A Unified Architecture for the computation of B-Spline Curves and Surfaces[1]

Meenakshisundaram Gopi Swami Manohar

Supercomputer Education and Research Centre

Indian Institute of Science

Bangalore 560012 INDIA

Keywords : NURBS, Graphics, Geometric Modeling, VLSI architecture.

Abstract

B-Splines in general, and Non-Uniform Rational B-Splines in particular, have become indispensable modeling primitives in computer graphics and geometric modeling applications. In this paper a novel high-performance architecture for the computation of uniform, non-uniform, rational and non-rational B-Spline curves and surfaces is presented. This architecture has been derived through a sequence of steps. First, a systolic architecture for the computation of the basis function values, the basis function evaluation array (the BFEA), is developed. Using the BFEA as its core, an architecture for the computation of non-uniform rational B-Spline curves is constructed. This architecture is then extended to compute NURBS surfaces. Finally, this architecture is augmented to compute the surface normals so that the output from this architecture can be directly used for rendering the NURBS surface.

The overall linear structure of the architecture, its small I/O requirements, its non-dependence on the size of the problem (in terms of the number of control points and the number of points on the curve/surface that has to be computed), and its very high throughput make this architecture highly suitable for integration into the standard graphics pipeline of high-end workstations. Results of the timing analysis indicate a potential throughput of one triangle with the normal vectors at its vertices, every two clock cycles.

[1]An extended abstract of this paper appears in [12]

1 Introduction

The explosive growth of computer graphics over the last two decades has been greatly facilitated by impressive hardware innovations: Raster graphics became popular due to the emergence of low-cost semiconductor memories for the frame buffer. Graphics co-processors and graphics display controllers designed with the goal of off-loading the graphics computation from the CPU resulted in the widespread availability of quality graphics cards for personal computers. The Geometry Engine [3] ushered in the era of high-performance graphics workstations. Successive generations of workstations have exploited increasing levels of pipelining and parallelism in the graphics pipeline (See for example, the Reality Engine [1]). The graphics applications however, have constantly increased their requirements so as to be continuously beyond the reach of available hardware capabilities. In this evolution, more and more complex graphics abstractions have been made available to the main processor: starting from simple lines, the current high-end systems support anti-aliased, Z-buffered, Gourard-shaded, 24 bits of color per pixel triangles in hardware.

We believe that the next step in this evolution is the migration of parametric curves and surfaces into hardware. An example of this trend is the recent microcode implementation of non-uniform rational B-Splines (NURBS) in a graphics workstation [6]. In this paper we present a unified architecture for the computation of various types of B-Spline curves and surfaces. We believe that this architecture is significant because of the following factors:

- This is the first solution to handle all types of B-Spline curves and surfaces.
- The architecture is capable of very high performance: one triangle with the normals at its vertices, every two clock cycles.
- The architecture has a linear structure that minimizes the number of pins required in a VLSI implementation.
- The architecture is independent of the size of the curve/patch (in terms of the number of control points, as well as the number of points on the curve/patch) to be computed.
- The above three features make this architecture highly suitable for integration into the graphics pipeline of high-end workstations.

We are focusing on B-Splines rather than many other parametric curves and surfaces that have been described in the literature, since NURBS has emerged as the modeling primitive of choice for the geometric design community. Non-uniform rational B-Spline curves and surfaces have been an Initial

Graphics Exchange Specification (IGES) standard since 1983 [14]. Many commercial or in-house modeling applications like Geomod and Proengineer are based on rational B-Spline representations.

The popularity of rational B-Splines is due to the following facts:

- They provide one common mathematical form for the accurate representation of standard analytic shapes, especially conics, as well as free-form curves and surfaces. Thus unification of all forms of curves and surfaces is done by NURBS.
- They offer an extra degree of freedom in the form of weights, apart from knot vector and control points, which can be used in designing wide variety of shapes.
- They are projection invariant: That is, the projection of the curve is achieved by projecting the control points and suitably modifying the weights of the control points.

Introduction to rational quadratic representation of conics can be found in [15]. Further information about the NURBS representation of circles is in [21],[24].

A few hardware implementations of B-Splines have been reported in the literature. One of the early papers in this direction is the work of T.Li *et al.*[16] where an architecture to generate Bezier curves and patches was proposed. De Rose [7] *et al.*, proposed a triangular architecture to generate B-Spline curves using the deBoor-Cox algorithm. Mathias [17], 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 [18]. Recently, Megson [19] 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. All the architectures presented in the literature have the following limitations that seriously restrict their practical implementation.

- the size of the hardware is tied to the size of the problem (the number of control points) that is to be solved.
- a large number of I/O pins are needed.

The architecture proposed in this paper overcomes these limitations and in addition, as pointed out earlier, provides a unified high-performance solution to the computation of B-Spline curves and surfaces.

In the next section, the fundamentals of B-Splines are explained. The properties of all types of B-Splines as well as their computation requirements are outlined in this section. An efficient algorithm to compute basis functions and its hardware implementation are presented in Section 3. Using

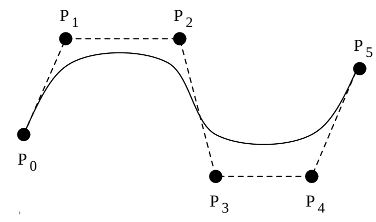

Figure 1: B-Spline Curve

the above basis function calculating architecture, a unified architecture to calculate Uniform/Non-Uniform Rational/Non-Rational B-Spline Curves and Surfaces is presented in Section 4. Finally, the architecture presented in this work is compared with other similar solutions proposed in the literature and is shown to be superior both in time and space efficiency.

2 Theory of B-Splines

The B-Spline curve is defined over a parameter u by the following equation

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

The curve is drawn for various values of u varying from u_{min} to u_{max}. Typically $u_{min} = 0$ and $u_{max} = 1$. 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. These control points are points in object space, using which the shape of the B-Spline curve can be controlled. In geometric modeling applications, the position of these control points are changed to achieve the required shape of the curve. The curve need not pass through the control points, though B-Spline curves always lie within the convex hull of control points. A typical B-Spline curve is shown in Figure 1. The basis function or the blending function is denoted by $N_{i,k}(u)$. These basis functions will decide the extent to which a particular control point controls the curve at a particular parametric value u. The parameter k is called the order (one more than the degree) of the curve. For a cubic curve, $k = 4$.

The basis function $N_{i,k}(u)$, depends on the parametric value and the order of the curve, and is recursively defined as follows.

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

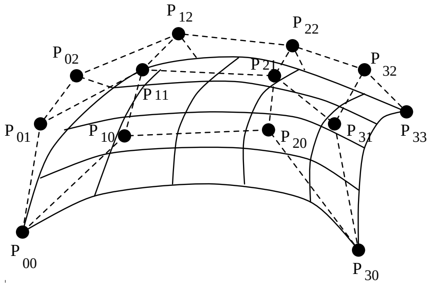

Figure 2: B-Spline Surface

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

The constants t_is, called *knot values*, are specific instances of the parametric value u and are strictly in non-decreasing order. There are $(n + k + 1)$ knot values (t_0 to t_{n+k}). All the knot values put together is called a *knot vector*. In section 2.1 we will see more about the knot vector and the properties of the basis functions.

The properties of basis functions and B-Splines are discussed in detail in [22]. For further reading on B-Splines [2][4][20] and [23] are suggested.

The extension of a B-Spline curve is the B-Spline surface given by

$$P(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} N_{i,k}(u) N_{j,l}(v), \tag{4}$$

where, $P(u, v)$ is the point on the surface for the parametric values u and v. The grid of $(n + 1) \times (m + 1)$ control points is denoted by P_{ij}. The basis functions in u and v directions are denoted by $N_{i,k}(u)$ and $N_{j,l}(v)$. The variables k and l denote the orders of the surface in the direction of u and v respectively. For a bi-cubic patch, $k = l = 4$. Figure 2 shows a bi-cubic B-Spline patch.

A Rational B-Spline curve is a normalized result in 3D, of a 4D non-rational B-Spline curve, defined by 4D control points. A Rational B-Spline curve is defined by the formula,

$$P(u) = \frac{\sum_{i=0}^{n} P_i w_i N_{i,k}(u)}{\sum_{i=0}^{n} w_i N_{i,k}(u)}. \tag{5}$$

The term w_i denotes the weight of the 3D control point P_i. The term $\frac{w_i N_{i,k}(u)}{\sum_{i=0}^{n} w_i N_{i,k}}$ denotes the extent to which the control point P_i has control over the curve. 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 influence over the curve. Detailed study of rational B-Spline was carried out first by Versprille [26]. More details about rational B-Splines can be found in [22], [24], [25].

A rational B-Spline surface is defined by the formula,

$$P(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij} w_{ij} N_{i,k}(u) N_{j,l}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{ij} N_{i,k}(u) N_{j,l}(v)} \tag{6}$$

Here again, the term $\frac{w_{ij}N_{i,k}(u)N_{j,l}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{ij}N_{i,k}(u)N_{j,l}(v)}$ denotes the extent of influence of the control point P_{ij} on the patch. There is a grid of $(n+1) \times (m+1)$ control points for the surface and with every control point is associated the weight of that control point.

2.1 Basis Functions and Control Points

In this section, we point out a few interesting aspects of the basis functions and the control points. These properties will be used later in this paper.

As the basis functions are dependent on the knot vector, we describe the knot vector in more detail. Let the parameter u vary from 0 to 1. Let the knot vector be [0.0, 0.1, 0.13, 0.3, 0.35, 0.35, 0.35, 0.4, 0.6, 0.7, 0.9, 1.0], and let the order of the curve k be 4 (cubic curve). The basis function curves, for this knot vector and order, is shown in Figure 3. As shown in the figure, the knot values are specific instances of u as it varies from its minimum value to the maximum value. Note that the knot values can be repeated as given in the example above, where $t_4 = t_5 = t_6 = 0.35$. The basis function for the control point P_0, namely $N_{0,4}$ will be non-zero for the values of u between $t_0 = 0.0$ and $t_4 = 0.35$. At all other values of u, this basis function will remain zero. The basis function $N_{1,4}$ will be non-zero only for the values of u from t_1 to t_5. In general, the basis function $N_{i,k}$ will be non-zero for the values of u from t_i to t_{i+k}. It can be shown that at any particular value of u in the valid range, there will be k and only k basis functions with non-zero values [8]. The valid range of u is from t_{k-1} to t_{n+1}. In our example, the valid range is from $u = 0.3$ to $u = 0.6$. Those basis functions with non-zero values for the value of u under consideration, are called *useful basis functions*. We will use this concept of useful basis functions later in this section to reduce the computational complexity of B-Splines.

We can now impose various restrictions on the knot vector. These restrictions give raise to various kinds of B-Splines. If the knot vector is such that, $t_1 - t_0 = t_2 - t_1 = t_3 - t_2 = \cdots = t_{n+k} - t_{n+k-1}$ then the resultant B-Spline is called an Uniform B-Spline [8]. A typical uniform knot vector would be [0.0, 0.1, 0.2, 0.3, $\cdots$, 0.9, 1.0]. Here we can see that the differences between the adjacent knot values are equal. For a Uniform B-Spline all basis function curves are identical (Figure 4). Hence it is enough to calculate the basis function curve only once, thus reducing the complexity of the

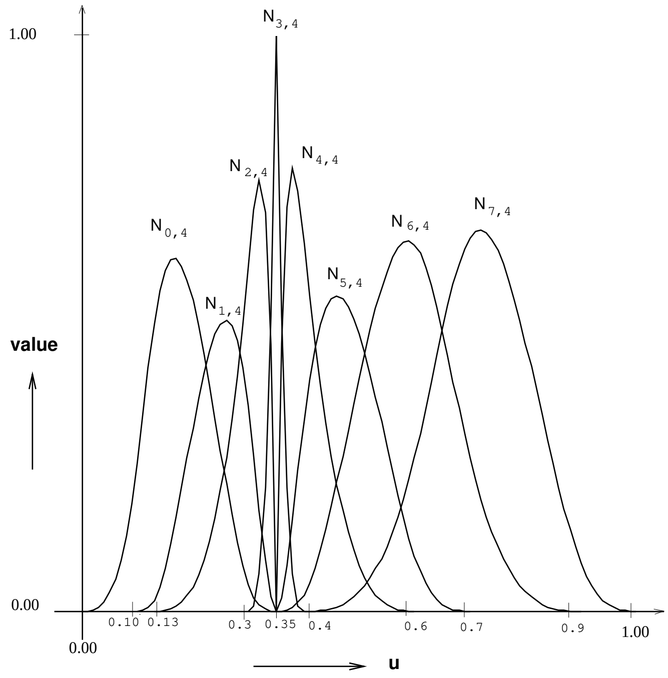

Figure 3: Basis Function Curves for Non-Uniform Knot Vector

B-Spline computation to a large extent. Taking advantage of this fact, a VLSI architecture to solve Uniform B-Spline curves has been proposed in [10] and a unified architecture to compute uniform rational/non-rational B-Spline curves/patches is also proposed in [11].

In the uniform knot vector, if the first and the last knot values are repeated k times then the resultant knot vector is called the Open knot vector. This forces the curve to start from the first control point and end in the last control point.

If the knot vector does not conform to the above two conditions then it is called a Non-Uniform knot vector and the B-Spline is called a Non-Uniform B-Spline. The example given in the beginning of this section is a non-uniform knot vector. Unlike uniform B-Splines, the basis function values for the B-Spline with non-uniform knot vector is to be computed for every value of the parameter. The rationalized B-Spline curves and surfaces with Non-Uniform knot vector are called NURBS (Non-Uniform Rational B-Spline) curves and surfaces.

Having described knot vectors and the types of B-Spline curves and surfaces, we now proceed to analyze the properties of the basis functions. It is clear from the equations of B-Spline curves and surfaces that only if the basis function value is non-zero, the corresponding control point controls

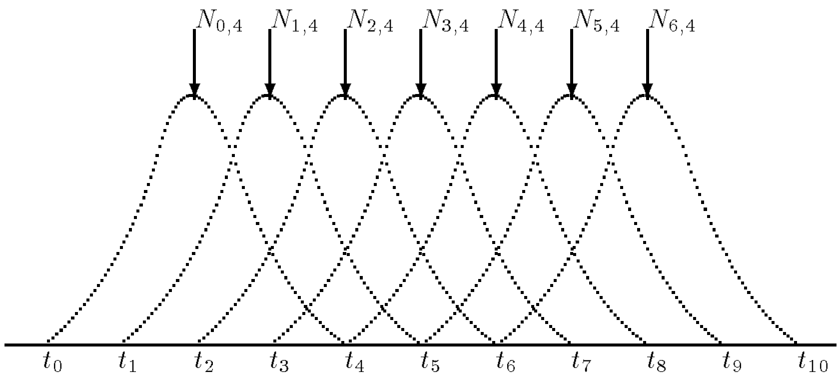

Figure 4: Basis Function Curves for Uniform Knot Vector

the shape of the curve. As there are only k useful basis functions at any value of u, as discussed earlier, only k control points contribute to the computation of the curve. These control points are called *active control points*. In case of a B-Spline surface, there will be a grid of $(k \times l)$ active control points.

If we know the useful basis functions at a particular value of u, we can compute only those basis function values and compute the point on the curve. The other basis functions and their corresponding control points need not be considered. Hence the complexity of the computation of the curve is dependent only on k, the order of the curve, and not on n, the number of control points of the curve. It should be noted that no approximations have been used in bringing down the complexity from $O(n)$ to $O(k)$. Instead, a careful study of the basis functions has led to the elimination of useless computations from the naive computation scheme.

So our problem boils down to finding the useful basis functions given the value of u. We use the fact that the knot values are strictly in non-decreasing order and any value of u is thus, clearly sandwiched between a pair of consecutive knot values. (This is a very important observation which is used in unfolding the recursion in the basis function computation. This observation leads to the conclusion that there is only one first order basis function with value 1, and the rest have value 0.)

If $t_i \leq u < t_{i+1}$ then the k useful kth order basis functions are $N_{i-k+1,k}$, $N_{i-k+2,k}, \cdots$, $N_{i,k}$. Using the first subscript of these useful basis functions, the active control points can also be found. In our example, if $u = 0.5$, then it lies between $t_7 = 0.4$ and $t_8 = 0.6$, and the useful basis functions are $N_{4,4}$, $N_{5,4}$, $N_{6,4}$ and $N_{7,4}$, which have non-zero value. Thus the active control points are P_4, P_5,

P_6 and P_7. If u crosses t_{i+1}, then the index i is incremented and the current useful basis functions and the active control points are found.

Similarly, for a surface, with order k and l, there are k and l useful basis functions for particular values of u and v respectively. Here again, the grid of $k \times l$ active control points can be found using the above method.

3 VLSI architecture for Basis Function Generation

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

The calculation of one point on the curve, requires $n + 1$ basis function values. Hence the total number of calls to the basis function routine would be $(n + 1)(2^k - 1)$. Using the discussion in the preceding section that there are only k useful basis functions, the number of calls reduces to $k(2^k - 1)$.

There are two problems in using the recursive equation 3 for the computation of basis functions. First, in the computation of these k basis functions, many lower order functions return zero. This observation shows that there are few computations that can be eliminated. Second, if there are multiple knots (such as, $t_i = t_{i+1} = t_{i+2} = t_{i+3}$), then the denominator of the Equation 3 may become zero for certain calls to the basis function routine (such as $N_{i,1}$, $N_{i+1,1}$, $N_{i+2,1}$, $N_{i,2}$, $N_{i+1,2}$, $N_{i,3}$), leading to division errors.

The above two difficulties are overcome by using the following method ([4][5]), which computes only those basis function values, including the lower order basis functions, which have non-zero value. This algorithm just unfolds the recursion and identifies a directed acyclic graph (DAG) of basis functions which have non-zero values. This DAG, while sorted in topological order, would give the order of computation of basis function values that eliminates the above two difficulties.

From the discussion in the previous section, we know that, for a given value of 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 5, we can see that from the non-zero first order basis function, k kth order basis functions can be calculated. The multiplicative factor along the edges is given in the inset and whenever two edges meet an addition is performed to get the basis function value at the meeting

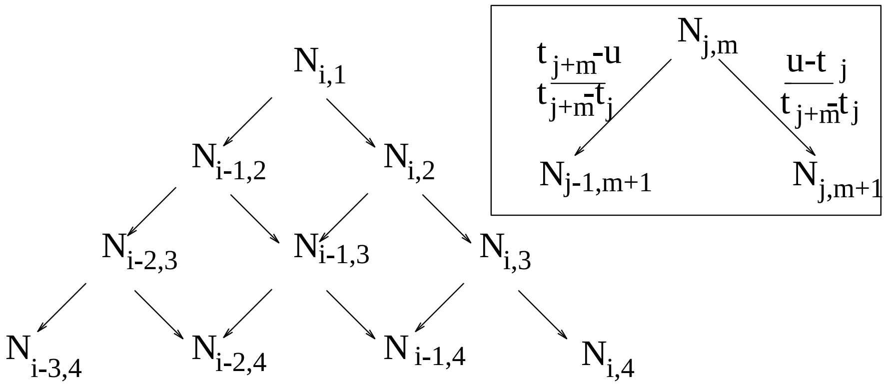

Figure 5: Basis Function Computation Graph

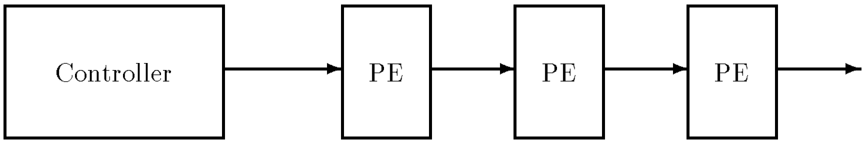

Figure 6: Basis Function Evaluation Array (BFEA) and the Controller

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.

3.1 Systolic computation of basis function

Figure 6 shows the systolic linear array for the computation of the basis function. The controller pumps the required input for the computation of basis functions to the first cell in the Basis Function Evaluation Array (BFEA). The design of BFEA and its input pattern are explained below.

Each cell in the BFEA computes one level in the DAG, shown in the Figure 5. The first level which has only one element $N_{i,1}(u)$ involves no computation as its value is always one for any i such that $t_i \leq u < t_{i+1}$. This value is pumped by the controller to the first cell in the BFEA.

Starting from the second level, one processing element is assigned the job of computing one level of the DAG. Hence it is required to have just $k-1$ processing elements. From the $(k-1)$th cell of the BFEA, the k basis functions of order k are output.

From the Figure 5 and the method of computing the basis functions explained in the previous section, it is 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}. The last cell receives $k-1$ knot pairs but outputs k blending function values. Hence one dummy pair is input to make the number of input and the output quantities the same. This dummy pair will follow the same path as that of other knot pairs.

Since this is a linear architecture, the data required by the cells downstream, has to be passed on by the controller, through the cells upstream. Thus, the ith processing cell performs $i+1$ steps of computation and $k-i-1$ steps of work to communicate the knot values to the cells downstream. Thus the total work by each cell is k. The $k-1$th cell will output one useful basis function value every clock cycle after the initial pipeline fill. Thus the whole process of computation and communication, proceeds systolically, through this linear architecture.

The next section elaborates the issues involved in designing each cell in the above Basis Function Evaluation Array.

3.2 Design of a cell in BFEA

One cell in the BFEA, with its functional units, is shown in Figure 7. We present below the mathematics behind the design of the core components of this cell.

One cell in the BFEA, say $(j-1)$th cell, computes basis functions $\cdots N_{i-1,j}$, $N_{i,j}$, $N_{i+1,j}, \cdots$ in sequence after receiving $\cdots N_{i-1,j-1}$, $N_{i,j-1}$, $N_{i+1,j-1}, \cdots$ and the knot pairs.

Hardware requirements of each cell can be greatly reduced by identifying the symmetry in the calculation of basis functions and suitably modifying the algorithm. This can be done in the following ways. We know that

$$N_{i-1,k} = \frac{(u - t_{i-1})N_{i-1,k-1}}{t_{i+k-2} - t_{i-1}} + \frac{(t_{i+k-1} - u)N_{i,k-1}}{t_{i+k-1} - t_i} \tag{7}$$

$$N_{i,k} = \frac{(u - t_i)N_{i,k-1}}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)N_{i+1,k-1}}{t_{i+k} - t_{i+1}} \tag{8}$$

$$N_{i+1,k} = \frac{(u - t_{i+1})N_{i+1,k-1}}{t_{i+k} - t_{i+1}} + \frac{(t_{i+k+1} - u)N_{i+2,k-1}}{t_{i+k+1} - t_{i+2}} \tag{9}$$

Let us consider the computation of $N_{i,k}$ (Equation 8). The first term in the RHS of Equation 8, can be decomposed as follows.

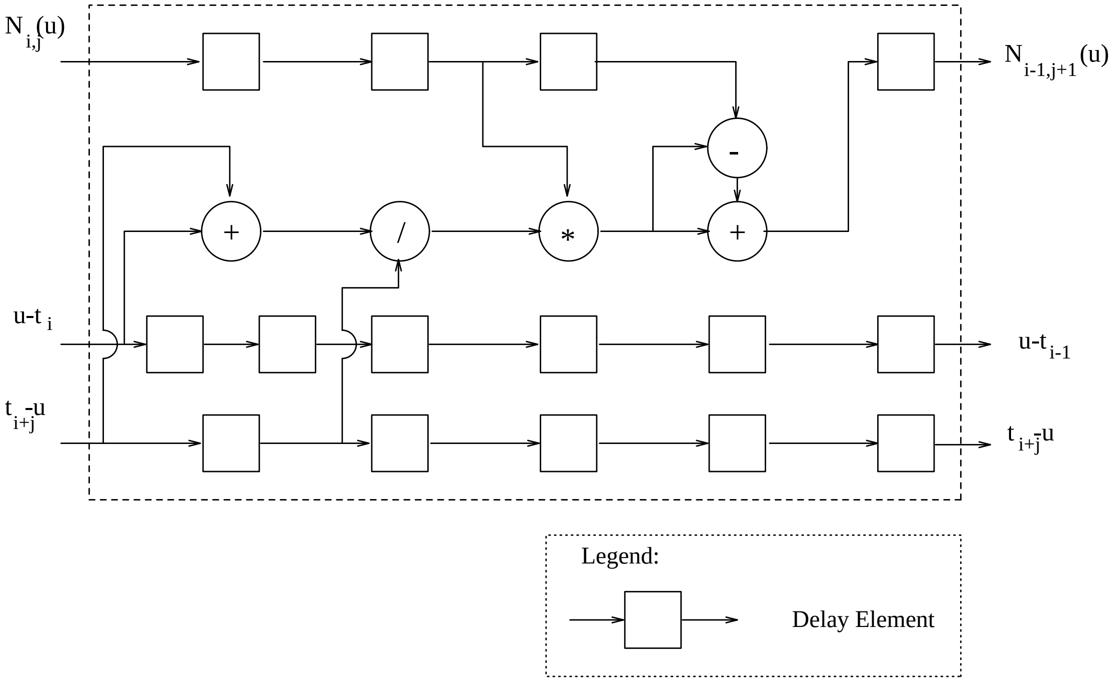

Figure 7: One Processing Cell of Basis Function Evaluation Array

$$\frac{(u-t_i)N_{i,k-1}}{t_{i+k-1}-t_i} = \frac{((t_{i+k-1}-t_i)-(t_{i+k-1}-u))N_{i,k-1}}{t_{i+k-1}-t_i} \tag{10}$$

$$= N_{i,k-1} - \frac{(t_{i+k-1}-u)N_{i,k-1}}{t_{i+k-1}-t_i} \tag{11}$$

1. It can be seen that the second term in the RHS of Equation 11, and the second term in Equation 7 are the same. Further the first term in Equation 11 is computed prior to $N_{i,k}$. Thus we can make use of pre-computed values, and get the first term of Equation 8, by just one mathematical operation, instead of four. Similarly the first term in the RHS of Equation 9, is calculated using the second term in the RHS of Equation 8. The subtractor of the adder-subtractor unit in Figure 7 computes the Equation 11 while Equation 8 is computed by the adder.

2. If we communicate t_{i+k}, t_i and u to calculate $(t_{i+k}-t_i)$ and $(u-t_i)$ then we require two subtractors. If $(t_{i+k}-u)$ is sent instead of t_{i+k} and $u-t_i$ is sent instead of t_i and as $(t_{i+k}-t_i) =$

$(t_{i+k} - u) + (u - t_i)$, two subtractors can be reduced to one adder. Further a separate line required to communicate the parametric value u is avoided.

From the data dependency analysis of the computation of basis functions it can be seen that each cell requires a minimum of five stages. The above observations are faithfully implemented in the processing cell. The design of the rest of the cell can be derived directly from the equation for the basis function computation.

The input to various processing cells at different time units are shown in the Figure 8. The figure shows the input to various cells for calculating k basis function values for two different values of u. The row gives the data at a specific point in space, at various time intervals and the column shows the data distribution in space at a specific time instant. Figure 8 has three data inputs for each cell: one each for three inputs of the BFEA. The first row is the input to the line marked as $N_{i,j}(u)$ in the Figure 7. The second and third row corresponds to the two inputs marked $u - t_i$ and $t_{i+j} - u$ in the Figure 7 respectively. The entry in the second and third rows just give the indices of the knots. For example, the entry $i - k + 1$ in the second row refers to the input $u - t_{i-k+1}$ and the entry $i + 1$ in the third row refers to the input $t_{i+1} - u$. The pattern of input to the first cell at various time instances is as shown in the Figure 9. The same pattern can be seen in Figure 8, for $k = 4$, from time 1 to 7 for input i, and from time 5 to 11 for input j. Note that the first order basis function is 1 only at that particular clock cycle when the indices i and $i + 1$ coincide and it is zero at all other times. The input to the first cell also includes t_{i-k+1} and t_{i+k} which are used as dummy inputs.

3.3 Time required for basis function generation

We assume for the sake of simplicity, throughout this paper, that all functional units take equal amount of time, namely, one time unit.

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

$$T_1 = k + 5(k - 1) \tag{12}$$

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

$$T_2 = 2k + 5(k - 1) - 1 = 7k - 6 \tag{13}$$

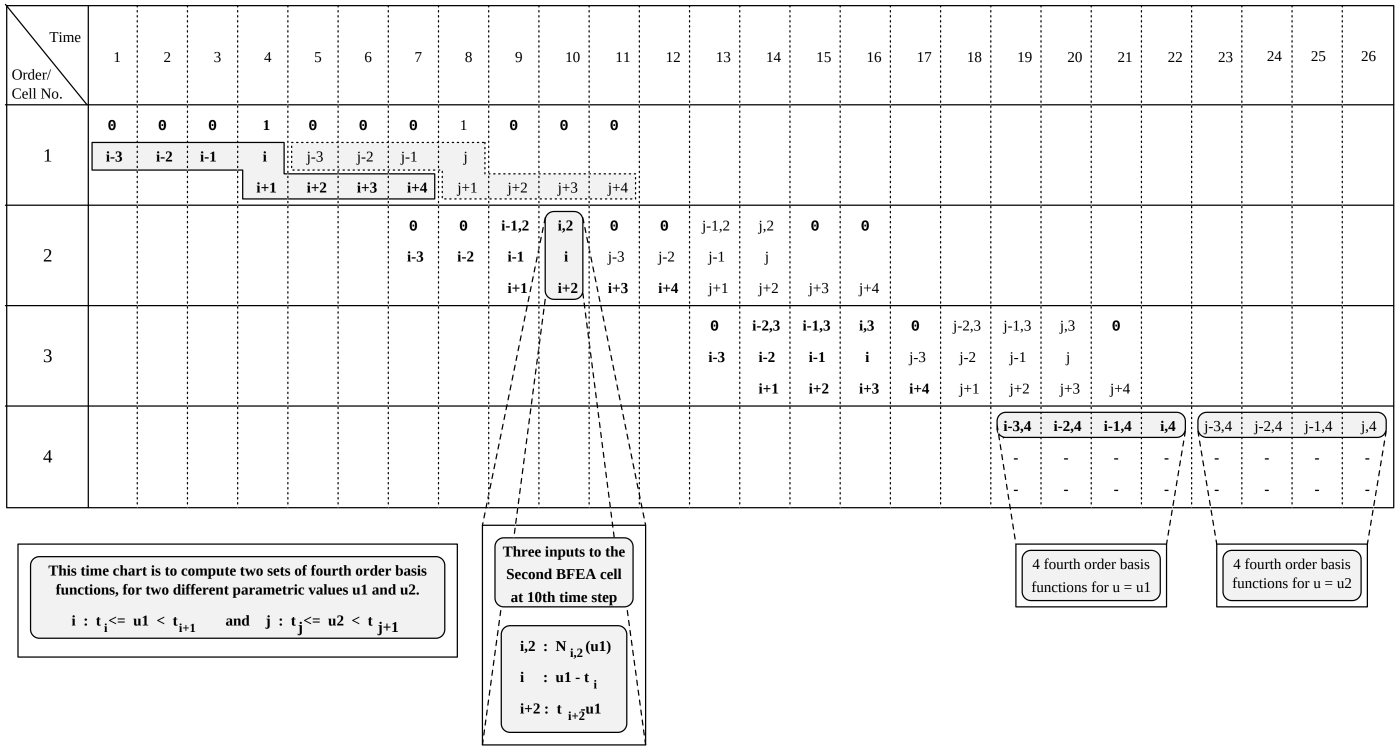

Figure 8: Input scheduling to various processing cells of BFEA

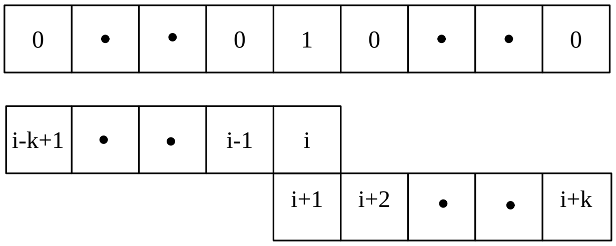

Figure 9: Pattern of Input to the first cell of BFEA

Figure 8 shows the input pattern for the computation of two sets of basis functions of the same order in succession. Such a case is true only in the computation of B-Spline curves. When a B-Spline surface is calculated, the orders in different directions of the parametric value u and v, need not be the same. If they are of different orders, the output is to be taken from two different cells of the BFEA. In this case, the input is timed in such a way so as to ensure that the data in the output line of the BFEA is not corrupted with two basis function values.

It can be seen from Figure 8 that the four useful basis functions are output for $u = u1$, from time 19 to 22. This is *immediately* followed by the second set of basis functions. Thus, one useful basis function is output every clock cycle. This is the best we can achieve out of this linear architecture, as there is only one output line.

Further, as BFEA forms the core of the NURBS architecture, the optimizations adopted in this design, have a direct impact on the design of the final architecture. For example, we are computing just the k non-zero *useful* basis functions, whereas, the solution proposed by Megson [19], computes all the $n+1$ basis functions. As $k << n$, the time required to generate the curve/surface is drastically reduced. Further, this fact, apart from making our architecture independent of the number of control points (n), also renders the time taken to compute the curve/surface, independent of n.

4 VLSI architecture for NURBS curves and surfaces

We describe in this section how the BFEA is effectively deployed to compute curves and surfaces of the most general form of B-Spline – the Non-Uniform Rational B-Spline.

4.1 NURBS curve computation

The NURBS curve computation is given by,

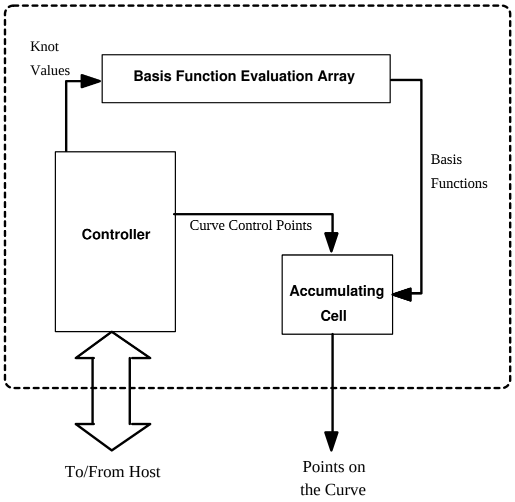

Figure 10: Architecture for the computation of NURBS Curve

$$P(u) = \frac{\sum_{i=0}^{n} P_i w_i N_{i,k}(u)}{\sum_{i=0}^{n} w_i N_{i,k}(u)}. \tag{14}$$

The architecture proposed for its computation is as shown in Figure 10. As seen in the previous section, the BFEA gives one useful basis function value every clock cycle after the initial set-up time. The numerator and the denominator of Equation 14 are calculated simultaneously by multiplying these *useful* basis function values with $P_i w_i$ and w_i separately, and summing these results independently in the Accumulating Cell (AC). The control points P_is are *active* control points and w_i are their corresponding weights. Finally the division is performed within the AC itself and the point on the curve is calculated. The product $P_i w_i$ is performed beforehand and is called a *weighted control point*. The weighted control points and w_i are pumped by the controller to the AC, synchronizing with the basis functions input.

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 sent to the AC. This process continues until all the points on the curve have been computed.

4.2 Time required to calculate a NURBS curve

As seen from Equation 13 the time required to calculate all the basis functions is $7k-6$. Delay involved in the AC for calculating the x coordinate is five time units. Hence the time required to generate the x coordinate of the first point is

$$T_3 = (7k-6)+5 = 7k-1 \tag{15}$$

In subsequent clock cycles, the y and z coordinates are output.

The x coordinate of the second point on the curve is output k clock cycles 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

$$T_4 = T_3 + k(C-1) = k(C+6)-1 \tag{16}$$

In subsequent clock cycles, the y and z coordinates of the last coordinate are also output. Hence the total time required to calculate the whole curve is

$$T_5 = T_4 + 2 = k(C+6)+1 \tag{17}$$

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

4.3 NURBS Surface Computation

The above architecture for the curve computation can be easily extended to compute NURBS surfaces. In Section 3, we started with an argument that the time taken to compute the basis function is the dominant factor in the computation of the curve or surface when compared with the inner product operation. However, in this section we will see that this inner product computation is to be speeded up if it is to cope with the output rate of BFEA. It would also be obvious at the end of this paper that the inner product computation cannot be speeded up arbitrarily, thus making any more attempts to improve the basis function computation useless.

The Equation 6 can be rewritten as follows.

$$P(u,w) = \frac{\sum_{j=0}^{m}(\sum_{i=0}^{n} P_{ij} w_{ij} N_{i,k}(u)) N_{j,l}(v)}{\sum_{j=0}^{m}(\sum_{i=0}^{n} w_{ij} N_{i,k}(u)) N_{j,l}(v)} \tag{18}$$

The terms inside the parenthesis in the numerator and in the denominator are called virtual control points, and the weights of the virtual control points respectively. The above equation can be viewed

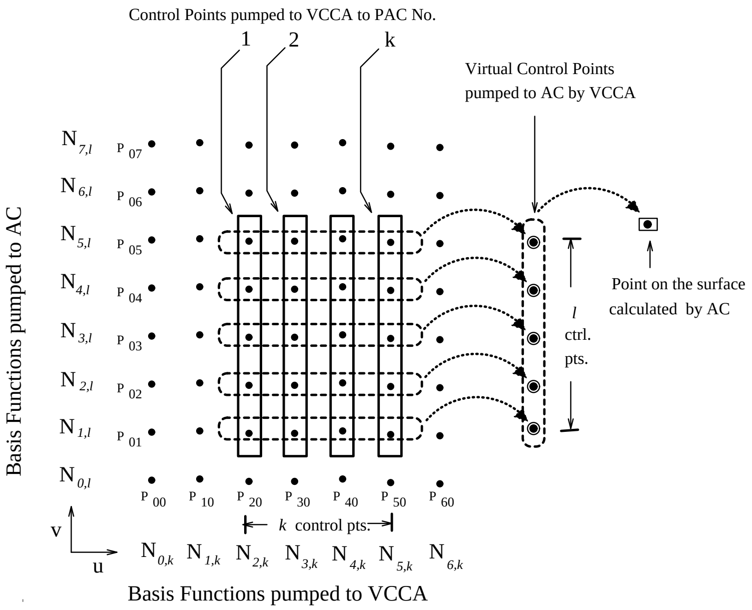

Figure 11: Algorithm for Surface generation

as the equation of a NURBS curve with the virtual control points and their weights playing the role of control points and their weights. Hence the architecture presented here initially computes the virtual control points and its weights. These values are used in the NURBS curve architecture to compute the NURBS surface. The architecture to calculate a NURBS surface is shown in Figure 12.

The NURBS surface has two orders associated with it — k, in the direction of u, and l, in the direction of v. Extending the arguments presented in Section 2, it is clear that for a given value of the parameters u and v there are only $k \times l$ active control points that correspond to the non-zero basis function values. The entities involved in the computation of one point on the surface are shown in Figure 11.

The grid of $k \times l$ active control points is loaded in columns of l values (as shown by the vertical rectangular boxes in Figure 11) in to the Partial Accumulating Cells (PACs) of the Virtual Control point Calculating Array (VCCA). The same BFEA is used to compute the basis function values for both the parametric values. Initially the kth order *useful* basis function values in the direction of u are calculated and are pumped to the VCCA (Figure 13), by the BFEA. As the name implies, the VCCA, computes the virtual control points and its weights. The VCCA computes the virtual control points by taking the dot product of a row of k active weighted control points (shown by horizontal dotted boxes in Figure 11) with the corresponding basis function values.

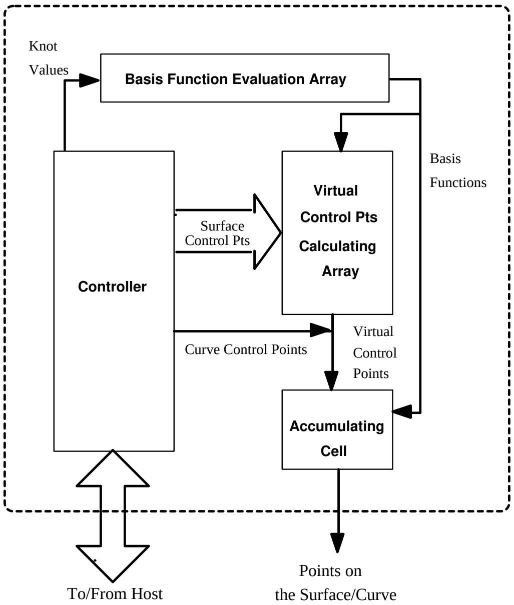

Figure 12: Architecture for the computation of NURBS Surface

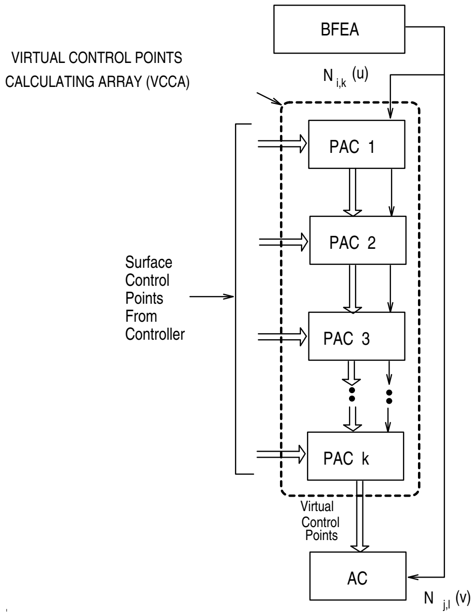

Figure 13: Virtual Control point Calculating Array

Once the virtual control points and their weights have been computed the problem of calculating a NURBS surface, boils down to a problem of computing a NURBS curve. The AC instead of getting the curve control points from the controller, gets the virtual control points from the VCCA. With these control points and the lth order basis functions directly from the BFEA, the AC calculates the point on the surface.

The points on the surface are computed column by column. A column of points is an isoparametric curve on the surface for a constant value of u and all discrete values of v. As u is constant for a particular column, the basis functions along u, once computed and stored in PACs, can be reused. The active control point grid is found for every u-v pair, and the weighted control points in the computed grid are sent to VCCA, for virtual control point computation. The lth order basis functions for the new value of v are calculated by the BFEA. The basis functions and the virtual control points are taken by the AC and the next point on the surface is generated. This process continues until all the discrete values of v are considered and an isoparametric curve for a particular value of u has been computed. Then the value of v is reset to its initial value and u is incremented. The new set of basis functions for u are computed as before and stored in the internal registers of PAC in VCCA. The above algorithm continues until all the discrete values of u have been considered.

Note that although every point on the surface requires the computation of both k and l basis function values in the direction of u and v, the k basis function values are computed only for every new value of u. These values are stored and used for the whole curve drawn for a particular value of u and for all values of v.

4.4 Time required to calculate a NURBS patch

The time required to generate a NURBS patch can be calculated with respect to the input to the BFEA.

If basis functions of different orders say k and l are computed one after the other, then a delay of β is to be introduced in the input of BFEA to ensure that the output line of BFEA is not corrupted with two basis function values. The following table shows the values of β and the delay in output experienced under various conditions.

Condition	Delay in the Input β	Delay in the Output
$k < l$	0	$5(l-k)$
$k = l$	0	0
$k > l$	$6(k-l)$	0

Further, when the second set of basis functions immediately follows the first set, there is a synchronization problem in the AC between the virtual control points and the second set of basis functions. This is because of the fact that the basis functions are computed earlier than the inner product computation. Hence an additional delay of α is introduced in the input of the BFEA. The value of α can be shown to be one when $k \geq l$, and zero otherwise. This shows that the performance of the system now depends on the inner product computation and not on the computation of basis functions.

Let us assume that C_u represents the number of discrete values of u and C_v the number of discrete values of v. Thus there are C_u isoparametric curves (constant u) with C_v points computed in each curve.

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

$$T_6 = C_u(k + \alpha + \beta + C_v l) + \beta(C_u - 1) - l \tag{19}$$

Each isoparametric curve computation requires, k inputs for the kth order basis function generation, followed by a delay of $(\alpha + \beta)$ time units, then C_v times l inputs for the generation of basis functions of order l. The first term gives the time to compute C_u such curves. The second term is the delay introduced between the computation of these curves. This delay is introduced for $C_u - 1$ times during the calculation of the whole surface. These two terms put together 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 $7l + 2$ (from the results of $T - 3$). Hence the total time required to calculate the whole surface is

$$T_7 = T_6 + 7l + 2 \tag{20}$$

$$= C_u(k + \alpha + \beta + C_v l) + \beta(C_u - 1) + 6l + 1 \tag{21}$$

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

4.5 Computation of normals to a NURBS Surface

In the standard graphics pipeline, when primitives like triangles are pumped, the vertices with their normal vector are required. These normal vectors are used in the lighting calculations. This section extends the above architecture to include the computation of surface normals also. Thus the NURBS surface can be tessellated into triangles, and the actual normal vectors at the vertices of the triangle can be computed using this architecture.

To calculate the normal to the NURBS surface at a point $P(u^{'}, v^{'})$, the tangent vector of the curve $P(u^{'}, v)$ $(= P^{v}(u^{'}, v^{'}))$ with respect to v at the point $v^{'}$ is computed. Then the tangent vector of the curve $P(u, v^{'})$ $(= P^{u}(u^{'}, v^{'}))$ with respect to u at the point $u^{'}$ is computed. The cross product of the above two quantities would give the normal vector to the surface at the point $P(u^{'}, v^{'})$.

Clearly, from the equation of the NURBS surface, the tangent vector calculation would involve the computation of the (first) derivative of the basis functions.

4.5.1 Computation of Basis Function Derivatives

As seen from the equation of the basis function, Equation 2, the derivative of first order basis function is zero. Further as the sum of the basis function values at a particular parametric value is one, the sum of their derivatives is zero. The derivative of the basis function is given by

$$N^{'}_{i,k}(u) = \frac{(u - t_i)N^{'}_{i,k-1}(u) + N_{i,k-1}}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)N^{'}_{i+1,k-1}(u) - N_{i+1,j-1}(u)}{t_{i+k} - t_{i+1}} \tag{22}$$

The new basis function evaluation cell, which calculates the derivatives of these basis functions also, is shown in the Figure 14.

4.5.2 Computation of Tangent Vectors

The derivative at a point on the NURBS surface, for constant u $(= u^{'})$, with respect to v is given by,

$$\begin{aligned} P^{v}(u, v) &= \frac{[\sum_{j=0}^{m} WN_{j,l}(v)][\sum_{j=0}^{m}(\sum_{i=0}^{n} XN^{'}_{j,l}(v))]}{[\sum_{j=0}^{m}(\sum_{i=0}^{n} WN_{j,l}(v))]^2} \\ &- \frac{[\sum_{j=0}^{m}(\sum_{i=0}^{n} XN_{j,l}(v))][\sum_{j=0}^{m}(\sum_{i=0}^{n} WN^{'}_{j,l}(v))]}{[\sum_{j=0}^{m}(\sum_{i=0}^{n} WN_{j,l}(v))]^2} \end{aligned} \tag{23}$$

where $W = (\sum_{i=0}^{n} w_{ij} N_{i,k}(u))$ and $X = (\sum_{i=0}^{n} P_{ij} w_{ij} N_{i,k}(u))$.

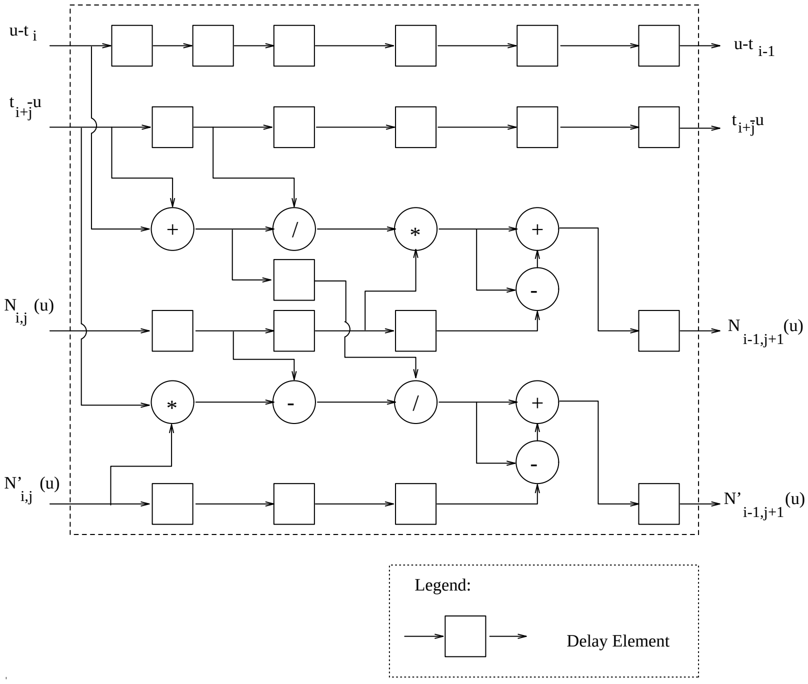

Figure 14: The Modified BFEA Cell

The derivative of the point, for constant v $(= v^{'})$, with respect to u is given by,

$$
\begin{aligned}
P^u(u,v) \quad = & \quad \frac{[\sum_{j=0}^{m} WN_{j,l}(v)][\sum_{j=0}^{m}(\sum_{i=0}^{n} X^u N_{j,l}(v))]}{[\sum_{j=0}^{m}(\sum_{i=0}^{n} WN_{j,l}(v))]^2} \\
- & \quad \frac{[\sum_{j=0}^{m}(\sum_{i=0}^{n} XN_{j,l}(v))][\sum_{j=0}^{m}(\sum_{i=0}^{n} W^u N_{j,l}(v))]}{[\sum_{j=0}^{m}(\sum_{i=0}^{n} WN_{j,l}(v))]^2}
\end{aligned}
\tag{24}
$$

where $W^u = (\sum_{i=0}^{n} w_{ij} N_{i,k}^{'}(u))$ and $X^u = (\sum_{i=0}^{n} P_{ij} w_{ij} N_{i,k}^{'}(u))$.

It can be seen that X and W are virtual control points and its weights respectively, and these quantities are computed by VCCA. As the form of the equations of W^u and X^u are same as that of W and X, these quantities can also be computed by VCCA. Figure 15, shows the modified VCCA to accommodate the computation of X^u and W^u. It contains an additional array of PACs for which the derivative of the basis function is given as the input. Both the PAC arrays are given the same set of control points. As a result, when X and W are calculated by the first PAC array, the values of X^u and W^u are computed by the second array. These four values are pumped to three Accumulating Cells as shown in the Figure 16. These accumulating cells are similar to the AC described earlier, but without the functional unit for division. These cells, with the above four values and the values

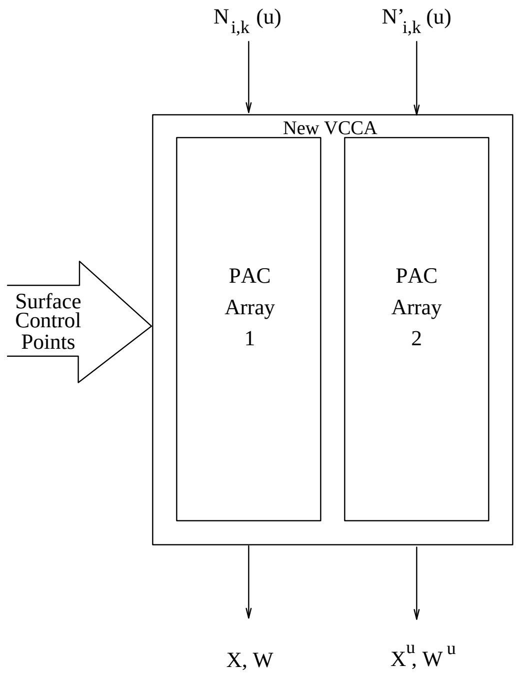

Figure 15: The Modified VCCA

of $N_{j,l}(v)$ and $N^{'}_{j,l}(v)$ from the BFEA, compute all the quantities required to calculate the point on the surface and its tangent vectors. After these tangent vectors are calculated, their cross product is computed to complete the process. The output of the point and its normal are synchronized by introducing delays in the path of the point. While calculating a point on the (3D) NURBS curve, where there is no concept of normal, only the center AC, out of the three is used by the the controller to send the weighted control points.

The performance loss due to the incorporation of the normal calculating hardware is very meager as the delay introduced in the path of the point, which is around ten, is insignificant when the whole surface is computed in thousands of clock cycles. Hence the performance is practically the same as that of the architecture for the computation of the point alone.

4.6 The Controller

The Controller pumps the required data at the appropriate time to various functional units. The controller itself can be divided into various functional modules as shown in the Figure 17. The detailed design of the various modules of the Controller have been presented in [9].

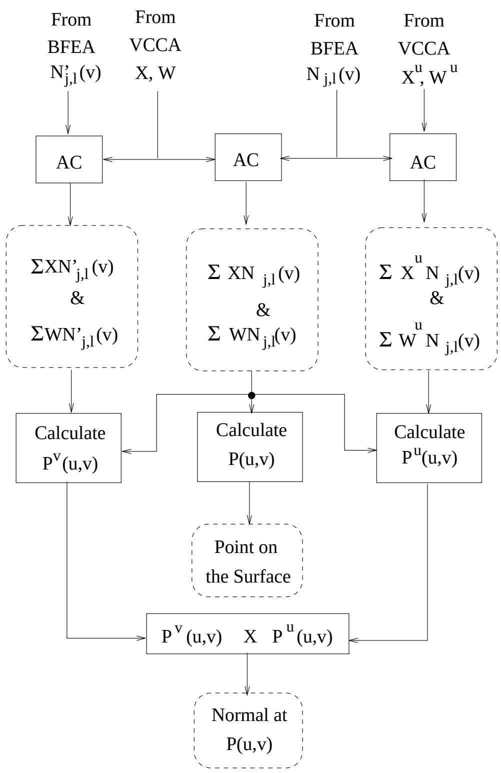

Figure 16: Computation of Surface Points and Normal Vectors

5 Performance Evaluation

The architecture presented above decouples the size of the problem to the extent possible. Every computation/result computed in BFEA is indispensable. From the Equation 21 it is clear that the coefficient of C_u is large, making the timing more dependent on C_u than on C_v. 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 calculated than when a few discrete points are needed. This is because the algorithm makes complete use of inter-dependency and the information sharing between consecutive points on the curve/surface. Thus this architecture outputs one triangle every two clock cycles after the initial pipeline fill.

We now compare the performance of this architecture with that of the architecture proposed by Megson [19]. If the C_u and C_v are proportional to n and m, then the time required for the computation of the curve/surface increases linearly when using the architecture proposed in this

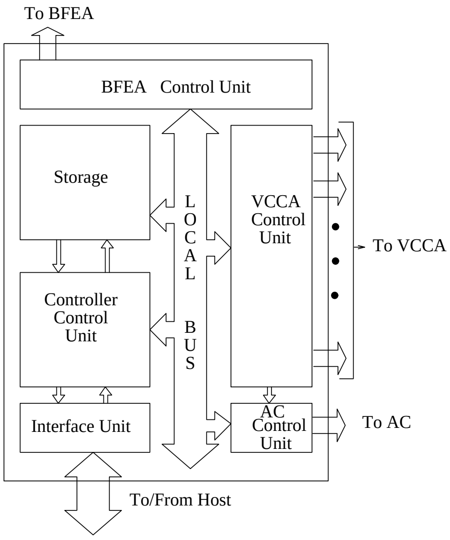

Figure 17: The Controller

paper. As the computation of every point on the curve/surface is dependent on n and m, the time required by the Megson's architecture increases quadratically. This can be seen in Figure 18 and Figure 19.

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 utmost $7max(k,l)+4$ 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 less 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.

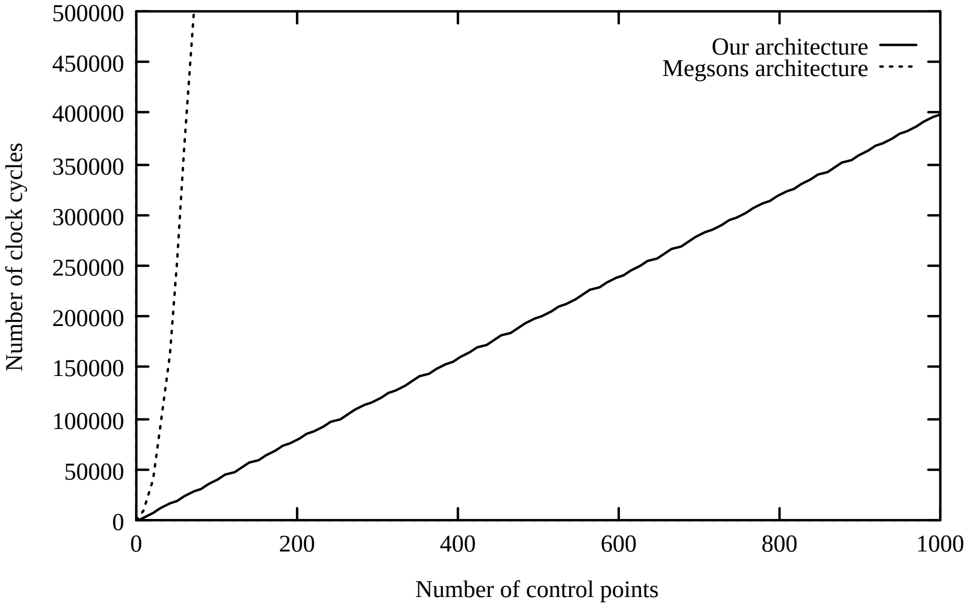

Figure 18: Curve Computation - Comparison

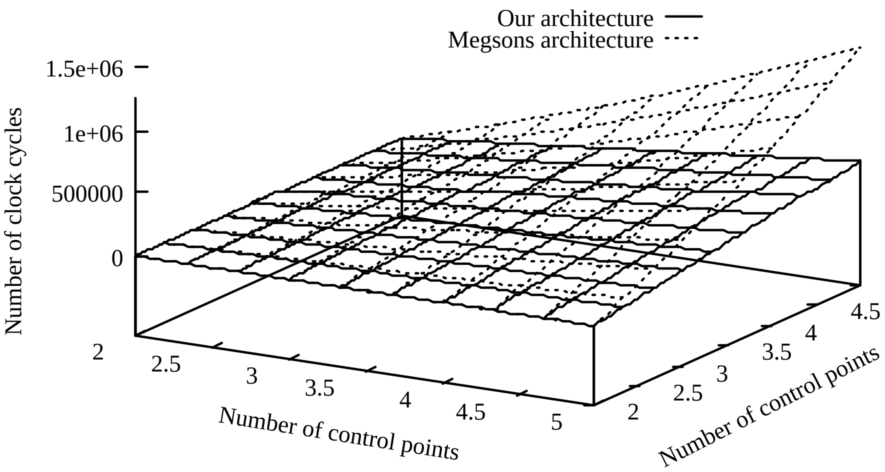

Figure 19: Surface Computation - Comparison

6 Summary

A unified architecture for the computation of B-Spline curves and surfaces has been presented. This architecture is called a unified architecture since it can compute uniform, non-uniform, rational and non-rational B-Spline curves and surfaces. We have considered the computations necessary for NURBS in the above description. To compute a non-rational curve, all the control point weights have to be set to unity. To compute a uniform curve no change is necessary since a uniform knot vector is a special case of a general knot vector. This unified architecture has been derived through a sequence of steps.

First, a systolic architecture for the computation of the basis function values, the BFEA, was developed. Using the BFEA as its core, an architecture for the computation of non-uniform rational B-Spline curves was constructed. This architecture was then extended to compute NURBS surfaces, by introducing the concept of virtual control points and by reducing the computation of a surface to the computation of a sequence of curves. Finally, this architecture was augmented to compute the surface normals so that the output from this architecture can be directly used for rendering the NURBS surface.

The overall linear structure of the architecture, the small number of data paths required by the architecture, the non-dependence of the architecture on the size of the problem (in terms of the number of control points and the number of points on the curve/surface that has to be computed) and the very high throughput of this architecture make it highly suitable for integration into the standard graphics pipeline of high-end workstations.

Results of the timing analysis indicate a potential performance of one triangle every two clock cycles, with its normal vectors at its vertices. It has also been shown that the BFEA is considerably better than the earlier solution for the basis function computation [19]. Further, improvements in the basis function computation are not immediately warranted since as seen in section 4.4, the BFEA has to be slowed down by the introduction of some delay so that its output can be utilized by the inner product computation units. Improvement in the inner product computation part cannot be done without compromising on the linear structure of the architecture. Hence further improvements will require the use of multiple linear arrays or other such configurations.

We conclude by pointing out that this is the first complete hardware solution for the computation of NURBS curves and surfaces.

References

[1] Kurt Akeley. Reality engine graphics. In *SIGGRAPH '93 Proceedings*, pages 109–116, Anaheim, California, Aug. 1993.

[2] B.A Barsky, R.H. Bartels, and J.C. Beatty. An introduction to use of splines in computer graphics. Technical report, Univ. of California, Berkeley, 1983.

[3] J.H. Clark. The geometry engine : A VLSI geometry engine for graphics. *ACM – Computer Graphics*, 16(3):127–133, July 1982.

[4] C. de Boor. A Practical Guide to Splines. *Applied Mathematical Sciences, Springer-Verlag*, 27:132–136, 1978.

[5] C. de Boor. Package for calculating with splines. *Journal of Numerical Analysis*, 14(3):441–472, 1977.

[6] M.F. Deering and Scott T. Nelson. Leo: A system for cost effective 3D shaded graphics. In *SIGGRAPH '93 Proceedings*, pages 101–108, Anaheim, California, Aug. 1993.

[7] T. DeRose and T. Holman. The triangle: A multiprocessor architecture for fast curve and surface generation. Technical report, Univ. of Washington, 1987.

[8] J.D. Foley, A.L. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice(2nd edition)*. Addison-Wesley Publishing Company Inc., 1990.

[9] M. Gopi. *Special Purpose Architectures for B-Splines*. Master's thesis, Supercomputer Education & Research Centre, Indian Institute of Science, Bangalore, India, 1994.

[10] M. Gopi and S. Manohar. VLSI architectures for the computation of uniform B-Spline curves. *Microprocessing and Microprogramming*, 40, pages 617–626, 1994.

[11] M. Gopi and S. Manohar. Parallel architecture for the computation of uniform rational B-Spline patches. *Journal of Parallel and Distributed Computing*, 30:91–98, Nov. 1995.

[12] M. Gopi and S. Manohar. VLSI architecture for the computation of NURBS patches. In *Proceedings of International Conference on VLSI Design*, New Delhi, India, Jan. 1995.

[13] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. In *SIGGRAPH '92 Proceedings*, pages 177–184, July 1992.

[14] IGES. *Initial Graphics Exchange Specification, version 3.0*. National Bureau of Standards, Gaithersburg, MD, USA, 1986.

[15] E.T.Y. Lee. *Rational quadratic Bezier representation for conics.* in Farin, G (ed.) Geometric modeling SIAM, 1986.

[16] T. Li, K.R. Smith, and D. Hanscon. VLSI systolic architectures for computer graphics. In *Proceedings of International Conference on Computers, Systems & Signal Processing*, pages 1527–1530, Bangalore, India, December 1984.

[17] P.C. Mathias. *Systolic Architectures for Realistic 3D Graphics.* PhD thesis, Indian Institute of Science, 1989.

[18] P.C. Mathias, L.M Patnaik, and Sudha Ramesh. Systolic architecture in curve generation. *Computers and Graphics*, 13(4):561–570, 1989.

[19] G.M. Megson. Systolic algorithms for B-Spline patch generation. *Journal of Parallel and Distributed Computing*, 11:231–238, 1991.

[20] M.E Mortenson. *Geometric Modeling.* John Wiley & Sons, 1985.

[21] L. Piegl. On the use of infinite control points in CAGD *Computer Aided Geometric Design*, 4(1–2):155–166, 1987.

[22] L Piegl and W. Tiller. Curve and surface constructions using rational B-Splines. *Computer Aided Design*, 19(9):485–498, 1987.

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

[24] W. Tiller. Rational B-Splines for curve and surface representation. *IEEE Computer Graphics and Applications*, 3(9):61–69, 1983.

[25] W. Tiller. Geometric modeling using non-uniform rational B-Splines: mathematical techniques. In *SIGGRAPH '86*, Tutorial Notes, 1986.

[26] K.J. Versprille. *Computer-aided design applications of the rational B-spline approximation form.* PhD thesis, Syracuse University, 1975.