A VLSI Architecture for the computation of NURBS Patches

(Extended Abstract)

Meenakshi Sundaram Gopi Swami Manohar
Supercomputer Education and Research Centre
Indian Institute of Science,
Bangalore 560012 INDIA

1 Introduction

B-Spline curves and patches are increasingly being used in several areas of computer graphics and geometric modeling. The rationalized counterpart of B-Spline called the Non-Uniform Rational B-Spline (NURBS) is invariably used in all the present day geometric modeling packages. For an interactive modeling session, thousands of NURBS patches have to be computed and drawn per second. Such performance is beyond the reach of even the most advanced workstations available today. Advances in hardware support for parametric curve and patch generation have thus acquired increased importance.

Substantial progress has been made in the theoretical aspects of B-Splines [1] and their applications [13]. However, very little work has been done on hardware solutions for B-Spline curves and surfaces.

One of the early papers in this direction is the work of T.Li *et al.*[8] where an architecture to generate Bezier curves and patches was proposed. De Rose [3] *et al.*, proposed a triangular architecture to generate B-Spline curves using the deBoor-Cox algorithm. Mathias [9], has developed a similar architecture for Bezier curves using the de Casteljau algorithm. He has also developed architectures for B-Spline inversion and B-Spline generation [10]. Recently, Megson [11] has come up with a design to calculate the basis functions required to generate B-Splines. He has also developed a composite design to calculate B-Spline patches.

In most of the above architectures the size of the hardware is tied to the size of the problem that is to be solved. Further, they require a large number of I/O pins. These limitations seriously restrict their practical implementation. Megson [11] addresses some implementation issues. However, the architecture proposed there suffers heavily from excess hardware requirements and intimate coupling with the size of the problem.

Apart from the above, two VLSI architectures to compute Uniform B-Spline curves have been presented in [6]. A unified architecture to compute uniform rational and non-rational B-Spline curve/patch is also presented in [7].

None of the above propose a hardware solution for the generation of NURBS curves and patches. In this paper we give a *complete* hardware solution for the generation of NURBS patches. we show that our architecture performs better.

2 Theory of B-Splines

We define B-Splines curve by the equation

$$P(u) = \sum_{i=0}^{n} P_i N_{i,k}(u) \tag{1}$$

The point on the curve at the parametric value u is denoted by $P(u)$. There are $(n+1)$ control points denoted by P_i.

The blending function or the basis function is denoted by $N_{i,k}(u)$. These basis functions will decide the extent to which a particular control point controls the curve. The parameter k is called the order (one more than the degree) of the curve.

$N_{i,k}$ is defined as follows.

$$N_{i,1} = \begin{cases} 1 & if\ t_i \le u < t_{i+1} \\ 0 & otherwise \end{cases} \tag{2}$$

$$N_{i,k} = \frac{(u - t_i)N_{i,k-1}}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)N_{i+1,k-1}}{t_{i+k} - t_{i+1}} \tag{3}$$

The constants t_is, called *knot values*, are specific instances of the parametric value u and are strictly in non-decreasing order. There are totally $(n+k+1)$ knot values. All the knot values put together is called a *knot vector*.

At any particular value of the parameter u only k basis functions will have non-zero values. Hence only k control points will have control over the shape of the curve. These non-zero basis functions are called *useful* basis functions and the corresponding control points are called *active* control points.

The *useful* basis functions are known *apriori*. If $t_i \le u < t_{i+1}$ then the k *useful* kth order basis functions are $N_{i-k+1}, N_{i-k+2}, \cdots, N_{i,k}$. Using the first subscript of these *useful* basis functions, the *active* control points can also be found.

While incrementing the parameter u, if it crosses t_{i+1}, then the basis function $N_{i-k+1,k}$ becomes zero and $N_{i+1,k}$ joins the set of *useful* basis functions.

Similarly the *active* control point set also changes. These facts are used later in this paper.

A Rational B-Spline curve is defined by the formula,

$$P(u) = \frac{\sum_{i=0}^{n} P_i w_i N_{i,k}(u)}{\sum_{i=0}^{n} w_i N_{i,k}(u)} \quad (4)$$

The term w_i denotes the weight of the control point P_i. When w_i tends to infinity, the curve is pulled towards P_i and when w_i is zero, the control point P_i does not have any control over the curve.

Rational B-Spline surface is defined by the formula,

$$P(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij} w_{ij} N_{i,k}(u) N_{j,l}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{ij} N_{i,k}(u) N_{j,l}(v)} \quad (5)$$

There is a grid of $(n+1)(m+1)$ control points for the surface.

2.1 Basis Function Computation

In the computation of a B-Spline curve or a surface, the basis function computation plays an important role. As seen from the Eqn.3, the basis function computation is recursive and apparently requires $2^k - 1$ function calls to itself.

As there are only k useful basis functions and are known *apriori*, only those need be calculated. Hence the total number of calls to the basis function routine need only be $k(2^k - 1)$.

In the calculation of basis functions using Eqn.3 many lower order functions return zero. If there are multiple knots (such as, $t_i = t_{i+1} = t_{i+2} = t_{i+3}$), then the denominator of the Eqn.3 may become zero for certain calls to the basis function routine (such as $N_{i,1}$, $N_{i+1,1}$, $N_{i+2,1}$, $N_{i,2}$, $N_{i+1,2}$, $N_{i,3}$).

The above two difficulties are overcome by using the following method [2].

At a particular value of the parameter u, only one basis function of order one is non-zero, because one can find only one i such that $t_i \leq u < t_{i+1}$ as the t_is are in non-decreasing order. From the Figure 1, we can see that from the non-zero first order basis function, k kth order basis functions can be calculated. The computation along the edges is given in the inset and whenever the two edges meet, an addition is performed to get the basis function value at the meeting node.

In this method, every computation is indispensable and the denominator does not become zero. In what follows, this method is used to develop a new systolic architecture for the computation of basis functions.

3 Systolic computation of basis function

Figure 2 shows the systolic linear array for the computation of the basis function. Each cell in the BFEA computes one level in the DAG shown in the Figure 1, starting from the second level. Thus there are $k-1$ processing cells in the BFEA and from the last cell the k basis functions of order k are output. The value of $N_{i,1}(u)$ is always one and is pumped to the first

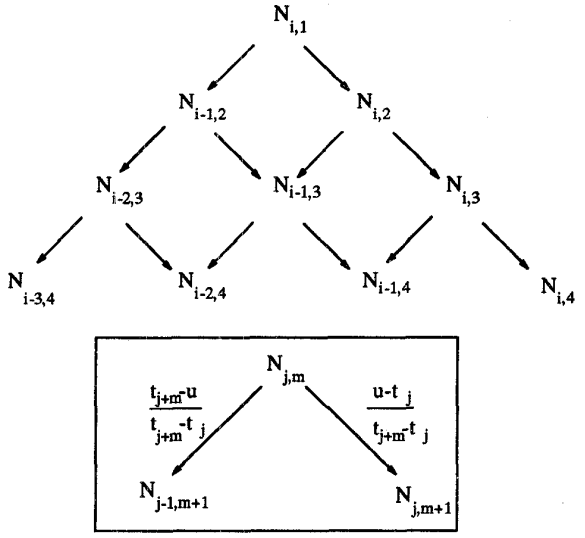

Figure 1: Basis Function Computation Graph

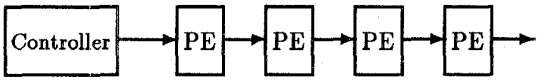

Figure 2: Basis Function Evaluation Array (BFEA) and the Controller

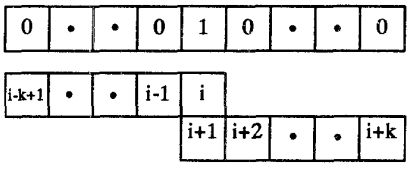

Figure 3: Pattern of Input to the first cell of BFEA

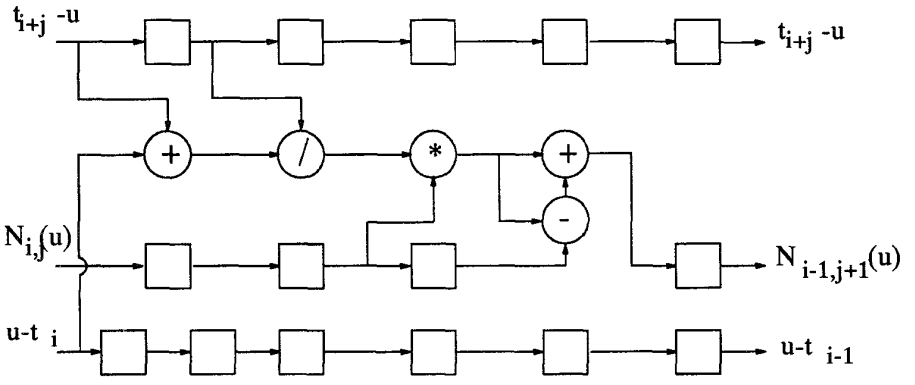

Figure 4: One Processing Cell of BFEA

cell of BFEA with its required knot values by the controller. From the Figure 1 and the method of computing the basis functions explained in the previous section, it would be clear that the first cell requires the knot pair t_i, t_{i+1}. The second cell requires one more pair t_{i-1}, t_{i+2} and so on. The $k-1$th cell requires, apart from the $k-2$ knot pairs used by its preceding cells, the knot pair t_{i-k+2}, t_{i+k-1}. Every processing cell communicates the knot pairs required by its succeeding cells. It can be seen that all the processing cells are equally loaded with k steps of computation and communication.

As there is no communication involved in the last cell, one useful basis function value is output every clock from that cell. As there is only one output line for the blending function, one useful result for every clock is the best we can achieve out of this linear architecture.

One *useful* result at every clock and equal load to all processors show that the scheduling algorithm used is the optimal scheduling algorithm for the given linear array architecture

Megson's architecture for the generation of basis functions generates all $n+1$ basis functions while this architecture generates only k basis functions. As $k << n$, the time required to generate the curve/surface is drastically reduced. Further, this fact makes our architecture independent of the number of control points (n).

By identifying the symmetry in the calculation of consecutive basis functions, the hardware required for each cell (refer Figure 4) is greatly reduced.

The pattern of input to the first cell is as shown in the Figure 3. There are three rows of input: one each for three input lines of the BFEA. The first row is the input to the line marked as $N_{i,j}(u)$ in the Figure 4 and the second and third rows specify the inputs to the two lines marked $u-t_i$ and $t_{i+j}-u$, respectively. The entries in the second and third rows just give the indices of the knots involved in the calculation. The third row starts with an offset such that the index $i+1$ coincides with the index i in the second row. Note that the first order basis function is 1 only at that particular clock when the indices i and $i+1$ coincide and is zero at all other times.

3.1 Time required for basis function generation

Each cell has a delay of five time units. The basis function of order 1 is pumped to the first cell at the kth clock. The time interval between the first input of a knot value and the generation of the corresponding basis function value of order k is

$$T_1 = k + 5(k-1) \tag{6}$$

The first term gives the time taken for the pumping of basis function of order 1. The second term gives the delay involved in $k-1$ cells before the first output.

Time required to get all the k outputs is

$$T_2 = 2k + 5(k-1) - 1 \tag{7}$$

The inputs for the calculation of the second set of k basis functions are timed such that it follows the output of the first set of k basis functions values. The input and output timings, if the orders of the consecutive set of basis functions to be calculated are different, are discussed in [5]. The design of the controller shown in the Figure 2 is also explained in detail in [5].

4 Architecture for the Computation of NURBS Curve

Using the above architecture to compute the basis functions, the procedure explained below computes the NURBS Curve. The BFEA generates and pumps the useful basis functions to an Accumulating Cell (AC). The homogeneous coordinates of the active control points (*i.e.* $x_i w_i, y_i w_i, z_i w_i, w_i$) are pumped to the AC by the controller. The numerator and the denominator of the Equation 4 are calculated simultaneously by multiplying the basis function values with $P_i w_i$ and w_i separately and adding these results independently in the AC. Finally the division is performed and the point on the curve is calculated. The product $P_i w_i$ is performed beforehand and is called a *weighted control point.*

To calculate the next point on the curve, the parametric value u is incremented, and the input to the BFEA is changed appropriately. Whenever u crosses

t_{i+1}, the index i is incremented and the new set of active weighted control points and their weights are pumped to the AC for the calculation of the point on the curve. This process continues till all the points on the curve have been computed.

4.1 Time required to calculate NURBS curve

Time required to calculate all basis functions is T_2. Delay involved in the AC for calculating the x coordinate is five time units [5]. Hence the time required to generate the x coordinate is

$$T_3 = (7k - 6) + 5 = 7k - 1 \quad (8)$$

The x coordinate of the second point on the curve is output k clocks after the x coordinate of the first point. If there are C points to be calculated on the curve, the time at which the x coordinate of the last point is output is

$$T_4 = T_3 + k(C - 1) = k(C + 6) - 1 \quad (9)$$

In subsequent clock pulses, the y and z coordinates of the last coordinate are also output.

Hence the total time required to calculate the whole curve is

$$T_5 = T_4 + 2 = k(C + 6) + 1 \quad (10)$$

Note that the above equation is independent of the number of control points n.

5 NURBS Surface Computation

Figure 5 pictorially represents the algorithm for the generation of a NURBS surface. In this figure the value of u lies between t_5 and t_6 and the value of v also lies between s_5 and s_6. (The knot vector in the u direction is represented by t and that in the v direction is represented by s). The value of k is 4 and that of l is 5.

Since for all the control points, except for those in the active control point grid, either or both the basis function values are zero, they need not be considered for the calculation of a point on the surface. Hence all the operations are done on the control points that are within the active control point grid.

¿In developing a VLSI architecture to calculate the NURBS surface, the above algorithm is slightly modified such that the number of multiplications are reduced.

The Equation 5 can be rewritten as follows.

$$P(u,w) = \frac{\sum_{i=0}^{n}(\sum_{j=0}^{m} P_{ij} w_{ij} N_{i,k}(u)) N_{j,l}(v)}{\sum_{i=0}^{n}(\sum_{j=0}^{m} w_{ij} N_{i,k}(u)) N_{j,l}(v)} \quad (11)$$

According to the algorithm presented above, every column of the control grid is multiplied with the useful basis functions shown below the grid. Then every row of this product is added to get a column of *virtual control points*. These virtual control points and their weights are represented by the terms inside the parenthesis in the numerator and the denominator of the

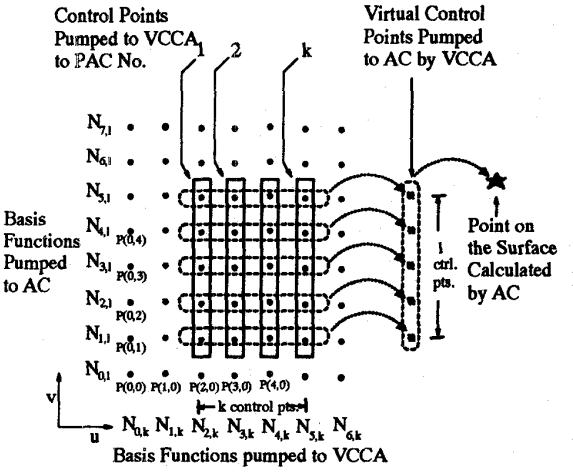

Figure 5: Algorithm used for Surface generation

Equation 11 respectively. This column of virtual control points is multiplied with the corresponding basis function values $N_{j,l}(v)$ shown in Figure 5. Then this column of elements is added to get a point on the surface.

5.1 Architecture for NURBS Surface Computation

We derive the architecture for the NURBS surface computation as a straightforward extension to the curve architecture. The architecture to calculate a NURBS surface is shown in Figure 6. The Virtual Control point Calculating Array (VCCA), shown in the figure, calculates the virtual homogeneous control points. This VCCA is an array of Partial Accumulating Cells (PACs). Each PAC is an inner product cell with a register to store the basis function value.

Initially the useful basis function values in the direction of u are calculated by BFEA and are pumped to the Virtual Control Point Calculating Array (VCCA). These k basis function values, which corresponds to one column each of the active control point grid, are stored one in each of the k PACs. The controller pumps to each PAC, one column of active control points from the active control point grid. Each control point is multiplied with the basis function value stored in the corresponding PAC register, added with the partial sum sent by the preceding PAC and the result is sent to the next PAC. The kth PAC outputs the homogeneous coordinates of the virtual control points to the AC. With these control points and the lth order basis functions from the BFEA, the AC calculates a point on the surface.

To calculate the next point on the surface, the value of the parameter v is incremented and the lth order basis functions are calculated by BFEA. As the value of u has not been changed, the same basis function values that are already in the internal registers of the VCCA are used. Depending on the present value of v the new active control point grid is found and pumped

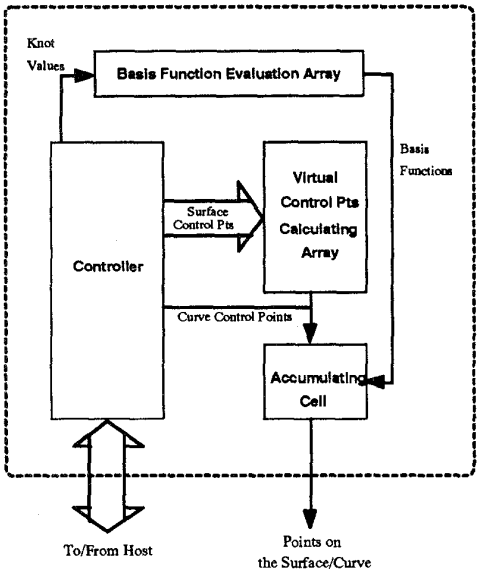

Figure 6: Architecture for the Computation of NURBS Surface

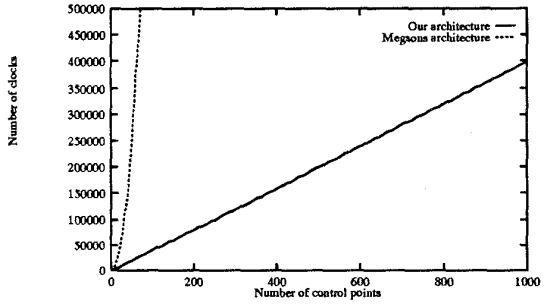

Figure 7: Curve Computation - Comparison

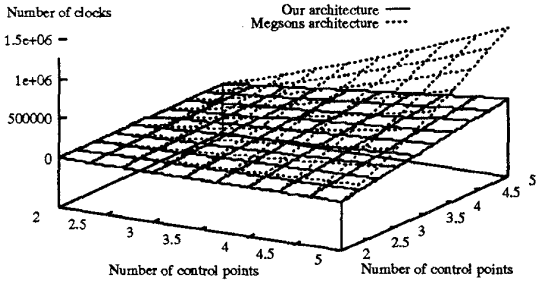

Figure 8: Surface Computation - Comparison

to the VCCA as before.

This continues till all the discrete values of v are considered. Then the value of v is reset to its initial value and u is incremented. The above explained process continues till all the values of u are considered. Note that only for every new value of u, k basis functions are calculated and these new values are stored in the PACs.

5.2 Time required to calculate NURBS patch

In the following calculation of time required to compute a NURBS patch, k is assumed to be equal to l. (The architecture can handle patches with unequal orders and the corresponding timing calculations are given in [5].) The time required to generate the NURBS patch can be calculated with respect to the input to the BFEA. In the input of the BFEA, a delay of one time unit is introduced to synchronize the arrival of the virtual control points and the lth order basis functions to the AC. Let us assume that C_1 number of discrete values of u and C_2 number of discrete values of v are to be considered.

Time required to pump the input to the BFEA for the last point on the surface would be

$$T_6 = C_1(k + 1 + C_2 l) - l \qquad (12)$$

Computation of one row of surface points requires, k inputs for the kth order basis function generation, followed by a delay of one time unit, then followed by C_2 times of l inputs for the generation of basis functions of order l. There are C_1 rows of surface points to be calculated. Hence the above quantity is multiplied by C_1. This term would give the total time taken for all the input including the last point on the surface. So to get the time at which the input for the computation of the last point starts, l is subtracted from the above quantity.

From the time T_6 the time taken to calculate the last point is given by $T_3 + 2$. Hence the total time required to calculate the whole surface is

$$T_7 = C_1(k + 1 + C_2 l) + 6l + 1 \qquad (13)$$

It can be seen that above equation is independent of the number of control points n and m.

6 Performance Evaluation

The architecture presented above decouples the size of the problem to the extent possible by making it independent of n and m. From the Equation 13 it is clear that the coefficient of C_1 is large, making the timing more dependent on C_1 than on C_2. Hence the proposed architecture performs well when the number of discrete values of u is less than the number of discrete values of v. Further, this architecture performs better when a whole curve or a surface is calcu-

lated than when a few discrete points on the surface are needed, because the algorithm makes complete use of inter-dependency and the information sharing between consecutive points on the curve/surface.

Let us now compare the performance of this architecture with that of the architecture proposed by Megson [11]. Assuming that the C_1 and C_2 are proportional to the n and m, the time required by the architecture proposed in this paper and by the architecture proposed in [11] to compute curve and surface are shown in Figures 7 and 8 respectively.

Analyzing the hardware complexity of the architecture proposed by Megson, it requires at most $5max(k,l)+3(max(m,n)+1)$ inner product cell equivalents and $3(m+1)(n+\delta+2)$ memory registers for surfaces with $(m+1)(n+1)$ control points and blending functions of degrees k and l and where $\delta = 5|k-l|$. The architecture presented in this paper requires at most $7max(k,l)+4$ inner product cell equivalents, $max(k,l) \times (5 + 4(max(k,l)))$ buffer registers and $4kl+k+l$ memory registers. It can be seen that both the processing element requirements and the memory requirements are much lesser than that of the Megson's architecture. Note that the hardware requirements specified here for this architecture is for the computation of NURBS whereas for Megson's architecture it is for the computation of just non-rational B-Splines.

7 Conclusion

To our knowledge the architecture presented here is the first unified architecture that can compute rational and non-rational uniform/non-uniform B-Spline curves and surfaces (When non-rational curves/surfaces are computed the weights of the control points are set to unity.). The above architecture is also shown to be better than the architectures, for some of the subproblems, proposed in the literature.

This architecture possesses characteristics that make it suitable for integration into the standard graphics pipeline of a graphics workstation: First, the architecture has a general linear structure with a small number of input lines and a single output line. Second, NURBS curves/surfaces are projection invariant. Hence, in the standard graphics pipeline, this architecture can be integrated after the transformation stage and before the clipping stage. The weights of the control point are suitably transformed when perspective projection of the curve/surface is performed. However, further work is needed to specify this integration more precisely.

References

[1] Cox, M.G. *The Numerical Evaluation of B-Splines*, J. Inst. Maths Applics., **10**, 1972, pp 134-149.

[2] de Boor, C. *Package for calculating with splines*, J. of Numerical Analysis,**14(3)**,1977, pp 441-472.

[3] De Rose, T. and Holman, T. *The Triangle: A multiprocessor architecture for Fast Curve and surface generation*, Tech Rep No 87-08-07, Dept of CS, FR-35, Univ. of Washington, Aug 1987.

[4] Foley, J.D., van Dam, A., Feiner, S.K. and Hughes, J.F. *Computer Graphics: Principles and Practice* Second Edition, Addition-Wesley Publishing Company Inc., pp. 491-500,522.

[5] Gopi, M. *Special Purpose architectures for B-Splines*, M.Sc. (Engg.) Thesis (under preparation), Indian Institute of Science, Bangalore.

[6] Gopi, M. and Manohar, S. *VLSI architectures for the computation of Uniform B-Spline curves*, accepted for publication in Microprocessing and Microprogramming.

[7] Gopi, M. and Manohar, S. *Parallel architecture for the computation of Uniform Rational B-Spline patches*, submitted for publication.

[8] Li, T., Smith, K.R. and Hanscon, D. *VLSI Systolic Architectures for Computer Graphics*, Proceedings of International Conference on Computers, Systems & Signal Processing, Bangalore, India, Dec 1984, pp 1527-1530.

[9] Mathias, P.C.*Systolic Architectures for Realistic 3D graphics*, Ph.D. Thesis, 1989,Indian Institute of Science, Bangalore.

[10] Mathias, P.C. and Patnaik, L.M. *Systolic Architectures in curve generation*, Computers and Graphics, **13**, No 4, 1989.

[11] Megson, G.M. *Systolic Algorithms for B-Spline Patch generation* Journal of Parallel and distributed computing,**11**, 1991, pp 231-238.

[12] Mortenson, M.E. *Geometric Modeling*, John Wiley & Sons, 1985 pp 125-149, pp 220-226.

[13] Riesenfeld, R.F. *Applications of B-Spline approximation to Geometric problems of Computer Aided Design*, Ph.D. dissertation, 1972, Syracuse Univ., Syracuse, NY. Also published as Univ. of Utah report UTEC-CSc-73-126.