AD-A056 861 UTAH UNIV SALT LAKE CITY DEPT OF COMPUTER SCIENCE F/G 12/1
A SYMBOLIC SYSTEM FOR COMPUTER-AIDED DEVELOPMENT OF SURFACE INT--ETC(U)
MAY 78 D Y FENG, R F RIESENFELD N00014-77-C-0157

UNCLASSIFIED NL

1 OF 1
AD A056861

END
DATE FILMED
9-78
DDC

(6) LEVEL II

SC

(6) A SYMBOLIC SYSTEM FOR COMPUTER-AIDED DEVELOPMENT OF SURFACE INTERPOLANTS.

(9) Technical rept.,

by

(10) David Y./Feng

and

Richard F./Riesenfeld

Department of Computer Science

University of Utah

Salt Lake City, Utah 84112

(15) N00014-77-C-0157,
NSF-MCS74-13017

Key Words: Symbolic computation, computer-aided design, computer graphics, Coons patch.

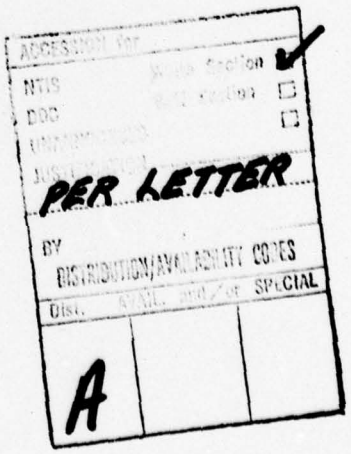
(11) May 78

(12) 57p.

404 949

ABSTRACT

The design and implementation of a symbolic input and computation package and its application to the development of several new surface interpolation schemes are described. Capabilities such as the composition of operators and symbolic differentiation have been incorporated into the system. This allows, in particular, the specification of boolean sum projectors. The new schemes which have been implemented include an interpolant to randomly spaced data and a "shape operator" which has quadratic precision.



Acknowledgement: The authors wish to thank the members of the Computer-Aided Geometric Design Group and Professor M. L. Griss at the University of Utah for their assistance and cooperation during the progress of this research. Also, appreciation is expressed for partial support for this research by the National Science Foundation (MCS-74-13017-A01) and the Office of Naval Research (N000 14-77-C-0157).

INTRODUCTION

Recent work in Computer-Aided Geometric Design can be classified into the following three areas [9]: graphical input/output, man-machine communications and mathematical representation. In any effort toward developing new mathematical schemes of surface representation, a computer, with its associated graphics hardware and software, has become an indispensable tool.

However, despite the current sophistication of computer graphics technology, there is still room for improvement at the computing end of CAGD research. Each time a new surface interpolation scheme is conceived, a significant amount of work in the form of mathematical manipulations, analysis and subsequent software implementation has to be done, before a computer display can be produced. A large amount of this work can be viewed as error-prone, time-consuming overhead when one considers that much of the underlying mathematical operations and software implementations of different polynomial interpolation schemes are basically similar in nature. They mostly deal with evaluating functions, taking partial derivatives, composing linear operators, and the like.

Much of the tedious symbolic algebraic manipulations heretofore done by hand could be implemented as part of a software system for researching CAGD techniques. Such an approach would especially be

useful to CAGD in view of the theorems developed by Barnhill and Gregory [2,3], Gordon [11], Cohen and Riesenfeld [7], which concern boolean sum interpolants and their interpolation and precision properties. These theorems provide a framework in which old or new interpolants can be "combined" to form other interpolants which will possess the desirable properties and eliminate some of the shortcomings of the original interpolants. One of the more powerful results of boolean sum interpolation theory is a composition theorem due to Barnhill and Gregory [2] that states the boolean sum of two projectors

$$P \oplus Q = P + Q - PQ$$

has at least the interpolation properties of P and the function precision of Q. Here we recall that an operator P is linear if

$$P(af + bg) = aP(f) + bP(g)$$

and idempotent if

$$P(P(f)) = P(f)$$

and that a projector is a linear, idempotent operator [11]. The polynomial precision of a projector P is the set of polynomials which P will reproduce.

Poeppelmeier [15] provides an illustration of this theorem by combining Shepard's projector, which interpolates to function and first derivative data at randomly positioned points, but has only linear precision (i.e., it only reproduces planes exactly) and the

Barnhill-Gregory nine-parameter interpolant, which has cubic precision, to obtain an interpolant with the interpolation properties of Shepard's formula, but with cubic precision. Barnhill and Gregory [2,3], Cohen and Riesenfeld [7] have also shown how boolean sum theory can be used to produce interpolants free from compatibility constraints, the requirements for some interpolants that the data given over the boundary of a patch be continuous everywhere on the boundary and that the mixed partials be equal at the patch corners.

One of the earliest (and by now classic), boolean sum interpolants is the Coons Patch [8]. It can be derived as follows: define projectors P_1 and P_2 by

$$P_1F(x,y) = (1-y)F(x,0) + yF(x,1) \tag{1.1}$$

$$P_2F(x,y) = (1-x)F(0,y) + xF(1,y)$$

(see Figure 1). The boolean sum of these projectors is then the bilinearly blended Coons Patch:

$$\begin{aligned}(P_1 \oplus P_2)F(x,y) = {} & (P_1 + P_2 - P_1P_2)F \\ & + (1-y)F(x,0) + yF(x,1) \\ & + (1-x)F(0,y) + xF(1,y) \\ & - (1-y)[(1-x)F(0,0) + xF(1,0)] \\ & - y[(1-x)F(0,1) + xF(1,1)] \end{aligned} \tag{1.2}$$

It can be readily seen that this last expression, especially the part involving the tensor product P_1P_2F, can be obtained formally by a process that is pure symbol manipulation.

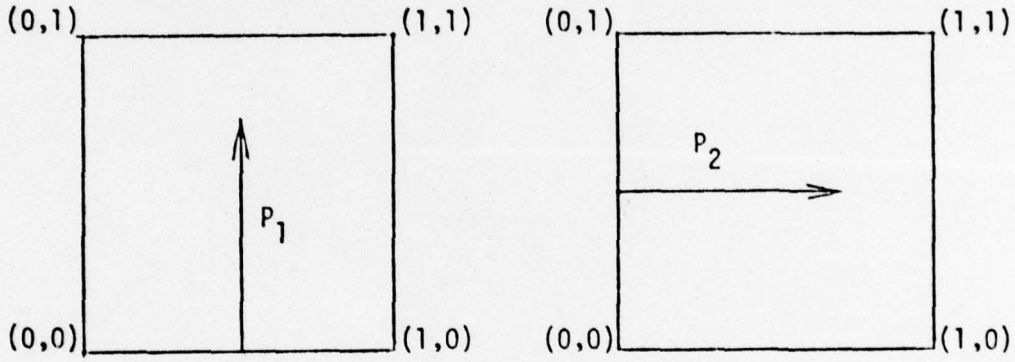


Figure 1. Projector definitions on the unit square.

In order to motivate this work further, we look at the definition of a few more boolean sum interpolants. First, for the bicubic Coons Patch, define P_1 and P_2 by

$$P_1F(x,y) = \phi_0(y)F(x,0) + \psi_0(y)F(x,1) + \phi_1(y)F_y(x,0) + \psi_1(y)F_y(x,1)$$

(1.3)

$$P_2F(x,y) = \phi_0(x)F(0,y) + \psi_0(x)F(1,y) + \phi_1(x)F_x(0,y) + \psi_1(x)F_x(1,y)$$

where

$$\phi_0(t) = (t-1)^2(2t+1) \quad , \quad \phi_1(t) = (t-1)^2t \tag{1.4}$$

$$\psi_0(t) = t^2(-2t+3) \quad , \quad \psi_1(t) = t^2(t-1)$$

are the cardinal basis functions for cubic Hermite interpolation on [0,1]. The boolean sum of P_1 and P_2

$$\begin{aligned}(P_1 \oplus P_2)F(x,y) = {} & \phi_0(y)F(x,0) + \psi_0(y)F(x,1) + \phi_1(y)F_y(x,0) \\ & + \psi_1(y)F_y(x,1) \\ & + \phi_0(x)F(0,y) + \psi_0(x)F(1,y) + \phi_1(x)F_x(0,y) + \psi_1(x)F_x(1,y) \\ & - \phi_0(y)[\phi_0(x)F(0,0) + \psi_0(x)F(1,0) + \phi_1(x)F_x(0,0) + \psi_1(x)F_x(1,0)] \\ & - \psi_0(y)[\phi_0(x)F(0,1) + \psi_0(x)F(1,1) + \phi_1(x)F_x(0,1) + \psi_1(x)F_x(1,1)]\end{aligned} \tag{1.5}$$

$$- \phi_1(y)[\phi_0(x)F_y(0,0) + \psi_0(x)F_y(1,0) + \phi_1(x)F_{xy}(0,0) + \psi_1(x)F_{xy}(1,0)]$$

$$- \psi_1(y)[\phi_0(x)F_y(0,1) + \psi_0(x)F_y(1,1) + \phi_1(x)F_{xy}(0,1) + \psi_x(x)F_{xy}(1,1)]$$

then yields the bicubic Coons Patch.

Now we examine an interpolant over a triangular domain of definition, Nielson's interpolant [4]. See Figure 2. Define

$$P_1F = xF(1-y,y) + yF(x,1-x) \tag{1.6}$$

$$P_2F = (Q_1 \oplus Q_2)F = F(0,y) + F(x,o) - F(0,0)$$

where

$$Q_1F = F(0,y) \qquad \text{and} \qquad Q_2F = F(x,0) \quad .$$

P_1F interpolates to F on E_3 and P_2F interpolates to F on $E_1 \cup E_2$,

$$\begin{aligned}(P_1 \oplus P_2)F = {} & xF(1-y,y) + yF(x,1-x) \\ & + F(0,y) + F(x,0) - F(0,0) \\ & - x[F(0,y) + F(1-y,0) - F(0,0)] \\ & - y[F(0,1-x) + F(x,0) - F(0,0)]\end{aligned} \tag{1.7}$$

interpolates to F all along the boundary of the standard triangle. Barnhill and Gregory have generalized this to interpolate cross boundary derivatives [3].

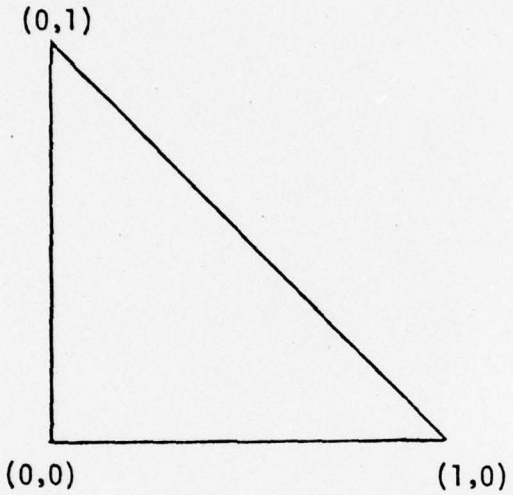


Figure 2. The standard triangle.

All of the above examples demonstrate quite vividly that the computation of boolean sums is a symbol manipulation process. While the above cases can all be done fairly easily by hand, examples abound where the computation of boolean sums is no longer a painless operation. Poeppelmeier's work [15] is a case in point. Recall that he used the Barnhill-Gregory interpolant

$$UF(x,y) = \sum_{i=0}^{1} \phi_i(\frac{y}{1-x})(1-x)^i F_{0,i}(x,0) + \sum_{i=0}^{1} \psi_i(\frac{y}{1-x})(1-x)^i F_{0,i}(x,1-x)$$

$$+ \sum_{i=0}^{1} \phi_i(\frac{x}{1-y})(1-y)^i \left[F_{i,0}(0,y) - \left[\frac{\partial^i P_2 F}{\partial x^i}\right] (0,y)\right] \tag{1.8}$$

$$- \frac{x^2y(x+y-1)}{x+y} \left[(\frac{\partial}{\partial y}(\frac{\partial F}{\partial x}))(0,0) - (\frac{\partial}{\partial x}(\frac{\partial F}{\partial y}))(0,0)\right]$$

where

$$(P_2F)(0,y) = \phi_0(y)F(0,0) + \phi_1(y)F_{0,1}(0,0) + \psi_0(y)F(0,1) \tag{1.9}$$

$$+ \psi_1(y)F_{0,1}(0,1)$$

$$\left[\frac{\partial P_2 F}{\partial x}\right](0,y) = y[\phi_0'(y)F(0,0) + \phi_1'(y)F_{0,1}(0,0) + \psi_0'(y)F(0,1) \tag{1.10}$$

$$+ \psi_1(y)F_{0,1}(0,1)] + \phi_0(y)F_{1,0}(0,0) + \phi_1(y)\left[-F_{0,1}(0,0)\right.$$

$$\left. + \left[\frac{\partial F_{0,1}(x,0)}{\partial x}\right]_{x=0}\right] + \psi_0(y)[F_{1,0}(0,1) - F_{0,1}(0,1)]$$

$$+ \psi_1(y)\left[-F_{0,1}(0,1) + \left[\frac{\partial F_{0,1}(x,1-x)}{\partial x}\right]_{x=0}\right]$$

The $\phi_i(t)$ and $\psi_i(t)$ are the cardinal basis functions for Hermite two point Taylor interpolation on (0,1) given in (1.4). U is itself a boolean sum of 2 projectors. It was then discretized, i.e., changed into a form which will accept discrete data, to obtain

$$UF(x,y) = \phi_0(\tfrac{y}{1-x})[\phi_0(x)F(0,0) + \phi_1(x)F_{1,0}(0,0) + \psi_0(x)F(1,0)$$

$$+ \psi_1(x)F_{1,0}(1,0)]$$

$$+ \phi_1(\tfrac{y}{1-x})(1-x)[(1-x)F_{0,1}(0,0) + xF_{0,1}(1,0)]$$

$$+ \psi_0(\tfrac{y}{1-x})[\phi_0(x)F(0,1) + \phi_1(x)[F_{1,0}(0,1) - F_{0,1}(0,1)]$$

$$+ \psi_0(x)F(1,0)$$

$$+ \psi_1(x)[F_{1,0}(1,0) - F_{0,1}(1,0)]]$$

$$+ \psi_1(\tfrac{y}{1-x})(1-x)[\tfrac{1}{2}((1-x)[F_{1,0}(0,1) + F_{0,1}(0,1)]$$

$$+ x[F_{1,0}(1,0) + F_{0,1}(1,0)]$$

$$- \phi_0'(x)F(0,1) - \phi_1'(x)[F_{1,0}(0,1) - F_{0,1}(0,1)] - \psi_0'(x)F(1,0)$$

$$- \psi_1'(x)[F_{1,0}(1,0) - F_{0,1}(1,0)]]]$$

$$+ \phi_0(\tfrac{x}{1-y})(1-y)\{(1-y)F_{1,0}(0,0) + yF_{1,0}(0,1) - y[\phi_0'(y)F(0,0)$$

$$+ \phi_1'(y)F_{0,1}(0,0)$$

$$+ \psi_0'(y)F(0,1) + \psi_1'(y)F_{0,1}(0,1)] - \phi_0(y)F_{1,0}(0,0)$$

$$- \phi_1(y)[-F_{0,1}(0,0) - F_{0,1}(0,0) + F_{0,1}(1,0)]$$

$$- \psi_0(y)[F_{1,0}(0,1) - F_{0,1}(0,1)]$$

$$- \psi_1(y)[-F_{0,1}(0,1) + \tfrac{1}{2}[-F_{0,1}(0,1) - F_{1,0}(0,1) + F_{0,1}(1,0)$$

$$+ F_{1,0}(1,0) + 6F(0,1)$$

$$+ 4(F_{1,0}(0,1) - F_{0,1}(0,1)) - 6F(1,0) + 2(F_{1,0}(1,0)$$

$$- F_{0,1}(1,0))]]\}$$

$$- \frac{x^2y(x+y-1)}{x+y}[-F_{1,0}(0,0) + F_{1,0}(0,1) + F_{0,1}(0,0) - F_{0,1}(1,0)].$$

At this point it should be obvious that generating formulas like the preceding by hand is a time-consuming, painstaking procedure. Increasing the complexity even further, Poeppelmeier then took the boolean sum of Shepard's projector S with the above interpolant,

$$SF = \sum_{i=0}^{m} A_i(x,y)[F(x_i,y_i) + (x-x_i)F_x(x_i,y_i) + (y-y_i)F_y(x_i,y_i)] \tag{1.11}$$

where

$$A_i(x,y) = \frac{\prod_{\substack{j=0 \\ j\neq i}}^{m} \left((x-x_j)^2 + (y-y_j)^2\right)}{\sum_{k=0}^{m} \prod_{\substack{\ell=0 \\ \ell\neq k}}^{m} ((x-x_\ell)^2 + (y-y_\ell)^2)} \tag{1.12}$$

to obtain a new interpolant.

As new interpolants are constantly being developed which invariably require the use of boolean sums at some stage, for considerations such as compatibility, interpolation and precision, we find an increasing need to incorporate an algebraic manipulation capability into a function and interpolant display system.

Toward the above goals, this research has been concerned with translating some of the mathematical objects and operations required in developing new interpolants into programming constructs. The implementation includes a command language processor, scanner, parser, symbolic computation, and formula evaluation routines. This package has been incorporated as a subsystem to SURFED, an interactive interpolant display and manipulation system implemented by the University of Utah CAGD Group. The entire system runs on an E & S Picture System connected to a PDP 11/45, taking 27K words of memory.

GENERAL FUNCTIONAL SPECIFICATION AND STRUCTURE OF THE SYSTEM

Before we proceed to describe the mechanisms used in the implementation of the system, we should be more precise about the problems we propose to solve. A good way to give this description is via a high-level functional specification of the system. Some capabilities in the following list, particularly those that pertain to interpolant display and manipulation, and domain specification, are part of the original SURFED system, and should be credited to the work of Riesenfeld, Little, Herron and Dube.

System Requirements

1. The system should run in interactive mode, allowing the user to enter interpolants and data from the terminal. The Picture System display and tablet allow him to specify different views of surfaces displayed. SURFED must also allow the user to interactively manipulate the shape of the surfaces being displayed by changing the data being approximated.
2. It should be possible to enter an interpolant in symbolic form and have it displayed as a surface in either parametric or explicit form. The forms of interpolants accepted are rational bivariate polynomials which contain point functional data.

Partial and mixed partial derivatives may be necessary for describing a functional expression.

3. Surfaces are defined for the standard domains below:
   a. Rectangular grid
   b. Triangulated grid
   c. Randomly spaced data points.
4. Function and interpolant definitions are stored by the system and can be referred to by name when used in subsequent definitions. This, along with a composition operator, allows the definition of boolean sum projectors. Thus expressions that may appear repeatedly in subsequent definitions need only be defined once.

The block diagram of Figure 3 further illustrates the structure of the system. The symbol manipulation capabilities are additions to the "Human Interface" and "Interpolant Computations" parts of the original SURFED system.

A more detailed explanation of the way the system functions can only be given by a user's manual of the system, therefore, we devote the rest of this chapter to such a manual.

User's Manual for the System

Input to the system is in the form of a command file. The commands accepted are the following:

DEFINE CONSTANT

DEFINE FUNCTION

DEFINE PROJECTOR

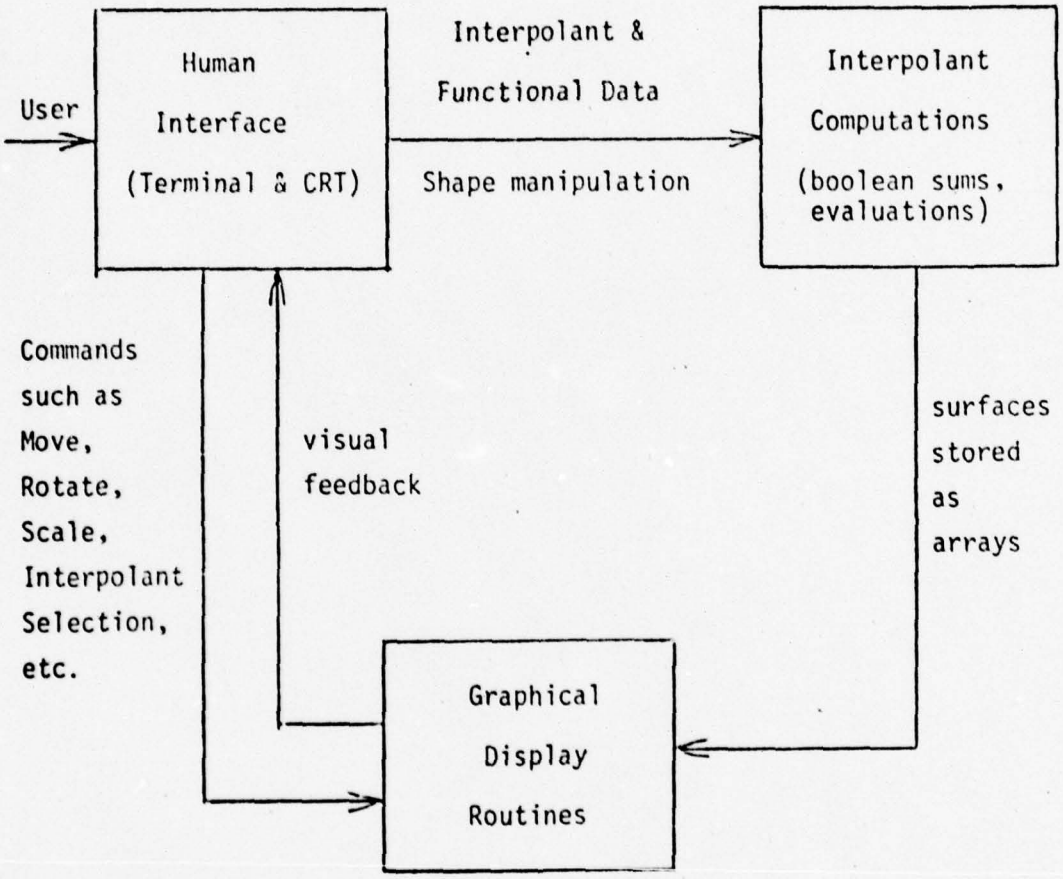


Figure 3. Structure of the system.

DELETE

DISPLAY

EXIT.

In addition, a REDEFINE can be substituted for the above occurrences of DEFINE.

We now give a more detailed description of the commands. The items enclosed in angle brackets: < > are described two sections later, where we give a BNF description of the expression syntax used in the system.

Commands

1. DEFINE CONSTANT

   This command begins a section of definitions, each of which must begin on a new line and end in a semicolon. Each constant definition has the syntax

   <name> = <integer> ;

2. DEFINE FUNCTION

   This begins a section of function definitions, each of which must be preceded by one of the following lines:

   GLOBAL

   LOCAL CIRCULAR

   LOCAL RECTANGULAR <integer>

   where the integer is between 1 and 4. The two LOCAL definition types indicate that the function definition that follows is identically zero outside either a circular or rectangular domain. This is used in defining, for example, haystack functions for

randomly spaced data. For each data point one local circular domain, and up to four local rectangular domains, may be specified. Each function has the form:

<name> = <expr> ;

3. DEFINE PROJECTOR

This command begins a section of projector definitions, each with the form:

<name> = <expr> ;

4. REDEFINE CONSTANT

REDEFINE FUNCTION

REDEFINE PROJECTOR

See Items 1, 2 and 3 for the respective syntax of these commands. Note that a <name> may appear in a REDEFINE definition section only if it has been previously defined.

5. DELETE

This deletes all definitions from the system except for the last one. This command is used to provide more free storage.

6. DISPLAY <name> <integer>

This command causes the function or interpolant named to be displayed on the CRT and passes control from the symbol manipulation subsystem to the rest of SURFED.

The name given must not have been defined to be a constant. The integer may be 1, 3 or 4, depending on whether the <name> is of an interpolant or function that is defined over randomly spaced data, triangular or square patches, respectively.

This command must be followed by a line containing either the word PARAMETRIC or EXPLICIT, which specifies the mode of display.

7. EXIT

This causes a normal termination of the SURFED system.

Command File Format

A. Each command must appear on a separate line. Blank lines are allowed. Only the first three letters of any command word needs to be specified. Thus, DEF FUN is equivalent to DEFINE FUNCTION and LOC REC to LOCAL RECTANGULAR.

B. Each definition section may contain an arbitrary number of definitions.

C. Definition sections may appear in any order and each type of definition section may appear more than once.

D. Constant, function and projector names appearing in a definition must be defined previous to the current definition.

Rules of Expressions Syntax

The syntax of arithmetic expression follows those of FORTRAN, with the following exceptions:

1. Continuation lines need not be denoted.
2. <expr>: = <a FORTRAN expression whose operand syntax has been changed as in item 3 below >|<iter> <expr>

3. Operand Syntax

<operand>: = <integer>|<name>|<var>|<functional>

<integer>: = <digit>|<digit><integer>

<digit>: = 0|1|2|....|9

<var>: = U|V

<name>: = <letter1>|<letter> <digit>

<letter>: = A|B|...|Y|Z

<letter1>: = A|... E|G|...|Q|S|T|W|...|Z

<sub-var>: = <var>(<sub>)|R(<sub>)

<sub>: = <arithmetic expression without parentheses, and whose operands may only be <name> or <integer>>

<functional>: = F(<arg1>,<arg2>)|<diff>F(<arg1>,<arg2>)

<arg1>: = 0|1|U(<sub>)

<arg2>: = 0|1|V(<sub>)

<diff>: = DU|DV|DUDV|DVDU

Examples:

$\frac{\partial}{\partial u} F(0,1)$ is expressed as DUF(0,1).

$\frac{\partial^2}{\partial u \partial v} F(u_i, v_i)$ as: DUDVF(U(I), V(I)).

4. Syntax of iterated product or sum notation:

   <iter>: = <op>[<range>]|<op>[<range><cond>]

   <op>: = @|#

   <range>: = <name> = <const>,<const>

   <const>: = <integer>|<name>

   <cond>: = <name>∿ = <const>

   Thus, the following mathematical expression:

$$\sum_{i=1}^{n} [(U_{i+1} + V_i) / \prod_{\substack{j=1 \\ j \neq 1}}^{10} (U_i - V_i) \frac{\partial}{\partial v} F(U_{j-1}, V_{j-1}))]$$

   would appear in our command file as:

```
@[I=1,N][(U(I+1)+V(I))/#[J=1,10,J∿=I]

[(U(I)-V(I))*DVF(U(J-1),V(J-1))]]
```

5. In addition, the composition operator CM(<projector name>, <projector name>) allows the composition of two projectors to be defined. Note that the second projector in a composition is currently restricted to not contain any iterated product or sum expression.

Software Modules Structure

This section describes the major routines that comprise the system. Figure 4 contains an overlay tree diagram of these software modules.

The Command File Processor reads the command file and interprets

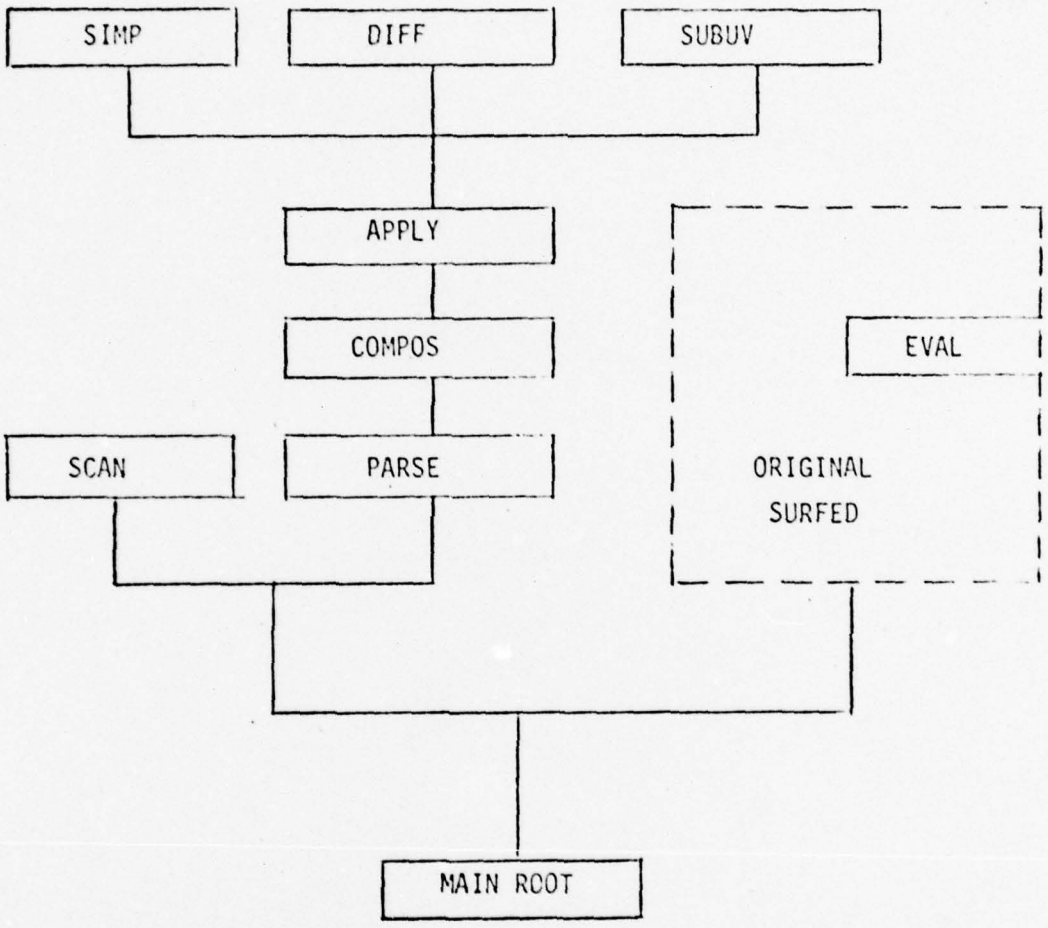


Figure 4. System software structure.

each command. When a definition is specified, it enters the name and type (e.g., PROJECTOR, LOCAL CIRCULAR FUNCTION, etc.) of the definition into the symbol table, then calls SCAN to perform lexical analysis on the input string. The parser parses the scanner output using the operator-precedence parsing technique, modified to take care of subscript expressions and nested summation and product expressions.

If the definition being processed contains a composition operator, PARSE calls COMPOS to perform the composition. It is here that APPLY is invoked to apply a point functional to a previously defined operator expression. When the functional being applied involves the taking of partial derivatives, DIFF is called to perform the symbolic differentiation. SUBUV is then invoked to substitute the arguments of the functional in the second operator of the composition. Some basic simplification is performed, e.g., checking for a 0 or 1 multiplicand or exponent.

When the Command File Processor encounters a DISPLAY command, it passes control to the SURFED System, which eventually calls EVAL to evaluate the postfix definition of the projector specified, and generate values for the surface to be displayed. Since the function values are not bound until evaluation time, a new expression need not be created each time the parameters for a surface are changed during shape manipulation. EVAL was written by simulating recursion in FORTRAN in order to be able to evaluate nested summation and product expressions.

In addition to the above, there are many utility routines which

are called by the above modules and reside in the lower nodes of the overlay tree of Figure 4.

Before we discuss the system in further detail, we consider the question of why we did not use an existing algebraic manipulation system such as REDUCE [13] as a preprocessor to perform the symbolic computation tasks of our system. The answer is that this method does not provide a sufficiently high level of user interaction. "... an algebraic manipulation system achieves its greatest effectiveness if used in a highly interactive man-machine environment. The steps which the user takes in solving a problem very often depends upon the results of preceding calculations and, therefore, a complete "Program" in the conventional sense is often impossible to write a priori." [13]. The system we implemented, which integrates symbolic computation and surface display functions, does satisfy the interaction requirement.

PROGRAMMING AND DATA STRUCTURING TECHNIQUES USED IN THE SYSTEM

We now discuss the major design choices involved in the conception of the system.

FORTRAN was selected for use in writing the system for the compelling reasons of portability, accessibility, efficiency and compatibility with the existing host system SURFED. It is well known to any system designer that the data structures to be used deserve primary consideration in the design process. Since the objects we are dealing with are of the general rational, bivariate polynomial form, how shall we represent them inside the computer?

One method is to use a square array, with indices based on zero. The coefficients of a bivariate polynomial can be stored in the entries of the array with the row number of the entry representing the power of x, and the column number representing the power of y. This method has the advantages of efficient computation--operations such as addition, multiplication, taking partial derivatives and evaluation are fairly straightforward to implement and fast to execute.

But this representation of polynomials has been criticized for wasting storage space in the case of sparse, high degree polynomials, see [6]. Thus existing algebraic manipulation systems, such as REDUCE

[13] and ALTRAN [14] reject this method in favor of a list representation which, along with storage management routines, does keep memory requirements down for operations with sparse, high degree polynomials, particularly if they are in more than two variables.

Although a list representation generally leads to easy-to-use recursively defined algorithms and elegant, top-down program development, there is the ususal time and space overhead associated with using a linked list data structure, and its requisite storage management routines. Time and space considerations often lead to the use of a linear array data structure, usually with the sacrifice of flexibility and elegance.

Moreover, there are several characteristics peculiar to the system that prompted the use of a linear array to represent the mathematical formulae. One such consideration is that the operands in our expressions may be arbitrary in length. They range from simple variables and integers to subscripted variables and mixed partial derivative functionals, e.g., DUDVF(U(I+1),V(I+1)). In order for a list structure to accommodate this, either variable-sized nodes or additional levels of indirection would be required. Pattern matching and searching, required by operations such as operator composition and variable value substitution, can be easily performed on a linear string array.

Considerations such as the above led to the final decision to store the mathematical expressions as a linear string array, in postfix form, similar to the internal form for expressions used in many compilers and interpreters [12]. Each token in the string has a

type and value field. A postfix formulation has the well-known advantage of efficient evaluation. It is also conveniently the output of the standard operator-precedence parsing technique and it presents the tree structure of an expression in an easy to access fashion. Thus, with this data representation, algorithms which either do or do not require knowledge of the tree structure of an expression can process an entire expression with a linear scan.

The most difficult operation that is required to be performed on the mathematical expressions in this system is that of composition of two operators. This often involves the taking of derivatives. In the next section a new symbolic differentiation algorithm is described, which has been developed to work on an expression in linear array postfix form.

A Non-recursive Algorithm for Symbolic Differentiation Using a Linear Array Data Structure

In this algorithm two stacks of pointers, rather than recursion, are used to treat nested subexpressions.

A. Data Structures: A linear array, SYMBOLS, two stacks A and B.

B. Assume the algebraic expression to be differentiated is in syntactically correct postfix form and stored on SYMBOLS beginning at FIRST and ending at LAST.

C. The expression consists of tokens which are either operators or operands. The operators belong to the set R = {+, -, *, /, ↑}. In the description that follows, we restrict R to be {+, *, ↑} for the sake of brevity. Extension of the algorithm to include

- and / should be obvious. The operands in the actual implementation may be integers, variables, subscripted variables or point functionals such as $F_{uv}(u_i,v_i)$, $F_u(1,0)$, etc.

We note here that, since the system currently only deals with rational polynomial forms, the parser only accepts integer exponents, since allowing exponents to be expressions would require a log function for purposes of differentiation.

D. The differentiated result will be another postfix expression stored beginning at SYMBOLS(LAST+1).

E. In describing the algorithm we make the simplifying assumption that each operator or operand occupies one entry in the SYMBOLS array.

The basic stragecy involves scanning the input string from left to right, since it is in postfix form; this corresponds to a post-order traversal of the expression tree.

Step 1. Begin by differentiating the first symbol, which has to be an operand.

Step 2. If the end of input has been reached, stop.

Step 3. The types of the next two symbols are used to identify three possibilities.

The next two symbols may be:

Case 1: operand-operator

We output the derivative of this subtree and set the current input pointer to point to the operator. Go back to Step 2.

Case 2: operand-operand

This means the right subtree associated with the left subtree just differentiated is not a simple operand but a subexpression. We set the current input pointer to the next operand, output two nulls on the output string, push a pointer to the subtree on the input string which has just been differentiated onto stack A and push a pointer to the two nulls on the output string onto stack B. Go back to Step 2.

Case 3: operator-whatever

This means that we have just finished differentiating and outputting a right subtree. Now we can pop stacks A and B to obtain pointers to the corresponding left subtree on both the input and output strings. This is necessary because for the * or / operator, we need to access both the differentiated and undifferentiated forms of both its left and right subtrees. Go back to Step 2.

A more precise description of the algorithm is given below using the programming language ALGOL. In this description we assume, for simplicity, that each operator and operand occupies one entry in the SYMBOLS array.

F. Utility routines:

1. PUSHA, POPA, PUSHB, POPB operate on stacks A and B.
2. Operator and operand are predicate functions which test the type of their arguments.
3. Get and Put gets from and puts to a token at the specified position on SYMBOLS and increments the given pointer.
4. Diff differentiates an operand.

```
begin
      I:=FIRST: FREE:= LAST + 1;
      put (FREE, diff (SYMBOLS(I))):
      while I<LAST do
         begin
           I1:=I+1; I2:=I+2;
             {Check next two symbol types.}
           If operand (SYMBOLS(I1)) then
             If operator (SYMBOLS(I2)) then
               begin
                 {Case 1:  The next two symbols are operand-opera-
                           tor}
           If SYMBOLS(I2)  = '+' then
             begin
               put (FREE, diff (SYMBOLS(I1)));
               put (FREE, '+')
             end
           else if SYMBOLS(I2)  = '*' then
             begin
           {Output pointer to SYMBOLS(I1).  We are not concerned
           with distinguishing between pointer symbols and con-
           stants here, and leave that to lower-level routines.}
             put (FREE, SYMBOLS(I1); put (FREE,'*')'
             put (FREE,diff (SYMBOLS(I1)));
              {output pointers to SYMBOLS(I)}
             put (FREE, I):
```

```
    put (FREE, '*'); put (FREE, '+')
    end
  else begin
  {SYMBOLS(I2) must be exponentiation '↑'}
recall that we only allow constant exponents.
  put (FREE,SYMBOLS(I1));
  put (FREE,'*')'
  {Output pointer to SYMBOLS(I)}
  put (FREE,I);
  put (FREE,SYMBOLS(I1));
  put (FREE,1); put (FREE,'-');
  put (FREE, '↑'); put (FREE,'*')
  end
  I;=I2
  end
  else begin
  {Case 2:  the next two symbols are operand-operand
  pushb(FREE); pusha(I);}
  put (FREE, NULL); put (FREE, NULL);
  I:=I1:
  put (DIFF(SYMBOLS(I)))
  end
  else begin
  {Case 3.  The next one symbol is an operator which
   cannot be '↑'}
  popa (APTR); popb (BPTR);
```

```
        If SYMBOLS (I1)   '+'
         then put (FREE, '+')
         else begin
         {SYMBOLS(I1) must be '*'}
         put (BPTR,I); put (BPTR,I)
         put (FREE,APTR); put (FREE, '*');
         put (FREE,'+')
         end
         I:=I1
         end
    end;
```

Note that in the resulting postfix expression there are pointers to the root of subtrees (subexpressions). These subexpressions are recovered by using a copy operation and noting that in a binary tree, the number of non-terminal nodes (operators) is always one less than the number of terminal nodes (operands). Routines that do basic simplification on the resulting expressions have been implemented to reduce space requirements.

AN APPLICATION

We now describe an application of the system which uses the Barnhill-Gregory composition theorem to combine an interpolant and an approximant to obtain a C^2 interpolant to randomly spaced data with quadratic precision. The symbolic processing capabilities added to SURFED permitted rapid implementation of these schemes, since the time required to translate from mathematical ideas to computer-acceptable form has been reduced to a fraction of the manual algebraic manipulation, programming and debugging time previously required.

Scheme 1. A C^2 Interpolant to Randomly Positioned Data

Assume we are given (x_i, y_i), $i = 1,\ldots,n$, arbitrarily spaced in R^2, and at each (x_i, y_i) we are given function value and tangent information $F(x_i,y_i)$, $F_x(x_i,y_i)$, $F_y(x_i,y_i)$. We define below a C^2 interpolant to this data.

First, pre-process the data by finding for each (x_i,y_i) a circle C_i with its center at (x_i,y_i) and radius R_i, chosen such that C_i does not contain any other data point in its interior. We can, for example, choose R_i to be the distance from (x_i,y_i) to its closest neighbor.

$$R_i = \min_{\substack{1\le j\le n \\ j\neq i}} \{(x_i - x_j)^2 + (y_i - y_j)^2\}$$

The distance function d_i, that is 0 at (x_i,y_i) and 1 on ∂C_i, is defined by

$$d_i(x,y) = \frac{(x - x_i)^2 + (y - y_i)^2}{R_i^2}$$

We can now define a mollifying function H_i at each (x_i,y_i):

$$H_i(x,y) = \begin{cases} q(d_i(x,y)) , & \text{for } (x,y) \in C_i \\ 0 , & \text{elsewhere} \end{cases}$$

where

$$q(t) = (t - 1)^3(3t - 1)$$

is the fourth degree polynomial with the following cardinal properties:

$$q(0) = 1,$$

and

$$q'(0) = q(1) = q'(1) = q''(1) = 0$$

To use these H_i as cardinal functions in the interpolant, we define the truncated Taylor operator:

$$L_i F(x,y) = F(x_i,y_i) + (x-x_i) F_x(x_i,y_i) + (y-y_i) F_y(x_i,y_i)$$

Then the projector P defined by

$$PF(x,y) = \sum_{i=1}^{n} H_i(x,y) L_i F(x,y)$$

yields a C^2 surface that interpolates to function and first derivative.

Scheme 2. A Precision Operator

We now present a projector which can be used as a "shape" operator, i.e., when it is used as the second projector in a boolean sum, it helps to produce a projector which preserves the shape of the given data. It has quadratic precision if we have 9 data points given on a rectangular grid, and is easy to compute. It is based on the idea that a multivariate polynomial is its own Taylor series expansion. That is, a truncated multivariate Taylor series can be thought of as an operator with polynomial precision up to the term of truncation. We specialize this to the case of a bivariate Taylor expansion about the point (a,b) and truncated after the terms of second degree. Thus we define the operator B:

$$\begin{aligned} B\,F(x,y) \;=\; & F(a,b) + (x-a)\,F_x(a,b) + (y-b)\,F_y(a,b) \\ & + (x-a)^2\,\frac{F_{xx}(a,b)}{2} + (y-b)^2\,\frac{F_{yy}(a,b)}{2} \\ & + (x-a)(y-b)\,F_{xy}(a,b) \end{aligned} \tag{4.1}$$

While this operator yields quadratic precision, it requires data that is almost never given, namely, the first and second derivatives. We recall that (Conte and deBoor, p. 196) for apolynomial of degree k, $P_k(x)$,

$$\frac{P_k^{(k)}(x)}{k!} \;=\; P_k[x_0,x_1,\ldots,x_k] \tag{4.2}$$

where the right hand term is the k-th divided difference taken over the points $x_0, x_1, \ldots, x_k$.

We also note that for

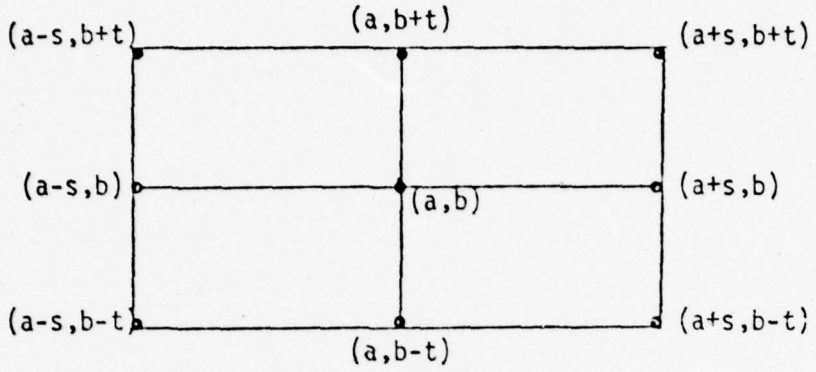
$$f(x) = x^2, \quad f'(a) = \frac{f(a+h) - f(a-h)}{2h} = f[a-h, a+h] \qquad (4.3)$$

Since we are only concerned with what effect our operator A will have on the polynomials 1, x, and x^2, (4.2) and (4.3) imply that we can replace the derivative terms in (4.1) with appropriate divided differences and retain the quadratic precision property.

Thus we obtain a bivariate quadratic precision operator:

$$\begin{aligned} B\,F(x,y) = {} & F(a,b) + (x-a)\,\frac{F(a+s,b) - F(a-s,b)}{2s} \\ & + (y-b)\,\frac{F(a,b+t) - F(a,b-t)}{2t} \\ & + (x-a)^2\,\frac{F(a+s,b) - 2F(a,b) + F(a-s,b)}{2s^2} \\ & + (y-b)^2\,\frac{F(a,b+t) - 2F(a,b) + F(a,b-t)}{2t^2} \\ & + (x-a)(y-b)\,\frac{F(a+s,b+t) - F(a+s,b-t) - F(a-s,b+t) + F(a-s,b-t)}{4st} \end{aligned}$$

Note that B only requires function value information at 9 points on a rectangle



and that B interpolates to the 5 points (a,b), (a,b+t), (a,b-t), (a-s,b), and (a+s,b).

Graphical Studies

Although Scheme I is a smooth interpolant to function value and derivative information at randomly positioned points, it does not have any precision property. This means that while this interpolant looks acceptable for data values close to 0, (see Table I and Figures 5 and 6), it does not have the shape-preservation property for larger magnitude data values. Thus for a second set of data, (Table II) Scheme 1 (Figure 7) looks "bumpy."

To remedy this problem we take the boolean sum of Scheme 1 with the shape operator of Scheme 2 (Figure 8) and obtain the resultant interpolant of Figure 9. The only "bumpiness" in this last figure is induced by a large variation in the given data. Figure 10 demonstrates the quadratic precision property of the interpolant which is a boolean sum of Schemes 1 and 2, by applying it to the function $x^2 + y^2$. Figures 11 through 18 show how Scheme 1 and its boolean sum with Scheme 2 behaves for the functions $(x^2 + y^2)/(1 + x + y)$,

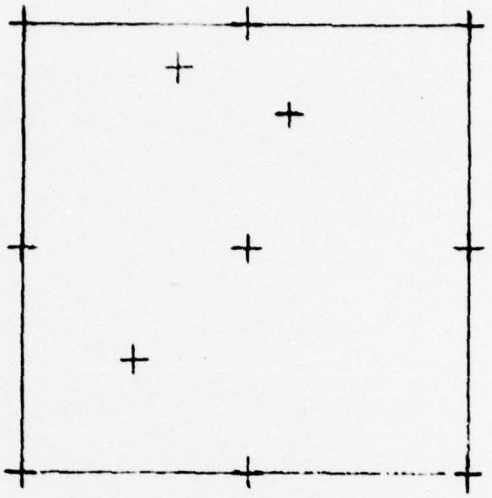


Figure 5. Positions of the data points in the [0,1] x [0,1] domain for the explicit surfaces of Figures 6 through 20. See Tables I and II.

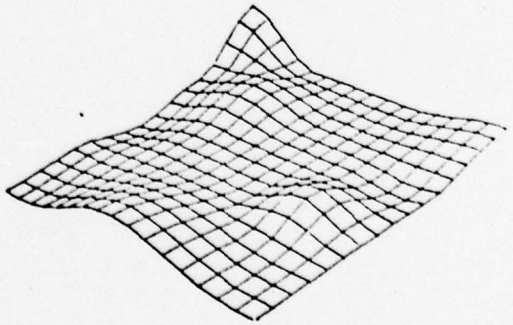


Figure 6. Scheme 1. Explicit. Data: Table I.

Table I. Data for the Explicitly Defined Surfaces in Figure 6.

$F_x(x_i,y_i) = F_y(x_i,y_i) = 0$ for all i.

i	x_i	y_i	$F(x_i,y_i)$
1	0.0	0.0	0.15
2	0.0	0.5	0.0
3	0.0	1.0	0.05
4	0.5	0.0	- 0.05
5	0.5	0.5	0.0
6	0.5	1.0	0.0
7	1.0	0.0	- 0.033
8	1.0	0.5	0.08
9	1.0	1.0	0.0
10	0.25	0.25	0.055
11	0.6	0.8	0.065
12	0.35	0.9	- 0.02

Table II. Data for the Explicitly Defined Surfaces in Figures 7 Through 11.

$F_x(x_i,y_i) = F_y(x_i,y_i) = 0$ for all i.

i	x_i	y_i	$F(x_i,y_i)$
1	0.0	0.0	0.15
2	0.0	0.5	0.0
3	0.0	1.0	0.05
4	0.5	0.0	- 0.05
5	0.5	0.5	0.0
6	0.5	1.0	0.0
7	1.0	0.0	- 0.033
8	1.0	0.5	0.08
9	1.0	1.0	0.0
10	0.25	0.25	0.055
11	0.6	0.8	0.065
12	0.35	0.9	- 0.02

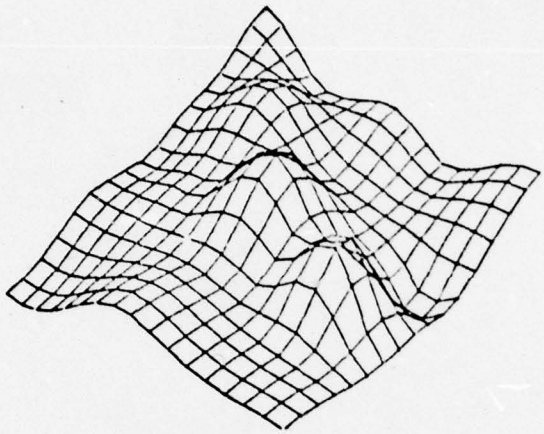


Figure 7. Scheme 1. Explicit. Data: Table II.

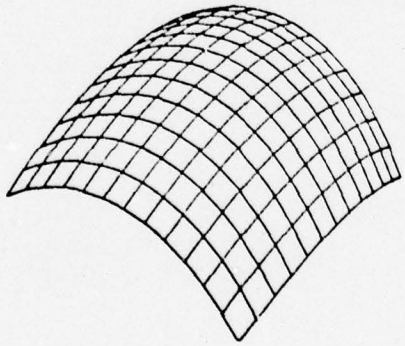


Figure 8. Scheme 2. Explicit. Data: Table II.

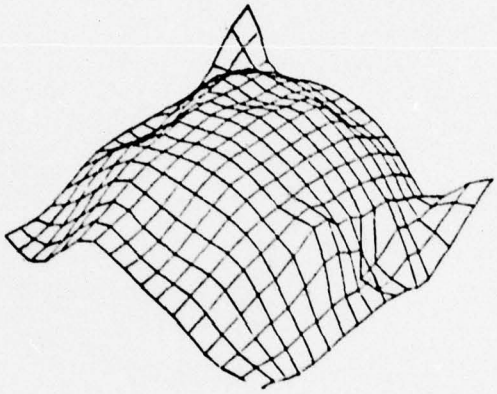


Figure 9. Scheme 1 boolean summed with Scheme 2. Explicit. Data: Table II.

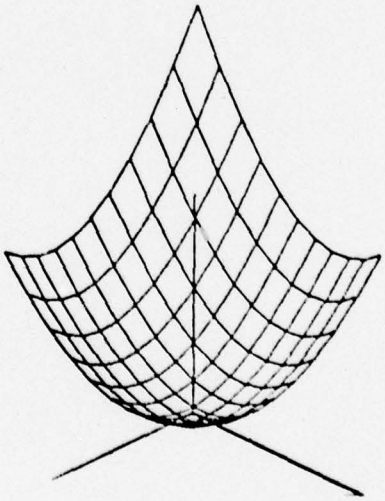


Figure 10. Scheme 1 boolean summed with Scheme 2. Explicit. Data: $x^2 + y^2$.

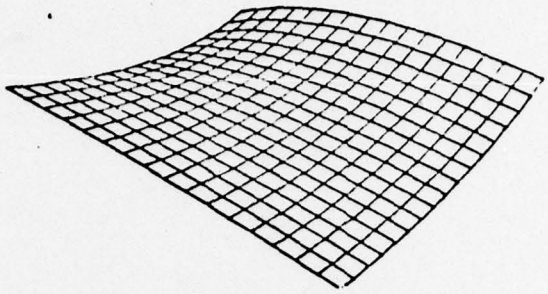


Figure 11. Test function $(x^2 + y)/(1 + x + y)$. Explicit.

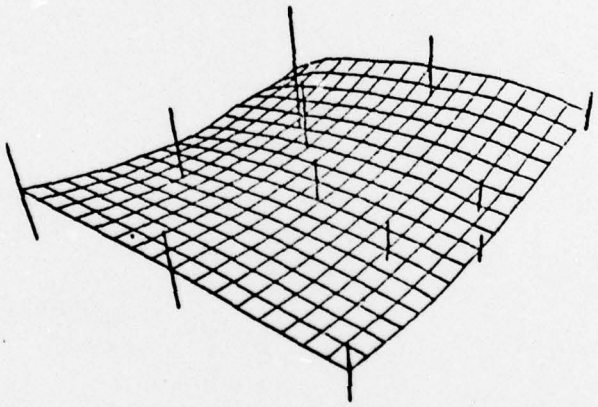


Figure 12. Scheme 1 boolean summed with Scheme 2. Explicit. Data: $(x^2 + y)/(1 + x + y)$. The midpoint of each stick indicates interpolation at that data point.

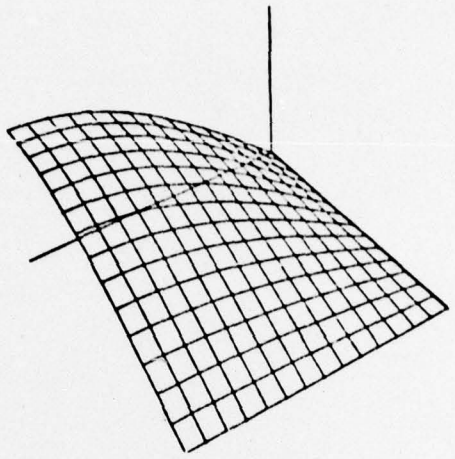


Figure 13. Test function 1/2 sin $x\pi/2$ cos $y\pi/2$. Explicit.

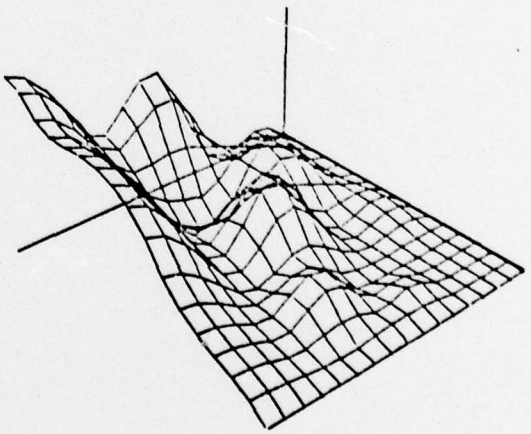


Figure 14. Scheme 1. Explicit. Data: 1/2 sin $x\pi/2$ cos $y\pi/2$.

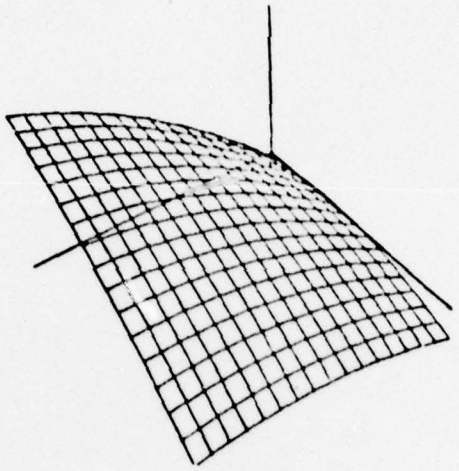


Figure 15. Scheme 2. Explicit. Data: 1/2 sin $x\pi/2$ cos $y\pi/2$.

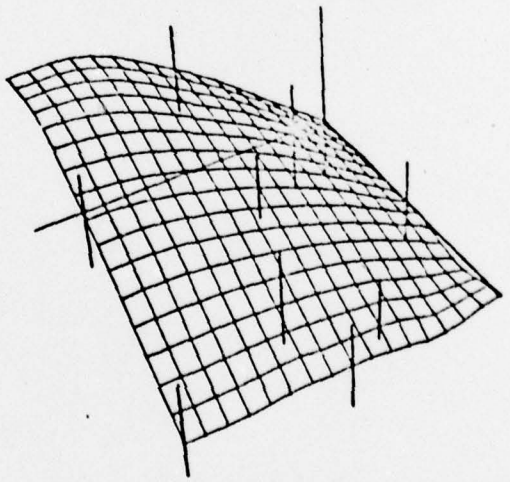


Figure 16. Scheme 1 boolean summed with Scheme 2. Explicit. Data: 1/2 sin $x\pi/2$ cos $y\pi/2$. The midpoint of each stick indicates interpolation at that data point.

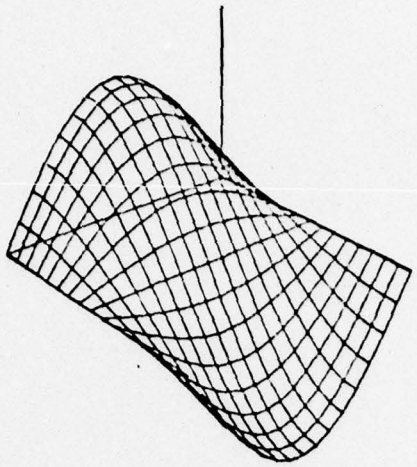


Figure 17. Test function 1/2 sin xπ cos yπ.

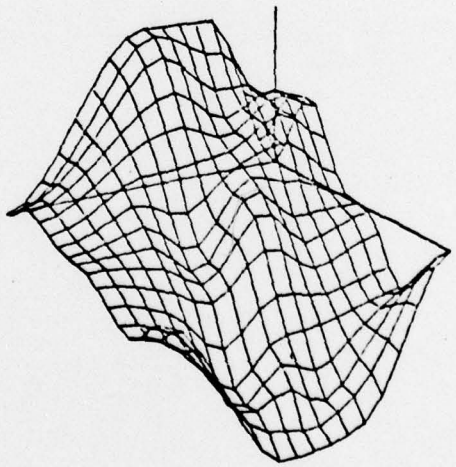


Figure 18. Scheme 1 boolean summed with Scheme 2. Explicit. Data: 1/2 sin xπ cos yπ.

1/2 sin $x\pi/2$ cos $y\pi/2$, and 1/2 sin $x\pi$ cos $y\pi$, over the area [0,1] x [0,1]. Note that all of the above figures are of interpolants defined explicitly over the domain [0,1] x [0,1].

The data sets for the figures mentioned above all contain nine points that are regularly positioned in the domain space. This is a constraint in using Scheme 2 as a shape operator. We could remedy this problem by doing parametric interpolation to the data in the (x,y,z)-space and specify nine of our points in the domain (u,v)-space to be regularly spaced. Figure 19 demonstrates the feasibility of this approach for the data set given in Table III.

Conclusion

We have seen that, with the addition of some basic symbolic computation capabilities to SURFED, the correct implementation of a large class of interpolation and approximation schemes now becomes a simple and routine matter. Tedious, error-prone and mechanical tasks, such as translating from mathematical formulation to program, composition of operators and formal differentiation are no longer the burden of the researcher.

For the schemes decribed here, the actual symbol manipulation time is on the order of a few seconds. The time-consuming operation has been generating values for a surface to be displayed. This involves calling EVAL hundreds of times for a random data interpolant. This, perhaps, suggests the need for more sophisticated simplification routines. Of more certain benefit would be to speed up the evaluation routine, maybe recoding it in a lower-level language.

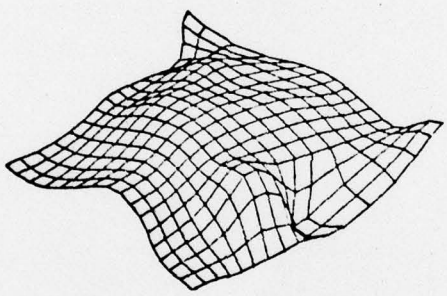


Figure 19. Scheme 1 boolean summed with Scheme 2. Parametric. Data: Table III.

Table III. Data for the Parametrically Defined Surfaces in Figures 21 and 22.

Note the "true" randomness in the x,y and z values.

i	u_i	v_i	$x(u_i,v_i)$	$y(u_i,v_i)$	$z(u_i,v_i)$	Partial Derivatives
1	0.0	0.0	0.1	- 0.1	0.15	For all i:
2	0.0	0.5	- 0.2	0.57	0.097	$x_u(u_i,v_i) = 1.0$
3	0.0	1.0	- 0.27	1.0	0.20	$y_u(u_i,v_i) = 0.0$
4	0.5	0.0	0.45	- 0.1	- 0.1	$z_u(u_i,v_i) = 0.0$
5	0.5	0.5	0.52	0.55	0.25	$x_v(u_i,v_i) = 0.0$
6	0.5	1.0	0.47	1.0	0.0	$y_v(u_i,v_i) = 1.0$
7	1.0	0.0	1.0	- 0.12	- 0.233	$z_v(u_i,v_i) = 0.0$
8	1.0	0.5	0.92	0.5	0.08	
9	1.0	1.0	1.0	1.0	0.1	
10	0.25	0.25	0.25	0.25	0.155	
11	0.6	0.8	0.6	0.8	0.265	
12	0.35	0.9	0.35	0.9	- 0.12	

This is a one-time effort which would improve the system performance for the implementation of all future schemes. This certainly has an advantage over the previous method of FORTRAN coding, manually optimizing and interfacing with SURFED for each new scheme being developed.

The symbolic computation part of SURFED currently occupies about 27K of 16-bit words on the PDP-11/45. Better memory management and use of secondary storage would probably also improve overall performance of the system.

Other useful extensions of the system involves the specification of expressions of more general forms. Foremost among these is perhaps allowing the definition of transfinite interpolants, including the identity operator. This may imply the need for automatic discretization of transfinite schemes, once again done symbolically. The expression types used may also be extended to include as part of their definition functions for which symbolic differentiation can be performed and which are common FORTRAN library functions, e.g., exp, log and the trigonometric functions. Projectors containing nested summation and product expressions can perhaps be specified as the second operator in a composition. This implies being able to symbolically differentiate such expressions.

The features implemented in this system enable the research mathematician to study graphically mathematical surfaces without becoming involved in the distracting details of software development, interfacing and implementation. This kind of application of symbolic computation to numerical analysis should benefit future research in the area of computer-aided geometric design, by allowing the user

of this package the facility to explore and experiment in a far-ranging and unemcumbered environment of interactive symbolic computation.

This research, as it brings together symbolic processing and curved surface definition, emphasizes the importance of the non-numerical tasks involved in computer-aided geometric design. Consequently, it may provide further impetus for researchers in this area to adopt programming languages like PASCAL, which are considerably better suited for these kinds of mixed computational problems.

COMMAND FILE FOR SCHEMES 1 AND 2 AND
THEIR BOOLEAN SUM

```
DEFINE FUNCTION
   GLOBAL
D1=((U-U(I))**2+(V-V(I))**2)/R(I)**2;
   LOCAL CIRCULAR
H1=(1-D1)**3*(1-3*D1);
   ;

DEFINE PROJECTOR                SCHEME 1
P1=@[I=1,12][H1*(F(UI),V(I))+(U-U(I))*DUF(U(I),V(I))
        +(V-V(I))*DVF(U(I),V(I)))];
        ;

DEFINE FUNCTION          SCHEME 2
   GLOBAL
H1=F(U5),V(I))+(U-1/2)*(F(U(8),V(8))-F(U(2),V(2)))
   +(V-1/2)*(F(U(6),V(6))-F(U(4),V(4)))+(U-1/2)**2
   *(F(U(8),V(8))-2*F(U(5),V(5))+F(U(2),V(2)))*2;
   GLOBAL
H2=(V-1/2)**2*(F(U(6),V(6))-2*F(U(5),V(5))+F(U(4),V(4)))
   *2+(U-1/2)*(V-1/2)*(F(U(9),V(9))-F(U(7),V(7))
   -F(U(3),V(3))+F(U(1),V(1)));
    ;

DEFINE PROJECTOR
T1=H1+H2;
A3=P1+T1-CM(P1,T1);     BOOLEAN SUM OF SCHEMES 1 AND 2
        ;

DISPLAY A3  1
 EXPLICIT

EXIT
*
```

BIBLIOGRAPHY

1. Barnhill, R. E. and Gregory, J. A., "Compatible Smooth Interpolation in Triangles," in Journal of Approximation Theory, 15, 3, 1975, 214-225.

2. Barnhill, R. E. and Gregory, J. A., "Smooth Polynomial Interpolation to Boundary Data on Triangles," Mathematics of Computation, 29, 1975, 726-735.

3. Barnhill, R. E. and Riesenfeld, R. F., editors, Computer-Aided Geometric Design, Academic Press, New York, 1974.

4. Bobrow, D. G., editor, Symbol Manipulation Languages and Techniques, North-Holland Publishing Company, Amsterdam, 1968.

5. Brillinger, P. C. and Cohen, D. J., Introduction to Data Structures and Non-Numeric Computations, Prentice-Hall, Englewood Cliffs, 1972.

6. Cohen, E. and Riesenfeld, R. F., "An Incompatibility Projector Based on an Interpolant of Gregory," University of Utah, Computer Science Department. (Submitted for publication.)

7. Coons, S. A., "Surfaces for Computer-Aided Design of Space Forms," M.I.T., MAC-TR-41, 1967.

8. Dube, R. P., Herron, G. J., Little, F. F. and Riesenfeld, R. F., "SURFED--An Interactive Editor for Free-Form Surfaces," (submitted for publication), 1977.

9. Feng, D. Y., A Symbolic System for Computer-Aided Development of Surface Interpolants, Master's Thesis, University of Utah, 1977.

10. Forrest, A. R., "Computational Geometry-Achievements and Problems," in Computer-Aided Geometric Design, Barnhill, R. E. and Riesenfeld, R. F., editors, Academic Press, New York, 1974.

11. Geary, S., A Computer Graphics System to Aid the Study of Interpolants to Curved Surfaces, M.S. Thesis, University of Utah, Salt Lake City, Utah, 1973.

12. Gordon, W. J., "Distributive Lattices and the Approximation of Multi-variate Functions," in Proceedings of the Symposium on Approximation with Special Emphasis on Splines, Schoenberg, I. J., editor, University of Wisconsin Press, 1969.

13. Gries, D., Compiler Construction for Digital Computers, Wiley, New York, 1971.

14. Hearn, A. C., "REDUCE: A User-Oriented Interactive System for Algebraic Simplification," in Interactive Systems for Experimental Applied Mathematics, Klerer, M. and Reinfelds, J., editors, Academic Press, New York, 1968.

15. McIlroy, M. D. and Brown, W. S., "The ALTRAN Language for Symbolic Algebra on a Digital Computer," Bell Telephone Laboratories, Murray Hill, New Jersey, 1966.

16. Poeppelmeier, C. C., A Boolean Sum Interpolant to Randomly Spaced Data, Master's Thesis, University of Utah, 1975.

17. Rogers, D. F. and Adams, J. A., Mathematical Elements for Computer Graphics, McGraw-Hill, New York, 1976.

MIL-STD-847A
31 January 1973

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A SYMBOLIC SYSTEM FOR COMPUTER-AIDED DEVELOPMENT OF SURFACE INTERPOLANTS	5. TYPE OF REPORT & PERIOD COVERED Technical 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) DAVID Y. FENG RICHARD F. RIESENFELD	8. CONTRACT OR GRANT NUMBER(s) N0001477C0157
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Utah, Computer Sci. Dept. Salt Lake City, Utah 84112	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, Dept. of Navy Arlington, Virginia 22217 (Denicoff)	12. REPORT DATE May 1978 13. NUMBER OF PAGES 52
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Resident	15. SECURITY CLASS. (of this report) U 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Unlimited DISTRIBUTION STATEMENT A Approved for public release; Distribution Unlimited	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)	
18. SUPPLEMENTARY NOTES	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Symbolic computation, computer-aided design, computer graphics, Coons patch	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The design and implementation of a symbolic input and computation package and its application to the development of several new surface interpolation schemes are described. Capabilities such as the composition of operators and symoblic differentiation have been incorporated into the system. This allows, in particular, the specification of boolean sum projectors. The new schemes which → next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

have been implemented include an interpolant to randomly spaced data and a "shape operator" which has quadratic precision.