On Particle Path Generation Based on Quadrilinear Interpolation and Bernstein-Bézier Polynomials

Bernd Hamann, *Member, IEEE,* Donghua Wu, and Robert J. Moorhead II

***Abstract*—Particle path computation in unsteady 3D vector fields given in discrete, structured form (i.e., as a hexahedral curvilinear grid) requires the local approximation of the vector field and the path. Quadrilinear interpolation and Bernstein-Bézier polynomials are used for the local vector field and path approximation. The next point in a sequence of points on a particle path is computed using this local approximation. Bernstein-Bézier polynomials are primarily used in geometric modeling, and their properties allow direct computation of points on a particle path.**

***Index Terms*—Approximation, Bernstein-Bézier polynomial, particle path, curvilinear grid, path line, scientific visualization, structured grid, trajectory, vector field.**

I. Introduction

THIS paper is concerned with the computation of a point sequence approximating a particle path in an unsteady 3D vector field. It is assumed that a 3D structured grid, composed of hexahedral elements, is given. The coordinates of the grid vertices are allowed to change over time, while the topology–the connectivity among grid vertices–must remain unchanged. Given an initial 3D position $\mathbf{X}_0$, the described algorithm generates a point sequence $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, ... on the resulting particle path based on quadrilinear interpolation and a local particle path approximate in Bernstein-Bézier representation. The time step used to generate a next point is chosen adaptively using a one-step method for differential equations.

The main reasons for the proposed method are its elegance, its numerical stability due to the use of Bernstein-Bézier polynomials with their "nice" numerical and arithmetical properties, and the fact that the proposed method uses a direct, explicit approach for approximating particle paths–instead of using a Runge-Kutta method. The proposed method uses adaptive time steps for generating points on a particle path, just like a Runge-Kutta method. Therefore, the resulting particle path approximations using either approach are very much much the same. The proposed method is merely an alternative to more standard methods.

B. Hamann is with the Department of Computer Science, University of California at Davis, Davis, CA 95616-8562; e-mail: hamann@cs.ucdavis.edu.

D. Wu and R.J. Moorhead II are with the NSF Engineering Research Center for Computational Field Simulation and the Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS 39762.

IEEECS Log Number V95017.

There must be a one-to-one map from each 4D physical, curvilinear grid cell to a 4D computational, rectilinear cell (= unit cube). Once the initial point $\mathbf{X}_0$ has been transformed from physical to computational space, all computations are performed in computational space. A point $\mathbf{x}$ in a cell in computational space is identified with a local parameter tuple (ξ, η, ζ, τ), where ξ, η, ζ, and τ are the relative offsets from the lower left corner of the cell. The relation between physical space, computational space, and local parameter space is illustrated for the 2D case in Fig. 1.

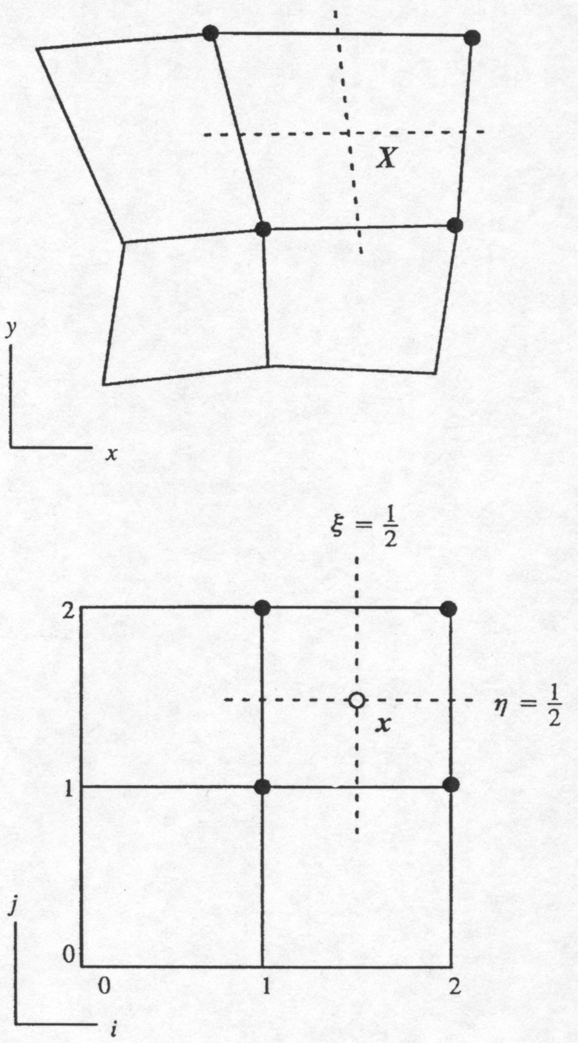


Fig. 1. Physical space, computational space, and local parameter space–2D case.

The local parameters of a point with respect to a cell in computational space are used to perform quadrilinear interpolation over the cell and locally approximate the vector field. Initially, vector values are provided at the grid vertices in physical space. These values are transformed to computational space by using local estimates of the Jacobian relating the two spaces.

Most current vector field visualization techniques require a robust, numerically stable, and adaptive method for approximating particle paths. The method proposed in this paper should be viewed as an alternative to existing methods that satisfy exactly these requirements. Recent progress in vector field visualization can be found in [5], [8], [9], [10], [11], [12], [13], [19]. In particular, particle tracing algorithms are described in [2], [7], [14], [16], [20]. A general overview of current scientific visualization techniques is provided in [4]. The technique presented here utilizes standard techniques of geometric modeling that are discussed in [3], [6]. In particular, certain arithmetical operations for curves in Bernstein-Bézier representation are needed which are covered in [15]. The most common 2D- and 3D-grid generation methods are described in detail in [17], and an overview of the state-of-the-art in grid generation is provided in [18].

In summary, the approach used to generate a particle path requires these modules:

1) Generating a uniform, rectilinear grid in 4D computational space and associated vectors at each computational grid point (i, j, k, l) from a curvilinear grid in 4D physical space.
2) Transforming an initial point $\mathbf{X}_0$ from 4D physical space to 4D computational space and computing local parameter tuple (ξ, η, ζ, τ) with respect to the grid cell in computational space containing the initial point.
3) Approximating the unsteady vector field in computational space using quadrilinear interpolation for each grid cell.
4) Approximating the particle path in a local parameter space by a Bézier curve considering position, velocity, and acceleration at the previous point on the particle path.
5) Computing the next point on the particle path by considering the local vector field and particle path approximates at the previous point.
6) Transforming the next point on the particle path from local parameter space to physical space.
7) Estimating the local error in physical space and–depending on some maximal error tolerance–repeating the steps 5) and 6) using a smaller time step.

These steps are described in detail in the following sections.

II. Transformations Relating Physical and Local Parameter Space

The computation of particle paths is extremely simplified when dealing with uniform computational space instead of curvilinear physical space. One needs to transform points and vectors from computational space to physical space and vice versa. Points on a particle path are computed in computational space and transformed to physical space. A point in a unit grid cell in computational space has local parameter values with respect to this cell and a quadrilinear interpolant. These parameter values are denoted by ξ, η, ζ, and τ ("local parameter space").

First, all vectors given at grid points in physical space are transformed to computational space using local estimates of the Jacobian

$$J = \begin{pmatrix} x_\xi & x_\eta & x_\varsigma & x_\tau \\ y_\xi & y_\eta & y_\varsigma & y_\tau \\ z_\xi & z_\eta & z_\varsigma & z_\tau \\ t_\xi & t_\eta & t_\varsigma & t_\tau \end{pmatrix} = \begin{pmatrix} \frac{\partial}{\partial \xi} x(\xi, \eta, \varsigma, \tau) \ldots & \frac{\partial}{\partial \tau} x(\xi, \eta, \varsigma, \tau) \\ \vdots & \vdots \\ \frac{\partial}{\partial \xi} t(\xi, \eta, \varsigma, \tau) \ldots & \frac{\partial}{\partial \tau} t(\xi, \eta, \varsigma, \tau) \end{pmatrix} \tag{2.1}$$

relating physical and computational space.

Central differences are used to obtain an estimate $\tilde{J}$ of the (transpose of the) Jacobian at each grid point $\mathbf{X}_{i,j,k,l}$. This estimate is given by

$$\tilde{J} = \frac{1}{2} \begin{pmatrix} x_{i+1,j,k,l} - x_{i-1,j,k,l} & \cdots & t_{i+1,j,k,l} \cdots - t_{i-1,j,k,l} \\ \vdots & & \vdots \\ x_{i,j,k,l+1} - x_{i,j,k,l-1} & \cdots & t_{i,j,k,l+1} - t_{i,j,k,l-1} \end{pmatrix} \tag{2.2}$$

Each vector $\mathbf{V}_{i,j,k,l} = (U_{i,j,k,l}, V_{i,j,k,l}, W_{i,j,k,l})$ (physical space) is mapped to the vector

$$\mathbf{v}_{i,j,k,l} = (u_{i,j,k,l}, v_{i,j,k,l}, w_{i,j,k,l}) = \mathbf{V}_{i,j,k,l}\, \tilde{J}^{-1}$$

(computational space).

Remark 2.1. It should be noted that this central difference scheme is only one of many ways to compute the terms in the Jacobian. If possible, the method used should be consistent with that used in the flow solver algorithm so that the velocities are correctly recovered. It is even better to save the contravariant velocities directly from the flow solver in order to not have to transform the velocity field. Another approach, which is more consistent with the quadrilinear interpolation function used here, is to use the partial derivatives of (2.3) (see below) to obtain analytical expressions for the terms in the Jacobian. A good scheme for the approximation of velocities is absolutely essential since the accelerations are derived from them.

Next, the initial point $\mathbf{X}_0 = (x_0, y_0, z_0, t_0)$ in physical space must be transformed to local parameter space. The grid point closest to $\mathbf{X}_0$ must be identified. The grid cells are stored in an octree, which speeds up the search process. The leaf node of this octree that corresponds to the region containing $\mathbf{X}_0$ is identified, and only the cells associated with this node are considered for the identification of the grid point closest to $\mathbf{X}_0$. All cells sharing this closest grid point as a common vertex (neighboring cells) are candidates that can contain the initial point. An iterative procedure is used to find out which of the neighboring cells contains the point. This procedure assumes that each cell can be expressed by a one-to-one quadrilinear map from (local) parameter space to physical space. Thus, a point **X** in a cell in 4D physical space is given by the map

$$\mathbf{X}(\xi,\eta,\zeta,\tau) = \big(x(\xi,\eta,\varsigma,\tau),\, y(\xi,\eta,\varsigma,\tau),\, z(\xi,\eta,\varsigma,\tau),\, t(\xi,\eta,\varsigma,\tau)\big) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1}\sum_{l=0}^{1} \mathbf{X}_{i,j,k,l} B_i^1(\xi) B_j^1(\eta) B_k^1(\varsigma) B_l^1(\tau), \quad \xi,\eta,\varsigma,\tau, \in [0,1], \tag{2.3}$$

where $X_{i,j,k,l}$ is a vertex of the 4D grid cell containing X, and $B_I^1(x)$ is the Bernstein-Bézier polynomial $(1-x)^{1-I}\, x^I$. If a grid cell contains the initial point $\mathbf{X}_0$, a (local) parameter tuple (ξ, η, ζ, τ), $\xi, \eta, \zeta, \tau \in [0, 1]$, can be computed. Otherwise, one of the (local) parameters will not be in the interval [0, 1].

Assuming that the point $\mathbf{X}_0$ lies within a particular cell, the associated local "parameter cube" for this cell is evaluated at $\mathbf{x}_0 = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$ yielding the point Y_0. The difference vector $\mathbf{D}_0 = X_0 - \mathbf{Y}_0$ is computed and transformed into (local) parameter space yielding the vector $\mathbf{d}_0 = \mathbf{D}_0 \tilde{J}^{-1}$, where $\tilde{J}$ is (the transpose of) a local approximate of the Jacobian at $\mathbf{x}_0$ obtained by performing quadrilinear interpolation of the estimates of the Jacobians at the grid vertices. A new parameter value $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{d}_0$ is computed, and the corresponding point in physical space is determined.

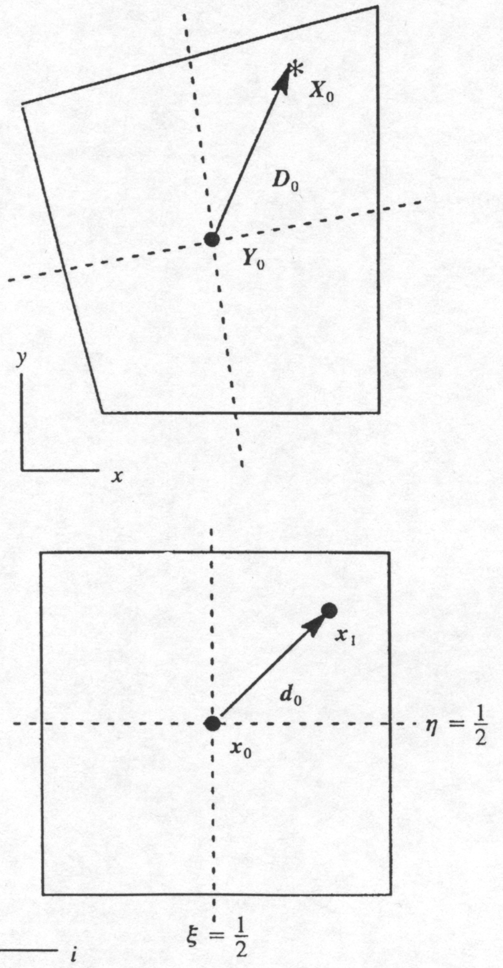


Fig. 2. Transforming from physical space to (local) parameter space and vice versa.

This procedure (Newton-Raphson method) is repeated until the Euclidean distance between the points $\mathbf{X}_0$ and $\mathbf{Y}_i$ is sufficiently small and the (local) parameter tuple associated with of $\mathbf{Y}_i$ lies in the unit cube. The procedure is also stopped when a parameter tuple has "moved" outside the unit cube (see [13]). Special care must be taken at the boundaries of the grid.

Obviously, the map from local parameter space to physical space must be one-to-one for this procedure to work (no degenerate cells allowed). Fig. 2 illustrates this procedure for the 2D case.

III. Local Approximation of the Unsteady Vector Field

From here on, all computations can be performed in uniform computational space. The unsteady vector field is approximated in (local) 4D parameter space using quadrilinear interpolation for each cell. The variables ξ, η, ζ, and τ denote the local parameters of a point in a cell in computational space. Assuming that both spatial dimensions and time dimension are spaced uniformly, the vector field $\mathbf{v} = (u, v, w)$ is approximated by

$$\mathbf{v}(\xi,\eta,\zeta,\tau) = \big(u(\xi,\eta,\varsigma,\tau),\, v(\xi,\eta,\varsigma,\tau),\, w(\xi,\eta,\varsigma,\tau)\big) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1}\sum_{l=0}^{1} \mathbf{v}_{i,j,k,l} B_i^1(\xi) B_j^1(\eta) B_k^1(\varsigma) B_l^1(\tau), \quad \xi,\eta,\varsigma,\tau, \in [0,1], \tag{3.1}$$

where $\mathbf{v}_{i,j,k,l}$ is the vector at a vertex of a 4D grid cell in computational space. Fig. 3 shows trilinear interpolation for the 3D steady case.

Higher-order approximates of the vector field could be used as well, e.g., tensor product cubic interpolation (see [3] and [6] for higher-order schemes). Unfortunately, this increases the number of coefficients of the approximate drastically.

IV. Local Approximation of the Particle Path

In order to reduce the particle path generation problem to a trivariate approximation problem, linear interpolationis performed in the time dimension first. For each point $\mathbf{x}_I = (\xi_I, \eta_I, \zeta_I, \tau_I)$ in computational space, linear interpolation is performed in the time dimension for $\tau = \tau_I$ yielding a particular "time slice," a 3D cube and vector values at its eight vertices. The vectors at the vertices of a 4D cube are linearly interpolated yielding the vectors at the vertices of the 3D cube for $\tau = \tau_I$.

The particle path in physical space will be approximated by a point sequence $\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \ldots$. This is done by first computing a point sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ in computational space and then mapping each point $\mathbf{x}_I$ to physical space. In the following, all computations are performed in computational space using local parameters ξ, η, and ζ. Denoting the last point computed by $\mathbf{x}_I$,

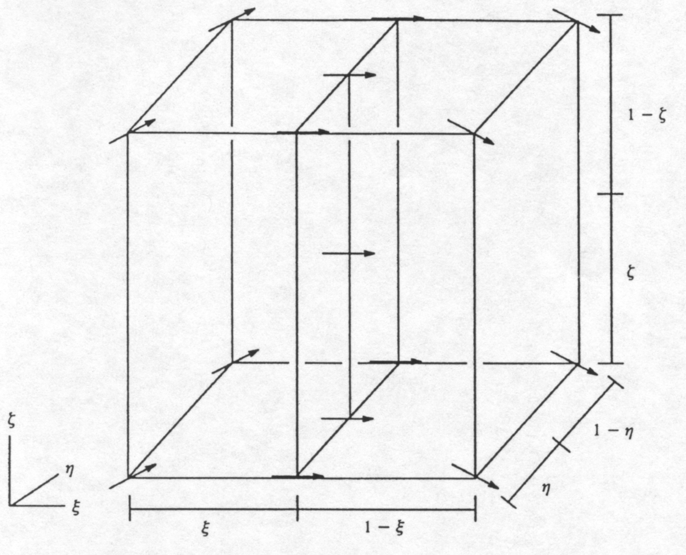


Fig. 3. Trilinear interpolation of 3D steady vector field in computational space.

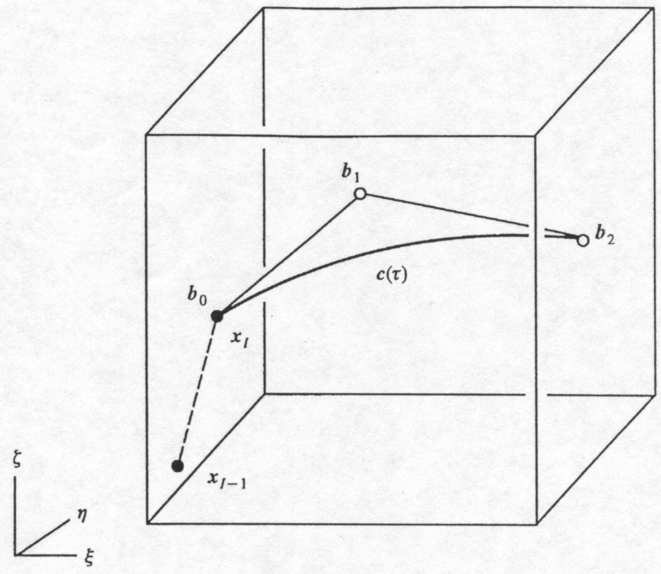


Fig. 4. Local approximation of particle path using quadratic Bézier curve.

its position and the velocity and acceleration at this point are used to compute a local quadratic particle path approximate.

Only the first three (i.e., spatial) coordinates are considered for the local path approximate. Since the velocity $\mathbf{v}_I$ at $\mathbf{x}_I$ is obtained by quadrilinear interpolation (3.1), the acceleration $\mathbf{a}_I$ at $\mathbf{x}_I$ is independent of the time dimension of the 4D cell. Thus, the acceleration at $\mathbf{x}_I = (\xi_I, \eta_I, \zeta_I)$ for $\tau = \tau_I$ is given by

$$\begin{aligned} \mathbf{a}_I &= \mathbf{a}(\xi_I, \eta_I, \varsigma_I) \\ &= \frac{\partial}{\partial \tau} \mathbf{v}(\xi, \eta, \varsigma, \tau)\Big|_{(\xi_I, \eta_I, \varsigma_I, \tau_I)} \\ &= \sum_{i=0}^{1} \sum_{j=0}^{1} \sum_{k=0}^{1} \left(\mathbf{v}_{i,j,k,l} - \mathbf{v}_{i,j,k,0}\right) B_i^1(\xi_I)\, B_j^1(\eta_I)\, B_k^1(\varsigma_I), \\ &\qquad \xi_I, \eta_I, \varsigma_I, \in [0,1], \end{aligned} \tag{4.1}$$

where $\mathbf{v}_{i,j,k,l}$, $i, j, k, l \in \{0, 1\}$, are the vectors at the vertices of the 4D cell containing $\mathbf{x}_I$.

For reasons that will become obvious in Section V, the quadratic curve defined by $\mathbf{x}_I$, $\mathbf{v}_I$, and $\mathbf{a}_I$ is represented as a Bézier curve, i.e.,

$$\mathbf{c}(\tau) = \sum_{i=0}^{2} \mathbf{b}_i\, B_i^2(\tau), \quad \tau \in [0, T], \tag{4.2}$$

where T is the time step used to compute the next point $\mathbf{x}_{I+1}$, and $B_i^2(\tau) = \frac{1}{T^2}\binom{2}{i}(T-\tau)^{2-i}\tau^i$. The Bézier control points are defined by the conditions $\mathbf{c}(0) = \mathbf{x}_I$, $\dot{\mathbf{c}}(0) = \mathbf{v}_I$, and $\ddot{\mathbf{c}}(0) = \mathbf{a}_I$. They are given by

$$\begin{aligned} \mathbf{b}_0 &= \mathbf{x}_I, \\ \mathbf{b}_1 &= \mathbf{x}_I + \frac{T}{2}\mathbf{v}_I, \end{aligned} \tag{4.3}$$

and

$$\mathbf{b}_2 = \mathbf{x}_I, + T\mathbf{v}_I + \frac{T^2}{2}\mathbf{a}_I,$$

where a Bézier control point is given by $\mathbf{b}_i = (\alpha_i, \beta_i, \gamma_i)$ (see [3]).

The time step T is chosen adaptively depending on the degree of curvature of the particle path (see Section VI). Fig. 4 illustrates the quadratic Bézier curve approximating the particle path at $\mathbf{x}_I$..

REMARK 4.1. Bernstein-Bézier polynomials are used for the representation of the local particle path approximate due to the existence of explicit formulas for products and integrals for these polynomials. This fact allows a more elegant computation for the next particle position (Section V). The method presented here is an alternative to the conventional Runge-Kutta methods using monomial basis functions. The method discussed here and standard Runge-Kutta methods allow the use of adaptive step sizes. Thus, one can generate equally good approximations of particle paths.

V. Computation of the Next Particle Position

In general, the next position $\mathbf{x}_{I+1}$ in computational space is given by

$$\mathbf{x}_{I+1} = \mathbf{x}_I + \int_0^T \mathbf{v}\left(\mathbf{c}(\tilde{\tau})\right) d\tilde{\tau}, \tag{5.1}$$

where $\tilde{\tau} = 0$ is associated with the previous point $\mathbf{x}_I$ (local time parameter). In coordinate form, this expression is equivalent to the three equations

$$\begin{aligned} (\xi_{I+1}, \eta_{I+1}, \varsigma_{I+1}) &= (\xi_I, \eta_I, \varsigma_I) \\ &+ \left(\int_0^T u\left(\mathbf{c}(\tilde{\tau})\right) d\tilde{\tau}, \int_0^T v\left(\mathbf{c}(\tilde{\tau})\right) d\tilde{\tau}, \int_0^T w\left(\mathbf{c}(\tilde{\tau})\right) d\tilde{\tau}\right). \end{aligned} \tag{5.2}$$

Therefore, the discussion is reduced to the computation of ξ_{I+1} in the following paragraphs.

Using trilinear interpolation of the vectors

$$\mathbf{v}_{i,j,k} = (u_{i,j,k}, v_{i,j,k}, w_{i,j,k}),\ i, j, k \in \{0, 1\},$$

known at the vertices of the current cell, and inserting the local quadratic path approximate $\mathbf{c}(\tau)$ given in (4.2) into (5.2) yields

$$
\begin{aligned}
\xi_{I+1} &= \xi_I + \int_0^T u\big(\mathbf{c}(\tilde{\tau})\big)\, d\tilde{\tau} \\
&= \xi_I + \int_0^T u\left(\sum_{l=0}^{2} \alpha_l B_l^2(\tilde{\tau}), \sum_{l=0}^{2} \beta_l B_l^2(\tilde{\tau}), \sum_{l=0}^{2} \gamma_l B_l^2(\tilde{\tau})\right) d\tilde{\tau} \\
&= \xi_I + \int_0^T \sum_{i=0}^{1} \sum_{j=0}^{1} \sum_{k=0}^{1} u_{i,j,k}\, B_i^1\left(\sum_{l=0}^{2} \alpha_l B_l^2(\tilde{\tau})\right) \\
&\quad B_j^1\left(\sum_{l=0}^{2} \beta_l B_l^2(\tilde{\tau})\right) B_k^1\left(\sum_{l=0}^{2} \gamma_l B_l^2(\tilde{\tau})\right) d\tilde{\tau} \\
&= \xi_I + \left(u_{0,0,0}\right)\int_0^T d\tilde{\tau} + \left(u_{1,0,0} - u_{0,0,0}\right)\int_0^T \xi(\tilde{\tau})\, d\tilde{\tau} \\
&\quad + \left(u_{0,1,0} - u_{0,0,0}\right)\int_0^T \eta(\tilde{\tau})\, d\tilde{\tau} + \left(u_{0,0,1} - u_{0,0,0}\right)\int_0^T \varsigma(\tilde{\tau})\, d\tilde{\tau} \\
&\quad + \left(u_{0,0,0} - u_{1,0,0} - u_{0,1,0} + u_{1,1,0}\right)\int_0^T \xi(\tilde{\tau})\, \eta(\tilde{\tau})\, d\tilde{\tau} \\
&\quad + \left(u_{0,0,0} - u_{1,0,0} - u_{0,0,1} + u_{1,0,1}\right)\int_0^T \xi(\tilde{\tau})\, \varsigma(\tilde{\tau})\, d\tilde{\tau} \\
&\quad + \left(u_{0,0,0} - u_{0,1,0} - u_{0,0,1} + u_{0,1,1}\right)\int_0^T \eta(\tilde{\tau})\, \varsigma(\tilde{\tau})\, d\tilde{\tau} \\
&\quad + \big(-u_{0,0,0} + u_{1,0,0} + u_{0,1,0} + u_{0,0,1} \\
&\quad - u_{1,1,0} - u_{1,0,1} - u_{0,1,1} + u_{1,1,1}\big)\int_0^T \xi(\tilde{\tau})\, \eta(\tilde{\tau})\, \varsigma(\tilde{\tau})\, d\tilde{\tau}.
\end{aligned}
$$

Three integral types of products of Bernstein-Bézier polynomials appear in (5.3). These integral types are given by

$$
\int_0^T \xi(\tilde{\tau})\, d\tilde{\tau} = \int_0^T \left(\sum_{i=0}^{2} \alpha_i B_i^2(\tilde{\tau})\right) d\tilde{\tau} = \frac{T}{3}\sum_{i=0}^{2} \alpha_i, \qquad (5.4)
$$

$$
\begin{aligned}
\int_0^T \xi(\tilde{\tau})\, \eta(\tilde{\tau}) d\tilde{\tau} &= \int_0^T \left(\sum_{i=0}^{2} \alpha_i B_i^2(\tilde{\tau})\right)\left(\sum_{j=0}^{2} \beta_j B_j^2(\tilde{\tau})\right) d\tilde{\tau} \\
&= \int_0^T \left(\sum_{i=0}^{2} \sum_{j=0}^{2} \alpha_i \beta_j \frac{\binom{2}{i}\binom{2}{j}}{\binom{4}{i+j}} B_{i+j}^4(\tilde{\tau})\right) d\tilde{\tau} \qquad (5.5) \\
&= \frac{T}{5}\sum_{i=0}^{2} \sum_{j=0}^{2} \alpha_i \beta_j \frac{\binom{2}{i}\binom{2}{i}}{\binom{4}{i+j}},
\end{aligned}
$$

and

$$
\begin{aligned}
&\int_0^T \xi(\tilde{\tau})\eta(\tilde{\tau})\varsigma(\tilde{\tau}) d\tilde{\tau} = \\
&\int_0^T \left(\sum_{i=0}^{2} \alpha_i B_i^2(\tilde{\tau})\right)\left(\sum_{j=0}^{2} \beta_j B_j^2(\tilde{\tau})\right)\left(\sum_{k=0}^{2} \gamma_k B_k^2(\tilde{\tau})\right) d\tilde{\tau} \qquad (5.6) \\
&= \int_0^T \left(\sum_{i=0}^{2}\sum_{j=0}^{2}\sum_{k=0}^{2} \alpha_i \beta_j \gamma_k \frac{\binom{2}{i}\binom{2}{j}\binom{2}{k}}{\binom{6}{i+j+k}} B_{i+j+k}^6(\tilde{\tau})\right) d\tilde{\tau} \\
&= \frac{T}{7}\sum_{i=0}^{2} \sum_{j=0}^{2}\sum_{j=0}^{2} \alpha_i \beta_j \gamma_k \frac{\binom{2}{i}\binom{2}{j}\binom{2}{k}}{\binom{6}{i+j+k}}.
\end{aligned}
$$

These integral expressions can be verified by using the equation

$$
\int_a^b B_i^n(x)\, dx = \frac{b-a}{n+1}, \quad x \in [a, b], \qquad (5.7)
$$

for Bernstein-Bézier polynomials defined over the general interval $[a, b]$ (see [3]) and

$$
\left(\sum_{i=0}^{m} b_i B_i^m(x)\right)\left(\sum_{j=0}^{n} \tilde{b}_j B_j^n(x)\right) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_i \tilde{b}_j \frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}} B_{i+j}^{m+n}(x) (5.8)
$$

(see [15]).

The point $\mathbf{x}_{I+1}$ is then transformed into physical space, and a one-step method is used to decide whether the time step T should be changed adaptively (see Section VI). Since the computational space is uniformly spaced, the cell containing the point $\mathbf{x}_{I+1}$ is immediately known from the point's coordinates.

REMARK 5.1. Formulas (5.4), (5.5), and (5.6) are not verified here since this is done in [15], a technical report. This report lists and verifies various properties of sums, products, and quotients of Bernstein-Bézier polynomials.

VI. CHOOSING THE TIME STEP T

The next point $\mathbf{X}_{I+1}$ (in physical space) might locally deviate from the particle path more than some maximal tolerance. The local deviation is measured in the following way: First, the point $\tilde{\mathbf{X}}_{I+1}$ is computed using $\mathbf{X}_I$ as previous point and $T/2$ as time step. Second, the point $\hat{\mathbf{X}}_{I+2}$ is computed using $\tilde{\mathbf{X}}_{I+1}$ as previous point and $T/2$ as time step. If the Euclidean distance d between $\mathbf{X}_{I+1}$ and $\hat{\mathbf{X}}_{I+2}$ is too large, i.e, larger than a specified maximal error tolerance, one replaces T by $T/2$ and computes a new $\mathbf{X}_{I+1}$. This process is iterated until the Euclidean distance d between the point pair $\mathbf{X}_{I+1}$ and $\hat{\mathbf{X}}_{I+2}$ is smaller than the specified maximal tolerance. Fig. 5 illustrates the one-step method for ordinary differential equations (see [1]).

The initial time step T used for the computation of a next point $\mathbf{x}_{I+1}$ in computational space is $T = 1/(2\|\mathbf{v}_I\|)$, where $\|\ \|$ indicates the Euclidean norm. It could also be defined in terms of a local curvature measure at $\mathbf{x}_I$ and the Jacobian. If the next point $\mathbf{X}_{I+1}$ in physical space does not lie in the same cell as $\mathbf{X}_I$, it is ensured that it lies in a cell that shares at least one vertex with the cell containing $\mathbf{X}_I$.

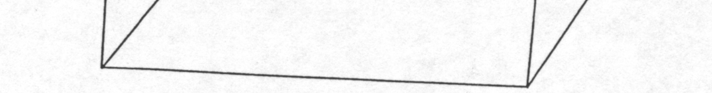


Fig. 5. One-step method for adaptively choosing time step.

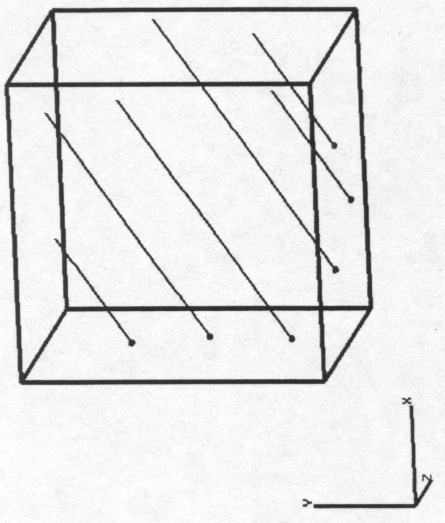


Fig. 6. Particle paths in constant 3D vector field (uniform, rectilinear grid, resolution $5 \times 5 \times 5 \times 5$; $\mathbf{V}(x, y, z) = (1, 1, 1)$, $x, y, z \in [0, 1]$).

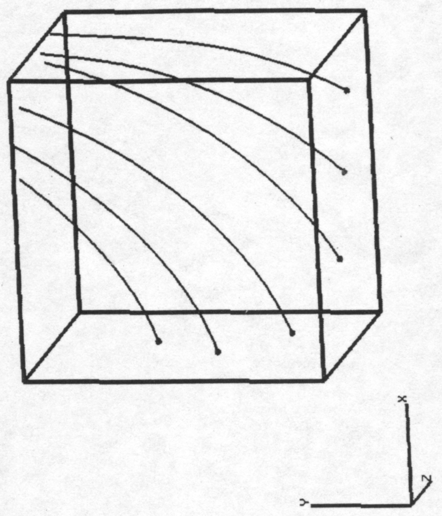


Fig. 7. Particle paths in steady 3D vector field (uniform, rectilinear grid, resolution $10 \times 10 \times 10 \times 10$; $\mathbf{V}(x, y, z) = \left(\sqrt{1-x}, \sqrt{(x)}, 0\right)$, $x, y, z \in [0, 1]$).

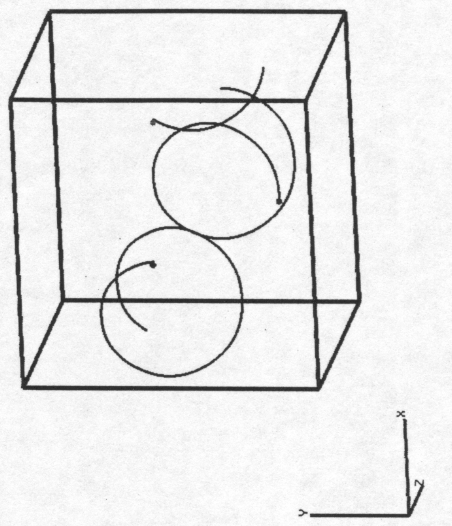


Fig. 8. Particle paths in unsteady 3D vector field (uniform, rectilinear grid, resolution $25 \times 25 \times 25 \times 25$; $\mathbf{V}(x, y, z, t) = (\cos(8\pi t), \sin(8pt), 0.25)$, $x, y, z, t \in [0, 1]$).

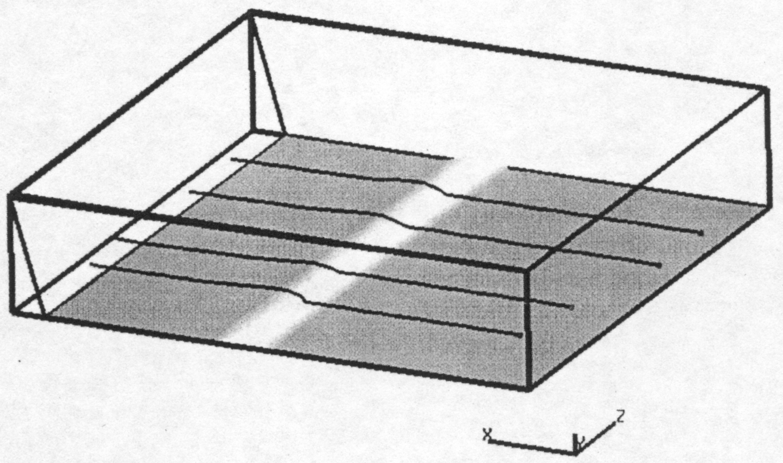


Fig. 9. Particle paths in steady 3D vector field—flow around aircraft wing (curvilinear grid, resolution $129 \times 33 \times 33$).

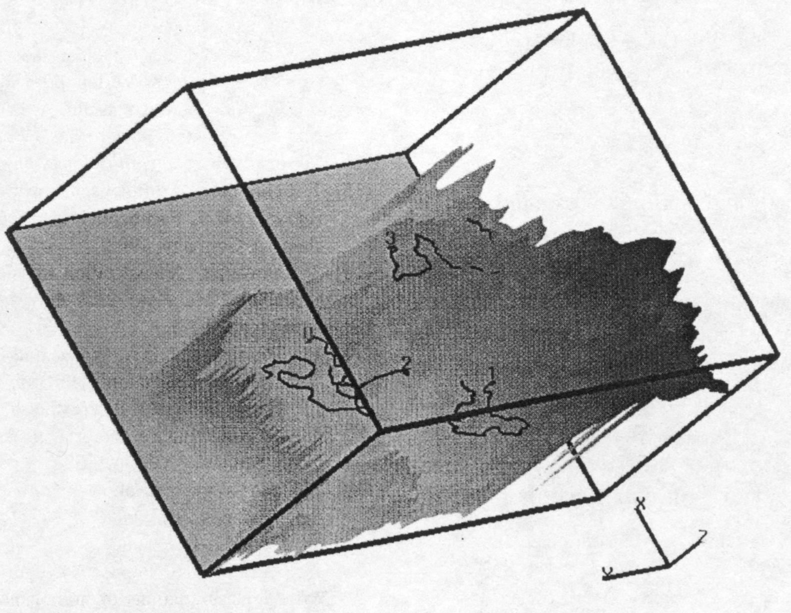


Fig. 10. Particle paths in unsteady 3D vector field—flow in Pacific Ocean (curvilinear grid, resolution $989 \times 657 \times 6 \times 5$).

VII. Examples

The method presented has been tested for analytically defined vector fields (for test purposes) and for steady/unsteady vector fields resulting from simulations of real-world configurations. The following examples are based on grids that do not change over time, i.e., the connectivity among grid vertices and the coordinates of grid vertices remain unchanged. In each figure, coordinate axes are placed closest to the vertex with index triple [0, 0, 0]. Several particle paths are shown in each vector field. The initial positions are marked by "bullets."

In Figs. 6, 7, and 8, the vector fields are defined analytically over the unit cube. Fig. 9 shows particle paths in the steady flow field around an aircraft wing. The wing is highlighted. For this particular example, the resulting particle paths have been "verified" by comparing them with the ones resulting from a standard fourth-order Runge-Kutta method, using the same initial positions, adaptive time steps, and an extremely small error tolerance. Fig. 10 shows particle paths in a portion of an unsteady flow field The shaded surface represents the bathymetry in the ocean.

REMARK 7.1. In addition, a fourth-order Runge-Kutta algorithm has been applied to the examples shown in Figs. 6, 7, 8, 9, and 10 using the same initial positions and the same maximal error tolerance. The resulting particle paths consist of nearly the same number of points. A general statement regarding the closeness of the particle paths generated by the new method and the ones generated by the fourth-order Runge-Kutta method cannot be made; the complexity of the flow field and the chosen maximal error tolerance have a major effect. Since the proposed method uses both an adaptive step size and a maximal error tolerance, an arbitrary accuracy of the particle path approximation can be achieved.

REMARK 7.2. The current implementation of the proposed method is slightly slower than most Runge-Kutta implementations applied to the examples shown in Figs. 6, 7, 8, 9, and 10. It is reasonable to assume that one can improve the efficiency of the current implementation of the proposed method by a factor of two to three by refining the spatial data structures and the search algorithms for the 3D/4D grids.

VIII. Conclusions

The particle path generation technique presented can be generalized to higher–order local approximations of both the vector field and the particle path. It is planned to extend the approach by using local cubic approximates of the vector field and considering a higher-order local path approximate. The use of Bernstein-Bézier polynomials for the local path approximation yields expressions that can be integrated exactly and easily.

The technique can be used to create stream lines, streak lines, and time lines (see [13]) for unsteady vector fields defined on curvilinear, structured grids.

Acknowledgments

This work was supported by the National Science Foundation under contract ASC–9210439 (Research Initiation Award) and contract EEC–8907070 (NSF Engineering Research Center for Computational Field Simulation) to Mississippi State University. Special thanks go to the members of the Scientific Visualization Thrust at the NSF Engineering Research Center for Computational Field Simulation at Mississippi State University.

References

[1] W. Boehm and H. Prautzsch, *Numerical Methods*. Wellesley, Mass.: AK Peters, Ltd., 1994.

[2] D. Darmofal and R. Haimes, "Visualization of 3-D vector fields: Variations on a stream," AIAA paper 92–0074, *Proc. 30th Aerospace Sciences Meeting and Exhibit*, Reno, Nev., 1992.

[3] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*. 3rd ed., San Diego, Calif.: Academic Press, 1993.

[4] H. Hagen, H., Müller, and G.M. Nielson, *Focus on Scientific Visualization*. New York: Springer-Verlag, 1993.

[5] A.J.S. Hin and F.H. Post, "Visualization of turbulent flow with particles," G.M. Nielson and D. Bergeron, eds., *Visualization '93*, Los Alamitos, Calif.: IEEE CS Press, pp. 46–51, 1993.

[6] J. Hoschek and D. Lasser, *Fundamentals of Computer Aided Geometric Design*. Wellesley, Mass.: AK Peters, Ltd., 1993.

[7] J.P.M. Hultquist, "Interactive numerical flow visualization using stream surfaces," PhD dissertation, Univ. of North Carolina, Chapel Hill, N.C., 1995.

[8] J.P.M. Hultquist, "Constructing stream surfaces in steady 3D vector fields," A.E. Kaufman and G.M. Nielson, eds., *Visualization '92*, Los Alamitos, Calif.: IEEE CS Press, pp. 171–178, 1992.

[9] D.N. Kenwright and G.D. Mallinson, "A 3-D streamline tracking algorithm using dual stream functions," A.E. Kaufman and G.M. Nielson, eds., *Visualization '92*, Los Alamitos, Calif.: IEEE CS Press, pp. 62–68, 1992.

[10] D.A. Lane, "Visualization of time-dependent flow fields," G.M. Nielson and D. Bergeron, eds., *Visualization '93*, Los Alamitos, Calif.: IEEE CS Press, pp. 32–38, 1993.

[11] W.C. de Leeuw and J.J. van Wijk, "A probe for local flow field visualization," G.M. Nielson and D. Bergeron, eds., *Visualization '93*, Los Alamitos, Calif.: IEEE CS Press, pp. 39–45, 1993.

[12] K.L. Ma and P.J. Smith, "Cloud tracing in convection–diffusion systems," G.M. Nielson and D. Bergeron, eds., *Visualization '93*, Los Alamitos, Calif.: IEEE CS Press, pp. 253–259, 1993.

[13] F.H. Post and T. van Walsum, "Fluid flow visualization," H. Hagen, H. Müller, and G.M. Nielson, eds., *Focus on Scientific Visualization*. New York: Springer-Verlag, pp. 1–40, 1993.

[14] A. Sadarjoen, T. van Walsum, A. Hin, and F. Post, "Particle tracing algorithms for 3-D curvilinear grids," *Proc. Fifth Eurographics Workshop on Visualization in Scientific Computing*, Rostock, Germany, 1994.

[15] T. Schreiber, "Arithmetische operationen auf bézierflächen," Internal Report 224/92, Fachbereich Informatik, Technische Universität Kaiserslautern, Germany, 1992.

[16] S. Shirayama, "Visualization of vector fields in flow analysis," AIAA paper 91–0801, *Proc. 29th Aerospace Sciences Meeting and Exhibit*, Reno, Nev., 1991.

[17] J.F. Thompson, Z.U.A. Warsi, and C.W. Mastin, *Numerical Grid Generation*. New York: North-Holland, 1985.

[18] J.F. Thompson and N.P. Weatherill, "Aspects of numerical grid generation: Current science and art," *Proc. 11th AIAA Applied Aerodynamics Conf.*, Monterey, Calif., 1993.

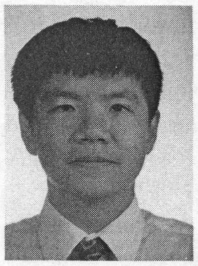
[19] J.J. van Wijk, "Implicit stream surfaces," G.M. Nielson and D. Bergeron, eds., *Visualization '93*, Los Alamitos, Calif.: IEEE CS Press, pp. 245–252, 1993.

[20] P. Yeung, and S. Pope, "An algorithm for tracking fluid particles in numerical simulations of homogeneous turbulence," *J. Computational Physics* 79, 1988.

Bernd Hamann is an associate professor in the Department of Computer Science at the University of California at Davis. Previously, he was an associate professor in the Department of Computer Science and a research faculty member at the NSF Engineering Research Center for Computational Field Simulation at Mississippi State University. His current research and teaching interests are scientific visualization, computer graphics, and computer-aided geometric design (CAGD).

Hamann received a BS in computer science, a BS in mathematics, and an MS in computer science from the Technical University of Braunschweig, Germany. He received his PhD in computer science from Arizona State University in 1991. Hamann was awarded a 1992 Research Initiation Award by Mississippi State University and a 1992 Research Initiation Award by the National Science Foundation. He was selected as one of two nominees from Mississippi State University for a 1995 Presidential Faculty Fellows (PFF) Award of the National Science Foundation and was awarded a Hearin-Hess Distinguished Professorship in Engineering in 1995 by the College of Engineering, Mississippi State University.

Hamann is a member of the ACM, the IEEE, and the SIAM.



Donghua Wu is a PhD student in computer engineering at Mississippi State University. His current research interests are computer graphics and scientific visualization.

Wu received a BS in computer science from Shanghai Jiao-Tong University, Shanghai, China, in 1987. He received an MS in computer science from Mississippi State University in 1992.

Robert J. Moorhead II is an associate professor in the Department of Electrical and Computer Engineering and team leader for scientific visualization at the NSF Engineering Research Center for Computational Field Simulation at Mississippi State University. From 1985 to 1988, he was a research staff member at the IBM T.J. Watson Research Center. His current research and teaching interests are multiresolutional visual analysis, vortex and flow visualization, and visualization of time-varying phenomena. Moorhead received a BS in electrical engineering in 1980 and an MS in electrical engineering in 1982, both from Geneva College, and a PhD in electrical and computer engineering from North Carolina State University in 1985. Moorhead has been author and coauthor of over 30 publications and was awarded a Hearin-Hess Distinguished Professorship in Engineering in 1994 and 1995 by the College of Engineering, Mississippi State University.