Journal of Computational Physics **167,** 22–38 (2001)
doi:10.1006/jcph.2000.6653, available online at http://www.idealibrary.com on IDEAL®

Application of Vector Calculus to Numerical Simulation of Continuum Mechanics Problems

Nicholas M. Bessonov* and Dong Joo Song†

*Institute of the Problems of Mechanical Engineering, Academy of Science of Russia, Bolshoj Pr. 61, St. Petersburg 199178, Russia; †School of Mechanical Engineering, Yeungnam University, Gyongsan, 712-160, Korea
E-mail: bessonov@bess.ipme.ru, djsong@yu.ac.kr

Received March 22, 2000; revised October 5, 2000

It is well known that vector–tensor notation is a compact and natural language for the mathematical formulation of continuum mechanics problems. Here we describe the application of vector technique to numerical simulation starting with a mathematical formulation. We provide an efficient numerical scheme and furnish an implementation as a computer program. As a result (in comparison with traditional "component" form), the two last steps are significantly simplified, especially for multidimensional problems with various boundary conditions in irregular geometries where nonorthogonal meshes are applied. Therefore, more attention can be focused on the physical nature of problems. Apart from the cleaner syntax, vector notation conserves the structure of traditional numerical algorithms and solves multidimensional problems with minimum additional programming effort. Complex practical applications of this technique are described as well.

Key Words: numerical simulation; continuum mechanics; vector–tensor calculus; finite difference; finite volume; finite element; object-oriented programming.

1. INTRODUCTION

We describe here a concise illustration of application of vector techniques to numerical simulation of several continuum mechanics problems from mathematical formulation (Section 1) through numerical scheme (Sections 2–4). We also describe a computer program based on vector–tensor types of variables, which can be developed with the help of object-oriented languages (Section 5). The uniform ("vector") point of view on all three steps allows one to simplify the process of simulation significantly, especially for multidimensional problems in irregular geometries.

We would like to illustrate with the help of a similar situation. Let us deal with problems from the area of algebra of complex numbers. If one uses a programming language without

0021-9991/01 $35.00

complex type variables (BASIC, for example), one has to prepare all formulas for real and imaginary components explicitly (i.e., in "component" form). On the other hand, when one uses programming language with an intrinsic type of complex variable (such as Fortran77), final formulas can be conserved in natural and compact "complex" form. Obviously both cases are equivalent from a mathematical point of view, but in the latter case, the user can significantly simplify final formulas and the program.

Vector–tensor notation is a natural language for continuum mechanics problems. Obviously when formulas for computer simulation are developed, it is important to prepare an "interface" of final formulas in a form that corresponds best to the programming language that will be used in the future. Unfortunately, Fortran77 (the basic language for professional numerical simulation) does not have the intrinsic type of vector (tensor) variables; therefore, a discrete analog of vector analysis must be prepared in "component" form for any component explicitly. In general, the case of multidimensional mechanics problems with vector or tensor variables and irregular geometries where nonorthogonal meshes render the computer programming has become increasingly complex.

Object-oriented languages such as C++ or Fortran90 (dated from 1980 and 1990 respectively) allow the development of new data types or new classes (for vector-tensor variables in our case), which encapsulate all features of these data types. The variables of this type can be as easily applied as variables of intrinsic types (for example, complex types in Fortran77). In other words, one does not have to develop subroutines for computing volumes and areas and so on, but one must develop and work with new data types (new classes) for vector (tensor) objects with overloaded operations such as dot, cross, and diadic (see sample in Section 5.2). Obviously, once a coordinate System is chosen, one must always be concerned about components, at least in non-Cartesian coordinate systems. New classes allow encapsulation of the features. The implementation of this technique allows a significant reduction in the amount of work.

The general system of continuum mechanics equations (conservation laws of mass, momentum, and energy) can be written as

$$\begin{aligned} \frac{d\rho}{dt} &= -\rho \nabla \cdot \mathbf{v} \\ \frac{d}{dt}\int_V \rho \mathbf{v}\, dV &= \oint_S (d\mathbf{s} \cdot \mathbf{T}) \\ \frac{d}{dt}\int_V \rho E\, dV &= \oint_S \mathbf{v} \cdot (d\mathbf{s} \cdot \mathbf{T}), \end{aligned} \tag{1}$$

where t is time, $\mathbf{v}$ is velocity, ρ is density, $\mathbf{T}$ is the stress tensor, E is energy, V is the Lagrangian volume of space enclosed by surface S, differential area vector $d\mathbf{s} = \mathbf{n}\, dS$, and $\mathbf{n}$ is an outward normal vector at S.

The mathematical formulation of the problem (governing equations) includes system (Eq. (1)) and constitutive relationships. For example, they are

$$\mathbf{T} = -p\mathbf{I} + h_1\mathbf{B} + h_2\mathbf{B} \cdot \mathbf{B} \tag{2}$$

for a hyperelastic solid [1] or

$$\mathbf{T} = -p\mathbf{I} + \mu(\nabla\mathbf{v} + \mathbf{v}\nabla) \tag{3}$$

for a Newtonian incompressible fluid. Here p is pressure; μ is viscosity; $\mathbf{I}$ is the unit tensor; $\mathbf{B} = \mathbf{F} \cdot \mathbf{F}^T$ is the left Cauchy–Green tensor; $\mathbf{F} = \mathbf{r}\nabla_0$; $\nabla = \partial/\partial\mathbf{r}$ (for rectangular coordinates (x_1, x_2, x_3) or (x, y, z)); $\nabla = \mathbf{i}_m \partial/\partial x_m$, where $\mathbf{i}_1$, $\mathbf{i}_2$, and $\mathbf{i}_3$ are basis vectors); $\nabla_0 = \partial/\partial\mathbf{r}_0$; $\mathbf{r}$ is the time-dependent Lagrangian position vector; $\mathbf{r}_0 = \mathbf{r}|_{t=0}$; and h_1 and h_2 are material properties that are functions of the invariants $\mathbf{B}$. The summation convention over dummy (repeated) subscripts is inferred over the dimensions of the problem; $\mathbf{a} \cdot \mathbf{b}$, $\mathbf{a} \times \mathbf{b}$, and $\mathbf{ab}$ are dot, cross, and dyadic products for vectors $\mathbf{a}$ and $\mathbf{b}$, respectively, and $\mathbf{a}\nabla \equiv (\nabla\mathbf{a})^T$.

Most of the difficulties in modern computational mechanics are caused during the study of 2D/3D problems with complex shapes and nonlinear rheological models.

Methods of creating a discrete analog of vector analysis on nonorthogonal meshes were developed starting around 1950 and are still evolving. A series of papers creating a discrete analog of vector analysis on nonorthogonal grids was developed by Shashkov, Hyman, Steinberg and others.

Different variants of natural discrete analogs of the divergence, gradient, and curl operators based on coordinate-invariant definitions have been developed [2–6, 10]. Those discrete operators defined by this self-consistent approach satisfy analogs of the major theorems of vector analysis relating the differential operators [6]. By the inclusion of boundary conditions (Dirichlet, Neumann, Robin) into finite-difference methods, the resulting approximation mimics the identities for the differential operators of vector and tensor calculus described as in [5, 8]. Various types of problems have been solved: steady-state equations, diffusion equations [12, 14], gas and fluid dynamics equations [8–11, 13, 15], and solid mechanics equations [10, 16, 17].

Since it is not easy to describe in one paper all possible variants of discrete analogs of vector analysis and computer programs (see [8, 11, 16, 17], and more), we describe here a simpler and more efficient (vector) form of well-known discrete analogs. This allows us to cover finite-volume (FV), finite-difference (FD), and finite-element (FE) approaches from a unified point of view. Obviously the vector notation does not eliminate the "component" notation. These two types of notation complement each other nicely. The final choice depends on concrete features of the problem.

2. VECTOR FORM OF OPERATOR ∇

Let 3D nonorthogonal mesh consist of linear tetrahedral elements (for the sake of simplicity). A typical tetrahedron "abcd" with position-vectors of vertices $\mathbf{r}_a$, $\mathbf{r}_b$, $\mathbf{r}_c$, and $\mathbf{r}_d$ is shown in Fig. 1, where

$$\delta\mathbf{r}_1 = \mathbf{r}_b - \mathbf{r}_a, \quad \delta\mathbf{r}_2 = \mathbf{r}_c - \mathbf{r}_a, \quad \delta\mathbf{r}_3 = \mathbf{r}_d - \mathbf{r}_a \tag{4}$$

are the differences between position-vectors of vertices along edges of a tetrahedron which form a right-handed triad.

The governing equations show that some variables (such as $\mathbf{T}$, p, ρ) at a given point of space depend on gradients of other variables (such as $\mathbf{v}$, $\mathbf{r}$, E) around this point and vice versa. Let us place the first variables within the tetrahedron (element variables) and the second ones on the vertices of the tetrahedron (nodal variables).

As an illustration of the vector form of discretization and object-oriented code, we assume here the types of cells that are popular in FV methods [11, 16] as well as in FE methods [17].

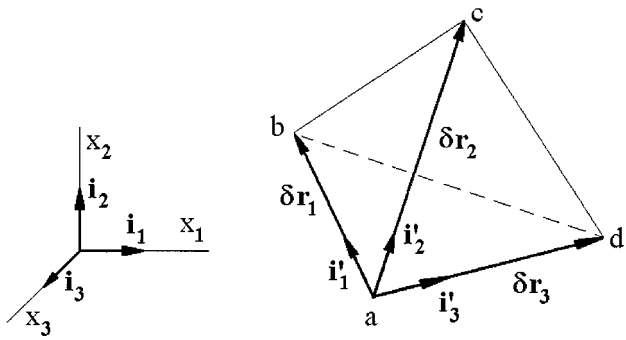

FIG. 1. Tetrahedral element of 3D mesh (• are nodes).

The discretization described here is called a staggered discretization [8], and all descriptions below are of this type of discretization. There are other types of discretization, for example, when all variables are nodal variables, or when some vectors are described by their projection to edges of the cell (electric field) [4–6]. There are many other variants of cells (for example, [2–4, 8, 10, 17]; however, these are beyond the scope of this paper.

2.1. *Vector Approximation of* ∇ *in the Finite Volume (FV) Method*

Popular FV methods, based on approximation of differential relations ∇a, $\nabla \cdot \mathbf{a}$, $\nabla \mathbf{a}$, etc., can be expressed symbolically as in the following [18],

$$\nabla \odot \Phi = \lim_{V \to 0} \frac{1}{V} \oint_S (d\mathbf{s} \odot \Phi), \tag{5}$$

where Φ is a scalar, vector, or tensor, and $\odot$ denotes a distributive operation (dot or cross product, etc.) permissible for Φ. The right-hand side of Eq. (5) can be approximated in the tetrahedron "abcd" as

$$\begin{aligned} \nabla \odot \Phi \approx \frac{1}{V} \Bigg[& \mathbf{s}_a \odot \frac{\Phi_b + \Phi_c + \Phi_d}{3} + \mathbf{s}_b \odot \frac{\Phi_c + \Phi_d + \Phi_a}{3} \\ & + \mathbf{s}_c \odot \frac{\Phi_d + \Phi_a + \Phi_b}{3} + \mathbf{s}_d \odot \frac{\Phi_a + \Phi_b + \Phi_c}{3} \Bigg], \end{aligned} \tag{6}$$

where Φ_a, Φ_b, Φ_c, and Φ_d are nodal variables; $\mathbf{s}_b = (\delta\mathbf{r}_3 \times \delta\mathbf{r}_2)/2$, $\mathbf{s}_c = (\delta\mathbf{r}_1 \times \delta\mathbf{r}_3)/2$, $\mathbf{s}_d = (\delta\mathbf{r}_2 \times \delta\mathbf{r}_1)/2$ and $\mathbf{s}_a = -\mathbf{s}_b - \mathbf{s}_c - \mathbf{s}_d$ are outward-oriented areas of the tetrahedron faces; and $V = \delta\mathbf{r}_1 \cdot (\delta\mathbf{r}_2 \times \delta\mathbf{r}_3)/6$ is volume of tetrahedron. The sign "$\approx$" means the first-order approximation. Hence, from Eq. (6) we could obtain

$$\nabla \odot \Phi \approx \mathbf{r}^m \odot \delta\Phi_m, \tag{7}$$

where

$$\delta\Phi_1 = \Phi_b - \Phi_a, \quad \delta\Phi_2 = \Phi_c - \Phi_a, \quad \delta\Phi_3 = \Phi_d - \Phi_a \tag{8}$$

correspond with Eq. (4), and

$$\mathbf{r}^1 = \frac{\delta\mathbf{r}_2 \times \delta\mathbf{r}_3}{6V}, \quad \mathbf{r}^2 = \frac{\delta\mathbf{r}_3 \times \delta\mathbf{r}_1}{6V}, \quad \mathbf{r}^3 = \frac{\delta\mathbf{r}_1 \times \delta\mathbf{r}_2}{6V}. \tag{9}$$

The sets of vectors $\delta\mathbf{r}_1, \delta\mathbf{r}_2, \delta\mathbf{r}_3$, and $\mathbf{r}^1$, $\mathbf{r}^2$, $\mathbf{r}^3$ are named the reciprocal vectors [18] with properties $\delta\mathbf{r}_k \cdot \mathbf{r}^m = \delta_{km}$ and $\delta\mathbf{r}_m\mathbf{r}^m = \mathbf{r}^m\delta\mathbf{r}_m = \mathbf{I}$, where δ_{km} is the Kroneker delta symbol.

Clearly, for an orthogonal rectangular mesh where $\delta\mathbf{r}_1 = \delta x\mathbf{i}_1$, $\delta\mathbf{r}_2 = \delta y\mathbf{i}_2$, and $\delta\mathbf{r}_3 = \delta z\mathbf{i}_3$, the gradient

$$\mathbf{i}_1\frac{\partial}{\partial x} + \mathbf{i}_2\frac{\partial}{\partial y} + \mathbf{i}_3\frac{\partial}{\partial z} \approx \mathbf{i}_1\frac{\delta}{\delta x} + \mathbf{i}_2\frac{\delta}{\delta y} + \mathbf{i}_3\frac{\delta}{\delta z}$$

follows from Eq. (7).

The construction of the left and the right part of Eq. (7) are similar; therefore, we can introduce the vector finite difference operator $\nabla^* = \mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3$ by the definition

$$\nabla^* \odot \Phi = \mathbf{r}^m \odot \delta\Phi_m \tag{10}$$

and Eq. (7) can be rewritten as

$$\nabla \odot \Phi \approx \nabla^* \odot \Phi. \tag{11}$$

From Eq. (11) some important vector operators can be obtained:

$$\begin{aligned}
\nabla \cdot \Phi &= \nabla^* \cdot \Phi = \mathbf{r}^m \cdot \delta\Phi_m \\
\nabla \times \Phi &= \nabla^* \times \Phi = \mathbf{r}^m \times \delta\Phi_m \\
\nabla\Phi &= \nabla^*\Phi = \mathbf{r}^m\delta\Phi_m \\
\Phi\nabla &= \Phi\nabla^* = \delta\Phi_m\mathbf{r}^m.
\end{aligned}$$

In papers [4–6, 12], the authors used orthogonal projections of vectors to the faces and edges of the cells (similar to directions defined in Fig. 1), which are coordinate invariants. Also, all operators are formulated in "vector" form.

2.2. *Vector Approximation of* ∇ *in the "Finite Difference (FD)" method*

We can obtain the same results in another way based on approximation of the differential operator ∇:

$$\nabla \odot \Phi = \mathbf{i}_n \odot \frac{\partial\Phi}{\partial x_n}. \tag{12}$$

Let variable Φ approximated by spatial function in tetrahedron "abcd," be presented as

$$\Phi = \mathbf{r} \cdot \mathbf{A} + \mathbf{b}, \tag{13}$$

where **b** and **A** are constant tensors of the same rank as and of rank one order higher than Φ, respectively. For vertices of the tetrahedron we can write

$$\begin{aligned}\Phi_a &= \mathbf{r}_a \cdot \mathbf{A} + \mathbf{b}\\ \Phi_b &= \mathbf{r}_b \cdot \mathbf{A} + \mathbf{b}\\ \Phi_c &= \mathbf{r}_c \cdot \mathbf{A} + \mathbf{b}\\ \Phi_d &= \mathbf{r}_d \cdot \mathbf{A} + \mathbf{b}\end{aligned} \tag{14}$$

or

$$\begin{aligned}\delta\Phi_1 &= \delta\mathbf{r}_1 \cdot \mathbf{A}\\ \delta\Phi_2 &= \delta\mathbf{r}_2 \cdot \mathbf{A}\\ \delta\Phi_3 &= \delta\mathbf{r}_3 \cdot \mathbf{A}\end{aligned} \tag{15}$$

After multiplication of Eq. (15) by respective vectors $\mathbf{i}_1$, $\mathbf{i}_2$, and $\mathbf{i}_3$ and summation of the results, it follows that

$$\mathbf{i}_k \delta\Phi_k = \mathbf{i}_m \delta\mathbf{r}_m \cdot \mathbf{A},$$

thus

$$\mathbf{A} = (\mathbf{i}_m \delta\mathbf{r}_m)^{-1} \cdot \mathbf{i}_k \delta\Phi_k = \mathbf{r}^m \delta\Phi_m. \tag{16}$$

Substituting Eqs. (13) and (16) into Eq. (12) leads to the following:

$$\nabla \odot \Phi \approx \mathbf{i}_k \odot \frac{\partial(\mathbf{r} \cdot \mathbf{A} + \mathbf{b})}{\partial x_k} = \mathbf{i}_k \odot \frac{\partial(\mathbf{i}_j x_j \cdot \mathbf{r}^m \delta\Phi_m)}{\partial x_k} = \mathbf{r}^m \odot \delta\Phi_m = \nabla^* \odot \delta\Phi.$$

It is important that $\delta\mathbf{r}_1$, $\delta\mathbf{r}_2$, and $\delta\mathbf{r}_3$ (with $\delta\Phi_1$, $\delta\Phi_2$, and $\delta\Phi_3$ defined in the same way) can be formed from other combinations of $\mathbf{r}_a$, $\mathbf{r}_b$, $\mathbf{r}_c$, and $\mathbf{r}_d$ not only as shown in Fig. 1, but also as in Fig. 3.

2.3. *Vector Approximation of* ∇ *in "Coordinate Transformation" Style*

Let us introduce local coordinate system (x_1', x_2', x_3') in tetrahedron "abcd" with basis-vectors $\mathbf{i}_1'$, $\mathbf{i}_2'$, and $\mathbf{i}_3'$ directed along $\delta\mathbf{r}_1$, $\delta\mathbf{r}_2$, and $\delta\mathbf{r}_3$, respectively (i.e., $\mathbf{i}_1' = \delta\mathbf{r}_1/|\delta\mathbf{r}_1|, \ldots$) as in Fig. 1. From the following relations,

$$\frac{\partial}{\partial x_k'} = \mathbf{i}_k' \cdot \nabla, \quad k = 1, 2, 3, \tag{17}$$

∇ can be written as

$$\nabla = \mathbf{i}'^k \frac{\partial}{\partial x_k'}, \tag{18}$$

where

$$\mathbf{i}'^1 = \frac{\mathbf{i}_2' \times \mathbf{i}_3'}{\mathbf{i}_1' \cdot (\mathbf{i}_2' \times \mathbf{i}_3')}, \quad \mathbf{i}'^2 = \frac{\mathbf{i}_3' \times \mathbf{i}_1'}{\mathbf{i}_1' \cdot (\mathbf{i}_2' \times \mathbf{i}_3')}, \quad \mathbf{i}'^3 = \frac{\mathbf{i}_1' \times \mathbf{i}_2'}{\mathbf{i}_1' \cdot (\mathbf{i}_2' \times \mathbf{i}_3')}. \tag{19}$$

Using Eqs. (9) and (19), Eq. (18) can be approximated with first-order accuracy as

$$\nabla \approx \mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3 = \nabla^*. \tag{20}$$

Comparison of the right parts of Eqs. (18) and (20) shows that vectors $\mathbf{r}^1$, $\mathbf{r}^2$, and $\mathbf{r}^3$ can be named the vector finite differences along directions x_1', x_2', and x_3'. From Eq. (20) it follows again that

$$\nabla \odot \Phi \approx \nabla^* \odot \Phi.$$

2.4. *Vector Approximation of* ∇ *in n Dimensional (nD) Case*

Expansion of the vector style approximation to nD space is possible; it helps to find general features of ∇^*. Let, by analogy, $\mathbf{r} = \mathbf{i}_k x_k$ be a position vector of a point in nD Cartesian space, where $x_1, \ldots, x_n$ are rectangular orthogonal coordinates with basis vectors $\mathbf{i}_1, \ldots, \mathbf{i}_n$ and the summation convention from 1 to n over dummy subscripts is applied [18]. Let a linear nD tetrahedron with $n + 1$ vertices be an element of an nD mesh.

A transformation similar to the above can be applied to the nD generalization of the formula Eq. (11) as follows:

$$\nabla^* = \mathbf{r}^1 + \mathbf{r}^2 + \cdots + \mathbf{r}^n,$$

$$\mathbf{r}^k = \begin{vmatrix} \delta r_{11} & \delta r_{12} & \cdots & \delta r_{1n} \\ \delta r_{21} & \delta r_{22} & \cdots & \delta r_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{i}_1 & \mathbf{i}_2 & \cdots & \mathbf{i}_n \\ \cdots & \cdots & \cdots & \cdots \\ \delta r_{n1} & \delta r_{n2} & \cdots & \delta r_{nn} \end{vmatrix} \Bigg/ \begin{vmatrix} \delta r_{11} & \delta r_{12} & \cdots & \delta r_{1n} \\ \delta r_{21} & \delta r_{22} & \cdots & \delta r_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \delta r_{k1} & \delta r_{k2} & \cdots & \delta r_{kn} \\ \cdots & \cdots & \cdots & \cdots \\ \delta r_{n1} & \delta r_{n2} & \cdots & \delta r_{nn} \end{vmatrix}, \quad k = 1, \ldots, n. \tag{21}$$

Here $\delta\mathbf{r}_1, \ldots, \delta\mathbf{r}_n$ are independent variations of position vectors of nD tetrahedron vertices. The denominator in Eq. (21) is equal to $n!V$, where V is volume of nD tetrahedron.

Equation (9) follows from Eq. (21) when $n = 3$. For $n = 2$ we obtain

$$\nabla^* = \mathbf{r}^1 + \mathbf{r}^2,$$

$$\mathbf{r}^1 = \begin{vmatrix} \mathbf{i}_1 & \mathbf{i}_2 \\ \delta r_{21} & \delta r_{22} \end{vmatrix} \Bigg/ \begin{vmatrix} \delta r_{11} & \delta r_{12} \\ \delta r_{21} & \delta r_{22} \end{vmatrix}, \quad \mathbf{r}^2 = \begin{vmatrix} \delta r_{11} & \delta r_{12} \\ \mathbf{i}_1 & \mathbf{i}_2 \end{vmatrix} \Bigg/ \begin{vmatrix} \delta r_{11} & \delta r_{12} \\ \delta r_{21} & \delta r_{22} \end{vmatrix}. \tag{22}$$

3. VECTOR APPROXIMATION AND FINITE ELEMENT (FE) METHOD

In the simpliest and popular case the field variables in the FE methods are approximated by linear combinations of known basis functions (or shape functions) $N_k(\mathbf{r})$. If Φ is an approximate solution, then we can write a series expansion [11, 17] and more,

$$\Phi(\mathbf{r}) = \sum_I \Phi_I N_I(\mathbf{r}), \tag{23}$$

where Φ_I are unknown nodal variables, and summation extends over all nodes I. In the standard FE method the functions $N_I(\mathbf{r})$ are chosen to be locally defined polynomials within

each element and to be zero outside the considered element. As a consequence the local shape functions satisfy the following conditions on each element (e) with I being a node of (e):

$$\begin{aligned} N_I^{(e)}(\mathbf{r}) &= 0, \qquad \text{if } \mathbf{r} \notin (e) \\ N_I^{(e)}(\mathbf{r}_J) &= \delta_{IJ}, \quad \text{for any node of } \mathbf{r}_J \\ \sum_I N_I^{(e)}(\mathbf{r}) &= 1, \qquad \text{for all } \mathbf{r} \in (e). \end{aligned} \tag{24}$$

The global function N_I in Eq. (23) is obtained by assembling the combinations $N_I^{(e)}$ of all the elements to which node I belongs.

Let the mesh consist of linear tetrahedral elements "abcd" (Fig. 1), and variable Φ is approximated by linear spatial function written in vector form. Regroup the relation into the form,

$$\Phi = N_a^{(e)}(\mathbf{r})\Phi_a + N_b^{(e)}(\mathbf{r})\Phi_b + N_c^{(e)}(\mathbf{r})\Phi_c + N_d^{(e)}(\mathbf{r})\Phi_d, \tag{25}$$

and find all $N_I^{(e)}(\mathbf{r})$. The classical "component" solutions of this problem can be found in almost any manual in FE methods. For thr vertices of the tetrahedron we can write vector form relations as follows. The value of $\mathbf{A}$ is presented by Eq. (16). The value of $\mathbf{b}$ can be defined from Eq. (14) as

$$\mathbf{b} = \Phi_a - \mathbf{r}_a \cdot \mathbf{A}. \tag{26}$$

Combining of Eqs. (13), (16), and (26) gives all local shape functions $N_I^{(e)}(\mathbf{r})$ in Eq. (25) such as

$$\begin{aligned} N_a^{(e)}(\mathbf{r}) &= 1 - (\mathbf{r}_a - \mathbf{r}) \cdot (\mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3) \\ N_b^{(e)}(\mathbf{r}) &= (\mathbf{r}_a - \mathbf{r}) \cdot \mathbf{r}^1 \\ N_c^{(e)}(\mathbf{r}) &= (\mathbf{r}_a - \mathbf{r}) \cdot \mathbf{r}^2 \\ N_d^{(e)}(\mathbf{r}) &= (\mathbf{r}_a - \mathbf{r}) \cdot \mathbf{r}^3. \end{aligned}$$

Obviously the conditions of Eq. (24) are satisfied.

We can obtain also the vector form for the relations,

$$\begin{aligned} \nabla N_a^{(e)}(\mathbf{r}) &= \mathbf{r}^1 + \mathbf{r}^2 + \mathbf{r}^3 \\ \nabla N_b^{(e)}(\mathbf{r}) &= -\mathbf{r}^1 \\ \nabla N_c^{(e)}(\mathbf{r}) &= -\mathbf{r}^2 \\ \nabla N_d^{(e)}(\mathbf{r}) &= -\mathbf{r}^3, \end{aligned}$$

which are widely used in the Galerkin method.

The results obtained show that the standard FE method of approximation can be performed in vector form also, again including the set of vectors $\mathbf{r}^1$, $\mathbf{r}^2$, and $\mathbf{r}^3$. It means

that the vectors $\mathbf{r}^1$, $\mathbf{r}^2$, and $\mathbf{r}^3$ play a general role in approximation in tetrahedron cells. Second, when Φ in Eq. (25) is the vector (or tensor) variable, then there is no need to repeat the similar transformations for all Φ components, as is usually done in computer implementation.

4. VECTOR APPROXIMATION VERSUS PROJECTION APPROXIMATION

Let us compare traditional (projection) form with vector form of approximation for ∇E, where E is a scalar (for the simplest case).

4.1. *1D Case*

For a 1D case, both types of approximation are equivalent.

4.2. *2D Case*

Let 2D nonorthogonal mesh in rectangular coordinates (x_1, x_2) or (x, y) consist of linear triangular elements (Fig. 2). Traditional ∇E approximation gives the following system (for example, [8], etc.):

$$\begin{aligned}\nabla_x E &\approx \frac{(E_b - E_a)(y_c - y_b) - (E_c - E_b)(y_b - y_a)}{(x_b - x_a)(y_c - y_b) - (x_c - x_b)(y_b - y_a)},\\ \nabla_y E &\approx \frac{(E_c - E_b)(x_b - x_a) - (E_b - E_a)(x_c - x_b)}{(x_b - x_a)(y_c - y_b) - (x_c - x_b)(y_b - y_a)}.\end{aligned} \tag{27}$$

Vector approximation of ∇E can be written as

$$\nabla E \approx \mathbf{r}^1 \delta E_1 + \mathbf{r}^2 \delta E_2, \tag{28}$$

where sets $\mathbf{r}^1$ and $\mathbf{r}^2$ are defined by Eq. (9).

4.3. *3D Case*

Let 3D nonorthogonal mesh in rectangular coordinates (x_1, x_2, x_3) or (x, y, z) consist of linear tetrahedron elements "abcd" (Fig. 1). A traditional approximation of ∇E leads to the

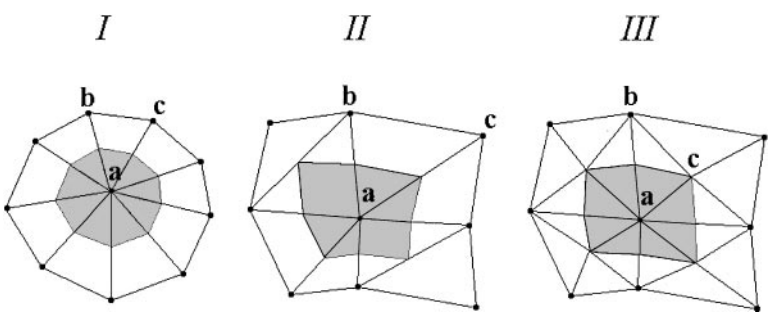

FIG. 2. Fragments of meshes of types *I*, *II*, and *III* (• is a node, "abc" is an element).

formulas

$$\begin{aligned}\nabla_x E \approx \{&[(y_c - y_b)(z_d - z_c) - (z_c - z_b)(y_d - y_c)](E_b - E_a) + [(y_c - y_b)(z_d - z_c)\\ &- (z_c - z_b)(y_d - y_c)](E_c - E_b) + [(y_b - y_a)(z_d - z_b)\\ &- (z_b - z_a)(y_d - y_b)](E_d - E_c)\}/\{[(y_c - y_b)(z_d - z_c)\\ &- (z_c - z_b)(y_d - y_c)](x_b - x_a) + [(z_c - z_b)(x_d - x_c)\\ &- (x_c - x_b)(z_d - z_c)](y_c - y_b) + [(x_b - x_a)(y_d - y_b)\\ &- (y_b - y_a)(x_d - x_b)](z_d - z_c)\}\end{aligned} \tag{29}$$

$$\nabla_y E \approx \ldots$$

$$\nabla_z E \approx \ldots,$$

where $\nabla_y E$ and $\nabla_z E$ are available by cyclic permutation in $\nabla_x E$. Vector approximation of ∇E gives

$$\nabla E \approx \mathbf{r}^1 \delta E_1 + \mathbf{r}^2 \delta E_2 + \mathbf{r}^3 \delta E_3. \tag{30}$$

Obviously, from a "mathematical" point of view Eqs. (27) and (28) or Eqs. (29) and (30) are equivalent, but from a "users" point of view Eqs. (28) and (30) are several times more compact and, apart from the cleaner syntax, conserve the same visual appearance of ∇E. The vector form of presentation is still more effective when E is a vector or tensor variable. One can easily approximate the more complex expressions such as $\mathbf{r}\nabla_0$, $\nabla_0\mathbf{r}$, and so forth.

5. SOME DETAILS OF VECTOR DIFFERENCE SCHEMES

We shall consider (shortly) some specific details of vector form difference schemes (in FV style, as a rule). Note that many variants of the numerical scheme in vector form are available depending on concrete problems, region shapes, order of approximation, type of cell, user's inclinations, and so forth. It is necessary to emphasize that we will not discuss numerical algorithms because they do not directly involve the vector form of approximation.

5.1. *Nonorthogonal Meshes*

Consider 2D or 3D nonorthogonal meshes whose cells are the unions of one or more rectangles or tetrahedrons, respectively. Any elements may have individual material properties (e.g., density). The schematics of 2D meshes are shown in Fig. 2 for one irregular and two regular meshes, respectively. The control volume is shown here by a grey color (we conserved term "volume" for 2D case also).

The type *I* mesh is formed of triangular elements directly. The meshes of type *II* and *III* are formed of quadrangular cells divided by triangular elements ("abc"). The directions of diagonals in the cells of the type *II* mesh can be in the same direction as in Fig. 2, alternating in a number of ways, or they can vary randomly. The cell midpoint here is placed in the middle of the diagonal. Each cell of type *III* mesh is divided by four triangles ("abc") with midpoint "c" chosen at center of mass of the cell. Midpoint is not an independent node and has no control volume.

For the sake of clarity, we shall use the mesh of type *III*, but the vector form of approximation can be applied to other types of cells and meshes, either irregular or mixed.

A 3D cell of type *III* mesh is shown in Fig. 3 and consists of 24 tetrahedra ("abcd," "abdf," ...). Each of the cells 12 edges is adjacent to two tetrahedra within that cell (for example, edge "ab" and tetrahedra "abcd" and "abdf"). Midpoints are found for all edges, faces, and the cell itself. For any midpoint "z," Φ_z is defined. For example, $\Phi_e = (\Phi_{i+1,j,k} + \Phi_{i,j,k})/2$, $\Phi_c = (\Phi_{i+1,j+1,k} + \Phi_{i,j+1,k} + \Phi_{i+1,j,k} + \Phi_{i,j,k})/4$, etc. The cell volume is separated between nodal control volumes $V_{i,j,k}$, $V_{i+1,j,k}$, $V_{i+1,j+1,k}$, $V_{i,j+1,k}$, $V_{i,j,k+1}$, $V_{i+1,j,k+1}$, $V_{i+1,j+1,k+1}$, and $V_{i,j+1,k+1}$. For example, the value of $V_{i,j,k}$ is the sum of 48 parts ("acde," "adef," ..., Fig. 3) from all tetrahedra that touch node (i, j, k), and the triangle "cde" is one of 48 triangles (less for boundaries and corner nodes) which form the control surface $S_{i,j,k}$ and enclose the control volume $V_{i,j,k}$. The outward-oriented surface element s_{cde} for triangle "cde" is equal to $(\delta\mathbf{r}_2 \times \delta\mathbf{r}_3)/2$ for the node (i, j, k) and the same but with opposite direction for the node $(i + 1, j, k)$.

5.2. *Vector Form of Computer Programming*

Vector schemes have theoretical interest, only unless we use the vector notation through all the stages: governing equations → numerical scheme → computer program nomenclature. Fortunately, algorithmic languages such as Fortran90 or C++ allow the definition of new data types. The variables of this type can be as easily applied as variables of intrinsic types (for example, as in complex types in Fortran77). New data types once developed and tested can be used further as standard. Obviously, they have to provide an intuitive interface to users. In our case, they are 2D and 3D vector and tensor data types.

In other words, one must not develop subroutines for computing volumes and areas, but must develop and work with new data types (new classes) for vector and tensor objects with overloaded operations such as dot, cross, and diadic (see sample of code below).

Let us discuss some examples of difference schemes in vector form in the region of complex shapes, where a 3D mesh of type *III* and of size $I \times J \times K$ is introduced. Let's study a 3D model problem,

$$\frac{\partial}{\partial t}\int_V \Phi\, dV = \oint_S (d\mathbf{s} \cdot \nabla\Phi), \tag{31}$$

where Φ is any type variable of (scalar, vector, or tensor).

The explicit scheme for Eq. (31) has the form,

$$\frac{\Phi^{n+1}_{i,j,k} - \Phi^n_{i,j,k}}{\delta t} V_{i,j,k} = \Theta^n_{i,j,k} \equiv \sum_{S_{i,j,k}} f^n_{rst}, \tag{32}$$

where δt is time step, n is number of the time step, f^n_{rst} is influx over "rst," "rst" is any triangle part of $S_{i,j,k}$, and $\Theta^n_{i,j,k}$ is total influx over $S_{i,j,k}$. For example,

$$f^n_{cde} = \mathbf{s}_{cde} \cdot \nabla^*\Phi^n = \frac{\delta\mathbf{r}_2 \times \delta\mathbf{r}_3}{2} \cdot \left(\mathbf{r}^1\delta\Phi^n_1 + \mathbf{r}^2\delta\Phi^n_2 + \mathbf{r}^3\delta\Phi^n_3\right) \tag{33}$$

for triangle "cde" in Fig. 3. It is not necessary to write all addendums in the right part of Eq. (32) explicitly. Instead, a simple algorithm can be used for computing $\Phi^{n+1}_{i,j,k}$ according to Eqs. (32) and (33)

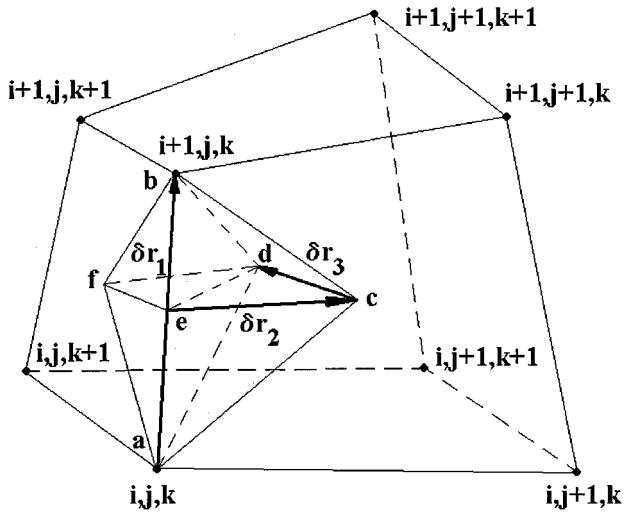

FIG. 3. 3D cell of type III mesh including 24 tetrahedral elements (• are nodes).

```
1  PhiType Φ(I, J, K), Θ(I, J, K), δΦ1, δΦ2, δΦ3, f_cde;   % PhiType and Vector are
2  Vector   r(I, J, K), δr1, δr2, δr3, r^1, r^2, r^3;      % new data types developed by user
3  Real     V(I, J, K), V_abcd;                            % Intrinsic type for real values
4  do i=1 to I; do j=1 to J; do k=1 to K;                  % Cycles over all nodes
5    r_{i,j,k} ← mesh generator;
6    Φ_{i,j,k} ← initial conditions;
7  enddo; enddo; enddo;
8  do t = 0 step δt to t_finish;                           % Cycle by time
9    do i=1 to I; do j=1 to J; do k=1 to K;                % Cycles over all nodes
10     Φ_{i,j,k} := 0;                                     % "0" is PhiType zero
11   enddo; enddo; enddo;
12   do i=1 to I-1; do j=1 to J-1; do k=1 to K-1;         % Cycles over all cells
13     do n=1 to 12;                                     % Cycle over all tetrahedra in this cell
14       ...                                             % Cyclic permutation of indices in the cell and definition
15       ...                                             % indices of vertices "a" and "b" in the tetrahedron "abcd"
16       δr1 := r_{i_b,j_b,k_b} − r_{i_a,j_a,k_a}; δr2 := ...; δr3 := ...;          % Eq. (4)
17       δΦ1 := Φ_{i_b,j_b,k_b} − Φ_{i_a,j_a,k_a}; δΦ2 := ...; δΦ3 := ...;          % Eq. (8)
18       V_abcd := δr1 · (δr2 × δr3)/6                   % Volume of tetrahedron "abcd"
19       r^1 := (δr2 × δr3)/(6V_abcd); r^2 := ...; r^3 := ...;                      % Eq. (9)
20       f_cde := (δr2 × δr3)/2 · (r^1 ⊙ δΦ^1 + r^2 ⊙ δΦ^2 + r^3 ⊙ δΦ^3);           % Eq. (33)
21       Θ_{i_a,j_a,k_a} := Θ_{i_a,j_a,k_a} + f_cde;     % f_cde is joining to Θ_{i_a,j_a,k_a} and Θ_{i_b,j_b,k_b}
22       Θ_{i_b,j_b,k_b} := Θ_{i_b,j_b,k_b} − f_cde;     % (taking into account the sign)
23       V_{i_a,j_a,k_a} := V_{i_a,j_a,k_a} + V_abcd/2;  % V_abcd is contributed to
24       V_{i_b,j_b,k_b} := V_{i_b,j_b,k_b} + V_abcd/2;  % corresponding nodal volumes
25     enddo; enddo;
26     do i=1 to I; do j=1 to J; do k=1 to K;            % Cycles over all nodes
27       Φ_{i,j,k} := Φ_{i,j,k} + δt * Θ_{i,j,k}/V_{i,j,k}   % Eq. (32). For boundary nodes Φ_{i,j,k} := BC
28     enddo; enddo; enddo;
29 enddo;
```

Using variables of new types (lines 1, 2) allows the conservation of the syntax of vector difference schemes in computer programming (lines 16–22, 27). For example, the type **Vector** (line 2) was developed for manipulations with 3D vector variables. As a result, the program size decreased and simplified in comparison with the traditional "component" program, where huge similar expressions for all dimensions such as Eq. (29) are necessary to the programming. Vector technology also allows a decrease in syntax errors and easier modification of algorithms and program codes.

To solve governing Eq. (31) with different types of Φ (scalar, vector, or tensor) we have to define **PhiType** (line 1) as **Real**, **Vector** or **Tensor**, respectively. When Φ is a tensor variable (for example), the object-oriented programming allows the introduction of one array of tensor type for Φ. In traditional programming we must introduce nine arrays for any components of tensor Φ, correspondingly.

The remainder "numerical" part of computer program (beginning from line 2) is reusable. It is easy to construct an absolutely implicit or ADI [11] numerical scheme in vector form.

Other examples of approximation in vector form are here.

1. The scheme for conservation law of mass in any tetrahedral element of 3D mesh can be written as

$$\frac{\rho^{n+1}-\rho^n}{\delta t} = -\rho \mathbf{r}^m \cdot \delta \mathbf{v}_m.$$

Explicit/implicit properties of right-hand side are not discussed here.

2. When fluid flows through Eulerian mesh, the conservation law of momentum includes the convective term

$$\oint_S (d\mathbf{s} \cdot \mathbf{v}\Phi), \tag{34}$$

where Φ is convected substance. An approximation of the surface integral Eq. (34) for the same typical part "cde" of the control surface $S_{i,j,k}$ can be obtained as in upwind difference schemes [11]

$$\int_{S_{cde}} (d\mathbf{s} \cdot \mathbf{v}\Phi) = \frac{a+|a|}{2}\Phi_{i,j,k} + \frac{a-|a|}{2}\Phi_{i+1,j,k},$$

where

$$a = \frac{\delta \mathbf{r}_2 \times \delta \mathbf{r}_3}{2} \cdot \frac{\mathbf{v}_{i,j,k} + \mathbf{v}_{i+1,j,k}}{2}.$$

3. The next scheme within a tetrahedral element

$$\mathbf{T} = -p\mathbf{I} + \left\{ h_1(\delta \mathbf{r}_m \mathbf{r}_0^m) \cdot \left(\mathbf{r}_0^n \delta \mathbf{r}_n\right) + h_2\left[\left(\delta \mathbf{r}_k \mathbf{r}_0^k\right) \cdot \left(\mathbf{r}_0^q \delta \mathbf{r}_q\right)\right]^2 \right\}$$

can be written for constitutive relationship Eq. (2). Here $\mathbf{r}_0^m = \mathbf{r}^m|_{t=0}$.

6. PRACTICAL "PROOF" OF THE PERFORMANCE OF THE VECTOR FORM OF DISCRETIZATION

As a provisional step the results of simulation of hyperelastic 3D pyramid twisting are shown in Fig. 4. Governing equations included the momentum equation and the rheological model Eq. (2), where

$$h_1 = 2\left(\frac{\partial W}{\partial I_1} + I_1\frac{\partial W}{\partial I_2}\right), \quad h_2 = -2\frac{\partial W}{\partial I_2}, \quad I_1 = \mathrm{tr}\mathbf{B}, \quad I_2 = \frac{1}{2}(\mathrm{tr}^2\mathbf{B} - \mathrm{tr}\mathbf{B}^2),$$

and W is the strain energy function defined as follows [19]:

$$W = C_{10}(I_1 - 3) + C_{01}(I_2 - 3) + C_{11}(I_1 - 3)(I_2 - 3) + C_{20}(I_1 - 3)^2 + C_{30}(I_1 - 3)^3. \tag{35}$$

The material properties $C_{10}, \ldots, C_{30}$ in Eq. (35) were taken similar to natural rubber. The volume of each tetrahedron element of 3D Lagrangian mesh (Fig. 3) was conserved. As a numerical method we used a combination of artificial compressibility and ADI methods [11].

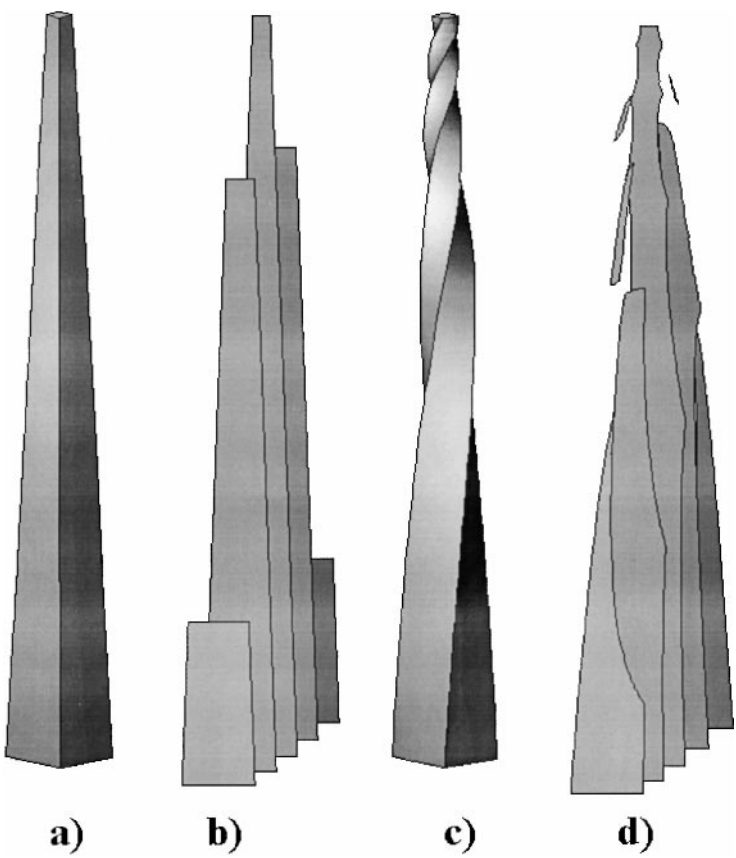

FIG. 4. 3D hyperelastic pyramid: (a) undeformed state; (c) after twisting around axis x_1 by 2π; (b) and (d) 2D cross-sections of the pyramid normal to direction x_3 at $x_3 = 0.1, 0.3, 0.5, 0.7, 0.9$ for (a) and (c), respectively.

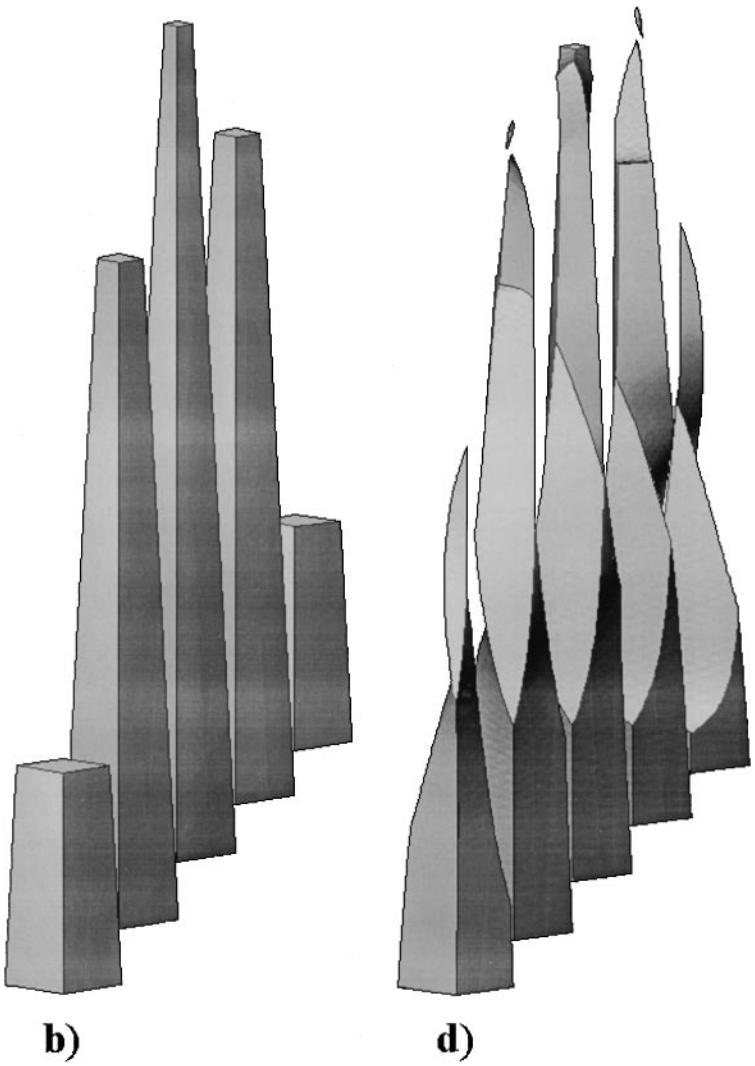

FIG. 5. 4D hyperelastic pyramid: (a) undeformed state and (c) after twisting around axis x_1 on 2π, axis x_2 on 2π and axis x_3 on 2π can't be shown (compared with Fig. 4); (b) and (d) 3D cross-sections of the pyramid normal to direction x_4 at $x_4 = 0.1, 0.3, 0.5, 0.7, 0.9$ for (a) and (c) states, respectively.

The nondimensional size of the pyramid in rectangular coordinates (x_1, x_2, x_3) was 10 in direction x_1; the square bottom (at $x_1 = 0$) and top (at $x_1 = 10$) were 1^2 and 0.25^2, respectively. The bottom was fixed; the top was twisted initially (before iteration begins) around axes x_1 by 2π and fixed; and other surfaces were free. The rheological models and algorithms were the same as before. Simulations of undeformed, and twisted 3D pyramids and their cross sections normal to axis x_3 are shown in Fig. 4.

Increasing problem dimensions, increasing complexity of the geometry of simulations, or going from linear to nonlinear problem usually increase programming efforts. So for a "proof" of vector technique performance we assumed that the governing equations conserve invariant vector form in nD Cartesian space and generalized the problem of pyramid twisting for 4D case. The nondimensional size of the 4D pyramid in rectangular coordinates (x_1, x_2, x_3, x_4) was 10 along direction x_1; the cubical bottom (at $x_1 = 0$) and top (at

$x_1 = 10$) were 1^3 and 0.25^3, respectively. The bottom was fixed; the top was initially twisted around each axis, x_1, x_2, and x_3, by 2π and fixed; and other 3D surfaces of the 4D pyramid were free. A Lagrangian mesh with $9 \times 9 \times 9 \times 85$ 4D cells of type *III* was used. Any cell of mesh consisted of 192 4D tetrahedra. Of course we can't draw the 4D solid explicitly, so subitems (a) and (c) are absent in Fig. 5 in contrast to Fig. 4. But the simulation allows the obtainment any 3D cross section of 4D pyramid. Some cross sections normal to axis x_4 before and after twisting are shown in Fig. 5.

The main performance of the vector technique in this example is that the listing of the "numerical" part of the computer program for 4D simulation does not increase many times but less than 40% in comparison with the 3D case (based on vector technique also). Approximately 70% of the 3D program was reusable.

7. CONCLUSIONS

The vector form of approximation and computer programming does not replace the projection forms. These two forms can complement each other (in the same way that Fortran does not replace an assembler). The vector form of approximation and programming can be applied with great effect in numerical simulation of problems with both "symmetrical" and "degenerated" governing equations, such as plate and rod space bending or fluid film flows.

ACKNOWLEDGMENTS

The authors thank Drs. J. A. Levitt and V. G. Lenss in the Ford Research Laboratories for their support and collaboration. The authors especially thank Professor W. W. Schultz at the University of Michigan for his help and many useful discussions. This work is partially supported by Yeungnam University BK21 project.

REFERENCES

1. P. G. Giarlet, *Mathematical Elasticity*, Vol. 1, *Three Dimensional Elasticity* (North-Holland, Amsterdam, 1993).
2. A. Solov'ev, E. Solov'eva, V. Tishkin, M. Shashkov, and A. Favorsskii, Approximation of finite difference operators on a mesh of dirichlet cells, *Diff. Eq.* **22**, 881 (1986).
3. A. Favorskii, A. Samarskii, M. Shashkov, and V. Tishkin, Operational finite-difference schemes, *Diff. Eq.* **17**, 854 (1981).
4. J. M. Hyman and M. Shashkov, Natural discretizations for the divergence, gradient, and curl on logically rectangular grids, *Int. J. Comput. Math. Appl.* **33**, 81 (1997).
5. M. Shashkov and J. Hyman, The approximation of boundary conditions for mimetic finite difference methods, *Comput. Math. Appl.* **36**, 79 (1998).
6. M. Shashkov and J. Hyman, The orthogonal decomposition theorems for mimetic finite difference methods, *SIAM J. Numer. Anal.* **36**, 788 (1999).
7. M. Shashkov and J. Hyman, The adjoint operators for natural discretizations for the divergence, gradient and curl on locally rectangular grids, *IMACS J. Appl. Numer. Math.* **25**, 413 (1997).
8. M. Shashkov, *Conservative Finite-Difference Methods on General Grids*, Symbolic and Numeric Computation Series (CRC Press, New York, 1996).
9. E. J. Caramana, D. E. Burton, M. Shashkov, and P. P. Whalen, The construction of compatible hydrodynamics algorithms utilizing conservation of total energy, *J. Comput. Phys.* **146**, 227 (1998).
10. A. Favorskii, A. Samarskii, M. Shashkov, and V. Tishkin, Employment of the reference-operator method in the construction of finite-difference analogs of tensor operations, *Diff. Eq.* **18**, 881 (1982).

11. C. Hirsch, *Numerical Computation of Internal and External Flows*, Vols. 1, 2, (Wiley, NewYork, 1988).
12. M. Shashkov and S. Steinberg, Support-operators finite-difference algorithms for general elliptic problems, *J. Comput. Phys.* **118**, 131 (1995).
13. M. Shashkov and S. Steinberg, Solving diffusion equations with rough coefficients in rough grids, *J. Comput. Phys.* **129**, 383 (1996).
14. J. Hyman, M. Shashkov, and S. Steinberg, The numerical solution of diffusion problems in strongly Heterogeneous non-isotropic Materials, *J. Comput. Phys.* **132**, 130 (1997).
15. K. Salari and S. Steinberg, Flux-corrected transport in a moving grid, *J. Comput. Phys.* **111**, 24 (1994).
16. M. L. Wilkins, *Calculation of Elastic–Plastic Flow, in Methods in Computational Physics*, edited by B. Alder, S. Fernbach, M. Rotenberg (Academic Press, NewYork, 1964), Vol. 3, p. 211.
17. O. C. Zienkewicz, *The Finite Element Method*, 3rd ed. (McGraw-Hill, NewYork, 1977).
18. M. R. Spiegel, *Vector Analysis and the Introduction to Tensor Analysis* (Schaum, NewYork, 1959).
19. R. S. Rivlin, *Large Elastic Deformations in Rheology, Theory and Application*, edited by F. R. Eirich (Academic Press Inc., San Diego, 1956), Vol. 1, p. 421.