed if it were not part of a computational method for general elements of this space.

Tensor product domains are well served by the tensor product B-spline basis. Such a set of functions over a general topology would support a new kind of design scheme in which the control points might be either regularly or arbitrarily spaced. Even then, before integration into a system, considerable experimentation would be necessary to determine what kinds of constellations of control points would lead to tractable and applicable design schemes, and which ones would be of academic interest only.

3.5.2.4 Systems Approach

Using modern computer science and software engineering principles and tools in the design and implementation of an experimental geometric modelling system is an important factor in the development of a modular, robust, and extensible testbed which supports the research into mathematical representations, interactive design paradigms, and numerical processing of geometric models. While not actively involved in developing the fundamentals of such techniques, we are pioneering their application to the design, implementation, and maintenance of a large geometric modelling system.

From the first, our testbed has been designed as an integrated system with the great advantage of using a single powerful and computationally stable family of geometric representations and algorithms. Modularity is incorporated in the design by structuring it as a set of "packages" with clean interfaces, which facilitates piecewise incremental evolution and eventual factoring of the system to use specialized hardware.

The software production environment provides advanced tools combined with mobility between computer systems in this period of explosive change. The Unix[1] family of operating systems has exceeded our expectations as an implementation base, both in the number of available, compatible implementations over a range of machines from supercomputers to workstation microprocessors, and in the high quality of the software engineering tools available in the Unix environment. To a large extent, our work is the beneficiary of the active research in software development tools being performed on Unix systems.

The choice of programming languages is crucial to the eventual success of a large system such as a powerful geometric modeller. The language must provide modern structured control and data structures and a module packaging facility. Most implementations of Pascal are not capable of packaging and are more suited to small programs. Ada is not yet available in a production programming system. We chose C as our base level implementation language due to its high quality and proved performance in implementing the Unix operating system and utilities. The C programming paradigm we have adopted includes tools that facilitate dynamic storage allocation of lists, trees and networks of objects, and object-oriented programming with "generic operations", avoiding the familiar disease of FORTRAN programs that are riddled with fixed array limits and cannot easily be extended to new data types or operations.

---

[1] Unix is a tradmark of Bell Laboratories

For an interactive geometric model construction environment, it is necessary to use a language implementation which provides an interpretive, extensible high-level command interface together with graceful descent into compiled code and effecient means of driving high-performance interactive graphics devices. Our approach involves the development of an interactive modelling environment incorporating window oriented display management and event driven graphical interaction embedded in the Portable Standard Lisp system developed at Utah.

Finally, the use of software engineering "meta-program" concepts to manage system evolution is adhered to in our software development paradigm. In particular, we incorporate the Unix tools for automatically synchronizing source code changes with executable compiled modules, allowing different researchers to experiment with variations on the system without impacting each other or the base system and automatically merging the tested extensions into the base system.

This approach has led to the often surprising ability of our experimental research testbed, Alpha_1, to accommodate the addition and modification of experimental datastructures and algorithms.

3.5.2.5 Interfacing to Analysis packages

Finite element packages exist to analyse a model's generalized structural, thermodynamic, aerodynamic, or hydrodynamic characteristics, as well as to perform flow analysis to measure suitability for injection molding. The ultimate goal of simulating real object performance by a geometric model requires interfacing with analysis packages for feedback in many domains which require integrated analysis of the entire model together with its environment.

The geometrical shape is typically just one component of the total description required by an analysis package. Material properties, initial conditions, and environmental loadings must all be provided in the model for extraction as analysis input. The detailed form of these "attributes" will vary between analysis domains and even between packages in the same domain.

Output from currently available analysis packages is typically produced as a voluminous serial file, originally intended for a line printer. Parsing the output stream will be a significant problem for the analysis package interface, since it must also associate back from the internal element node numbers used by the analysis package to locations on the parametric surfaces of the original geometric model.

The analysis result could augment the model with additional attributes or attribute distributions, which could be displayed to the designer in appropriate ways such as distorted geometry or continuous variation of model color. Analysis attributes integrated into the model could also be used to guide automatic iterative design procedures which optimize a design across multiple analysis runs.

3.5.2.6 Hardware Integration

We think that realization of intrinsic geometry algorithms in hardware would magnify the effect of the theoretical computer geometry results. First, the process of specification of what geometric processes are "basic" or most frequently needed during the design process and the employment of inherently parallel structures in modelling algorithms would lead to a decrease in the effort needed to model an object. Just as Von Neumann architectures in computers lead to certain types of numerical methods, system integration between software and hardware components will lead to new design paradigms and tools. They will encourage the development of new modelling algorithms, tools, and representations which can build upon the hardware. Further the hardware realizations might have more widespread use. Such a candidate would be a hardware realization of " exact" or "arbitrary precision" arithmetic. The processes of subdivision, extended boolean operations, and rendering all require many arithmetic operations. Since the implementations now use floating point arithmetic, the errors make it difficult to determine equality. The question of "Is this point on two surfaces?" becomes impossible to answer exactly. In fact, if the simpler question "Is this point on the specified line?" can have two answers if the computations are done in two different ways.

Further, other frequently used operations which are computationally intensive slow the interaction response time. It would be desirable to speed up these processes so that the designers will wait for the response and be interested enough to interact immediately.

3.6 Example: Preliminary and Detailed Turbine Engine Blade Design

Another example illustrates how such design systems would fit together in layers. Design detailing tasks receive design goals from preliminary designs of the overall system, which must in some cases specify the performance of components, rather than deriving the performance from as-yet-undesigned component geometry. Analysis of an aircraft preliminary design must assume reasonable values for the thrust, weight, and size of the engines, rather than simulating the function of every engine component.

Similarly, the preliminary design of the turbine engine must specify the goals of the turbine stages in terms of work put into the gas flow by compressor stages or extracted by turbine stages. An interactive design environment capable of analyzing such a specification in a variety of operating speeds and temperatures and displaying the simulation results in real time would be valuable in honing the preliminary design.

In the detailed design, e.g. of the turbine blades of a single stage, the preliminary design specifications for the stage would guide the development of blade aerodynamic geometry which meets the design goals under gas-flow analysis derived from the preliminary simulations. Then the structural and thermodynamic analysis and simulation of the blade in the gas-flow environment would interactively guide the development of internal cooling passage geometry of hollow blades and the mechanical attachment geometry where the blades are joined to the disk.

3.7 References

1. Armit, A. P., "A Multipatch Design System for Coons' Patches," *IEEE Conference Publication No. 51*, April 1969.

2. Baer, A.; Eastman, C. M.; and Henrion, M., "Geometric Modelling: A Survey," *Computer-Aided Design*, Vol. 11, No. 5, September, 1979, pp. 253-272, Also Research Report No. 66, Institute of Physical Planning, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March 1977

3. Barnhill, R. E. and Riesenfeld, R. F. (editors), *Computer Aided Geometric Design*, Academic Press, 1974.

4. Bezier, P. E., *Mathematical and Practical Possibilities of UNISURF*, Academic Press, New York, 1974.

5. de Boor, C., "On Calculating with B-splines," *Journal of Approximation Theory*, Vol. 6, No. 1, July, 1972, pp. 50-62.

6. Boyse, J. W. and Gilchrist, J. E., "GMSolid: Interactive Modeling for Design and Analysis of Solids," *IEEE Computer Graphics and Applications*, Vol. 2, No. 2, March, 1982, pp. 27-40,

7. Carlson, W. E., "Techniques for the Generation of Three Dimensional Data for Use in Complex Image Synthesis," Ph.D. dissertation, Ohio State University, 1982.

8. Clark, J. H., "Designing Surfaces in 3-D," *Communications of the ACM*, Vol. 19, No. 8, August, 1976, pp. 454-460.

9. Clark, J. H., "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM*, Vol. 19, No. 10, October, 1976, pp. 547-554.

10. Cohen, Elaine, "Some Mathematical Tools For A Modeller's Workbench," *Computer-Aided Geometry Modeling*, Langley Research Center, National Aeronautics And Space Administration , Hampton, Virginia, 1983, pp. 195-200.

11. Cohen, E.; Lyche, T.; and Riesenfeld, R. F., "Discrete B-splines and Subdivision Techniques in Computer-aided Geometric Design and Computer Graphics," *Computer Graphics and Image Processing*, Vol. 14, No. 2, October, 1980, pp. 87-111, Also Tech. Report No. UUCS-79-117, Department of Computer Science, University of Utah, October 1979

12. Coons, S. A., "Surfaces for Computer-Aided Design of Space Forms," Tech. report MAC-TR-41, Project MAC, M.I.T., Cambridge, Massachusetts, June 1967. Available as AD-663 504 from NTIS, Springfield, Virginia

13. Dahmen, W., "On Multivariate B-splines," *SIAM Journal of Numerical Analysis*, Vol. 17, No. 2, April, 1980, pp. 179-191.

14. Dahmen, Wolfgang and Micchelli, Charles A., "Multivariate Splines - A New Constructive Approach," *Conference on CAD - Oberwolfach*, Conference on CAD - Oberwolfach, W. Germany, April 1982.

15. Eastman, C. M. and Weiler, K. J., "Geometric Modelling Using the Euler Operators," *Proceedings of the First Annual Conference on Computer Graphics in CAD/CAM Systems*, M.I.T., 9-11 April 1979, pp. 248-254, Also Research Report No. 78, Institute of Physical Planning, Carnegie-Mellon University, Pittsburgh, Pennsylvania, February 1979

16. Faux, I. D. and Pratt, M. J., *Computational Geometry for Design and Manufacture*, Ellis Horwood Ltd., 1979.

17. Forrest, A. R., "A Unified Approach to Geometric Modelling," *Proceedings of SIGGRAPH '78*, ACM, August 1978, pp. 264-269.

18. Gordon, W. J. and Riesenfeld, R. F., "Bernstein-Bezier Methods for the Computer-Aided Design of Free Form Curves and Surfaces," *Journal of the ACM*, Vol. 21, No. 2, April, 1974, pp. 293-310, Also Research Publication GMR-1176, General Motors Research Laboratories, March 1972

19. Gordon, W. J. and Riesenfeld, R. F., *B-spline Curves and Surfaces*, Academic Press, New York, 1974, pp. 95-126.

20. Requicha A. A. G. and Voelcker, H. B., "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications*, Vol. 2, No. 2, March, 1982. pp. 9-24.

21. Requicha, A. A. G. and Tilove, R. B., "Mathematical Foundations of Constructive Solid Geometry: General Topology of Regular Closed Sets," Technical Memo 27, Production Automation Project, University of Rochester, Rochester, N.Y., March 1978.

22. A. A. Requicha, "Representations for rigid solids: Theory, methods, and systems," *Computing Surveys*, Vol. 12, No. 4, December, 1980, pp. 437-464.

23. Riesenfeld, R. F., "Applications of B-spline Approximation to Geometric Problems of Computer-Aided Design," Ph.D. dissertation, Syracuse University, May 1973. Available as Tech. Report No. UTEC-CSc-73-126, Department of Computer Science, University of Utah

24. Riesenfeld, R. F., "Aspects of Modelling in Computer Aided Geometric Design," *Proceedings of the National Computer Conference*, AFIPS, May 1975, pp. 597-602.

25. Riesenfeld, R. F., "Homogenous Coordinates and Projective Planes in Computer Graphics," *IEEE Computer Graphics and Applications*, Vol. 1, No. 1, January, 1981, pp. 50-56.

26. Riesenfeld, R. F.; Cohen, E.; Fish, R. D.; Thomas, S. W.; Cobb, E. S.; Barsky, B. A.; Schweitzer, D. L.; and Lane, J. M., "Using the Oslo Algorithm as a Basis for CAD/CAM Geometric Modelling," *Proceedings of the Second Annual Conference of the NCGA*, Aangeenbrug, R. T., ed., National Computer Graphics Association, Inc., Baltimore, 14-18 June 1981.

27. Rogers, D. F. and Satterfield, S. G., "B-spline Surfaces for Ship Hull Design," *Proceedings of SIGGRAPH '80*, ACM, July 1980, pp. 211-217.

28. Thomas, Spencer Woodlief, "Modelling Volumes Bounded by B-spline Surfaces," Ph.D. dissertation, University of Utah, Sept 1983.

29. Voelcker, H. B.; Requicha, A. A. G.; Hartquist, E.; Fisher, W. B.; Metzger, J.; Tilove, R. B.; Birrell, N.; Hunt, W.; Armstrong, G.; Check, T.; Moote, R.; and McSweeney, J., "The PADL-1.0/2 System for Defining and Displaying Solid Objects," *Proceedings of SIGGRAPH '78*, ACM, August 1978, pp. 257-263.

4. VLSI Component

4.1 Introduction

This research will be centered on the development of a design methodology known as "structured tiling" which can be used to help bridge the gap between system-level descriptions and the composite layout of an integrated circuit. Structured tiling incorporates knowledge about the best known implementations of specific integrated circuit structures in a given technology (such as dynamic NMOS RAM which requires very specialized design techniques) into an overall scheme for designing and building systems in VLSI. This methodology permits the design of a set of unified CAD tools which map system-level descriptions to a set of well-known and throughly tested functional blocks (tiles). These tiles may themselves be hierarchically constructed from lower-level tiles, with the lowest level tiles being actual composite layouts. Each level of tiling above the composite level is an easily understood functional element and requires little knowledge of transistor level descriptions. This enables system designers to avoid many of the problems inherent with transistor and composite level design.

Structured tiling permits the use of advanced technologies by inexperienced designers of integrated circuits. For example, CMOS requires that substrate and P-well contacts be placed in specific locations in order to prevent latchup. Another example is the strict device orientation with respect to crystal lattice orientation in order to prevent back-gating effects in GaAs devices. The use of tiling hides such layout constraints from the system designer, reducing the knowledge he must have of the underlying technology.

Some tiles are best described as parameterized modules. For example, an n-bit shift-register (which at one level is thought of as a single tile) is best thought of as a collection of single shift-register bits and interconnecting tiles. This n-bit shift-register tile can be parameterized to meet certain performance, shape, and I/O placement requirements. Structured tiling permits the development of CAD tools which can incorporate such parameterized modules in circuit designs.

An analogy can be drawn between programming and circuit design. The binary 1's and 0's used in machine language programming are analogous to the rectangles (CIF descriptions) used in the composite layout of integrated circuits. Programming in hexadecimal instead of binary machine language is similar to the placement of more symbolic circuit structures such as the entire contact hole structure or even an entire transistor [11]. A significant amount of research at this level is presently being done [6, 1] that is related to this level of design. Some of the schemes which have been developed also provide routing (through the addition of wires or stretching of cells) but are still directly and explicitly concerned with transistor placement and interconnection, and have not yet moved to the functional level. Integrated systems design is still largely done at the machine-language or hexadecimal levels.

In the spectrum of programming languages, assembly language programming permits the development of larger systems than are possible with machine-level programming. The programmer is no longer concerned with the exact meaning of a single binary 1 or 0 and

can concentrate more on the function of the program. At its lower levels, structured tiling is analogous to assembly language programming -- with the beginning of an emphasis on function. At this level, however, a reasonable amount of knowledge about the underlying machine or technology is still required. The Path-Programmable Logic [14, 7] methodology is an example of this level of design. High-level programming languages can be thought of as analogous to the higher levels (system levels) of the structured tiling approach to integrated systems design. At this level, technology and circuit-level implementation dependencies are completely factored out of the system description. An example of this is the research on the automatic implementation of state-machines from behavioral specifications [2, 3].

The development of the structured tiling methodology naturally requires an evolution of ideas. Just as computer programming began with binary machine code and progressed to high-level, portable languages, so integrated systems design will progress from composite layout to true system-level design. The research of the last several years by the VLSI group at the University of Utah has formed a foundation for the development of structured tiling.

The general objectives of this research are to:

* develop structured integrated circuit design techniques for technologies other than NMOS (i.e. CMOS and/or GaAs),

* implement a structured tiling approach to data-path design through the use of:

    - computational tiles (arithmetic),

    - parameterized switching and storage tiles (modules).

4.2 The Definition and Design of Structures in CMOS

The research work on structured tiling for integrated circuit design requires the development of predefined cells (tiles) for use in current technologies. Present work at the University of Utah [13, 8, 12] has defined a type of structured logic known as Path Programmable Logic (PPL) which will be used as the basis for the development of the new structures. We propose to extend the present techniques to include new static and dynamic structures in a two-layer metal, short-channel CMOS technology. The basis for this effort is described in an attached paper [14].

Essentially any general structure can be designed using the present PPL methodology. This has been demonstrated by designs performed for the Ada-to-Silicon project [9] (the Read_Init_Parameters Chip) and by a number of student projects here at the University of Utah in the VLSI design classes (i.e. the design of an area-filling RAM for graphics applications done by Eric Brunvand whose report is attached). These designs demonstrate that there are at least two major problems associated with the present

techniques which must be addressed in order to successfully implement the proposed Alpha-1 system. These problems are (1) power and performance of the present NMOS circuits and (2) the definition of data path modules and special purpose structures to complement the present structures. The rest of this section is concerned with the resolution of problem (1). The second problem is dealt with in the following two sections.

The present PPL cell set is designed using a four-micron NMOS silicon-gate process with quasi-static circuits. A 200 by 200 mil chip contains approximately 100 columns and 200 rows of PPL cells and will consume approximately 500 milli-watts of power. This 100 column by 200 row circuit is roughly equivalent to a circuit which contains approximately 15,000 two-input gates [14]. These circuits will operate with maximum clock frequencies of approximately 4 MHz. The extension of these techniques to the CMOS process should give a factor of 4 decrease in size and an order of magnitude increase in speed.

4.3 The Definition and Design of Structures in GaAs

We will choose a few simple circuits from the hardware description of the Alpha_1 machine which require very high speed. These circuits will be designed using the GaAs process and design rules which are available to DARPA contractors through Rockwell at Thousand Oaks, Calif. The circuits will be fabricated using the three process runs which have presently been committed to DARPA. These designs in GaAs will be full custom circuits designed using our Computervision CAD machine to take advantage of the speed in the GaAs process as opposed to the use of structured logic which will impose limitations on high speed performance. One advantage, however of the structured approach is the development of modules which have been completely verified in previous experiments. Because of the developmental nature of the GaAs process, the approach of using predefining structures may be valuable if no significant speed disadvantage is incurred. This definition of structured logic will probably not be attempted until the GaAs process is mature (within a few years).

4.4 A Structured Tiling Approach to Arithmetic Unit Design

The successful implementation of the Alpha_1 engine will require hardware implementations of algorithms (such as the Oslo algorithms) which demand high precision, efficient arithmetic processing. In order to achieve this, the use of rational arithmetic will allow for exact representations of numbers, thus solving many of the arithmetic problems encountered in the current software version of Alpha_1.

The development of high precision rational arithmetic processors will most probably require a *digit-slice* approach to the problem where the range of the operands can be easily extended either by parallel interconnections of such processors or by serial interconnection. In her dissertation [5], Chow has proposed an implementation of a such a variable precision arithmetic processor capable of arbitrary precision calculation of the four basic arithmetic computations in essentially constant time. She has also developed algorithms for using the processor for arbitrary precision floating-point operations. A

similar approach would have to be taken in developing a *variable precision rational arithmetic processor*.

The variable precision arithmetic processor makes use of the concept of digit sets. Building on this concept of digit sets, Robertson has developed a theory of decomposition of structures for binary addition and subtraction [10]. Using this theory, the variable precision arithmetic processor is easily designed in terms of much simpler elements which can be directly and easily implemented as integrated circuit tiles. Some preliminary work has been done by Carter at the University of Utah on this subject and has led to an NMOS implementation of the combinational logic portion of the processor for a single radix-16 digit-slice [4].

Simulations of the combinational logic portion of the digit-slice indicate that integer addition, subtraction or multiplication can be done in almost constant time over arbitrary operand widths. A paper, by Carter, entitled "The Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique" treats some of the aspects of this work and is attached.

The development of the arithmetic portions of the Alpha_1 architecture is therefore twofold in nature. First, a detailed understanding of the applications of rational arithmetic processing must be gained. This includes both the use of rational arithmetic algorithms in computations central to Alpha_1 (such as the Oslo algorithms) and the architectural structure of a variable precision rational arithmetic processor. Second, the physical implementation of the rational arithmetic processors must be relatively easy and error free. The application of Robertson's theory of decomposition and a more refined *structured tiling* approach to integrated circuit design as described by Carter [4] will permit the quick and efficient implementation of these processors in integrated circuits. CMOS implementations of the rational arithmetic processors will receive the primary emphasis with NMOS serving as a technology for the early prototyping of key structures.

4.5 Parameterized Switching and Storage Tiles in Structured Tiling

In addition to computational tiles, the development of the Alpha_1 engine will require switching and storage tiles. These include the following general purpose tiles: RAM, ROM, Registers, multiplexors, etc. These parameterized tiles will be formed in such a way that they can be integrated directly with the a surrounding PPL program. Some of the parameterized tiles may be constructed from PPL cells. Others will require new custom designed cells which may or may not conform to the PPL grid. The tiles will be considered to be single PPL cells which may be manipulated and placed with existing tools. Thus, this portion of the research will have three major thrusts:

* The identification of the parameterized tiles to be implemented and the definition of a library of low-level cells for each parameterized tile. These will be designed so that by combining these cells together, a module results which can be integrated onto a common grid with PPL cells and other modules.

* The development of algorithms and corresponding CAD programs which can use the low level library of custom and PPL cells to generate the parameterized switching and storage tiles. ASSASSIN fits into this category. Other generators might include those for RAM, ROM, etc.

* The modification of the current design tools [8, 13, 7] to allow for the manipulation of these parameterized tiles as cells for editing, simulation, etc.

EARL and ALI are two currently available procedural 'layout' languages which could be used to design the custom low-level cells and the parameterized layouts by not defining any constraints thereby preventing stretching. However, they are difficult to interface to our existing symbolic editing, simulation, and compaction tools. The difficulty is primarily due to EARL and ALI's use of geometric layout data rather than symbolic functional data.

These switching and storage tiles will be parameterized such that the 'user' of the cell (human or machine) will supply the necessary parameters to customize the tile. For example, such a tile might be a RAM block. The parameters might include: address width, word width, module shape, output drive requirements, performance requirements, refresh type (dynamic or static), and location of inputs and outputs. Using these parameters, the CAD system would generate a RAM tile conforming as closely as possible to the specified parameters. This could be done by putting together predefined RAM-bit tiles, decoder tiles, and output drive tiles in an appropriate manner.

The tiles will be designed such that, at each level in the design hierarchy, a common placement and routing grid will be employed. For example, if the top-level is the PPL-level then the PPL grid can be used to build the random logic and perform the routing which glues the other tiles together.

4.6 References

1. R. J. Lipton, et al., "ALI: A Procedural Language to Describe VLSI Layouts," *Proceedings of the 19th Design Automation Conference*, IEEE, Jun. 1982, pp. 467-474.

2. T. M. Carter, "ASSASSIN: An Assembly, Specification and Analysis System for Speed-Independent Control-Unit Design in Integrated Circuits Using PPL," Master's thesis, Department of Computer Science, University of Utah, June 1982.

3. T. M. Carter, "ASSASSIN: A CAD System for Self-Timed Control-Unit Design," Technical Report UTEC-82-020, Department of Computer Science, University of Utah, October 1982, A version of this paper has been accepted for publication in IEEE Transactions on Computer-Aided Design of Circuits and Systems.

4. T. M. Carter and L. A. Hollaar, "Implementation of a Radix-16 Digit-Slice Using a Cellular VLSI Technique," *Proceedings of the 1983 International Conference on Computer Design*, IEEE, Oct. 1983, pp. to be determined.

5. C. Y. F. Chow. "A Variable Precision Processor Module," Ph.D. dissertation, University of Illinois at Urbana-Champaign, July 1980.

6. C. Kingsley, "Earl: An Integrated Circuit Design Language," Tech. report #5021, California Institute of Technology, 1982.

7. B. E. Nelson, "ASYLIM: A Simulation and Placement Checking System for Path-Programmable Logic Integrated Circuits," Master's thesis, University of Utah, October 1982.

8. B. E. Nelson; K. F. Smith; and T. M. Carter, "Cost Effective VLSI Design System," *IEEE International Symposium on Circuits and Systems*, May 1983.

9. Organick, E.I., Keller, R. M., Lindstrom, G., Smith, K.F., Subrahmanyam, P.A., Carter, T., Klass, D., Maloney, M. P., Nelson, B.E., Purushothaman, S., Rajopadhye, S., "Transformation of Ada Programs into Silicon. Third SemiAnnual Technical Report," Tech. report UTEC-83-026, University of Utah, April 1983.

10. J. E. Robertson, *A Theory of Decomposition of Structures for Binary Addition and Subtraction*, This is a document in preparation for publication from the University of Illinois at Urbana-Champaign.

11. D. Gorman, "Trace and Compare," *Proceedings of the 1982 Custom Integrated Circuits Conference*, IEEE, May 1982.

12. K. F. Smith; T. M. Carter; and C. E. Hunt, "Structured Logic Design of Integrated Circuits Using the Stored Logic Array," *IEEE Transactions on Electron Devices*, Vol. ED-29, No. 4, April, 1982, pp. 765-776.

13. K. F. Smith; B. E. Nelson; T. M. Carter; A. B. Hayes, "Computer-Aided Design of Integrated Circuits Using Path-Programmable Logic," *IEEE Electro 83 Professional Program Session Record, New York, New York*, April 1983.

14. K. F. Smith, "Design of Regular Arrays Using CMOS in PPL," *Proceedings of the*

1983 International Conference on Computer Design, IEEE, Oct. 1983, pp. to be determined.

5. Portable Symbol Manipulation Systems Subproject

5.1 Problem Statement and Objectives

We are standing on the threshold of a potential revolution in computing. Personal machines are finally appearing with sufficient power to support sophisticated symbol manipulation systems. Large research programs now maturing in the areas of computer aided instruction, VLSI, expert systems, computer algebra and general knowledge engineering have the potential for dramatic impact when delivered on widely available personal machines. There is great potential for exciting synergy in the combination of several of these systems into a unified environment. Traditionally, LISP and LISP-based systems have been the vehicle used for the rapid prototyping of these and similar applications. New LISP implementations on VAX's and personal machines are attracting the attention of "dyed-in-the-wool" FORTRAN and C programmers.

However, impediments exist to the successful realization of this revolution. The power of symbolic computing comes at the price of significantly more complex programs. A large variety of incompatible LISP dialects have developed on different machines, each with an avid following. This confusing state of affairs has caused many potential LISP users to avoid LISP for "real" applications, citing the the lack of an accepted. transportable and efficient implementation. Unless we can find new ways to reduce the plethora of incompatible alternatives, and manage this complexity in a uniform, coherent manner, we will fail to realize the potential inherent in this new role for computers.

For many years research at Utah has focused on automating the rapid prototyping of efficient and portable symbol manipulation systems, and encouraging their use in non-traditional environments. Our most recent system, PSL, is recognized as an interesting and effective new LISP implementation. PSL has become a significant component of experimental CAD systems for CAGD and VLSI. The current version of PSL runs on most of the machines in the department, providing a uniform language and multi-machine environment to host these applications. The essential ingredients for its success are the writing of a much larger portion of the language in itself than has been done for previous LISP systems, and the development of an optimizing compiler driven by tables describing the target hardware and software environment.

As diverse application areas continue to develop, demands are made for continued improvements and evolution of PSL into a more powerful LISP. This may result in the evolution of PSL into the newly developed Common Lisp [20] dialect. Greater execution efficiency; improved storage management and additional data-typing capabilities are a common need. Therefore, we propose to direct our research toward investigation in the following areas:

* High quality, portable compilation of LISP and SYSLISP (the LISP-like implemention language);

* More efficient handling of "traditional" data types like floating point numbers, matrices and strings; and

* Further improvement of the capabilities of the SYSLISP level, through additional data-typing facilities.

5.1.1 Long Range Objectives

The primary long-term goal of this work is the production of a portable symbol manipulation system capable of supporting powerful engineering applications. Such systems would run on large address space, personal workstations and would permit the combination of graphics, screen editing and browsing paradigms, computer algebra and AI technology. Besides this integration of these various disciplines into one coherent environment, of prime consideration is efficiency. The systems must be of sufficient efficiency to permit fast user interaction. Another goal is to continue to maintain this LISP compatibility across a wide variety of machines.

With a single dialect (or family) available on a range of machines from 68000-based workstations to Cray-1 super computers, a number of multi-machine experiments can be explored. CAD "workbenches" can be built using reactive, user friendly workstations for rapid design, and a back-end "symbol-cruncher" on the larger machines such as a CRAY-1 to do rapid simulation.

Portions of the desired environment for CAD and CAI are now being prototyped on top of the 68000 PSL systems. Work at Utah, HP and Rand has lead to the recent bootstrap of the REDUCE computer algebra system onto the 68000 PSL. The ensuing environment could also include LISP-based graphics packages, EMACS-like screen editor (NMODE, successor to EMODE [8]), an AI representation language (HPRL, successor to FRL [19]) and an object oriented LISP extension (GLISP [17]) all running well on PSL. A number of these sub-systems have begun to interact effectively, and will in the future permit the exploration of exciting applications. Earlier experiments in a less than ideal environment have indicated the importance of the "synergistic" combinations. An initial application involving algebraic computation to be explored at Utah will include incorporating a computer algebra facility into the ALPHA-1 system to do analytic surface manipulations, followed by immediate interactive rendering. This continues the Feng and Riesenfeld [5] experiments into the new discrete B-spline regime. CAGD is a highly mathematically oriented application which requires sophisticated mathematical tools. The REDUCE algebra system [12] which runs on PSL is a perfect match for providing algebraic operations and would be well suited for applications like analytic surface analysis in CAGD. Further exploitation of the NMODE screen editor/browser paradigm will lead to a powerful interactive graphics system, with the potential of a graceful merging of menu driven approaches with linguistic approaches.

5.1.2 State-of-the-Art

5.1.2.1 External to Utah

There are many different varieties of LISP, and the environments they provide and availability on current machines vary greatly. There is very little compatibility between the various versions, prompting the need for a portable approach, and related family of

implementations. Most noteworthy has been the recent attempt to merge the various MACLISP derived dialects (SPICE LISP, NIL, LISP-MACHINE LISP, etc.) into a common dialect, or at least a common subset of these LISP's, called Common LISP [20]. In the coming years, we will see the maturing of a number of independent Common LISP (or Common LISP compatible) implementations; some of these will use common code, others will be created by modifying existing LISP implementations or loading "compatibility" packages. The current CMU SLISP system sources (using a byte code for a micro-coded implementation) are being used by CMU to do a VAX implementation, and by Rutgers to do an extended addressing DEC-20 version. An S-1 implementation of Common LISP [1] uses a different set of sources and compiler technology [2]. The new Scheme-like dialect of LISP under development at Yale [18] is also of great interest.

5.1.2.2 Internal to Utah

Utah has been involved with portable symbol manipulation systems for over a decade. A previous portable "interchange" dialect, Standard LISP [15], has been widely used to transport significant application programs in the area of computer algebra and computer aided instruction. An efficient portable LISP compiler was used with these systems [10] with great success on a number of machines. In early 1981, a new LISP development began. Its two main goals were: to further improve the portability and efficiency of LISP systems; and to permit applications to portably exploit machine oriented features not available in the subset LISP visible via Standard LISP.

Our latest system, PSL (Portable Standard Lisp [11, 21]) provides a single portable dialect of LISP, with a fairly rich environment typical of modern LISP's, yet is directly portable to a wide variety of target machines. PSL has a machine oriented "mode" for Systems programming in LISP (SYSLISP) [11] which permits access to the target machine about as efficiently as in C or PASCAL. As a consequence, PSL has an execution speed quite comparable to that of Franz LISP (written in C) or MACLISP (kernel hand-coded in assembly code). This uniform interface (PSL) to LISP and "Fast Systems LISP" is an advantage for computationally bound applications typical of CAGD and VLSI, which can directly exploit SYSLISP to dramatically speed up applications. Also, with the entire system and application written in PSL, it is quite easy to "flow" between the various levels thus saving implementation and debugging times.

PSL Version 3.1 now runs successfully on the Digital Equipment Corporation's DecSystem-20 (recently including extended addressing) and Vax Series computers (running Berkeley 4.1 Unix), the Hewlett-Packard 9836 computer and the Apollo Domain 68000 based computer. Some limited distributions to the outside community have been made. All four versions are completely compatible, allowing software developed on one system to be run directly on any of the others. This system-wide compatibility is the strongest feature of the PSL system. Previous versions of PSL running under VAX/VMS with Eunice and under TENEX are in use by collaborators, and may be upgraded for distribution. Implementations are underway for CRAY-1 and IBM-370.

5.1.3 Present Objectives

The major objectives for this proposal are to expand the range and efficiency of the technology developed for our current implementations, in preparation for the more sophisticated and demanding applications to come. The work detailed in the following sections is directed at:

* Research into improved compiler efficiency and portability - development of improved techniques for register allocation; study methods to exploit declarations in Lisp systems; investigate ways of improving the handling of numeric code; comparison of various Lisp compilers with careful studies of emitted code and performance; research into improved source-to-source and target-code transformations; development of architectural description languages in the context of compilers.

* Development of a more powerful SYSLISP model that would permit the implementation of untyped data structure access - this could be accomplished by adding a strong typing capability, to interface with the declaration phase in compilation; modules and packages. This improved model would permit the replacement of some C or PASCAL used in some applications by efficient SYSLISP code.

* Extended family of LISP-like systems - the PSL language is a fairly old dialect of LISP. Research will be performed to determine how to extend PSL with features derived from Common LISP; object oriented programming facilities, derived from Flavors, Scheme, GLISP, etc.; interface static typing of SYSLISP with dynamic typing; isolate common-kernel to support PSL, Common LISP and successor systems.

5.2 Method of Approach

5.2.1 Overview

The key to PSL's current popularity as a "lean and mean" LISP is the careful combination of efficiency and portability. The current PSL compiler represents a compromise between size and efficient compilation of LISP and SYSLISP for a portable system. This has permitted us to move PSL to a number of different machines, and bootstrap off an older LISP with reasonable cost. The relatively small size of the compiler permits it to be run resident, incrementally compiling LISP (with SYSLISP extensions) to quite efficient machine code. This permits interactive testing of machine oriented code that will later be added to the PSL kernel, and also has encouraged users to exploit optimizations that result in 5-10 times speed up over "normal" compiled LISP code. Using SYSLISP for this purpose is MUCH easier than trying to write the equivalent code as machine code, C or Pascal procedures to be called from LISP.

During the early PSL development and bootstrap period, most of the work on the compiler was directed at those constructs that were needed to write a LISP in itself portably. Little

attention has been given so far to efficiently compiling things that are important only to applications; thus, for example, integer, string and vector operations are currently well compiled, but floating point operations are poor. It was also an important engineering decision that once acceptable efficiency in an area was achieved (measured by the speed of current LISP systems), we moved on to other more pressing problems. Unsystematic examinations of output code have revealed a number of places where quite significant improvements are still possible in either the basic compilation, the final code-generation, or even decisions related to the efficiency/portability trade-off.

Therefore, the primary research goal of this project will be directed toward developing techniques for improving Lisp compilation strategies. Our secondary goal will be focused upon research into the techniques for adding sophisticated data-typing/declaration mechanisms to Lisp and the effects of adding this type of mechanism on the power and expressiveness of the language. We believe that such a mechanism will permit improved efficiency of the CAGD and VLSI applications which make heavy use of floating point numbers and matrices.

5.2.2 New Compilation Strategies

A major component of the proposed work is to develop newer compilation strategies than that which is used in the current PSL system. These strategies could be incorporated into the current PSL compiler, and along with a better interface to the machine oriented constructs would dramatically improve the performance of applications that are now turning to PSL (robotics, graphics and geometric design, VLSI, some high-performance AI systems). The current PSL compiler was constructed upon an earlier portable LISP compiler, and so uses rather old compiler technology and (quite successful) heuristic optimization techniques.

We will investigate the newer compiler technologies (based on Bliss, PQCC, S-1 LISP and PL.8), and determine what areas are relevant considering that portability and the relative ease of maintenance are important issues. The goal is to develop strategies that would make the compiler even easier to retarget to a wider range of architectures. In particular we want to prepare for the powerful RISC-like machines that seem already to favor the register oriented model used by the PSL system. Despite its current success, we now believe the target machine parameterization used in PSL is not powerful enough for symbol manipulation systems of the future. The system must be capable of managing simultaneous experimentation with alternative compilation strategies; and handle multiple target machines and evolving evaluation models. Symbol manipulation systems will rapidly evolve from a standard LISP dialect, such as PSL, under the influence of new object-oriented programming styles and the desire to exploit available machine power.

The research will involve the following steps:

5.2.2.1 Extensive measurement of PSL performance and code generation

Much of the current PSL development has been guided by a number of static and dynamic measurements (including many measurements performed in previous LISP

implementations that lead to the current PSL. However, most of these measurements were from either earlier Standard Lisp systems or collected from LISP's external to Utah. It is quite important that these measurements accurately reflect the current system as they show possible weakness in design or bad choices for particular implementation strategies. The results of the measurements will also allow us to give guidance to the designers of new hardware, and to understand the compiler/hardware trade-off in greater detail. Therefore, we will spend considerable time first determining what is appropriate to measure, performing the measurement and finally analyzing the data provided.

Most measurements done so far by us, and others, have studied the target (abstract) machine statistics, and were mostly used to improve micro-coded LISP machines [7]. These already reflect the compilation strategy. We need to analyze both the interpreted LISP level (using an instrumented EVAL for the dynamic measurements), as well as the current Intermediate Level (the ALM or CMACRO level in PSL) and target machines. We will measure a number of significant LISP programs, such as the REDUCE algebra system, the PSL compiler, an AI representation language (the new FRL, HPRL is a candidate), the BIGNUM package, the NMODE screen editor and the PSL kernel sources. Another area that we will measure concerns a significant set of graphics modules that were written in C and translated to PSL. Their execution in LISP is quite inefficient (for example, matrix operations on floating point numbers), therefore their study should provide some insight into some inefficiencies in PSL/SYSLISP.

We will inspect the output of the current PSL compiler, and other LISP compilers that seem appropriate for a variety of machines. The goal is to understand what is wrong with the emitted code, and to what degree small changes and additions to the current compiler will have significant effects. (This exercise itself will in fact result in some improvement to the PSL code-generators). We have some initial hypotheses about the need for an improved register allocator [3] and changes in the current PSL abstract LISP machine; we need to be sure how important that is before embarking on a major re-write of the compiler.

5.2.2.2 Introduction of Architectural Description Languages

The PSL compiler currently uses a quite successful, but "ad hoc" interpretive pattern matching method to describe the target machine code generators. Like many other "interpretive" code-generator schemes in use in current portable compilers [9], it requires the programmer retargeting the compiler to express a "macro" for each of the abstract LISP machine forms in terms of one or more of the available target machine instructions. This has made it harder to get maximum efficiency as we begin to use more instructions, since it is quite hard to understand or exploit the interactions of the various instructions. Recent work at Utah [13] and elsewhere [9] indicate that it should be economically feasible to derive the code-generator tables mechanically from a description of the target machine. A different variant of these techniques is being used in the PQCC project to produce high quality, retargetable compilers with some success. This same input can then also be used to derive appropriate code optimizers [4, 14]. We will study these systems and determine their applicability toward Lisp compilation.

The current state of the art of this work has resulted in a variety of incompatible machine

description forms (some derived from ISP, others more LISP-like), and many different methods for actually generating the code generators from the machine tables; some are rather slow and bulky. We will review the current state of these systems, and explore to what degree we can incorporate the benefits of such an approach in a production quality compiler for LISP and other symbol manipulation systems, without losing the current speed and compactness of the PSL/SYSLISP compilation strategy. The benefits are enormous: higher quality code, fewer errors in writing the code generators, and more rapid retargeting. Automated tests (and greater initial confidence) of the code-generators will replace laborious hand coding and testing.

Once we have gained some experience with the architectural description driven code generation, we will explore the possibility of extending this approach into other target machine dependent areas. We may be able to use the architectural description to specify some of the operating system and loader interfaces, and automate the generation of the assembler portion of LAP and FASL (now coded by hand by modifying an "almost" standard version). It is quite possible that we could develop a complete compiler and loader system automatically from a tabular description of the target machine.

5.2.2.3 Appropriate Level of Compiler Source-to-Source Transformations

Recent work on LISP compilation [2] and other systems suggest the practicality of using some "algebraic" simplification of the input program. This is performed by a powerful rule based transformation system which can do many of the important compilation and optimization tasks as source-to-source manipulations. This would make the subsequent code-generation phases do a better job with less effort, and make it easier to share optimizations between systems. An interesting future goal is to explore the use of a more general purpose AI-ish representation language and powerful rule interpreter. We might be able to approach complete Rule based compilation; the interesting question will be the true practicality for a "real" compiler.

It is important to combine information from the data-definition level and an appropriate architectural description level in the compiler. This will permit many new source-to-source transformations during compilation, and perhaps permit rule-based compilation all the way from the source to target code. There are considerable opportunities for extensive "algebraic" simplification of expressions generated during compilation, yet most current compilers have not been able to avail themselves of the (mostly LISP based) general purpose algebra systems, and make do with weaker substitutes. We therefore believe that it is important to be able to experiment with a Lisp system that includes a running algebra package. Thus we will initially exploit a subset of REDUCE running in PSL for this purpose. We will also use this subset for symbolic execution in assessing program costs and complexity analytically, and exploration of algorithm and transformation complexity.

5.2.3 Data-Typing and Declarations

We will investigate the effects of adding data-typing and declarations to Lisp. This will be couched in the current version of SYSLISP which is similar to BCPL except that there

are very few declarations and no variable typing. The lack of a sophisticated type management facility will increasingly hamper the rapid development of improved versions of PSL and successors. Even now we are encountering problems with confusing code, inconsistent software packages and inability to exercise precise control over machine-oriented data-types in as flexible and portable a manner as we desire. The ability to declare variables and data-structures to be WORD, BYTE, S-EXPR, POINTER, etc. will lead to a more secure and efficient system. As far as possible, we desire the declarations to work at the most abstract level, informing the system of an intended type, and leaving to the compiler or preprocessor to do whatever it sees fit for efficiency. The compiler can generally produce better code if variable and expression types are known. Types can often be deduced from the form of expressions and constants, but declarations will be required to resolve some ambiguities, or give added security by confronting deduced types with declared types.

We propose to that such a facility could be implemented as a module independent of the actual compiler, much in the nature of a FLAVORS or Object oriented package. The goal is to transform "typed" LISP, decorated with declarations, data-type and module definitions, using "generic" procedure names, into type-specific LISP that can be compiled more effectively, but can also be interpreted. The correct design will permit the interpretive use of SYSLISP and the free intermixing of SYSLISP and LISP code. In fact, we expect the distinction between SYSLISP and LISP to disappear, merging into a continuous spectrum from untyped interpreted LISP at one end to efficient C-like code at the other. The addition of typing information will provide an efficiency mode to the LISP level that is easily invoked by the appropriate declarations. Thus, the LISP level can be viewed as a highly reactive development environment that will support continuously varying capabilities from generic LISP to very efficient typed LISP. This environment will support very machine oriented applications that can be ultimately "excised" from the LISP environment for standalone execution and export.

There have been many attempts to incorporate these well known benefits of data abstraction, security and efficiency of data-typing/data-description, object-oriented languages into a LISP-like language [17], including two experiments at Utah [16]. Many LISP systems have record declaration packages, user defined type systems, and type-declarations that the compiler can take advantage of to select more machine specific operations. However, in all current systems, there is no checking that declared variables and procedures are in fact used in correctly typed expressions, leading to rather subtle bugs in the name of efficiency. Efforts to add a static-typing facility without losing the benefits of dynamic-typing essential to the interactive environment have met with varying degrees of success. A correct solution will permit a smooth transition from high-level algebraic specifications down to efficient machine implementation. This will correctly handle the detection of legal generic operator/operand combinations and their transformation into more machine specific operators and data-structures.

Our research will begin by investigating the recent ADA-like typing system (with simple type inference and hierarchies) developed at Utah [16]. It runs within the current PSL environment, but has not been fully integrated or exploited to write "real" code. As it evolves, we expect to see some influence from more object-oriented programming styles recently implemented in PSL.

5.3 References

1. Brooks, R. A.; Gabriel, R. P. and Steele, G. L. Jr., "S-1 Common LISP Implementation," *The Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, Carnegie-Mellon University, Pittsburgh, August 1982, pp. 108-113.

2. Brooks, R. A.; Gabriel, R. P.; and Steele, G. L. Jr, "An Optimizing Compiler for Lexically Scoped LISP," *The Proceedings of the 1982 ACM Symposium on Compiler Construction, Boston, MASS*, June 1982, pp. 261-275.

3. Chaitin, G. J., "Register Allocation and Spilling via Graph Coloring," *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, ACM SIGPLAN, January 1982, pp. 98-105.

4. Davidson, J. W. and Fraser, C. W., "Eliminating Redundant Object Code," *Conference Record of Ninth Annual ACM Symposium on Principles of Programming Languages*, ACM, New York, ACM, January 1982, pp. 128-132.

5. Feng, D. Y. and R. F. Riesenfeld, "A Symbolic System for Computer Aided Development of Surface Interpolants," *Software- Practice and Experience*, Vol. 8, 1978, pp. 461-481.

6. Fenichel, R. R., "An On-line System for Algebraic Manipulation," Ph.D. dissertation, Harvard University, 1966.

7. Gabriel, R. P.; and Masinter, L. M., "LISP Evaluation and Timing," *The Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, Carnegie-Mellon University, Pittsburgh, August 1982, pp. 123-142.

8. Galway, W.; and Griss, M. L. "An Editor Interface for a Portable Lisp Programming Environment," Utah Symbolic Computation Group, Opnote 64, University of Utah, Department of Computer Science, January 1982.

9. Ganapathi, M.; Fischer, C. N.; and Hennessy, J. L., "Retargetable Compiler Code Generation," *ACM Computing Surveys*, Vol. 14, No. 4, December, 1982. pp. 348-375.

10. Griss, M. L.; Benson, E.; and Hearn, A. C., "Current Status of a Portable LISP Compiler," *Proceedings of the SIGPLAN 1982 Symposium on Compiler Construction*, ACM SIGPLAN. June 1982, pp. 276-283.

11. Griss, M. L.; E. Benson and G. Q. Maguire Jr, "PSL: A Portable LISP System," *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, ACM, Carnegie-Mellon University, Pittsburgh, Pa., 1982, pp. 88,97, (Also available as Utah Symbolic Group Report UCP-83)

12. Hearn, A. C., "REDUCE 2 Users Manual - Third Edition," Tech. report, The Rand Corporation, Santa Monica, Ca., 1983.

13. Kessler, R. R., "COG: An Architectural-Description-Driven Compiler Generator," Ph.D. dissertation, Department of Computer Science, University of Utah, January 1981.

14. Kessler, R. R., "Peephole Optimization in COG," Utah Symbolic Computation Group, OpNote 76, University of Utah, Department of Computer Science, June 1983.

15. Marti, J. B., et al., "Standard LISP Report," *SIGPLAN Notices*, Vol. 14, No. 10, October, 1979, pp. 48-68.

16. Morrison, D., "Betty: A Type Analysis Program for LISP," Master's thesis, Department of Computer Science, University of Utah, August 1982.

17. Novak, G. S., Jr., "GLISP User's Manual," Heuristic Programming Project Report HPP-82-1, Computer Science Department, Stanford University, November 1982.

18. Rees, J. A. and Adams, N. I., "T: A Dialect of LISP or, LAMBDA: The Ultimate Software Tool," *The Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, Carnegie-Mellon University, Pittsburgh, August 1982, pp. 114-122.

19. Roberts, R. B. and I. P. Goldstein, "The FRL Manual," Memo 409, M.I.T A.I., Laboratory, June 1977.

20. Steele, G. L. Jr, "An Overview Of Common LISP," *The Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, Carnegie-Mellon University, Pittsburgh, August 1982, pp. 98-107.

21. The Utah Symbolic Computation Group, "The Portable Standard LISP Users Manual," Utah Symbolic Computation Group Technical Report TR-10, University of Utah, Department of Computer Science, March 1981, (PSL version 3.1, 7 Feb 1983)

6. Contract Specifics

6.1 Deliverables

At the conclusion of this contract we will have developed a hardware prototype system of the selected geometric processes and demonstrate it. In order to demonstrate this part of a total geometric design system, it will be integrated into the experimental geometric modelling testbed Alpha_1 and applied to a selected model. The results of supporting and related research into computer geometry, geometric modelling, VLSI, and Portable Symbol Manipulation, as well as the integration of these areas, will be reported.

6.2 Coordination Between VLSI and Geometry Research

This project involves the close coordination of research efforts in computer geometry and special purpose VLSI designs. The target algorithms for casting in VLSI will come from the computer geometry milieu, and intensive dialogue and cooperation will have to ensue in order to assure a clear specification and protocol. The following correlation of intermediate results will accord each effort the necessary time for planning strategy and for implementing the tools required to complete the work. As the activity chart indicates, both the VLSI and geometry groups will engage in continuous interaction throughout the contract period.

Year	First Year		Second Year		Third Year	
Quarter	1	2	1	2	1	2
a. Analyze Algorithms (Subdivision and Intersection)	<--->					
b. Hardware/Software Interfaces	<---------->					
c. Select Portion of Algorithm for Hwe Subsys Implement.		<---------->				
d. Emulate Subsys			<---------->			
e. IC Implementation (2 or 3 chips to be Used in Subsystem)			<---------->			
f. Subsystem Eval. (Performance -- Speed/Accuracy)						<--->

6.3 Geometry

6.3.1 Statement of Work

1. Design mathematical spline representations and corresponding data structures to be used internally to the Alpha_1 modeller. Considerable experience has been accumulated with the initial requirements of such a modeller. An evaluation will be made of data structures which are too costly to maintain in our current prototype software hosted on a general purpose computer, but which might be viable in the context of a higher performance custom implementation.

2. Demonstrate the power of the unified spline approach by modelling difficult mechanical pieces in the Alpha_1 testbed. One example part which the modeller will address is a typical molded plastic or structural foam housing for a piece of electronic equipment. These parts are composed of primitive volumetric shapes with sculptured junctions between them, a class of object which is currently beyond the state-of-the-art. Other examples will include

composite objects which are more complex because they have many simple components and also because they include difficult freeform surfaces. Challenging modelling candidates, like a modern truck or a personnel carrier, will be chosen to be of interest to the DoD community. Demonstration will result after creating and implementing the necessary modelling tools. The result will be a three dimensional mathematical representation.

3. Develop and demonstrate an exact arithmetic algorithm appropriate for use in the intersection operation. Analyze candidate algorithms for suitability with regard to specialized hardware implementation.

4. Specify a functional architecture of thetestbed Alpha_1 modeller which shows relationships between candidate algorithms and data components to allow initial determination of an architecture for a VLSI implementation. This analysis should allow initial estimation of local memory and processor/memory bandwidth requirements. Specialized characteristics of the algorithms which are amenable to VLSI implementation techniques will be identified if they exist.

5. Develop a semiautomatic methodology for extracting a finite element model from the proposed spline based master geometry representation which will allow us to interface to a large body of existing sophisticated finite element analysis packages like Adina. Demonstrate the methodology on a selected testcase.

6.3.2 Approximate Milestones

The following are milestones to be demonstrated for the computer geometry and modelling component:

1. Implement an interactive design editor capable of modelling parts with unions and intersections of both simple primitive geometry and sculptured surfaces on a currently available general purpose computer. The objective of this editor is to avail to the user a richer variety of geometric representations, B-spline bounded solid objects, than has been heretofore available. Such an editor would permit the user to define a more faithful geometric model, particularl y when sculptured surfaces are involved. (24 mos)

2. Complete an analysis of the algorithms used by the modeller to determine which areas require improvement in order to make the modeller perform well enough for practical use. There are known complications with both reproducibility of floating point arithmetic operations and their time requirements which limit the practicality of the present approach. If exact and efficient computations were available, then the crucial intersection and rendering algorithms could be implemented and executed in a manner that is more closely related to the governing theory. It currently contains many extraneous adaptations to account for the disparity between the theory and its implementation. (12 mos)

3. Complete the design of a hardware/software environment suitable for testing the performance of the prototype design editor in concert with special function VLSI processors that have been built to enhance performance of the computationally intensive operations like subdivision, intersection of surfaces, or ray casting. (12 mos)

4. Complete a prototype VLSI implementation of special purpose processors capable of a critical geometric function like subdivision, intersection, or exact arithmetic. (30 mos)

5. Complete a graphical simulation showing the process of generating a manufacturing prototyping capability. This requires calculating offset surfaces and parallel generating surface cuts on a family of successive approximations to the final part surface. The simulation will be demonstrated as a graphical animation sequence involving a modelled part and a modelled cutter. (24 mos)

6. Complete an interface from the Alpha_1 representation with that of some standard and appropriate finite element analysis package like Adina. That is, we will have the ability to subject a geometric object defined by a spline boundary to a standard finite element package to see the results of a finite element analysis. (24 mos)

6.4 VLSI

6.4.1 Statement of Work

a. Formalize the structured tiling integrated circuit layout methodology and the structured arithmetic tiling design methodology.

b. Develop static and dynamic PPL cell libraries using the CMOS technology. This will involve circuit design, simulation, layout, and testing. We expect to use the MOSIS 3-micron technology.

c. Design, implement and test selected CMOS PPL circuits for the special purpose geometric processors to be integrated into the Alpha_1 testbed. These circuits will be fabricated through MOSIS and will include circuits which implement portions of the intersection and subdivision algorithms used in Alpha_1.

d. Define, design, implement and test several high-speed full-custom GaAs circuits of interest in geometric modelling. This will be done in conjuction with Rockwell, using their design rules, and fabrication capabilities. Circuit simulation will be done using an elementary model developed at Rockwell.

e. Identify, design, implement (in NMOS and CMOS) and test a group of

parameterized tiles (other than arithmetic tiles) which will be required in the implementation geometric processors. Since VLSI-based arithmetic computations will be a key portion of the project, a specialized arithmetic integrated circuit design capability will be developed, based on the structured arithmetic tiling design methodology. Test structures will be fabricated using the MOSIS facility.

f. Design, implement and test a variable-precision fixed-point (or rational) arithmetic processor for the geometric processors. This will be done using the structured arithmetic tiling capability developed in the previous point. Preliminary discussions have led to a tentative consideration of a design done at the University of Illinois at Urbana-Champaign by Chow, under the direction of Dr. James E. Robertson.

g. Develop algorithms and tools for the automatic generation of parameterized tiles and arithmetic structures. The parameterized tiles which will be considered include RAM, ROM, multiplexors, and data switches. Arithmetic structures will be developed directly from arithmetic set equations as developed by Robertson.

h. Develop a unified set of tools which, in conjunction with the existing set of PPL design tools, will be used for the design, verification and implementation of the circuits described above. These tools will be built on a relational database system and will include a hierarchical structured tiling design editor, a hierarchical, mixed-level simulator, an advanced state-machine generator, an arithmetic module generator, and test sequence generators.

6.4.2 Approximate Milestones

Year	First Year		Second Year		Third Year	
Half	1	2	1	2	1	2
a. Formalize tiling	<--->					
b. Static CMOS PPL	<---------->					
Dynamic CMOS PPL	<---------->					
c. CMOS PPL Circuits		<---------->				
d. GaAs Custom Circuits	<---------->					
e. Arith. Tiling Meth.	<---------->					
f. Arith. Tiling NMOS		<---------->				
Arith. Tiling CMOS			<---------->			
g. Var. Prec. Proc.			<---------->			
h. Identify Parm. Tiles	<---------->					
Design, etc. Parm. T		<---------->				
i. Parm. Algorithms		<---------->				
Parm. Generators		<---------->				
j. Unified Design Sys.	<---------->					
k. VLSI for Geom. Proc.				<---------->		

6.5 Graphical Support Software Environment

6.5.1 Statement of Work

Study and develop a graphical support software environment capable of efficiently supporting VLSI and CAGD applications. This environment and programming language must have the following characteristics:

1. The environment must be portable across many different workstations and be easily transported to the new workstations soon to appear. Therefore, we plan on utilizing architectural description language techniques to transport the environment to each new workstation;

2. A language interpreter for program development to aid in the rapid prototyping of the VLSI and CAGD applications;

3. Highly optimizing compiler for the incremental development of efficient production quality code.

The following outlines the steps necessary to produce the graphical support system as described above:

a. Integrate the architectural description language techniques into the current environment used for VLSI and CAGD to provide enhanced optimization techniques.

b. Expand the architectural description techniques to include code generation that will allow more flexible and efficient production code. This enhanced environment will be released to the CAGD and VLSI groups for testing.

c. Research into the integration of the architectural description techniques into the entire environment. These new techniques will allow the graphical support software environment to be transported to a new target architecture simply by writing a new machine description.

d. Research and development into techniques for the specification of data types and declarations in an environment which includes both interpretive and compiled program development.

e. Investigate the extension of the data type and declaration mechanism to provide efficient floating point and matrices as needed in VLSI and CAGD

f. Release the advanced portable graphical support environment for testing by the VLSI and CAGD projects.

6.5.2 Approximate Milestones

Year	First Year		Second Year		Third Year	
	1	2	1	2	1	2
a. Arch Desc/Opt	<--------->					
b. Arch Desc/Code Gen		<---------->				
c. Arch Desc/Environ				<---------->		
d. Declare/Data Types		<---------->				
e. New Constructs					<---------->	
f. Release System						<--->