Systolic Evaluation of Polynomial Expressions

P. C. MATHIAS AND L. M. PATNAIK, SENIOR MEMBER, IEEE

Abstract—**High-speed evaluation of a large number of polynomial expressions has potential applications in the modeling and real-time display of objects in computer graphics. Using VLSI techniques, chips called pixel planes have been built by Fuchs and his group to evaluate linear expressions on frame buffers. Extending the linear evaluation to quadratic evaluation, however, has resulted in the loss of regularity of interconnection among the cells. In this paper, we present two types of organizations for frame buffers of $m \times m$ pixels: one, a single wavefront complex cell array requiring $O(m^2n)$ space and the other a simple cell multiple wavefront array with $O(m^2)$ area and $O(n^2)$ wavefronts. Both these organizations have two main advantages over the earlier proposed method. The cells and the interconnection among them are regular and hence are suitable for efficient VLSI implementation. The organization also permits evaluation of higher order polynomials.**

Index Terms—**Computer graphics, polygon scan conversion, polynomial expression evaluation, systolic arrays, VLSI.**

I. Introduction

VLSI has paved the way for many applications in high-speed interactive computer graphics which is an ever-expanding field with diverse applications in many branches of science and engineering. As a result, many researchers have been attracted towards this area of designing specialized hardware for high-performance graphics. Of particular interest are the systolic and wavefront arrays [12]–[14], which use the desirable properties of pipelining and multiprocessing.

VLSI techniques have already been used for the design and implementation of frame buffers in computer graphics. Even a three-dimensional frame buffer called solids buffer is a practical reality [5]. The frame buffer is one of the most important resources in raster graphics systems [15]. In this paper, we consider the frame buffer to be a two-dimensional memory array storing the picture elements (pixels) that are to be displayed on the screen. With each pixel, one or more memory registers are associated to store the data such as depth, mask, intensity, and color of the pixel. The power of the frame buffer increases as the number of registers associated with each pixel increases. For a formal treatment on this subject, the reader is referred to [2].

Advances in technology have reduced the cost of the frame buffer to a large extent. As a result, the frame buffer has been designed as an array of identical processing elements. Each element has limited processing power and a small number of memory registers. Such an approach has been adopted by Fuchs [3] and his group, where they have designed and fabricated a linear expression evaluator (LEE) which simultaneously computes a linear function of the form,

$$f(x, y) = Ax + By + C, \qquad \text{for all } 1 \le x, y \le m.$$

A multiplier tree has been used for this purpose. It is shown in [4] and [8] that the linear expression evaluation has applications in drawing lines and polygons, generating circles, shading, hidden surface elimination, and for many other functions. By pipelining the coefficients of a large number of expressions, a realistic scene containing thousands of polygons can be rendered in real-time.

In a recent paper, Fuchs *et al.* [9] have modified their multiplier tree to evaluate a quadratic expression. However, the regularity of the interconnection is lost in the process.

We describe two different organizations for frame buffers by adding a few adders to the already existing frame buffer pixel registers, to evaluate a general polynomial of any degree. Cells with many adders in each cell called *c*_cells are used for the single wavefront complex cell method, by which, the polynomial is evaluated using a single wavefront. In the other method, very simple cells called *s*_cells are used and a polynomial is evaluated with multiple wavefronts.

The frame buffer is arranged as a two-dimensional pipeline. The computations proceed as a diagonal wavefront and hence the evaluation of all the pixels is not time aligned. Although this gives rise to increased length of the pipeline as compared in [9], the period for pipelining, i.e., the smallest time between the input data of two successive polynomials presented to the array, remains unity.

The organizations proposed in this paper differ from [9] in many ways. The main advantages of the proposed organizations over [9] are the regularity of cells and their local interconnections. Moreover, higher order polynomials can also be evaluated.

In Section II, we present the motivation of our work with two representative examples. The evaluation of a general polynomial in a single variable using a single wavefront complex cell is described in Section III. We then extend this for the evaluation of polynomials in two variables. In the following section, we describe the polynomial evaluation on simpler cells using multiple wavefronts. Finally, we conclude with a brief discussion on the merits of the proposed array.

II. Motivation

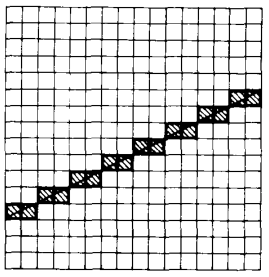
We illustrate our motivation with line and circle drawing examples. Drawing lines is a fundamental primitive in com-

Manuscript received August 14, 1987; revised May 17, 1988.

P. C. Mathias is with the Sophisticated Instruments Facility, Indian Institute of Science, Bangalore, India 560 012.

L. M. Patnaik is with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India 560 012.

IEEE Log Number 9034539.


(a)

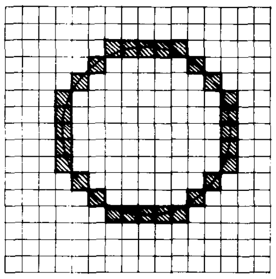

(b)

Fig. 1. (a) Line drawing on a square grid. (b) Circle generation.

puter graphics. Thousands of lines need to be drawn on the screen for displaying a picture. In Fig. 1(a), the screen is represented as a fixed square grid of $m \times m$ pixels. The line is drawn by illuminating the pixels close to the line. The equation for a line in explicit form is given by $Ax + By + C = 0$, in the screen coordinates. Let each one of the pixels in the frame buffer be a processing element and let all pixels evaluate $Ax + By + C$, for their corresponding coordinate points (x, y). Then, if the line equation given above is in the normalized form, the value of $Ax + By + C$ at a point (x, y) is the perpendicular distance of the pixel (x, y) from the line. Hence, we draw the entire line by illuminating all the pixels for which ABS $(Ax + By + C) < 1$ (by setting the pixels' register bit to one). We can also draw thicker lines (sometimes used for anti-aliasing) by keeping the threshold greater than 1 in the above inequality. Half planes can be evaluated by illuminating all pixels for which $Ax + By + C < 0$. By applying logical operators on half planes and lines, many objects can be rendered in real-time using the linear expression evaluation [4].

Generation of arcs and circles involves evaluating quadratic expressions. A circle of radius r centered at point (a, b) can be drawn by illuminating all pixels at (x, y) which are close to the circle. The expression to be evaluated then becomes ABS $(r^2 - ((x-a)^2 + (y-b)^2)) \leq t$, where t is the threshold value. If d is the permitted fixed distance of an illuminated point from the circle boundary, then it can be easily shown that $t = d(2r + d)$, which is linear in r. By keeping d in the above inequality larger, thicker boundaries for circles can be drawn. If the entire interior of the circle is to be illuminated, the inequality used is $(r^2 - ((x-a)^2 + (y-b)^2)) \geq 0$. For applications involving display of molecules, the atoms are modeled as spheres and the bonds between them as cylinders. The orthogonal projection of the atoms and their bonds reduces to evaluating quadratic expressions.

Many objects using constructive solid geometry (CSG) are represented by polynomial expressions. Goldfeather *et al.* have demonstrated [10] the fast display of CSG objects in pixel powers graphics system [9] which evaluates quadratic expressions.

While the discussion in this paper is restricted to polynomial evaluation in x and y, the same ideas can be used for parametric polynomial evaluation with parameters u and v. Such an evaluation finds applications in Bezier and B-spline curve and surface generations, texture mapping, etc. [15].

Having discussed the potential applications of a general polynomial evaluation in computer graphics, we describe in the next section the organization of a single wavefront array to compute these expressions at a very high speed.

III. Single Wavefront Complex Cell Array

This section describes the evaluation of the polynomial by a single wavefront c_cell array. We make use of the forward difference method [11], [15] as applied to polynomials in single variable and later extend it to polynomials in two variables. The main property used is that the nth difference for a polynomial of degree n is a constant.

A. Preliminaries

Consider the polynomial of degree n given by

$$P_n(x) = a_0 + a_1x + a_2x^2 +, \cdots, +a_nx^n = \sum_{i=0}^{n} a_i x^i. \quad (1)$$

The forward difference $P_n(x+1) - P_n(x)$ is a polynomial $P_{n-1}(x)$ in degree $(n-1)$. Applying the same reasoning repeatedly [11] we get

$$\begin{aligned} P_n(x+1) &- P_n(x) &&= P_{n-1}(x) \\ P_{n-1}(x+1) &- P_{n-1}(x) &&= P_{n-2}(x) \\ P_{n-2}(x+1) &- P_{n-2}(x) &&= P_{n-3}(x) \\ \vdots \quad & \quad \vdots && \quad \vdots \\ P_1(x+1) &- P_1(x) &&= P_0(x) = \text{Constant}. \end{aligned} \quad (2)$$

Thus, if we know $P_i(0)$ for all $0 \leq i \leq n$ we can evaluate the polynomial incrementally using (2) at all the grid points $1 \leq x \leq m$. The crux of this paper lies in identifying this potential for use as systolic arrays in frame buffers.

We call $P_{i-1}(x)$ as the predecessor of $P_i(x)$ or $P_i(x)$ as the successor of $P_{i-1}(x)$. From (2), each $P_i(x+1)$ is the sum of the polynomial $P_i(x)$ and its predecessor $P_{i-1}(x)$. The polynomial $P_0(x)$ is a constant, with all its predecessors as zeros which can be omitted. We refer to $P_i(0)$ for $0 \leq i \leq n$, as the initial vector. $P_0(x)$ is called the first element and from this the polynomials of higher degree are obtained. $P_n(x)$ is called the last element and it is this polynomial we need to evaluate.

The initial vector $[P_0(0), P_1(0), \cdots, P_n(0)]^T$ can be obtained by computing the polynomial at grid points 0 to n and

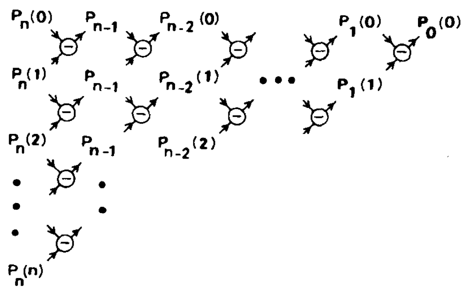


Fig. 2. Initial vector evaluation.

TABLE I
COMPUTATION OF THE INITIAL VECTOR

x	$P_3(x)$	$P_2(x)$	$P_1(x)$	$P_0(x)$
0	a_0	$a_1+a_2+a_3$	$2a_2+6a_3$	$6a_3$
1	$a_0+a_1+a_2+a_3$	$a_1+3a_2+7a_3$	$2a_2+12a_3$	
2	$a_0+2a_1+4a_2+8a_3$	$a_1+5a_2+19a_3$		
3	$a_0+3a_1+9a_2+27a_3$			

using the forward difference technique [11] as shown in Fig. 2. Table I illustrates the evaluation of the initial vector for the example of a cubic polynomial.

Another method to compute the initial vector, which is attractive because of its systolic implementation, is to consider it a linear transform W of the coefficients of the polynomial. This permits the pipelining of coefficients of a large number of polynomials. It can be easily shown that the initial vector is given by

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ \cdot \\ \cdot \\ P_n(0) \end{bmatrix} = \begin{bmatrix} & & & & w_{0,n} \\ & & & w_{1,n-1} & w_{1,n} \\ & & \cdot & & \cdot \\ & \cdot & & & \cdot \\ w_{n,0} & w_{n,1} & \cdot \quad \cdot & w_{n,n-1} & w_{n,n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ a_n \end{bmatrix} . \tag{3}$$

The matrix W is a lower right triangular one, and the elements $w_{i,j}$ for $n \leq i+j \leq 2n$, are integers which depend only on n. They can be precomputed and stored as a matrix depending on the polynomial degree.

The sequential algorithm to compute the polynomial at grid points $1 \leq x \leq m$ is given below.

```
    Algorithm polynomial_in_x;
        begin
step 1.   Compute the initial vector $P_i(0)$, for $0 \leq i \leq n$.
step 2.   for $x := 0$ to $m - 1$ do
step 3.       { * compute the polynomial at $x$ * }
              begin
                  $P_0(x+1) := P_0(x)$;
step 4.           for $i := 1$ to $n$ do
step 5.               $P_i(x+1) := P_{i-1}(x) + P_i(x)$;
              end;
        end;
```

B. The x_array of the Frame Buffer

As mentioned earlier [2], the frame buffer consists of a memory array with a set of memory registers devoted to each pixel. We show in the following sections that by adding a few adders to evaluate polynomials, the frame buffer is made intelligent and its processing capability enhanced considerably.

We consider in this paper only word parallel organizations. However, actual physical implementation constraints of cost and size may require bit serial organizations. Instead of implementing the whole frame buffer, a smaller array (say 16×16 cells) representing a window in the frame buffer would permit word parallel organizations from the point of view of cost and size. By windowing, clipping, and bounding box techniques, high performance at the expense of little reduction in speed can be achieved.

From the sequential algorithm we see a convenient way of unwinding the "for" loops that exactly fits into our notion of the frame buffer organization. The unwinding process also creates a pipeline without feedback such that the initial vector of a large number of polynomials can be completely pipelined [1].

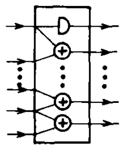
First the description of a generic *c*_cell [shown in Fig. 3(a)] to evaluate a polynomial of degree n is presented. It consists of n adders and a delay element. The delay element can be considered as a two-input adder with one of its inputs permanently connected to logic zero. The cell evaluates in parallel, in unit time the steps 3–5 in the above algorithm. For simplicity, we assume throughout this paper that the output of an adder is available one cycle after its inputs are made available. This can be easily achieved by inserting delay registers at the output of every adder in each cell.

The computation of the polynomial in a systolic array consisting of generic *c*_cells is illustrated using the example of a cubic polynomial given by

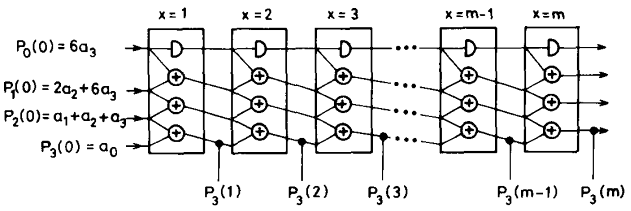
$$P_3(x) = a_0 + a_1x + a_2x^2 + a_3x^3.$$

Using difference technique shown in Table I, we get the initial vector

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ P_2(0) \\ P_3(0) \end{bmatrix} = \begin{bmatrix} 6a_3 \\ 2a_2 + 6a_3 \\ a_1 + a_2 + a_3 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 6 \\ 0 & 0 & 2 & 6 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} . \tag{4}$$



(a)



(b)

Fig. 3. (a) A generic complex cell. (b) The *x*_array using *c*_cells for evaluation of cubic polynomial.

TABLE II
THE PROGRESS OF THE WAVEFRONT FOR THE CUBIC POLYNOMIAL FOR $x = 1$ TO 3

	Input to the array at t=0 (initial vector)	Output of x_array cell		
		x=1, t=1	x=2, t=2	x=3, t=3
$P_0(x)$	$6a_3$	$6a_3$	$6a_3$	$6a_3$
$P_1(x)$	$2a_2+6a_3$	$2a_2+12a_3$	$2a_2+18a_3$	$2a_2+24a_3$
$P_2(x)$	$a_1+a_2+a_3$	$a_1+3a_2+7a_3$	$a_1+5a_2+19a_3$	$a_1+7a_2+37a_3$
$P_3(x)$	**a_0**	**$a_0+a_1+a_2$ $+a_3$**	**$a_0+2a_1+4a_2$ $+8a_3$**	**$a_0+3a_1+9a_2$ $+27a_3$**

Unwinding the "for" loop of step 2, we get a linear array which evaluates the polynomial. Fig. 3(b) shows the array for the case of the cubic polynomial. The host computes the initial vector and feeds it to the array as shown in the figure. The computation proceeds from left to right as a wavefront. Table II shows the outputs of each cell as the wavefront progresses. The last row (in bold characters) corresponds to the evaluation of the polynomial.

For the linear, quadratic, and cubic expressions of the form $P_n(x) = \sum_{i=0}^{n} a_i x^i$, one can easily show that

$$\begin{bmatrix} P_0(0) \\ P_1(0) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \text{—linear case} \tag{5}$$

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ P_2(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \text{—quadratic case} \tag{6}$$

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ P_2(0) \\ P_3(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 6 \\ 0 & 0 & 2 & 6 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \text{—cubic case.} \tag{7}$$

We use these results when we extend the polynomial evaluation to the case of two variables.

To evaluate a polynomial of degree n, for m pixels in the array it takes $m(n+1) = O(mn)$ space. The pipelining period is unity.

C. Polynomial Evaluation in Two Variables Using c_cells

We write the bivariable polynomial of degree n as

$$Q_n(x, y) = \sum_{0 \leq i+j \leq 0} a_{i,j} x^i y^j.$$

Treating $Q_n(x, y)$ as a polynomial $P_n(x)$ of a single variable of degree n, we can write

$$P_n(x) = \sum_{i=0}^{n} f_{n-i}(y)x^i. \tag{8}$$

Each coefficient in the above equation, i.e., $f_{n-i}(y)$, is a polynomial in a single variable y of degree $n-i$. For a particular y value, these $f_{n-i}(y)$ can be evaluated to yield the coefficients of the polynomial $P_n(x)$. But the initial vector that is required at the input of the x_array is the linear transform W of these coefficients according to (3), and not just the coefficients alone. If $P_i(x)$ are the polynomials in x, using (3) we get

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ \cdot \\ \cdot \\ \cdot \\ P_n(0) \end{bmatrix} = \begin{bmatrix} & & 0 & & & w_{0,n} \\ & & & & w_{1,n-1} & w_{1,n} \\ & & & \cdot & & \cdot \\ & & \cdot & & & \cdot \\ & \cdot & & & & \cdot \\ w_{n,0} & w_{n,1} & \cdot\ \cdot\ \cdot & & w_{n,n-1} & w_{n,n} \end{bmatrix} \begin{bmatrix} f_n(y) \\ f_{n-1}(y) \\ \cdot \\ \cdot \\ \cdot \\ f_0(y) \end{bmatrix} = \begin{bmatrix} g_o(y) \\ g_1(y) \\ \cdot \\ \cdot \\ \cdot \\ g_n(y) \end{bmatrix}. \tag{9}$$

Treating $f_{n-i}(y)$ as the coefficients of a polynomial $P_i(x)$, we have

$$P_i(0) = \sum_{j=0}^{n} w_{i,j} f_{n-j}(y) = g_i(y), \qquad \text{for } 0 \le i \le n$$

where $w_{i,j}$'s are the elements of the linear transform W, which is an $(n+1) \times (n+1)$ lower right triangular matrix and is fixed for a given n. From (4) we get

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ P_2(0) \\ P_3(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 6 \\ 0 & 0 & 2 & 6 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_3(y) \\ f_2(y) \\ f_1(y) \\ f_0(y) \end{bmatrix} = \begin{bmatrix} 6f_0(y) \\ 2f_1(y) + 6f_0(y) \\ f_2(y) + f_1(y) + f_0(y) \\ f_3(y) \end{bmatrix} = \begin{bmatrix} 6a_{3,0} \\ (2a_{2,0} + 6a_{3,0}) + 2a_{2,1}y \\ (a_{1,0} + a_{2,0} + a_{3,0}) + (a_{1,1} + a_{2,1})y + a_{1,2}y^2 \\ a_{0,0} + a_{0,1}y + a_{0,2}y^2 + a_{0,3}y^3 \end{bmatrix}.$$

Each $P_i(0)$ is a polynomial in y of degree i and is suitable for feeding it into the x_array. The sequential algorithm to evaluate $Q_n(x, y)$ for all $1 \le x, y \le m$, is given below.

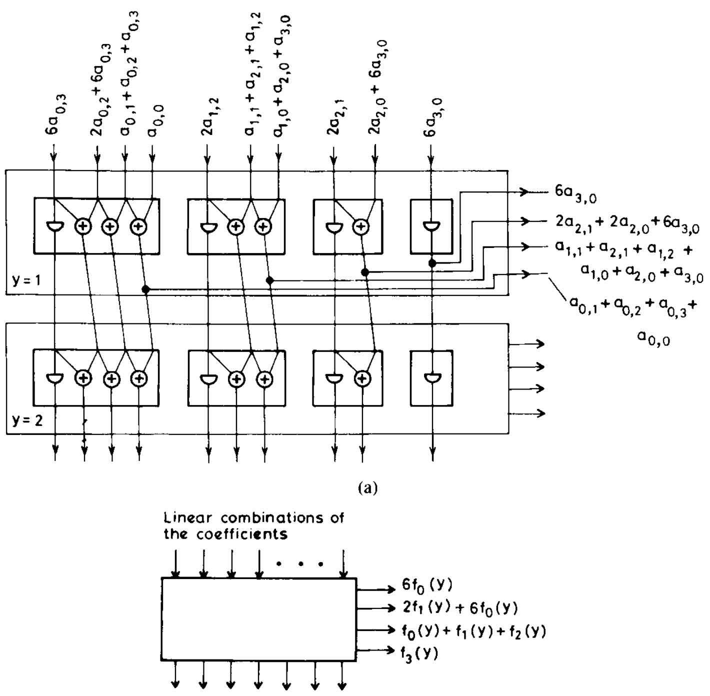
```
    Algorithm polynomial_in_xy;
            begin
step 1.     for $y := 1$ to $m$ do
               begin
step 2.        for $i := 0$ to $n$ do
step 3.           Compute the initial vector $P_i(0) := g_i(y)$;
step 4.        for $x := 0$ to $m - 1$ do
                  begin
step 5.              $P_0(x+1) := P_0(x)$;
step 6.              for $i := 1$ to $n$ do
step 7.                 $P_i(x+1) := P_{i-1}(x) + P_i(x)$;
                  end;
               end;
            end;
```

The y_array of the Frame Buffer: The unwinding of step 1 in the above algorithm yields the y_array. The y_array treats the polynomials $f_{n-i}(y)$ as coefficients, and evaluates the transformed coefficients $g_i(y)$ such that they form the initial vectors for the x_arrays. Steps 2 and 3 of the algorithm suggest a way to design each cell of the y_array. In order to evaluate step 2 in parallel, we require $(n+1)$ generic cells of size i, $0 \le i \le n$. Steps 4–7 represent the evaluation of polynomial in x, using the x_array described earlier. We illustrate the above method for the cubic polynomial in two variables given by

$$\begin{aligned} Q_3(x, y) &= \sum_{0 \le i+j \le 3} a_{i,j} x^i y^j \\ &= (a_{0,0} + a_{0,1}y + a_{0,2}y^2 + a_{0,3}y^3) \\ &\quad + (a_{1,0} + a_{1,1}y + a_{1,2}y^2)x \\ &\quad + (a_{2,0} + a_{2,1}y)x^2 \\ &\quad + a_{3,0}x^3. \\ &= f_3(y) + f_2(y)x + f_1(y)x^2 + f_0(y)x^3. \end{aligned}$$

Each row i in the above matrix is a polynomial in y of degree i, for $0 \le i \le 3$. Thus, we have to evaluate linear, quadratic, and cubic expressions of y using (5), (6), and (7). For this, the evaluation on the y_array is shown in Fig. 4. Table III(a) shows the progress of the wavefront downwards for the y_array cells for $y = 1$ to $y = 4$. The entire frame buffer consists of an array of x_arrays, one for each y and shown in Fig. 5. The y_array computes the initial vector for each x_array and feeds it at the left side of the array as shown. From the figure it is clear that in the frame buffer the wavefront advances diagonally. Table III(b) and (c) lists the outputs of the x_array for $y = 1$ and $y = 2$, respectively. As before, the contents of the last row in the tables are the result of the cubic polynomial evaluation.

The linear combinations of the coefficients fed to the y_array can be computed by the host, or else, a simple matrix vector multiplier chip can be built such that the chip receives



$6a_{0,3}$
$2a_{0,2}+6a_{0,3}$
$a_{0,1}+a_{0,2}+a_{0,3}$
$a_{0,0}$
$2a_{1,2}$
$a_{1,1}+a_{2,1}+a_{1,2}$
$a_{1,0}+a_{2,0}+a_{3,0}$
$2a_{2,1}$
$2a_{2,0}+6a_{3,0}$
$6a_{3,0}$
y = 1
y = 2
$6a_{3,0}$
$2a_{2,1}+2a_{2,0}+6a_{3,0}$
$a_{1,1}+a_{2,1}+a_{1,2}+a_{1,0}+a_{2,0}+a_{3,0}$
$a_{0,1}+a_{0,2}+a_{0,3}+a_{0,0}$
(a)
Linear combinations of the coefficients
$6f_0(y)$
$2f_1(y)+6f_0(y)$
$f_0(y)+f_1(y)+f_2(y)$
$f_3(y)$
(b)

Fig. 4. (a) The c_y array for $n = 3$. (b) Block schematic of c_y_array cell for $n = 3$.

the coefficients as inputs from the host and outputs the required linear transforms. Thus, the host can pipeline to the chip the coefficients of a large number of polynomials which need to be evaluated for the display of curves and surfaces, polygons, hidden surface removal, and many other applications.

Each cell in the y array requires $O(n^2)$ space. The y_array having m cells requires $O(mn^2)$ space. The entire frame buffer requires $O(m^2n+mn^2)$ space. The pipelining period is unity. Usually for computer graphics applications, the degree n of the polynomial is small compared to m and is less than or equal to 3. Hence, the space required is $O(m^2n)$.

In the following section, we present the organization of the multiple wavefront array for the evaluation of polynomials using the simple s_cells.

IV. Multiple Wavefront Polynomial Evaluation

There are many transformation techniques that can be exploited to aid the design of systolic systems [1], [6], [7]. Instead of deriving the results using well-known techniques [1], [7], we directly state the results. Either by space time transformation and geometric projection [1] on the array of Fig. 3(b), or by index set transformations [7], on the sequential algorithm polynomial_in_x, we get the array as shown in Fig. 6(b). The primitive equation $P_i(x+1) = P_i(x) + P_{i-1}(x)$, in step 5 of algorithm polynomial_in_x, is transformed to the equation given below:

$$\text{Cell}(x+1, t+1) = \text{Cell}(x, t) + \text{Cell}(x, t-1)$$

i.e., the output of the cell at $x+1$ evaluated at time $t+1$ is equal to the sum of the outputs of cell at x at times t and $t-1$. The parallel evaluation of the polynomial at a cell (single wavefront) is reduced to pipelining (multiple wavefronts). Each generic cell shown in Fig. 6(a) contains a single two-input adder and a storage register. As before, we assume that the output of the adder is available one cycle after its inputs are made available. The polynomial given by

$$P_n(x) = \sum_{i=0}^{n} a_i x^i \tag{10}$$

is evaluated by pipelining the elements of the initial vector into the x_array with $P_0(0)$ first, followed by its successors. As the cells are independent of the degree of the polynomial, any polynomial can be evaluated with multiple wavefronts. An m pixel x_array requires $O(m)$ space and $(n+1)$ wavefronts. We illustrate the above procedure with an example of a cubic polynomial given by

$$P_3(x) = a_3x^2 + a_2x^2 + a_1x + a_0.$$

The initial values of the wavefronts $P_0(0)$, $P_1(0)$, $P_2(0)$, and $P_3(0)$ from (4) are

$$\begin{bmatrix} P_0(0) \\ P_1(0) \\ P_2(0) \\ P_3(0) \end{bmatrix} = \begin{bmatrix} 6a_3 \\ 2a_2 + 6a_3 \\ a_1 + a_2 + a_3 \\ a_0 \end{bmatrix}.$$

TABLE III
(a) OUTPUTS OF THE *y*_ARRAY. (b) THE OUTPUTS OF THE *x*_ARRAY FOR $y = 1$. (c) THE OUTPUTS OF THE *x*_ARRAY FOR $y = 2$.

Time	Cell No.	$f_3(y)$	$f_0(y)+f_1(y)+f_2(y)$	$2f_1(y)+6f_0(y)$	$6f_0(y)$
1	y=1	$a_{0,0}+a_{0,1}+a_{0,2}+a_{0,3}$	$a_{1,0}+a_{2,0}+a_{3,0}+a_{1,1}+a_{2,1}+a_{1,2}$	$2a_{2,1}+2a_{2,0}+6a_{3,0}$	$6a_{3,0}$
2	y=2	$a_{0,0}+2a_{0,1}+4a_{0,2}+8a_{0,3}$	$a_{1,0}+a_{2,0}+a_{3,0}+2a_{1,1}+2a_{2,1}+4a_{1,2}$	$2a_{2,0}+6a_{3,0}+4a_{2,1}$	$6a_{3,0}$
3	y=3	$a_{0,0}+3a_{0,1}+9a_{0,2}+27a_{0,3}$	$a_{1,0}+a_{2,0}+a_{3,0}+3a_{1,1}+3a_{2,1}+9a_{1,2}$	$2a_{2,0}+6a_{3,0}+6a_{2,1}$	$6a_{3,0}$
4	y=4	$a_{0,0}+4a_{0,1}+16a_{0,2}+64a_{0,3}$	$a_{1,0}+a_{2,0}+a_{3,0}+4a_{1,1}+4a_{2,1}+16a_{1,2}$	$2a_{2,0}+6a_{3,0}+8a_{2,1}$	$6a_{3,0}$

(a)

Input	Output of x_array cell		
	x=1	x=2	x=3
$6a_{3,0}$	$6a_{3,0}$	$6a_{3,0}$	$6a_{3,0}$
$2a_{2,1}+2a_{2,0}+6a_{3,0}$	$2a_{2,1}+2a_{2,0}+12a_{3,0}$	$2a_{2,1}+2a_{2,0}+18a_{3,0}$	$2a_{2,1}+2a_{2,0}+24a_{3,0}$
$a_{1,0}+a_{2,0}+a_{3,0}+a_{1,1}+a_{2,1}+a_{1,2}$	$a_{1,0}+3a_{2,0}+7a_{3,0}+a_{1,1}+3a_{2,1}+a_{1,2}$	$a_{1,0}+5a_{2,0}+19a_{3,0}+a_{1,1}+5a_{2,1}+a_{1,2}$	$a_{1,0}+7a_{2,0}+37a_{3,0}+a_{1,1}+7a_{2,1}+a_{1,2}$
$a_{0,0}+a_{0,1}+a_{0,2}+a_{0,3}$	$a_{0,0}+a_{0,1}+a_{0,2}+a_{0,3}+a_{1,0}+a_{2,0}+a_{3,0}+a_{1,1}+a_{2,1}+a_{1,2}$	$a_{0,0}+a_{0,1}+a_{0,2}+a_{0,3}+2a_{1,0}+4a_{2,0}+8a_{3,0}+2a_{1,1}+4a_{2,1}+2a_{1,2}$	$a_{0,0}+a_{0,1}+a_{0,2}+a_{0,3}+3a_{1,0}+9a_{2,0}+27a_{3,0}+3a_{1,1}+9a_{2,1}+3a_{1,2}$

(The inputs to the x_array are taken from table III(a))

(b)

Input	Output of x_array cell		
	x=1	x=2	x=3
$6a_{3,0}$	$6a_{3,0}$	$6a_{3,0}$	$6a_{3,0}$
$2a_{2,0}+6a_{3,0}+4a_{2,1}$	$2a_{2,0}+4a_{2,1}+12a_{3,0}$	$2a_{2,0}+4a_{2,1}+18a_{3,0}$	$2a_{2,0}+4a_{2,1}+24a_{3,0}$
$a_{1,0}+a_{2,0}+a_{3,0}+2a_{1,1}+2a_{2,1}+4a_{1,2}$	$a_{1,0}+3a_{2,0}+7a_{3,0}+2a_{1,1}+6a_{2,1}+4a_{1,2}$	$a_{1,0}+5a_{2,0}+19a_{3,0}+2a_{1,1}+10a_{2,1}+4a_{1,2}$	$a_{1,0}+7a_{2,0}+37a_{3,0}+2a_{1,1}+14a_{2,1}+4a_{1,2}$
$a_{0,0}+2a_{0,1}+4a_{0,2}+8a_{0,3}$	$a_{0,0}+2a_{0,1}+4a_{0,2}+8a_{0,3}+a_{1,0}+a_{2,0}+a_{3,0}+2a_{1,1}+2a_{2,1}+4a_{1,2}$	$a_{0,0}+2a_{0,1}+4a_{0,2}+8a_{0,3}+2a_{1,0}+4a_{2,0}+8a_{3,0}+4a_{1,1}+8a_{2,1}+8a_{1,2}$	$a_{0,0}+2a_{0,1}+4a_{0,2}+8a_{0,3}+3a_{1,0}+9a_{2,0}+27a_{3,0}+6a_{1,1}+18a_{2,1}+12a_{1,2}$

(The inputs to the x_array are taken from table III(a))

(c)

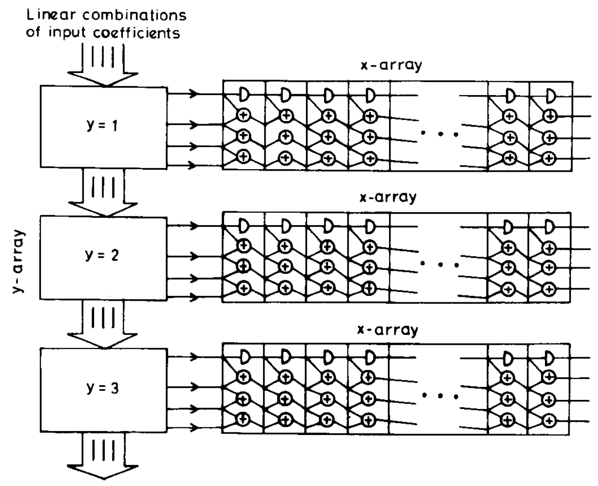


Fig. 5. The frame buffer organization for the evaluation of a cubic polynomial using complex c_cells.

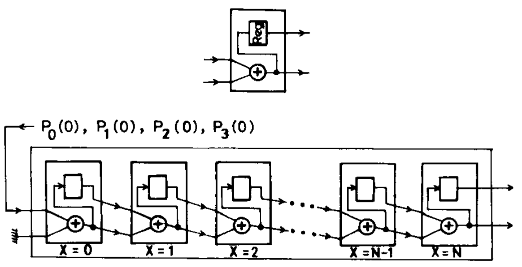


Fig. 6. (a) A generic simplified s_cell. (b) Evaluation of a cubic polynomial using s_array cells.

By pumping the value $6a_3$ followed by $2a_2+6a_3$, $a_1+a_2+a_3$, and a_0 into the pipeline the polynomial $P_3(x)$ is evaluated. It takes four wavefronts to compute $P_3(x)$. Table IV shows the contents of the cell's registers as the four wavefronts progress.

Each cell has to reset the output of the adder and the contents of its register to zero before a new polynomial is evaluated. As $(n+1)$ wavefronts have to cross each cell for the polynomial evaluation, each cell can reset at the end of these $(n+1)$ wavefronts. Alternatively, a wavefront constituting the reset instruction can be pipelined between the coefficients of the two polynomials in the array. We do not account for this wavefront in our complexity calculations.

Polynomial Evaluation in Two Variables Using Simplified Cells: Inspection of a y_array cell in Fig. 4(a) suggests two levels of simplification that can be made on the space required for the cell. As before a bivariable polynomial can be written as

$$Q_n(x,y) = \sum_{0\le i+j\le n} a_{i,j}x^iy^j = \sum_{i=0}^{n} f_{n-i}(y)x^i = P_n(x). \tag{11}$$

Instead of having $(n+1)$ generic cells of size i, $0\le i\le n$, as shown in Fig. 3(a), we now have a single generic cell of size n to evaluate the $(n+1)$ elements of the initial vector one after another using $(n+1)$ wavefronts. This gives rise to the first level of simplification. We call this c_y_array cells which is similar to the x_array cells of Section III. A second level of simplification on the c_y_array cells yields the simple y_array cells called s_y_array cells similar to the simple x_array cells.

a) Using c_y_array Cells: The first level of simplification yields the y_array, shown in Fig. 7 for the case of a cubic polynomial, which has n adders and a delay element. The host inputs to the y_array the coefficients for the evaluation of the element $P_0(0)$ of the initial vector, followed by the coefficients for the evaluation of $P_1(0)$ and so on. It can be seen that it takes $(n+1)$ wavefronts, spaced one time unit apart to compute the elements of the initial vector. The disadvantage of this approach is that the y_array is a function of the degree of the polynomial. For the case of a cubic polynomial we have

$$\begin{aligned}Q_3(x,y) &= \sum_{0\le i+j\le 3} a_{i,j}x^iy^j\\ &= (a_{0,0}+a_{0,1}y+a_{0,2}y^2+a_{0,3}y^3)\\ &\quad+(a_{1,0}+a_{1,1}y+a_{1,2}y^2)x\\ &\quad+(a_{2,0}+a_{2,1}y)x^2\\ &\quad+a_{3,0}x^3\\ &= f_3(y)+f_2(y)x+f_1(y)x^2\\ &\quad+f_0(y)x^3 = P_3(x).\end{aligned}$$

As before, the initial vector is given by

$$\begin{bmatrix}P_0(0)\\P_1(0)\\P_2(0)\\P_3(0)\end{bmatrix} = \begin{bmatrix}6f_0(y)\\2f_1(y)+6f_0(y)\\f_2(y)+f_1(y)+f_0(y)\\f_3(y)\end{bmatrix} = \begin{bmatrix}6a_{3,0}\\(2a_{2,0}+6a_{3,0})+2a_{2,1}y\\(a_{1,0}+a_{2,0}+a_{3,0})+(a_{1,1}+a_{2,1})y+a_{1,2}y^2\\a_{0,0}+a_{0,1}y+a_{0,2}y^2+a_{0,3}y^3\end{bmatrix}.$$

TABLE IV
THE CONTENTS OF THE x_ARRAY CELLS FOR $x = 0$ TO 3

		Output of x_array cell			
Time	Input	cell 0	cell 1	cell 2	cell 3
0	$6a_3$				
1	$6a_3+2a_2$	$6a_3$			
2	$a_1+a_2+a_3$	$6a_3+2a_2$			
3	a_0	$a_1+a_2+a_3$	$12a_3+2a_2$	$6a_3$	
4		$\mathbf{a_0}$	$a_1+3a_2+7a_3$	$18a_3+2a_2$	$6a_3$
5			$\mathbf{a_0+a_1+a_2+a_3}$	$a_1+5a_2+19a_3$	$24a_3+2a_2$
6				$\mathbf{a_0+2a_1+4a_2+8a_3}$	$a_1+7a_2+37a_3$
7					$\mathbf{a_0+3a_1+9a_2+27a_3}$

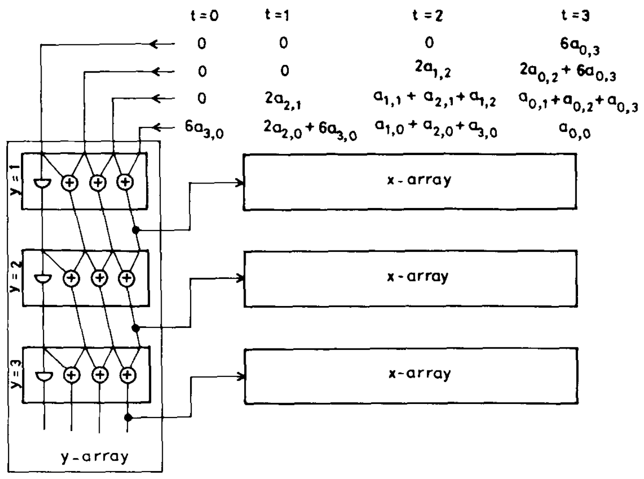


Fig. 7. The organization of the frame buffer using complex c_y_array cells.

Fig. 7 shows the c_y_array cells together with the coefficients that are pipelined. The first wavefront in the c_y_array computes $6f_0(y)$, the second wavefront computes $2f_1(y)+6f_0(y)$ and so on. Table V(a) lists the values flowing through the c_y_array which are input to the x_array. Table V(b) shows the contents of the cell's registers in the x_array for $y = 1$ and Table V(c) for $y = 2$. The last entry in each column of the tables (in bold characters) is the value of the cubic polynomial.

The c_y_array with m cells requires $m(n+1)$ space and $(n+1)$ wavefronts. Thus, the entire frame buffer can be organized in $O(m^2)$ cells. The pipelining period is $(n+1)$.

b) Using s_y_array Cells: The second level of simplification on the c_y_array cells yields the simplified generic cell similar to that of Fig. 6(a). The s_y_array (see Fig. 8) computes, using one wavefront, the element $P_0(0)$, of the initial vector, which is a polynomial in y of degree 0. At the end of one wavefront, the y_array outputs the result to the x_array. The y_array then computes the successor element $P_1(0)$ using two wavefronts and at the end of two wavefronts, it outputs $P_1(0)$ to the x_array and so on. The organization of the frame buffer is shown in Fig. 8. It can be seen that it takes $(n+1)(n+2)/2$ wavefronts (time units), for the evaluation of the polynomial of degree n. The space requirement is $O(m^2)$. For the case of cubic polynomial, the computation of $6f_0(y)$ requires one wavefront and the computation of $2f_1(y)+6f_0(y)$

TABLE V
(a) THE WAVEFRONTS FLOWING THROUGH THE y_ARRAY. (b) THE OUTPUTS OF THE x_ARRAY FOR $y = 1$. (c) THE OUTPUTS OF x_ARRAY FOR $y = 2$.

Time	Initial vector input to the y_array	Output of the y_array cells for		
		y=1	y=2	y=3
0	0			
	0			
	0			
	$6a_{3,0}$			
1	0	$6a_{3,0}$		
	0			
	$2a_{2,1}$			
	$2a_{2,0}+6a_{3,0}$			
2	0	$2a_{2,0}+6a_{3,0}$	$6a_{3,0}$	
	$2a_{1,2}$	$+2a_{2,1}$		
	$a_{1,1}+a_{2,1}+a_{1,2}$			
	$a_{1,0}+a_{2,0}+a_{3,0}$			
3	$6a_{0,3}$	$a_{1,0}+a_{2,0}+a_{3,0}$	$2a_{2,0}+6a_{3,0}$	$6a_{3,0}$
	$2a_{0,2}+6a_{0,3}$	$+a_{1,1}+a_{2,1}+a_{1,2}$	$+4a_{2,1}$	
	$a_{0,1}+a_{0,2}+a_{0,3}$			
	$a_{0,0}$			
4		$a_{0,1}+a_{0,2}+a_{0,3}$	$a_{1,0}+a_{2,0}+a_{3,0}$	$2a_{2,0}+6a_{3,0}$
		$+a_{0,0}$	$+2a_{1,1}+2a_{2,1}+4a_{1,2}$	$+6a_{2,1}$
5			$a_{0,0}+2a_{0,1}$	$a_{1,0}+a_{2,0}$
			$+4a_{0,2}+8a_{0,3}$	$+a_{3,0}+3a_{1,1}$
				$+3a_{2,1}+9a_{1,2}$
6				$a_{0,0}+3a_{0,1}$
				$+9a_{0,2}+27a_{0,3}$

(Refer fig. 7)

(a)

Time	Input	Output of x_array cells for		
		x = 0	x = 1	x = 2
1	$6a_{3,0}$			
2	$2a_{2,1}+2a_{2,0}$	$6a_{3,0}$		
	$+6a_{3,0}$			
3	$a_{1,0}+a_{2,0}$	$2a_{2,1}+2a_{2,0}$	$6a_{3,0}$	
	$+a_{3,0}+a_{1,1}$	$+6a_{3,0}$		
	$+a_{2,1}+a_{1,2}$			
4	$a_{0,0}+a_{0,1}$	$a_{1,0}+a_{2,0}$	$2a_{2,1}+2a_{2,0}$	$6a_{3,0}$
	$+a_{0,2}+a_{0,3}$	$+a_{3,0}+a_{1,1}$	$+12a_{3,0}$	
		$+a_{2,1}+a_{1,2}$		

(b)

TABLE V. (*Continued*)

		Output of x_array cells for		
Time	Input	x = 0	x = 1	x = 2
5		**$a_{0,0}+a_{0,1}$ $+a_{0,2}+a_{0,3}$**	$a_{1,0}+3a_{2,0}+7a_{3,0}$ $+a_{1,1}+3a_{2,1}+a_{1,2}$	$2a_{2,1}+2a_{2,0}$ $+18a_{3,0}$
6			**$a_{0,0}+a_{0,1}+a_{0,2}$ $+a_{0,3}+a_{1,0}+a_{2,0}$ $+a_{3,0}+a_{1,1}+a_{2,1}$ $+a_{1,2}$**	$a_{1,0}+5a_{2,0}$ $+19a_{3,0}+a_{1,1}$ $+5a_{2,1}+a_{1,2}$
7				**$a_{0,0}+a_{0,1}$ $+a_{0,2}+a_{0,3}$ $+2a_{1,0}+4a_{2,0}$ $+8a_{3,0}+2a_{1,1}$ $+4a_{2,1}+2a_{1,2}$**

(The input is taken from table V(a))

(b)

		Outputs of the x_array for		
Time	Input	x=0	x=1	x=2
1	$6a_{3,0}$			
2	$2a_{2,0}+6a_{3,0}$ $+4a_{2,1}$	$6a_{3,0}$		
3	$a_{1,0}+a_{2,0}$ $+a_{3,0}+2a_{1,1}$ $+2a_{2,1}+4a_{1,2}$	$2a_{2,0}+6a_{3,0}$ $+4a_{2,1}$	$6a_{3,0}$	
4	$a_{0,0}+2a_{0,1}$ $+4a_{0,2}+8a_{0,3}$	$a_{1,0}+a_{2,0}$ $+a_{3,0}+2a_{1,1}$ $+2a_{2,1}+4a_{1,2}$	$2a_{2,0}+4a_{2,1}+12a_{3,0}$	$6a_{3,0}$
5		$a_{0,0}+2a_{0,1}$ $+4a_{0,2}+8a_{0,3}$	$a_{1,0}+3a_{2,0}+7a_{3,0}$ $+2a_{1,1}+6a_{2,1}+4a_{1,2}$	$2a_{2,0}+4a_{2,1}$ $+18a_{3,0}$
6			**$a_{0,0}+a_{0,1}+4a_{0,2}$ $+8a_{0,3}+a_{1,0}+a_{2,0}$ $+a_{3,0}+2a_{1,1}+2a_{2,1}$ $+4a_{1,2}$**	$a_{1,0}+5a_{2,0}$ $+19a_{3,0}+2a_{1,1}$ $+10a_{2,1}+4a_{1,2}$
7				**$a_{0,0}+2a_{0,1}$ $+4a_{0,2}+8a_{0,3}$ $+2a_{1,0}+4a_{2,0}$ $+8a_{3,0}+4a_{1,1}$ $+8a_{2,1}+8a_{1,2}$**

(c)

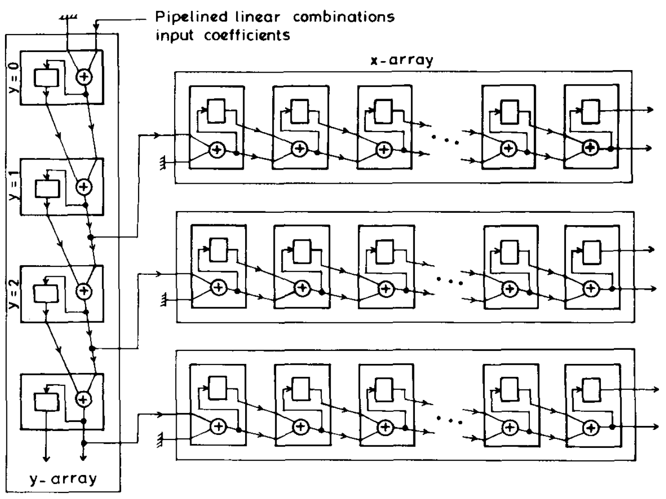


Fig. 8. Organization of the frame buffer with simplified cells.

TABLE VI
COMPLEXITIES OF THE FRAME BUFFER ORGANIZATIONS

Array	Area A	period p	A * p	A * p^2
c_cell array (fig. 3)	$O(m^2n)$	$O(1)$	$O(m^2n)$	$O(m^2n)$
s_cell x_array c_cell y_array (fig. 7)	$O(m^2)$	$O(n)$	$O(m^2n)$	$O(m^2n^2)$
s_cell x_array, s_cell y_array (fig. 8)	$O(m^2)$	$O(n^2)$	$O(m^2n^2)$	$O(m^2n^4)$

requires two wavefronts and so on. The input wavefronts to the *x*_array are not equispaced but are linearly spaced in time. It requires ten wavefronts to be input to the *s_y*_array to compute a cubic polynomial.

V. CONCLUSIONS

In this paper, we have described organizations of frame buffers to evaluate polynomials. From a sequential algorithm we derive a single wavefront complex cell array. The space requirement is reduced to yield multiple wavefront-simplified cell array. Table VI compares the complexities of the organizations for their area × period and area × period^2. A further reduction in area which is possible but not considered in this paper is due to the bit serial pipeline organization. This will increase the number of wavefronts by a factor equal to the word length but uses very simple hardware. The advantage of the organizations described in this paper is that the computation in any cell is carried out using the results of the neighboring cells only and without the need for any broadcast. Hence, it is very attractive to realize the frame buffer in the form of a VLSI chip. The VLSI array can be laid out on the chip as a rectangle such that there is a one to one correspondence between the coordinates of the pixel on the screen and the coordinates of the pixel registers on the chip. Hence, a light emitting device constituting the screen can become a part of the chip. The array being linear in nature provides all the advantages of fault tolerance of linear arrays and efficient layout for VLSI.

If we assume the time taken for the wavefront to move from one cell to the next as 1 ms (as only simple operations are involved), then the frame buffer can compute on the order of one million expressions per second in the steady state, once the pipeline is full.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their comments.

REFERENCES

[1] P. R. Cappello and K. Steiglitz, "Unifying VLSI array designs with geometric transformations," in *Proc. 1983 Int. Conf. Parallel Processing*, Aug. 1983, pp. 448-457.
[2] A. Fournier and D. Fussell, "On the power of the frame buffer," Dynamic Graphics Project, Tech. Rep. DGP84-2, Univ. of Toronto, Mar. 1983.

[3] H. Fuchs, J. Poulton, A. Paeth, and A. Bell, "Developing pixel planes, A smart memory based raster graphics system," in *Proc. 1982 MIT Conf. Advanced Res. VLSI*, Dedham, MA, Artech House, pp. 137-146.
[4] H. Fuchs, J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, F. P. Brooks, J. G. Eyles, and J. Poulton, "Fast spheres, textures, transparencies, and image enhancements in pixel-planes," *Comput. Graphics*, vol. 19, no. 3, pp. 111-120, 1985.
[5] G. Hunter, "3D frame buffers for interactive analysis of 3D data," in *Proc. SPIE's 28th Tech. Symp.*, Aug. 1984.
[6] M. S. Lam and J. Mostow, "A transformational model of VLSI systolic design," *IEEE Comput. Mag.*, 1985.
[7] D. I. Moldovan, "On the design of algorithms and synthesis of VLSI systems," *Proc. IEEE*, pp. 113-120, Jan. 1983.
[8] J. Poulton, H. Fuchs, J. D. Austin, J. G. Eyles, J. Heinecke, C. H. Hsieh, J. Goldfeather, J. P. Hultquist, and S. Spach, "PIXEL-PLANES: Building a VLSI based raster graphics system," in *Proc. 1985 Chapel Hill Conf. VLSI*, pp. 35-60.
[9] J. Goldfeather and H. Fuchs, "Quadratic surface rendering on a logic enhanced frame buffer memory," *IEEE Comput. Graphics Appl.*, pp. 48-59, Jan. 1983.
[10] J. Goldfeather, J. P. Hultquist, and H. Fuchs, "Fast constructive solid geometry display in the pixel-powers graphics system," *Comput. Graphics*, vol. 20, no. 4, pp. 107-116, 1986.
[11] D. E. Knuth, *The Art of Computer Programming, Vol. 2*. Reading, MA: Addison Wesley.
[12] H. T. Kung, "Why systolic architectures," *IEEE Comput. Mag.*, pp. 37-46, Jan. 1982.
[13] S. Y. Kung, K. S. Arun, R. J. Gal-Ezer, and D. V. Bhaskar Rao, "Wavefront array processor: Language, architecture, and applications," *IEEE Trans. Comput.*, vol. C-31, Nov. 1982.
[14] P. C. Mathias and L. M. Patnaik, "A systolic evaluator for linear, quadratic and cubic expressions," *J. Parallel Distributed Comput.*, vol. 5, pp. 729-740, Dec. 1988.
[15] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, 2nd ed. New York: McGraw-Hill, 1979.

P. C. Mathias received the M. Tech degree in electrical engineering from IIT, Kanpur, in 1974.

At present he is a Principal Scientific Officer in the Sophisticated Instruments Facility, Indian Institute of Science, Bangalore, and currently he is working towards his Doctoral degree in systolic architectures for computer graphics. His main interests are in the areas of parallel architectures, computer graphics, CAD, microprocessor hardware, and nuclear magnetic resonance spectroscopy.

Mr. Mathias is a member of the Computer Society of India.

L. M. Patnaik (S'75-M'76-SM'86) received Ph.D. degree in the area of Real-Time Software for Industrial Automation in 1978 and the D.Sc. degree in June 1989 for his research work in the area of computer systems and architectures.

At present, he is a Professor of the Department of Computer Science and Automation, and Chairman of the Microprocessor Applications Laboratory at the Indian Institute of Science, Bangalore. During the last 18 years of his service at the Institute, his teaching, research, and development interests have been in the areas of parallel and distributed computing, computer architecture, computer graphics, CAD of VLSI systems, and expert systems. To his credit, he has over 150 publications in these areas in refereed international journals and conference proceedings. He has supervised several research theses and has been Principal Investigator for a number of government and industry funded projects. He is a co-author of a book on functional programming to be published by the Springer-Verlag.

Dr. Patnaik is a senior member of the Computer Society of India, a Founder Member of the Executive Committee of the recently-launched Association for Advancement of Fault-Tolerant and Autonomous Systems, a Fellow of the Indian Academy of Sciences, National Academy of Sciences, and the Institution of Electronics and Telecommunications Engineers, in India.