MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 01591 3712
LIBRARY
Michigan State
University

This is to certify that the

thesis entitled

A VLSI Chip Architecture For The B-spline Surface Evaluation

presented by

Chia-Yiu Maa

has been accepted towards fulfillment
of the requirements for

Master degree in Science

Michael Shantz
Major professor

Date 8-28-87

O-7639

MSU is an Affirmative Action/Equal Opportunity Institution

RETURNING MATERIALS:
Place in book drop to remove this checkout from your record. FINES will be charged if book is returned after the date stamped below.

A VLSI CHIP ARCHITECTURE FOR THE B-SPLINE SURFACE EVALUATION

By

Chia-Yiu Maa

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering

1987

ABSTRACT

A VLSI CHIP ARCHITECTURE FOR THE B-SPLINE SURFACE EVALUATION

By

Chia-Yiu Maa

This thesis describes the architecture design of the SE (Surface Evaluation) chip, a VLSI chip dedicated to the B-spline surface evaluation, which can be used to ease the Computer-Aided Geometric Design (CAGD) of B-spline surfaces. The reconfigurable structure of the SE chip can perform on-chip blending function formulation and high-speed vector-matrix multiplication at less hardware cost. Pipelining and parallel processing techniques are applied to the inter- and intra-functional unit designs to speed up the calculation of the surface evaluation algorithm. The symbolic example and the description of the SE chip written in RTL (Register Transfer Language) verify the correctness of the design. Area estimation confirms the feasibility of a semi-custom implementation approach with current technology and speed estimation shows at least an order of magnitude improvement over existing graphics hardware in performing surface evaluation. The SE chip architecture can also be used to execute the coordinate transformation, tangent and normal vector derivation, and shading.

ACKNOWLEDGMENTS

I wish to express my greatest appreciation to my parents for getting me started right and continued support and encouragement throughout my studies, and my wife Mei-Ling, without her patience, understanding and love, nothing would have become reality. I express my deep gratitude to my advisors, Dr. M. Shanblatt and Dr. E. Goodman, for their guidance and encouragement throughout this research. Special thank is also due to Jane Hawkins for providing the mathematical preliminary of B-spline and helpful discussions.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Geometric modelling is a broad field in computer graphics which uses computational geometry to develop representation techniques for three-dimensional objects. It can be divided into two sub-fields: 1) surface modelling, which represents three-dimensional objects using various types of closed surfaces (quadratics, polygonal meshes, parametric bicubic patches, etc.), and 2) solid modelling, which aims at developing object representations which are complete, unambiguous, realizable, accurate, efficient, productive, flexible and minimal. Surface modelling is especially useful in the aerospace and automotive industries, while solid modelling is gradually becoming useful for a variety of mechanical CAD/CAM applications. In this thesis, we shall deal only with the topic of surface modelling, but insofar as most solid modelling systems utilize a curve -- and surface -- based boundary representation (B-rep) for at least graphics calculations, the work described here is quite directly applicable to solid modelling system development.

The process used to generate the realistic computer images from a geometric data base is called image synthesis. It generally involves several steps:

1) transformation of the data base from world coordinates to the

appropriate viewing coordinates (see Figure 1.1);

2) elimination of hidden surfaces by way of depth sorting (sort the polygons by their distance from the viewpoint and to place them into the refresh buffer in order of decreasing distance), z-buffering (a buffer is used to record the smallest z value encountered for each (x,y) as the search porgresses), scan-line processing (by using scan-line coherence and edge coherence to determine the projecting result of all polygons onto the xy-plane), or area subdivision (recursively subdivide the area of the projection plane image until it is easy to decide which polygon or polygons are visible in the area);

3) shading the visible surfaces by taking the light sources, the surface characteristics, and the relative positions and orientations of the surfaces and sources into consideration;

4) taking account of shadows, transparent and translucent materials, surface texture, surface detail, and aliasing (some

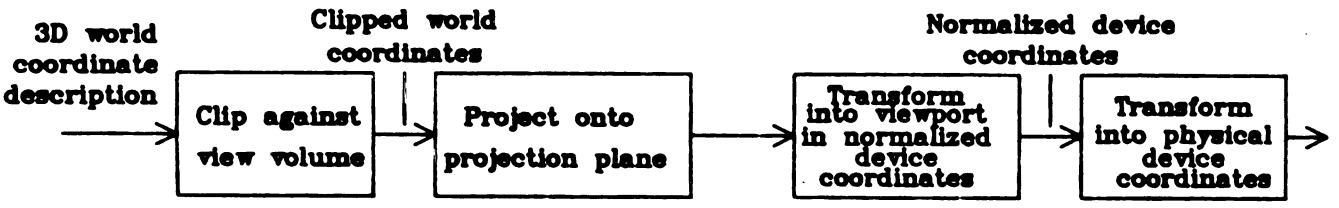


Figure 1.1. Viewing of 3D objects.

of these can be handled by ray tracing).

The formulation of a realistic three-dimensional image does not necessarily follow the order of steps stated above. For example, a chip set designed by Hewlett-Packard Co. has a graphics pipeline as shown in Figure 1.2 [2].

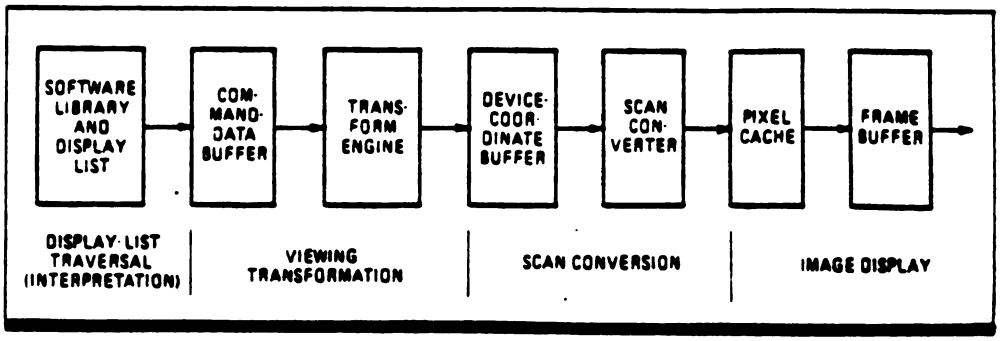


Figure 1.2. An enhanced graphics pipeline [2].

1.1 Problem Statement

Researchers at the A. H. Case Center for Computer-Aided Design, Michigan State University, have developed a surface assessment package called MSU COLORSCOPE to produce accurate shaded images by calculating and shading surfaces evaluated at the pixel level. Recently, several algorithms developed by Pickelmann [1] allowed for more efficient evaluation of the entire range of rational/non-rational curves and surfaces. Though these algorithms speed up the response time of the interactive CAD/CAM system which uses them, the time for generating a

shaded image is still too long (typically more than few minutes) and thus far away from the goal of high quality "real-time" realistic image generating for the interactive CAD graphics system.

It has been shown that the time required for three-dimensional surface evaluation and transformation demands a significant percentage of the total time needed to realize a new shaded image. If we can download the surface evaluation and transformation to high speed hardware, the overall computation time will be reduced dramatically.

The surface evaluation routines shown in [1] involve numerous matrix multiplications and thus are candidates good for implementing on regular structured hardware. The very best candidate for the hardware implementation is VLSI (Very Large Scale Integration), which has shown great potential for improving system performance by implementing concurrent numerical algorithms directly in hardware. Furthermore, advances in CAD (Computer-Aided Design) and CAE (Computer-Aided Engineering) systems for VLSI have enabled such implementation to be carried out in a custom design with relatively low costs in both time and money.

1.2 Approach

The purpose of this research is to investigate and specify the major hardware components of a custom-designed VLSI chip architecture for the surface evaluation algorithm given in [1].

The surface evaluation chip is to work as a coprocessor to the host, a graphics processor taking care of all the graphics processing

except for surface evaluation. It downloads necessary information from the host and evaluates either a point or a patch (by recusively evaluating the point on the patch). Fixed-point calculations are assumed throughout the design. This implies that any necessary scaling is performed in the host system.

The overall design follows the top-down approach. The system criteria will be stated first, followed by the definition of each building block. When specifying the system criteria, the I/O problem, one of the main considerations in dealing with matrix operations, must be solved in order to guarantee better performance. A proper VLSI array structure suited for matrix multiplication is also to be employed in the design.

In order to achieve the highest system throughout, the pipeline concept is extensively applied to the inter- and intra-functional units. The timing scheme for supporting pipelining will also be introduced. The storage scheme of the surface evaluation chip will be addressed particularly because of the special pattern of memory access used. A reconfigurable structure with higher resource utilization and less number of functional components will then be given.

In order to simulate as well as verify the design, the chip architecture will be expressed in the form of a program written in RTL (Register Transfer Language). By comparing the simulation results with the software surface evaluation algorithm, mistakes will be corrected and modifications will be made to improve the system performance.

1.3 A Overview of the Thesis

Chapter Two introduces basic VLSI technology and design methodologies. Chapter Three presents the mathematical preliminaries required for the surface evaluation algorithm. A literature review is given in Chapter Four. Chapter Five gives the surface evaluation algorithm and the idea of shading. The development of the chip architecture is covered in Chapter Six. The simulation results and conclusions are presented in Chapter Seven.

CHAPTER II

VLSI SYSTEM DESIGN

The goal of this thesis is to develop a dedicated VLSI chip for surface evaluation. This chapter introduces the characteristics as well as trends of VLSI technology. The discussion will emphasize semicustom and custom VLSI, also called application-specific integrated circuit (ASIC) design. VLSI design methodologies such the silicon compiler and AI approaches will also be covered.

2.1 Characteristics and Trends in VLSI Design

Recent advances in VLSI technology -- in system and circuit design environments, process sophistication, manufacturing cost effectiveness, reliability, and packaging -- have spurred the identification of new system applications which are now feasible with more advanced technology. Offering excellent noise immunity, wide operating margins with respect to power supply voltage and temperature, lower drive currents, decreased power consumption, ease of design for VLSI circuits, and scalability to submicrometer dimensions with improved performance and reliability relative to alternative technologies, CMOS is the most suitable technology for the majority of these emerging applications. CMOS designs present a disadvantage in that the die size required is

larger than that required by NMOS for the same circuit. New technologies, however, such as domino circuits and clocked circuits, are reducing the die size difference between CMOS and NMOS.

In the ultra-high-speed region, ECL and CML technologies are used. Their performance superiority is irresistible, but other costs are higher. In a much higher speed region, new semiconductor devices are being explored. Gallium arsenide devices are expected to be applied to real-world systems in the near future and are one of the most promising technologies. Figure 2.1 shows the distribution of operation speed and scale of integration for gate arrays achieved by various technologies [3].

Two CMOS technologies have been prominently used: the silicon-gate

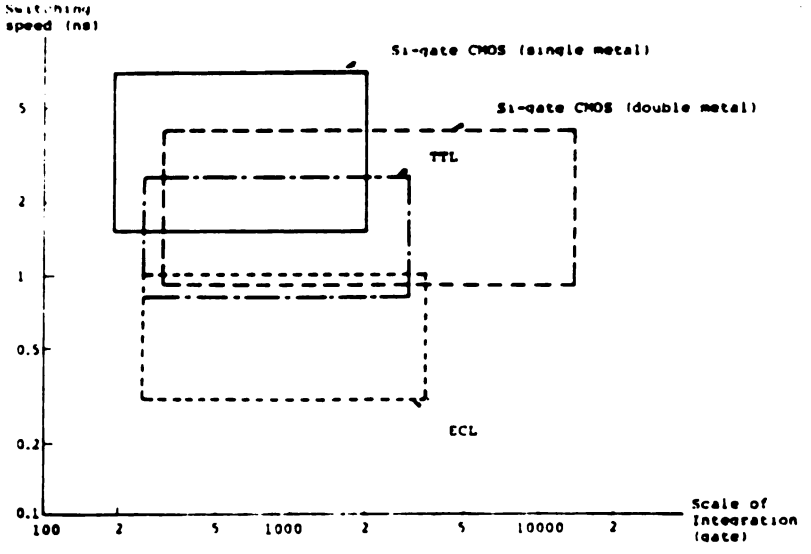


Figure 2.1. Speed versus integration scale [3].

CMOS and metal-gate CMOS. The former has become dominant because it allows for higher operation speed and higher degrees of integration by using finer pattern geometries. For example, 1.2 to 1.5 μm design rule devices exhibit 1.0 to 1.2 ns delays and subnanosecond delay is expected by using submicrometer rules [4].

The types of integrated circuits that are used for semicustom and custom designs include:

1) Gate Arrays (GAs). These are prefabricated unwired standard transistor arrays. The function of the gate array is defined by properly interconnecting transistor patterns on the chip.
2) Standard Cells (SCs). These allow for the design of application-specific integration circuits by composing circuits from elements stored in cell library.
3) Full Custom integration circuits (FCs). These are hand customized by system designers. Full custom designs are the most time-consuming to produce, but are capable of the highest density.

2.2 Design Methodology

Custom chips developed for particular applications tend to be more dedicated to the target system and less general purpose. The term "application-specific IC" clearly describes the present situation where many different types of circuits are being used in small quantities for specific applications. Smaller size, lower cost, higher speed and

higher reliability have been goals in the development of ASICs. Higher integration of devices, however, has led to longer design times, and made it difficult to maintain design reliability. In order to solve this dilemma, research and development work on various customization technologies and automated design technologies is being accelerated, with emphasis on design methodologies and automated tools.

Design methodology is defined as a set of codified techniques that are applicable to the VLSI design process. The objective of studying design methodology according to Baller [5] is to facilitate the creation of better designs. A final design must be functionally and physically correct, qualitatively acceptable, testable, and easily modified.

The prevalent design methodology today is hierarchical in nature. A hierarchical layout using the standard cell approach is given as an example in Figure 2.2 [6]. The VLSI techniques are categorized by those used for design, or synthesis, of a VLSI circuit and those used for its verification. In both categories, a further distinction is made between techniques relating to physical and functional views of the design. Physical design supports partitioning, layout, and topological analysis at all design levels. Functionality, testability, and physical design must be considered in parallel throughout the design process.

A design model is first defined according to some initial objectives and requirements. A top-down design process is normally used to decompose the desired operations of a circuit into a network of orchestrated smaller and simpler functional modules. Once a functional

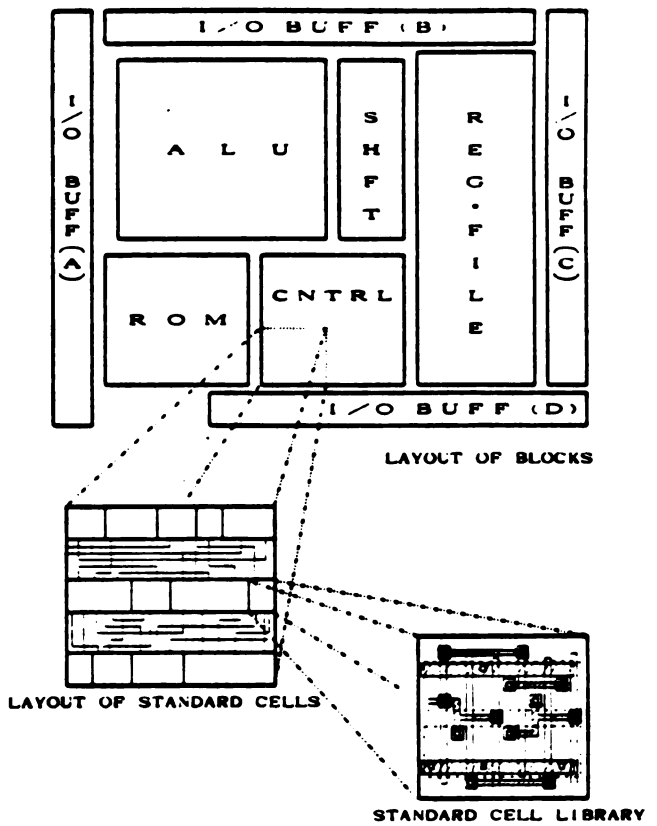


Figure 2.2. Hierarchical layout.

implementation strategy has been determined, a bottom-up process is commonly used to complete the physical design. Design results are simulated at different levels throughout the design process to evaluate correctness and performance [7].

The "Y" chart shown in Figure 2.3 has three axes that depict functional (behavioral), structural (architecture), and geometric (physical) representations of VLSI design. The functional axis

describes the functionality of the design, the structural axis represents the structural implementation of the design, and the geometric dimension represents the hardware realization of the design. Higher level of abstractions are encountered moving out radially from the origin. Commonly used levels of structural representation are the processor/memory/switch (PMS), the register-transfer, and the circuit level. Geometric representations include layout planning, cells, and physical mask geometries. VLSI design methodology, from this perspective, is a sequence of one-to-many mappings from higher to lower levels of abstraction and between different design representations in the design space. A particular mapping sequence taken by a certain school of thought, not surprisingly, more or less reflects a particular environment (usually expressed in needs and constraints of human, technological and financial resources) under which that particular methodology has evolved.

The design tools available to IC designers today usually encompass several of the lower level representations shown on the Y chart. For example, the designer can expect to enter at the logic level or perhaps the circuit level and find support for layout, logic and circuit simulation, design rule checking, and timing and critical path analysis. Many of these systems support designer interaction at any level of the design process and provide consistent database services across the entire range of design activities [4].

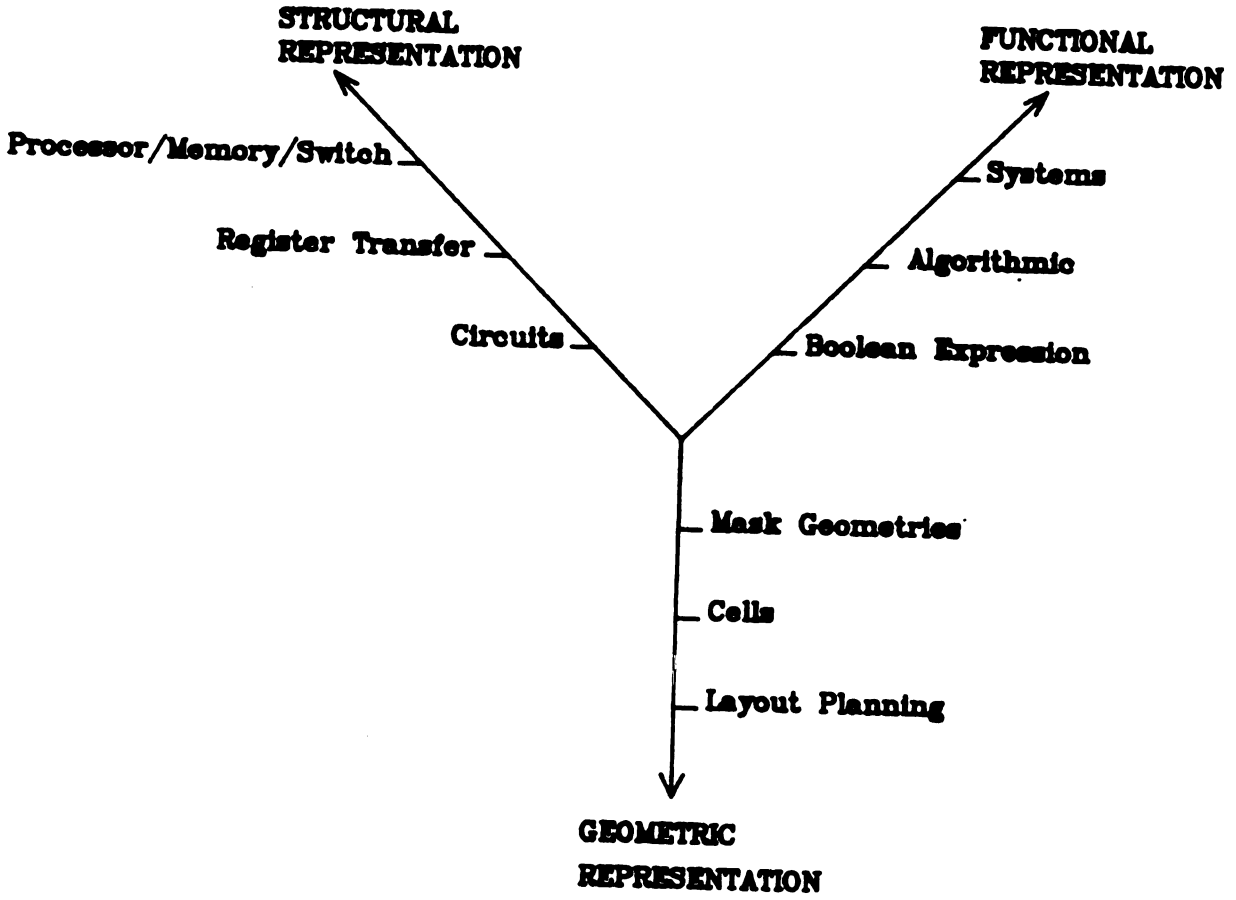


Figure 2.3. The "Y-chart" display of design representations.

2.3 New Design Technology

In software design, for the ease of programming, algorithms are written in high level languages. The same principle is applied to hardware design for which high level hardware description languages (HDL) are necessary. More than clarifying module specification definitions as in the early age, a ideal HDL should provide both functional and structural descriptions at all levels of abstraction. Many HDL's exist, but no one is widely used.

Besides HDL, silicon compilers have drawn much attention from CAE system developers. Silicon compilers have the goal of generating mask patterns directly from an HDL. In this case, the entire physical design process of a VLSI system can be reduced into a compilation process such that most of the characteristics of the chip under design can be accurately estimated. This allows system designers to focus on higher level design and to experiment with various algorithm/architecture alternatives with much less effort and cost. An elementary version of a silicon compiler is the module generator used to generate the mask layouts of some regular modules, such as basic cells in gate arrays and standard cells. "Concorde" is an example of one released by Seattle Silicon Technology and is more precisely classified as a macro-compiler. But the chip geometry generated by a macro-compiler is usually not optimal. How to automatically generate close-to-optimal geometry and achieve full compilation rather than macro-compilation are the two issues remaining to be solved for the silicon compiler in the near

future.

There has recently been much interest in using expert systems modeled on human expert knowledge. Such systems aim to capture the style and expertise of the human expert and provide flexibility to deal with a wide range of applications. Designs in VLSI usually involve certain forms of heuristics, such as those used for placement and routing, and thus it is appropriate to use expert system techniques to convert a hierarchical circuit description into a full-custom VLSI layout. The expert system for VLSI design may consist of a number of cooperating expert agents connected through a central manager. An example is given in Figure 2.4 [8]. Much effort has gone into the development of an expert system for VLSI design. Several issues, however, still remain open.

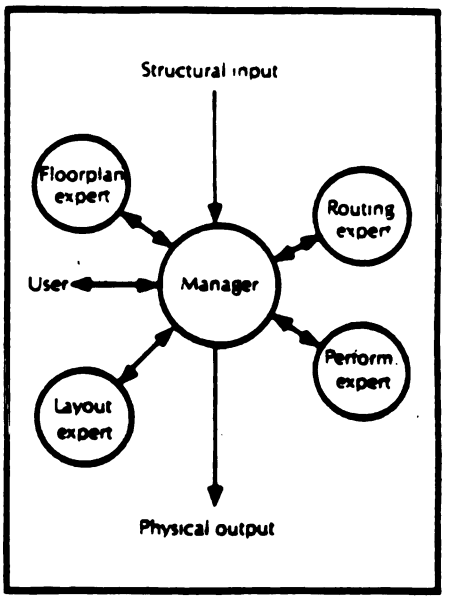


Figure 2.4. Cooperating expert agents for VLSI design [8].

1) The expert system should contain as much knowledge of VLSI design as possible.
2) The machine's learning capabilities should be augmented.
3) Special artificial intelligence machines msut be developed for providing high throughput expert systems.

This thesis focuses mainly on the design of a surface evaluation chip at the system, algorithmic, and register transfer levels. The final design will be expressed by a program written in RTL (Resigter Transfer Language).

CHAPTER III

MATHEMATICAL PRELIMINARIES

This chapter covers the mathematical preliminaries on which the surface evaluation algorithm is based. The B-spline space curve and rational B-spline space curve will be introduced first. Next the B-spline surface patch is defined and extended to the rational B-spline patch. In order to give the reader a clear picture of B-splines, simple examples are given. Finally, various classifications of B-spline are discussed.

3.1 B-spline Curve

A curve can be represented by a piecewise linear approximation in which each segment of the approximation is a straight line. Likewise, a curve can be represented as a piecewise polynomial approximation in which each segment of the composite figure is defined by a polynomial. A piecewise polynomial curve is called a spline.[10]

A B-spline is a parametric spline which consists of one or more segments. Each segment represents the curve over a range of the parameter t, for example $1 \leq t < 2$. The curve is partially defined by a set of control points (P_0 P_1 P_2 ..... P_n) (coordinate values) which produce an open-sided polygon when connected sequentially by straight

lines. Each segment of the curve is defined by a subset of the control points. A three-segment cubic B-spline curve with its control point polygon is shown in Figure 2.1. Segment ends are denoted by an X. The control points that are used to compute a particular segment are listed beside that segment [11].

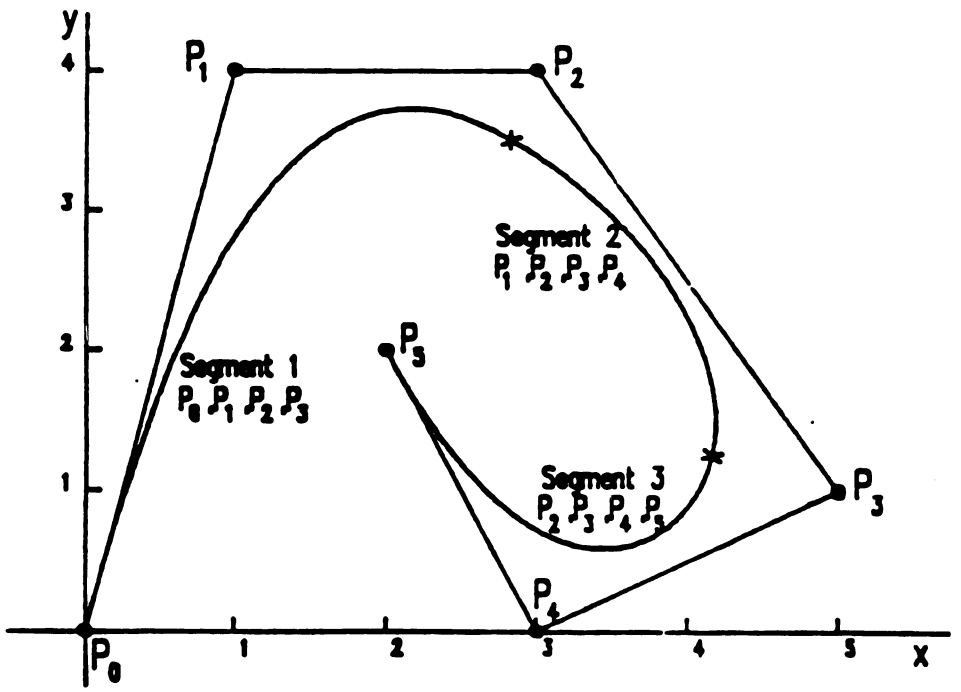


Figure 3.1. B-spline curve.

Let P(t) be the Cartesian position along a curve as a function of the parametric variable t. A segment of the spline is defined by the following equation:

$$P(t) = \sum_{i=0}^{n} P_i \, N_{i,k}(t) \qquad (2.1)$$

where

P_i is the control point,

n+1 is the number of control points,

$N_{i,k}$ is the blending function associated with P_i, and

k is the order of the blending functions.

The order k of the B-spline curve is the degree + 1, i.e., a quadratic is an order 3 curve and a cubic is an order 4 curve. P_i represents the vector of the control point coordinates such as $[x_i,y_i]$ for a 2-space curve, and likewise P(t) represents the vector of solution equations for the coordinates of points along the curve, such as [x(t),y(t)]. If the number of control points exceeds the order of the curve, the B-spline will have more than one segment. In this case, only a subset of the of control points is used for each segment, because the result of formulation of the blending functions associated with these unused control points for the segment are zero. Examples will be given in the following sections.

The number of segments the curve will have can then be represented as a function of the number of control points, n+1 and the order, k, as

$$\text{number of segments} = n - k + 2. \qquad (3.2)$$

The parameter range for each segment can be determined by simple observation of the knot vector which will be introduced next.

3.1.1 Knot Vector

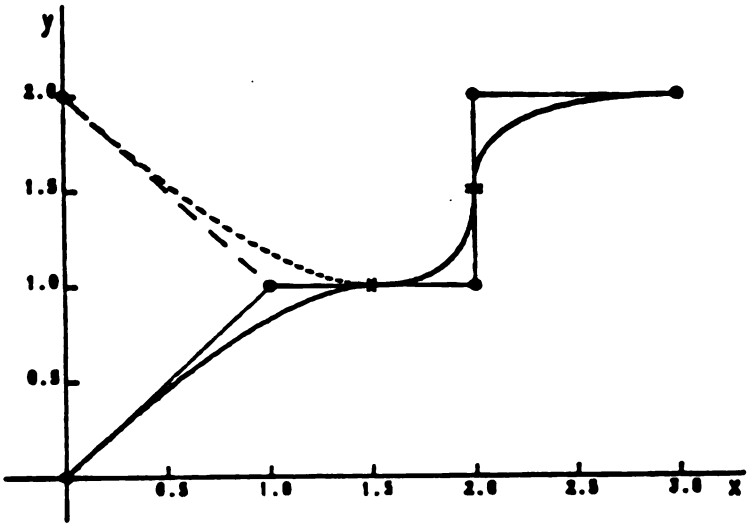
The blending functions are determined by a set of values called knots. They are displayed as [x_0 x_1 x_2 x_3 ..... x_ℓ] and are called the knot vector. In order to make the curve pass through the end points, the first and last entries of the knot are repeated k times where k is the order of the curve. Thus $x_0=x_1=\ldots=x_{k-1}$ and $x_{\ell-(k-1)}=\ldots=x_{\ell-1}=x_\ell$. The length of the knot vector refers to the number of elements (knots) in the vector. The above vector has length = $\ell+1$. The length is a function of the control points (n+1) and the order (k) of the curve, given by n+k+1. The order of the curve must be less than or equal to the number of control points.[11]

For example, if n+1 = 4 and k = 4 (cubic), the knot vector is [0 0 0 0 1 1 1 1]. This curve is a special case of the B-spline, called the Bezier spline, and has only one segment. If n+1=4 and k=3 (quadratic), one possible knot vector is [0 0 0 1 2 2 2]. There are n - k + 2 = 2 segments. The first segment of the curve, produced by the knot vector shown, has the parameter range t=0 to t=1 and a zero blending function with the fourth control point. The second segment has a zero function associated with the first control point and covers the parameter range t=1 to t=2.

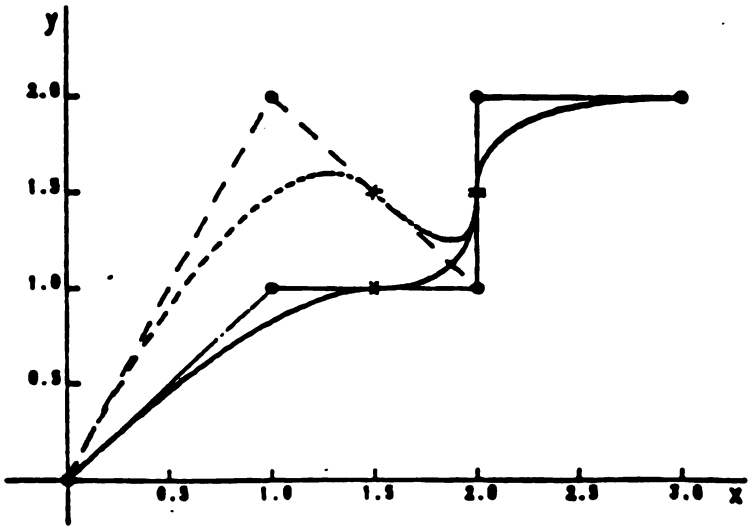
Since each segment of a spline depends on a subset of control points, each control point affects only a local area of the B-spline curve. This property is called locality and is one of the key features of the B-spline. The B-spline curve can be modified by simply moving

the locations of some control points. An example of locality is given in Figure 3.2 with n=4, k=3 (quadratic), and knot vector [0 0 0 1 2 3 3 3] which results in a three segment curve. The first segment depends only on the first three control points and has the parameter range of t=0 to t=1. The second segment depends on the middle three control points and has the parameter range of t=1 to t=2. The third and last segment depends on the last three control points and has the parameter range of t=2 to t=3. In Figure 3.2(a), the location of the first control point is moved, and in Figure 3.2(b) the second control point is moved.

A knot vector with a repeated interior knot, such as [0 0 0 1 1 2 2 2] with n=4 and k=3, causes the length of the second segment of the curve to be zero. Usually, the order of the continuous derivative of a B-spline curve at its segment joints is up to k-2. For example, the order of a B-spline curve with k=4 should have continuous zero, first, and second derivatives. Repeating a knot once (multiplicity of knot = 2) causes the curve to lose the highest order derivative continuity at the segment joint corresponding to that knot. The relationship between multiplicity m, the order of the curve k, and the order of the derivatives that are continuous across a segment joint is (0, 1, ..., k-m-1). Thus, if the B-spline curve is a cubic (k=4) and an interior knot has a multiplicity of 4, then k-m-1=-1 and the curve will be in general discontinuous pointwise and in all derivatives.



(a)



(b)

Figure 3.2. Example of locality (a) effect of moving the first control point, (b) effect of moving the second control point.

It is not necessary that the knot values be integer. If all interior knots are consecutive integeters, this type of knot vector is called a uniform knot vector. Relaxing a uniform knot vector to allow repeated interior knots an produces enhanced uniform knot vector [1]. The general knot vector with monotonically non-decreasing real knot values produces a nonuniform knot vector, for example [0 0 0 .3 .7 1 1 1].

3.1.2 Blending Functions

As illustrated in the previous examples, the relationships between the knot vector, the control points, and the curve are that the curve is defined by the control points and the blending functions which are functions of the knots. The formal definition of blending functions are

$$N_{i,1}(t) = \begin{cases} 1 \text{ if } x_i \le t < x_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

$$N_{i,k}(t) = \frac{(t - x_i)\, N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(k_{i+k} - t)\, N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}}. \quad (3.3)$$

The x_i is the value of the ith position of the knot vector, for example:

x = [0 0 0 0 1 2 3 4 5 6 7 8 9 9 9 9].

i = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

From equation (2.3), if $x_i = x_{i+1}$, then $N_{i,1}(t) = 0$, which causes the

curve to lose the highest-order derivative continuity as mentioned in the last section. For each segment there is one $N_{i,1}(t)$ that is nonzero. This nonzero function cascades down to the final blending functions associated with that segment as shown in the Figure 3.3. Note that the number of nonzero blending functions per segment is equal to the order of the curve.[11]

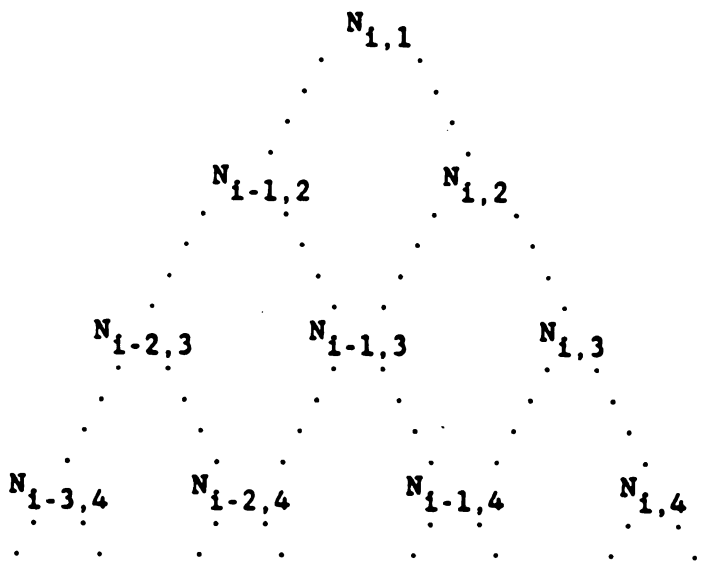


Figure 3.3. Blending function cascade.

A clear example of blending functions quoted from [2] is:

Control points : (0,0) , (2,2) , (3,1) , (3,0)
Order : k = 3
Knot vector : [0 0 0 1 2 2 2]
i = 0 1 2 3 4 5 6.

From equation (2.1),the spline equation is $P(t) = \sum_{i=0}^{3} P_i N_{i,3}$. There

are four blending functions $N_{0,3}$, $N_{1,3}$, $N_{2,3}$, and $N_{3,3}$ that need to be formulated. Tracing from the bottom of the tree shown in Figure 3.4, we know which blending functions need to be computed.

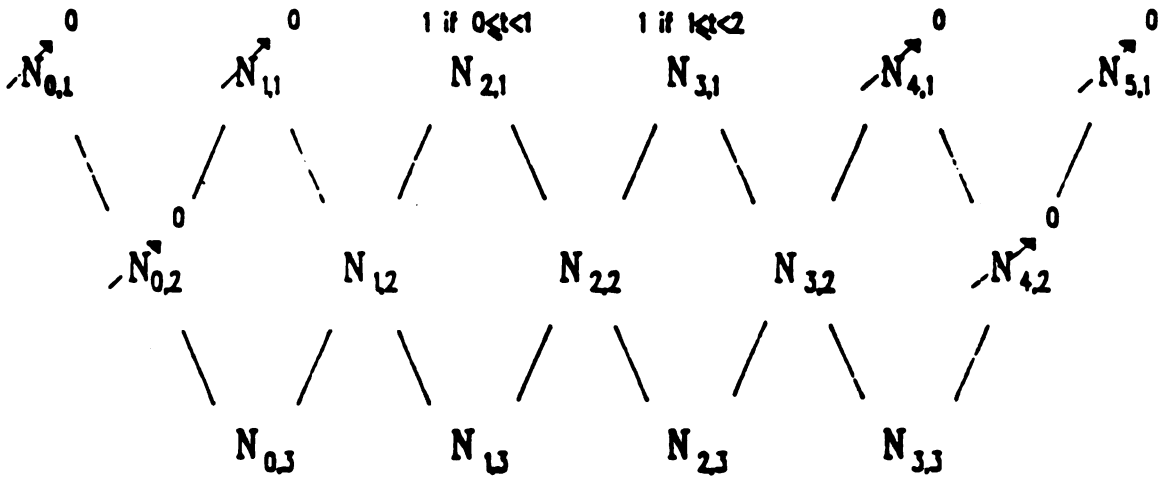


Figure 3.4. Blending function tree.

In this example, only two of $N_{i,1}$ are not zero, which agrees with the fact that the curve will have two segments. The function tree used for determining the first segment is given in Figure 3.5 Therefore, the blending functions $N_{0,3}$, $N_{1,3}$, and $N_{2,3}$ are nonzero and will be a function of $N_{1,2}$ and $N_{2,2}$ found by substituting the corresponding values into equation (3.2). Then one can get the following blending functions:

$$N_{1,2} = \frac{(x_3 - t)\, N_{2,1}}{x_3 - x_2} = 1 - t \tag{3.4}$$

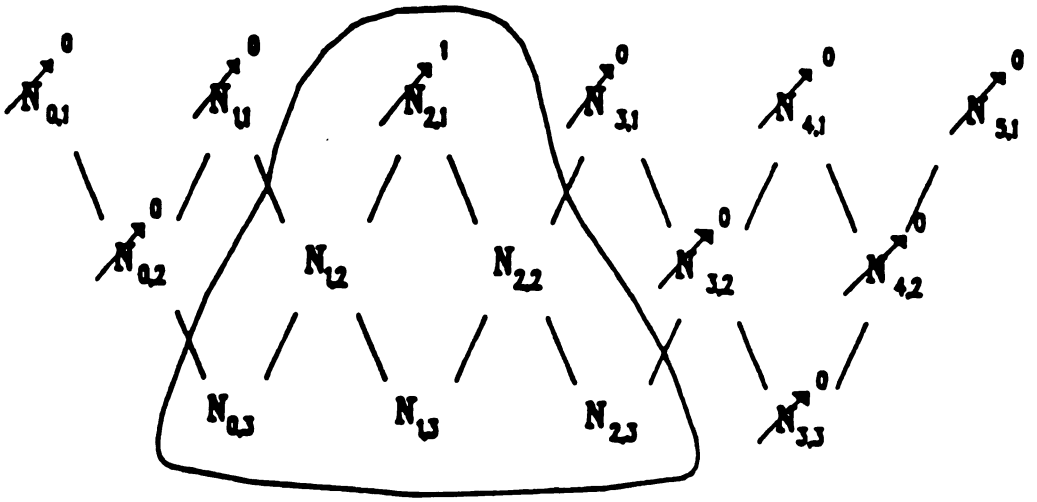


Figure 3.5. Blending function tree - segment 1.

$$N_{2,2} = \frac{(t - x_2)\, N_{2,1}}{x_3 - x_2} = t \tag{3.5}$$

$$N_{0,3} = \frac{(t - x_1)\, N_{0,2}}{x_2 - x_0} + \frac{(x_3 - t)\, N_{1,2}}{x_3 - x_1} = t^2 - 2t + 1 \tag{3.6}$$

$$N_{1,3} = \frac{(t - x_1)\, N_{1,2}}{x_3 - x_1} + \frac{(x_4 - t)\, N_{2,2}}{x_4 - x_2} = -1.5t^2 + 2t \tag{3.7}$$

$$N_{2,3} = \frac{(t - x_2)\, N_{2,2}}{x_4 - x_2} + \frac{(x_5 - t)\, N_{3,2}}{x_5 - x_3} = .5t^2 \tag{3.8}$$

Note that the convention of $0/0 = 0$ is used when computing the blending functions.

For the second segment the blending functions $N_{1,3}$, $N_{2,3}$, and $N_{3,3}$ are nonzero and can be formulated by $N_{2,2}$ and $N_{2,3}$ as shown in Figure

3.6. These are computed in a manner similar to the first segment as:

$$N_{2,2} = \frac{(x_4 - t)\, N_{3,1}}{x_4 - x_3} = 2 - t \tag{3.9}$$

$$N_{3,2} = \frac{(t - x_3)\, N_{3,1}}{x_4 - x_3} = t - 1 \tag{3.10}$$

$$N_{1,3} = \frac{(t - x_1)\, N_{1,2}}{x_3 - x_1} + \frac{(x_4 - t)\, N_{2,2}}{x_4 - x_2} = .5t^2 - 2t + 2 \tag{3.11}$$

$$N_{2,3} = \frac{(t - x_2)\, N_{2,2}}{x_4 - x_2} + \frac{(x_5 - t)\, N_{3,2}}{x_5 - x_3} = -1.5t^2 + 4t - 2 \tag{3.12}$$

$$N_{3,3} = \frac{(t - x_3)\, N_{3,2}}{x_5 - x_3} + \frac{(x_6 - t)\, N_{4,2}}{x_6 - x_4} = t^2 - 2t + 1. \tag{3.13}$$

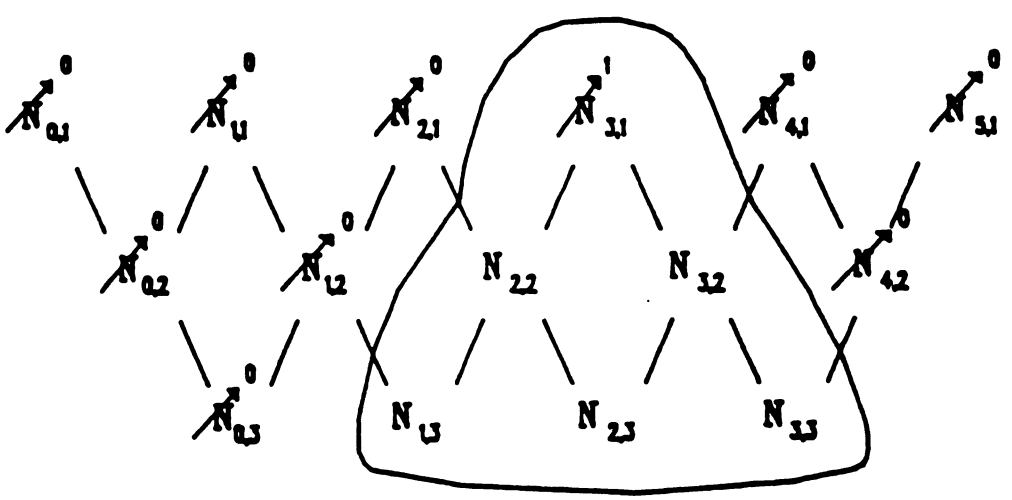


Figure 3.6. Blending function tree - segment 2.

Table 3.1 summarizes the blending functions computed above and Figure 3.7 gives a plot of the blending functions.

After the blending functions have been determined, the spline equation of the first segment, $0 \le t < 1$, can be expressed as

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 3 & 3 \\ 0 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^2 - 2t + 1 \\ -1.5t^2 + 2t \\ 0.5t^2 \\ 0 \end{bmatrix} \tag{3.14}$$

and for the second segment, where $1 \le t < 2$,

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 3 & 3 \\ 0 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0.5t^2 - 2t + 2 \\ -1.5t^2 + 2t - 2 \\ t^2 - 2t + 1 \end{bmatrix} \tag{3.15}$$

Table 3.1. Blending function table of the example.

i	$N_{i,3}(t)$, $0 \le t < 1$	$N_{i,3}(t)$, $1 \le t < 2$
0	$t^2 - 2t + 1$	0
1	$-1.5t^2 + 2t$	$0.5t^2 - 2t + 2$
2	$0.5t^2$	$-1.5t^2 + 4t - 2$
3	0	$t^2 - 2t + 1$

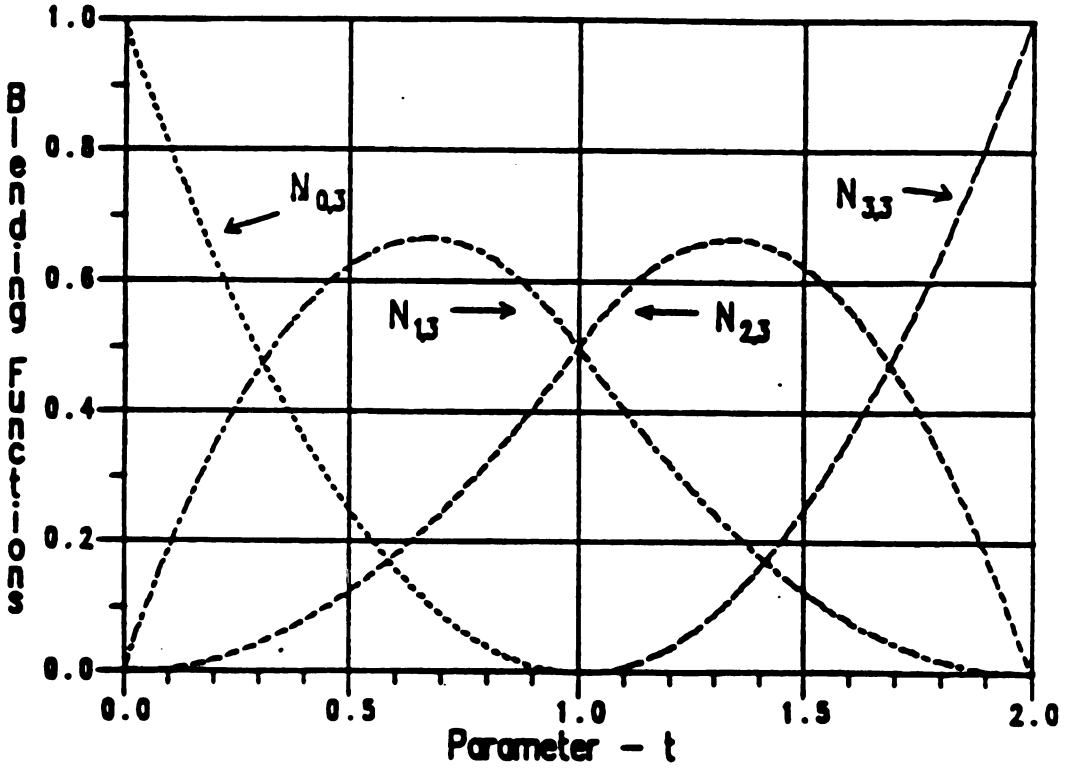


Figure 3.7. Plot of blending functions.

The plot of this two segment curve is shown in Figure 3.8 along with the polygon of control points; the segment joint is marked with an X.

As demonstrated in the above example, when the blending functions for a segment of a B-spline are cmputed, only a subset of the knot vector is used. This subset of the knot vector is called the effective knot vector. The length of the effective knot vector is equal to 2(k-1), where k is the order. In above example, the length of the effective

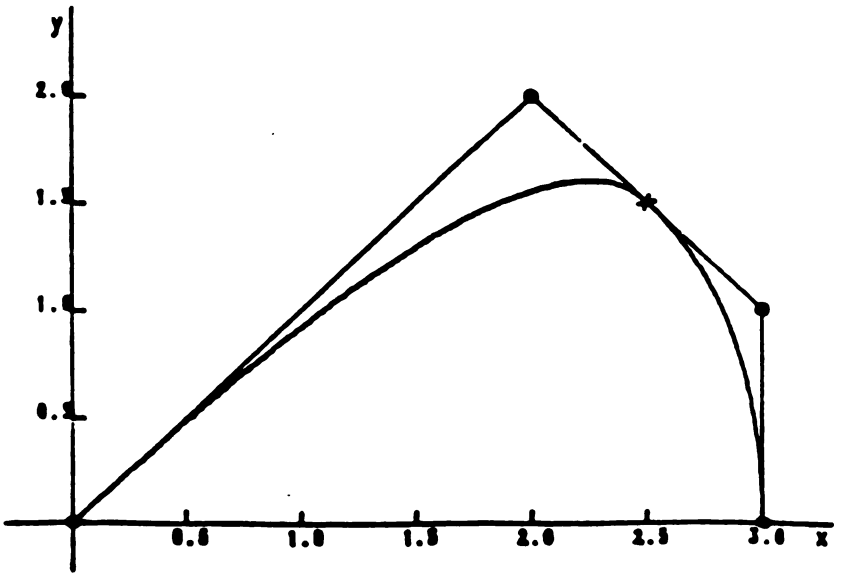


Figure 3.8. Third order B-spline.

knot vector is 2(3-1)=4, which agrees with the computation of blending functions. The only knot values used in the computation of the blending functions of the first segment are x_1, x_2, x_3, and x_4, or [0 0 1 2], while the blending functions for the second segment use only the knot values x_2, x_3, x_4, and x_5, or [0 1 2 2].

3.2 Rational B-splines

A rational B-spline is one in which the control points are expressed in homogeneous coordinates. A point is given an additional coordinate which can be thought of as a scale factor. Instead of P(x,y) a point is represented as P'(wx, wy, w). The last coordinate is the

homogeneous variable. There are an infinite number of homogeneous representations of a single coordinate point, for example the coordinate x=1, y=2, or (1,2), can be represented by (1,2,1), (2,4,2), or (5,10,5). If the homogeneous variable is equal to 1, then the other variables represent the actual Cartesian coordinates. If the homogeneous variable, also called the weight in rational B-splines, is not equal to 1, then the transformation back to actual coordinates, P(x,y,1), is calculated by dividing each coordinate of P' by the calculated value of w.

There is no effect on the blending functions when the control points are expressed in homogeneous coordinates, because the blending functions depend only on the knot vector. A point WP(t) along the spline in homogeneous coordinates can then be expressed as

$$WP(t) = \sum_{i=0}^{n} WCP_i * N_{i,k}(t), \tag{3.16}$$

where the WCP_i are the homogeneous coordinate values of the weighted control points. WP(t) represents wx(t), wy(t), and w(t), which is the vector of homogeneous coordinate values. The Cartesian coordinate values of points P(t) along the spline are then calculated as follows.

$$x(t) = \frac{wx(t)}{w(t)} \tag{3.17}$$

$$y(t) = \frac{wy(t)}{w(t)} \tag{3.18}$$

Extending the earlier example,let the weights of control points

(0,0), (2,2), (3,1), and (3,0) be 1, 2, 1, and 1, respectively, for example. The control points in weighted or homogeneous form are:

$$P_0 = (0,0,1);$$
$$P_1 = (4,4,2);$$
$$P_2 = (3,1,1);$$
$$P_3 = (3,0,1).$$

The computation of coordinates WP(t) for the first segment is then straightforward.

$$\begin{bmatrix} wx(t) \\ wy(t) \\ w(t) \end{bmatrix} = \begin{bmatrix} 0 & 4 & 3 \\ 0 & 4 & 1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} t^2 - 2t + 1 \\ -1.5\,t^2 + 2t \\ .5\,t^2 \end{bmatrix}$$

$$= \begin{bmatrix} -4.5\,t^2 + 8t \\ -5.5\,t^2 + 8t \\ -1.5\,t^2 + 2t + 1 \end{bmatrix} \qquad (3.19)$$

Thus $x(t) = \dfrac{-4.5\,t^2 + 8t}{-1.5\,t^2 + 2t + 1}$ and $y(t) = \dfrac{-5.5\,t^2 + 8t}{-1.5\,t^2 + 2t + 1}$

for $0 \le t < 1$. Notice that x(t) and y(t) are now ratios of polynomials in t. Similarly, we can calculate the WP(t) for the second segment.

$$\begin{bmatrix} wx(t) \\ wy(t) \\ w(t) \end{bmatrix} = \begin{bmatrix} 4 & 3 & 3 \\ 4 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix} \begin{bmatrix} .5\,t^2 - 2t + 2 \\ -1.5\,t^2 + 4t - 2 \\ t^2 - 2t + 1 \end{bmatrix}$$

$$= \begin{bmatrix} .5\,t^2 - 2\,t + 5 \\ .5\,t^2 - 4\,t + 6 \\ .5\,t^2 - 2\,t + 3 \end{bmatrix} \tag{3.20}$$

Thus $x(t) = \dfrac{.5\,t^2 - 2\,t + 5}{.5\,t^2 - 2\,t + 3}$ and $y(t) = \dfrac{.5\,t^2 - 4\,t + 6}{.5\,t^2 - 2\,t + 3}$

for $1 \leq t < 2$.

The solid line curve in Figure 3.9 is the rational B-spline derived above and the dashed line is the nonrational B-spline from the previous example. Note how the double weight on the (2,2) coordinate pulled the curve toward it.[11]

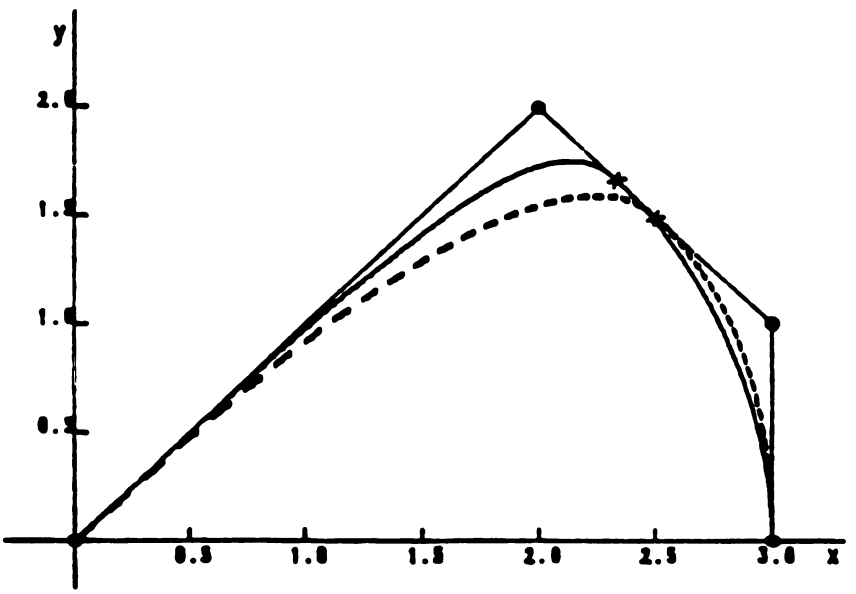


Figure 3.9. Strictly rational B-spline.

Weighting each of the control points gives another degree of freedom to the curve per control point and allows the representation of curves that can not be fitted with nonrational B-splines, such as circular arcs derived from conic sections.

The example shown above is restricted to 2-dimensional curves. The representation of curves in 3-dimension space (x(t), y(t), and z(t)) with B-splines can be achieved by simply adding another row to the control point matrix for the calculation of zw(t).

3.3 B-spline Surface

The extension from the spline curve to the surface patch is done by adding a second parametric coordinate and an associated knot vector. The most commonly used surface representation is the four sided patch. Each point on the patch is a function of two parameters, s and t. One edge or side of the patch and its opposite side are chosen as curves that depend only on s and the other two edges are functions only of t. Both s-varying sides share the same knot vector, as do both t-varying sides. The s knot vector does not have to equal the t knot vector. They can be of different orders and have a different number of control points. If the s edges have m+1 control points and the t edges have n+1, then there will be a total of $(m+1)\cdot(n+1)$ control points. The control points joined together by straight lines form what is called a control point net. The general shape of the patch will mimic the net (more precisely, the surface is contained within the convex hull formed by the control

points).

The general formula for determining the coordinates of a point on the B-spline surface is

$$P(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} M_{j,h}(s) \cdot P_{i,j} \cdot N_{i,k}(t) \tag{3.21}$$

or in matrix form

$$P(s,t) = \begin{bmatrix} M_{0,h}(s) & M_{1,h}(s) & \cdots & M_{m,h}(s) \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0n} \\ P_{10} & P_{11} & \cdots & P_{1n} \\ \vdots & \vdots & & \vdots \\ P_{m0} & P_{m1} & \cdots & P_{mn} \end{bmatrix} \begin{bmatrix} N_{0,k}(t) \\ N_{1,k}(t) \\ \vdots \\ N_{n,k}(t) \end{bmatrix} \tag{3.22}$$

where

P_{ij} is one of the $(n+1)\cdot(m+1)$ control points,

k is the order of the curve in the t direction,

h is the order of the curve in the s direction,

$n+1$ is the number of control points in the t direction,

$m+1$ is the number of control points in the s direction,

N_{ij} is one of the $n+1$ blending functions of order k,

M_{ij} is one of the $m+1$ blending functions of order h.

Just as the B-spline curve consisted of segments, the B-spline surface consists of subpatches. The s edge B-splines have $(m-h+2)$ segments and the t edge B-splines have $(n-k+2)$ segments. Each subpatch is formed by a segment of the s edge paired with a segment of the t edge. Therefore, the number of subpatches is $(m-h+2)\cdot(n-k+2)$. Each

subpatch covers a range of s and a range of t.

Example [2]:

s edges m+1=4, h=3 (quadratic), knot vector [0 0 0 1 2 2 2]

t edges n+1=2, k=2 (linear), knot vector [0 0 1 1]

There are two subpatches in this case, since there are two segments in the s direction and one in the t direction. One subpatch covers the range of s=0→1 and t=0→1, and the other covers the range of s=1→2 and t=0→1. There are altogether (n+1)•(m+1) = 8 control points, but each subpatch use only a subset of the control points. The number of control points that contribute to each subpatch is h•k, which is 6 in this example. The subpatch with the range of s=0→1 and t=0→1 has the following blending functions.

$$M_{0,3}(s) = s^2 - 2s + 1 \qquad N_{0,2}(t) = 1 - t$$
$$M_{1,3}(s) = -1.5s^2 + 2s \qquad N_{1,2}(t) = t$$
$$M_{2,3}(s) = 0.5s^2$$
$$M_{3,3}(s) = 0$$

The coordinate of a surface point on this subpatch can then be calculated as

$$P(s,t) = \begin{bmatrix} s^2-2s+1 & -1.5s^2+2s & 0.5s^2 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \\ P_{20} & P_{21} \end{bmatrix} \begin{bmatrix} 1-t \\ t \end{bmatrix}. \qquad (3.23)$$

The first column elements of the control point matrix are the first three control points of one of the s edges. The first row elements of the control point matrix are the two control points on one t edge. The surface points for the other subpatch with the range of s=1→2 and t=0→1 are computed by

$$P(s,t)=\begin{bmatrix}0.5s^2-2s+2 & -1.5s^2+4s-4 & s^2-2s+1 & 0.5s^2\end{bmatrix}\begin{bmatrix}P_{10} & P_{11}\\ P_{20} & P_{21}\\ P_{30} & P_{31}\end{bmatrix}\begin{bmatrix}1-t\\ t\end{bmatrix}. \quad (3.24)$$

If we choose the control points given in Figure 3.10 for this example, we can generate the B-spline surface shown in Figure 3.11.[11]

More complex surfaces may be constructed if a weight is also assigned to each control point of the B-spline surface. To determine

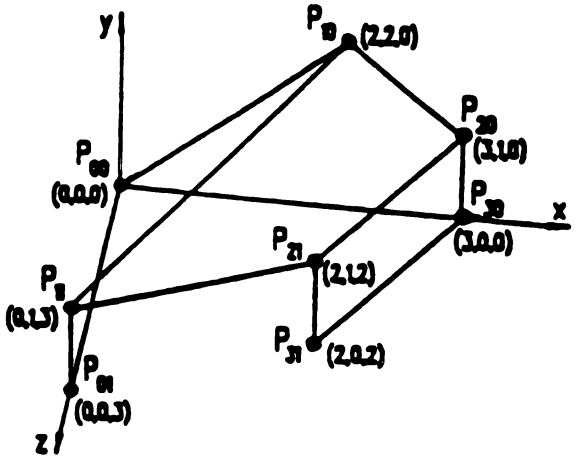


Figure 3.10. B-spline control point net.

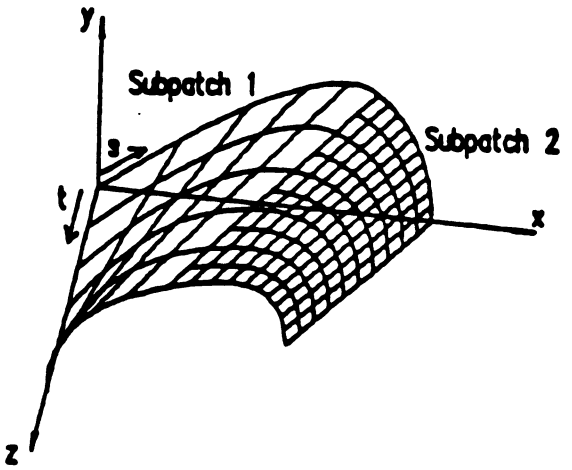


Figure 3.11. B-spline surface.

points on the surface, equation (3.16) is used in the homogeneous coordinate system as described above.

More examples of B-spline curves and surfaces can be found in [11] and in [1]. The next chapter will give a literature review of the work being done by others related to surface representation and evaluation.

CHAPTER IV

LITERATURE REVIEW

A fundamental area of interest in computer graphics and computer-aided geometric design is the representation of a 3-space object whose shape is delineated by free-form surfaces. Various mathematical models as well as special purpose hardware have been developed to meet the requirement for implementation in an interactive computer-aided geometric design system. This chapter presents a review of techniques in use for the display of free-form surfaces.

4.1 Uniform Subdivision

There are two stages involved in displaying parametrically defined surfaces in most existing system -- creating a polygonal approximation of the surface and displaying the approximation by use of the standard polygon display techniques [10]. Uniform subdivision is the simplest technique to produce the polygonal database for displaying the surface. In uniform subdivision, a single patch is divided uniformly in parameter space and the resulting mesh of evaluated points are treated as vertices of the approximating triangles, for example. The more subdivisions, the smoother the approximating surface.

Note that the vertices of the approximating triangle are used for interpolating the points on the patch. Yet the control points used in B-splines define the patch rather than interpolate the patch. This is the key difference between B-spline and piecewise linear approximation.

The first attempt at displaying a bicubic surface, a surface with order equal to 4 in both the s and t directions, directly from the parametric mathematical definition of the surface (i.e., without polygonalizing) was done by Catmull [13]. Catmull subdivided the patch in parameter space down to the pixel basis. Then the subpatch was passed to displaying and shading routines.

4.2 Adaptive Subdivision

Adaptive subdivision, proposed by Lane and Carpenter [14], adaptively divides a patch into polygons according to the flatness of the patch. A "flat" patch can be displayed as a single polygon while a curved patch is subdivided into smaller subpatches until each of the subpatches meets the criteria of the flatness test. Compared to uniform subdivision, adaptive subdivision generates "enough" polygons to adequately represent the surface whereas the uniform subdivision spends a longer time to generate "more than enough" polygons [10].

All surface patches are represented in terms of an appropriate tensor product of Bernstein basis functions. Using Bernstein basis functions, curves are generated by interpolating their first and last control points. Then the points are assembled into quadrilaterals. The

flatness test is carried out on the quadrilaterals, which are further subdivided if necessary. The final subdivision produces polygons for display by a conventional routine [10].

In order to decide which portion is the most curved one of the segment or patch, the calculation of curvature is usually involved. Yet the evaluation of curvature is difficult and time-consuming. Van Hook has since developed a method similar to the Lane-Carpenter method, but in which no calculation of curvature is necessary. Van Hook evaluated more than enough points on the surface to achieve the flatness test. This method was classified by Whitton [10] as a semi-adaptive subdivision.

4.3 Cox-deBoor Algorithm

The most popular B-spline algorithm in many CAD/CAM systems is the Cox-deBoor algorithm [16,17]. It works well for point evaluation as well as the derivative evaluation of the point. The algorithm for formulating the blending functions presented by deBoor [17] is reproduced below:

```
N(1,1) = 1;
for s = 1 to k-1
begin
    DP(s) = t_{i+s} - t;
    DM(s) = t - t_{i+1-s};
```

```
N(1,s+1) = 0;
for r = 1 to s
begin
    M = N(r,s) / ( DP(r) + DM(s+1-r) );
    N(r,s+1) = N(r,s+1) + DP(r) * M;
    N(r+1,s+1) = DM(s+1-r) * M;
endfor,
endfor.
```

Where

k is the order of the blending functions,

t_i are the knots of the knot vector,

and $t_i \leq t < t_{i+1}$.

The B-spline is then evaluated by

$$P(t) = \sum_{i=0}^{k-1} CP_i \cdot N_{i,k}(t), \tag{4.1}$$

where $N_{i,k}$ is the k^{th} column of N in the above algorithm.

Note that N is a triangular matrix. To evaluate derivatives, the appropriate column of N would be used. The general routine given by deBoor for derivative evaluation is

$$F^{(j)}(t) = (k-1) \cdot \cdot \cdot (k-j) \sum_{i=0}^{k-j-1} A_i^{(j)} \cdot N_{i,k-j}(t), \tag{4.2}$$

where

j is the order of derivative and $0 \le j < k$,

$N_{i,k-j}$ is the $(k-j)^{th}$ column of N,

and
$$\begin{cases} A_i^{(0)} = CP_i, \\ A_i^{(j)} = (A_i^{(j-1)} - A_{i-1}^{(j-1)}) \,/\, (t_{i+k-j} - t_i). \end{cases} \qquad (4.3)$$

4.4 Pickelmann's Algorithm

One problem embedded in the Cox-deBoor algorithm is that when the knot values of the knot vector become very large, the error will also increase. Knot vectors are, in Pickelmann's work [1], translated (or shifted) so that the parameter range being evaluated (the middle of the effective knot vector) is always [0,1). This translation both limits the number of possible effective knot vectors and has a beneficial effect on numerical accuracy of the blending function calculations. The CURBS algorithm proposed by Pickelmann [1] thus produces a three-to-one reduction in machine operations over the Cox-deBoor algorithm, for cubic splines. For higher order splines, the reduction ratio is even larger.

From the last chapter, only the effective knot vector (a subset of the knot vector is used to compute any given blending function. Also, the length of the effective knot vector is 2(k-1). Pickelmann showed that a B-spline with an infinite number of possible knot vectors can be processed as several individual subsegments, each with one effective knot vector. In what follows, only enhanced uniform knot vectors will

be considered (note that Pickelmann showed that this is not actually a restriction on the class of all nonuniform rational B-splines; he derived an algorithm for converting an arbitrary NURB surface into one with an enhanced uniform knot vector). And for EURB knot vectors, using the knot vector translation just mentioned, the number of possible effective knot vectors is finite and equal to $2^{(2k-4)}$. For a cubic, k=4, there are $2^{(2*4-4)}$ or 16 possible effective knot vectors for a subsegment. For k=5, there are 64 possible knot vectors.

Each of the $2^{(2k-4)}$ knot vectors yields a unique set of blending functions. In order to easily determine which set of blending functions should be used for a given subsegment, each subsegment is given a label or pointer to the correct set of preformulated blending functions. The label is determined by a special subroutine in which the effective knot vector is compressed into a binary-encoded label. Then the shape of a subpatch is formed by the labels in both the s and t directions and the corresponding control points. To get a clear picture of the algorithm, an example is given as follows.

Example of B-spline surface

s edges - number of control points : m+1 = 4

order : h = 3 knot vector : [0 0 0 1 2 2 2]

t edges - number of control points : n+1 = 4

order : k = 3 knot vector : [0 0 0 1 2 2 2]

Control point matrix : $$\begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

For k=3, there are $2^{(2\cdot3-4)} = 4$ possible knot vectors. The above knot vector uses only two of them, [0 0 1 2] and [0 1 2 2], and the second effective knot vector can be translated to [-1 0 1 1] which will have the same label as [0 1 2 2]. The label of [0 0 1 2] is LAB1 = 2, and the label of [-1 0 1 1] is LAB2 = 1. The way to label the effective knot vector was given in [1].

Since (n-k+2) = 2, we know there are two segments in both the s and t directions, which results altogether in four subpatches. The blending functions of one of the four subpatches can then be retrieved from the preformulated blending function basis according to LAB1 and LAB2. The labels and the corresponding control points used to form the subpatch are given in Table 4.1. Figure 4.1 shows the B-spline control net and Figure 4.2 gives the B-spline surface.

Conceptually, the four subpatches have different parameter ranges as shown in the above table. But for purposes of computation, subpatches can be treated individually as if each of them had the parameter range of s=0→1 and t=0→1, so long as the "bookkepping" of the proper control point indices is maintained.

TABLE 4.1. Subpatch and corresponding control points

subpatch	label of s	label of t	control ponints
s = 0→1 t = 0→1	LAB1	LAB1	P_{00} P_{01} P_{02} P_{10} P_{11} P_{12} P_{20} P_{21} P_{22}
s = 1→2 t = 0→1	LAB2	LAB1	P_{10} P_{11} P_{12} P_{20} P_{21} P_{22} P_{30} P_{31} P_{32}
s = 0→1 t = 1→2	LAB1	LAB2	P_{01} P_{02} P_{03} P_{11} P_{12} P_{13} P_{21} P_{22} P_{23}
s = 1→2 t = 1→2	LAB2	LAB2	P_{11} P_{12} P_{13} P_{21} P_{22} P_{23} P_{31} P_{32} P_{33}

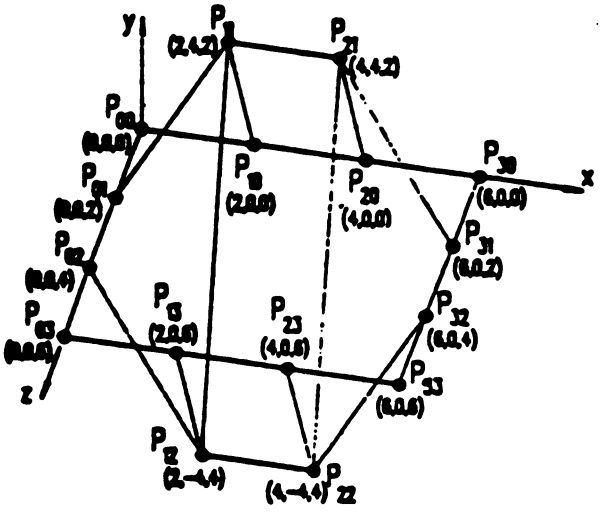


Figure 4.1. B-spline control point net [11].

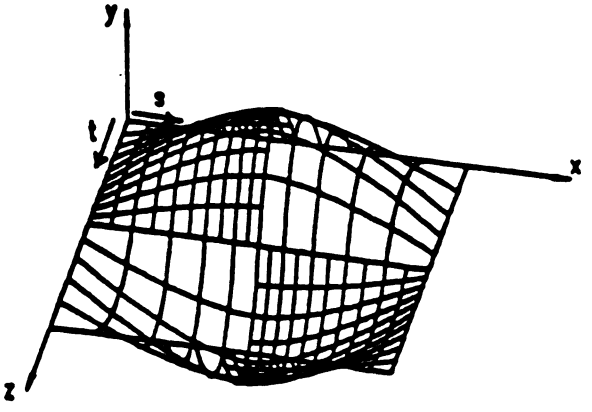


Figure 4.2. B-spline surface [11].

CHAPTER V

SURFACE EVALUATION AND SHADING

This chapter describes the surface evaluation algorithm for both nonrational B-splines and rational B-splines. The computation of the derivative and the normal vector of the points on the B-spline surface are given. Finally, based on the normal of the surface, the concept of shading is introduced and the shaded image of the B-spline surface is visualized.

5.1 Evaluation Algorithm

From equation (3.20) we know a point on the B-spline patch can be represented in matrix form as

$$P(s,t) = [BF(s)] \cdot [CP] \cdot [BF(t)] \tag{5.1}$$

where CP is the control point matrix with size $(m+1)\times(n+1)$, and BF(s) and BF(t) are the blending function vectors of s and t, respectively. In the last chapter, we showed that the computation of the points on a subpatch can be treated individually such that the point on a subpatch can now be defined as

$$P'(s,t) = [BF(s)] \cdot [CP'] \cdot [BF(t)] \tag{5.2}$$

where CP' is the corresponding control point matrix with size h×k, and each element of BF(s) and of BF(t) is a polynomial of s and of t, respectively. For a bicubic surface, h=k=4, we may rewrite equation (5.2) as

$$P'(s,t) = [\ 1\ s\ s^2\ s^3\ ] \begin{bmatrix} M_{4\times4} \end{bmatrix} \begin{bmatrix} CP'_{4\times4} \end{bmatrix} \begin{bmatrix} N_{4\times4} \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}. \qquad (5.3)$$

$M_{4\times4}$ and $N_{4\times4}$ are the blending coefficient matrices by which the blending function is formulated. This assumes that all the blending coefficient matrices have been preformulated according to the labels determined from the effective knot vector. Note that the formulation of the blending functions is done only once and the results are stored for later use. The algorithm, as stated below, also assumes that the subsegment labels are calculated or read in with the control points that define the surface.

The algorithm requires the following information for evaluating a subpatch:

1) corresponding control point matrix for the subpatch (CP');

2) subpatch labels in s and in t direction (LABS and LABT);

3) orders of the subpatch in s and in t direction (h and k);

4) parametric value (between 0.0 and 1.0) (s and t);

5) the value of rational indicator (RATNOT).

The major steps of the algorithm for evaluating a subpatch are:

Step 1: Use LABS and h to retrieve the proper set of blending coefficient matrix, $M_{h\times h}$. Similarly use LABT and k to retrieve $N_{k\times k}$.

Step 2: Construct the s vector $[\ 1\ s\ \ldots\ s^{h-1}\ ]$;
construct the t vector $[\ 1\ t\ \ldots\ t^{k-1}\ ]$.

Step 3: Evaluate blending functions
$[BF(s)] = [\ 1\ s\ \ldots\ s^{h-1}\ ] \cdot [\ M_{h\times h}\ ]$;
$[BF(t)] = [\ N_{k\times k}\ ] \cdot [\ 1\ t\ \ldots\ t^{k-1}\ ]^T$.

Step 4: Calculate homogeneous points
$WX = [BF(s)] \cdot [CP'(x)] \cdot [BF(t)]$;
$WY = [BF(s)] \cdot [CP'(y)] \cdot [BF(t)]$;
$WZ = [BF(s)] \cdot [CP'(z)] \cdot [BF(t)]$;
and if it is a strictly rational B-spline, then
$W = [BF(s)] \cdot [CP'(w)] \cdot [BF(t)]$.

Step 5: Calculate the three-dimensional point if strictly rational
$X = WX/W$;
$Y = WY/W$;
$Z = WZ/W$.

5.2 The Derivative and Normal Vectors

If the derivatives of the point with respect to the parametric variable are required, the algorithm must be modified. The appropriate derivative of the s vector and t vector are taken and used in step 3 to evaluate an alternative set of blending functions. This set of blending function derivatives is then used in step 4. An example to get the derivative of bicubic patch in both the s and t directions is given below.

$$\frac{\partial P(s,t)}{\partial s} = \begin{bmatrix} 0 & 1 & 2s & 3s^2 \end{bmatrix} \begin{bmatrix} M_{4\times4} \end{bmatrix} \begin{bmatrix} CP'_{4\times4} \end{bmatrix} \begin{bmatrix} N_{4\times4} \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} \tag{5.4}$$

$$\frac{\partial P(s,t)}{\partial t} = \begin{bmatrix} 1 & s & s^2 & s^3 \end{bmatrix} \begin{bmatrix} M_{4\times4} \end{bmatrix} \begin{bmatrix} CP'_{4\times4} \end{bmatrix} \begin{bmatrix} N_{4\times4} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2t \\ 3t^2 \end{bmatrix} \tag{5.5}$$

If the surface is nonrational, then $\partial X/\partial s$, $\partial Y/\partial s$, and $\partial Z/\partial s$ can be obtained by substituting CP'(x), CP'(y), and CP'(z), respectively. The derivatives in the t direction can be derived similarly by applying equation (5.5). If the surface is strictly rational, then the chain rule must be employed to replace step 5 to calculate the derivative. For example, to calculate $\partial X/\partial s$ it would be necessary to use

$$\frac{\partial X}{\partial s} = \frac{\partial (WX/W)}{\partial s} = \frac{1}{W} \cdot \frac{\partial WX}{\partial s} - \frac{WX}{W^2} \cdot \frac{\partial W}{\partial s} \cdot \tag{5.6}$$

The tangent vector to parametric curve $P = P(s,t_o)$, where t_o is a constant, is a multiple of the vector $\partial P/\partial s = (\partial X/\partial s \ \partial Y/\partial s \ \partial Z/\partial s)$. Similarly, the tangent vector to the curve $P = P(s_o, t)$ is a multiple of $\partial P/\partial t = (\partial X/\partial t \ \partial Y/\partial t \ \partial Z/\partial t)$. The tangent plane at the intersection of these curves at $P(s_o, t_o)$ contains these two tangent vectors, so that the normal to the surface is a multiple of their vector product (see Figure 5.1). The unit normal vector n is then given by

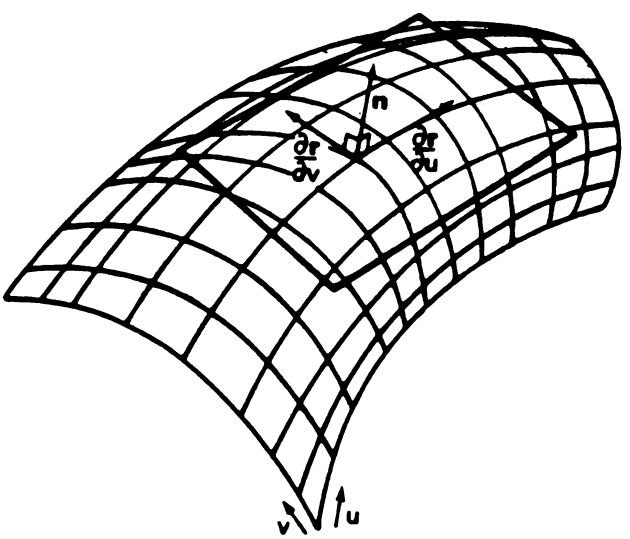


Figure 5.1. Normal vector and tangent vectors of a point on the surface [18].

$$n = \frac{(\partial P/\partial s \times \partial P/\partial t)}{| \partial P/\partial s \times \partial P/\partial t |}$$

$$\text{where } (\partial P/\partial s \times \partial P/\partial t) = \begin{vmatrix} i & j & k \\ \partial X/\partial s & \partial Y/\partial s & \partial Z/\partial s \\ \partial X/\partial t & \partial Y/\partial t & \partial Z/\partial t \end{vmatrix} = ai + bj + ck$$

and $| \partial P/\partial s \times \partial P/\partial t | = (a^2 + b^2 + c^2)^{1/2}$.

5.3 Shading

The process of taking a three-dimensional description of a scene and generating a two-dimensional array (typically 1024x1024) of intensities (pixels) that will be displayed on a CRT is called **rendering**. The description of a scene may be modeled by polygons, parametric patches, or splines. Generally, rendering of a scene also needs information about the (one or more) light sources, such as the intensity, direction, and model of the light sources. Many alternative models for light sources and their interactions with surfaces are in common usage.

Rendering begins by transforming the objects into the eye-coordinate system, or what is called **viewing transformation** (clipping, perspective transformation also included) in which objects outside the field of view are clipped away (see Figure 1.1). The remaining objects must be compared to decide which objects, and/or portions of objects, are visible from the view point. Then the hidden surface removal

problem is taken into account. Three popular hidden surface removing algorithms are the z-buffer, priority, and scan-line algorithms [20].

The next step in visualizing an image, after hidden surfaces have been removed, is to shade the visible surfaces taking into account the light sources, surface characteristics, and the positions and orientations of the surfaces and sources. Next, we examine the interaction of different components of point light sources shining on the surfaces of objects.

The light reflected from a surface can be modeled in a common simplification with two components, diffuse and specular. For diffuse reflection, scattering of light is assumed to be equally in all directions, so that the surfaces appear to have the same brightness from all viewing angles. Ideal specular surface, however, reradiates light in only one direction, the reflected light direction. For real objects, the reflected light contains both diffuse and specular components. For example, illuminate an apple with a bright light; the highlight is caused by specular reflection, while the light reflected from the rest of the apple is caused by diffuse reflection. In order to generate realistic images, both components need to be taken into consideration. In Figure 5.2, E is unit vector that points to the eye while L is unit vector to the light source. E' indicates the reflected eye direction and N is the unit normal vector at the point P. It is very simple to compute the diffuse component. According to Lambert's Law, the diffuse component can be calculated by N•L, the inner product of N and L. The computation of the specular component, however, is much more

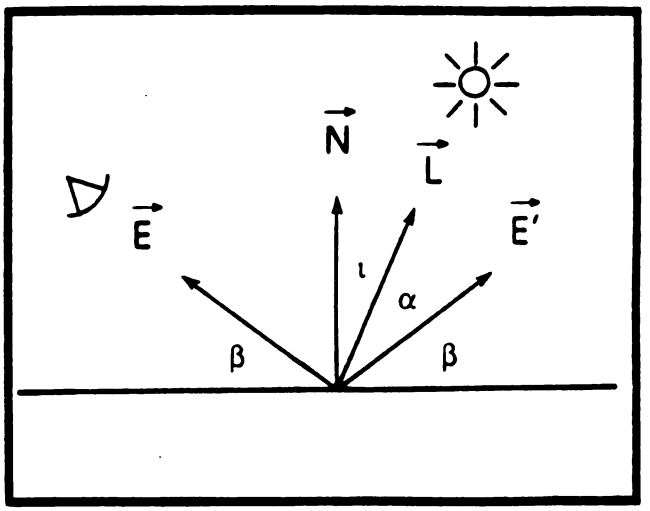


Figure 5.2 The geometry of shading [12]

complicated.

An approximation of the specular component proposed by Phong [21] took the form

$$W(\iota)\cdot\cos^{n}(\alpha) \tag{5.7}$$

where ι is the incident angle and α is the angle between E' and L. $W(\iota)$ is used to model the change of the ratio of incident light to reflected light as the incident angle changes. In practice, according to [20], $W(\iota)$ has been ignored by most implementers. The order of $\cos(\alpha)$, n, is the shininess factor. When L and E' in the same direction, i.e. α=0, $\cos^{n}(\alpha)$ reaches maximum, and it will decrease very fast in the off-axis direction as n increase.

More complex methods to calculate the specular component can be found in [19]. The modeling of a third component, termed ambient light, is given also in [19].

Once we know how to shade a point, we can shade a surface according to the same principle. Rather than going through the extra shading computation for every point to be displayed (i.e., pixel), several approximation methods have been proposed [21,22]. The drawback of these approximation methods is that they do not generally represent the original surface precisely. If high-speed hardware was available to reduce the computation cost of shading for each point, more precise visualizing of a surface would then be possible. Based on the B-splines and the algorithm introduced in previous chapters, a VLSI chip architecture is proposed in the next chapter which performs a crucial part of the work to achieve real-time shading, now a bottleneck of interactive computer graphics.

CHAPTER VI

ARCHITECTURE DESIGN

The architecture of a computational system can be characterized in many ways depending on the designer's view as well as the design level. The architecture here is collectively defined by a specification of the functional capabilities of its physical components, the logical structure of their interconnections, the nature of the information flow, and the control mechanism.

In the section on system specification, the design objectives and criteria as well as special functional configuration requirements will be presented. The functional units and the clocking scheme used in the design are discussed in detail. The data flow is then manipulated in such a way that maximum system throughput is achieved. The architecture which meets the system specification is then presented in block diagram form. The algorithm of surface evaluation is expressed in the form of a program written in RTL (Register Transfer Language).

6.1 System Specification

6.1.1 Design Objective

The objective is to design a basic VLSI chip architecture dedicated to the fixed-point computation of the surface evaluation for a host graphics processor. This chip will work as a coprocessor to the host. It will be activated whenever a surface evaluation request is invoked; otherwise, it remains in a quiescent mode.

The following criteria have been established, based on the application requirement analysis, to guide in making various design decisions.

- CMOS technology with a feature size of 1.5μm or below and a clock of up to 20MHz are to be used.
- Blending coefficient matrices will be stored in ROM and the blending functions will be generated on chip in order to avoid possible I/O bottlenecks.
- In order to simplify the overall design, the surface to be evaluated is assumed to be defined by a bicubic patch.
- Intra-functional unit pipelining is to be used in order to achieve higher system throughput.
- Minimum cycle time and minimum interconnections are sought, with the former being a major concern.

In the design of the SE (Surface Evaluator) chip, the three following rules from Mead and Conway [23], which are adequate to guarantee the correct functioning of a bus-oriented chip design, are

rigidly followed:

Rule 1: The system is driven by a two-phase, nonoverlapped clock; input of phase one is from phase two output and vice-versa.

Rule 2: Functional units are timeless C/L (Combinational Logic); operations on data are separated by the two-phase clock.

Rule 3: A standard bus scheme is used; data transfer through the bus occurs during phase one, and the bus is precharged during phase two.

6.1.2 I/O Specification

The input to the proposed SE chip consists of **LABS** and **LABT**, values of both **s** and **t**, an information bit for indicating rational or nonrational, and the three (four if rational) 4-by-4 control point matrices. The output of the SE chip is the three-dimensional point (x, y, z). If the normal vector at this particular point is desired, further calculations are needed. All of the information, except for the rational indication bit, are 32-bit, two's complement, fixed-point representations. Necessary pre-scaling and post-scaling of the data are assumed to be done by the host processor.

Communication of the control point matrices is the bottleneck of the SE chip. Efficient pipelining of the input stage and the functional units of the SE chip in order to obtain the highest throughput, is, therefore, the major design endeavor.

6.2 Architecture Development

6.2.1 Functional Units

The elementary arithmetic operations needed for the surface evaluation calculation are multiplication, addition, and division. The schematic block diagram of a Baugh-Wooley two's complement array multiplier [24] is shown in Figure 6.1. It is the basic structure of a multiply-and-add cell (MAC).

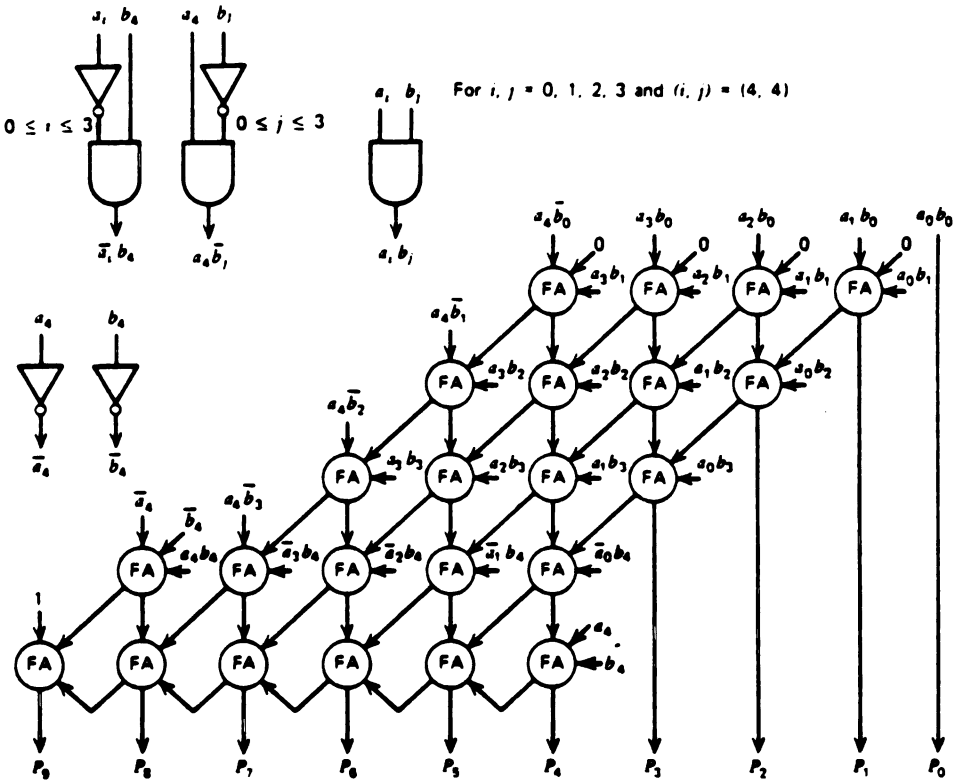
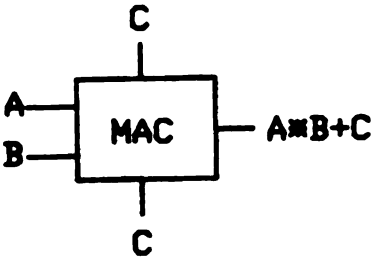


Figure 6.1. The block diagram of a 5-by-5 Baugh-Wooley two's complement array multiplier [24].

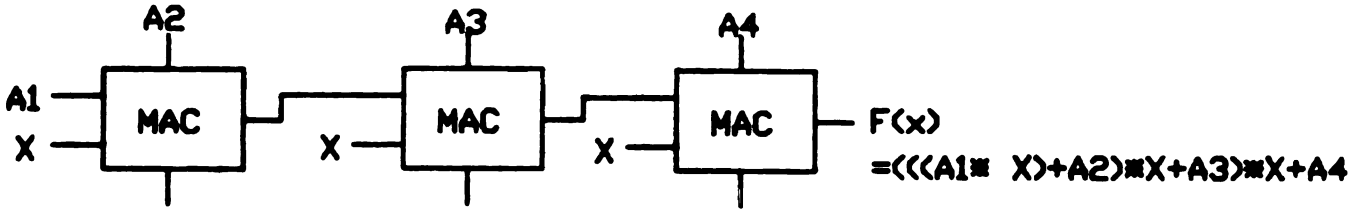
The first step of the surface evaluation calculation is to form the blending functions of s and of t. If we compute the blending function of s, for example, by the multiplication of vector [1 s s^2 s^3] and the corresponding blending coefficient matrix chosen according to LABS, we need to calculate the square and the cube of s beforehand. This requires at least two more cycles to perform. However, there is an alternative way of forming the blending functions, which does not require computing the square and the cube in advance.

If the MAC, represented in block form shown in Figure 6.2(a), is used, then the polynomial of a variable x with order equal to 3 can be obtained by using three MAC's as shown in Figure 6.2(b). Calculation of the first derivative of a polynomial of x with order = 3 can then be performed by the block diagram shown in Figure 6.2(c). Based on this approach, Figure 6.3 gives the data flow and the structure to calculate the blending function of s as well as its first derivative. The blending function of t and its first derivative can be obtained similarly. Note that in Figure 6.3 there is a need to multiply by 2 and 3 (*2 and *3) in the forming of the first derivative. Actually, no multiplication is necessary here. Multiplication by 2 can be achieved by just shifting the input data one bit left and multiplication by 3 can be obtained by adding three duplicates of the operand. The reason for adding the 4-stage delay is that piplined MAC will has 4 segments as will be covered shortly.

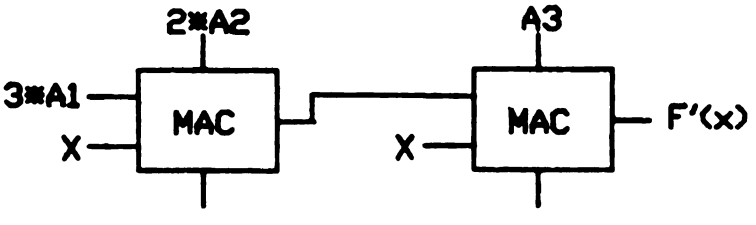
In the evaluation process, many matrix-vector multiplications are involved. After multiplying elements of the vector [1 s s^2 s^3] with



(a)



(b)



(c)

Figure 6.2. (a) Basic MAC block. (b) MAC array structure to compute a polynomial of order 3. (c) MAC array structure to find the first derivative of a polynomial with order = 3.

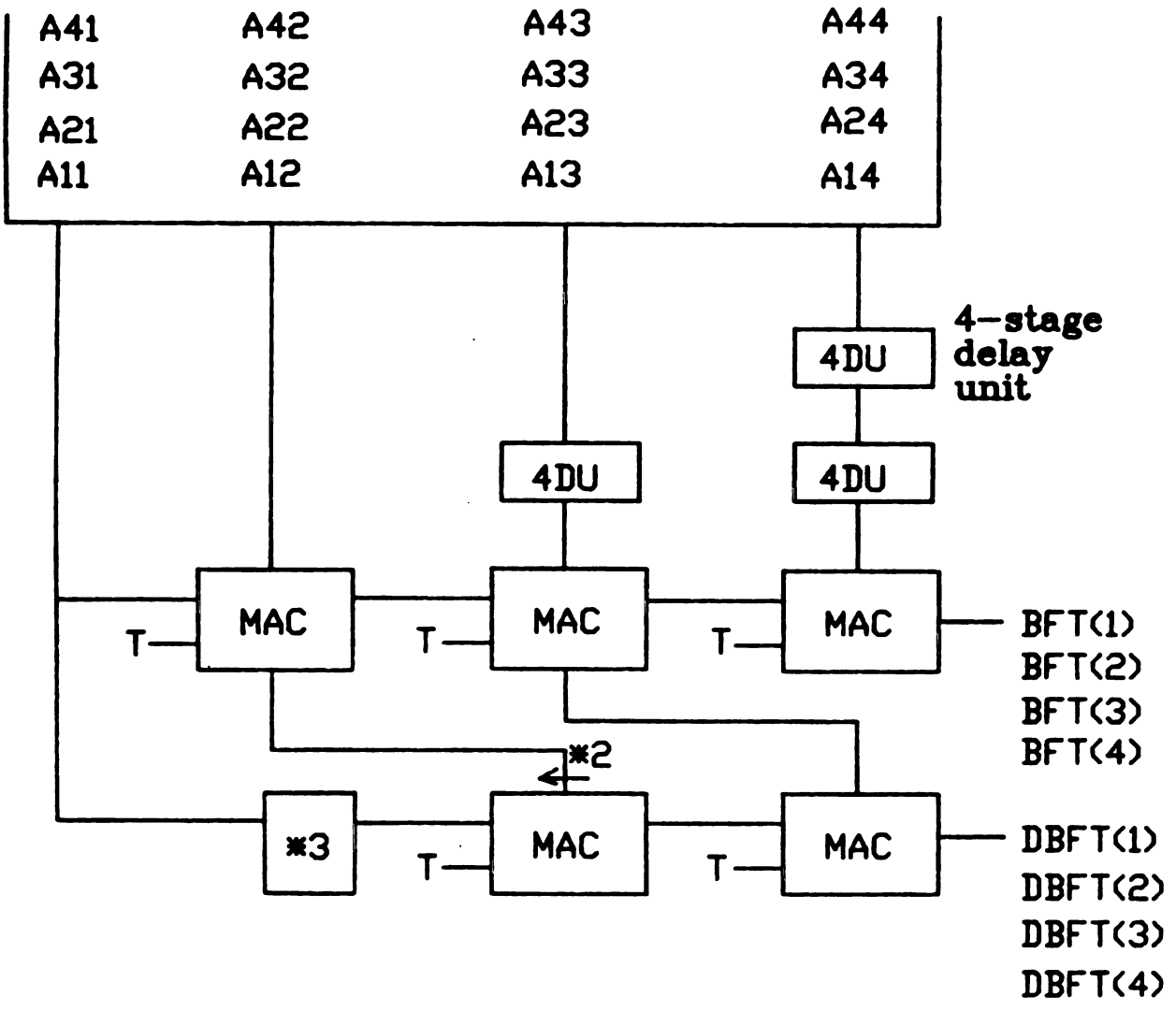


Figure 6.3. The block diagram for calculating the blending functions for evaluating the point and the first derivative.

the corresponding elements of a row (or a column) of the blending coefficient matrix, we need to sum up the products to get the final result. The MAC can be used to achieve matrix-vector multiplication as given in Figure 6.4. This requires four MAC cycles to get the final result; a time cost that is too high. However, we may perform the multiplications concurrently and then add the products in one step. In so doing, we need a special multi-operand adder. A four-operand adder formed by carry-save adder (CSA) trees is given in Figure 6.5. Note that the "←" on the carry output lines indicates that the carries are shifted left one bit with a zero entering from the right end before they are fed into the CSA inputs at the next level. The partial carry and partial sum from the last stage of the CSA is then feed into a carry lookahead adder. The functional block diagram of a full carry lookahead adder is shown in Fig. 6.6 and the details of each unit are given in Figures 6.7 and 6.8.

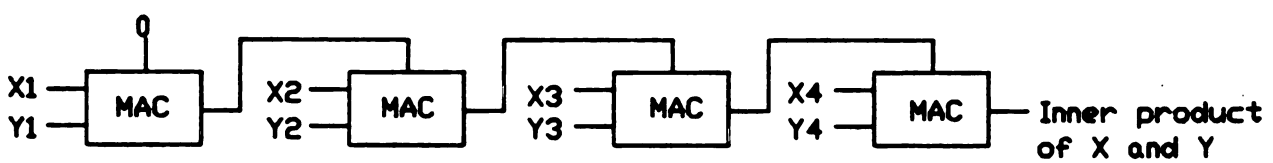


Figure 6.4. Block diagram for calculation of the inner product of X and Y.

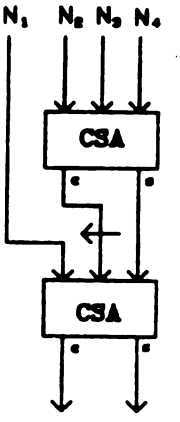


Figure 6.5. A four-operand adder formed by CSA tree.

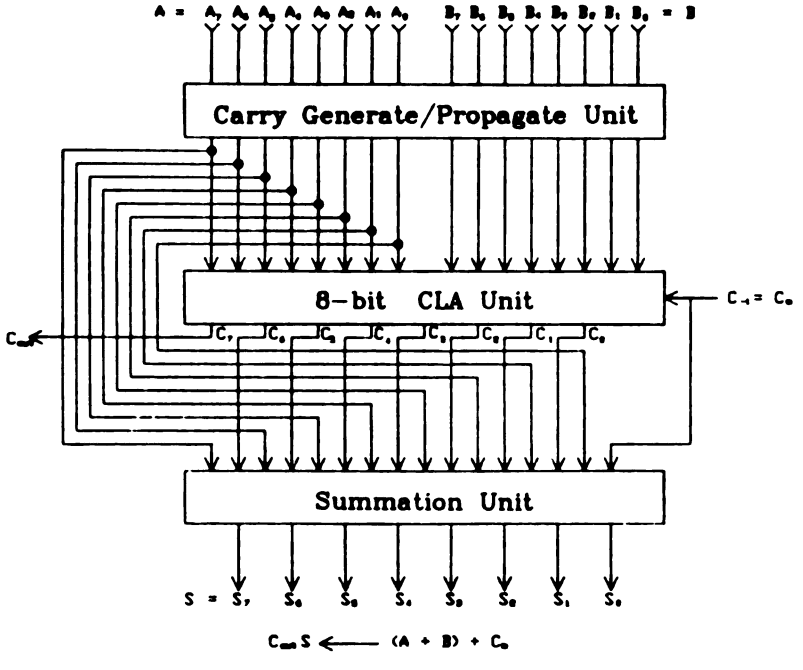


Figure 6.6. The functional block diagram of a full carry lookahead adder.

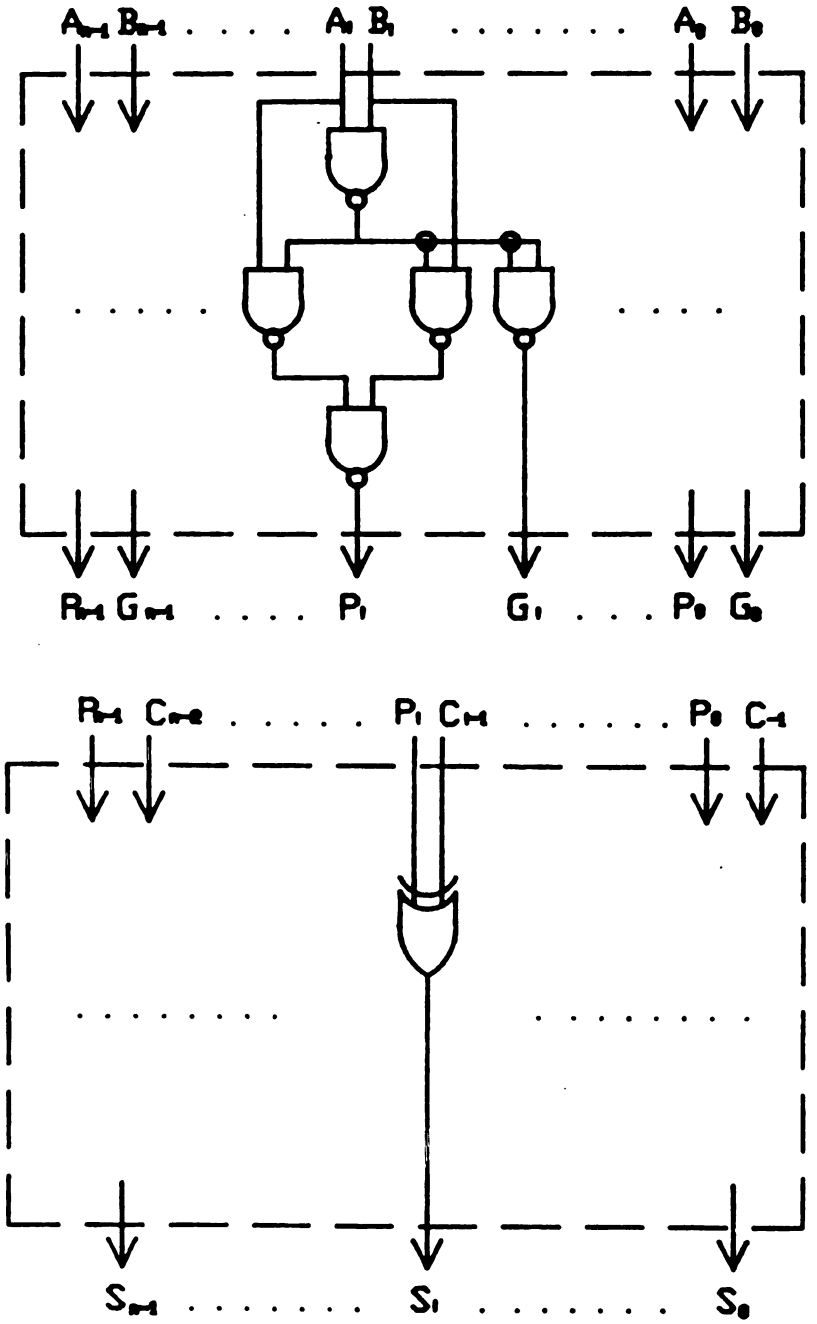


Figure 6.7. Logic circuits for realizing the carry-generate, carry-propagate, and sum functions.

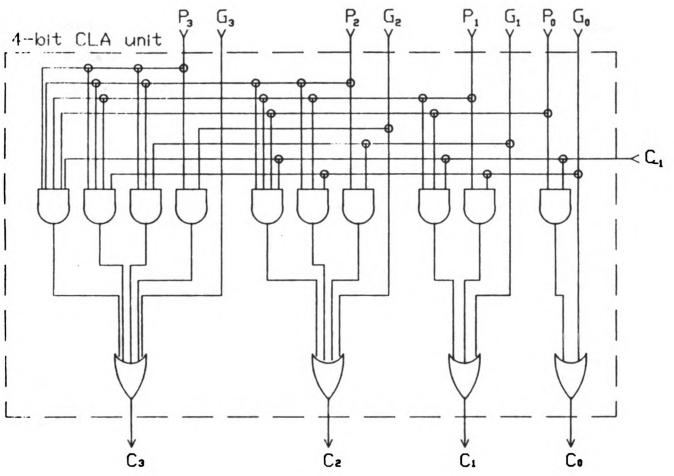


Figure 6.8. The schematic logic circuit of a 4-bit carry lookahead (CLA) unit.

The last requried functional unit is a divider. There are three types of cellular array dividers: restoring division, nonrestoring division, and nonrestoring with carry lookahead array dividers. The area cost of the CLA dividers is higher than the other two array dividers, but on the average it is 5 times faster [25]. Since in the SE chip speed is the major concern, the nonrestoring CLA array divider is

chosen for the calculation of division in the evaluation of a rational surface. Figure 6.9 shows the diagram of a carry lookahead array divider and Figure 6.10 gives the three basic types of logic cells used in Figure 6.9 [25].

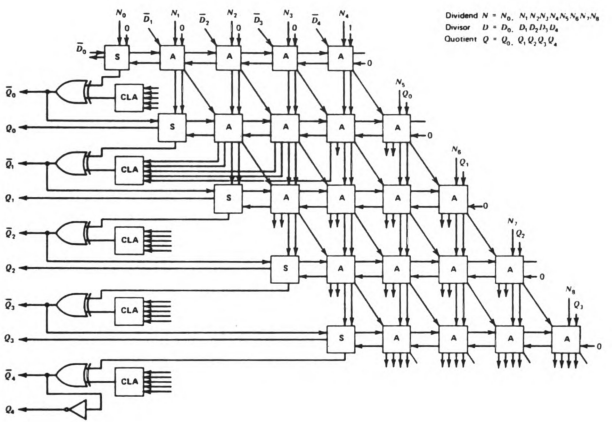


Figure 6.9. A carry lookahead array divider with 8-bit divisor in two's complement representation [25].

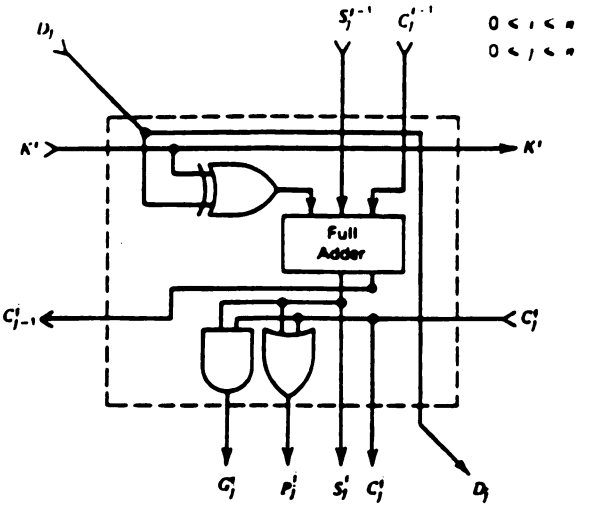


Figure 6.10. Three basic types of cells to be used in constructing the division array of Figure 6.9. (a) The A cell located at the jth digit position of the ith row; (b) the S cell located at sign position of the ith row; and (c) the CLA cell for the ith row [25].

6.2.2 Timing and Pipelining

In the synchronous discipline of design, tasks are accomplished through a logical sequence of events with each event realized under the constraint of the physical timing of transporting information (electrical signals) from one point to another. This dual meaning of timing is bound by a system-wide clock which serves both as the sequence reference and as the time reference. Hence the design of the clocking scheme has paramount important to the system's correct functioning and performance [26].

The general form of a data path and the corresponding two-phase clock scheme are illustrated in Figure 6.11 and 6.12.

Since the evaluation process involves repeated vector-matrix multiplications, it is of great advantage to have a multiplier and an adder form a computational pipeline. This is because the piped operation will save bus bandwidth and increase the overall throughput as well as eliminate the need to store temporary results.

From the timing diagram of the two-phase clocking scheme, it is clear that the minimum clock period is the sum of the maximum C/L delays of phase-one and phase-two plus some small propagation overhead and preset time. Intuitively, the C/L delay due to the multiplier may constitute the largest delay component in the minimum clock period calculation and thus one may further be motivated to introduce additional "subpipelining" into the multiplier to shorten the clock period.

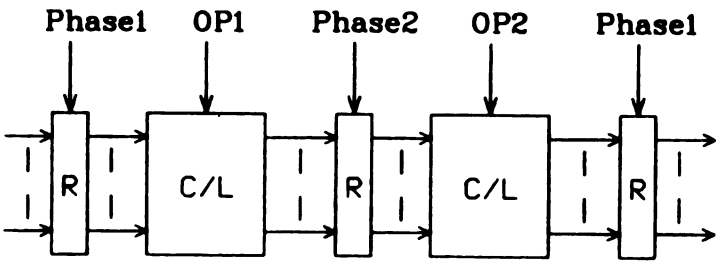


Figure 6.11. The general form of a data path [23].

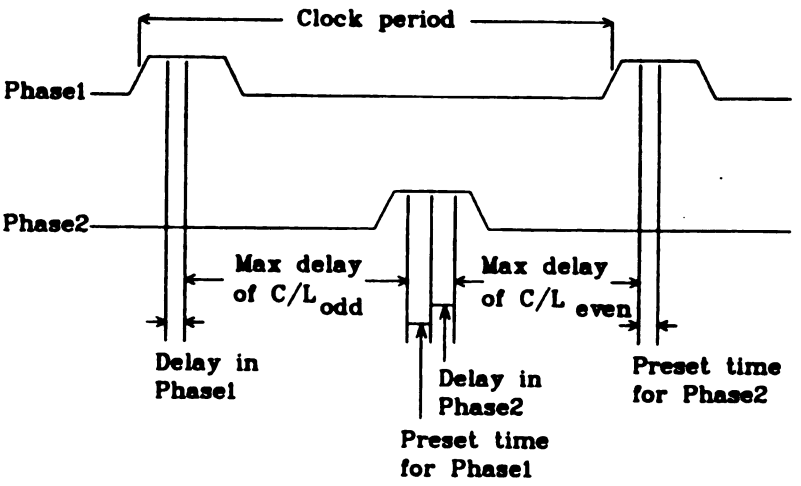


Figure 6.12. The two-phase clocking scheme [26].

A pipelined four-segemnt multiplier is given in Figure 6.13 with its clocking scheme [26]. Note that the clocking scheme used here is slightly different from the general single phase latches as shown in

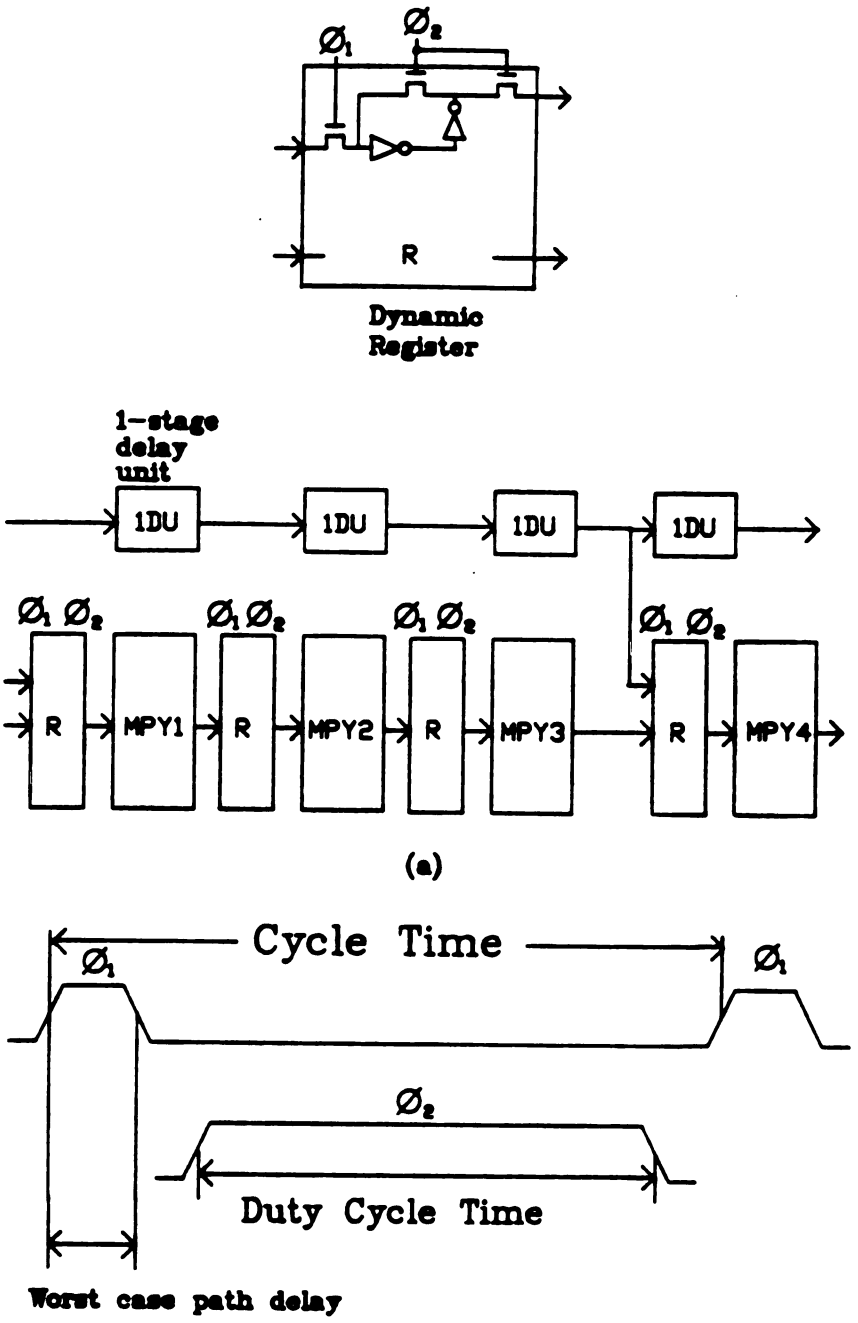


Figure 6.13. (a) Pipelined four-segment multiplier and (b) its clock scheme.

Figure 6.12. When using the clocking scheme in Figure 6.12, each C/L section is actually activated during only half of a clock cycle. However, the clocking scheme in Figure 6.13 allows the C/L sections to operate in a single phase and thus allows a shorter clock period; the price is a slight increase in chip area because of the use of dynamic registers.

The advantage of pipelined multiplication of the blending functions and control point matrix is in more efficient computation. The data flow and the interconnection between adder and multiplier are illustrated in Figure 6.14. The adder and four four-segment multipliers form a whole pipeline for calculation of the inner product.

Note that the above discussion of pipelining doesn't include the divider. It should be segmented into 10 segments for the sake of synchronizing with the system clock.

6.2.3 Storage Scheme

The SE chip architecture takes advantage of parallelism and pipelining, but two key problems have not yet been discussed: I/O and the data storage scheme. In the case of evaluation of bicubic rational surfaces, four 4-by-4 control point matrices need to be fed into the SE chip. Because of the constraint on the number of I/O pins, it is impossible to input the data in a few cycles. But one thing should be noted: as long as the values of s and t and the labels are given, the

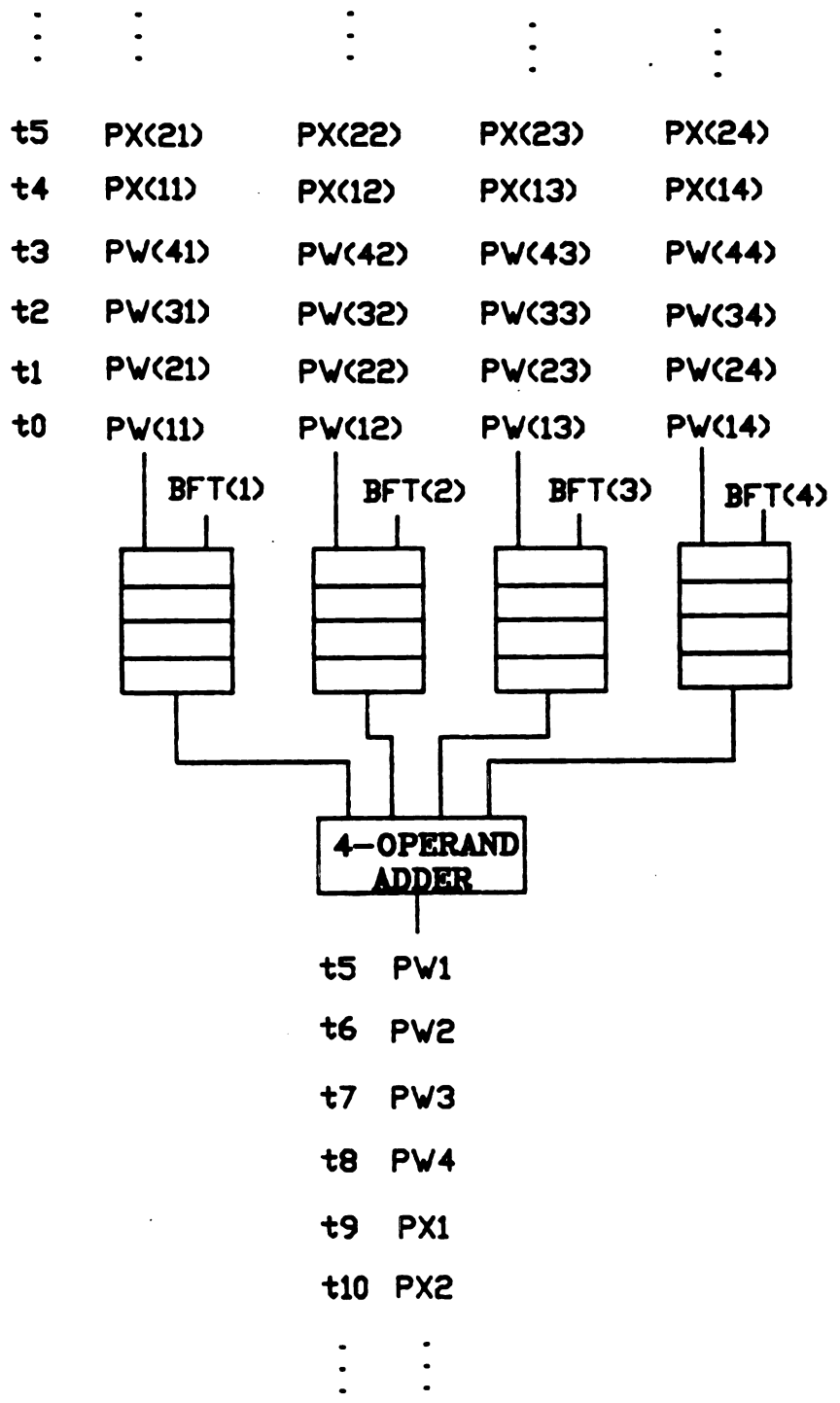


Figure 6.14. Dataflow of vector-matrix multiplication.

formation of blending functions can be overlapped with the input of control point matrices. By so doing, the I/O bottleneck can be reduced if not eliminated.

The question remaining is how to store the data in order to support the pipelined process. One straightforward way is to store the control point matrices in registers. The area cost of using registers is quite high and the control of that many registers is complicated. An alternative is to store the data in RAM (Random Access Memory), which is easier to control and occupies much less chip area. In the SE design, the latter is used to store the control point matrices.

As stated in the last section, the inner product multiplications are executed concurrently. In order to provide the necessary data for parallel multiplications, the RAM is divided into four modules and each module stores one fourth of the three (four, if rational) control point matrices. But row accessing as well as column accessing are necessary for the matrix-vector multiplication. In order to access either a row or a column of the control point matrices in one cycle, a special skewed storage scheme is used.

Let S (stride or skewing distance) be the distance measured in columns that each array row has been shifted relative to the row above it. Let M be the total number of memory modules, $m(a_{ij})$ be the memory module number of element a_{ij}, and $q(a_{ij})$ be the local address within that module. Then the storage scheme can be represented by the following equations.

$$m(a_{ij}) = (i \cdot S + j) \bmod M \tag{6.1}$$

$$q(a_{ij}) = i \tag{6.2}$$

The storage pattern used in the SE chip is a skewed storage scheme with S=1 and M=4; an example is given in Figure 6.15. A multiplexing

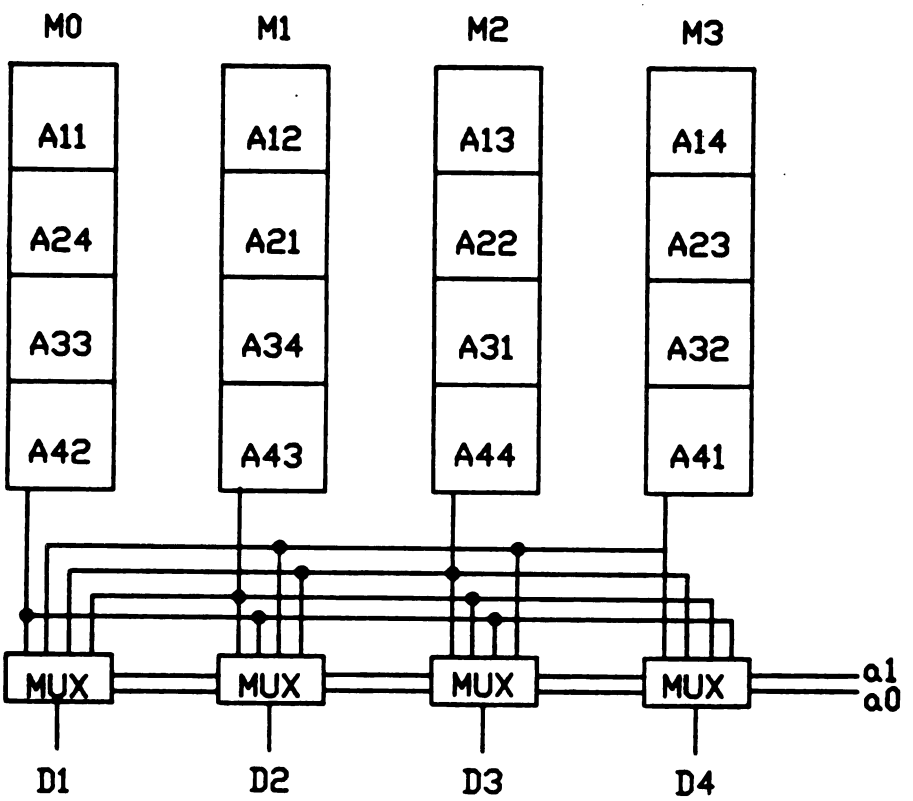
A_{11}	A_{12}	A_{13}	A_{14}
A_{21}	A_{22}	A_{23}	A_{24}
A_{31}	A_{32}	A_{33}	A_{34}
A_{41}	A_{42}	A_{43}	A_{44}

(a)

A_{11}	A_{12}	A_{13}	A_{14}
A_{24}	A_{21}	A_{22}	A_{23}
A_{33}	A_{34}	A_{31}	A_{32}
A_{42}	A_{43}	A_{44}	A_{41}

(b)

Figure 6.15. Skewed storage pattern with (a) S=0, and (b) S=1.



INPUT		OUTPUT			
a1	a0	D1	D2	D3	D4
0	0	A11	A12	A13	A14
0	1	A21	A22	A23	A24
1	0	A31	A32	A33	A34
1	1	A41	A42	A43	A44

Figure 6.16. Skewed storage with output multiplexing.

output circuit of the skewed storage module is shown in Figure 6.16. The table in the figure gives an example of row accessing. If column accessing is desired, proper addressing must be applied to each module in order to get the appropriate elements.

6.3 Architecture of SE Chip

6.3.1 Architecture Synthesis

After the presentation of the details of the functional units, clock scheme, and storage scheme, a simplified block diagram of the SE chip can now be drawn as shown in Figure 6.17. When the SE chip is activated by the host processor, one of the two main routines in the control memory will become active according to the rational/nonrational indication bit received from the system control input. One routine is in charge of rational surface evaluation and the other takes care of the nonrational case.

The first four input items received by the I/O circuit are the values of s, t, LABS, and LABT, which will be stored in the register file. As long as these four items are known, the formation of blending functions of s and t can be performed in the arithmetic unit by retrieving the proper blending coefficient matrices according to LABS and LABT. At the same time, the I/O circuit will store the incoming control point matrices in RAM based on the skewed storage scheme. The register keeps the blending function vectors, the intermediate output of

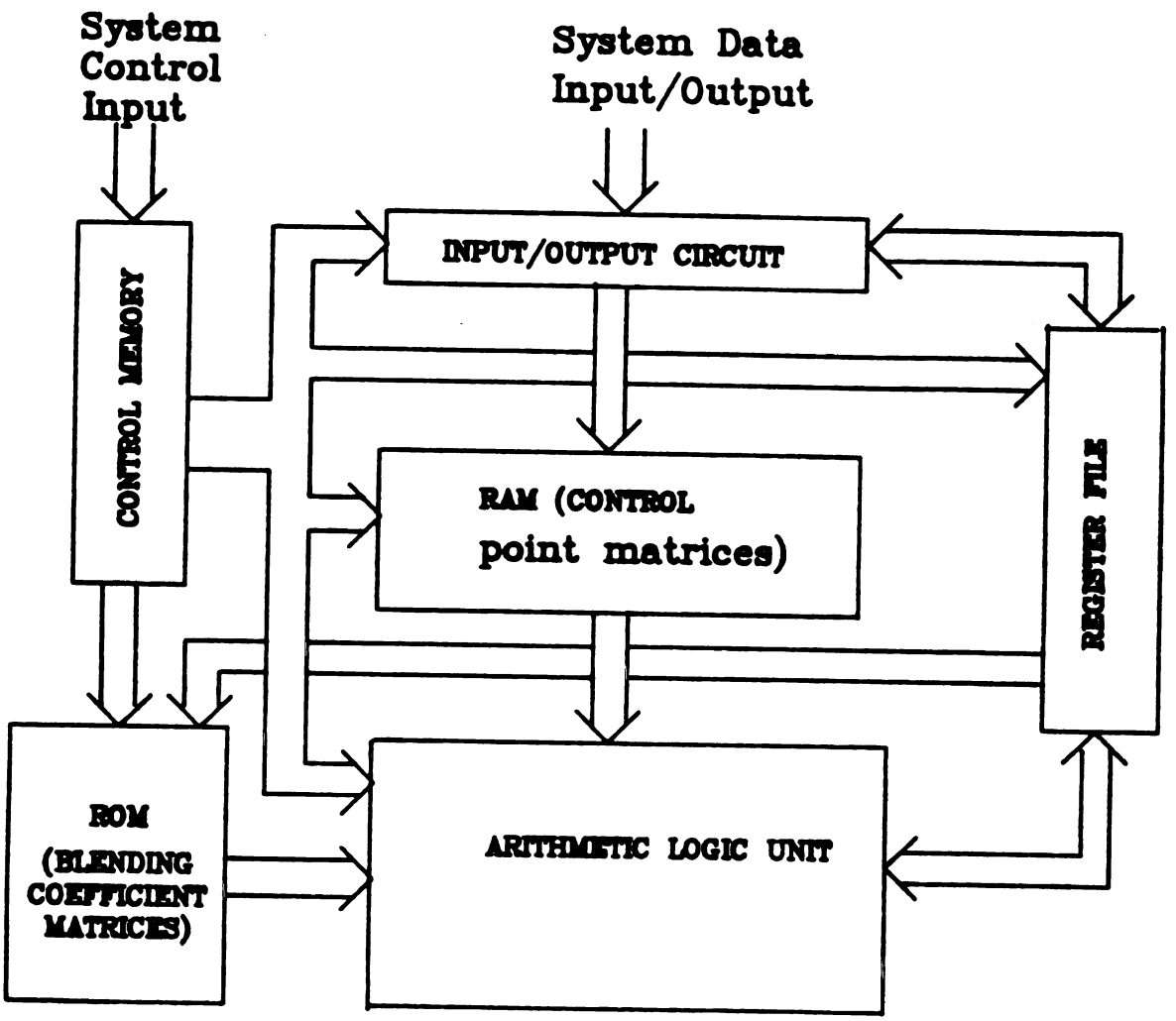


Fig. 6.17. Simplified diagram of SE chip.

the matrix-vector multiplication, and the final result. All calculations, including formation of blending function, matrix-vector multiplication, and division, are performed in the arithmetic unit. According to the functional dependency, the arithmetic unit can be subdivided into two parts: division unit and multiplying-and-adding (M&A) unit. The data flow of the division unit is trivial. It receives data WX, WY, WZ, and W (if rational) from the register file and then sends the division results X, Y, and Z back to the register file. Data flow and interconnection of the M&A unit, however, is much more complicated.

The design of the M&A unit must enable calculation of both blending functions and matrix-vector multiplication with a minimum number of functional units. To meet this requirement, the circuit diagram shown in Figure 6.18 is developed. The interconnection of the M&A unit is determined by only one control signal, MODE. When MODE is low, the M&A unit acts as the circuit showed in Figure 6.3. All the 0*th* input bits of the multiplexers will be selected, thus the blending functions as well as the first derivatives of the blending functions, can be computed. If MODE becomes high, the M&A unit acts as four parallel multipliers with a four-operand adder to sum up the inner-product, as showed in Figure 6.14.

For simplicity, Figure 6.18 shows only the interconnection of the M&A unit with MODE=0. When MODE=1, one of the inputs of the four active MAC's is from RAM and the others are from the register file. Note also that the multiplexing is now implicitly embedded in the four-operand

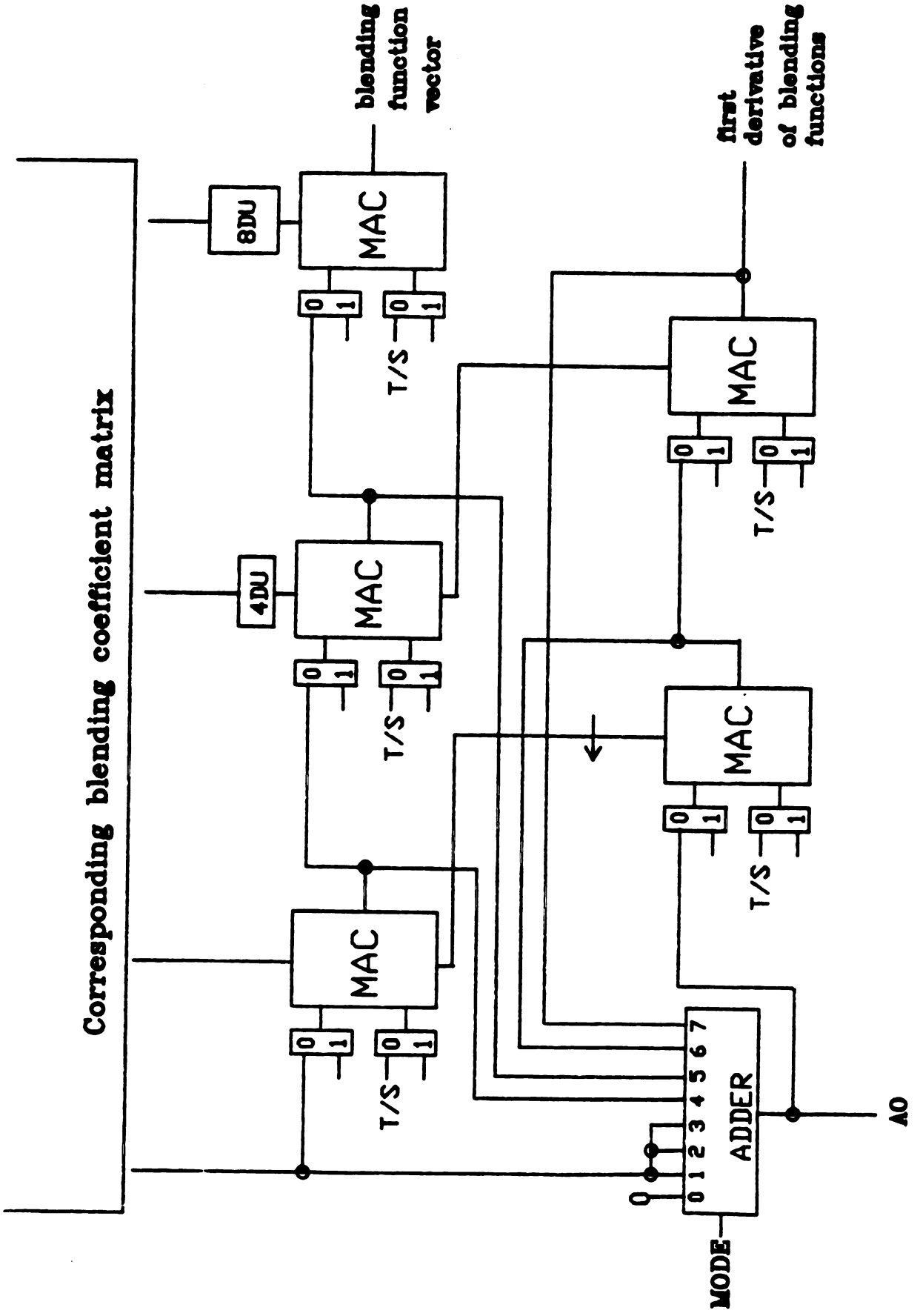


Figure 6.18. Multiplying-and-adding (M&A) unit.

adder. As MODE=0, the addition is performed on the first four inputs (pins 0, 1, 2, and 3) of the adder, and while MODE=1, the addition is performed on the other four inputs (pins 4, 5, 6, and 7).

6.3.2 Hardware Description in RTL

The last section covered only the structural specification of the SE chip. The functional description is yet to be given. A program written in RTL (Register Transfer Language) and given in the Appendix 1 is used to functionaly describe and verify the SE chip. The RTL program specifies the data flow among the functional units and storage units. Table 6.1 gives the cycle count for different evaluation routines based on the RTL program. The patch evaluation in the table means the evaluation of multiple points on the same patch. The startup time is the time mainly to input the control point matrices whereas the average

Table 6.1. Cycle count for different evaluation routines

Routine	Startup Delay (cycles)	Average Cycles
nonrational point evaluation	52	79
nonrational patch evaluation	52	49
rational point evaluation	68	110
rational patch evaluation	68	64

cycle time is the average time to evaluate multiple points on the patch except for the first evaluated point. The average cycle time for single point evaluation is longer than the average time for patch evaluation (or multiple point evaluation), because for single point evaluation, it includes startup delay for each evaluation routine, but for the patch evaluation, only the evaluation of the first point involves the startup delay. But one should note that if the multiple points to be evaluated were defined by two separate effective knot vectors, then you have to pay the startup delay twice. In other words, as long as the point to be evaluated across the patch boundary defined by the effective knot vector, you have to repay the price for startup time delay. The drawback of the RTL program is that, because of its simplicity, some details of the architecture can not be described thoroughly, such as the skewed storage scheme.

CHAPTER VII

ARCHITECTURE EVALUATION

Speed and area costs are the two most prevailing parameters for evaluating the "goodness" of a hardware design. Without physical design and simulation, however, it is impossible to get the precise values of these parameters. Even worse, different technologies, such as NMOS or CMOS, and different physical design approaches, like gate array, macrocell, or full-custom design, result in different time/area evaluations. Without at least a qualitative estimation of the time/area complexity of the chip, however, a decision of whether or not to carry on the physical design work is difficult to make.

In this research, no attempt is made to acquire accurate speed and area costs of the SE chip. Yet, some basic statistics of the chip are provided in order to show the practicality of implementation of the chip under current technology. This chapter starts with the architecture verification of the SE chip. The area estimation is then given in detail followed by the speed estimation.

7.1 Architecture Verification

To better understand the evaluation algorithm in the SE chip, a flowchart is given in Figure 7.1. There are two main routines, one for the evaluation of nonrational spline points and the other for the evaluation of rational spline points. When the SE chip is activated according to the value of RAT, it takes four cycles to input the parameters, s and t, and their labels, LABS and LABT. As long as the parameters and their labels are known, the formulation of the blending functions and their first derivatives can be performed in parallel with the input of the control point matrices. Then the Cartesian coordinates of the point are evaluated and sent back to the host. If a rational spline point is to be evaluated, three more divisions are necessary to obtain the desired coordinates. If a patch is to be evaluated, then the parameter values for the evaluation of next patch point will be given followed by the formulation of blending functions and their first derivatives, otherwise stop. Note that when evaluating points of a patch, the control point matrices need to be brought in only once.

A symbolic example is given in the Appendices to illustrate the dataflow of the evaluation of a nonrational point. Starting from top of the flowchart in Figure 7.1, the input of s, t, LABS, and LABT are trivial and thus not shown in the example. Appendix 2 gives the input sequences of control point matrices and Xij, Yij, and Zij (the Cartesian coordinates of a control point). Appendix 3 shows the content of each

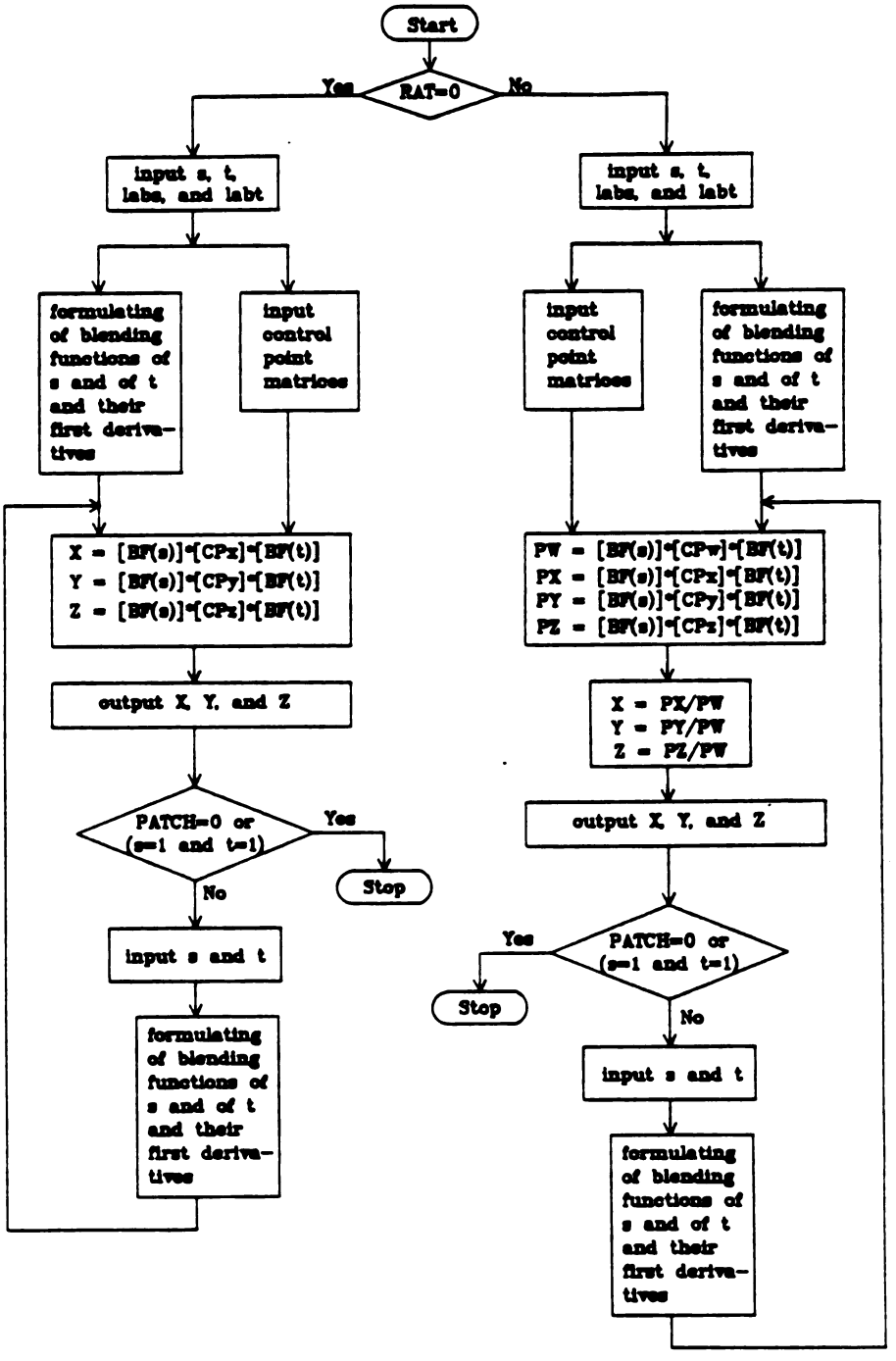


Figure 7.1. Flowchart of the evaluation algorithm in the SE chip.

latch element in every cycle for the formulation of blending functions and their first derivatives. Appendix 4 gives the result in each functional stage. S*ij* and T*ij* are the blending coefficient matrices retrieved from ROM according to LABS and LABT, respectively. BFS(i) and BFT(i) are the blending functions and DBFS(i) and DBFT(i) are their first derivatives. All the intermediate output are shown in parenthesises. For example, (M2_3) is the intermediate output in the third stage of the second MAC unit.

Similarly, Appendix 5 shows the contents of each element for the evaluation of the Cartesian coordinates and Appendix 6 gives the result in each functional stage. The IT(*i*,*j*) in the Appendices are used to store the result of the inner product of BFS and the control point matrices.

It is noted that in Appendices 3 and 5, the contents of the latches are strobed in at the beginning of each cycle and held throughout the cycle whereas in Appendices 4 and 6, the result of each functional stage is valid at the end of the cycle. If a patch evaluation is desired, the same time table in Appendices 3 and 4 can be used to formulate the blending functions and their first derivatives of next point to be evaluated. The principle of this example can be easily applied to the evaluation of rational spline points by considering one more control point matrix and three more divisions.

7.2 Area Estimation

The real size of a VLSI chip depends on many physical design factors such as transistor layout, placement of building blocks, and routing (wiring). Optimal design is very hard to obtain in any aspect and even a close-to-optimal solutions cost the designer much effort to achieve.

Without actual layout, placement, and routing of the chip, we are short of knowledge of all physical design factors about the chip. To overcome this dilemma, this research used the physical device count, or transistor count more precisely, as the major parameter in order to get a realistic picture of the area required for the SE chip.

The transistor counts for various function units and components in the SE chip are listed in Table 7.1. All transistor counts are based on CMOS technology.

The central part of the MAC is a Baugh-Wooley array multiplier which requires n(n-1)+3 full adders, n^2 2-input AND gates, and 2n inverters for the multiplication of two n-bit inputs, where n is the bandwidth of the data [24]. In terms of physical layout, the full adder, 2-input AND gate, and inverter take 28, 6, and 2 transistors, respectively [29]. Thus the total transistor count for a n-by-n Baugh-Wooley multiplier is

$$\begin{aligned} N_T(BW) &= [n\cdot(n-1) + 3]\cdot 28 + n^2\cdot 6 + 2n\cdot 2 \\ &= 34n^2 - 24n + 84 \end{aligned} \tag{7.1}$$

Table 7.1. Transistor count of the SE chip.

Function	cell type	No. of Transistors per units	No. of Units	Total No. of Transistors
MAC	Dynamic Reg	10	32x40	12,800
	Delay stage	8	32x17	4,352
	Array Multiplier	34,132	5	170,660
	2-1 MUX	4	32x10	1,280
	Ripple-carry Add	28	5x48	6,720
			subtotal	195,812
Divider	Dynamic Reg	12	32x30	11,520
	Array Divider	72,708	1	72,708
			subtotal	84,228
Adder	Dynamic Reg	12	64x4	3,072
	2-1 MUX	4	64x4	1,024
	CSA Tree	24	64x2	3,072
	CLA(16x4)	4,100	1	4,100
			subtotal	11,268
RAM	RAM cell	6	32x64+64x32	24,576
	Decoder	4	20	80
	Driving Buf	4	16x42+4x42 +2x42+2x42	4,032
	4-1 MUX	12	32x4	1,536
			subtotal	30,224
ROM	ROM cell	1	32x16x16	8,192
	Decoder	4	63	252
	Driving Buf	4	64x(2+8+32)	10,752
			subtotal	19,196
Driver	I/O Buf	12	32x2	768
Delay unit	2-phase latch	8	32x15	3,840
Register	2-phase latch	14	32x2+64x4 +4x2+1+1	3,300
			Grand Total	348,636

The transistor count of an array divider is not available, but the gate count of a nonrestoring with 1-level CLA and carry-save divider is shown in [30]. A unit gate assumed in [24] is a 2-input NAND gate which requires 4 transistors when implemented in CMOS technology. So the transistor count of a nonrestoring array divider (AD) can be roughly estimated by

$$\begin{aligned} N_T(AD) &= (18n^2 + 28n + 11)\cdot 4 \\ &= 72n^2 + 112n + 44, \end{aligned} \tag{7.2}$$

where n is 15 in this case instead of 16, because the sign bit does not contribute.

One might dispute the accuracy of this estimation. It is true that the number of transistors will increase dramatically if there are many gates with more than two input pins. On the other hand, hardware implementation at the transistor level usually takes only 1/3 or 1/4 the area when compared to implementation at the gate level [31]. These two factors may balance each other out somewhat and thus make the above estimation still reasonable.

The rest of the work on counting the transistors is based on CMOS circuits taken from [29]. In counting the transistors of the CLA, a two-level CLA(8×4) is assumed, which takes one 8-bit CLA and eight 4-bit BCLA (Block Carry Lookahead Adder) units.

Another issue to note is the signal driving capability. No signal can drive an infinite number of loads, and thus certain fan-out has to be defined in order to guarantee the validity of the driving signal. In

the SE chip, parallel retrieval of data from RAM or ROM requires one decoded signal to drive 64 storage cells simultaneously, but this is impossible under current technology. A reasonable solution is to use a cascaded driving buffers. If we assume conservatively that one driving buffer can drive four successive buffers, then three buffer stages are good enough to drive the 64 cells. This is used to calculate the number of transistors in the driving buffers for the RAM and ROM. Also note that the I/O buffer requires larger area in order to drive a large external load, so the I/O driving buffer is usually counted separately.

The transistor count of the control logic remains unknown until the actual logic design is carried out. One may implement the control circuit by ROM or PLA (Programmable Logic Array), whichever is more convenient. With the help of the program written in RTL, this can be easily achieved.

Under current VLSI technology, more than several million devices have been successively fabricated in a single chip [33]. For a gate array, it is quite normal to have more than 100,000 gates per chip and each gate with 4 transistors [34]. According to these criteria and the grand transistor count, it is possible that the SE architecture could be fabricated on a single chip.

In this research, I counted the transistors of the SE chip first based on 16-bit data bandwidth and obtained nearly 120,000 transistors for the grand total count. When we increased the data bandwidth up to 32-bit, the overall increase is less than 3 times, even though the increases of the array multipliers and array divider are 4 times. The

same principle would apply to a 64-bit data bandwidth; it is likely to get a grand total count of nearly 1,000,000 transistors. In this case, an SE chip set, instead of a single SE chip, will be the choice to fulfill the proposed architecture. Also noted that replacing the array divider with a reciprocal generating unit for the generation of the reciprocal of a divisor may somewhat reduce the space cost for fulfilling the division operations.

7.3 Speed Estimation

The SE architecure uses a two-phase nonoverlapped clock scheme as mentioned previously. Thus, the minimum clock period is the sum of the duty clock widths of phase one and phase two. The phase one duty clock width, according to the specified clocking scheme, is the worst case path propagation delay. Consider the transfer of data from the register to the input latch of the functional units in Figure 6.18 as the worst case path. The delay components include the register response time, bus propagation delay, and the input latch response time. Following the estimation process in [26], the phase one delay would be 12ns if a 0.6ns gate delay is assumed [34].

The phase two clock width is the maximum combinational logic delay. In the SE chip, one stage of the multiplier pipe or divider pipe can be considered as the maximum combinational logic delay. Kawahito et al. reported on the design and fabrication of a high-speed 32x32-bit binary multiplier using 2-μm CMOS technology with a multiply time of 59ns [38].

A CMOS 32×32-bit Wallace tree multiplier with a multiply time of 56ns has been also reported [39]. Since in the SE chip the multiplier has been segmented into four stages and the delay time for the four stages are about the same, the phase two clock is set to 15ns, which is nearly one fourth of the multiply time.

Summing the delay times of phase one and phase two, the clock period of the SE chip is 27ns. Table 7.2 gives the computation time for different evaluation routines. Compare with the transform engine of the HP 320SRX graphics workstation, which adopts 32-bit IEEE floating point data format and transforms 180,000 coordinates per second, the SE chip is 12.6 times faster for nonrational patch evaluation and 9.6 times faster for the rational patch evaluation (noted that three coordinates per evaluation in the SE architecture). But reader should note that the SE chip adopted 32-bit fix-point operation whereas the HP 320SRX used 32-bit floating point operation. Also it is not clear whether the performance of the HP 320SRX to transform 180,000 coordinates per second is recorded for coordinate transformation (such as rotation or shifting) or B-spline point evaluation or both. If it is for coordinate transformation, the only operations are matrix-vector multiplications. When the matrix-vector multiplication implemented in the SE chip, based on the above criteria, 216ns is enough to calculate the multiplication of 4-by-4 matrix and 4-by-1 vector. In other words, 4.63 millions of such matrix-vector multiplications per second is achieveable by using the SE architecture.

Table 7.2. Evaluation time for different evaluation routines

Routine	Startup Delay (ns)	Average Evaluation Time (ns)	Number of Evaluations per second
nonrational point evaluation	1,404	2,133	468,823
nonrational patch evaluation	1,404	1,323	755,857
rational point evaluation	1,836	2,970	336,700
rational patch evaluation	1,836	1,728	578,703

CHAPTER VIII

CONCLUSION

The goal of this research is to develop a dedicated VLSI chip for surface evaluation, and the proposed SE chip architecture satisfies the major design objectives. The architecture of the SE chip may not be optimal, which is not our goal, however much effort has been devoted in order to achieve higher system performance.

Compared to the hardware given in [10], the main advantage of the SE chip is speed. This is mainly due to the introduction of pipelining of functional units. Other features, such as parallel data retrieving, and the overlapping of the input of control point matrices and the formulating of the blending functions, also guarantee the high throughput for the system. One feature not found in any other known hardware design is the on-chip formulation of the blending function. It not only saves time in calculating the blending functions, but also prevents unnecessary I/O of the blending functions. In order to achieve higher utilization of the functional units as well as to save hardware components, a reconfigurable structure has been proposed for the M&A unit.

In Chapter 6, the cycle counts for various evaluation routines are calculated under the assumption that the chip receives one control point coordinate at a time from the host processor. If we increase the I/O

bandwidth to send or receive two or even four control point coordinates at a time by increasing the number of I/O pads and of I/O buffers, we can reduce the startup time for patch evaluation and the overall evaluation time for point evaluation.

Besides surface evaluation, different point transformations, such as position translation, rotation, scaling and reflection, can easily be done by the proposed architecture. Parallel projection and central projection can also be achieved by the SE chip with a little more complicated control routine.

Recall that in the formulation of the blending functions, their first derivatives are being calculated at the same time, but they remain unused for the following computation. The derivatives of the blending functions are mainly for the calculating of the tangent and normal vectors of the evaluated point, and the vectors can then be used to shade the patch.

According to Pickelmann [1], a complete algorithm for interactive design of B-spline surface should include the conversion of non-uniform knot vectors to enhanced uniform knot vectors (defined in Section 3.1.1), knot vector translation, enhanced uniform knot vector compression (or labelling), blending function formulation, point evaluation, tangent vector and normal vector calculation, and shading. The SE chip can now perform blending function formulation and point evaluation. The routines for computing the tangent and normal vector and shading are still to be done. The rest of the algorithm is assumed to be accomplished by the host processor. The rest of the algorithm,

however, can be fully implemented into dedicated hardware. If this implementation can be achieved at reasonable cost, it would be possible to design a chip, or a chip set, to handle the preprocessing for the interactive design of B-spline surfaces. But the data transmission between the preprocessing chip or chip set and the SE chip is still a bottleneck. A possible solution to this is to design a specific processor to accomplish the whole algorithm. Future research should emphasize developing a dedicated VLSI processor for the Computer-Aided Geometric Design (CAGD) of B-spline surfaces.

APPENDICES

APPENDIX 1

DESCRIPTION OF THE SE CHIP IN RTL

```
/* THE RTL DESCRIPTION OF SURFACE EVALUATION CHIP */
type REGISTER1 : Bit<31:0>;
     REGISTER2 : Bit<63:0>;
     DYNREG : Bit<63:0>;
     LATCH1 : Bit<31:0>;
     LATCH2 : Bit<63:0>;
     RAM : Bit<31:0>;
     ROM : Bit<31:0>;

var /* basic functional elements */
    MPY1_1, MPY1_2, MPY1_3, MPY1_4 OF Bit<63:0>;  /* segments of MAC1 */
    MPY2_1, MPY2_2, MPY2_3, MPY2_4 OF Bit<63:0>;  /* segments of MAC2 */
    MPY3_1, MPY3_2, MPY3_3, MPY3_4 OF Bit<63:0>;  /* segments of MAC3 */
    MPY4_1, MPY4_2, MPY4_3, MPY4_4 OF Bit<63:0>;  /* segments of MAC4 */
    MPY5_1, MPY5_2, MPY5_3, MPY5_4 OF Bit<63:0>;  /* segments of MAC5 */
    ADDER OF Bit<63:0>;  /* 4-operand Adder */
    D1,D2,D3,D4,D5,D6,D7,D8,D9,D10 OF Bit<63:0>; /* segments of Dividor */

    /* storage elements */
    BLD(16,4,4) OF ROM;  /* blending coefficient matrices */
    CPM(4,4,4) OF RAM;  /* control point matrices */
    IT(4,4) OF RAM;  /* intermediate product term */
    RBLD(2,4) OF RAM;  /* for storing blending functions */
    RBLDD(2,4) OF RAM; /* first derivatives of blending functions */

    /* dynamic registers and latches */
    M1D3, M2D3, M3D3, M4D3, M5D3 OF LATCH2; /* 3-stage delay within MAC's */
    M1D1, M2D1 OF LATCH2;  /* 1-stage delay within MAC1 and MAC2 */
    DLY4 OF LATCH1;  /* 4-stage delay unit */
    DLY8 OF LATCH1;  /* 8-stage delay unit */
    LAD3 OF LATCH1;  /* 3-stage delay */
    L1_1, L1_2 OF DYNREG;  /* input latches of MAC1 */
    L2_1, L2_2 OF DYNREG;  /* input latches of MAC2 */
    L3_1, L3_2 OF DYNREG;  /* input latches of MAC3 */
    L4_1, L4_2 OF DYNREG;  /* input latches of MAC4 */
    L5_1, L5_2 OF DYNREG;  /* input latches of MAC5 */
    LA1, LA2, LA3, LA4 OF DYNREG;  /* input latches of Adder */
    LD1, LD2 OF DYNREG;  /* input latches of Dividor */

    /* registers and control signal */
    INPUT, OUTPUT OF REGISTER1;  /* input and output buffer */
    RS, RT OF REGISTER1;  /* registers for parameter s and t */
    R(4) OF REGISTER2;  /* registers for WX, WY,WZ, and W */
    LABS, LABT OF Bit<3:0>;  /* labels of s and of t */
    RAT OF Bit<0:0>;  /* 1 for rational, 0 for nonrational */
    PATCH OF Bit<0:0>;  /* 1 for patch evaluation, 0 for point
                           evaluation */

Begin     /* the routine for nonrational spline point evaluation */
Cycle  (0): if RAT=0 then
        begin
                  (1): RS<-INPUT;
                  (2): RT<-INPUT;
                  (3): LABS<-INPUT;
```

```
                (4): LABT<-INPUT;
                (5): CPM(1,1,1)<-INPUT, L1_1<-BLD(LABS,1,1),
                     M1D3<-BLD(LABS,1,2), DL4<-BLD(LABS,1,3),
                     DL8<-BLD(LABS,1,4), L1_2<-RS, LAD3<-BLD(LABS,1,1);
                (6): CPM(2,1,1)<-INPUT, L1_1<-BLD(LABS,2,1),
                     M1D3<-BLD(LABS,2,2), DL4<-BLD(LABS,2,3),
                     DL8<-BLD(LABS,2,4), L1_2<-RS, LAD3<-BLD(LABS,2,1);
                (7): CPM(3,1,1)<-INPUT, L1_1<-BLD(LABS,3,1),
                     M1D3<-BLD(LABS,3,2), DL4<-BLD(LABS,3,3),
                     DL8<-BLD(LABS,3,4), L1_2<-RS, LAD3<-BLD(LABS,3,1);
                (8): CPM(1,1,2)<-INPUT, L1_1<-BLD(LABS,4,1),
                     M1D3<-BLD(LABS,4,2), DL4<-BLD(LABS,4,3),
                     DL8<-BLD(LABS,4,4), L1_2<-RS, LA1<-0, LA2<-LAD3,
                     LA3<-LAD3, LA4<-LAD3, LAD3<-BLD(LABS,4,1);
                (9): CPM(2,1,2)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                     M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS LA1<-0, LA2<-LAD3,
                     LA3<-LAD3, LA4<-LAD3;
               (10): CPM(3,1,2)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                     M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS LA1<-0, LA2<-LAD3,
                     LA3<-LAD3, LA4<-LAD3;
               (11): CPM(1,1,3)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                     M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS LA1<-0, LA2<-LAD3,
                     LA3<-LAD3, LA4<-LAD3;
               (12): CPM(2,1,3)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                     M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (13): CPM(3,1,3)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                     L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
               (14): CPM(1,1,4)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                     L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
               (15): CPM(2,1,4)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                     L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
               (16): CPM(3,1,4)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                     L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
/* blending functions of s ----------------------------------   */
               (17): CPM(1,2,2)<-INPUT, RBLD(1,1)<-MPY3_4,
                     RBLDD(1,1)<-MPY5_4;
               (18): CPM(2,2,2)<-INPUT. RBLD(1,2)<-MPY3_4,
                     RBLDD(1,2)<-MPY5_4;
               (19): CPM(3,2,2)<-INPUT, RBLD(1,3)<-MPY3_4,
                     RBLDD(1,3)<-MPY5_4;
               (20): CPM(1,2,3)<-INPUT, RBLD(1,4)<-MPY3_4,
                     RBLDD(1,4)<-MPY5_4;
/* ------------------------------------------------------------  */
               (21): CPM(2,2,3)<-INPUT, L1_1<-BLD(LABT,1,1),
                     M1D3<-BLD(LABT,1,2), DL4<-BLD(LABT,1,3),
                     DL8<-BLD(LABT,1,4), L1_2<-RT, LAD3<-BLD(LABT,1,1);
               (22): CPM(3,2,3)<-INPUT, L1_1<-BLD(LABT,2,1),
                     M1D3<-BLD(LABT,2,2), DL4<-BLD(LABT,2,3),
                     DL8<-BLD(LABT,2,4), L1_2<-RT, LAD3<-BLD(LABT,2,1);
               (23): CPM(1,2,4)<-INPUT, L1_1<-BLD(LABT,3,1),
                     M1D3<-BLD(LABT,3,2), DL4<-BLD(LABT,3,3),
                     DL8<-BLD(LABT,3,4), L1_2<-RT, LAD3<-BLD(LABT,3,1);
               (24): CPM(2,2,4)<-INPUT, L1_1<-BLD(LABT,4,1),
                     M1D3<-BLD(LABT,4,2), DL4<-BLD(LABT,4,3),
                     DL8<-BLD(LABT,4,4), L1_2<-RT, LA1<-0, LA2<-LAD3,
                     LA3<-LAD3, LA4<-LAD3, LAD3<-BLD(LABT,4,1);
```

```
                (25): CPM(3,2,4)<-INPUT, L4_1<-ADDER, L4_2<-RT,
                      M4D3<-M1D1, L2_1<-MPY1_4,L2_2<-RTLA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3;
                (26): CPM(1,2,1)<-INPUT, L4_1<-ADDER, L4_2<-RT,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RTLA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3;
                (27): CPM(2,2,1)<-INPUT, L4_1<-ADDER, L4_2<-RT,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RTLA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3;
                (28): CPM(3,2,1)<-INPUT, L4_1<-ADEER, L4_2<-RT,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RT;
                (29): CPM(1,3,3)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                      L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
                (30): CPM(2,3,3)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                      L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
                (31): CPM(3,3,3)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                      L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
                (32): CPM(1,3,4)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                      L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
/* blending functions of t ------------------------------------   */
                (33): CPM(2,3,4)<-INPUT, RBLD(2,1)<-MPY3_4,
                      RBLDD(2,1)<-MPY5_4;
                (34): CPM(3,3,4)<-INPUT, RBLD(2,2)<-MPY3_4,
                      RBLDD(2,2)<-MPY5_4;
                (35): CPM(1,3,1)<-INPUT, RBLD(2,3)<-MPY3_4,
                      RBLDD(2,3)<-MPY5_4;
                (36): CPM(2,3,1)<-INPUT, RBLD(2,4)<-MPY3_4,
                      RBLDD(2,4)<-MPY5_4;
/* ---------------------------------------------------------   */
                (37): CPM(3,3,1)<-INPUT
                (38): CPM(1,3,2)<-INPUT
                (39): CPM(2,3,2)<-INPUT
                (40): CPM(3,3,2)<-INPUT
                (41): CPM(1,4,4)<-INPUT
                (42): CPM(2,4,4)<-INPUT
                (43): CPM(3,4,4)<-INPUT
                (44): CPM(1,4,1)<-INPUT
                (45): CPM(2,4,1)<-INPUT
                (46): CPM(3,4,1)<-INPUT
                (47): CPM(1,4,2)<-INPUT
                (48): CPM(2,4,2)<-INPUT
                (49): CPM(3,4,2)<-INPUT
                (50): CPM(1,4,3)<-INPUT
                (51): CPM(2,4,3)<-INPUT
                (52): CPM(3,4,3)<-INPUT
                (53): L1_1<-CPM(1,1,1), L1_2<-RBLD(1,1), M1D3<-0,
                      L2_1<-CPM(1,1,2), L2_2<-RBLD(1,2), M2D3<-0,
                      L4_1<-CPM(1,1,3), L4_2<-RBLD(1,3), M4D3<-0,
                      L5_1<-CPM(1,1,4), L5_2<-RBLD(1,4), M5D3<-0;
                (54): L1_1<-CPM(1,2,2), L1_2<-RBLD(1,1), M1D3<-0,
                      L2_1<-CPM(1,2,3), L2_2<-RBLD(1,2), M2D3<-0,
                      L4_1<-CPM(1,2,4), L4_2<-RBLD(1,3), M4D3<-0,
                      L5_1<-CPM(1,2,1), L5_2<-RBLD(1,4), M5D3<-0;
                (55): L1_1<-CPM(1,3,3), L1_2<-RBLD(1,1), M1D3<-0,
                      L2_1<-CPM(1,3,4), L2_2<-RBLD(1,2), M2D3<-0,
                      L4_1<-CPM(1,3,1), L4_2<-RBLD(1,3), M4D3<-0,
```

```
      L5_1<-CPM(1,3,2), L5_2<-RBLD(1,4), M5D3<-0;
(56): L1_1<-CPM(1,4,4), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(1,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(1,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(1,4,3), L5_2<-RBLD(1,4), M5D3<-0;
(57): L1_1<-CPM(2,1,1), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,1,2), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,1,3), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,1,4), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4;
(58): L1_1<-CPM(2,2,2), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,2,3), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,2,4), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,2,1), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,1)<-ADDER;
(59): L1_1<-CPM(2,3,3), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,3,4), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,3,1), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,3,2), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,2)<-ADDER;
(60): L1_1<-CPM(2,4,4), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,4,3), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,3)<-ADDER;
(61): L1_1<-CPM(3,1,1), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,1,2), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,1,3), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,1,4), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,4)<-ADDER;
(62): L1_1<-CPM(3,2,2), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,2,3), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,2,4), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,2,1), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,1)<-ADDER;
(63): L1_1<-CPM(3,3,3), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,3,4), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,3,1), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,3,2), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,2)<-ADDER;
(64): L1_1<-CPM(3,4,4), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,4,3), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,3)<-ADDER;
(65): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,4)<-ADDER;
(66): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
```

```
                    LA4<-MPY5_4, IT(3,1)<-ADDER;
               (67): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                    LA4<-MPY5_4, IT(3,2)<-ADDER;
               (68): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                    LA4<-MPY5_4, IT(3,3)<-ADDER;
               (69): IT(3,4)<-ADDER;
               (70): L1_1<-IT(1,1), L1_2<-RBLD(2,1), M1D3<-0,
                    L2_1<-IT(1,2), L2_2<-RBLD(2,2), M2D3<-0,
                    L4_1<-IT(1,3), L4_2<-RBLD(2,3), M4D3<-0,
                    L5_1<-IT(1,4), L5_2<-RBLD(2,4), M5D3<-0;
               (71): L1_1<-IT(2,1), L1_2<-RBLD(2,1), M1D3<-0,
                    L2_1<-IT(2,2), L2_2<-RBLD(2,2), M2D3<-0,
                    L4_1<-IT(2,3), L4_2<-RBLD(2,3), M4D3<-0,
                    L5_1<-IT(2,4), L5_2<-RBLD(2,4), M5D3<-0;
               (72): L1_1<-IT(3,1), L1_2<-RBLD(2,1), M1D3<-0,
                    L2_1<-IT(3,2), L2_2<-RBLD(2,2), M2D3<-0,
                    L4_1<-IT(3,3), L4_2<-RBLD(2,3), M4D3<-0,
                    L5_1<-IT(3,4), L5_2<-RBLD(2,4), M5D3<-0;
               (73):
               (74): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                    LA4<-MPY5_4;
               (75): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                    LA4<-MPY5_4, R(2)<-ADDER;
               (76): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                    LA4<-MPY5_4, R(3)<-ADDER, OUTPUT<-R(2);
               (77): R(4)<-ADDER, OUTPUT<-R(3);
               (78): OUTPUT<-R(4);
               (79): if (PATCH=0 or (s=1 and t=1)) then stop;  /* stop here*/
          else begin
/* input new parameter values */
               (80): RS<-INPUT;
               (81): RT<-INPUT;
               (82): L1_1<-BLD(LABS,1,1), M1D3<-BLD(LABS,1,2),
                    DL4<-BLD(LABS,1,3), DL8<-BLD(LABS,1,4),
                    L1_2<-RS, LAD3<-BLD(LABS,1,1);
               (83): L1_1<-BLD(LABS,2,1), M1D3<-BLD(LABS,2,2),
                    DL4<-BLD(LABS,2,3), DL8<-BLD(LABS,2,4),
                    L1_2<-RS, LAD3<-BLD(LABS,2,1);
               (84): L1_1<-BLD(LABS,3,1), M1D3<-BLD(LABS,3,2),
                    DL4<-BLD(LABS,3,3), DL8<-BLD(LABS,3,4),
                    L1_2<-RS, LAD3<-BLD(LABS,3,1);
               (85): L1_1<-BLD(LABS,4,1), M1D3<-BLD(LABS,4,2),
                    DL4<-BLD(LABS,4,3), DL8<-BLD(LABS,4,4),
                    L1_2<-RS, LAD3<-BLD(LABS,4,1), LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3;
               (86): L1_1<-BLD(LABT,1,1), M1D3<-BLD(LABT,1,2),
                    DL4<-BLD(LABT,1,3), DL8<-BLD(LABT,1,4),
                    L1_2<-RT, LAD3<-BLD(LABT,1,1), LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS,;
               (87): L1_1<-BLD(LABT,2,1), M1D3<-BLD(LABT,2,2),
                    DL4<-BLD(LABT,2,3), DL8<-BLD(LABT,2,4),
                    L1_2<-RT, LAD3<-BLD(LABT,2,1), LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (88): L1_1<-BLD(LABT,3,1), M1D3<-BLD(LABT,3,2),
```

```
                    DL4<-BLD(LABT,3,3), DL8<-BLD(LABT,3,4),
                    L1_2<-RT, LAD3<-BLD(LABT,3,1), LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (89): L1_1<-BLD(LABT,4,1), M1D3<-BLD(LABT,4,2),
                    DL4<-BLD(LABT,4,3), DL8<-BLD(LABT,4,4),
                    L1_2<-RT, LAD3<-BLD(LABT,4,1), LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (90): LA1<-0, LA2<-LAD3, LA3<-LAD3, LA4<-LAD3,
                    L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                    M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                    L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                    L2_2<-RT;
               (91): LA1<-0, LA2<-LAD3, LA3<-LAD3, LA4<-LAD3,
                    L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                    M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                    L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                    L2_2<-RT;
               (92): LA1<-0, LA2<-LAD3, LA3<-LAD3, LA4<-LAD3,
                    L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                    M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                    L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                    L2_2<-RT;
               (93): L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                    M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                    L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                    L2_2<-RT;
               (94): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                    M5D3<-M2D1, RBLD(1,1)<-MPY3_4, RBLDD(1,1)<-MPY5_4;
               (95): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                    M5D3<-M2D1, RBLD(1,2)<-MPY3_4, RBLDD(1,2)<-MPY5_4;
               (96): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                    M5D3<-M2D1, RBLD(1,3)<-MPY3_4, RBLDD(1,3)<-MPY5_4;
               (97): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                    M5D3<-M2D1, RBLD(1,4)<-MPY3_4, RBLDD(1,4)<-MPY5_4;
               (98): RBLD(2,1)<-MPY3_4, RBLDD(2,1)<-MPY5_4;
               (99): RBLD(2,2)<-MPY3_4, RBLDD(2,2)<-MPY5_4;
              (100): RBLD(2,3)<-MPY3_4, RBLDD(2,3)<-MPY5_4;
              (101): RBLD(2,4)<-MPY3_4, RBLDD(2,4)<-MPY5_4, go to (53);
               end else;
        end;

/* the routine for rational spline point evaluation */
        else begin
                (1): RS<-INPUT;
                (2): RT<-INPUT;
                (3): LABS<-INPUT;
                (4): LABT<-INPUT;
                (5): CPM(1,1,1)<-INPUT, L1_1<-BLD(LABS,1,1),
                    M1D3<-BLD(LABS,1,2), DL4<-BLD(LABS,1,3),
                    DL8<-BLD(LABS,1,4), L1_2<-RS, LAD3<-BLD(LABS,1,1);
                (6): CPM(1,1,2)<-INPUT, L1_1<-BLD(LABS,2,1),
                    M1D3<-BLD(LABS,2,2), DL4<-BLD(LABS,2,3),
                    DL8<-BLD(LABS,2,4), L1_2<-RS, LAD3<-BLD(LABS,2,1);
                (7): CPM(1,1,3)<-INPUT, L1_1<-BLD(LABS,3,1),
```

```
                    M1D3<-BLD(LABS,3,2), DL4<-BLD(LABS,3,3),
                    DL8<-BLD(LABS,3,4), L1_2<-RS, LAD3<-BLD(LABS,3,1);
                (8): CPM(1,1,4)<-INPUT, L1_1<-BLD(LABS,4,1),
                    M1D3<-BLD(LABS,4,2), DL4<-BLD(LABS,4,3),
                    DL8<-BLD(LABS,4,4), L1_2<-RS, LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3, LAD3<-BLD(LABS,4,1);
                (9): CPM(1,2,2)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3;
               (10): CPM(1,2,3)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3;
               (11): CPM(1,2,4)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3;
               (12): CPM(1,2,1)<-INPUT, L4_1<-ADDER, L4_2<-RS,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (13): CPM(1,3,3)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                    L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
               (14): CPM(1,3,4)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                    L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
               (15): CPM(1,3,1)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                    L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
               (16): CPM(1,3,2)<-INPUT, L3_1<-MPY2_4, L3_2<-RS,
                    L5_1<-MPY4_4, L5_2<-RS, M5D3<-M2D1;
/* blending functions of s ----------------------------------   */
               (17): CPM(1,4,4)<-INPUT, RBLD(1,1)<-MPY3_4,
                    RBLDD(1,1)<-MPY5_4;
               (18): CPM(1,4,1)<-INPUT. RBLD(1,2)<-MPY3_4,
                    RBLDD(1,2)<-MPY5_4;
               (19): CPM(1,4,2)<-INPUT, RBLD(1,3)<-MPY3_4,
                    RBLDD(1,3)<-MPY5_4;
               (20): CPM(1,4,3)<-INPUT, RBLD(1,4)<-MPY3_4,
                    RBLDD(1,4)<-MPY5_4;
/* ------------------------------------------------------   */
               (21): CPM(2,1,1)<-INPUT, L1_1<-BLD(LABT,1,1),
                    M1D3<-BLD(LABT,1,2), DL4<-BLD(LABT,1,3),
                    DL8<-BLD(LABT,1,4), L1_2<-RT, LAD3<-BLD(LABT,1,1);
               (22): CPM(2,1,2)<-INPUT, L1_1<-BLD(LABT,2,1),
                    M1D3<-BLD(LABT,2,2), DL4<-BLD(LABT,2,3),
                    DL8<-BLD(LABT,2,4), L1_2<-RT, LAD3<-BLD(LABT,2,1);
               (23): CPM(2,1,3)<-INPUT, L1_1<-BLD(LABT,3,1),
                    M1D3<-BLD(LABT,3,2), DL4<-BLD(LABT,3,3),
                    DL8<-BLD(LABT,3,4), L1_2<-RT, LAD3<-BLD(LABT,3,1);
               (24): CPM(2,1,4)<-INPUT, L1_1<-BLD(LABT,4,1),
                    M1D3<-BLD(LABT,4,2), DL4<-BLD(LABT,4,3),
                    DL8<-BLD(LABT,4,4), L1_2<-RT, LA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3, LAD3<-BLD(LABT,4,1);
               (25): CPM(2,2,2)<-INPUT, L4_1<-ADDER, L4_2<-RT,
                    M4D3<-M1D1, L2_1<-MPY1_4,L2_2<-RTLA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3;
               (26): CPM(2,2,3)<-INPUT, L4_1<-ADDER, L4_2<-RT,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RTLA1<-0, LA2<-LAD3,
                    LA3<-LAD3, LA4<-LAD3;
               (27): CPM(2,2,4)<-INPUT, L4_1<-ADDER, L4_2<-RT,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RTLA1<-0, LA2<-LAD3,
```

```
                    LA3<-LAD3, LA4<-LAD3;
               (28): CPM(2,2,1)<-INPUT, L4_1<-ADEER, L4_2<-RT,
                    M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RT;
               (29): CPM(2,3,3)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                    L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
               (30): CPM(2,3,4)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                    L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
               (31): CPM(2,3,1)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                    L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
               (32): CPM(2,3,2)<-INPUT, L3_1<-MPY2_4, L3_2<-RT,
                    L5_1<-MPY4_4, L5_2<-RT, M5D3<-M2D1;
/* blending functions of t ------------------------------------   */
               (33): CPM(2,4,4)<-INPUT, RBLD(2,1)<-MPY3_4,
                    RBLDD(2,1)<-MPY5_4;
               (34): CPM(2,4,1)<-INPUT, RBLD(2,2)<-MPY3_4,
                    RBLDD(2,2)<-MPY5_4;
               (35): CPM(2,4,2)<-INPUT, RBLD(2,3)<-MPY3_4,
                    RBLDD(2,3)<-MPY5_4;
               (36): CPM(2,4,3)<-INPUT, RBLD(2,4)<-MPY3_4,
                    RBLDD(2,4)<-MPY5_4;
/* ---------------------------------------------------------   */
               (37): CPM(3,1,1)<-INPUT
               (38): CPM(3,1,2)<-INPUT
               (39): CPM(3,1,3)<-INPUT
               (40): CPM(3,1,4)<-INPUT
               (41): CPM(3,2,2)<-INPUT
               (42): CPM(3,2,3)<-INPUT
               (43): CPM(3,2,4)<-INPUT
               (44): CPM(3,2,1)<-INPUT
               (45): CPM(3,3,3)<-INPUT
               (46): CPM(3,3,4)<-INPUT
               (47): CPM(3,3,1)<-INPUT
               (48): CPM(3,3,2)<-INPUT
               (49): CPM(3,4,4)<-INPUT
               (50): CPM(3,4,1)<-INPUT
               (51): CPM(3,4,2)<-INPUT
               (52): CPM(3,4,3)<-INPUT
               (53): CPM(4,1,1)<-INPUT
               (54): CPM(4,1,2)<-INPUT
               (55): CPM(4,1,3)<-INPUT
               (56): CPM(4,1,4)<-INPUT
               (57): CPM(4,2,2)<-INPUT
               (58): CPM(4,2,3)<-INPUT
               (59): CPM(4,2,4)<-INPUT
               (60): CPM(4,2,1)<-INPUT
               (61): CPM(4,3,3)<-INPUT
               (62): CPM(4,3,4)<-INPUT
               (63): CPM(4,3,1)<-INPUT
               (64): CPM(4,3,2)<-INPUT
               (65): CPM(4,4,4)<-INPUT
               (66): CPM(4,4,1)<-INPUT
               (67): CPM(4,4,2)<-INPUT
               (68): CPM(4,4,3)<-INPUT
               (69): L1_1<-CPM(1,1,1), L1_2<-RBLD(1,1), M1D3<-0,
                    L2_1<-CPM(1,1,2), L2_2<-RBLD(1,2), M2D3<-0,
                    L4_1<-CPM(1,1,3), L4_2<-RBLD(1,3), M4D3<-0,
```

```
      L5_1<-CPM(1,1,4), L5_2<-RBLD(1,4), M5D3<-0;
(70): L1_1<-CPM(1,2,2), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(1,2,3), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(1,2,4), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(1,2,1), L5_2<-RBLD(1,4), M5D3<-0;
(71): L1_1<-CPM(1,3,3), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(1,3,4), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(1,3,1), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(1,3,2), L5_2<-RBLD(1,4), M5D3<-0;
(72): L1_1<-CPM(1,4,4), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(1,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(1,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(1,4,3), L5_2<-RBLD(1,4), M5D3<-0;
(73): L1_1<-CPM(2,1,1), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,1,2), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,1,3), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,1,4), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4;
(74): L1_1<-CPM(2,2,2), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,2,3), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,2,4), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,2,1), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,1)<-ADDER;
(75): L1_1<-CPM(2,3,3), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,3,4), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,3,1), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,3,2), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,2)<-ADDER;
(76): L1_1<-CPM(2,4,4), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(2,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(2,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(2,4,3), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,3)<-ADDER;
(77): L1_1<-CPM(3,1,1), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,1,2), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,1,3), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,1,4), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(1,4)<-ADDER;
(78): L1_1<-CPM(3,2,2), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,2,3), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,2,4), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,2,1), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,1)<-ADDER;
(79): L1_1<-CPM(3,3,3), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(3,3,4), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,3,1), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,3,2), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,2)<-ADDER;
(80): L1_1<-CPM(3,4,4), L1_2<-RBLD(1,1), M1D3<-0,
```

```
      L2_1<-CPM(3,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(3,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(3,4,3), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,3)<-ADDER;
(81): L1_1<-CPM(4,1,1), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(4,1,2), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(4,1,3), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(4,1,4), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(2,4)<-ADDER;
(82): L1_1<-CPM(4,2,2), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(4,2,3), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(4,2,4), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(4,2,1), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(3,1)<-ADDER;
(83): L1_1<-CPM(4,3,3), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(4,3,4), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(4,3,1), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(4,3,2), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(3,2)<-ADDER;
(84): L1_1<-CPM(4,4,4), L1_2<-RBLD(1,1), M1D3<-0,
      L2_1<-CPM(4,4,1), L2_2<-RBLD(1,2), M2D3<-0,
      L4_1<-CPM(4,4,2), L4_2<-RBLD(1,3), M4D3<-0,
      L5_1<-CPM(4,4,3), L5_2<-RBLD(1,4), M5D3<-0,
      LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(3,3)<-ADDER;
(85): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(3,4)<-ADDER;
(86): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(4,1)<-ADDER;
(87): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(4,2)<-ADDER;
(88): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4, IT(4,3)<-ADDER;
(89): IT(4,4)<-ADDER;
(90): L1_1<-IT(1,1), L1_2<-RBLD(2,1), M1D3<-0,
      L2_1<-IT(1,2), L2_2<-RBLD(2,2), M2D3<-0,
      L4_1<-IT(1,3), L4_2<-RBLD(2,3), M4D3<-0,
      L5_1<-IT(1,4), L5_2<-RBLD(2,4), M5D3<-0;
(91): L1_1<-IT(2,1), L1_2<-RBLD(2,1), M1D3<-0,
      L2_1<-IT(2,2), L2_2<-RBLD(2,2), M2D3<-0,
      L4_1<-IT(2,3), L4_2<-RBLD(2,3), M4D3<-0,
      L5_1<-IT(2,4), L5_2<-RBLD(2,4), M5D3<-0;
(92): L1_1<-IT(3,1), L1_2<-RBLD(2,1), M1D3<-0,
      L2_1<-IT(3,2), L2_2<-RBLD(2,2), M2D3<-0,
      L4_1<-IT(3,3), L4_2<-RBLD(2,3), M4D3<-0,
      L5_1<-IT(3,4), L5_2<-RBLD(2,4), M5D3<-0;
(93): L1_1<-IT(4,1), L1_2<-RBLD(2,1), M1D3<-0,
      L2_1<-IT(4,2), L2_2<-RBLD(2,2), M2D3<-0,
      L4_1<-IT(4,3), L4_2<-RBLD(2,3), M4D3<-0,
      L5_1<-IT(4,4), L5_2<-RBLD(2,4), M5D3<-0;
(94): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
      LA4<-MPY5_4;
```

```
                (95): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                      LA4<-MPY5_4, R(1)<-ADDER;
                (96): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                      LA4<-MPY5_4, LD1<-ADDER, LD2<-R(1);
                (97): LA1<-MPY1_4, LA2<-MPY2_4, LA3<-MPY4_4,
                      LA4<-MPY5_4, LD1<-ADDER, LD2<-R(1);
                (98): R(4)<-ADDER, LD1<-ADDER, LD2<-R(1);
                (99):
               (100):
               (101):
               (102):
               (103):
               (104):
               (105):
               (106): R(2)<-D10;
               (107): R(3)<-D10, OUTPUT<-R(2);
               (108): R(4)<-D10, OUTPUT<-R(3);
               (109): OUTPUT<-R(4);
               (110): if (PATCH=0 or (s=1 and t=1)) then stop;  /* stop here *
          else begin
/* input new parameter values */
               (111): RS<-INPUT;
               (112): RT<-INPUT;
               (113): L1_1<-BLD(LABS,1,1), M1D3<-BLD(LABS,1,2),
                      DL4<-BLD(LABS,1,3), DL8<-BLD(LABS,1,4),
                      L1_2<-RS, LAD3<-BLD(LABS,1,1);
               (114): L1_1<-BLD(LABS,2,1), M1D3<-BLD(LABS,2,2),
                      DL4<-BLD(LABS,2,3), DL8<-BLD(LABS,2,4),
                      L1_2<-RS, LAD3<-BLD(LABS,2,1);
               (115): L1_1<-BLD(LABS,3,1), M1D3<-BLD(LABS,3,2),
                      DL4<-BLD(LABS,3,3), DL8<-BLD(LABS,3,4),
                      L1_2<-RS, LAD3<-BLD(LABS,3,1);
               (116): L1_1<-BLD(LABS,4,1), M1D3<-BLD(LABS,4,2),
                      DL4<-BLD(LABS,4,3), DL8<-BLD(LABS,4,4),
                      L1_2<-RS, LAD3<-BLD(LABS,4,1), LA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3;
               (117): L1_1<-BLD(LABT,1,1), M1D3<-BLD(LABT,1,2),
                      DL4<-BLD(LABT,1,3), DL8<-BLD(LABT,1,4),
                      L1_2<-RT, LAD3<-BLD(LABT,1,1), LA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS,;
               (118): L1_1<-BLD(LABT,2,1), M1D3<-BLD(LABT,2,2),
                      DL4<-BLD(LABT,2,3), DL8<-BLD(LABT,2,4),
                      L1_2<-RT, LAD3<-BLD(LABT,2,1), LA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (119): L1_1<-BLD(LABT,3,1), M1D3<-BLD(LABT,3,2),
                      DL4<-BLD(LABT,3,3), DL8<-BLD(LABT,3,4),
                      L1_2<-RT, LAD3<-BLD(LABT,3,1), LA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
               (120): L1_1<-BLD(LABT,4,1), M1D3<-BLD(LABT,4,2),
                      DL4<-BLD(LABT,4,3), DL8<-BLD(LABT,4,4),
                      L1_2<-RT, LAD3<-BLD(LABT,4,1), LA1<-0, LA2<-LAD3,
                      LA3<-LAD3, LA4<-LAD3, L4_1<-ADDER, L4_2<-RS,
                      M4D3<-M1D1, L2_1<-MPY1_4, L2_2<-RS;
```

```
            (121): LA1<-0, LA2<-LAD3, LA3<-LAD3, LA4<-LAD3,
                   L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                   M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                   L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                   L2_2<-RT;
            (122): LA1<-0, LA2<-LAD3, LA3<-LAD3, LA4<-LAD3,
                   L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                   M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                   L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                   L2_2<-RT;
            (123): LA1<-0, LA2<-LAD3, LA3<-LAD3, LA4<-LAD3,
                   L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                   M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                   L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                   L2_2<-RT;
            (124): L3_1<-MPY2_4, L3_2<-RS, L5_1<-MPY4_4, L5_2<-RS,
                   M5D3<-M2D1, LA1<-0, LA2<-LAD3,LA3<-LAD3, LA4<-LAD3,
                   L4_1<-ADDER, L4_2<-RT, M4D3<-M1D1, L2_1<-MPY1_4,
                   L2_2<-RT;
            (125): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                   M5D3<-M2D1, RBLD(1,1)<-MPY3_4, RBLDD(1,1)<-MPY5_4;
            (126): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                   M5D3<-M2D1, RBLD(1,2)<-MPY3_4, RBLDD(1,2)<-MPY5_4;
            (127): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                   M5D3<-M2D1, RBLD(1,3)<-MPY3_4, RBLDD(1,3)<-MPY5_4;
            (128): L3_1<-MPY2_4, L3_2<-RT, L5_1<-MPY4_4, L5_2<-RT,
                   M5D3<-M2D1, RBLD(1,4)<-MPY3_4, RBLDD(1,4)<-MPY5_4;
            (129): RBLD(2,1)<-MPY3_4, RBLDD(2,1)<-MPY5_4;
            (130): RBLD(2,2)<-MPY3_4, RBLDD(2,2)<-MPY5_4;
            (131): RBLD(2,3)<-MPY3_4, RBLDD(2,3)<-MPY5_4;
            (132): RBLD(2,4)<-MPY3_4, RBLDD(2,4)<-MPY5_4, go to (69);
             end else;
        end else;
End.
```

APPENDIX 2

INPUT SEQUENCE OF CONTROL POINT MATRICES

```
Cycle     (5): CPM(1,1,1) <- X11
          (6): CPM(2,1,1) <- Y11
          (7): CPM(3,1,1) <- Z11
          (8): CPM(1,1,2) <- X12
          (9): CPM(2,1,2) <- Y12
         (10): CPM(3,1,2) <- Z12
         (11): CPM(1,1,3) <- X13
         (12): CPM(2,1,3) <- Y13
         (13): CPM(3,1,3) <- Z13
         (14): CPM(1,1,4) <- X14
         (15): CPM(2,1,4) <- Y14
         (16): CPM(3,1,4) <- Z14
         (17): CPM(1,2,2) <- X21
         (18): CPM(2,2,2) <- Y21
         (19): CPM(3,2,2) <- Z21
         (20): CPM(1,2,3) <- X22
         (21): CPM(2,2,3) <- Y22
         (22): CPM(3,2,3) <- Z22
         (23): CPM(1,2,4) <- X23
         (24): CPM(2,2,4) <- Y23
         (25): CPM(3,2,4) <- Z23
         (26): CPM(1,2,1) <- X24
         (27): CPM(2,2,1) <- Y24
         (28): CPM(3,2,1) <- Z24
         (29): CPM(1,3,3) <- X31
         (30): CPM(2,3,3) <- Y31
         (31): CPM(3,3,3) <- Z31
         (32): CPM(1,3,4) <- X32
         (33): CPM(2,3,4) <- Y32
         (34): CPM(3,3,4) <- Z32
         (35): CPM(1,3,1) <- X33
         (36): CPM(2,3,1) <- Y33
         (37): CPM(3,3,1) <- Z33
         (38): CPM(1,3,2) <- X34
         (39): CPM(2,3,2) <- Y34
         (40): CPM(3,3,2) <- Z34
         (41): CPM(1,4,4) <- X41
         (42): CPM(2,4,4) <- Y41
         (43): CPM(3,4,4) <- Z41
         (44): CPM(1,4,1) <- X42
         (45): CPM(2,4,1) <- Y42
         (46): CPM(3,4,1) <- Z42
         (47): CPM(1,4,2) <- X43
         (48): CPM(2,4,2) <- Y43
         (49): CPM(3,4,2) <- Z43
         (50): CPM(1,4,3) <- X44
         (51): CPM(2,4,3) <- Y44
         (52): CPM(3,4,3) <- Z44
```

APPENDIX 3

TIME TABLE OF LATCH ELEMENTS (I)

Time table of latch elements for the formulation of blending functions and their first derivatives.

Cycle	LAD3	LA1	LA2	LA3	LA4	L1_1	L1_2	M1D3	L2_1	L2_2	M2D3
5	S11					S11	S	S12			
6	S21					S21	S	S22			
7	S31					S31	S	S32			
8	S41	0	S11	S11	S11	S41	S	S42			
9		0	S21	S21	S21				(M1_4)	S	S13
10		0	S31	S31	S31				(M1_4)	S	S23
11		0	S41	S41	S41				(M1_4)	S	S33
12									(M1_4)	S	S43
13											
14											
15											
16											
17											
18											
19											
20											
21	T11					T11	T	T12			
22	T21					T21	T	T22			
23	T31					T31	T	T32			
24	T41	0	T11	T11	T11	T41	T	T42			
25		0	T21	T21	T21				(M1_4)	T	T23
26		0	T31	T31	T31				(M1_4)	T	T33
27		0	T41	T41	T41				(M1_4)	T	T43
28											
29											
30											
31											
32											

Time table of latch elements for the formulation of blending functions and their first derivatives (cont'd).

Cycle	L3_1	L3_2	M3D3	L4_1	L4_2	M4D3	L5_1	L5_2	M5D3	D4	D8
5										S13	S14
6										S23	S24
7										S33	S34
8										S43	S44
9				3xS11	S	2xS12					
10				3xS21	S	2xS22					
11				3xS31	S	2xS32					
12				3xS41	S	2xS42					
13	(M2_4)	S	S14				(M4_4)	S	S13		
14	(M2_4)	S	S24				(M4_4)	S	S23		
15	(M2_4)	S	S34				(M4_4)	S	S33		
16	(M2_4)	S	S44				(M4_4)	S	S43		
17											
18											
19											
20											
21										T13	T14
22										T23	T24
23										T33	T34
24										T43	T44
25				3xT11	T	2xT12					
26				3xT21	T	2xT22					
27				3xT31	T	2xT32					
28				3xT41	T	2xT42					
29	(M2_4)	T	T14				(M4_4)	T	T13		
30	(M2_4)	T	T24				(M4_4)	T	T23		
31	(M2_4)	T	T34				(M4_4)	T	T33		
32	(M2_4)	T	T44				(M4_4)	T	T43		

APPENDIX 4

TIME TABLE OF FUNCTIONAL STAGES (I)

Time table of functional stages for the formulation of blending functions and their first derivatives.

Cycle	ADDER	MPY1_1	MPY1_2	MPY1_3	MPY1_4	MPY2_1	MPY2_2	MPY2_3	MPY2_4
5		(M1_4)							
6		(M1_4)	(M1_2)						
7		(M1_4)	(M1_2)	(M1_3)					
8	3xS11	(M1_4)	(M1_2)	(M1_3)	(M1_4)				
9	3xS21		(M1_2)	(M1_3)	(M1_4)	(M2_1)			
10	3xS31			(M1_3)	(M1_4)	(M2_1)	(M2_2)		
11	3xS41				(M1_4)	(M2_1)	(M2_2)	(M2_3)	
12						(M2_1)	(M2_2)	(M2_3)	(M2_4)
13							(M2_2)	(M2_3)	(M2_4)
14								(M2_3)	(M2_4)
15									(M2_4)
16									
17									
18									
19									
20									
21		(M1_4)							
22		(M1_4)	(M1_2)						
23		(M1_4)	(M1_2)	(M1_3)					
24	3xT11	(M1_4)	(M1_2)	(M1_3)	(M1_4)				
25	3xT21		(M1_2)	(M1_3)	(M1_4)	(M2_1)			
26	3xT31			(M1_3)	(M1_4)	(M2_1)	(M2_2)		
27	3xT41				(M1_4)	(M2_1)	(M2_2)	(M2_3)	
28						(M2_1)	(M2_2)	(M2_3)	(M2_4)
29							(M2_2)	(M2_3)	(M2_4)
30								(M2_3)	(M2_4)
31									(M2_4)
32									
33									
34									
35									

Time table of functional stages for the formulation of blending functions and their first derivatives (cont'd).

Cycle	MPY3_1	MPY3_2	MPY3_3	MPY3_4	MPY4_1	MPY4_2	MPY4_3	MPY4_4
5								
6								
7								
8								
9					(M4_1)			
10					(M4_1)	(M4_2)		
11					(M4_1)	(M4_2)	(M4_3)	
12					(M4_1)	(M4_2)	(M4_3)	(M4_4)
13	(M3_1)					(M4_2)	(M4_3)	(M4_4)
14	(M3_1)	(M3_2)					(M4_3)	(M4_4)
15	(M3_1)	(M3_2)	M(3_3)					(M4_4)
16	(M3_1)	(M3_2)	M(3_3)	BFS(1)				
17		(M3_2)	M(3_3)	BFS(2)				
18			M(3_3)	BFS(3)				
19				BFS(4)				
20								
21								
22								
23								
24								
25					(M4_1)			
26					(M4_1)	(M4_2)		
27					(M4_1)	(M4_2)	(M4_3)	
28					(M4_1)	(M4_2)	(M4_3)	(M4_4)
29	(M3_1)					(M4_2)	(M4_3)	(M4_4)
30	(M3_1)	(M3_2)					(M4_3)	(M4_4)
31	(M3_1)	(M3_2)	M(3_3)					(M4_4)
32	(M3_1)	(M3_2)	M(3_3)	BFT(1)				
33		(M3_2)	M(3_3)	BFT(2)				
34			M(3_3)	BFT(3)				
35				BFT(4)				

Time table of functional stages for the formulation of blending functions and their first derivatives (cont'd).

Cycle	MPY5_1	MPY5_2	MPY5_3	MPY5_4
5				
6				
7				
8				
9				
10				
11				
12				
13	(M5_1)			
14	(M5_1)	(M5_2)		
15	(M5_1)	(M5_2)	(M5_3)	
16	(M5_1)	(M5_2)	(M5_3)	DBFS(1)
17		(M5_2)	(M5_3)	DBFS(2)
18			(M5_3)	DBFS(3)
19				DBFS(4)
20				
21				
22				
23				
24				
25				
26				
27				
28				
29	(M5_1)			
30	(M5_1)	(M5_2)		
31	(M5_1)	(M5_2)	(M5_3)	
32	(M5_1)	(M5_2)	(M5_3)	DBFT(1)
33		(M5_2)	(M5_3)	DBFT(2)
34			(M5_3)	DBFT(3)
35				DBFT(4)

APPENDIX 5

TIME TABLE OF LATCH ELEMENTS (II)

Time table of latch elements for the evaluation of X, Y, and Z.

Cycle	LAD3	LA1	LA2	LA3	LA4	L1_1	L1_2	M1D3
53						X11	BFS(1)	0
54						X21	BFS(1)	0
55						X31	BFS(1)	0
56						X41	BFS(1)	0
57		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Y11	BFS(1)	0
58		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Y21	BFS(1)	0
59		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Y31	BFS(1)	0
60		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Y41	BFS(1)	0
61		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Z11	BFS(1)	0
62		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Z21	BFS(1)	0
63		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Z31	BFS(1)	0
64		(M1_4)	(M2_4)	(M4_4)	(M5_4)	Z41	BFS(1)	0
65		(M1_4)	(M2_4)	(M4_4)	(M5_4)			
66		(M1_4)	(M2_4)	(M4_4)	(M5_4)			
67		(M1_4)	(M2_4)	(M4_4)	(M5_4)			
68		(M1_4)	(M2_4)	(M4_4)	(M5_4)			
69								
70						IT(1,1)	BFT(1)	0
71						IT(2,1)	BFT(1)	0
72						IT(3,1)	BFT(1)	0
73								
74		(M1_4)	(M2_4)	(M4_4)	(M5_4)			
75		(M1_4)	(M2_4)	(M4_4)	(M5_4)			
76		(M1_4)	(M2_4)	(M4_4)	(M5_4)			

Time table of latch elements for the evaluation of X, Y, and Z (cont'd).

Cycle	L2_1	L2_2	M2D3	L4_1	L4_2	M4D3	L5_1	L5_2	M5D3
53	X12	BFS(2)	0	X13	BFS(3)	0	X14	BFS(4)	0
54	X22	BFS(2)	0	X23	BFS(3)	0	X24	BFS(4)	0
55	X32	BFS(2)	0	X33	BFS(3)	0	X34	BFS(4)	0
56	X42	BFS(2)	0	X43	BFS(3)	0	X44	BFS(4)	0
57	Y12	BFS(2)	0	Y13	BFS(3)	0	Y14	BFS(4)	0
58	Y22	BFS(2)	0	Y23	BFS(3)	0	Y24	BFS(4)	0
59	Y32	BFS(2)	0	Y33	BFS(3)	0	Y34	BFS(4)	0
60	Y42	BFS(2)	0	Y43	BFS(3)	0	Y44	BFS(4)	0
61	Z12	BFS(2)	0	Z13	BFS(3)	0	Z14	BFS(4)	0
62	Z22	BFS(2)	0	Z23	BFS(3)	0	Z24	BFS(4)	0
63	Z32	BFS(2)	0	Z33	BFS(3)	0	Z34	BFS(4)	0
64	Z42	BFS(2)	0	Z43	BFS(3)	0	Z44	BFS(4)	0
65									
66									
67									
68									
69									
70	IT(1,2)	BFT(2)	0	IT(1,3)	BFT(3)	0	IT(1,4)	BFT(4)	0
71	IT(2,2)	BFT(2)	0	IT(2,3)	BFT(3)	0	IT(2,4)	BFT(4)	0
72	IT(3,2)	BFT(2)	0	IT(3,3)	BFT(3)	0	IT(3,4)	BFT(4)	0
73									
74									
75									
76									

APPENDIX 6

TIME TABLE OF FUNCTIONAL STAGES (II)

Time table of MAC stages for the evaluation of X, Y, and Z.

Cycle	ADDER	MPY1_1	MPY1_2	MPY1_3	MPY1_4	MPY2_1	MPY2_2	MPY2_3	MPY2_4
53		(M1_1)				(M2_1)			
54		(M1_1)	(M1_2)			(M2_1)	(M2_2)		
55		(M1_1)	(M1_2)	(M1_3)		(M2_1)	(M2_2)	(M2_3)	
56		(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
57	IT(1,1)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
58	IT(1,2)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
59	IT(1,3)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
60	IT(1,4)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
61	IT(2,1)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
62	IT(2,2)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
63	IT(2,3)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
64	IT(2,4)	(M1_1)	(M1_2)	(M1_3)	(M1_4)	(M2_1)	(M2_2)	(M2_3)	(M2_4)
65	IT(3,1)		(M1_2)	(M1_3)	(M1_4)		(M2_2)	(M2_3)	(M2_4)
66	IT(3,2)			(M1_3)	(M1_4)			(M2_3)	(M2_4)
67	IT(3,3)				(M1_4)				(M2_4)
68	IT(3,4)								
69									
70		(M1_1)				(M2_1)			
71		(M1_1)	(M1_2)			(M2_1)	(M2_2)		
72		(M1_1)	(M1_2)	(M1_3)		(M2_1)	(M2_2)	(M2_3)	
73			(M1_2)	(M1_3)	(M1_4)		(M2_2)	(M2_3)	(M2_4)
74	X			(M1_3)	(M1_4)			(M2_3)	(M2_4)
75	Y				(M1_4)				(M2_4)
76	Z								

Time table of functional stages for the evaluation of X, Y, and Z (cont'd).

Cycle	MPY4_1	MPY4_2	MPY4_3	MPY4_4	MPY5_1	MPY5_2	MPY5_3	MPY5_4
53	(M4_1)				(M5_1)			
54	(M4_1)	(M4_2)			(M5_1)	(M5_2)		
55	(M4_1)	(M4_2)	(M4_3)		(M5_1)	(M5_2)	(M5_3)	
56	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
57	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
58	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
59	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
60	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
61	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
62	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
63	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
64	(M4_1)	(M4_2)	(M4_3)	(M4_4)	(M5_1)	(M5_2)	(M5_3)	(M5_4)
65		(M4_2)	(M4_3)	(M4_4)		(M5_2)	(M5_3)	(M5_4)
66			(M4_3)	(M4_4)			(M5_3)	(M5_4)
67				(M4_4)				(M5_4)
68								
69								
70	(M4_1)				(M5_1)			
71	(M4_1)	(M4_2)			(M5_1)	(M5_2)		
72	(M4_1)	(M4_2)	(M4_3)		(M5_1)	(M5_2)	(M5_3)	
73		(M4_2)	(M4_3)	(M4_4)		(M5_2)	(M5_3)	(M5_4)
74			(M4_3)	(M4_4)			(M5_3)	(M5_4)
75				(M4_4)				(M5_4)
76								

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Pickelmann, M.N., "The Design of Rational B-spline Algorithms for Interactive Color Shading of Surfaces," Ph.D. dissertation, Michigan State University, 1985.

2. "Now 3-D Image Can Be Moved in Real Time," Electronics, August 7, 1986, pp. 97-100.

3. "Survey of custom and semicustom ICs," *VLSI Des.*, Oct. 1984, pp. 30-43.

4. Holton, W.C. and Cavin, R.K., "A Perspective on CMOS Technology Trends," *Proceedings of the IEEE*, Vol. 74, No. 12, Dec 1986, pp. 1646-1668.

5. Baller, H.H., "VLSI/Software Engineering Design Methodology," *IEEE 1984 Workshop on the Engineering of VLSI and Software*, IEEE Computer Society Press, 1984, pp. 11-14.

6. Okuda, N., Sugai, M., and Goto, N., "Semicustom and Custom LSI Technology," *Proceedings of the IEEE*, Vol. 74, No. 12, Dec 1986, pp. 1636-1645.

7. Vaí, M.K. and Shanblatt, M.A., "Some Issues in the Design Automation of VLSI," Technical Report, Department of Electrical Engineering, Michigan State Unversity, 1986.

8. Watanabe, W, and Ackland, B., "FLUTE: An Expert Floor Planner for Full-custom VLSI Design," *IEEE Design & Test*, Feb. 1987.

9. Subrahmanyam, P.A., "Synapse: An Expert System for VLSI Design," *Computer*, July 1986, pp. 78-89.

10. Whitton, M.C., "Special Purpose Hardware For the Display of Free Form Surfaces," Master thesis, North Carolina State University, 1984.

11. Hawkins, J.L., "The Wonder of B-splines," Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, 1986.

12. Vanderploeg, M.J., "Surface Assessment Using Color Graphics," Ph.D. dissertation, Michigan State University, 1982.

13. Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces," Dept. of Computer Science, University of Utah, Dec. 1974.

14. Lane, J.M., Carpenter, L.C., Whitted, T., and Blinn, J.F., "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Communication of the ACM*, Vol. 23, No. 1, Jan. 1980, pp. 23-34.

15. Lane, J.M. and Riesenfeld, R.F., "A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces," *IEEE Trans. on Pattern Recognition and Machine Intelligence*, Jan. 1980, pp. 35-46.

16. Cox, M.G., "The Numerical Evaluation of B-splines," *J. Inst. Maths. Applics*, Vol. 10, 1972, pp. 134-147.

17. DeBoor, C., "On Calculation with B-splines," *J. Approx. Theory*, Vol. 6, 1972, pp. 50-62.

18. Computational Geometry for Design and Manufacture, I.D. Faux and M.J. Pratt, Ellis Horwood Limited, 1983, England.

19. Fundamentals of Interactive Computer Graphics, J.D. Foley and A. Van Dam, Addison-Wesley Publishing Company, 1982.

20. Amanatides, J., "Realism in Computer Graphics: A Survey," *Computer Graphics and Applications*, Vol. 7, No.1, Jan. 1987, pp. 44-56.

21. Phong, B.T., "Illumination for Computer General Pictures," *Comm. ACM*, Vol. 18, No. 6, June 1975, pp. 311-317.

22. Gouraud, H., "Continuous Shading of Curved Surfaces," *IEEE Trans. Computers*, Vol. C-20, No. 6, June 1971, pp. 623-629.

23. Introduction to VLSI Systems, C. Mead and L. Conway, Readings, MA: Addison-Wesley, 1978.

24. Computer Arithmetic, K. Hwang, John Wiley and Son, 1979, p. 184.

25. K. Hwang, op. cit., Chapter 8.

26. Leung, S.S. and Shanblatt, M.A., "A VLSI Chip Architecture for the Computation of the Kinematic Solution of a Robotic Manipulator," Dept. of EESS, Michigan State University, April 1985.

27. Gajski, D.D. and Kuhn, R.H., "New VLSI Tools," *IEEE Computer*, Vol. 16, No. 11, Dec. 1983, pp. 11-14.

28. Hartmann, A.C., "Software or Silicon? The Designer's Option," *Proceedings of the IEEE*, Vol. 74, No. 6, June 1986, pp. 861-874.

29. Principles of CMOS VLSI Design, N. Weste and K. Eshraghian, Addison-

Wesley, 1985.

30. K. Hwang, op. cit., p. 279.

31. Vai, M.K., "Performance-Design Tradeoff of Hierarchical VLSI Design Entry Points," M.S. Thesis, Michigan State University, 1985.

32. Ramachandran, V., "On Driving Many Long Wires in a VLSI Layout," *J. of the ACM*, Vol. 33, No. 4, Oct. 1986, pp.687-701.

33. "TRW's Superchip Passes First Milestone," *Electronics*, July 10, 1986, pp. 49-54.

34. Panasuck, C., "High-density Gate Arrays," *Electronic Design*, Oct. 16, 1986, pp. 94-100.

35. Winard, H., "Standard-cell Libraries Expand," *Electronic Design*, Sept. 11, 1986, pp. 179-184.

36. Guttag, K., Aken, J.V., and Asal, M., "Requirements for a VLSI Graphics Processor," *Computer Graphics and Application*, Vol. 6, No. 1, Jan. 1986, pp.32-47.

37. Ikedo, T., "High-Speed Techniques for a 3-D Color Graphics Terminal," *Computer Graphics and Application*, Vol. 4, No. 3, May 1984, pp. 46-58.

38. Kawahito, S., Kameyama, M., Higuchi, T., and Yamada, H., "A High-Speed Compact Multiplier Based on Multiple-Valued Bi-directional Current-Mode Circuit," *Proc. 17th Int'l Symposium on Multiple-Valued Logic*, Computer Society Press, Los Angles, Calif., May 1987, pp. 172-180.

39. Gamal, A.E., et al., "A CMOS 32b Wallace Tree Multiplier-Accumulator," *IEEE Digest Int'l Solid-State Circuits Conf.*, Feb. 1986, pp. 194-195.

MICHIGAN STATE UNIV. LIBRARIES
31293015913712