MATHEMATICS OF COMPUTATION
Volume 66, Number 220, October 1997, Pages 1521–1553
S 0025-5718(97)00862-4

A UNIFIED APPROACH TO EVALUATION ALGORITHMS FOR MULTIVARIATE POLYNOMIALS

SURESH K. LODHA AND RON GOLDMAN

Abstract. We present a *unified* framework for most of the known and a few new evaluation algorithms for multivariate polynomials expressed in a wide variety of bases including the Bernstein-Bézier, multinomial (or Taylor), Lagrange and Newton bases. This unification is achieved by considering evaluation algorithms for multivariate polynomials expressed in terms of L-bases, a class of bases that include the Bernstein-Bézier, multinomial, and a rich subclass of Lagrange and Newton bases. All of the known evaluation algorithms can be generated either by considering up recursive evaluation algorithms for L-bases or by examining change of basis algorithms for L-bases. For polynomials of degree n in s variables, the class of up recursive evaluation algorithms includes a parallel up recurrence algorithm with computational complexity $O(n^{s+1})$, a nested multiplication algorithm with computational complexity $O(n^s \log n)$ and a ladder recurrence algorithm with computational complexity $O(n^s)$. These algorithms also generate a new generalization of the Aitken-Neville algorithm for evaluation of multivariate polynomials expressed in terms of Lagrange L-bases. The second class of algorithms, based on certain change of basis algorithms between L-bases, include a nested multiplication algorithm with computational complexity $O(n^s)$, a divided difference algorithm, a forward difference algorithm, and a Lagrange evaluation algorithm with computational complexities $O(n^s)$, $O(n^s)$ and $O(n)$ per point respectively for the evaluation of multivariate polynomials at several points.

1. Introduction

The Bernstein-Bézier, multinomial (or Taylor), Lagrange and Newton bases are some of the most popular and useful representations for expressing polynomials of degree n in s variables. Several algorithms for evaluating multivariate polynomials, represented in these bases, have been proposed. The de Casteljau algorithm with computational complexity $O(n^{s+1})$ is well-known for evaluating multivariate Bernstein-Bézier polynomials [13, 14]. Several algorithms for evaluating multivariate polynomials in Bernstein-Bézier or multinomial (Taylor) form with computational complexity $O(n^s)$ have been described [5, 12, 42]. Generalizations

Received by the editor July 26, 1995 and, in revised form, July 22, 1996.
1991 *Mathematics Subject Classification.* Primary 65D17, 65Y20, 68Q25; Secondary 65Y25, 68U05, 68U07.
Key words and phrases. Algorithms, Bernstein, Bézier, change of basis, evaluation, Lagrange, multivariate, Newton, polynomials, recurrence, Taylor.
The first author was supported in part by NSF Grants CCR-9309738, IRI-9423881 and by faculty research funds granted by the University of California, Santa Cruz.
The second author was supported in part by NSF Grant CCR-9113239.

with computational complexity $O(n^{s+1})$ of the Aitken-Neville algorithm for evaluating univariate Lagrange polynomials to certain subclasses of multivariate Lagrange polynomials have been proposed [6, 34, 44]. Evaluation algorithms with computational complexity $O(n^s)$ for multivariate polynomials expressed in Newton bases have also been described [19]. In addition, evaluation algorithms related to the generalizations of forward and divided difference algorithms to multivariate polynomials have also been discussed [45, 11, 46, 25, 41].

Most of these evaluation algorithms seem to have been discovered independently from one another and, therefore, the literature as cited above is scattered and the various algorithms appear to be unrelated. One of the major goals of this work is to demonstrate that these seemingly disparate algorithms are, in fact, closely related. Indeed, all of these algorithms can be derived from the evaluation algorithms for multivariate L-bases, a class of bases that include the Bernstein-Bézier, multinomial, and certain proper subclasses of Lagrange and Newton bases. This unification of these algorithms provides a deeper, cleaner and much richer understanding of a large class of algorithms for evaluating multivariate polynomials. This understanding, in turn, has helped us to design a few new, more efficient algorithms, for evaluating multivariate polynomials.

Our work easily generalizes to arbitrary dimensions. However, for the sake of simplicity, the results are presented and derived here only for bivariate polynomials.

We begin by describing a general parallel up recurrence algorithm with computational complexity $O(n^3)$ for the evaluation of bivariate L-bases. This algorithm specializes to the standard de Casteljau algorithm for evaluation of bivariate Bernstein-Bézier surfaces [13, 14]. The up recurrence also specializes to $O(n^2)$ algorithms for the evaluation of multinomial (Taylor) bases, that include the algorithms proposed earlier by Carnicer-Gasca [5] and de Boor-Ron [12]. In addition, the up recurrence specializes to an algorithm for evaluating Newton polynomials, that has been previously discussed by Carnicer and Gasca [5, 19]. The up recurrence also yields a new generalization of the Aitken-Neville algorithm with computational complexity $O(n^3)$ for the evaluation of bivariate Lagrange L-bases. By removing redundant computations, we show that the general parallel up recurrence algorithm can be improved to a new recurrence algorithm with computational complexity $O(n^2 \log n)$ for the evaluation of arbitrary L-bases. Furthermore, the up recurrence algorithm can be altered into a ladder recurrence algorithm with computational complexity $O(n^2)$ by changing the structure of the parallel up recurrence diagram for the evaluation of bivariate L-bases.

Next we describe another class of algorithms for evaluating L-bases, based on certain change of basis algorithms between L-bases. This class of algorithms include a divided difference algorithm with computational complexity $O(n^2)$ per point, a forward difference algorithm with $O(n^2)$ additions and $O(1)$ multiplications per point, and a new Lagrange evaluation algorithm with amortized computational complexity $O(n)$ per point. The change of basis algorithm can be specialized into a nested multiplication algorithm with computational complexity $O(n^2)$ for the evaluation of bivariate L-bases. This class of algorithms includes an algorithm for evaluating multivariate polynomials in multinomial (Taylor) form described earlier by Schumaker and Volk [42].

This paper is organized in the following manner. Section 2 reviews the definition of L-bases and presents examples of Bernstein-Bézier, multinomial, Lagrange and

Newton bases as L-bases. Section 3 describes up recurrence algorithms – a parallel up recurrence algorithm with computational complexity $O(n^3)$, a nested multiplication evaluation algorithm with computational complexity $O(n^2 \log n)$, and a ladder recurrence algorithm with computational complexity $O(n^2)$. Section 4 presents down recursive evaluation algorithms – a Lagrange evaluation algorithm with computational complexity $O(n)$ per point, a divided difference algorithm with computational complexity $O(n^2)$ per point, a forward difference algorithm with computational complexity $O(n^2)$ per point, and a nested multiplication evaluation algorithm with computational complexity $O(n^2)$. The difference between the nested multiplication evaluation algorithm presented in Section 3.2 and the nested multiplication evaluation algorithm presented in Section 4.4 is highlighted by presenting an example of the evaluation of a bivariate Lagrange L-basis. Section 5 presents a brief description of other evaluation algorithms including hybrid algorithms and derivative evaluation algorithms, that can be obtained by combining or extending some of the above algorithms. Finally, Section 6 concludes with some discussion of future work.

2. L-bases

Here we review the basic definitions and certain well-known examples of L-bases. We also provide very brief geometric interpretations for the algebraic entities associated with the L-bases in order to explain how certain bivariate Lagrange and Newton bases arise as special cases of L-bases. Complete details are provided in [27].

Throughout this paper, we shall adopt the following notation. A multi-index α is a 3-tuple of non-negative integers. If $\alpha = (\alpha_1, \alpha_2, \alpha_3)$, then $|\alpha| = \alpha_1 + \alpha_2 + \alpha_3$ and $\alpha! = \alpha_1!\alpha_2!\alpha_3!$. Other multi-indices will be denoted by β and γ. A unit multi-index e_k is a 3-tuple with 1 in the k-th position and 0 everywhere else. Scalar indices will be denoted by i, j, k, l.

A collection $\mathcal{L}$ of 3 sequences $\{L_{1,j}\}$, $\{L_{2,j}\}$, $\{L_{3,j}\}$, $j = 1, \cdots, n$, of linear polynomials in two variables is called a *knot-net* of polynomials if $(L_{1,\alpha_1+1}, L_{2,\alpha_2+1}, L_{3,\alpha_3+1})$ are linearly independent polynomials for $0 \le |\alpha| \le n-1$. An L-basis $\{l_\alpha^n, |\alpha| = n\}$ is a collection of $\binom{n+2}{2}$ bivariate polynomials defined as follows:

$$l_\alpha^n = \prod_{i=1}^{\alpha_1} L_{1i} \prod_{j=1}^{\alpha_2} L_{2j} \prod_{k=1}^{\alpha_3} L_{3k}. \tag{2.1}$$

It is well-known that $\{l_\alpha^n, |\alpha| = n\}$ is, in fact, a basis for the space of polynomials of degree n on R^2 [7].

We assign to each linear polynomial $ax + by + c$ the corresponding line in the affine plane defined by the equation $ax + by + c = 0$. (The polynomial c corresponds to the line at infinity in the projective plane.) For full details, please refer to [27]. Observe that this correspondence between lines and linear polynomials depends on the coordinate system and is unique only up to constant multiples. Nevertheless, in the following discussions, we shall identify the linear polynomial with the line and vice-versa, whenever the coordinate system and constant multiples are irrelevant for the context at hand. The advantage of this correspondence is to allow us to think of algebraic entities such as linear polynomials in terms of geometric entities such as lines.

2.1. **Bernstein-Bézier Bases.** We can easily realize Bernstein-Bézier bases as special cases of L-bases by choosing the knot-net of polynomials $L_{i,j} = L_i, 1 \leq j \leq n$, $L_1 = a_1x + b_1y + c_1$, $L_2 = a_2x + b_2y + c_2$, $L_3 = a_3x + b_3y + c_3$, where L_1, L_2 and L_3 are linearly independent polynomials such that no two of the associated lines are parallel. It can easily be verified that the corresponding L-basis is, up to constant multiples, the Bernstein-Bézier basis defined by the three intersection points of L_1, L_2 and L_3. In particular, $L_1 = x$, $L_2 = y$ and $L_3 = 1 - x - y$, yields the standard Bernstein-Bézier basis, up to constant multiples, that is, $l^n_\alpha = x^{\alpha_1}y^{\alpha_2}(1-x-y)^{\alpha_3}$. For a detailed discussion of Bernstein-Bézier bases and their properties, we refer the reader to [15].

2.2. **Multinomial Bases.** The multinomial basis is the standard generalization of the monomial basis to the multivariate setting. For example, the basis 1, x, y, x^2, xy and y^2 is the bivariate multinomial basis of degree 2. Sometimes the terminology Taylor basis or power basis is also used instead of monomial or multinomial basis. However, we shall refer to this basis as the multinomial basis in accordance with [23] and reserve the term power basis for the basis each element of which is an n-th power of some linear polynomial [23].

The standard multinomial basis is realized as a special case of an L-basis by choosing the knot-net of polynomials $L_{i,j} = L_i, 1 \leq j \leq n$, $L_1 = x$, $L_2 = y$ and $L_3 = 1$. This yields: $l^n_\alpha = x^{\alpha_1}y^{\alpha_2}$. More general multinomial bases can also be realized as L-bases [27].

2.3. **Lagrange Bases.** Let $\{ \{L_{ij}\}, \{L_{2j}\}, \{L_{3j}\}, j = 1, \cdots, n \}$ be a knot-net of polynomials. Suppose that the polynomials $(L_{1,\alpha_1+1}, L_{2,\alpha_2+1}, L_{3,\alpha_3+1})$ are linearly dependent for $|\alpha| = n$, $0 \leq \alpha_k \leq n-1$. It has been established in [27] that, up to constant multiples, the corresponding L-basis is a Lagrange basis; that is, there exist points $\mathbf{v}_\alpha$ such that $l^n_\alpha(\mathbf{v}_\beta) = l^n_\alpha(\mathbf{v}_\alpha)\delta_{\alpha\beta}$. Therefore, $\{\frac{l^n_\alpha}{l^n_\alpha(\mathbf{v}_\alpha)}\}$ forms a Lagrange basis.

To describe the points $\mathbf{v}_\alpha$, let us analyze the dependency conditions. Overloading the notation, let L_{ij} also denote the line in the plane defined by the equation: $L_{ij} = 0$. The linear dependency condition on the polynomials L_{i,α_i+1} means that the three lines $(L_{1,\alpha_1+1}, L_{2,\alpha_2+1}, L_{3,\alpha_3+1})$ are concurrent for $|\alpha| = n$, $0 \leq \alpha_k \leq n-1$, that is, these three lines pass through a common point. In general, these lines then meet at a point, in the affine plane if the lines intersect or at infinity in the projective plane if the lines are parallel. Let $\mathbf{v}_\alpha = \bigcap_{k=1}^3 L_{k,\alpha_k+1}$ for $|\alpha| = n$, $0 \leq \alpha_k \leq n-1$. These intersections give rise to $\binom{n+2}{2} - 3$ points corresponding to $\binom{n+2}{2} - 3$ dependency conditions. To these points, we shall add three more points: $\mathbf{v}_{n00} = L_{31} \cap L_{21}$, $\mathbf{v}_{0n0} = L_{11} \cap L_{31}$, and $\mathbf{v}_{00n} = L_{11} \cap L_{21}$. These are precisely the $\binom{n+2}{2}$ points that give rise to Lagrange interpolation conditions.

In our earlier work, we described several interesting lattice or point-line configurations that generate Lagrange L-bases [27]. Here we simply describe one interesting lattice, known as a *principal lattice* or *geometric mesh* [8], that admits unique interpolation by a Lagrange L-basis. Figure 1 is an example of a *principal lattice* or *geometric mesh* [8] of order n, which can be described by three sets of n lines $\{\{L_{1i}\}, \{L_{2j}\}, \{L_{3k}\}, 1 \leq i, j, k \leq n\}$ such that each set of three lines $\{L_{1,i+1}, L_{2,j+1}, L_{3,k+1}, i + j + k = n\}$ intersect at exactly one common point $\mathbf{v}_{ijk}$. The lines in Figure 1 satisfy both the linear independence condition for $(L_{1,\alpha_1+1}, L_{2,\alpha_2+1}, L_{3,\alpha_3+1})$, $0 \leq |\alpha| \leq n-1$, which is required to define a knot-net

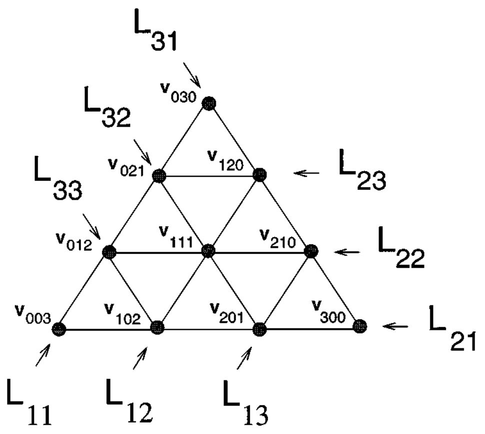

FIGURE 1. Geometric mesh of order 3 for Lagrange L-basis

of polynomials and the linear dependence condition that is required to define a Lagrange basis. It is clear from the above construction that every geometric mesh of order n gives rise to a Lagrange L-basis.

2.4. **Newton Bases.** By choosing the polynomials $L_{1i} = x - a_i$, $L_{2i} = y - b_i$ and $L_{3i} = 1$, we obtain the following Newton basis:

$$l_\alpha^n = \prod_{i=1}^{\alpha_1}(x - a_i)\prod_{j=1}^{\alpha_2}(y - b_j).$$

For example, when $n = 2$ this construction yields the basis functions: 1, $(x - a_1)$, $(x - a_1)(x - a_2)$, $(y - b_1)$, $(y - b_1)(y - b_2)$ and $(x - a_1)(y - b_1)$. A slightly more general Newton L-basis is obtained by simply choosing the polynomials $L_{1i} = a_{1i}x + b_{1i}y + c_{1i}$, $L_{2i} = a_{2i}x + b_{2i}y + c_{2i}$, and $L_{3i} = a_3x + b_3y + c_3$. In our earlier work [27], we established that each Newton L-basis can be associated with a point and derivative interpolation problem with the following properties: (i) there exists a unique solution to the general interpolation problem expressed in this basis and (ii) the coefficients a_α of the interpolant $L(\mathbf{u}) = \sum_{|\alpha|=n} a_\alpha l_\alpha^n$ expressed in the Newton L-basis are the solutions of a lower triangular system of linear equations.

In [27], we also exhibited a rich collection of lattices or point-line configurations that admit unique and natural solutions to an appropriate interpolation problem by means of a Newton L-basis. These lattices include the principal lattices or geometric meshes (and the associated Lagrange interpolation problems) as well as *natural lattices* of order n, which are defined by $n + 2$ distinct lines. The corresponding Newton L-bases solve the Lagrange interpolation problem at the associated $\binom{n+2}{2}$ distinct points of intersection. Figure 2 shows a natural lattice of order 3. [27] also presents some examples of lattices usually associated with Hermite interpolation problems [31, 37] that can be solved uniquely with Newton L-bases.

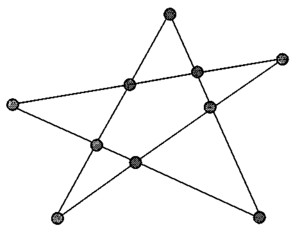

FIGURE 2. Natural lattice of order 3 for Newton L-basis

3. Up recurrence evaluation algorithms

This section describes a class of evaluation algorithms for L-bases that arise from variations on a general parallel up recurrence algorithm, which is discussed next.

3.1. **Parallel Up Recurrence Algorithm.** We first describe the parallel up recurrence algorithm and then establish its correctness. Let $L(\mathbf{u})$ be a polynomial expressed in terms of an L-basis $\{l_\alpha^n\}$ defined by the knot-net $\mathcal{L} = \{\{L_{1j}\}, \{L_{2j}\}, \{L_{3j}\}, j = 1, \cdots, n\}$. Suppose we wish to evaluate $L(\mathbf{u}) = \sum_{|\alpha|=n} S_\alpha l_\alpha^n(\mathbf{u})$ at an arbitrary but fixed $\mathbf{u}$. The parallel up recurrence algorithm uses the up recurrence illustrated in Figure 3 to evaluate $L(\mathbf{u})$, where the coefficients C_α^0 are normalized as $C_\alpha^0 = \frac{\alpha!}{n!} S_\alpha$. The diagram is to be interpreted as follows: the computation starts at

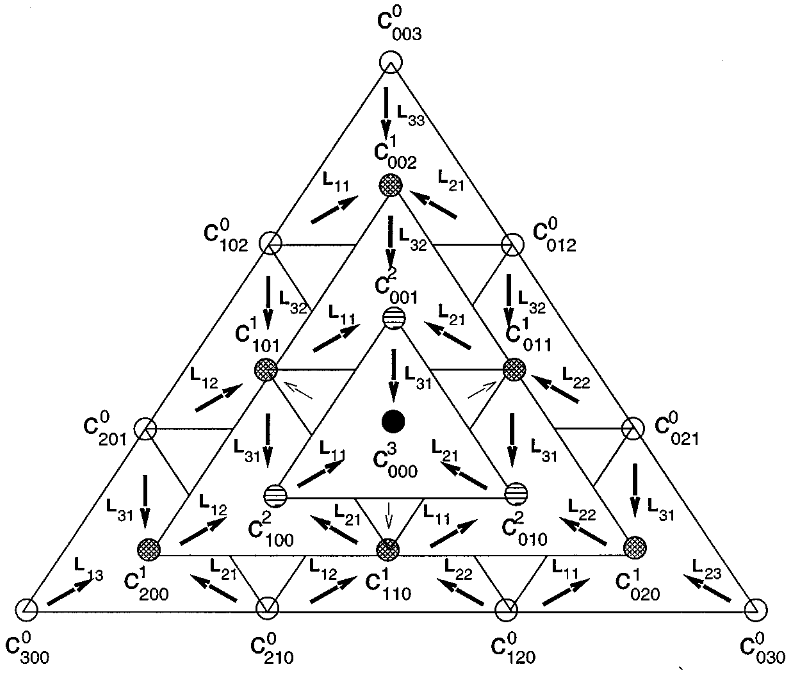

FIGURE 3. Parallel up recurrence algorithm for L-bases

the base of the tetrahedron. The value at any node is computed by multiplying the value along each arrow which enters the node by the value of the node from which the arrow emerges and adding the results. Observe that the central node of the base of the tetrahedron is occluded by the apex node in Figure 3 and is therefore not shown. The value of $L(\mathbf{u})$ is then given by C^n_{000} at the apex of the tetrahedron. More formally the recurrence is described as follows:

$$\begin{aligned} C^0_\alpha &= \frac{\alpha!}{n!} S_\alpha, \\ C^i_\alpha &= \sum_{k=1}^{3} C^{i-1}_{\alpha+e_k} * L_{k,\alpha_k+1} \text{ for } i = 1, \cdots, n, \quad \text{for} \quad |\alpha| = n - i. \end{aligned}$$

This algorithm generalizes the parallel up recurrence algorithm for the evaluation of univariate polynomials expressed in terms of Pólya basis functions [23]. The computational complexity of this algorithm is $O(n^3)$, because there are $\frac{n(n+1)(n+2)}{6}$ nodes in the tetrahedron.

We now establish the correctness of the parallel up recurrence algorithm. A careful examination of the algorithm or the recurrence equation above reveals that the label along the edge from the node $\alpha + e_i$ to the node α is L_{i,α_i+1}. This property was referred to as the *parallel* property in the univariate case in [23]. Due to this property of the recurrence diagram, the labels along the "parallel" edges of the tetrahedron are the same. For example, the labels along the edges from C^2_{100} to C^3_{000}, from C^1_{110} to C^2_{010}, and from C^0_{120} to C^1_{020} in Figure 3 are the same. By the parallel property the products of the labels along each path from a node α at the base to the node at the apex are identical. These products are all equal to l^n_α because to pass from the multi-index α to the multi-index $(0, 0, 0)$ each entry α_i must be reduced to 0; thus each of the factors $L_{i,k}$, $i = 1, 2, 3$, $0 \le k \le \alpha_i$, must be encountered exactly once along each path. Now observe that there are exactly $\frac{n!}{\alpha!}$ paths from the node α at the base of the tetrahedron to the node at the apex of the tetrahedron. Therefore, a coefficient C^0_α at the base is multiplied by the *same* value l^n_α along any of these $\frac{n!}{\alpha!}$ paths. Hence the total contribution to the apex is $\sum_{|\alpha|=n} \frac{n!}{\alpha!} C^0_\alpha l^n_\alpha$. This observation establishes the correctness of the algorithm.

3.2. **Nested Multiplication Algorithm.** The nested multiplication evaluation algorithm for L-bases arises by observing that if we do not scale the coefficients in the parallel up recurrence algorithm described in Section 3.1, the same computation can be accomplished by choosing exactly *one* path from each node at the base of the tetrahedron to the node at the apex. This structure can easily be achieved by making sure that at every level of the computation there is exactly *one* arrow pointing upwards from each node. It is remarkable that it does *not* matter which set of arrows are used at each level so long as they satisfy this uniqueness condition. Therefore, this observation gives rise to a whole class of algorithms, all of which have many fewer arrows than the general up recurrence algorithm described in the previous section. A simple symmetric rule for selecting the arrows is to choose all the arrows in the topmost upright triangles, the two lower arrows on the rightmost triangles and the leftmost arrow in all the remaining triangles as shown in Figure 4. However, although the number of arrows is reduced with this choice, the computational complexity of the algorithm still remains $O(n^3)$ because we have removed slightly less than $\frac{2}{3}$ of the arrows.

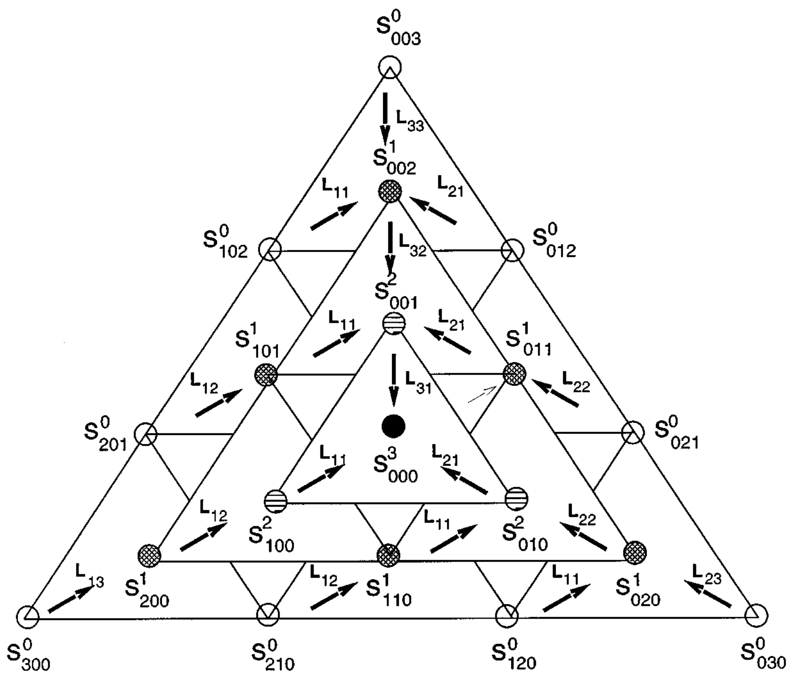

FIGURE 4. Nested multiplication evaluation algorithm for L-bases

Carnicer and Gasca [5] describe a class of evaluation algorithms for those multivariate polynomials that are expressed as the sum of a constant plus some other polynomials, each of which can be written as a product of a linear polynomial with a polynomial of degree strictly lower than the degree of the original polynomial. Polynomials represented in terms of L-bases clearly satisfy this property. Therefore, the evaluation algorithms of Carnicer and Gasca can be applied to L-bases. However their algorithm and the resulting computational complexity depend upon the way the polynomial is represented. In general, the computational complexity of their algorithm for bivariate polynomial bases including bivariate Lagrange bases [6] is $O(n^3)$. We shall refer to this class of algorithms, including up recurrence algorithms and algorithms described by Carnicer and Gasca, as nested multiplication algorithms because these algorithms can clearly be viewed as nested multiplication. Both Carnicer and Gasca [5] and de Boor and Ron [12] describe this class of nested multiplication algorithms as the generalization of Horner's nested multiplication algorithm for the evaluation of univariate polynomials. However, as we shall see later in Section 4, there is another *different* class of nested multiplication algorithms for the evaluation of multivariate polynomials that also qualify as generalizations of Horner's algorithm. Therefore, it is not clear which of these algorithms can claim to be *the* generalization of Horner's evaluation algorithm, although all of them are a form of nested multiplication.

We can reduce the computational complexity of the our nested multiplication algorithm from $O(n^3)$ to $O(n^2 \log n)$ by observing that it is *not* necessary to have an arrow emerging from a node if there are no arrows entering that node. Such a node will be referred to as a *dead* node. Other nodes will be referred to as *active* nodes. To reduce the computational complexity of the algorithm, we desire to increase the

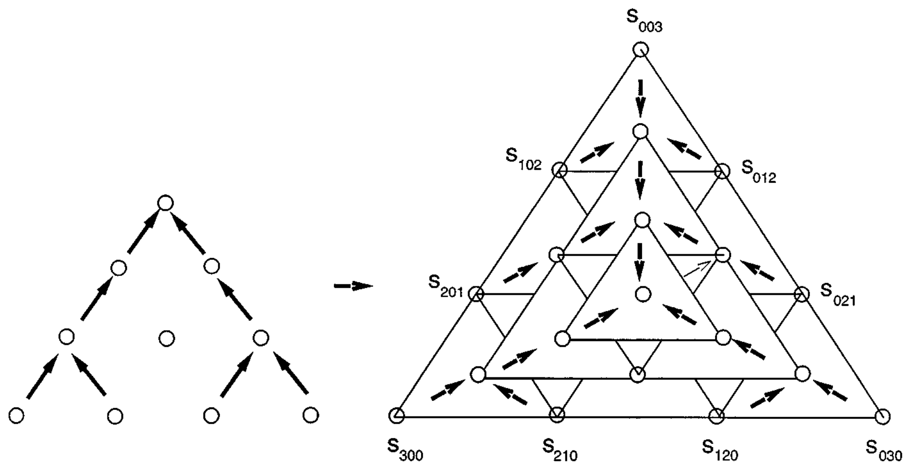

FIGURE 5. Nested multiplication evaluation algorithm for L-bases

number of dead nodes as much as possible. One way to achieve this end is to point as many arrows as possible towards the corners and towards the sides rather than towards the center.

We now briefly illustrate how this idea can be used to reduce the computational complexity of the analogous evaluation algorithm for univariate polynomials from $O(n^2)$ to $O(n \log n)$. This problem is purely combinatorial; one simply needs to choose enough arrows so that there is exactly one path from every node at the base to the node at the apex. The left diagram of Figure 5 shows how this can be achieved for univariate polynomials when $n = 3$. This approach can be generalized to polynomials of arbitrary degree n by a divide and conquer strategy. Here a problem of size n is broken down into two subproblems of size $\lfloor \frac{n}{2} \rfloor$ at the base followed by at most $\lceil \frac{n}{2} \rceil$ arrows on both sides of the triangle. An example for $n = 7$ is shown in Figure 6, where evaluation of a univariate polynomial of degree 7 is broken down into two subproblems of evaluation for univariate polynomials of degree 3 at the base. This approach yields the following recurrence equation for the computational complexity $T(n)$:

$$T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + O(n).$$

Solving this recurrence, we obtain $T(n) = n \log n$, which establishes the claim for univariate polynomials.

To generalize this technique to the bivariate case, we run the univariate algorithm layer by layer as illustrated in the right diagram of Figure 5. For the nodes in the frontmost layer of the base of the tetrahedron, we run an evaluation algorithm for a univariate L-basis of degree n. For the layer behind it at the base of the tetrahedron, we run the evaluation algorithm for a univariate L-basis of degree $n - 1$ and so on. Thus we run evaluation algorithms for univariate L-bases of degrees all the way from n down to 1. This approach yields values at all the nodes along one of the edges of the tetrahedron, namely the back edge. We now simply put arrows along this back edge to compute the value at the apex of the tetrahedron. These new arrows require an additional $O(n)$ computations. This method yields the following

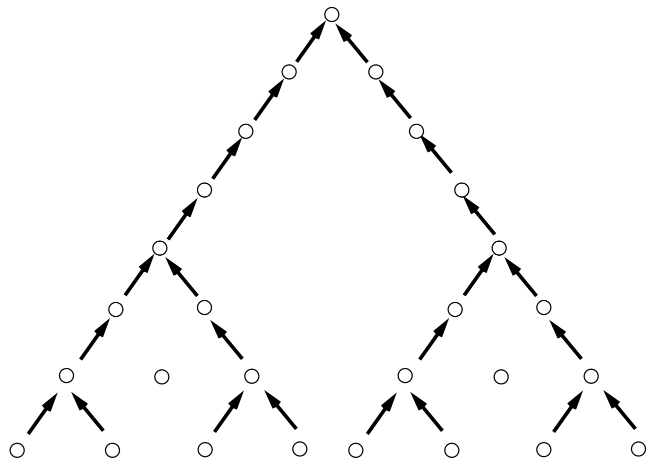

FIGURE 6. Nested multiplication evaluation algorithm for univariate L-bases of degree 7

equation for the computational complexity $T(n)$:

$$T(n) = \sum_{k=1}^{n} O(k \log k) + O(n).$$

Therefore, the total computational complexity of this algorithm is $O(n^2 \log n)$. Although this algorithm seems to give the best possible computational complexity that one can obtain by removing arrows from a parallel up recurrence algorithm, it does not give the best possible constant because one can easily figure out other ways of removing arrows which actually yield fewer arrows than the algorithm described above. However we found that this description was the easiest way to provide a simple proof that by removing arrows one can obtain an evaluation algorithm with computational complexity $O(n^2 \log n)$.

3.3. **Examples.** This section describes how the parallel up recurrence algorithm and the nested multiplication algorithm for evaluation of bivariate L-bases can be specialized to obtain evaluation algorithms for bivariate Bernstein-Bézier bases, multinomial bases, Lagrange L-bases and Newton L-bases.

3.3.1. *Multinomial Evaluation.* Since one of the knot-nets in the multinomial basis is always 1, considerable simplifications take place in both the parallel up recurrence algorithm and the nested multiplication algorithm for evaluation with respect to the multinomial bases. In this case it pays to choose arrows with the label 1 as often as possible. The left diagram of Figure 7 shows such a choice, where the arrows pointing downward have the label 1. Since no multiplication needs to be done for arrows with the label 1, one can "pull up" these nodes as shown in the right diagram of Figure 7. The value at any node is now computed by adding the value at that node to the value computed as before by multiplying the label along each arrow which enters the node by the value of the node from which the arrow emerges and adding the results. The right diagram of Figure 7 shows the same computation as in the left diagram, but now arranged in a triangular format. Since the triangular format has only n^2 nodes, this figure shows that the computational

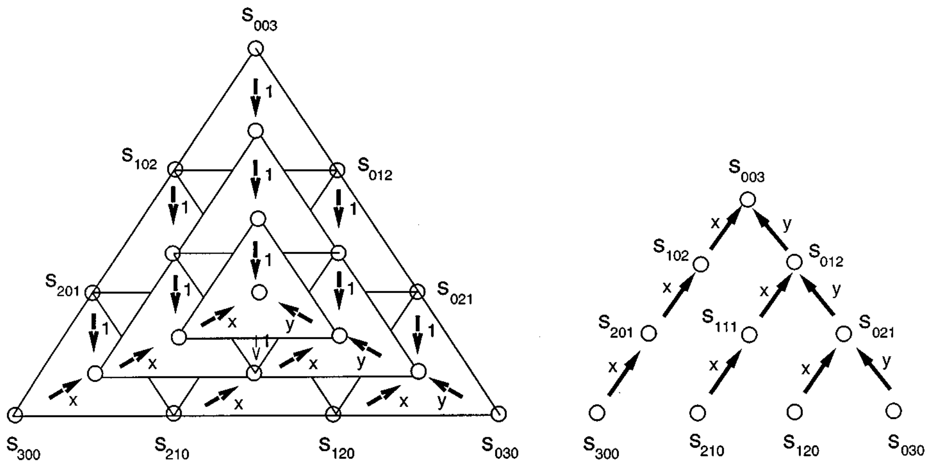

FIGURE 7. Efficient evaluation algorithm for multinomial bases

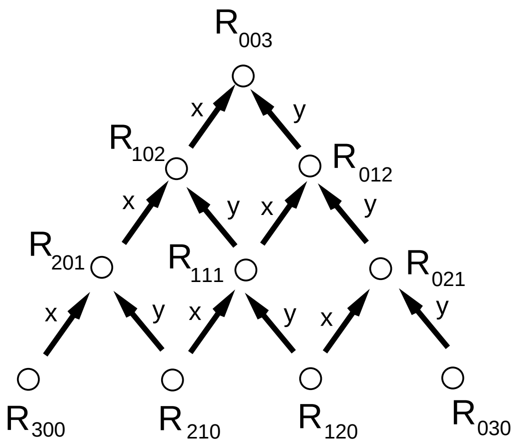

FIGURE 8. de Boor-Ron's evaluation algorithm for multinomial bases

complexity of this evaluation algorithm is $O(n^2)$. Although Carnicer and Gasca [5] do not explicitly discuss the evaluation of multivariate polynomials expressed in the multinomial bases, the right diagram of Figure 7 is very similar to the diagram in [6], which appears in an analogous but slightly different context. Therefore, this evaluation algorithm can be construed to be equivalent to the one proposed by Carnicer and Gasca [5, 6].

Interestingly, de Boor and Ron [12] describe an evaluation algorithm for multinomial bases that is closely related to these algorithms, although not quite the same. In fact, their algorithm is a hybrid of the parallel up recurrence and the efficient parallel up recurrence algorithm, where some duplicate paths have not been removed. They take advantage of the arrows with label 1 and "pull up" the nodes, but they do not remove redundant paths. Therefore, their starting coefficients are slightly different from both C_α and S_α. In fact, their starting coefficients are $R_\alpha = \frac{\alpha_1!\alpha_2!}{(\alpha_1+\alpha_2)!}S_\alpha$. The diagram of Figure 8 shows the evaluation recurrence of

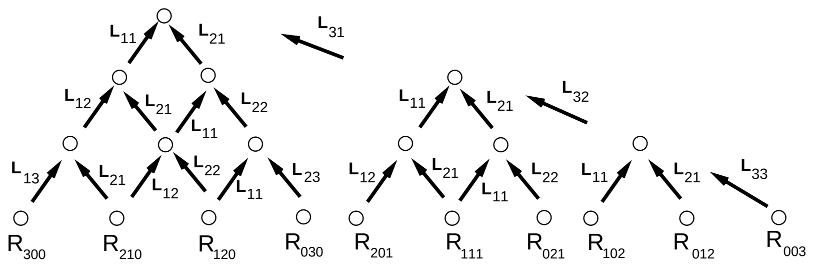

FIGURE 9. Evaluation algorithm for L-bases with coefficients R_α

de Boor and Ron for the evaluation of multinomial bases. This algorithm also has computational complexity of $O(n^2)$, although with a larger constant.

In fact, following this strategy of using the coefficients R_α instead of the coefficients C_α or S_α, one can write down yet another form of nested multiplication algorithm for the evaluation of multivariate polynomials expressed in terms of L-bases. This algorithm is illustrated in Figure 9. Observe that the tetrahedral structure of the parallel up recurrence diagram in Figure 3 can be shown conveniently as in Figure 9, where the different layers (a layer being defined as nodes α with same α_3 value) of the tetrahedral structure are shown side-by-side. Since there is exactly one edge between different layers of this diagram, there are no "cross-edges" as there are in Figure 3. Therefore, the side-by-side diagram appears uncluttered. The algorithm in Figure 9 utilizes R_α as coefficients because some of the duplicate paths have not been removed. This algorithm has computational complexity $\sum_{k=1}^{n} O(k^2) = O(n^3)$. By removing all the duplicate paths and using the coefficients S_α, one can redraw the right diagram of Figure 5 as Figure 10. This algorithm has computational complexity $\sum_{k=1}^{n} O(k \log k) = O(n^2 \log n)$.

3.3.2. *Bernstein-Bézier Evaluation.* The parallel up recurrence algorithm specializes to the de Casteljau evaluation algorithm for Bernstein-Bézier surfaces with computational complexity $O(n^3)$ [13, 14]. In this case, observe that $C_\alpha^0 = \frac{\alpha!}{n!} S_\alpha$ are indeed the coefficients of the multivariate polynomials in Bernstein-Bézier form.

There is another interesting way of evaluating polynomials of total degree n expressed in terms of Bernstein-Bézier bases by viewing the polynomial as a tensor product of bi-degree $n \times n$ [47]. This tensor product evaluation algorithm for the standard Bernstein-Bézier basis can be derived by reorganizing the computation as

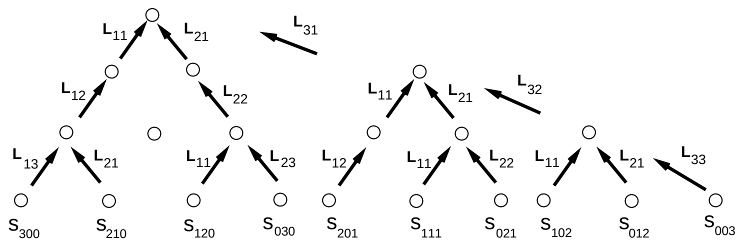

FIGURE 10. Another diagram for the evaluation algorithm for L-bases with coefficients S_α

follows:

$$\begin{aligned}
L(\mathbf{u}) &= \sum_{|\alpha|=n} \frac{n!}{\alpha!} C_\alpha x^{\alpha_1} y^{\alpha_2} (1-x-y)^{\alpha_3} \\
&= \sum_{|\alpha|=n} \frac{n!}{\alpha!} C_\alpha \left(\frac{x}{x+y}\right)^{\alpha_1} \left(\frac{y}{x+y}\right)^{\alpha_2} (x+y)^{\alpha_1+\alpha_2}(1-x-y)^{\alpha_3} \\
&= \sum_{|\alpha|=n} \frac{n!}{\alpha_3!(\alpha_1+\alpha_2)!} \frac{(\alpha_1+\alpha_2)!}{\alpha_1!\alpha_2!} C_\alpha \left(\frac{x}{x+y}\right)^{\alpha_1} \left(\frac{y}{x+y}\right)^{\alpha_2} \\
&\quad (x+y)^{\alpha_1+\alpha_2}(1-x-y)^{\alpha_3} \\
&= \sum_{k=0}^{n} \frac{n!}{\alpha_3!(\alpha_1+\alpha_2)!} \left(\sum_{\alpha_1+\alpha_2=k} \frac{(\alpha_1+\alpha_2)!}{\alpha_1!\alpha_2!} C_{\alpha_1,\alpha_2,n-k} (1-s)^{\alpha_1} s^{\alpha_2} \right) \\
&\quad (1-t)^{\alpha_1+\alpha_2} t^{\alpha_3},
\end{aligned}$$

where $s = \frac{y}{x+y}$ and $t = (1-x-y)$.

The inner sum in the above computation can be evaluated using the univariate de Casteljau evaluation algorithm for degrees 0 to n. These univariate de Casteljau algorithms for degree 3 are shown in the bottom part of Figure 11 where the parameter s appears. The outer sum in the above computation is another univariate de Casteljau evaluation algorithm for degree n. This evaluation algorithm is shown in the top part of Figure 11 where the parameter t appears.

This tensor product algorithm has $\frac{n(n+1)(n+5)}{3}$ arrows for the evaluation of polynomials of degree n. Therefore this algorithm is still $O(n^3)$. However observe that the standard bivariate de Casteljau algorithm for the evaluation of polynomials of total degree n has $\frac{n(n+1)(n+2)}{2}$ arrows, as shown in Figure 3. Therefore the tensor

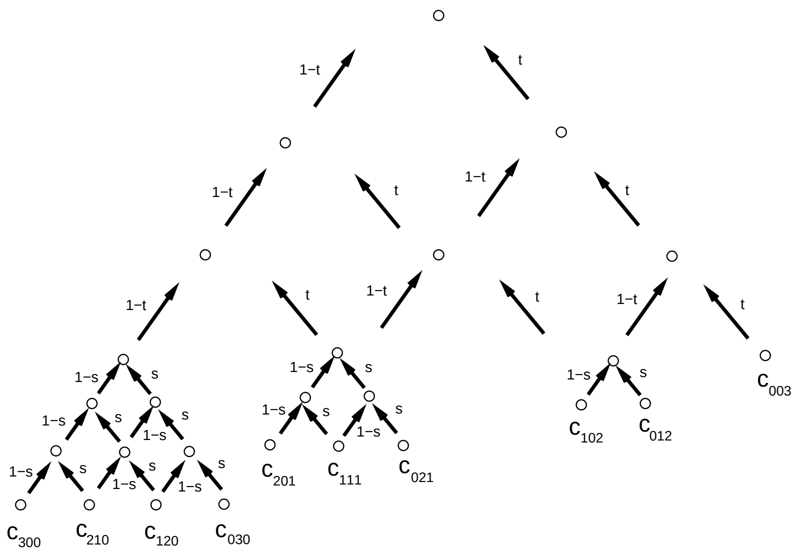

FIGURE 11. Tensor product evaluation algorithm for Bernstein-Bézier polynomials

product algorithm is more efficient than the standard bivariate de Casteljau algorithm for the evaluation of polynomials of total degree greater than or equal to 4.

3.3.3. *Newton Evaluation.* Since all the polynomials in one of the sequences in the knot-net of a Newton basis are 1, considerable simplifications take place as well in both the parallel up recurrence algorithm and the nested multiplication algorithm for the evaluation of Newton bases. In this case too it pays to choose arrows with labels 1 as often as possible. The left diagram of Figure 12 shows such a choice, where the arrows pointing downwards have the label 1. Since no multiplication needs to be done for arrows with labels 1, one can "pull up" these nodes as shown in the right diagram of Figure 12. The right diagram of Figure 12 shows the same computation as in the left diagram, but now arranged in a triangular format. Since the triangular format has only $O(n^2)$ nodes, this figure shows that the computational complexity of this evaluation algorithm is $O(n^2)$. This algorithm is exactly the same as the evaluation algorithm described by Gasca in [19].

3.3.4. *Lagrange Evaluation: Generalization of the Aitken-Neville Algorithm.* We now derive a variation of the parallel up recurrence diagram for the evaluation of Lagrange L-bases, which gives rise to a bivariate generalization of the univariate Aitken-Neville algorithm. In the past the Aitken-Neville algorithm has been generalized to some restricted classes of bivariate Lagrange polynomials [1, 36, 19, 6, 5, 34, 20, 22, 33, 44, 35]. This new generalization extends the class of multivariate Lagrange polynomials to which an Aitken-Neville algorithm can be applied.

Given a knot-net $\mathcal{L} = \{\{L_{1j}\}, \{L_{2j}\}, \{L_{3j}\}, j = 1, \cdots, n\}$ of linear polynomials, consider the three knot-nets $\mathcal{M}_1 = \{(\hat{L}_{11}, \cdots, L_{1n}), (L_{21}, \cdots, \hat{L}_{2n}), (L_{31}, \cdots, \hat{L}_{3n})\}$, $\mathcal{M}_2 = \{(L_{11}, \cdots, \hat{L}_{1n}), (\hat{L}_{21}, \cdots, L_{2n}), (L_{31}, \cdots, \hat{L}_{3n})\}$, and $\mathcal{M}_3 = \{(L_{11}, \cdots, \hat{L}_{1n}), (L_{21}, \cdots, \hat{L}_{2n}), (\hat{L}_{31}, \cdots, L_{3n})\}$ respectively, where $\hat{L}$ means that the term L is missing. Observe that if $\mathcal{L}$ satisfies the linear independence condition on the knot-net of polynomials for $0 \le |\alpha| \le n-1$, then the knot-net $\mathcal{M}_1$ satisfies the linear independence condition for $0 \le |\beta| \le n-2$. This

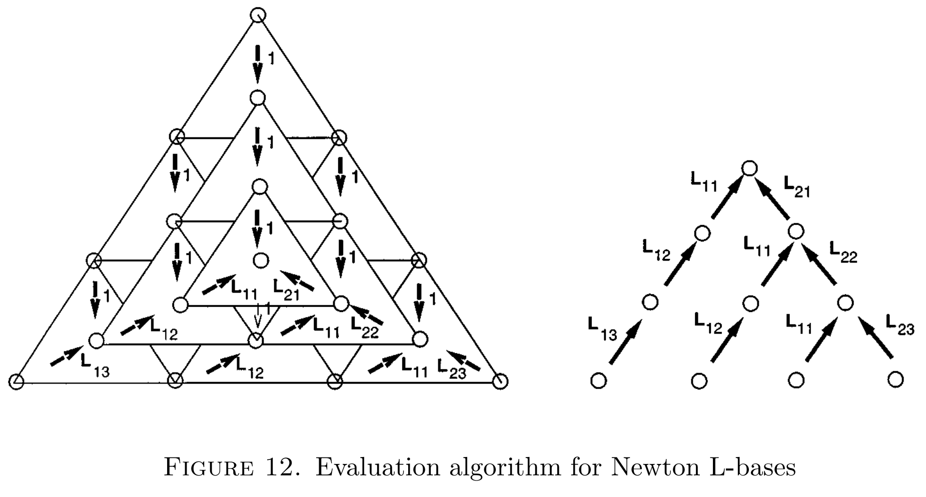

FIGURE 12. Evaluation algorithm for Newton L-bases

observation follows by setting $\alpha_1 = \beta_1 + 1$, $\alpha_2 = \beta_2$ and $\alpha_3 = \beta_3$. Similarly the knot-nets $\mathcal{M}_2$ and $\mathcal{M}_3$ satisfy the linear independence condition for $0 \le |\beta| \le n-2$. Moreover, if the knot-net $\mathcal{L}$ satisfies the linear dependence condition for a Lagrange L-basis, then the three knot-nets $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ also satisfy the linear dependence condition for a Lagrange L-basis. Let $\{l^n_\alpha\}$, $\{p^{n-1}_{1,\alpha}\}$, $\{p^{n-1}_{2,\alpha}\}$ and $\{p^{n-1}_{3,\alpha}\}$ be the L-bases corresponding to the knot-nets $\mathcal{L}$, $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ respectively.

We can associate point interpolation problems to these knot-nets as follows. Let $L(\mathbf{u})$ be the unique polynomial of degree n that interpolates the values S_α at the points $\mathbf{v}_\alpha$, where the points $\mathbf{v}_\alpha$ are the points of intersection of certain lines from the collection $\mathcal{L}$ as described in Section 2.3. Then, $L(\mathbf{u}) = \sum_{|\alpha|=n} S_\alpha \frac{l^n_\alpha}{l^n_\alpha(\mathbf{v}_\alpha)}$. Similarly let $M_1(\mathbf{u})$, $M_2(\mathbf{u})$ and $M_3(\mathbf{u})$ be the unique polynomials of degree $n-1$ that satisfy the point interpolation conditions $M_i = S_{\alpha+e_i}\delta_{\alpha\beta}$ for $|\alpha| = n-1$. Then $M_i(\mathbf{u}) = \sum_{|\alpha|=n-1} S_{\alpha+e_i} \frac{p^{n-1}_{i,\alpha}}{p^{n-1}_{i,\alpha}(\mathbf{v}_\alpha)}$.

Now create the following variation of the parallel up recurrence algorithm: Replace the labels L_{k,α_k} by the labels $\frac{L_{k,\alpha_k}}{L_{k,\alpha_k}(\mathbf{v}_{\alpha+le_k})}$ on the arrows pointing from the node C^l_α to the node $C^{l+1}_{\alpha-e_k}$. Then we can express the solution to the point interpolation problem as a linear combination of the solutions to the three subproblems of point interpolation and thus generalize the Aitken-Neville algorithm for the evaluation of Lagrange polynomials from univariate to bivariate polynomials [1, 36].

Theorem 3.1.

$$L(\mathbf{u}) = \frac{L_{11}}{L_{11}(\mathbf{v}_{n00})} M_1(\mathbf{u}) + \frac{L_{21}}{L_{21}(\mathbf{v}_{0n0})} M_2(\mathbf{u}) + \frac{L_{31}}{L_{31}(\mathbf{v}_{00n})} M_3(\mathbf{u}).$$

Proof. The proof is by induction. The case $n = 1$ reduces to a simple triangular point interpolation problem and in this case the linear solution $L(\mathbf{u})$ is indeed a barycentric combination of constant solutions given by

$$L(\mathbf{u}) = \frac{L_{11}}{L_{11}(\mathbf{v}_{100})} C_{100} + \frac{L_{21}}{L_{21}(\mathbf{v}_{010})} C_{010} + \frac{L_{31}}{L_{31}(\mathbf{v}_{001})} C_{001}.$$

The inductive hypothesis assumes that the statement of the theorem is true for $n-1$. We will now prove that the statement of the theorem holds for n. First, observe that for $i = 1, 2, 3$

$$M_i(\mathbf{u}) = \sum_{|\alpha|=n-1} S_{\alpha+e_i} \frac{p^{n-1}_{i,\alpha}}{p^{n-1}_{i,\alpha}(\mathbf{v}_\alpha)}. \tag{3.1}$$

The inductive hypothesis asserts that the values at the three nodes that contribute to the apex of the tetrahedron, which are simply the apexes of the recurrence diagrams of the three sub-problems, are $M_1(\mathbf{u})$, $M_2(\mathbf{u})$ and $M_3(\mathbf{u})$. Now let

$$M(\mathbf{u}) = \frac{L_{11}}{L_{11}(\mathbf{v}_{n00})} M_1(\mathbf{u}) + \frac{L_{21}}{L_{21}(\mathbf{v}_{0n0})} M_2(\mathbf{u}) + \frac{L_{31}}{L_{31}(\mathbf{v}_{00n})} M_3(\mathbf{u}). \tag{3.2}$$

We will prove that $M(\mathbf{u}) = L(\mathbf{u})$. Substituting Equation 3.1 into Equation 3.2 and recalling also that

$$L_{i1} p^{n-1}_{i,\alpha} = l^n_{\alpha+e_i},$$

we obtain

$$M(\mathbf{u}) = \sum_{|\alpha|=n} S_\alpha \frac{l^n_\alpha}{l^n_\alpha(\mathbf{v}_\alpha)} I(\mathbf{v}_\alpha),$$

where

$$I(\mathbf{v}) = \frac{L_{11}(\mathbf{v})}{L_{11}(\mathbf{v}_{n00})} + \frac{L_{21}(\mathbf{v})}{L_{21}(\mathbf{v}_{0n0})} + \frac{L_{31}(\mathbf{v})}{L_{31}(\mathbf{v}_{00n})}.$$

To show that $I = 1$, we observe that I is a linear polynomial, which evaluates to 1 at three affinely independent (that is, non-collinear) points $\mathbf{v}_{n00}$, $\mathbf{v}_{0n0}$ and $\mathbf{v}_{00n}$; therefore I is identically equal to 1. This proves that $M(\mathbf{u}) = L(\mathbf{u})$ and establishes the theorem. □

3.4. **Ladder Recurrence Algorithm.** By removing redundant arrows, we were able to design an $O(n^2 \log n)$ algorithm for the evaluation of bivariate L-bases in Section 3.2. Can we do better? The answer is yes, although we must modify still further the structure of the up recurrence algorithm. In Section 3.2, we derived an $O(n^2 \log n)$ algorithm by extending a univariate evaluation algorithm with computational complexity $O(n \log n)$. By modifying the structure of the univariate parallel up recurrence, Warren developed a ladder recurrence algorithm for the evaluation of univariate L-bases (or Pólya bases) with computational complexity $O(n)$ [48]. This univariate technique can be extended to bivariate L-bases and yields a ladder recurrence algorithm for the evaluation of bivariate L-bases with computational complexity $O(n^2)$ [28].

We first describe the ladder recurrence algorithm and then sketch a proof of its correctness. This ladder recurrence algorithm is obtained by replacing each triangle of height p in Figure 10 by a "ladder" of height p that yields the same result. Figure 13 presents an example of the bivariate ladder recurrence algorithm for degree 3. The recurrence starts at the bottom of Figure 13, at all the nodes shown as cross-hatched circles. These nodes have values 1. As before, the value at any node is computed by multiplying the value along each arrow that enters the node by the value of the node from which the arrow emerges and adding the results. The computation proceeds upwards and the value of $L(\mathbf{u})$ emerges at the apex node of the triangle, shown as a black circle in Figure 13. Since a ladder with p steps requires $O(p)$ computations, the computational complexity of this algorithm is $\sum_{p=1}^{n} O(p) = O(n^2)$. Observe that in contrast to other algorithms described in this work, this algorithm employs coefficients as labels along the edges.

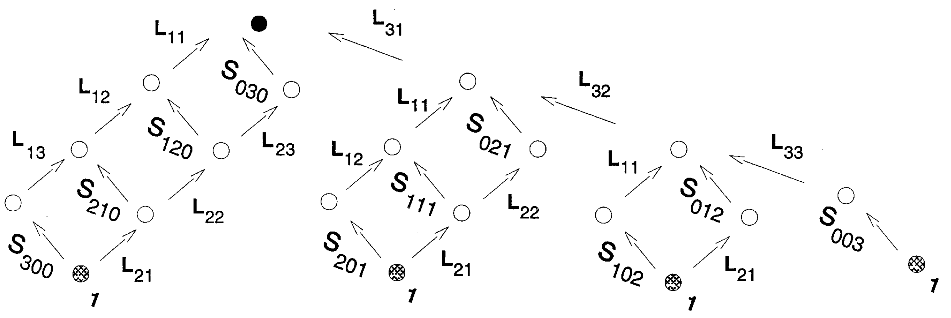

FIGURE 13. Ladder recurrence algorithm for bivariate L-bases

To establish the correctness of the ladder recurrence algorithm, observe that this algorithm reorganizes the computation as follows:

$$\begin{aligned} L(\mathbf{u}) &= \sum_{|\alpha|=n} S_\alpha l_\alpha^n \\ &= \sum_{k=0}^{n} L_{31}\cdots L_{3k} \sum_{\alpha_1+\alpha_2=n-k} S_{\alpha_1,\alpha_2,k} L_{11}\cdots L_{1\alpha_1} L_{21}\cdots L_{2\alpha_2} \\ &= \sum_{k=0}^{n} L_{31}\cdots L_{3k} P_k, \end{aligned}$$

where $P_k = \sum_{\alpha_1+\alpha_2=n-k} S_{\alpha_1,\alpha_2,k} L_{11}\cdots L_{1\alpha_1} L_{21}\cdots L_{2\alpha_2}$. The multiplications by $L_{3,\cdot}$'s are shown along the diagonal in Figure 13. The computation of P_k can be arranged as a ladder of size $n-k$ where $L_{1,\cdot}$'s appear on the left side of the ladder and $L_{2,\cdot}$'s appear on the right side of the ladder indexed in reverse order. The S_α's appear along the rungs of the ladder. This situation is illustrated in Figure 13 for the evaluation of bivariate L-bases of degree 3. The coefficient $S_{\alpha_1,\alpha_2,k}$ is placed on the rung α_1 steps below the top and α_2 steps above the bottom of the kth ladder. Thus in the ladder computation $S_{\alpha_1,\alpha_2,k}$ gets multiplied by the factor $L_{11}\cdots L_{1,\alpha_1}$ along the left edge of the ladder and by the factor $L_{21}\cdots L_{2,\alpha_2}$ along the right edge of the ladder, yielding the desired result for P_k. A more formal description of this algorithm as well as a more rigorous proof of the correctness of the algorithm using induction can be found in [28].

4. Down recurrence evaluation algorithms

So far we have established that a large class of evaluation algorithms for multivariate polynomials expressed in terms of L-bases can be obtained as variations of parallel up recurrence algorithms. In this section, we want to establish that another large class of evaluation algorithms can be obtained as variations of certain change of basis algorithms for L-bases. In Section 4.5, we establish that a divided difference algorithm and a forward difference algorithm both with computational complexity $O(n^2)$ for evaluation of bivariate polynomials can be viewed as change of basis algorithms from a Newton L-basis to another Newton L-basis. In Section 4.4, we obtain a nested multiplication evaluation algorithm with computational complexity $O(n^2)$ by specializing the change of basis algorithm between L-bases. This nested multiplication evaluation algorithm is different than the one presented in Section 3.2 which, in general, has computational complexity $O(n^2 \log n)$. More interestingly, we also present in Section 4.3 a Lagrange evaluation algorithm for evaluating bivariate polynomials of degree n with amortized computational complexity $O(n)$ per point that can be obtained as a change of basis algorithm from an arbitrary L-basis to a Lagrange L-basis. Thus this class of evaluation algorithms are tied together conceptually and are different than the class of evaluation algorithms based on up recurrence algorithms discussed in Section 3.

Now, a note on terminology. We could refer to this class of evaluation algorithms as “change-of-basis” evaluation algorithms because, as stated above, they are all derived by considering certain change of basis algorithms between L-bases. However, we have decided to refer to these algorithms as *down recurrence evaluation algorithms* for two reasons: (i) First, we shall soon see that in the most general case the computation in these algorithms has been arranged in such a way that the original coefficients lie on a lateral face of the tetrahedron and the computation proceeds *downwards*. (ii) Second, this terminology contrasts most appropriately with the *up* recurrence algorithms described in Section 3.

4.1. **Conceptual Overview.** In order to describe the underlying unity of this class of evaluation algorithms, we need to discuss certain change of basis algorithms between L-bases. In contrast to the parallel up recurrence evaluation algorithms for L-bases discussed in Section 3.1, the description and proof of which are fairly easy, the change of basis algorithms for L-bases are deeper and have been discovered only recently [26]. These change of basis algorithms were originally derived using an elegant theory combining blossoming, homogenization and duality [39, 40, 43, 7, 26, 27]. To keep this paper self-contained, here we provide a new, more direct proof.

In the following discussion, we shall assume that we are given the coefficients R_α of a polynomial L of degree n with respect to the L-basis $\{l^n_\alpha\}$ defined by the knot-net $\mathcal{L} = \{\{L_{1j}\}, \{L_{2j}\}, \{L_{3j}\}\}$, $j = 1, \cdots, n$. We would like to compute the coefficients U_α of this polynomial L with respect to another L-basis $\{m^n_\alpha\}$ defined by another knot-net $\mathcal{M} = \{\{M_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, \cdots, n\}$.

Below we shall use the term *lineal* polynomial to mean any polynomial of degree n that is a product of n linear factors. Observe that every element of an L-basis is a lineal polynomial.

There are two crucial elements in deriving these change of basis algorithms between L-bases: (i) an up recurrence diagram for L-bases, that represents the basis functions of one L-basis in terms of the basis functions of another closely related L-basis, and (ii) a technique to convert this up recurrence diagram into a down recurrence diagram that represents the coefficients of a polynomial with respect to one L-basis in terms of the coefficients of the same polynomial with respect to another closely related L-basis.

In the following discussion, we shall also employ the "intermediate" L-bases $\{p^n_\alpha\}$ and $\{q^n_\alpha\}$ defined respectively by the knot-nets $\{\{L_{1j}\}, \{L_{2j}\}, \{M_{3j}\}, j = 1, \cdots, n\}$ and $\{\{L_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, \cdots, n\}$. In particular, we shall need to assume that these "intermediate" knot-nets satisfy the linear independence condition; that is, $\{L_{1,\alpha_1+1}, L_{2,\alpha_2+1}, M_{3,\alpha_3}\}$ and $\{L_{1,\alpha_1+1}, M_{2,\alpha_2+1}, M_{3,\alpha_3}\}$ are linearly independent polynomials for $0 < |\alpha| \leq n-1$.

An up recurrence diagram for L-bases: We shall introduce a sequence of *three* up recurrence diagrams. The first up recurrence diagram relates the L-bases $\{l^n_\alpha\}$ and $\{p^n_\alpha\}$. This up recurrence diagram is shown in Figure 14 for the cubic case. The L-basis functions $\{p^n_\alpha\}$ appear at the nodes on the base of the tetrahedron. The L-basis functions $\{l^n_\alpha\}$ emerge at the nodes on a lateral face of the tetrahedron. The computation proceeds upwards and the value at any node is computed by multiplying the value along each arrow which enters the node by the value of the node from which the arrow emerges and adding the results as in the parallel up recurrence algorithm described in Section 3.1. For every node of the tetrahedron, there are three polynomials one level below from which the polynomial at the node is computed. Observe that the first two sequences of the knot-net of the L-bases $\{l^n_\alpha\}$ and $\{p^n_\alpha\}$ are the same. Therefore, the three lineal polynomials contributing to any node have all common factors except precisely one. These three non-common factors L_{1,α_1+1}, L_{2,α_2+1}, and M_{3,α_3} are linearly independent. Therefore, any linear polynomial can always be expressed as a linear combination of these three linear

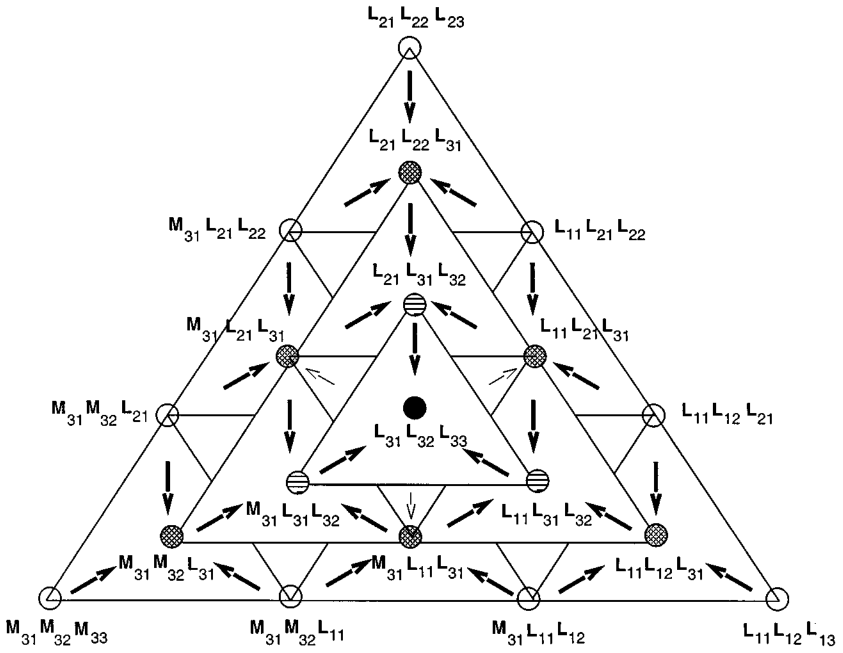

FIGURE 14. An up recurrence diagram for L-bases

polynomials. In particular, one can write any $L_{3,i}$ as a linear combination of these three polynomials, that is,

$$L_{3,i} = g_{1,\alpha}L_{1,\alpha_1+1} + g_{2,\alpha}L_{2,\alpha_2+1} + g_{3,\alpha}M_{3,\alpha_3+1}.$$

The multipliers $g_{k,\alpha}$ are then precisely the labels along the edges of the tetrahedron. These labels are not shown in Figure 14 to avoid cluttering the diagram. However, these labels are shown in Figure 15 for the quadratic case and are explained further in greater detail in the next section.

The second up recurrence diagram is very similar to the first up recurrence diagram. This diagram relates the L-bases $\{p^n_\alpha\}$ and $\{q^n_\alpha\}$. Observe that the first and the third sequence of knot-nets of these two L-bases are the same. Therefore, by an argument similar to the one presented in the previous paragraph, it is possible to build an up recurrence diagram for these two L-bases. Here the L-basis functions $\{q^n_\alpha\}$ appear at the nodes on the base, and the L-basis functions $\{p^n_\alpha\}$ emerge at the nodes on a lateral face of the tetrahedron.

Finally, in the third recurrence diagram, which is similar to the first two, the L-basis functions $\{m^n_\alpha\}$ appear at the nodes on the base of the tetrahedron and the L-basis functions $\{q^n_\alpha\}$ emerge at the nodes on a lateral face of the tetrahedron. Taken together, these three tetrahedra represent a transformation T between two arbitrary L-bases $\{l^n_\alpha\}$ and $\{m^n_\alpha\}$.

Conversion to down recurrence diagram: Now suppose L is a polynomial of degree n. Then L can be represented in terms of the L-bases $\{l^n_\alpha\}$, $\{p^n_\alpha\}$, $\{q^n_\alpha\}$, and $\{m^n_\alpha\}$. More precisely, let $L = \sum_{|\alpha|=n} R_\alpha l^n_\alpha = \sum_{|\alpha|=n} S_\alpha p^n_\alpha = \sum_{|\alpha|=n} T_\alpha q^n_\alpha = \sum_{|\alpha|=n} U_\alpha m^n_\alpha$. Given R_α, we are going to use three down recurrence algorithms to compute S_α, T_α, and U_α respectively.

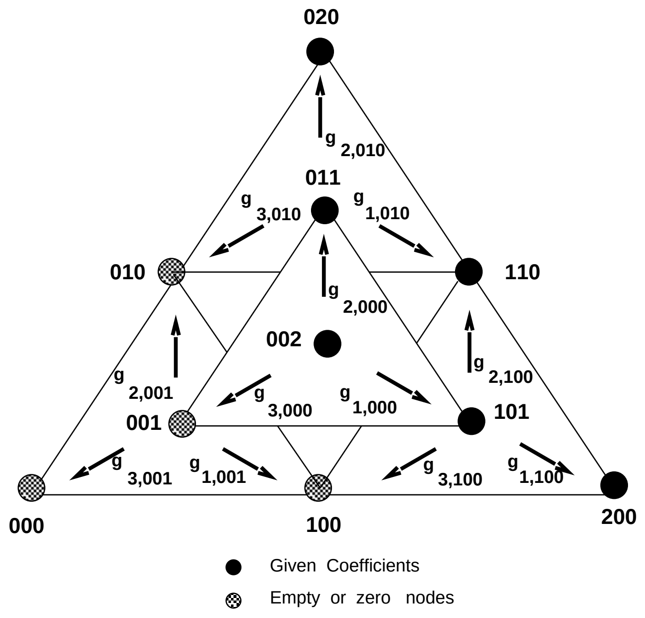

FIGURE 15. Labeling of the tetrahedron

To this purpose, observe that the first up recurrence diagram represents the matrix T, that computes $\{l_\alpha^n\}$ from $\{p_\alpha^n\}$, that is,

$$[l_\alpha^n] = T \cdot [p_\alpha^n],$$

where $[l_\alpha^n]$ and $[p_\alpha^n]$ are column vectors, T is a square matrix and $\cdot$ represents matrix multiplication. Suppose $[R_\alpha]$ and $[S_\alpha]$ are row vectors representing the coefficients $\{R_\alpha\}$ and $\{S_\alpha\}$ respectively. Then

$$[S_\alpha] \cdot [p_\alpha^n] = [R_\alpha] \cdot [l_\alpha^n] = [R_\alpha] \cdot T \cdot [p_\alpha^n].$$

Since p_α^n is a basis, we have $[S_\alpha] = [R_\alpha] \cdot T$ or equivalently taking the transpose, $[S_\alpha]^t = T^t \cdot [R_\alpha]^t$. In other words, the transformation from the coefficients R_α to the coefficients S_α is the transpose of the transformation T between the *opposite* bases $\{p_\alpha^n\}$ and $\{l_\alpha^n\}$.

We now establish that the transformation T^t can be achieved by taking the up recurrence diagram that represents the matrix transformation T, keeping the same labels and *reversing* the arrows. To begin, observe that $T_{\alpha\beta}$ is the sum over all the paths of the product of the labels along the paths from the node α at the base of the tetrahedron to the node β along the lateral face of the tetrahedron. If $T^{'}$ represents the matrix for the same diagram with the arrows reversed, then $T^{'}_{\beta\alpha}$ is the sum over all the paths of the product of the labels along the paths from the node β on the lateral face of the tetrahedron to the node α on the base of the tetrahedron. In other words, $T^{'}_{\beta\alpha}$ is the sum over all the paths of the product of the labels along the paths from the node α on the base of the tetrahedron to the node β on the lateral face of the tetrahedron. However, since the labels along the path remain the same and the multiplication is commutative, the direction of the path is immaterial. Therefore, $T^{'}_{\beta\alpha} = T_{\alpha\beta}$; that is, $T^{'} = T^t$.

In summary, the following steps of the first down recurrence algorithm allow us to compute S_α from R_α:

1. reverse the arrows of the first up recurrence diagram,
2. keep the same labels along the edges,
3. place the coefficients R_α on the nodes of a lateral face of the tetrahedron,
4. start the computation at the apex of the tetrahedron and proceed downwards,
5. compute the value at an empty node by multiplying the label along each arrow that enters the node by the value of the node from which the arrow emerges and add the results; compute the value at a non-empty node by applying the same procedure and simply add the value already at that node,
6. collect the coefficients S_α at the base of the tetrahedron.

The second and the third down recurrence diagram allows us to compute T_α from S_α and U_α from T_α by reversing respectively the second and the third up recurrence diagrams. Thus the three down recurrence diagrams taken together represent a change of basis algorithm between two arbitrary L-bases.

This general change of basis algorithm is $O(n^3)$ since each tetrahedron has $O(n^3)$ nodes. We record the following differences between the up recurrence evaluation algorithms described in Section 3 and the change of basis algorithms described here which will be used shortly to derive several special down recurrence evaluation algorithms: (i) The up recurrence evaluation algorithm uses only one tetrahedron. In contrast, the change of basis algorithms use, in general, three tetrahedra. In particular, the Lagrange evaluation algorithm described in Section 4.3, the nested multiplication evaluation algorithm described in Section 4.4, the divided difference algorithm and the forward difference evaluation algorithms described in Section 4.5 use 3, 2, 2 and 2 tetrahedra respectively. In some cases, as in the nested multiplication evaluation algorithm, the divided difference evaluation algorithm and the forward difference evaluation algorithms, these tetrahedra get simplified. (ii) In the up recurrence evaluation algorithms, all the original coefficients are at the base, the computation proceeds upwards and the final value emerges at the apex of the tetrahedron. In the change of basis algorithms and the down recurrence evaluation algorithms to be discussed shortly, the original coefficients are placed on a lateral face of the tetrahedron and the computation proceeds downward. Observe that in this case even if the tetrahedron were to be rotated so that the original coefficients were placed at the base of the tetrahedron, the computation will proceed upwards as well as "sideways" since there will be "side" arrows in the base of the tetrahedron. There are no such "side" arrows in the up recurrence evaluation algorithms; that is, all the arrows in the up recurrence evaluation algorithms point upward.

4.2. **Change of Basis Algorithms.**

We now describe this change of basis algorithms between L-bases in greater detail. Although these change of basis algorithms between L-bases are easy to implement, it is somewhat tedious to describe the labeling in full generality and complete detail. Therefore, we will begin with the description of a change of basis algorithm between two *quadratic* L-bases. We will then describe this change of basis algorithm between L-bases of arbitrary degree.

To describe the change of basis algorithm, we construct three tetrahedra. We first explain the labeling scheme for these tetrahedra. For each tetrahedron, $\frac{(3-i)(4-i)}{2}$ nodes are placed at the i-th level of the tetrahedron for $i = 0, 1, 2$ and the nodes along one of the lateral faces are indexed by α for $|\alpha| = 2$. An arrow is placed

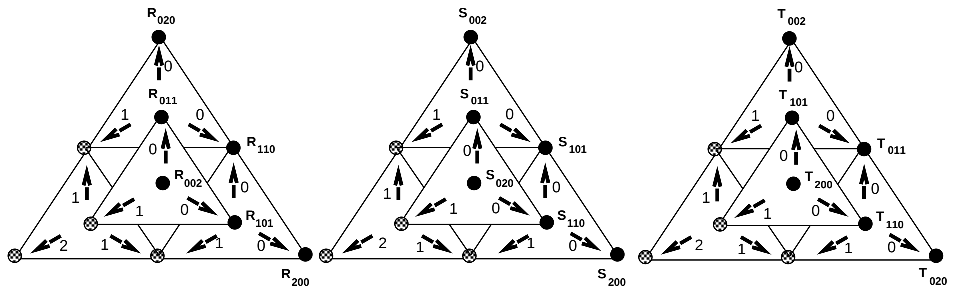

FIGURE 16. Change of basis from Bernstein-Bézier L-basis to Lagrange L-basis

pointing downward from a node α at i-th level to the three nodes $\alpha + e_1 - e_3$, $\alpha + e_2 - e_3$ and $\alpha - e_3$ at $(i-1)$-st level directly below it. This labeling scheme for the nodes is shown in Figure 15. Values, referred to as *labels*, are placed along the arrows. The labels are indexed as $g_{k,\alpha}$ for $k = 1, 2, 3$ and $|\alpha| = 0, 1, 2$ for an arrow from a node $(\alpha_1, \alpha_2, 2 - |\alpha|)$ at the $|\alpha|$-th level to the three nodes below it. This labeling scheme for the labels and the arrows is also shown in Figure 15.

For the first tetrahedron the known coefficients R_α with $|\alpha| = 2$ are placed at the nodes along one of the lateral faces of the tetrahedron as depicted in the first diagram of Figure 16. The labels $g_{k,\alpha}$ are computed as follows: For $|\alpha| = 0, 1$, let $i = 2 - |\alpha|$; then

$$L_{3i} = g_{1,\alpha} L_{1,\alpha_1+1} + g_{2,\alpha} L_{2,\alpha_2+1} + g_{3,\alpha} M_{3,\alpha_3+1}.$$

Thus finding $g_{k,\alpha}$ amounts to solving a 3×3 system of linear equations.

The computation is now carried out as follows. At the start all the nodes at all levels of the tetrahedron are empty or zero other than the nodes α with $|\alpha| = 2$, where the coefficients R_α are placed. The empty or zero nodes are shown as hatched circles in Figures 15 and 16. The computation starts at the apex of the tetrahedron and proceeds downwards. A value at an empty node is computed by multiplying the label along each arrow that enters the node by the value of the node from which the arrow emerges and adding the results. A value at a non-empty node is computed by applying the same procedure and simply adding the value already at that node. After the computation is complete, the new coefficients $S_{\alpha+(2-|\alpha|)e_3}$ emerge at the nodes α on the base triangle. These new coefficients now express the polynomial L with respect to the L-basis defined by the knot-net $\{\{L_{1j}\}, \{L_{2j}\}, \{M_{3j}\}, j = 1, 2\}$.

We now repeat the above procedure with a second tetrahedron, where the coefficients S_α are placed at the nodes α with $|\alpha| = 2$ as shown in the middle diagram of Figure 16. The labels on the tetrahedron are permuted from (i, j, k) to (i, k, j) because now we wish to retain the polynomial M_{3j} and replace the polynomials L_{2j} by M_{2j}. The labels $g_{k,\alpha}$ are now computed as follows: For $|\alpha| = 0, 1$, let $i = 2 - |\alpha|$; then

$$L_{2i} = g_{1,\alpha} L_{1,\alpha_1+1} + g_{2,\alpha} M_{2,\alpha_2+1} + g_{3,\alpha} M_{3,\alpha_3+1}.$$

These labels are also shown in the middle diagram of Figure 16. After the computation is complete, the new coefficients T_α emerge at the nodes on the base triangle. These coefficients now express the polynomial L with respect to the L-basis defined by the knot-net $\{\{L_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, 2\}$.

Finally we repeat the above procedure with a third tetrahedron, where the coefficients T_α are placed at the nodes α with $|\alpha| = 2$ as shown in the rightmost diagram of Figure 16. The labels on this tetrahedron are permuted from (i, j, k) to (j, k, i) because now we wish to retain the polynomials M_{2j} and M_{3j} and replace the polynomials L_{1j} by M_{1j}. The labels $g_{k,\alpha}$ are computed as follows: For $|\alpha| = 0, 1$, let $i = 2 - \alpha$; then

$$L_{1i} = g_{1,\alpha} M_{1,\alpha_1+1} + g_{2,\alpha} M_{2,\alpha_2+1} + g_{3,\alpha} M_{3,\alpha_3+1}.$$

After the computation is complete, the new coefficients U_α emerge at the nodes on the base triangle. These new coefficients are the desired coefficients that represent the polynomial in the new L-basis.

A general change of basis algorithm from any L-basis to any other L-basis is obtained by following essentially the same procedure. The general change of basis algorithm is constructed in the following manner:

1. Build three tetrahedra. For each tetrahedron, $\frac{(n+1-i)(n+2-i)}{2}$ nodes are placed at the i-th level of the tetrahedron for $i = 0, \cdots, n$. The labels $g_{k,\alpha}$ along the edges of the first tetrahedron are computed for $|\alpha| = 0, \cdots, n-1$, from

$$L_{3i} = g_{1,\alpha} L_{1,\alpha_1+1} + g_{2,\alpha} L_{2,\alpha_2+1} + g_{3,\alpha} M_{3,\alpha_3+1}, \quad i = n - |\alpha|.$$

   The labels for the second and the third tetrahedron are computed in a similar fashion. We assume that the intermediate knot-nets $\{\{L_{1j}\}, \{L_{2j}\}, \{M_{3j}\}, j = 1, \cdots, n\}$ are linearly independent.
2. Point the arrows on the tetrahedron downwards and place the original coefficients R_α along the lateral face of the tetrahedron. Carry out the computation and collect the new coefficients S_α along the base of the tetrahedron.
3. Repeat steps 1 and 2 two more times with the second and third tetrahedra using the output of the previous step as the input into the next step. After 3 steps, the coefficients at the base of the tetrahedron are the desired coefficients U_α.

4.3. **Lagrange Evaluation Algorithm.** A Lagrange evaluation algorithm is obtained by carrying out the change of basis algorithm from a given L-basis to the Lagrange L-basis defined by the geometric mesh or the principal lattice configuration described in Section 2.3. The coefficients obtained are the values of the given polynomial at $O(n^2)$ points $\mathbf{v}_\alpha$ up to constant multiples. These coefficients must be multiplied by $l_\alpha^n(\mathbf{v}_\alpha)$ to obtain the value of the given polynomial at these points. Since the computational cost of the change of basis algorithm is $O(n^3)$, the computational cost of evaluating the polynomial at $O(n^2)$ points is $O(n^3)$, that is, an amortized cost of $O(n)$ per point. This algorithm can be used repeatedly to evaluate any bivariate L-basis on a regular rectangular lattice with an amortized cost of $O(n)$ computations per point.

We illustrate this Lagrange evaluation algorithm by presenting an example. Suppose we are given the coefficients R_α of a quadratic polynomial L with respect to the Bernstein-Bézier L-basis $\{l_\alpha^n\}$ defined by a knot-net $\mathcal{L} = \{\{L_{1j}\}, \{L_{2j}\}, \{L_{3j}\}, j = 1, 2\}$, where

$$\begin{aligned} L_{11} &= x; & L_{12} &= x; \\ L_{21} &= y; & L_{22} &= y; \\ L_{31} &= 1 - x - y; & L_{32} &= 1 - x - y. \end{aligned}$$

The corresponding Bernstein-Bézier L-basis is then given by $l^2_{200} = x^2$, $l^2_{020} = y^2$, $l^2_{002} = (1-x-y)^2$, $l^2_{110} = xy$, $l^2_{101} = x(1-x-y)$, and $l^2_{011} = y(1-x-y)$. If the coefficients R'_α of the quadratic polynomial L are given with respect to the quadratic Bernstein-Bézier basis x^2, y^2, $(1-x-y)^2$, $2xy$, $2x(1-x-y)$, and $2y(1-x-y)$, we first compute the coefficients R_α with respect to the Bernstein-Bézier L-basis by $R_\alpha = \frac{n!}{\alpha!}R'_\alpha$, where $n = 2$ in this case.

We would like to compute the coefficients U_α of this quadratic polynomial L with respect to the Lagrange L-basis $\{m^n_\alpha\}$ defined by the knot-net $\mathcal{M} = \{\{M_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, 2\}$, where

$$\begin{aligned} M_{11} &= x; & M_{12} &= x - \tfrac{1}{2}; \\ M_{21} &= y; & M_{22} &= y - \tfrac{1}{2}; \\ M_{31} &= 1-x-y; & M_{32} &= \tfrac{1}{2} - x - y. \end{aligned}$$

The corresponding Lagrange L-basis is then given by $m^2_{200} = x(x-\frac{1}{2})$, $m^2_{020} = y(y-\frac{1}{2})$, $m^2_{002} = (1-x-y)(\frac{1}{2}-x-y)$, $m^2_{110} = xy$, $m^2_{101} = x(1-x-y)$, and $m^2_{011} = y(1-x-y)$.

For this example, the labels for the first tetrahedron are: $g_{3,001} = 2$, and $g_{1,001} = g_{2,001} = g_{3,010} = g_{3,100} = g_{3,000} = 1$. The rest of the labels are zero. These labels are shown in the first diagram of Figure 16. By carrying out the computation downwards in the first tetrahedron, one obtains the following coefficients: $S_{200} = R_{200}$, $S_{110} = R_{110}$, $S_{020} = R_{020}$, $S_{101} = R_{002} + R_{101}$, $S_{011} = R_{002} + R_{011}$, $S_{002} = 2R_{002}$. These new coefficients now express the polynomial L with respect to the L-basis defined by the knot-net $\{\{L_{1j}\}, \{L_{2j}\}, \{M_{3j}\}, j = 1, 2\}$.

For our example, the labels in the second tetrahedron turn out to be the same as in the first tetrahedron and are shown in the middle diagram of Figure 16. After carrying out the computation in the second tetrahedron, the coefficients are as follows: $T_{200} = S_{200}$, $T_{110} = S_{020} + S_{110}$, $T_{020} = 2S_{020}$, $T_{101} = S_{101}$, $T_{011} = S_{020} + S_{011}$, $T_{002} = S_{002}$. These coefficients now express the polynomial L with respect to the L-basis defined by the knot-net $\{\{L_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, 2\}$.

Again in our example, the labels in the third tetrahedron are the same as in the first tetrahedron and are shown in the rightmost diagram of Figure 16. After the computation in the third tetrahedron, the new coefficients are as follows: $U_{200} = 2T_{200}$, $U_{110} = T_{200}+T_{110}$, $U_{020} = T_{020}$, $U_{101} = T_{200}+T_{101}$, $U_{011} = T_{011}$, $U_{002} = T_{002}$. These coefficients express the polynomial L with respect to the L-basis defined by the knot-net $\mathcal{M} = \{\{M_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, 2\}$. The change of basis algorithm is now complete. In terms of the original coefficients R_α, the final coefficients U_α, are: $U_{200} = 2R_{200}$, $U_{110} = R_{200} + R_{020} + R_{110}$, $U_{020} = 2R_{020}$, $U_{101} = R_{200} + R_{002} + R_{101}$, $U_{011} = R_{020} + R_{002} + R_{011}$, $U_{002} = 2R_{002}$.

Since the Lagrange basis is given by $\{\frac{m^n_\alpha}{m^n_\alpha(\mathbf{v}_\alpha)}\}$, while the Lagrange L-basis is m^n_α, these bases differ by constant multiples. The coefficients U'_α with respect to the Lagrange basis are computed by $U'_\alpha = U_\alpha m^n_\alpha(\mathbf{v}_\alpha)$. In the current case, since $\mathbf{v}_{200} = (1, 0)$, $\mathbf{v}_{020} = (0, 1)$, $\mathbf{v}_{002} = (0, 0)$, $\mathbf{v}_{110} = (\frac{1}{2}, \frac{1}{2})$, $\mathbf{v}_{101} = (\frac{1}{2}, 0)$, $\mathbf{v}_{011} = (0, \frac{1}{2})$, we find $m^2_{200}(\mathbf{v}_{200}) = \frac{1}{2}$, $m^2_{020}(\mathbf{v}_{020}) = \frac{1}{2}$, $m^2_{002}(\mathbf{v}_{002}) = \frac{1}{2}$, $m^2_{110}(\mathbf{v}_{110}) = \frac{1}{4}$, $m^2_{101}(\mathbf{v}_{101}) = \frac{1}{4}$, $m^2_{011}(\mathbf{v}_{011}) = \frac{1}{4}$. The coefficients U'_α are the values of the quadratic polynomial L at the points $\mathbf{v}_\alpha$; that is, $L(\mathbf{v}_\alpha) = U'_\alpha$. This completes the evaluation algorithm.

The general Lagrange evaluation algorithm is obtained similarly by first converting the coefficients in a given basis to an L-basis by multiplying if necessary by appropriate constants, running the change of basis algorithm from the given L-basis to the Lagrange L-basis, and finally multiplying the derived coefficients by appropriate constants to obtain the value of the polynomial at the points defined by the Lagrange L-basis.

4.4. **Nested Multiplication Evaluation Algorithm.** We now describe a nested multiplication evaluation algorithm for an L-basis that is obtained by specializing the change of basis algorithms between L-bases described in Section 4.2.

To evaluate a polynomial expressed in terms of an L-basis at a point $\mathbf{u}$, let M and N be two polynomials that vanish at $\mathbf{u}$, that is, $M(\mathbf{u}) = N(\mathbf{u}) = 0$. We now peform a change of basis algorithm from the L-basis defined by the given knot-net $\mathcal{L}$ to the L-basis defined by the knot-net $\{L_{1j}, L_{2j}, N; j = 1, \cdots n\}$ and then from this basis to the L-basis defined by the knot-net $\{L_{1j}, M, N; j = 1, \cdots n\}$. Since all the basis functions in the L-basis defined by this last knot-net have a factor of M or N except the basis function $l^n_{n00} = L_{11} \cdots L_{1n}$, the value of the given polynomial at $\mathbf{u}$ can now be computed simply by multiplying the coefficient of l^n_{n00} with $l^n_{n00}(\mathbf{u})$.

Notice that in this algorithm, one performs computations only for two tetrahedra in contrast to computations for three tetrahedra in the general change of basis algorithm. Also, for the first tetrahedron, the computation needs to be carried out only along one of the faces of the tetrahedron, as shown in the left diagram of Figure 17, because only the values along one of the edges at the base of the tetrahedron are needed as input for the next tetrahedron. Therefore, the computational complexity for the first tetrahedron is only $O(n^2)$. For the second tetrahedron, the computation needs to be carried out only along one edge of the tetrahedron, as shown in the right diagram of Figure 17, because only the value at one corner of the base of the tetrahedron is needed to evaluate the given polynomial. The computational complexity of this step is therefore only $O(n)$, giving an overall computational complexity of $O(n^2)$.

It is shown in [26] that for polynomials written in terms of the multinomial basis, this algorithm specializes to a bivariate generalization of Horner's evaluation algorithm for univariate polynomials expressed in the monomial (Taylor) basis, and agrees with the algorithm for evaluation of multivariate polynomials proposed by Schumaker and Volk [42].

Observe that this nested multiplication evaluation algorithm can be generalized somewhat in order to further simplify the computation of the labels along the edges. In this evaluation algorithm if we are given only one value of $\mathbf{u}$, then we have a good deal of flexibility in choosing M and N. In fact, even more generally, we can choose

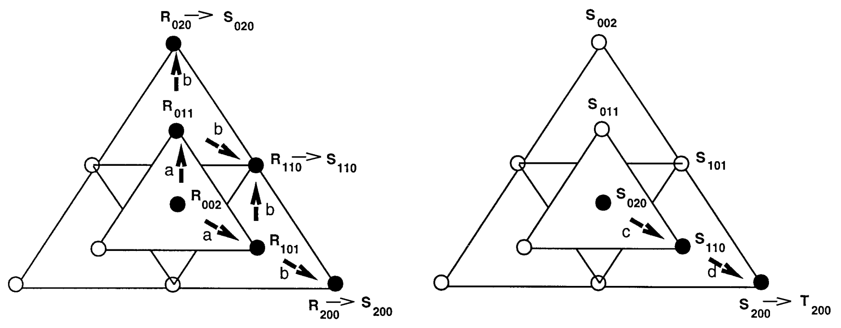

FIGURE 17. Nested multiplication evaluation algorithm for L-bases

any set of $2n$ lines $\{M_i, N_i, i = 1, \cdots, n\}$ passing through $\mathbf{u}$ and the same argument goes through as long as the intermediate knot-nets are linearly independent.

In the case of a Bernstein-Bézier or multinomial L-basis, $L_{1j} = L_1$, $L_{2j} = L_2$ and $L_{3j} = L_3$. Given $\mathbf{v}_0 = (x_0, y_0)$, one possible choice is $M = x - x_0$ and $N = y - y_0$. But another possible choice, $M = L_3(\mathbf{v}_0)L_1 - L_1(\mathbf{v}_0)L_3$ and $N = L_2(\mathbf{v}_0)L_1 - L_1(\mathbf{v}_0)L_2$, is more symmetric and has the advantage that the labels $g_{k,\alpha}$ are much simpler, as the following identities reveal:

$$L_3 = 0 \times L_2 + \frac{L_3(\mathbf{v}_0)}{L_1(\mathbf{v}_0)}L_1 - \frac{1}{L_1(\mathbf{v}_0)}M,$$

$$L_2 = 0 \times M + \frac{L_2(\mathbf{v}_0)}{L_1(\mathbf{v}_0)}L_1 - \frac{1}{L_1(\mathbf{v}_0)}N.$$

We close this section by highlighting with an example the difference between the nested multiplication evaluation algorithm described in Section 3.2 and the nested multiplication algorithm introduced here. Consider a polynomial $L(\mathbf{u})$ represented in terms of the Lagrange L-basis defined by the knot-net

$$\begin{array}{ll} L_{11} = x; & L_{12} = x - \frac{1}{2}; \\ L_{21} = y; & L_{22} = y - \frac{1}{2}; \\ L_{31} = 1 - x - y; & L_{32} = \frac{1}{2} - x - y. \end{array}$$

Suppose $L(\mathbf{u}) = \sum_{|\alpha|=n} R_\alpha l_\alpha^n$ is to be evaluated at a point (x_0, y_0). Then a typical nested multiplication evaluation algorithm described in Section 3 organizes the computation as follows:

$$\begin{aligned} L(\mathbf{u}) \quad = \quad & R_{200}(x_0 - \frac{1}{2})x_0 + (R_{110}x_0 + R_{020}(y_0 - \frac{1}{2}))y_0 \\ & +(R_{101}x_0 + R_{011}y_0 + R_{002}(\frac{1}{2} - x_0 - y_0))(1 - x_0 - y_0). \end{aligned}$$

In contrast, the nested multiplication algorithm described in this section organizes the computation very differently. We first need to make choices. Let us define $N = x + y - x_0 - y_0$ and $M = y - y_0$. For the first tetrahedron, one obtains the following required labels by solving the appropriate system of linear equations: $g_{1,000} = g_{2,000} = a$, $g_{1,100} = g_{2,100} = g_{1,010} = g_{2,010} = b$, where $a = \frac{1}{2(x_0+y_0)} - 1$, and $b = \frac{1}{2(x_0+y_0)-1} - 1$, as shown on the left diagram of Figure 17. This tetrahedron yields the computation: $S_{200} = (R_{002}a + R_{101})b + R_{200}$, $S_{110} = (R_{002}a + R_{101})b + (R_{002}a + R_{011})b + R_{110}$, and $S_{020} = (R_{002}a + R_{011})b + R_{020}$. For the second tetrahedron, the necessary labels are: $g_{1,000} = c$, and $g_{1,100} = d$, where $c = \frac{2y_0-1}{2x_0}$ and $d = \frac{2y_0}{2x_0-1}$. These labels are shown on the right diagram of Figure 17. This tetrahedron yields the computation: $T_{200} = (S_{020}c + S_{110})d + S_{200}$. The value of the original polynomial is then finally obtained by multiplying the value T_{200} by $l_{200}^2(x_0, y_0)$, which in this case is $x_0(x_0 - \frac{1}{2})$.

4.5. **Divided and Forward Difference Algorithms.**

In this section, we establish that the divided difference algorithm and the forward difference algorithm for evaluating bivariate polynomials can be obtained as a change of basis algorithm from a Newton L-basis to another Newton L-basis.

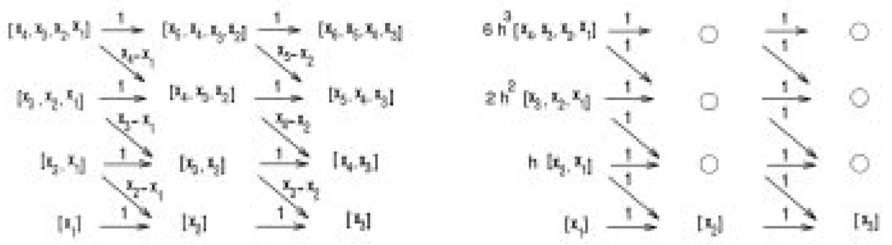

FIGURE 18. Univariate divided and forward difference evaluation algorithms

The divided difference algorithm is an algorithm for evaluating polynomials at several points, where successive computations take advantage of previous computations. The forward difference evaluation algorithm is a divided difference algorithm for evaluating a polynomial at several *equidistant* points.

To explain the divided difference algorithm in the multivariate setting, we first very briefly describe this algorithm in the univariate setting. Suppose a polynomial $L(x)$ of degree n in one variable is to be evaluated at points $x_1, \cdots x_m$, where m is much larger than the degree n of the polynomial. For the sake of illustration, assume that the given polynomial is of degree three. Moreover suppose that the polynomial is represented in the Newton basis 1, $x - x_1$, $(x - x_1)(x - x_2)$ and $(x-x_1)(x-x_2)(x-x_3)$. If that is not the case, then the polynomial can always be evaluated at $n+1$ distinct points and then the coefficients of the polynomial $L(x)$ with respect to the Newton basis can be computed using well-known techniques. The coefficients of the polynomial with respect to the Newton basis are the divided differences as shown below:

$$\begin{aligned} L(x) &= L[x_1] + L[x_2, x_1](x - x_1) + L[x_3, x_2, x_1](x - x_1)(x - x_2) \\ &\quad + L[x_4, x_3, x_2, x_1](x - x_1)(x - x_2)(x - x_3). \end{aligned}$$

Given these coefficients, one can compute the new coefficients of the same polynomial with respect to the new Newton basis 1, $x - x_2$, $(x - x_2)(x - x_3)$ and $(x-x_2)(x-x_3)(x-x_4)$ by applying the change of basis algorithm from one Newton basis to another Newton basis as shown in the left diagram of Figure 18. Observe that the coefficient of the polynomial with respect to the basis function 1 in the new Newton basis (designated as $[x_2]$ in Figure 18) is the value of the polynomial at x_2. We now "roll" this algorithm along to compute the values of the given polynomial at successive points x_3, x_4 and so on. This algorithm has computational complexity $O(n)$.

The forward difference algorithm is akin to the divided difference algorithm but takes advantage of the fact that the points at which the polynomial is to be evaluated are equidistant. In this case, it is possible to get rid of all multiplications so that each successive evaluation is based purely on addition. This goal is achieved by premultiplying the divided differences by appropriate constants, which depend upon the fixed distance between successive points, and then reducing the rest of the computation to pure addition as shown on the right diagram of Figure 18, where h refers to the distance between successive equidistant points. Both the divided and forward difference algorithms are discussed in standard numerical analysis textbooks [10]. We refer the readers to [32, 21] for a generalization of divided differences to the multivariate setting.

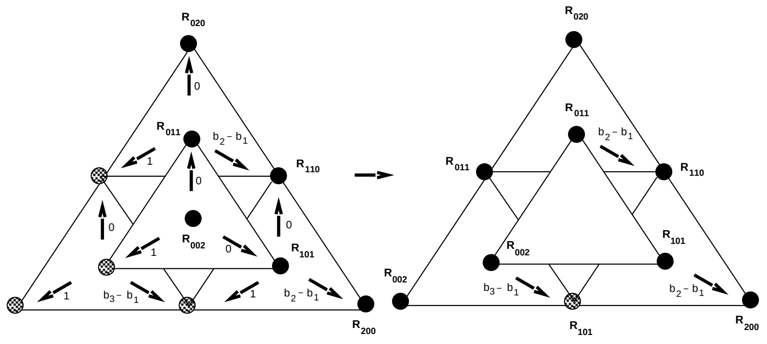

FIGURE 19. One step of bivariate divided difference evaluation algorithm

We now describe how a bivariate version of the divided difference and forward difference algorithms can be obtained to evaluate a bivariate polynomial along a grid $\{(a_j, b_k)\}$. We proceed by specializing the change of basis algorithms between L-bases described in Section 4.2. As in the univariate case, there is a startup step and a marching step. Given any bivariate polynomial of degree n described in terms of an L-basis, we first apply the standard change of basis algorithm between L-bases to compute the representation of the polynomial in the Newton L-basis defined by the knot-net $\mathcal{L} = \{\{L_{1j}\}, \{L_{2j}\}, \{L_{3j}\}, j = 1, \cdots, n\}$, where $L_{1j} = 1$, $L_{2j} = x - a_j$ and $L_{3j} = y - b_j$. Let the coefficients of the given polynomial with respect to this basis be R_α. We can now march in either the x or y-directions. To march in the x-direction, we perform a change of basis from the current Newton basis to the next Newton basis in the x-direction, which is defined by the knot-net $\mathcal{N} = \{\{N_{1j}\}, \{N_{2j}\}, \{N_{3j}\}, j = 1, \cdots, n\}$, where $N_{1j} = 1$, $N_{2j} = x - a_{j+1}$ and $N_{3j} = y - b_j$. Similarly to march in the y-direction, we perform a change of basis from the current Newton basis to the next Newton basis in the y-direction, which is defined by the knot-net $\mathcal{M} = \{\{M_{1j}\}, \{M_{2j}\}, \{M_{3j}\}, j = 1, \cdots, n\}$, where $M_{1j} = 1$, $M_{2j} = x - a_j$ and $M_{3j} = y - b_{j+1}$. The left diagram of Figure 19 illustrates the marching in the y-direction for the case $n = 2$. Since many arrows are 0 or 1, this left diagram simplifies to the right diagram of Figure 19. In general for degree n, there will be $i + 1$ arrows with label $b_{n+1-i} - b_1$, $i = 0, \cdots, n - 1$. Therefore this step of the algorithm requires $O(n^2)$ computations. Similar simplifications hold for marching in the x-direction, which is illustrated for the case $n = 2$ in Figure 20. By marching in both the x and y-directions, we can compute the Newton coefficients of the original polynomial along an arbitrary grid $\{(a_j, b_k)\}$. The computational complexity of this bivariate divided difference algorithm is $O(n^2)$ computations per point.

The bivariate forward difference algorithm takes advantages of equidistant points by premultiplying the Newton coefficients by appropriate constants, just as in the univariate case, thus reducing all the computations to pure additions. To be more precise, the coefficients R_α in the right diagram of Figure 19 are multiplied by $\alpha_3! k^{\alpha_3}$ and the coefficients S_α in the right diagram of Figure 20 are multiplied by $\alpha_2! h^{\alpha_2}$, where h is the distance between the grid points in the x-direction and k is the distance between the grid points in the y-direction. After these premultiplications, the forward difference algorithm is obtained simply by changing all the labels on the arrows in these diagrams to 1. Thus every successive evaluation

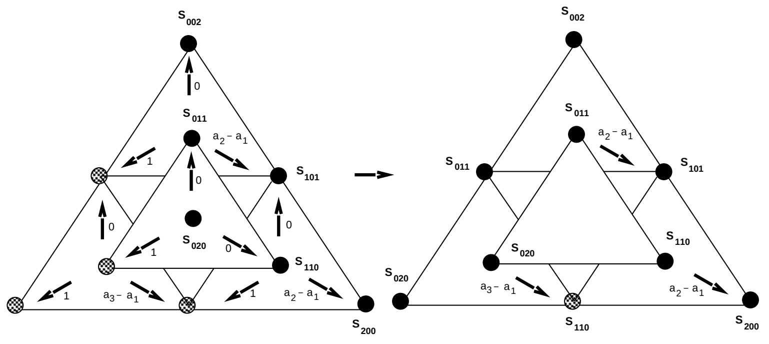

FIGURE 20. Another step of bivariate divided difference evaluation algorithm

can be achieved simply by addition, without any multiplication, thus reducing the cost to $O(n^2)$ additions but only $O(1)$ multiplications per point. Also observe that both the divided difference algorithm and the forward difference algorithm require computation within only one tetrahedron rather than three.

5. Other algorithms

In this section, we briefly review a few other algorithms that are closely related to the evaluation algorithms discussed so far.

5.1. **Hybrid Algorithms.**

In addition to the algorithms for evaluating multivariate polynomials discussed so far, we are aware of some different algorithms for evaluating multivariate polynomials, most of which are spurred by considerations of stability in numerical computations [25, 46, 38]. First, Volk [45] has proposed a hybrid univariate nested multiplication and divided difference algorithm. Volk [46] has also proposed a slightly different evaluation algorithm based on the forward difference algorithm for evaluation of polynomials at several points. In order to improve the stability of the forward difference evaluation algorithm, Volk reduces the level of indirection in the computation by solving an upper triangular system [46]. A second method, discussed by Peters [38], is again motivated by concerns of stability and efficiency. This algorithm extracts univariate polynomials along isoparameter lines and then performs the evaluation algorithms for univariate polynomials using a forward difference or a nested multiplication algorithm. We have not addressed the very important considerations of stability in numerical computations, for which we refer the reader to [18, 16, 17], because rather than stability the focus of this work has been to provide a conceptual framework for the unification of a large variety of evaluation algorithms for multivariate polynomials.

5.2. **Derivative and Other Algorithms.**

Although we have focused only on algorithms for evaluating multivariate polynomials, these algorithms are closely related to and can easily be extended to procedures to compute derivatives of multivariate polynomials. This technique of unifying evaluation algorithms and derivative algorithms has been discussed for univariate polynomials by Goldman and Barry [23]. Here we briefly indicate how this extension from evaluation algorithms to derivative algorithms can be derived. The key observation is that a multinomial (Taylor) basis is defined by a point and two vectors and, therefore, change of basis algorithms to a multinomial basis defined by a point $\mathbf{p}$ and two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ amounts to computing the directional derivatives of the polynomial at the point $\mathbf{p}$ in the directions $\mathbf{v}_1$ and $\mathbf{v}_2$. Therefore to compute the directional derivatives at a point p along the vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of a polynomial $L(\mathbf{u})$ given with respect to an L-basis, one need only perform the change of basis algorithm from the given L-basis to the uniform L-basis defined by the following three lines: the line through p along the direction $\mathbf{v}_1$, line through p along the direction $\mathbf{v}_2$, and the line at infinity.

Using principles of homogenization, blossoming, and duality [3, 2, 9, 29, 30, 4] univariate evaluation and differentiation algorithms have been unified with several other very well-known algorithms, formulas, and identities, including the Oslo algorithm, Boehm's knot-insertion and derivative algorithms, Marsden's identity, the binomial theorem, Ramshaw's blossoming algorithm, a two-term differentiation algorithm, and a two-term degree elevation formula. Although in this work we have focused solely on evaluation algorithms, it follows in conjunction with our earlier work [26, 27] that the principles of blossoming, duality, and homogenization can be extended to provide a similar unification in the multivariate setting between L-bases and B-bases.

6. Conclusions and future work

We have presented a unified framework for evaluation algorithms for multivariate polynomials expressed in a wide variety of polynomial bases including the Bernstein-Bézier, multinomial, Lagrange, and Newton bases. Although in the past several different evaluation algorithms have been constructed by organizing the nested multiplications in different ways, the interpretation and unification of all these algorithms either as a way of reorganizing the computation in an up recurrence algorithm or as change of basis algorithms is new.

Variations of the up recurrence algorithm include a parallel up recurrence algorithm with computational complexity $O(n^3)$, a nested multiplication algorithm with computational complexity $O(n^2 \log n)$, a ladder recurrence algorithm with computational complexity $O(n^2)$, and a generalization of the Aitken-Neville algorithm for Lagrange L-bases with computational complexity $O(n^3)$. Specializations of this class of algorithms include the de Casteljau algorithm for Bernstein-Bézier bases, the evaluation algorithms for multinomial and Newton bases proposed by Carnicer and Gasca, and the evaluation algorithms for multinomial bases proposed by de Boor and Ron.

Variations of change of basis algorithms between L-bases yield a divided difference algorithm with computational complexity $O(n^2)$ per point, a forward difference algorithm with $O(1)$ multiplications and $O(n^2)$ additions per point, and a Lagrange evaluation algorithm with amortized computational complexity $O(n)$ per point for the evaluation of polynomials at several points. This class of algorithms also includes a nested multiplication algorithm with computational complexity $O(n^2)$ which along with the nested multiplication evaluation algorithm described in Section 3.2 can be considered as a generalization of Horner's algorithm for the evaluation of univariate polynomials. The evaluation algorithm for multinomial bases proposed by Schumaker and Volk [42] is a special case of the nested multiplication evaluation algorithm presented in Section 4.4.

New algorithms derived and discussed in this work can best be appreciated in the case of Lagrange bases, where unlike multinomial or Newton bases considerable simplifications do *not* occur. For multivariate polynomials expressed in Lagrange L-bases, we have described several evaluation algorithms including a nested multiplication algorithm with computational complexity $O(n^2 \log n)$ generated by removing redundant arrows from the up recurrence algorithm, a generalization of the Aitken-Neville algorithm with computational complexity $O(n^3)$, a ladder recurrence algorithm with computational complexity $O(n^2)$, and a nested multiplication algorithm with computational complexity $O(n^2)$. We have presented specific examples to demonstrate that these algorithms are all distinct both conceptually and in practice.

It has been very satisfying to discuss all the well-known algorithms for evaluating multivariate polynomials in a single unified framework. It would be satisfying to integrate into our formulation evaluation algorithms for multivariate polynomials expressed in terms of other useful bases such as multivariate Hermite bases. A generalization of the Aitken-Neville recurrence for Hermite bases defined over geometric meshes is currently under investigation by Habib, Goldman and Lyche [24].

Acknowledgments

We wish to thank the anonymous referee whose comments helped us to improve the presentation of this work.

References

1. A. G. Aitken, *On interpolation by iteration of proportional parts without the use of differences*, Proceedings of Edinburgh Mathematical Society (1932), 56–76.
2. P. Barry and R. Goldman, *Algorithms for progressive curves: Extending B-spline and blossoming techniques to the monomial, power, and Newton dual bases*, Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces (R. Goldman and T. Lyche, eds.), SIAM, 1993, pp. 11–64. CMP 93:11
3. P. Barry, R. Goldman, and T. DeRose, *B-splines, Pólya curves and duality*, Journal of Approximation Theory **65** (1991), no. 1, 3–21. MR **92f:**41018
4. W. Boehm, *Inserting new knots into B-spline curves*, Computer-Aided Design **12** (1980), 199–201.
5. J. Carnicer and M. Gasca, *Evaluation of multivariate polynomials and their derivatives*, Mathematics of Computation **54** (1990), no. 189, 231–243. MR **90h:**12001
6. ———, *On the evaluation of multivariate Lagrange formulae*, Proceedings of the Conference Mehrdimensionale Knostruktive Funktiontheorie, Oberwalch, Birkhauser Verlag, 1990, pp. 65–72. MR **91c:**65014
7. A. S. Cavaretta and C. A. Micchelli, *Pyramid patches provide potential polynomial paradigms*, Mathematical Methods in CAGD and Image Processing (T. Lyche and L. L. Schumaker, eds.), Academic Press, 1992, pp. 1–40. MR **93h:**65023
8. C. K. Chung and T. H. Yao, *On lattices admitting unique Lagrange interpolations*, Siam Journal on Numerical Analysis **14** (1977), 735–743. MR **56:**3502
9. E. Cohen, T. Lyche, and R. F. Riesenfeld, *Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics*, Computer Graphics and Image Processing **14** (1980), 87–111.
10. S. D. Conte and C. de Boor, *Elementary numerical analysis*, McGraw-Hill, New York, 1980, Third Edition.
11. C. de Boor, *A practical guide to splines*, Springer Verlag, New York, 1978, Applied Mathematical Sciences, Volume 27. MR **80a:**65027

12. C. de Boor and Amos Ron, *Computational aspects of polynomial interpolation in several variables*, Mathematics of Computation (1992), 705–727. MR **92i:**65022
13. P. de Casteljau, *Formes á pôles*, Hermes, Paris, 1985.
14. G. Farin, *Triangular Bernstein-Bézier patches*, Computer Aided Geometric Design **3** (1986), no. 2, 83–128. MR **87k:**65014
15. ______, *Curves and surfaces for computer aided geometric design: A practical guide*, Academic Press Inc., New York, 1988. MR **90c:**65014
16. R. Farouki, *On the stability of transformations between power and Bernstein form*, Computer Aided Geometric Design (1991), no. 1, 29–36. MR **91m:**65042
17. R. Farouki and V. Rajan, *On the numerical condition of polynomials in Bernstein form*, Computer Aided Geometric Design **4** (1987), 191–216. MR **89a:**65028
18. ______, *Algorithms for polynomials in Bernstein form*, Computer Aided Geometric Design **5** (1988), no. 1, 1–26. MR **89c:**65033
19. M. Gasca, *Multivariate polynomial interpolation*, Computation of Curves and Surfaces (W. Dahmen, M. Gasca, and C. A. Micchelli, eds.), Kluwer Academic Publishers, 1990, pp. 215–236. MR **91k:**65028
20. M. Gasca and E. Lebrón, *On Aitken-Neville formulae for multivariate interpolation*, Numerical Approximation of Partial Differential Equations, Elsevier Science Publication, North Holland, 1987, pp. 133–140. MR **89b:**41005
21. M. Gasca and A. López-Carmona, *A general recurrence interpolation formula and its applications to multivariate interpolation*, Journal of Approximation Theory (1982), 361–374. MR **83f:**41003
22. M. Gasca and J. I. Maeztu, *On Lagrange and Hermite interpolation in R^k*, Numer. Math. **39** (1989), 1–14. MR **83g:**65012
23. R. N. Goldman and P. J. Barry, *Wonderful triangle: A simple, unified, algorithmic approach to change of basis procedures in computer aided geometric design*, Mathematical Methods in CAGD II (T. Lyche and L. Schumaker, eds.), Academic Press, 1992, pp. 297–320. MR **93e:**65027
24. A. Habib, R. Goldman, and T. Lyche, *A recursive algorithm for Hermite interpolation over a triangular grid*, Journal of Computational and Applied Mathematics **73** (1996), 95–118.
25. S. L. Lien, M. Shantz, and V. Pratt, *Adaptive forward differencing for rendering curves and surfaces*, Computer Graphics **21** (1987), 111–118.
26. S. K. Lodha and R. Goldman, *Change of basis algorithms for surfaces in CAGD*, Computer Aided Geometric Design **12** (1995), 801–824. CMP 96:04
27. ______, *Lattices and algorithms for bivariate Bernstein, Lagrange, Newton and other related polynomial bases based on duality between L-bases and B-bases*, University of California, Santa Cruz, CA, UCSC-CRL-95-14, Submitted for publication, 1995.
28. S. K. Lodha, R. Goldman, and J. Warren, *A ladder recurrence algorithm for the evaluation of L-patches*, Annals of Numerical Mathematics **3** (1996), 209–220. CMP 96:14
29. T. Lyche and K. Morken, *Making the Oslo algorithm more efficient*, SIAM Journal of Numerical Analysis (1986), 663–675. MR **87m:**65031
30. M. J. Marsden, *An identity for spline functions with applications to variation-diminishing spline approximation*, Journal of Approximation Theory **3** (1970), 7–49. MR **40:**7682
31. A. Le Méhauté, *Approximation of derivatives in R^n. Applications: construction of surfaces in R^2*, Multivariate Approximation (S. P. Singh, J. H. W. Burry, and B. Watson, eds.), Reidel, 1984, pp. 361–378. CMP 17:12
32. G. Mühlbach, *A recurrence formula for generalized divided differences and some applications*, Journal of Approximation Theory **9** (1973), 165–172. MR **50:**6106
33. ______, *Newton and Hermite interpolation mit Cebysev-systemen*, Z. Angew. Math. Mech. **50** (1974), 97–110. MR **50:**8913
34. ______, *The general Neville-Aitken algorithm and some applications*, Numerische Mathematik **31** (1978), 97–110. MR **80a:**65025
35. ______, *On multivariate interpolation by generalized polynomials in subsets of grids*, Computing **40** (1988). MR **90c:**65020
36. E. H. Neville, *Iterative interpolation*, Journal of Indiana Mathematical Society (1934), 87–120.
37. G. Nürnberger and Th. Riessinger, *Lagrange and Hermite interpolation by bivariate splines*, Numerical Functional Analysis and Optimization **13** (1992), no. 1, 75–96. MR **93f:**41048
38. J. Peters, *Evaluation of the multivariate bernstein-bézier form on a regular lattice*, ACM Transactions on Mathematical Software **20** (1994). CMP 96:06

39. L. Ramshaw, *Blossoming: A connect-the-dots approach to splines*, Digital Systems Research Center, Report 19, Palo Alto, California., 1987.
40. ———, *Blossoms are polar forms*, Computer Aided Geometric Design **6** (1989), 323–358. MR **91d:**65026
41. L. L. Schumaker, *Spline functions: Basic theory*, John Wiley, New York, 1981. MR **82j:**41001
42. L. L. Schumaker and W. Volk, *Efficient evaluation of multivariate polynomials*, Computer Aided Geometric Design (1986), 1'49–154.
43. H. P. Seidel, *Symmetric recursive algorithms for surfaces: B-patches and the de Boor algorithm for polynomials over triangles*, Constructive Approximation **7** (1991), 259–279. MR **92c:**41010
44. H. C. Jr. Thacher and W. E. Milne, *Interpolation in several variables*, J. SIAM (1960), 33–42. MR **22:**8645
45. W. Volk, *An efficient raster evaluation method for univariate polynomials*, Computing **40** (1988), 163–173. MR **89f:**65029
46. ———, *Making the difference interpolation method for splines more stable*, J. of Computing and Applied Mathematics **33** (1990), 53–59. MR **92a:**65047
47. J. Warren, *Creating rational multi-sided Bernstein-Bézier surfaces using base points*, ACM Transactions on Graphics **11** (1992), no. 2, 127–139.
48. ———, *An efficient evaluation algorithm for polynomials in the Pólya basis*, Computing **24** (1995), 1–5.

Department of Computer Science, University of California, Santa Cruz, California 95064
E-mail address: `lodha@cse.ucsc.edu`

Department of Computer Science, Rice University, Houston, Texas 77251-1892
E-mail address: `rng@cs.rice.edu`