GRAPHICAL MODELS AND IMAGE PROCESSING
Vol. 58, No. 3, May, pp. 262–285, 1996
ARTICLE NO. 0022

A New Approach to Through-the-Lens Camera Control[1]

MIN-HO KYUNG, MYUNG-SOO KIM, AND SUNG JE HONG

Department of Computer Science, POSTECH, Pohang 790-784, South Korea

Received June 23, 1995; revised December 26, 1995; accepted January 4, 1996

The through-the-lens camera control technique originally proposed by M. Gleicher and A. Witkin [*Comput. Graphics* 26(2), 1992, 331–340], provides a powerful user interface for the control of the virtual camera in 3D computer graphics and animation. Their techique is based on locally inverting the nonlinear perspective viewing transformation. However, given m image control points, the Jacobian matrix is derived as a quite complex $2m \times 8$ matrix; furthermore, the Jacobian matrix always has at least one redundant column since its rank can be 7 at most. For the overconstrained case of $m \geq 4$, the Lagrange equation is always singular since its $2m \times 2m$ square matrix has rank 7 at most. All these complications result from removing the constraint $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$ for unit quaternions $(q_w, q_x, q_y, q_z) \in S^3$ which represent the camera rotations. In this paper, we interpret the problem as a target tracking problem and formulate it as a constrianed nonlinear inversion problem. The problem is then solved by integrating a tangent vector field defined on the configuration space of the virtual camera. The vector field is given by the least-squares solutions of the Jacobian matrix equations. The row and column weighting scheme for the Jacobian matrix provides a convenient way to control the desired least-squares solutions and the associated vector field. The Lie group structure of the unit quaternion space S^3 enables us to derive a simple $2m \times 7$ Jacobian matrix, which improves both the computational efficiency and numerical stability of the overall algorithm. For the overconstrained case of $m \geq 4$, the Jacobian matrix equation is solved (in the least-squares sense) by using an efficient projection method with $O(m)$ time complexity.

1. INTRODUCTION

In computer graphics, virtual camera models are used to specify how a 3D scene is to be viewed on the display screen. For example, the 3D viewing parameters look-at/look-form/view-up represent one of the most popular virtual camera models [10]. When the camera model is poor, the user may experience much difficulty in describing the exact viewing of a 3D scene in mind. The situation is even worse for computer animation, in which a continuous camera motion is often required to generate a smooth scene change. With a poor camera model, the computer animator has to spend considerable time in fully describing natural scene changes in the animation movie. The virtual camera model thus plays an important role in 3D computer graphics and animation.

The camera status is usually described by seven parameters: one for focus, three for position, and three for orientation. In some specialized systems such as RenderMan [27], other parameters are also used to control the optical properties of the camera (e.g., the depth of field, shutter speed, atmospheric effects) and the image properties (e.g., the pixel aspect ratio). In this paper, we assume a simple camera model which allows the user to control the virtual camera with seven degrees of freedom. However, even with such a simple camera model, it is not easy to control all seven camera parameters simultaneously. This is because most user interfaces (e.g., mouse) do not support the control of all the required seven parameters at the same time. For some camera effects (requiring certain coordinated control of all camera parameters), it is not easy to generate smooth camera motion that produces scene changes which appear natural to the human eye. For example, when the relative speeds of the translation and rotation of a camera are not synchronized, the animated scene shakes.

Virtual cameras are usually described by specialized viewing transformations [10]. Using the geometric relationship between the world space and the view space, Blinn [3] developed a camera control technique that automatically computes the camera focus, position, and orientation from a specification of two object placements in the viewing plane. By taking one object as the spacecraft in the foreground and the other object as a planet or moon in the background, this technique was effectively used in making space movies [3]. However, this technique, based on vector algebra, allows only a few restricted input specifications (for example, the calculation of the camera direction from the inputs of a view-up vector and an object position on the screen); its usage as a general virtual camera control scheme is therefore quite limited.

[1] This research was partially supported by the Korean Ministry of Science and Technology under Contract 94-S-05-A-03-A of STEP 2000. A preliminary version of this paper appeared in *Proc. of Graphics Interface '95*, pp. 171–178.

1077-3169/96 $18.00

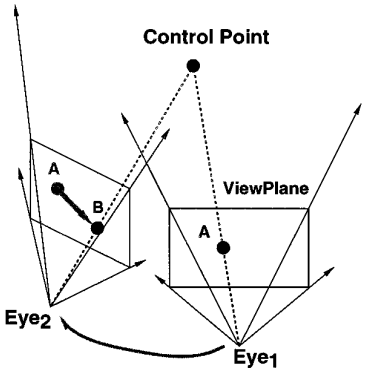

FIG. 1. Through-the-lens camera control.

Gleicher and Witkin [12] suggested the *through-the-lens camera control* scheme to provide a general solution to the virtual camera control problem in computer animation. Instead of controlling the camera parameters directly, the 2D points on the display screen are controlled by the user. The required changes of camera parameters are automatically generated so that the picked screen points are moved on the screen as the user has specified. That is, when the user picks some 2D points and moves them into new positions, all the camera parameters are automatically changed so that the corresponding 3D data points are projected onto the new 2D points. In Fig. 1, the virtual camera is in the position Eye1 and the given 3D point is projected onto the 2D point A in the viewing plane. When the user moves the projected point A into a new position at B, the camera parameters are automatically changed so that the given 3D point is now projected onto the new position B with the new virtual camera at the position Eye2. Through-the-lens camera control is a useful tool, especially for computer animators who have little knowledge about the mathematical model of the virtual camera. There is actually no need to know how each camera parameter changes the viewing of a scene and which camera parameters are the right ones to get the desired camera effect. Through-the-lens camera control thus provides a very powerful user interface to virtual camera control in image composition and computer animation.

Through-the-lens camera control can be formulated as a constrained nonlinear inversion problem. Let $\mathbf{x}$ denote the camera parameter vector $(f, u_x, u_y, u_z, q_w, q_x, q_y, q_z) \in R^4 \times S^3$, where $f \in R$ is the focal length, $(u_x, u_y, u_z) \in R^3$ is the camera position, and $(q_w, q_x, q_y, q_z) \in S^3$ is the unit quaternion for the camera rotation [16, 23]. In representing the orientations, unit quaternions are quite useful since they are free of signularities such as gimbal lock (see [16, 23, 28] and Section 8.1). Thus, instead of three parameters, the unit quaternion (i.e., four parameters with one constraint) is used to represent the camera rotation. Given m points $p_1, \ldots, p_m \in R^3$, the perspective viewing transformation for the m points, V_P: $R^4 \times S^3 \to R^{2m}$, is defined by

$$V_P(\mathbf{x}) = \mathbf{h}, \tag{1}$$

where $P = (p_1, \ldots, p_m) \in R^{3m}$ and $\mathbf{h} = (x_1, y_1, \ldots, x_m, y_m) \in R^{2m}$ with each (x_i, y_i) being the perspective viewing transformation of p_i onto the 2D display screen. The perspective projection of V_P produces a nonlinear relationship between the camera parameters and the projected 2D image points because the equation of the perspective projection is a rational expression (see Section 3.4). The through-the-lens control problem concerns how to compute the camera parameter vector $\mathbf{x}$ in Eq. (1) for the given P and $\mathbf{h}$. To compute $\mathbf{x}$, a nonlinear inverse map, V_P^{-1}, should be constructed. However, due to the dimensional mismatch between $R^4 \times S^3$ and R^{2m} (which are 7 and $2m$, respectively), the nonlinear map V_P is not invertible even in any local neighborhood.

A simple way to compute the solution $\mathbf{x}$ is to approximate the nonlinear inversion problem by a sequence of linear inversion problems. That is, instead of solving $V_P(\mathbf{x}) = \mathbf{h}$ directly, we can solve a sequence of linear differential equations: $J(\mathbf{x})\dot{\mathbf{x}} = \dot{\mathbf{h}}$, where $J(\mathbf{x})$ is the Jacobian matrix of V_P at $\mathbf{x}$ and $\dot{\mathbf{h}} = \mathbf{h} - V_P(\mathbf{x})$. Even though the Jacobian matrix $J(\mathbf{x})$ is not invertible, the solution $\dot{\mathbf{x}} \in T_{\mathbf{x}}(R^4 \times S^3)$ can be approximated as

$$\dot{\mathbf{x}} = J(\mathbf{x})^+ \dot{\mathbf{h}} \in T_{\mathbf{x}}(R^4 \times S^3),$$

where $J(\mathbf{x})^+$ is the pseudo inverse of $J(\mathbf{x})$ and $T_{\mathbf{x}}(R^4 \times S^3)$ is the tangent space of $R^4 \times S^3$. Therefore, the solution $\dot{\mathbf{x}}$ at each camera configuration $\mathbf{x}$ generates a tangent vector field on the constraint space $R^4 \times S^3$. By integrating the vector field $\dot{\mathbf{x}}$, we can construct an integral curve $\mathbf{x}(t)$ which eventually converges to a local optimal solution $\mathbf{x}$ of $V_P(\mathbf{x}) = \mathbf{h}$, such that $\|\dot{\mathbf{x}}\| < \varepsilon$ at the solution, for some given tolerance $\varepsilon > 0$.

The general construction scheme outlined above can be applied to any constrained nonlinear mapping, F: $M \to N$, where M and N are complete Riemannian manifolds in which the geodesic between any two given points is defined [20, 25]. (Mainfold may be considered as a generalization of regular surface to higher dimensional regular hypersurface embedded in R^n [25].) Given a target point $q \in N$, we can define a tangent vector field on the domain M as follows (see Fig. 2): For any point $p \in M$, consider the shortest distance geodesic $\gamma(s) \in N$, $0 \le s \le 1$, such that $\gamma(0) = F(p)$ and $\gamma(1) = q$. The velocity vector $\gamma'(0) \in T_{F(p)}(N)$ defines a tangent vector $v_p \in T_p(M)$ as

$$v_p = (dF_p)^+(\gamma'(0)) \in T_p(M),$$

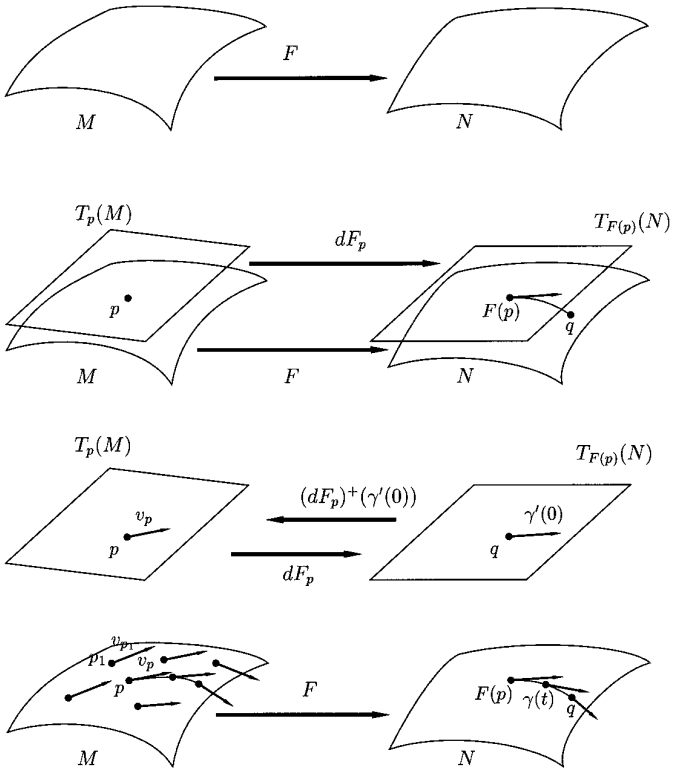

FIG. 2. General nonlinear inversion.

where $(dF_p)^+$ is the pseudo inverse of the linear differential, dF_p: $T_p(M) \to T_{F(p)}(N)$, where $T_p(M)$ and $T_{F(p)}(N)$ are the tangent spaces of M and N at p and $F(p)$, respectively. (Note that the Jacobian matrix is the matrix representation of a linear differential [8].) Given a tangent vector field $v_p \in T_p(M)$ defined at each $p \in M$, an integral curve $p(t) \in M$, $t \in R$, is defined to be a differentiable curve which satisfies the condition: $p'(t) = v_{p(t)} \in T_{p(t)}(M)$. When the target point is a moving curve $q(t) \in N$, we have a time-dependent vector field $v_p(t) \in T_p(M)$. The integral curve $p(t)$ then satisfies $p'(t) = v_{p(t)}(t) \in T_{p(t)}(M)$. The target tracking approach is popular in computer vision for the control of a real camera [21]; recently, it has also become a common motion control technique in computer animation [17].

The construction of an exact integral curve requires the solution of a nonlinear differential equation. An integral curve is usually approximated by tracing along the geodesic curve on M (starting from p in the direction of $v_p \in T_p(M)$) by an infinitesimal distance dt and repeating the same procedure at the new point. The exponential map, $\exp_p$: $T_p(M) \to M$, is defined to transform each tangent vector $v_p \in T_p(M)$ into a point $\hat{p} \in M$, where $\hat{p}$ is the point at a distance $\|v_p\|$ from p along the geodesic curve starting from p in the direction of v_p. Once we have the exponential map, the discrete integration of the vector field generates a new point, $\exp_p(v_p\, dt) \in M$, which is precisely contained in the constraint manifold M itself. This nice property of the exponential map eliminates the constraint deviation problem which is common in Euler's method.

In a general Riemannian manifold, the geodesic curve is given as a solution of a second order ordinary differential equation [20, 25]. However, in the simple cases of R^n and S^n, the construction of geodesic curves is quite straightforward; therefore, the construction of exponential maps is a relatively simple task. Most notably, the spheres S^1, S^3, S^7 from *Lie groups* with their complex, quaternion, and Clifford algebra products defined in their respective embedding spaces R^2, R^4, R^8, respectively. A Lie group is a smooth manifold in which a group operation is defined; furthermore, the group product and inverse operations are smooth mappings. In a Lie group G, the derivative of a curve $p(t) \in G$ is always represented by $p'(t) = v(t) \cdot p(t)$, where $v(t) \in T_1(G)$ is a tangent vector at the identity element $1 \in G$ and $\cdot$ is the Lie group operation [7]. The Lie group structures of S^1, S^3, S^7 greatly simplify the differential structures; that is, any tangent vector $p'(t) \in T_{p(t)}(G)$ can be identified with a tangent vector $v(t) \in T_1(G)$ ($\equiv R^1$, R^3, R^7, respectively), and vice versa (see Section 3.1). Therefore, any tangent vector $p'(t)$ can be represented by the canonical coordinate system given to the tangent space $T_1(G)$. Since the configurations of multilink body systems are usually described by the product spaces of R^n, S^1, S^3, the Lie group structures of S^1 and S^3 have great potential in the application of the above mentioned nonlinear inversion scheme to computer animation. The Lie group structure also simplifies the construction of an exponential map. For a Lie group G, thanks to the canonical identification of $T_1(G)$ and $T_p(G)$, we need to construct only one exponential map exp: $T_1(G) \to G$. All the other exponential maps $\exp_p$: $T_p(G) \to G$ are defined by $\exp_p(v_p) = \exp(v_p) \cdot p^{-1}) \cdot p \in M$, for $v_p \in T_p(M)$, where $\cdot$ is the Lie group operation. The canonical exponential map, exp, is defined by the following Taylor series (see [7, 16]):

$$\exp(v) = \sum_{n=0}^{\infty} \frac{1}{n!} v^n, \quad \text{for } v \in T_1(G).$$

Here v^n is the nth power of v in the Lie group operation.

Gleicher and Witkin [12] transformed the constraint space S^3 of unit quaternions into $R^4\backslash\{0\}$ by using a nonzero quaternion q to represent the rotation implied by the unit quaternion $q/\|q\|$. At the first glance, this technique may look as if it is simplifying the nonlinear inversion problem. Unfortunately, it does not work out so nicely. The radial quaternions tq, $t \in R$, represent the same rotation implied by the unit quaternion $q/\|q\|$. This produces a redundant column in the $2m \times 8$ Jacobian matrix J of the viewing transformation V_P: $R^8 \to R^{2m}$. The discrepancy between

the rectangular coordinate system of R^4 and the spherical coordinate system of S^3 results in a very complex formula for the Jacobian matrix J. Furthermore, Gleicher and Witkin [12] solved the linear inversion problem: $J(\mathbf{x})\dot{\mathbf{x}} = \dot{\mathbf{h}}$ by minimizing a quadratic objective function. The problem is consequently converted to a Lagrange equation, formulated as a $2m \times 2m$ square matrix equation. Since the Jacobian matrix J has rank 7 at most, the $2m \times 2m$ square matrix JJ^{T} also has rank 7 at most. For the overconstrained case of $m \geq 4$, the square matrix is always singular as will be seen in Section 2. A general $2m \times 2m$ square matrix equation takes $O(m^3)$ time to be solved.

Utilizing the special structure of JJ^{T}, Gleicher [11] computes $\dot{\mathbf{x}}^{\mathrm{T}}JJ^{\mathrm{TT}}\dot{\mathbf{y}} = (J^{\mathrm{T}}\dot{\mathbf{x}})^{\mathrm{T}}(J^{\mathrm{T}}\dot{\mathbf{y}})$ in $O(m)$ time and use the conjugate gradient method to solve the Lagrange matrix equation. The conjugate gradient method iterates a maximum of $2m$ times; in each iteration, the most expensive computation is the evaluation of $\dot{\mathbf{x}}^{\mathrm{T}}JJ^{\mathrm{T}}\dot{\mathbf{y}}$, which takes $O(m)$ time. Therefore, it may take $O(m^2)$ time to solve the matrix equation. To apply the conjugate gradient method, the matrix JJ^{T} is required to be a positive definite matrix; however, there is no such guarantee since the square matrix JJ^{T} is always singular for $m \geq 4$. Nevertheless, in practice, the conjugate gradient method usually works for positive semidefinite matrices. (It is easy to show that the matrix JJ^{T} is positive semidefinite since $\dot{\mathbf{x}}^{\mathrm{T}}JJ^{\mathrm{T}}\dot{\mathbf{x}} = (J^{\mathrm{T}}\dot{\mathbf{x}})^{\mathrm{T}}(J^{\mathrm{T}}\dot{\mathbf{x}}) \geq 0$, for all $\dot{\mathbf{x}}$.) Consequently, the time complexity $O(m^2)$ of Gleicher [11] may be acceptable in practice.

In this paper, the Lie group structure of the unit quaternion space S^3 enables us to derive a simple Jacobian matrix, which is computationally efficient and numerically stable. First of all, a $2m \times 7$ Jacobian matrix is derived using quaternion calculus [15, 16, 23] which provides an appropriate tool for the analysis of 3D rotations. This $2m \times 7$ Jacobian matrix is simpler in its algebraic expression than the previous Jacobian matrix [12], and thus the construction is speeded up. With one less column than the $2m \times 8$ Jacobian matrix, our Jacobian matrix J is less redundant for the case of $m \geq 4$. Furthermore, we use a weighted least-squares method coupled with the singular value decomposition (SVD) [13, 22, 26], which makes the overall computation numerically stable. For the overconstrained case of $m \geq 4$, we can use an efficient projection method to compute the least squares solution [26]. The time complexity grows only linearly, i.e., $O(m)$ time for m control points, which is more efficient than the time complexity $O(m^2)$ of Gleicher [11]. The row and column weighting scheme provides a way for the user to specify the relative importance of each control point and each camera parameter; it is more intuitive to specify weights to the control points and camera parameters than to assign quadratic optimization functions. It is also easy to generalize the row and column weighting scheme to the time varying weighting functions, which can be used effectively for computer animation with drmatic scene changes.

In a recent MIT Ph.D. thesis, Drucker [9] proposed a camera control technique which has more general applications than the through-the-lens control in computer animation environments. In this approach, a large variety of objective functions and geometric constraints can be specified, and the virtual camera is controlled so that the objective functions are minimized while satisfying given geometric constraints. The Sequential Quadratic Programming (SQP) technique [5] is used to compute $\dot{\mathbf{x}}$ by solving an $(m + 7) \times (m + 7)$ square matrix equation, where m is the number of constraints and 7 is the number of camera parameters. The matrix has terms which are given by the second order partial derivatives of the objective functions and constraint equations with respect to the camera parameters; therefore, the matrix equation is quite time-consuming to set up. Moreover, the matrix dimension $(m + 7)$ does not match with the dimension $(7 - m)$ of the constraint space; that is, the camera parameter space with m independent constraints is a $(7 - m)$-dimensional manifold in general. To improve the computational efficiency, we can reformulate the given optimization problem into a certain target tracking problem and apply our least-squares solution method to the resulting Jacobian matrix equation which can be represented by the first order partial derivatives only. We discuss more details of this approach in Section 8.

The rest of this paper is organized as follows. In Section 2, we review the previous method of Gleicher and Witkin [12] and discuss some of its shortcomings. Quaternion calculus is introduced in Section 3 to derive a simple Jacobian matrix for the perspective viewing transformation. In Sections 4 and 5, the linear differential matrix equations are derived for through-the-lens camera control and they are solved by using the weighted least-squares method. The implementation details and some experimental results are discussed in Section 6. In Section 7, we discuss the application of through-the-lens camera control to the keyframing control of the virtual camera. In Section 8, we describe how to extend the result of this paper to other camera models and to other camera and motion control techniques in general. Finally, Section 9 concludes the paper.

2. PREVIOUS WORK

2.1. *Review of Previous Work*

Gleicher and Witkin [12] solve the matrix equation: $J\dot{\mathbf{x}} = \dot{\mathbf{h}}$ as a quadratic optimization problem which minimizes the quadratic energy function

$$E = \frac{1}{2}\|\dot{\mathbf{x}} - \dot{\mathbf{x}}_0\|^2 \tag{2}$$

subject to the linear constraint

$$J\dot{\mathbf{x}} = \dot{\mathbf{h}}_0, \tag{3}$$

where $\dot{\mathbf{h}}_0 \in R^{2m}$ and $\dot{\mathbf{x}}_0 \in R^8$ are the initial velocity vectors. The value of $\dot{\mathbf{x}}$ minimizing the energy function E implies the minimal variation of the camera motion with respect to the given initial solution $\mathbf{x}_0$. The problem is converted into a Lagrange equation

$$dE/d\dot{\mathbf{x}} = \dot{\mathbf{x}} - \dot{\mathbf{x}}_0 = J^{\mathrm{T}}\lambda \tag{4}$$

for some value of the $2m$-vector λ of Lagrange multipliers. The Lagrange equation is then converted into

$$JJ^{\mathrm{T}}\lambda = \dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0, \tag{5}$$

which is solved for λ. Then the value of $\dot{\mathbf{x}}$ is obtained by

$$\dot{\mathbf{x}} = \dot{\mathbf{x}}_0 + J^{\mathrm{T}}\lambda,$$

and it is used to update the virtual camera parameters $\mathbf{x}$ as follows:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\dot{\mathbf{x}}(t). \tag{6}$$

When Eq. (3) has at least one solution $\dot{\mathbf{x}}_1 \in R^8$, the solution space of Eq. (3) is nonempty and generates a hyperplane $L_J \subset R^8$. Note that L_J contains $\dot{\mathbf{x}}_1 \in R^8$ and is orthogonal to each row vector of J (equivalently, to each column vector of J^{T}). The dimension of L_J is the same as $8 - r$, where r is the rank of J and $r \leq \min\{2m, 8\}$. There is a unique point $\dot{\mathbf{x}} \in L_J$ which is the closest to $\dot{\mathbf{x}}_0$. For the unique optimal solution $\dot{\mathbf{x}}$ of Eqs. (2)–(3), the solution space of λ in Eq. (4) generates a $(2m - r)$-dimensional hyperplane $L_{J^{\mathrm{T}}} \subset R^{2m}$. Each point λ of $L_{J^{\mathrm{T}}}$ is also realized as a solution of Eq. (5), and vice versa. (This is because the two matrices J^{T} and JJ^{T} have the same rank.) Therefore, any solution $\lambda \in R^{2m}$ of Eq. (5) provides the unique optimal solution $\dot{\mathbf{x}} = \dot{\mathbf{x}}_0 + J^{\mathrm{T}}\lambda \in R^8$ which minimizes the quadratic energy of Eq. (2).

When Eq. (3) has a solution, we show that Eq. (5) is always guaranteed to have a solution even if the $2m \times 2m$ square matrix JJ^{T} is singular. In the underconstrained case of $m \leq 3$, the square matrix JJ^{T} is singular if and only if the columns of J^{T} (equivalently, the rows of J) are linearly dependent. In the overconstrained case of $m \geq 4$, JJ^{T} is always singular. For a singular matrix JJ^{T}, Eq. (5) has a solution if and only if the $2m$ vector $\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0$ is contained in the column space of JJ^{T} (i.e., the subspace of R^{2m} which is spanned by the columns of JJ^{T}). Since we assume that there is at least one solution of Eq. (3), there is a unique optimal solution $\dot{\mathbf{x}}$ of Eq. (2). This optimal solution $\dot{\mathbf{x}}$ satisfies Eq. (4) for some λ, and the solution space of $\lambda \in R^{2m}$ in Eq. (4) is nonempty. Therefore, in Eq. (5), the vector $\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0$ is represented by a linear combination of the columns of JJ^{T} (with the components of the $2m$ vector λ as their coefficients), and thus it is guaranteed to be in the column space of JJ^T. (This is because the two solution spaces of λ in Eqs. (4) and (5) are exactly the same.)

2.2. *Limitation in the Overdetermined Case*

There is, however, no guarantee that Eq. (3) has a solution in general. When there is no solution of $\dot{\mathbf{x}} \in R^8$ in Eq. (3), we have to project the vector $\dot{\mathbf{h}}_0$ into the column space of J. This projection also results in a similar projection of $\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0$ into the column space of JJ^{T}, and vice versa. (Note that each column of JJ^{T} is a linear combination of the columns of J, and the two matrices J and JJ^{T} have the same rank. Consequently, J and JJ^{T} have the same column space embedded in R^{2m}.) Then, we can proceed to solve for λ in Eq. (5). To illustrate the computational issues involved in the solution process, we give a detailed discussion on the geometric constructions associated with the least squares solution of Eq. (5). (This discussion is mainly for the purpose of comparison with other solution methods. Our solution method does not try to solve the least-squares solution of Eq. (5).)

To compute the least-squares solution, we need to project the vector $\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0$ into the column space of JJ^{T}. An orthogonal projection minimizes the approximation error:

$$\|JJ^{\mathrm{T}}\lambda - (\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0)\|.$$

The singular value decomposition (SVD) of JJ^{T} provides a coordinate transformation of R^{2m}, in which the column space projection can be reduced to a trivial task of simply dropping the last $(2m - r)$ coordinates. Under another coordinate transformation of R^{2m}, the least-squares solution is similarly obtained by dropping the last $(2m - r)$ coordinates of a solution vector. We give more details below.

Assume the following singular value decomposition of JJ^{T}:

$$JJ^{\mathrm{T}} = UWV^{\mathrm{T}}.$$

Here U and V are two rotation matrices of R^{2m}, and $W_{ij} = 0$, for all i, j such that $i \neq j$ or $i = j > r$, where r is the rank of J. Eq. (5) now reduces to

$$\begin{aligned} UWV^{\mathrm{T}}\lambda &= \dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0, \quad \text{and} \\ W(V^{\mathrm{T}}\lambda) &= U^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0). \end{aligned} \tag{7}$$

Since the last $(2m - r)$ rows of W are zero vectors, the vector $U^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0)$ is in the column space of W if and only if its last $(2m - r)$ components are zeros. We can

truncate the last $(2m - r)$ components of $U^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0)$ by multiplying this vector by the $2m \times 2m$ matrix I_r such that $(I_r)_{ii} = 1$, for $1 \le i \le r$, and $(I_r)_{ij} = 0$, for all other i and j. The rotation by U of the resulting vector produces an orthogonal projection of $\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0$ into the column space of JJ^{T}; that is, the orthogonal projection is given by the following vector:

$$UI_rU^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0).$$

By truncating the last $(2m - r)$ components of the right-hand side of Eq. (7), we obtain the equation

$$W(V^{\mathrm{T}}\lambda) = I_rU^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0), \tag{8}$$

in which the solution of $V^{\mathrm{T}}\lambda$ may have arbitrary elements in the last $(2m - r)$ components of the solution vector (because the last $(2m - r)$ columns of W are zero column vectors). By filling these last $(2m - r)$ components of $V^{\mathrm{T}}\lambda$ with zeros, we can get the least-squares solution of $V^{\mathrm{T}}\lambda$. By concatenating all the intermediate steps in the construction sequence, we obtain the following least-squares solution λ:

$$\lambda = VW^{+}I_rU^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0) = VW^{+}U^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0).$$

Here the $2m \times 2m$ matrix W^{+} is defined by $(W^{+})_{ii} = (W_{ii})^{-1}$, for $1 \le i \le r$, and $(W^{+})_{ij} = 0$, for all other i and j. Since the last $(2m - r)$ rows of W^{+} are zero vectors, the vector $W^{+}U^{\mathrm{T}}(\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0)$ has zeros in the last $(2m - r)$ components.

We can apply the same procedure to compute the least squares solution of Eq. (3). The orthogonal projection of $\dot{\mathbf{h}}_0$ into the column space of J corresponds to that of $\dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0$ into the column space of JJ^{T}, and vice versa. However, the two projection procedures have a big difference in their computational complexities, especially for a large $m \ge 4$. For a $2m \times 2m$ matrix JJ^{T}, the SVD takes $O(m^3)$ time, whereas a $2m \times 8$ matrix J takes only $O(m)$ time. Furthermore, since any solution λ produces the same optimal solution $\dot{\mathbf{x}}$, there is no need to compute the least-squares solution λ. However, the SVD approach produces the least squares solution at no additional cost since it is the simplest as well as the most efficient solution among many others in the solution space.

In this paper, for computational efficiency, we directly solve the least squares solution of Eq. (3) within $O(m)$ time. Moreover, to fully utilize the least squares solution at no additional cost, we may incorporate the optimization criteria of Eq. (2) into the Jacobian matrix equation of Eq. (3). That is, instead of solving Eq. (3), we can solve the least-squares solution for $\ddot{\mathbf{x}}$ in the following matrix equation:

$$J(\ddot{\mathbf{x}} + \dot{\mathbf{x}}_0) = \dot{\mathbf{h}}_0$$

$$J\ddot{\mathbf{x}} = \dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0.$$

When there is nonempty solution space for this equation, the least squares solution $\ddot{\mathbf{x}}$ provides the optimal solution $\dot{\mathbf{x}} = \dot{\mathbf{x}}_0 + \ddot{\mathbf{x}}$ for Eqs. (2)–(3). If the solution space is empty, the least squares solution $\ddot{\mathbf{x}}$ minimizes the difference $\|J\ddot{\mathbf{x}} + J\dot{\mathbf{x}}_0 - \dot{\mathbf{h}}_0\|$ at the same time. Section 5 discusses more details on how to change the optimization criteria by scaling the columns and rows of J with different ratios.

Gleicher [11] deals with the overconstrained case of $m \ge 4$, by using the following quadratic energy function to attack the least-squares problem:

$$E = \frac{1}{2}\|J^{\mathrm{T}}\lambda - \dot{\mathbf{x}}\|^2 + \mu\|\lambda\|^2. \tag{9}$$

Here μ is a constant weighting factor for the Lagrange multiplier λ. Differentiating with respect to λ, the corresponding Lagrange equation is set up as

$$(JJ^{\mathrm{T}} + \mu I)\lambda = \dot{\mathbf{h}}_0, \tag{10}$$

where I is the $2m \times 2m$ identity matrix. When Eq. (3) has no solution, based on a similar reasoning as discussed above, the Lagrange Equation (10) is no longer a necessary condition for the quadratic optimization problem of Eq. (9). Furthermore, since any solution λ produces the same optimal solution $\dot{\mathbf{x}} = J^{\mathrm{T}}\lambda$, the minimization of $\|\lambda\|$ is not an important criteria in the optimization. Therefore, ignoring the term μI in Eq. (10), we obtain the equation

$$JJ^{\mathrm{T}}\lambda = \dot{\mathbf{h}}_0, \tag{11}$$

which is the same as Eq. (5) with the additional condition: $\dot{\mathbf{x}}_0 = 0$. That is, it is the same as the optimization problem of Eq. (2) with the energy function $E = \frac{1}{2}\|\dot{\mathbf{x}}\|^2$. Consequently, there is no essential difference from the original formulation of Gleicher and Witkin [12]. The addition of μI to the singular square matrix JJ^{T} has an effect of perturbing the matrix JJ^{T} into a nonsingular matrix $JJ^{\mathrm{T}} + \mu I$. Nevertheless, when a small perturbation μ is used, the nonsingularity is not always guaranteed in general. The use of a sufficiently large magnitude of μ results in the correspondingly large approximation error. The right amount of perturbation is quite difficult to determine automatically.

2.3. *Rank Deficiency of the Jacobian Matrix*

In the camera model, the unit quaternions (q_w, q_x, q_y, q_z) have a constraint for the unit length; i.e., $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$. Gleicher and Witkin [12] eliminated this constraint by using any nonzero quaternion q to represent the rotation R_q implied by the unit quaternion $q/\|q\|$. That is,

for a unit quaternion $q = (q_w, q_x, q_y, q_z) \in S^3$, the rotation matrix R_q is given by Shoemake [23]:

$$R_q = 2\begin{pmatrix} \frac{1}{2} - q_y^2 - q_z^2 & q_x q_y + q_w q_z & q_x q_z - q_w q_y & 0 \\ q_x q_y - q_w q_z & \frac{1}{2} - q_x^2 - q_z^2 & q_w q_x + q_y q_z & 0 \\ q_w q_y + q_x q_z & q_y q_z - q_w q_x & \frac{1}{2} - q_x^2 - q_y^2 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}. \tag{12}$$

For a general quaternion $q = (q_w, q_x, q_y, q_z) \in R^4$, by inserting $q/\|q\|$ into the above rotation matrix, Gleicher and Witkin [12] obtained the following rotation matrix:

$$R_q = \frac{2}{|q|^2}\begin{pmatrix} \frac{|q|^2}{2} - q_y^2 - q_z^2 & q_x q_y + q_w q_z & q_x q_z - q_w q_y & 0 \\ q_x q_y - q_w q_z & \frac{|q|^2}{2} - q_x^2 - q_z^2 & q_w q_x + q_y q_z & 0 \\ q_w q_y + q_x q_z & q_y q_z - q_w q_x & \frac{|q|^2}{2} - q_x^2 - q_y^2 & 0 \\ 0 & 0 & 0 & \frac{|q|^2}{2} \end{pmatrix}. \tag{13}$$

This method derives a rotation matrix for any non-zero quaternion and eliminates the constraint for a unit quaternion. However, the Jacobian matrix of the rotation is rank-deficient, that is, the column vectors are not linearly independent, which makes the computation of $\mathbf{x}$ numerically unstable.

For a given nonzero quaternion q, the quaternions tq represent the same rotation matrix as Eq. (13) for all nonzero $t \in R$, i.e.,

$$R_{tq} = R_q, \quad \text{for } t(\neq 0) \in R.$$

For given m points $p_1, \ldots, p_m \in R^3$ and a nonzero quaternion q, consider the transformation U_i: $R^4\backslash\{0\} \to R^3$, i.e.,

$$U_i(q) = R_q(p_i), \quad \text{for } i = 1, \ldots, m.$$

The differential of U_i is given by a linear transformation: $d(U_i)_q$: $R^4 \to R^3$, which can be represented by a 3×4 matrix [8]. Consider the rotation $R_{q(t)}$ implied by a quaternion curve $q(t) \in R^4$, for $t \in R$, and let $\hat{p}_i(t) \in R^3$ be the curve generated by the rotated points of p_i:

$$\hat{p}_i(t) = U_i(q(t)) = R_{q(t)}(p_i).$$

When the quaternion curve $q(t)$ is given by a radial line $q(t) = tq$, $t \in R$, we have

$$\hat{p}_i(t) = U_i(tq) = R_{tq}(p_i) = R_q(p_i).$$

That is, the curve $\hat{p}_i(t)$ is a constant curve. The derivative $(d\hat{p}_i/dt)(t)$ at $t = 1$ is given as a zero vector:

$$\left.\frac{d}{dt}\right|_{t=1} \hat{p}_i(t) = \left.\frac{d}{dt}\right|_{t=1} U_i(tq) = d(U_i)_q \left.\frac{d}{dt}\right|_{t=1} (tq)$$
$$= d(U_i)_q(q) = 0.$$

This means that the three rows (which are 4D vectors) of the Jacobian matrix of $d(U_i)_q$ are all orthogonal to the 4D vector q. Thus, $d(U_i)_q$ has rank 3, for $i = 1, \ldots, m$. Furthermore, for the transformation U: $R^4\backslash\{0\} \to R^{3m}$ defined by

$$U(q) = (R_q(p_1), \ldots, R_q(p_m)),$$

the differential dU_q is represented by a $3m \times 4$ Jacobian matrix and all the $3m$ rows of dU_q are orthogonal to the 4D vector q. Thus, dU_q has rank 3. In the formulation of the Jacobian matrix J of Ref. [12], its rank deficiency essentially results from that of the Jacobian matrix dU_q for the rotational degrees of freedom.

The Jacobian matrix J derived by Gleicher and Witkin [12] is as follows (denoted by using our notations of Section 3):

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial V_p}{\partial f} & \frac{\partial V_p}{\partial u_x} & \frac{\partial V_p}{\partial u_y} & \frac{\partial V_p}{\partial u_z} & \frac{\partial V_p}{\partial q_w} & \frac{\partial V_p}{\partial q_x} & \frac{\partial V_p}{\partial q_y} & \frac{\partial V_p}{\partial q_z} \end{pmatrix}$$
$$= \begin{pmatrix} \frac{\partial P_f}{\partial f} \\ \frac{\partial P_f}{\partial \hat{p}} R_q \frac{\partial T_u}{\partial u_x}(p) \\ \frac{\partial P_f}{\partial \hat{p}} R_q \frac{\partial T_u}{\partial u_y}(p) \\ \frac{\partial P_f}{\partial \hat{p}} R_q \frac{\partial T_u}{\partial u_z}(p) \\ \frac{\partial P_f}{\partial \hat{p}} \frac{\partial R_q}{\partial q_w} T_u(p) \\ \frac{\partial P_f}{\partial \hat{p}} \frac{\partial R_q}{\partial q_x} T_u(p) \\ \frac{\partial P_f}{\partial \hat{p}} \frac{\partial R_q}{\partial q_y} T_u(p) \\ \frac{\partial P_f}{\partial \hat{p}} \frac{\partial R_q}{\partial q_z} T_u(p) \end{pmatrix}^T.$$

The matrices $\partial P_f/\partial f$ and $\partial P_f/\partial \hat{p}$ are given in Eq. (20). (When we use the homogeneous coordinate, we have to add a zero column vector to the fourth column of $\partial P_f/\partial \hat{p}$.) The vectors $R_q(\partial T_u/\partial u_x)(p)$, $R_q(\partial T_u/\partial u_y)(p)$, and $R_q(\partial T_u/\partial u_z)(p)$, are the first, second, and third columns of R_q, respectively. Therefore, the computation of the first four components of $J(\mathbf{x})$ is relatively simple and efficient. However, the last four components have complex formulation because of the four partial derivatives of R_q given as follows:

$$\frac{\partial R_q}{\partial q_w} = \frac{-q_w}{|q|^2} R_q + \frac{2}{|q|^2} \begin{pmatrix} q_w & q_z & -q_y & 0 \\ -q_z & q_w & q_x & 0 \\ q_y & -q_x & q_w & 0 \\ 0 & 0 & 0 & q_w \end{pmatrix}$$

$$\frac{\partial R_q}{\partial q_x} = \frac{-q_x}{|q|^2} R_q + \frac{2}{|q|^2} \begin{pmatrix} q_x & q_y & q_z & 0 \\ q_y & -q_x & q_w & 0 \\ q_z & q_w & -q_x & 0 \\ 0 & 0 & 0 & q_x \end{pmatrix}$$

$$\frac{\partial R_q}{\partial q_y} = \frac{-q_y}{|q|^2} R_q + \frac{2}{|q|^2} \begin{pmatrix} -q_y & q_x & -q_w & 0 \\ q_x & q_y & q_z & 0 \\ q_w & q_z & -q_y & 0 \\ 0 & 0 & 0 & q_y \end{pmatrix}$$

$$\frac{\partial R_q}{\partial q_z} = \frac{-q_z}{|q|^2} R_q + \frac{2}{|q|^2} \begin{pmatrix} -q_z & q_w & q_x & 0 \\ -q_w & -q_z & q_y & 0 \\ q_x & q_y & q_z & 0 \\ 0 & 0 & 0 & q_z \end{pmatrix}.$$

Substituting these expressions into the last four components of the above Jacobian matrix $J(\mathbf{x})$ is more complex and time consuming than the construction of the last three columns of our Jacobian matrix given in Eq. (22). (Our Jacobian matrix also has one less column for the rotational component.) All these complications are due to the discrepancy between the rectangular coordinate structure of R^4 and the spherical coordinate structure of S^3. In this paper, we use the Lie group structure of S^3 to provide a canonical coordinate system of $R^3 \equiv T_1(S^3)$ to every tangent space $T_q(S^3)$, for $q \in S^3$. This Lie group structure enables us to derive a simple Jacobian matrix in Section 3.

3. DERIVATION OF A SIMPLE JACOBIAN MATRIX

In this section, we review some mathematical preliminaries on quaternion calculus [15, 16] and use them to derive a simple Jacobian matrix J for the perspective viewing transformation V_P.

3.1. *Quaternion and Rotation*

Given two 4D vectors $q_i = (q_{i,w}, q_{i,x}, q_{i,y}, q_{i,z}) \in R^4$, for $i = 1, 2$, we may interpret q_i as $q_i = [q_{i,w}, (q_{i,x}, q_{i,y}, q_{i,z})] = (q_{i,w}, q_{i,(x,y,z)}) \in R \times R^3$. The quaternion multiplication $q_1 \cdot q_2 = q_{12} = (q_{12,w}, q_{12,x}, q_{12,y}, q_{12,z}) = (q_{12,w}, q_{12,(x,y,z)}) \in R \times R^3 \equiv R^4$ is defined as follows:

$$q_{12,w} = q_{1,w}q_{2,w} - \langle q_{1,(x,y,z)}, q_{2,(x,y,z)} \rangle$$

$$q_{12,(x,y,z)} = q_{2,w}q_{1,(x,y,z)} + q_{1,w}q_{2,(x,y,z)} + q_{1,(x,y,z)} \times q_{2,(x,y,z)}.$$

Throughout this paper, $\langle , \rangle$ denotes the inner product and $\cdot$ denotes the quaternion multiplication. The above quaternion multiplication is closed on unit quaternions: for any $q_1, q_2 \in S^3$, we have $q_1 \cdot q_2 \in S^3$. Moreover, $(1, 0, 0, 0) = [1, (0, 0, 0)] \in S^3$ is the multiplication identity, and the inverse of q_i is given by: $q_i^{-1} = (q_{i,w}, -q_{i,x}, -q_{i,y}, -q_{i,z}) = (q_{i,w}, -q_{i,(x,y,z)}) = \overline{q}_i \in S^3$, where $\overline{q}_i$ denotes the conjugate of q_i. Note that the relation $\overline{q_1 \cdot q_2} = \overline{q}_2 \cdot \overline{q}_1$ holds for quaternion multiplication.

Given a unit quaternion $q \in S^3$, a 3D rotation $R_q \in SO(3)$ is defined as

$$R_q(p) = q \cdot p \cdot \overline{q}, \quad \text{for } p \in R^3, \tag{14}$$

where $p = (x, y, z)$ is interpreted as a quaternion $(0, x, y, z)$. (In this paper, any 3D vector used in the quaternion multiplication is assumed to be a quaternion with zero as the first component.) Let $q = (\cos\theta, \sin\theta(a, b, c)) \in S^3$, for some angle θ and unit vector $(a, b, c) \in S^2$, then $R_q \in SO(3)$ is the rotation by angle 2θ about the axis (a, b, c). The multiplicative constant, 2, in the angle of rotation, 2θ, is due to the fact that q appears twice in Eq. (14). Also note that $R_q \equiv R_{-q}$; that is, two antipodal points, q and $-q$ in S^3, represent the same rotation in $SO(3)$. Therefore, the two spaces S^3 and $SO(3)$ have the same local topology and geometry.

The derivative of a unit quaternion curve $q(t) \in S^3$ is always given in the following form:

$$q'(t) = v(t) \cdot q(t), \quad \text{for some } v(t) \in R^3. \tag{15}$$

This is due to the Lie group structure of S^3. For any differentiable curve $q(t) \in S^3$, we may consider a curve $q(t + s)$ which is parameterized by $s \in (-\varepsilon, \varepsilon)$, for a small constant $\varepsilon > 0$:

$$q(t + s) = q(t + s) \cdot q(t)^{-1} \cdot q(t), \quad \text{for } -\varepsilon < s < \varepsilon.$$

Then, we have

$$
\begin{aligned}
q'(t) &= \left.\frac{d}{ds}\right|_{s=0} q(t+s) \\
&= \left.\frac{d}{ds}\right|_{s=0} (q(t+s)\cdot q(t)^{-1})\cdot q(t) \\
&= (q'(t)\cdot q(t)^{-1})\cdot q(t).
\end{aligned}
$$

Since the curve $q_1(s) = q(t+s)\cdot q(t)^{-1}$ passes through the identity element $1 \in S^3$ when $s = 0$, we have $q_1'(0) = q'(t)\cdot q(t)^{-1} \in T_1(S^3) \equiv R^3$. Therefore, we have $q'(t) = v(t)\cdot q(t)$ for some $v(t) \in R^3$.

3.2. *Quanternion Calculus*

Let $q(t) = (q_w(t), q_{(x,y,z)}(t)) \in S^3$, for $t \in R$, be a unit quaternion curve. When the fixed point $p \in R^3$ is rotated by $R_{q(t)} \in SO(3)$, it generates a path $\hat{p}(t) \in R^3$:

$$\hat{p}(t) = R_{q(t)}(p) = q(t)\cdot p\cdot \overline{q(t)}.$$

The derivative $\hat{p}'(t)$ is given by

$$
\begin{aligned}
\hat{p}'(t) &= q'(t)\cdot p\cdot \overline{q(t)} + q(t)\cdot p\cdot \overline{q'(t)} \\
&= v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} + q(t)\cdot p\cdot \overline{v(t)\cdot q(t)} \\
&= v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} + q(t)\cdot p\cdot \overline{q(t)}\cdot \overline{v(t)} \\
&= v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} - \overline{q(t)\cdot p\cdot \overline{q(t)}\cdot \overline{v(t)}} \\
&= v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} - v(t)\cdot q(t)\cdot (-p)\cdot \overline{q(t)} \\
&= v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} + v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} \\
&= 2v(t)\cdot q(t)\cdot p\cdot \overline{q(t)} \\
&= 2v(t)\cdot \hat{p}(t) \\
&= 2v(t)\times \hat{p}(t).
\end{aligned}
$$

When we interpret $\omega(t) = 2v(t) \in R^3$ as the angular velocity, the above is exactly the same as the formula given in classical dynamics [19, 28]:

$$\hat{p}'(t) = \omega(t)\times \hat{p}(t). \tag{16}$$

3.3. *The Jacobian Matrix of the Transformation U*

Given fixed 3D points $p_i \in R^3$ (for $i = 1, \ldots, m$), let $\hat{p}_i(t)$ be the rotated point of p_i by the 3D rotation $R_{q(t)}$ of the unit quaternion $q(t)$. Then, we have

$$\hat{p}_i'(t) = \omega(t)\times \hat{p}_i(t),$$

where $\hat{p}_i(t) = R_{q(t)}(p_i)$. Thus, for the transformation U_i: $S^3 \to R^3$, i.e.,

$$U_i(q) = R_q(p_i) = \hat{p}_i,$$

the differential $d(U_i)_q$ is given by $d(U_i)_q$: $T_q(S^3) \to R^3$, i.e.,

$$d(U_i)_q(q') = \omega \times \hat{p}_i.$$

Since the isomorphism

$$
\begin{aligned}
E&: T_q(S^3) \to R^3 \\
q' &= \tfrac{1}{2}\omega\cdot q \mapsto \omega
\end{aligned}
$$

identifies the tangent space $T_q(S^3)$ with the 3D Euclidean space R^3, we may interpret the differential $d(U_i)_q$ as $d(U_i)_q$: $R^3 \to R^3$; i.e.,

$$d(U_i)_q(\omega) = \omega \times \hat{p}_i = -\hat{p}_i \times \omega.$$

Thus, the Jacobian matrix of U_i can be represented by a 3×3 square matrix,

$$\begin{pmatrix} 0 & \hat{z}_i & -\hat{y}_i \\ -\hat{z}_i & 0 & \hat{x}_i \\ \hat{y}_i & -\hat{x}_i & 0 \end{pmatrix},$$

where $\hat{p}_i = (\hat{x}_i, \hat{y}_i, \hat{z}_i) \in R^3$. When an angular velocity ω is computed, the quaternion $q(t)$ is updated to a new quaternion $q(t + \Delta t)$ by

$$q(t+\Delta t) = \exp\left(\frac{\Delta t}{2}\omega\right)\cdot q(t),$$

where Δt is the time interval for the integration, and the transformation, exp: $R^3 \to S^3$, is the exponential map. (See [7, 15, 16] for more details on the exponential map.) Figure 3 shows how the exponential map, *exp*, projects the tangent vector $\Delta t\ q' \in T_{q(t)}(S^3)$ at $q(t)$ into a unit quaternion $q(t + \Delta t) \in S^3$ which is at the distance $\|\Delta t\ q'\|$ from $q(t)$ in the direction of $\Delta t\ q'$.

3.4. *The Jacobian Matrix for a Viewing Transformation*

Let the position and orientation of the virtual camera at time t be given by $\overline{u}(t) \in R^3$ and $\overline{q}(t) \in S^3$. For a given fixed 3D point $p = (x, y, z)$ in the world coordinate system, the projected 2D image point $h(t) \in R^2$ can be represented by

$$h(t) = P_{f(t)} \circ R_{\overline{q}(t)^{-1}} \circ T_{-\overline{u}(t)}(p).$$

(See Foley *et al.* [10] for a detailed explanation of the above.) $P_{f(t)}$ is the perspective projection with a focal length

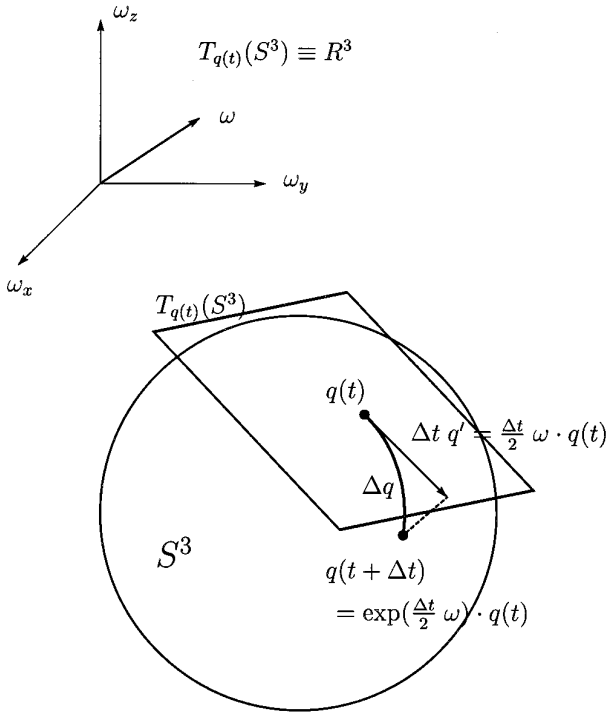

FIG. 3. Construction of $q(t + \Delta t)$.

$f(t)$, $R_{q(t)}$ is the rotation for a unit quaternion $q(t)$, and $T_{u(t)}$ is the translation by $u(t) \in R^3$. Note that the order of translation and rotation is different from that of Gleicher and Witkin [12]. Our experiments show that, for the 3D point p, it is more efficient and numerically stable to do translation first and rotation later, rather than the other way around. Let $u(t) = (u_x(t), u_y(t), u_z(t)) = -\overline{u}(t)$ and $q(t) = (q_w(t), q_x(t), q_y(t), q_z(t)) = \overline{q}(t)^{-1}$. Thus, $h(t)$ is rewritten as

$$\begin{aligned} h(t) &= P_{f(t)} \circ R_{q(t)} \circ T_{u(t)}(p) \\ &= V_p(\mathbf{x}(t)), \end{aligned} \tag{17}$$

where $\mathbf{x}(t) = (f(t), u_x(t), u_y(t), u_z(t), q_w(t), q_x(t), q_y(t), q_z(t))$.

The 3D rigid transformation: $\hat{p} = (\hat{x}, \hat{y}, \hat{z}) = R_q \circ T_u(p)$ is given by

$$\begin{aligned} \hat{p}(t) &= R_{q(t)} \circ T_{u(t)}(p) \\ &= q(t) \cdot (p + u(t)) \cdot \overline{q}(t), \end{aligned}$$

where $\overline{q}(t) = \overline{q(t)}$. The perspective transformation $P_{f(t)}$ is then applied to $\hat{p}(t)$ as follows:

$$\begin{aligned} h(t) &= P_{f(t)}(\hat{p}(t)) = P_{f(t)}(\hat{x}(t), \hat{y}(t), \hat{z}(t)) \\ &= \left(\frac{f(t)\hat{x}(t)}{\hat{z}(t)}, \frac{f(t)\hat{y}(t)}{\hat{z}(t)}\right). \end{aligned} \tag{18}$$

To derive the Jacobian matrix J of the viewing transformation V_p, we differentiate Eq. (17),

$$\begin{aligned} \frac{dh}{dt} &= \frac{dV_p}{dt} = \frac{d}{dt} P_{f(t)} \circ R_{q(t)} \circ T_{u(t)}(p) \\ &= \frac{\partial P_f}{\partial f}\frac{df}{dt} + \frac{\partial P_f}{\partial \hat{p}}\frac{d}{dt}(R_{q(t)} \circ T_{u(t)}(p)), \end{aligned} \tag{19}$$

where $P_f(\hat{p})$ is considered to be a map with two arguments: f and $\hat{p}$ (see Eq. (18)). It is easy to show that

$$\frac{\partial P_f}{\partial f} = \begin{pmatrix} \frac{\hat{x}(t)}{\hat{z}(t)} \\ \frac{\hat{y}(t)}{\hat{z}(t)} \end{pmatrix} \quad \text{and}$$

$$\frac{\partial P_f}{\partial \hat{p}} = \begin{pmatrix} \frac{f(t)}{\hat{z}(t)} & 0 & -\frac{f(t)\hat{x}(t)}{\hat{z}^2(t)} \\ 0 & \frac{f(t)}{\hat{z}(t)} & -\frac{f(t)\hat{y}(t)}{\hat{z}^2(t)} \end{pmatrix}. \tag{20}$$

Using the formula: $\hat{p}'(t) = \omega(t) \times \hat{p}(t) = -\hat{p}(t) \times \omega(t)$ derived in Section 3.2, we have

$$\begin{aligned} &\frac{d}{dt}(R_{q(t)} \circ T_{u(t)}(p)) \\ &= \frac{d}{dt}(q(t) \cdot (p + u(t)) \cdot \overline{q}(t)) \\ &= \omega(t) \times \hat{p}(t) + q(t) \cdot u'(t) \cdot \overline{q}(t) \\ &= -\hat{p}(t) \times \omega(t) + R_{q(t)}(u'(t)) \\ &= \begin{pmatrix} 0 & \hat{z} & -\hat{y} \\ -\hat{z} & 0 & \hat{x} \\ \hat{y} & -\hat{x} & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} + R_{q(t)} \begin{pmatrix} u'_x \\ u'_y \\ u'_z \end{pmatrix} \\ &= \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 & \hat{z} & -\hat{y} \\ R_{21} & R_{22} & R_{23} & -\hat{z} & 0 & \hat{x} \\ R_{31} & R_{32} & R_{33} & \hat{y} & -\hat{x} & 0 \end{pmatrix} \begin{pmatrix} u'_x \\ u'_y \\ u'_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}, \end{aligned} \tag{21}$$

where R_{ij} is the ijth component of the matrix $R_{q(t)}$. By applying Eqs. (20) and (21) to Eq. (19), we obtain

$$\frac{dh}{dt} = \begin{pmatrix} \frac{\hat{x}}{\hat{z}} \\ \frac{\hat{y}}{\hat{z}} \end{pmatrix} f' + \begin{pmatrix} \frac{f}{\hat{z}} & 0 & -\frac{f\hat{x}}{\hat{z}^2} \\ 0 & \frac{f}{\hat{z}} & -\frac{f\hat{y}}{\hat{z}^2} \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 & \hat{z} & -\hat{y} \\ R_{21} & R_{22} & R_{23} & -\hat{z} & 0 & \hat{x} \\ R_{31} & R_{32} & R_{33} & \hat{y} & -\hat{x} & 0 \end{pmatrix} \begin{pmatrix} u'_x \\ u'_y \\ u'_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = J \begin{pmatrix} f' \\ u'_x \\ u'_y \\ u'_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix},$$

where

$$J = \begin{pmatrix} \frac{\hat{x}}{\hat{z}} & \frac{fR_{11}}{\hat{z}} - \frac{f\hat{x}R_{31}}{\hat{z}^2} & \frac{fR_{12}}{\hat{z}} - \frac{f\hat{x}R_{32}}{\hat{z}^2} & \frac{fR_{13}}{\hat{z}} - \frac{f\hat{x}R_{33}}{\hat{z}^2} & -\frac{f\hat{x}\hat{y}}{\hat{z}^2} & f + \frac{f\hat{x}^2}{\hat{z}^2} & -\frac{f\hat{y}}{\hat{z}} \\ \frac{\hat{y}}{\hat{z}} & \frac{fR_{21}}{\hat{z}} - \frac{f\hat{y}R_{31}}{\hat{z}^2} & \frac{fR_{22}}{\hat{z}} - \frac{f\hat{y}R_{32}}{\hat{z}^2} & \frac{fR_{23}}{\hat{z}} - \frac{f\hat{y}R_{33}}{\hat{z}^2} & -f - \frac{f\hat{y}^2}{\hat{z}^2} & \frac{f\hat{y}\hat{x}}{\hat{z}^2} & \frac{f\hat{x}}{\hat{z}} \end{pmatrix}. \tag{22}$$

This 2×7 Jacobian matrix J is much simpler than the one described in Section 2.3.

4. CAMERA CONTROL BY MOVING IMAGE CONTROL POINTS

4.1. *Moving a Single Image Control Point*

For the camera control, we need to compute the parameter $\mathbf{x}$ satisfying the equation

$$V_p(\mathbf{x}) = h_0 \tag{23}$$

where $p \in R^3$ is the given 3D point, and h_0 is the 2D point onto which the point p is required to be projected. We approximate the solution $\mathbf{x}$ of Eq. (23) by using the Newton method [4, 6]. The Newton approximation is carried out by solving a sequence of linear equations, which are obtained by differentiating the given nonlinear equation. In each linear equation, the unknowns are the velocities of the camera parameters, that is, $\dot{\mathbf{x}} = (f', u'_x, u'_y, u'_x, \boldsymbol{\omega}) \in R^7 \equiv R^4 \times T_q(S^3)$. By integrating the velocities, we can obtain the solution $\mathbf{x}$ for the given nonlinear system of Eq. (23). Let F: $R^4 \times S^3 \to R^2$, be defined by

$$F(\mathbf{x}) = V_p(\mathbf{x}) - h_0 = \left(\frac{f\hat{x}}{\hat{z}} - (h_0)_x, \frac{f\hat{y}}{\hat{z}} - (h_0)_y \right). \tag{24}$$

The solution $\mathbf{x}$ for $F(\mathbf{x}) = 0$ satisfies $V_p(\mathbf{x}) = h_0$.

When $\mathbf{x}^{(i)} = (f^{(i)}, u^{(i)}, q^{(i)}) \in R^4 \times S^3$, the value of F at $\mathbf{x}^{(i+1)}$ is given as

$$F(\mathbf{x}^{(i+1)}) = F(\mathbf{x}^{(i)}) + dF_{\mathbf{x}^{(i)}}(\dot{\mathbf{x}}^{(i)}) + O(\|\dot{\mathbf{x}}^{(i)}\|^2),$$

where $dF_{\mathbf{x}^{(i)}}$: $R^4 \times T_q(S^3) \to R^2$ is the differential of F at $\mathbf{x}^{(i)}$ (see [8]). The value of $\dot{\mathbf{x}}^{(i)}$ is approximated to

$$\Delta\mathbf{x}^{(i)} = (\Delta f^{(i)}, \Delta u^{(i)}, \boldsymbol{\omega}^{(i)}) \in R^7 \equiv R^4 \times T_{q^{(i)}}(S^3).$$

Ignoring the last term $O(\|\Delta\mathbf{x}^{(i)}\|^2)$, we have

$$F(\mathbf{x}^{(i+1)}) = F(\mathbf{x}^{(i)}) + dF_{\mathbf{x}^{(i)}}(\Delta\mathbf{x}^{(i)}) = 0.$$

Since the matrix representation of the differential $dF_{\mathbf{x}^{(i)}}$ is the Jacobian matrix $J(\mathbf{x}^{(i)})$, the following linear system is obtained:

$$J(\mathbf{x}^{(i)})(\Delta\mathbf{x}^{(i)}) = -F(\mathbf{x}^{(i)}). \tag{25}$$

The next camera parameter vector $\mathbf{x}^{(i+1)}$ is obtained by

$$\mathbf{x}^{(i+1)} = \left(f^{(i)} + \Delta f^{(i)}, u^{(i)} + \Delta u^{(i)}, \exp\left(\frac{\boldsymbol{\omega}^{(i)}}{2} \right) \cdot q^{(i)} \right) \in R^4 \times S^3.$$

It should be noted that $J(\mathbf{x}^{(i)})$ is not a square matrix and therefore not invertible. By using the singular value decomposition of $J(\mathbf{x}^{(i)})$, we will approximate the inverse matrix. This is discussed in more detail in Section 5.

4.2. *Moving Multiple Image Control Points*

The linear system for a single control point has been derived as a 2×7 matrix equation in Section 4.1. When a single control point is not enough to fully control the virtual camera, the user gains more control through the use of multiple control points.

For m image control points, Eq. (24) can be generalized as follows:

$$F(f, u_x, u_y, u_z, q_w, q_x, q_y, q_z) = \begin{pmatrix} \frac{f(\hat{p}_1)_x}{(\hat{p}_1)_z} - (h_1)_x \\ \frac{f(\hat{p}_1)_y}{(\hat{p}_1)_z} - (h_1)_y \\ \vdots \\ \frac{f(\hat{p}_i)_x}{(\hat{p}_i)_z} - (h_i)_x \\ \frac{f(\hat{p}_i)_y}{(\hat{p}_i)_z} - (h_i)_y \\ \vdots \\ \frac{f(\hat{p}_m)_x}{(\hat{p}_m)_z} - (h_m)_x \\ \frac{f(\hat{p}_m)_y}{(\hat{p}_m)_z} - (h_m)_y \end{pmatrix}. \quad (26)$$

For this function F, the Jacobian matrix $J(\mathbf{x}^{(i)})$ now becomes a $2m \times 7$ matrix. The linear equation $J(\mathbf{x}^{(i)})\ \Delta\mathbf{x}^{(i)} = -F(\mathbf{x}^{(i)})$ may have many solutions or no solution at all, depending on the rank of $J(\mathbf{x}^{(i)})$ and the value of $F(\mathbf{x}^{(i)})$. Thus, we need proper criteria for determining the solution $\mathbf{x}$ for each case; such criteria are given in Section 5.

4.3. *Tracking 3D Moving Data Points*

We have derived the Jacobian matrix J under the assumption that all the picked 3D points p_i's are stationary points. However, when the 3D points p_i's are allowed to move, we need to consider this fact when controlling the virtual camera parameters. For the moving 3D point $p(t)$, Eq. (21) is rederived as follows:

$$\begin{aligned} &\frac{d}{dt}\left(R_{q(t)} \circ T_{u(t)}(p(t))\right) \\ &\quad = \frac{d}{dt}\left(q(t) \cdot (p(t) + u(t)) \cdot \overline{q}(t)\right) \\ &\quad = q'(t) \cdot (p(t) + u(t)) \cdot \overline{q}(t) + q(t) \cdot (p(t) + u(t)) \cdot \overline{q}'(t) \\ &\qquad + q(t) \cdot (p'(t) + u'(t)) \cdot \overline{q}(t) \\ &\quad = \omega(t) \times \hat{p}(t) + q(t) \cdot (p'(t) + u'(t)) \cdot \overline{q}(t) \\ &\quad = -\hat{p}(t) \times \omega(t) + R_{q(t)}(u'(t) + p'(t)). \end{aligned}$$

Thus, for the perspective viewing transformation $h(t) = V_{p(t)}(\mathbf{x}(t))$, we have

$$\frac{dh}{dt} = J(\mathbf{x}^{(i)}) \begin{pmatrix} f' \\ u'_x + p'_x \\ u'_y + p'_y \\ u'_z + p'_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}. \quad (27)$$

where $J(\mathbf{x}^{(i)})$ is the Jacobian matrix derived in Section 3.4. Thus, Eq. (25) is replaced by

$$J(\mathbf{x}^{(i)})(\Delta\mathbf{x}^{(i)}) = -F(\mathbf{x}^{(i)}) - J(\mathbf{x}^{(i)})(0, p'_x, p'_y, p'_z, 0, 0, 0)^{\mathrm{T}}.$$

5. COMPUTATION OF $\Delta\mathbf{x}$

5.1. *Computation of Pseudo Inverse*

When there are m control points in the image space, the system to be solved for $\Delta\mathbf{x}$ is a $2m \times 7$ linear system,

$$J(\mathbf{x})\ \Delta\mathbf{x} = -F(\mathbf{x}), \quad (28)$$

where $\Delta\mathbf{x} = (f', u'_x, u'_y, u'_z, \omega_x, \omega_y, \omega_z)$. Since the matrix $J = J(\mathbf{x})$ is not a square matrix, it is always singular. Therefore, we need to compute the least squares solution $\Delta\mathbf{x}$ of Eq. (28). (In Section 2.2, we discussed the geometric constructions associated with the least-squares solution.)

To compute the least squares solution, we use the singular value decomposition (SVD) of the Jacobian matrix J and construct its pseudo inverse J^+ [13, 22, 26]. For the case of $m \geq 4$, the pseudo inverse construction requires the SVD of a $2m \times 7$ matrix, which takes $O(m)$ time only. For a large $m \geq 4$, it is more efficient to compute the pseudo inverse J^+ by using the projection method [26]:

$$J^+ = (J^{\mathrm{T}}J)^{-1}J^{\mathrm{T}}.$$

The projection method also produces the least squares solution $\Delta\mathbf{x}$ which at the same time minimizes the magnitude: $\|J\ \Delta\mathbf{x} + F(\mathbf{x})\|$.

The geometric constructions associated with the projection method can be explained as follows. Assume $J\ \Delta\mathbf{x}$ is the orthogonal projection of $-F(\mathbf{x})$ into the column space of J. Then the difference vector $J\ \Delta\mathbf{x} + F(\mathbf{x})$ is orthogonal to the column space of J (i.e., orthogonal to each column of J). Therefore, we have

$$J^{\mathrm{T}}(J\,\Delta\mathbf{x} + F(\mathbf{x})) = 0,$$
$$J^{\mathrm{T}}J\,\Delta\mathbf{x} = J^{\mathrm{T}}(-F(\mathbf{x})). \tag{29}$$

Since the two matrices $J^{\mathrm{T}}J$ and J^{T} have the same column space, it is clear that the vector $J^{\mathrm{T}}(-F(\mathbf{x}))$ is in the column space of $J^{\mathrm{T}}J$. Therefore, Eq. (29) has a $(7 - r)$-dimensional solution space, where r is the common rank of $J^{\mathrm{T}}J$ and J. This solution space is exactly the same as the solution space of the equation

$$J\,\Delta\mathbf{x} = -\hat{F}(\mathbf{x}), \tag{30}$$

where $-\hat{F}(\mathbf{x})$ is the projection of the vector $-F(\mathbf{x})$ into the column space of J. This is because Eq. (29) is a necessary condition of Eq. (30) and the two solution spaces have the same dimension $(7 - r)$. Therefore, the least-squares solution $\Delta\mathbf{x}$ of Eq. (29) provides the least squares solution for the given linear system of Eq. (28). The corresponding pseudo inverse is given as follows:

$$J^{+} = (J^{T}J)^{+}J^{\mathrm{T}}.$$

The construction of the 7×7 square matrix $J^{\mathrm{T}}J$ takes $O(m)$ time and the pseudo inverse operation for $J^{\mathrm{T}}J$ takes constant time only. Therefore, the formulation of the square matrix $J^{\mathrm{T}}J$ is the dominating factor in the overall computation of the least squares solution of Eq. (28). In the underconstrained case of $m \leq 3$, it is quite difficult to specify (even intentionally) conflicting inputs on different image control points; that is, our Jacobian matrix J usually has its full rank $2m$. In the overconstrained case of $m \geq 4$, however, it is inevitable to have some conflicts because of the upper limit 7 for the rank of the matrix $J^{\mathrm{T}}J$. Unfortunately, in most of the test examples (for the case of $m \geq 4$), we have experienced that our Jacobian matrix has rank 6 only, not its full rank 7. This is because the effect of a focal length control can be achieved by translating the camera forward/backward to the camera viewing direction. Consequently, the first column of J (corresponding to the focal length control) is nearly redundant. In keyframe animation, the virtual camera is usually controlled with a fixed focal length, except in a few scenes which require special camera effects. Therefore, we use a fixed focal length in the overconstrained case. The resulting Jacobian matrix J is a $2m \times 6$ matrix which is obtained by removing the first column of our Jacobian matrix J.

When the square matrix $J^{\mathrm{T}}J$ is nonsingular, we can apply a more efficient matrix inversion algorithm (e.g., the LU decomposition of Gauss Elimination) to compute $(J^{\mathrm{T}}J)^{-1}$ instead of using the SVD method (which takes about five times more computation). This is also an important advantage of our Jacobian matrix over that of Gleicher and Witkin [12] which always produces a singular square matrix $J^{\mathrm{T}}J$.

5.2. *Weight of Camera Parameters and Image Control Points*

The pseudo inverse J^{+} provides the least squares solution $\Delta\mathbf{x}$ for Eq. (28). However, in some situations, other solutions $\Delta\mathbf{x}$ may be required. For example, when the animator wants to move the camera with little change of focus and/or camera rotation to facilitate more comfortable viewing of the scene, a different solution $\Delta\mathbf{x}$ should be chosen. That is, higher weights should be given to the parameters for fewer changes. Furthermore, the camera parameters (i.e., focus, translation, and rotation) have different units of measure; thus, it is irrational to treat them with equal weight.

A simple way to enforce this change is to scale the camera parameter space in different ratios. To give different weights to the camera parameters, we change $J\,\Delta\mathbf{x} = -F(\mathbf{x})$ into $JD_cD_c^{-1}\,\Delta\mathbf{x} = -F(\mathbf{x})$ and solve for $\Delta y = D_c^{-1}\,\Delta\mathbf{x}$ in the following equation:

$$(JD_c)\,\Delta\mathbf{y} = -F(\mathbf{x}). \tag{31}$$

The solution $\Delta\mathbf{x}$ is then obtained by $\Delta\mathbf{x} = D_c\,\Delta\mathbf{y}$. The column weighting matrix D_c is a 7×7 diagonal matrix such that $(D_c)_{ii}$ is a weight value for the ith parameter of $\Delta\mathbf{x}$. The solution $\Delta\mathbf{y}$ obtained from Eq. (31) is a skewed version of the solution $\Delta\mathbf{x}$. The solution $\Delta\mathbf{x} = D_c\,\Delta\mathbf{y}$ becomes a weighted solution.

As we assign different weights to the image control points, the space R^{2m} is scaled with different ratios along different axes. Therefore, the column space of J also changes into a different vector subspace in R^{2m}. Consequently, the orthogonal projection vector $-\hat{F}(\mathbf{x})$ and the least-squares solution $\Delta\mathbf{x}$ also change. This means that the selection of the least squares solution $\Delta\mathbf{x}$ can be controlled by assigning different weights to the image control points. However, this change does not take effect when there is at least one solution $\Delta\mathbf{x}$, that is, when the vector $-F(\mathbf{x})$ is already in the column space of J. In this case, $-F(\mathbf{x})$ projects onto itself no matter how the column space of J is transformed, and we always have $\|J\,\Delta\mathbf{x} + F(\mathbf{x})\| = 0$.

The row weighting scheme is useful in camera control. For example, the animator may wish to design the main camera motion with two or three control points and add a few more additional fine controls. To determine the contribution of each control point to the selection of the least squares solution $\Delta\mathbf{x}$, we give different weight to each row of the matrix J. The given linear system $J\,\Delta\mathbf{x} = -F(\mathbf{x})$ is then changed into the form $D_rJ\,\Delta\mathbf{x} = -D_rF(\mathbf{x})$, where the row weighting matrix D_r is a $2m \times 2m$ diagonal matrix. The diagonal element $(D_r)_{ii}$ is a weight value for the

$\lceil i/2 \rceil$th control point. This row weighting can be combined with the column weighting of Eq. (31) as follows:

$$(D_r J D_c)\,\Delta\mathbf{y} = -D_r F(\mathbf{x})$$

The row weighting scheme can be used quite effectively in computer animation as follows. For an animation movie with dramatic scene changes, it is not sufficient to have only a few control points. The control points suitable for the start of the scene may not work well at the end of the scene. It is desirable to limit the effect of each control point to a certain time interval while keeping the smoothness of the camera motion. To do this, each control point p_i is assigned with an active time interval $[s_i, e_i]$ during which the control point is valid. Furthermore, the *active set* $A(t)$ at time t is defined to be the set of image control points which is active at time t,

$$A(t) = \{p_i \mid t \in [s_i, e_i],\ 1 \le i \le n\},$$

where p_i is the ith control point. The Jacobian matrix J at time t is constructed from the active control points in $A(t)$. To keep the smoothness of the camera motion at $t = s_i$ and $t = e_i$, for each $p_i \in A(t)$, we need to use a non-negative smooth function $w_i(t) = (D_r)_{ii}$ which is 0 for $t \le s_i$ or $t \ge e_i$.

6. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The camera control process is briefly summarized in the following pseudo code:

Camera-Control $(f_s, f_e, \mathbf{x}_{f_s}, H_{f_e})$
Input: f_s, f_e : the start and end frames;
$\mathbf{x}_{f_s}$: the start camera parameters;
H_{f_e} : the end positions of the image control points;
Output: $\mathbf{x}_{f_s}, \ldots, \mathbf{x}_{f_e}$: a sequence of camera parameters;
begin
$H_{f_s} = V_P(\mathbf{x}_{f_s})$;
for $f_j := f_s + 1$ **to** f_e **do**
begin
$\Delta H := \dfrac{1}{f_e - f_j + 1}(H_{f_e} - H_{(f_j - 1)})$;
$H_{f_j} := H_{(f_j - 1)} + \Delta H$;
$\mathbf{x}_{f_j} :=$ Newton$(\mathbf{x}_{(f_j-1)}, H_{(f_j-1)}, H_{f_j})$;
end
end
Newton $(\mathbf{x}^{(0)}, H^{(0)}, H_0)$
Input: $\mathbf{x}^{(0)}$: the initial camera parameters;
$H^{(0)} = (h_1^{(0)}, \ldots, h_m^{(0)})$: the start positions of the image control points;
H_0 : the destination positions of the image control points;
Output: $\mathbf{x}^{(i+1)}$: the approximate solution of $V_P(\mathbf{x}) = H_0$;
begin
/* $H^{(i)} = (h_1^{(i)}, \ldots, h_m^{(i)})$: the image control points at the ith iteration step */
for $i = 0$ **to** *MAX-ITERATION* $- 1$ **do**
begin
$F(\mathbf{x}^{(i)}) := H^{(i)} - H_0$;
Construct the Jacobian matrix: $J(\mathbf{x}^{(i)})$;
Compute $\Delta\mathbf{x}^{(i)}$ in the matrix equation:
$J(\mathbf{x}^{(i)})\,\Delta\mathbf{x}^{(i)} = -F(\mathbf{x}^{(i)})$;
/* $\mathbf{x}^{(i)} = (f^{(i)}, u^{(i)}, q^{(i)})$,
$\Delta\mathbf{x}^{(i)} = (\Delta f^{(i)}, \Delta u^{(i)}, \omega^{(i)})$,
Δt is the time step for the Newton Approximation */
$\mathbf{x}^{(i+1)} = \left(f^{(i)} + \Delta t\ \Delta f^{(i)},\ u^{(i)} + \Delta t\ \Delta u^{(i)},\ \exp\left(\dfrac{\Delta t\,\omega^{(i)}}{2}\right) \cdot q^{(i)}\right)$;
$H^{(i+1)} = V_P(\mathbf{x}^{(i+1)})$;
if $\left|\|H^{(i+1)} - H_0\| - \|H^{(i)} - H_0\|\right| < \varepsilon$ **then**
return $(\mathbf{x}^{(i+1)})$;
end
end

The procedure *Camera-Control* produces a sequence of camera parameters from the start time f_s to the end time f_e. The control of the camera motion is given by H_{f_e}, and the path of the image control points P is generated as the straight line (which can be replaced with other curves) from the initial position H_{f_s} to the final position H_{f_e}:

$$H(t) = H_{f_s} + \frac{t - f_s}{f_e - f_s}(H_{f_e} - H_{f_s}), \quad \text{for } f_s \le t \le f_e.$$

At each time step f_j, the camera parameter $\mathbf{x}_{f_j}$ is obtained from the procedure *Newton*, where the solution $\mathbf{x}_{f_j}$ is numerically computed by the Newton approximation method described in Section 4, starting from the initial value $\mathbf{x}_{f_j - 1}$. The procedure *Newton* is designed for the applications in keyframe animation, which require high precision numerical approximation. For the real-time applications in 3D user interface, the computation time is more important than the numerical precision. In that case, we can set the iteration number *MAX-ITERATION* to 1 and simply return the first value $\mathbf{x}^{(1)}$ evaluated in the procedure.

Three experimental results are demonstrated in Fig. 4 and Table 1. The cases of controlling three and four image points are shown in Figs. 4a and 4b, respectively. In these two cases, the 3D points are stationary points. Figure 4c shows the camera control for three moving 3D points. The numerical approximations up to three control points are accurate as shown in Figs. 4a and 4c. However, in the overconstrained case of controlling more than three points, we have experienced large approximation errors as shown in Fig. 4b.

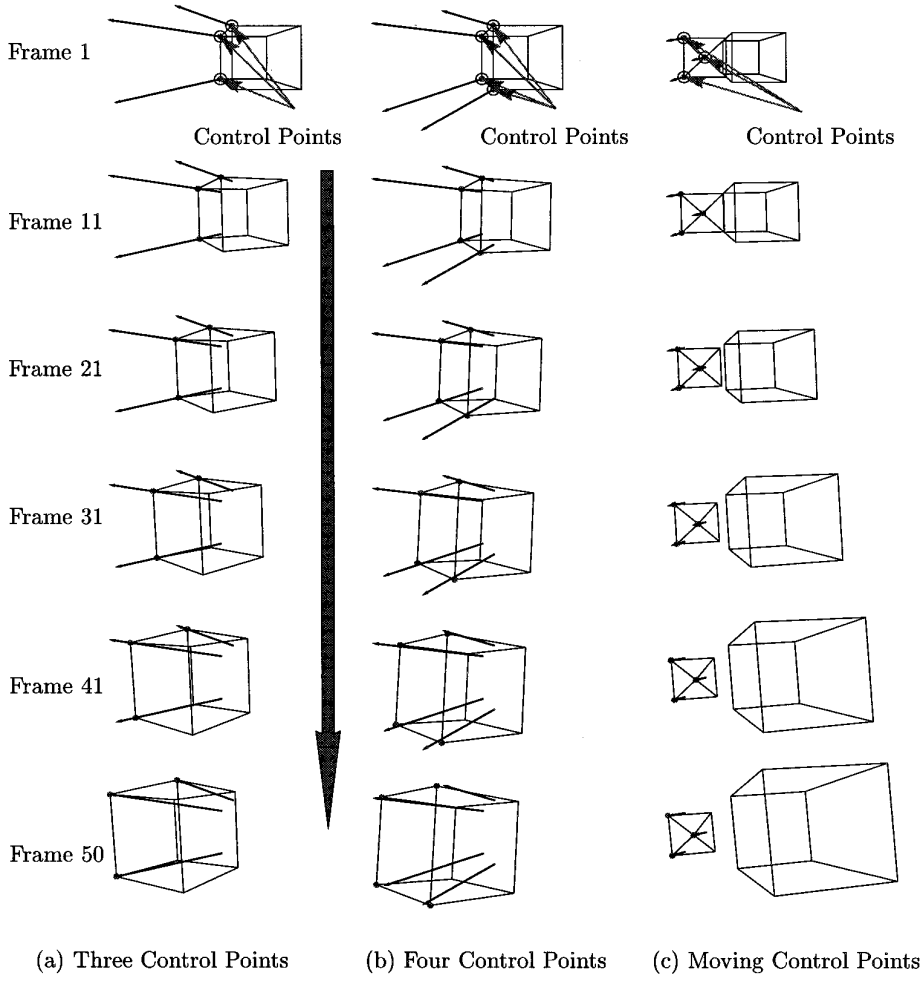

FIG. 4. Experimental results.

7. KEYFRAME ANIMATION OF VIRTUAL CAMERA PARAMETERS

Most computer animation systems control the camera motion using keyframe animation technique, i.e., by interpolating the camera parameters specified at each keyframe. Each parameter is interpolated without regard to other camera parameters. Therefore, the interpolation does not facilitate the synchronization of different camera parameters to generate natural camera motion. In this section, we consider how to use through-the-lens camera control technique to resolve this problem.

7.1. *Blending Two Sequences of Camera Parameters*

Consider the interpolation of two consecutive keyframes. First of all, the user specifies a few control points (e.g., three points), and specifies a trace curve for each control point on the camera view plane. Then, the interpolation is done, using the procedure *Camera-Control* of Section 6. However, there is a serious problem that should be resolved before we apply the method. In the procedure *Camera-Control*, the camera parameter $\mathbf{x}_i$ at the ith frame is used as the initial value for the procedure *Newton* to compute the camera parameter $\mathbf{x}_{i+1}$ at the $(i + 1)$th frame. There is no consideration of the end frame parameter $\mathbf{x}_{f_e}$. Therefore, the last frame parameter generated by *Newton* is not guaranteed to be identical to the given end frame parameter $\mathbf{x}_{f_e}$. For example, consider the case of manipulating three image control points. In this case, even if all three image control points are interpolated exactly, there is still one extra degree of freedom left for the final camera parameter. The sequence $\{\mathbf{x}_i\}$ may converge to any of them, not necessarily to the end frame parameter $\mathbf{x}_{f_e}$. Moreover, in the overconstrained case, due to the error in the least-squares approximation, the sequence may not converge to $\mathbf{x}_{f_e}$ exactly.

To resolve this problem, we blend two camera parameter sequences: one generated from $\mathbf{x}_{f_s}$ toward $\mathbf{x}_{f_e}$ (forward con-

TABLE 1
Camera Parameters for Fig. 4

Frame no.	f	u_x	u_y	u_z	q_w	q_x	q_y	q_z
				(a) Camera parameters for Fig. 4a				
1	1.00000	−2.00000	0.00000	5.00000	1.00000	0.00000	0.00000	0.00000
11	1.18103	−2.67962	0.10983	4.69282	0.99002	−0.00507	−0.14025	0.00016
21	1.29468	−3.05391	0.20476	4.24188	0.96961	−0.01120	−0.24344	0.00046
31	1.35404	−3.20815	0.27546	3.77187	0.94516	−0.01694	−0.32501	0.00076
41	1.39954	−3.29613	0.29848	3.41337	0.91884	−0.01901	−0.39294	0.00098
50	1.45408	−3.40317	0.27863	3.20593	0.89473	−0.01692	−0.44505	.001109
				(b) Camera parameters for Fig. 4b				
1	1.00000	−2.00000	0.00000	5.00000	1.00000	0.00000	0.00000	0.00000
11	1.26354	−2.55113	0.03187	5.03110	0.99264	0.00175	−0.12052	0.00013
21	1.48100	−2.91775	0.05547	4.92033	0.97910	0.00315	−0.20249	0.00035
31	1.67344	−3.16255	0.08010	4.77705	0.96457	0.00384	−0.26279	0.00054
41	1.84954	−3.32806	0.10688	4.64016	0.95079	0.00387	−0.30871	0.00068
50	2.00000	−3.43635	0.13446	4.53059	0.93921	0.00332	−0.34222	0.00077
				(c) Camera parameters for Fig. 4c				
1	1.00000	−2.00000	0.00000	5.00000	1.00000	0.00000	0.00000	0.00000
11	1.17363	−2.01659	0.01329	4.88576	0.99987	0.00022	−0.01327	0.00009
21	1.29678	−2.03892	0.03225	4.50468	0.99951	−0.00015	−0.02458	0.00037
31	1.39584	−2.06444	0.05539	3.99351	0.99900	−0.00092	−0.03464	0.00080
41	1.52755	−2.09683	0.08607	3.63481	0.99838	−0.00218	−0.04282	0.00139
50	1.69409	−2.13590	0.12422	3.53793	0.99779	−0.00374	−0.04840	0.00205

trol), and the other generated from $\mathbf{x}_{f_e}$ toward $\mathbf{x}_{f_s}$ (backward control). The blended sequence interpolates $\mathbf{x}_{f_s}$ and $\mathbf{x}_{f_e}$ at the frames $t = f_s$ and f_e, respectively. (See [16, 18] for similar techniques to generate rotational and dynamic motion curves while interpolating given boundary conditions.) The pseudo code is given as follows:

Blend-Camera-Control $(f_s, f_e, \mathbf{x}_{f_s}, \mathbf{x}_{f_e}, H_{f_s}, H_{f_e})$
Input: f_s, f_e : the start and end keyframes;
$\mathbf{x}_{f_s}, \mathbf{x}_{f_e}$: the start and end keyframe camera parameters;
H_{f_s}, H_{f_e} : the start and end positions of the image control points;
Output: $\mathbf{x}_{f_s+1}, \ldots, \mathbf{x}_{f_e-1}$: a sequence of camera parameters;
begin
Camera-Control$(f_e, f_s, \mathbf{x}_{f_e}, H_{f_s})$
for $i = f_s + 1$ **to** $f_e - 1$ **do**
$\mathbf{y}_i = \mathbf{x}_{f_s+f_e-i}$;
Camera-Control$(f_s, f_e, \mathbf{x}_{f_s}, H_{f_e})$;
$t = 0$;
$\Delta t = \dfrac{1}{f_e - f_s}$;
for $i = f_s + 1$ **to** $f_e - 1$ **do**
begin
$t = t + \Delta t$;
$(\mathbf{x}_i)_f = (1 - \mathscr{B}(t))\ (\mathbf{x}_i)_f + \mathscr{B}(t)\ (\mathbf{y}_i)_f$;
$(\mathbf{x}_i)_u = (1 - \mathscr{B}(t))\ (\mathbf{x}_i)_u + \mathscr{B}(t)\ (\mathbf{y}_i)_u$;
$(\mathbf{x}_i)_q = Slerp((\mathbf{x}_i)_q, (\mathbf{y}_i)_q, \mathscr{B}(t))$;
end
end

In the above, $\mathscr{B}(t)$ is a smooth blending function such that: $\mathscr{B}(f_s) = 0$, $\mathscr{B}(f_e) = 1$, $0 \le \mathscr{B}(t) \le 1$, for $f_s \le t \le f_e$. The spherical linear interpolation [23] is defined by

$$Slerp(q_1, q_2, t) = \exp(t \log(q_2 \cdot q^{-1})) \cdot q_1,$$

which is the internal division of the geodesic arc from q_1 to q_2 in the ratio of $t : (1 - t)$. (See [7, 15, 16] for the definition of the exponential and logarithmic maps: exp and log.) The first *Camera-Control* in the above pseudo code is a backward control process which starts from $\mathbf{x}_{f_e}$ and has H_{f_s} as the end positions of the image control points. The end value of the backward control may be different from the start camera control parameter $\mathbf{x}_{f_s}$. The second *Camera-Control* generates a forward control sequence of camera parameters. By blending the two sequences (one forward and the other backward), we get a sequence of camera parameters which is continuously changing from $\mathbf{x}_{f_s}$ to $\mathbf{x}_{f_e}$. (See Fig. 8.)

When the result of the forward control is quite different from that of the backward control, the blended sequence of camera parameters generates the trace curves of image control points which are much deviated from the given curves H_i's. In this case, we can further postprocess the blended sequence of camera parameters so that it generates trace curves which fit tightly to the given H_i's by using the correction technique to be discussed in Section 7.2.

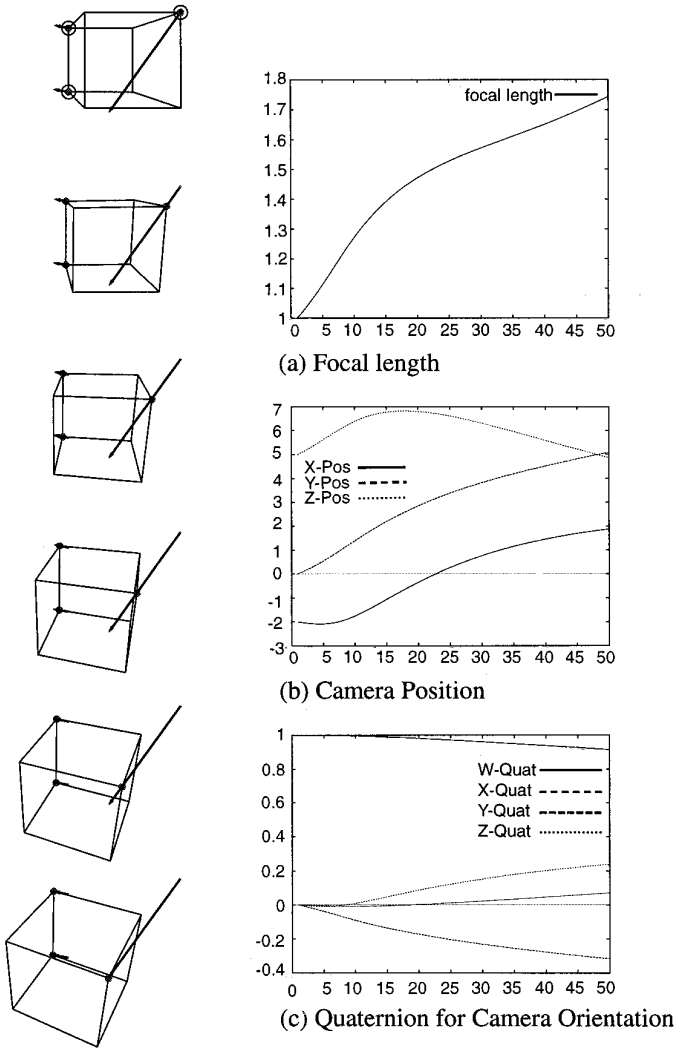

FIG. 5. Graphs of camera parameters.

7.2. *Correction to Frame Sequence of Camera Parameters*

Through-the-lens control technique can also be applied to modify the sequence of camera parameters which is generated by other virtual camera control methods. When the trace curves of image control points have complex shapes, the generated scene whirls or shakes dizzily. The awkwardness of the scene is related to the shape complexity of the trace curves. For example, in the case of shaking scenes, the trace curves have many wiggles and/or cusps. Therefore, by modifying the trace curves to those with simple shapes, we can facilitate a much smoother scene change. However, it is not easy to quantify the shape complexity of the trace curves and to search for the optimal curves which have the lowest shape complexity. To simplify the trace curve shape, we use *correction curves* to which the trace curves are to be modified. The correction curves are designed by the user. Then, using the gradient method, the trace curves are automatically modified to the correction curve shapes. For example, the simplest correction curve may be given as the straight line which connects the two end points of the trace curve.

Given a sequence of camera parameters, $X = \{\mathbf{x}_{f_s}, \mathbf{x}_{f_s+1}, \ldots, \mathbf{x}_{f_e}\}$, and an m-tuple of correction curves, $H_c(t)$, the energy function $E(X)$ is defined by

$$E(X) = \sum_{i=f_s+1}^{f_e-1} (E_i(\mathbf{x}_i) + k_c C_i(\mathbf{x}_i)),$$

where $E_i(\mathbf{x}_i) = \frac{1}{2}\|F_i(\mathbf{x}_i)\|^2$, with $F_i(\mathbf{x}_i) = V_P(\mathbf{x}_i) - H_c(t_i)$, and $C_i(\mathbf{x}_i)$ is the acceleration penalty function which gives penalty in proportion to the magnitude of the acceleration of the camera parameter, and k_c is the penalty weight. Figure 6 shows two sequences of the camera parameters. Curve (a) is obtained by minimizing the energy function $\sum_{i=f_s+1}^{f_e-1}(E_i(\mathbf{x}_i))$, while Curve (b) is obtained by minimizing the energy function $\sum_{i=f_s+1}^{f_e-1}(E_i(\mathbf{x}_i) + k_c C_i(\mathbf{x}_i))$. In Curve (a), there is a stiff interval where the camera parameter changes rapidly; whereas in Curve (b), the change is much slower. Stiff camera parameter change generates discontinuous scene change which may look awkward. The stiffness occurs when a point $\mathbf{x}_i$ on an energy hill is about to go down the hill far from the nearby points $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$, which should be prohibited. Thus, the penalty function $C_i(\mathbf{x}_i)$ is required to reduce the stiffness. The resulting algorithm is given as follows:

for $i = f_s + 1$ **to** $f_e - 1$ **step** 2 **do**
$\quad \mathbf{x}_i \leftarrow \mathbf{x}_i - \alpha(\nabla E_i(\mathbf{x}_i) + k_c \nabla C_i(\mathbf{x}_i))$;
for $i = f_s + 2$ **to** $f_e - 1$ **step** 2 **do**
$\quad \mathbf{x}_i \leftarrow \mathbf{x}_i - \alpha(\nabla E_i(\mathbf{x}_i) + k_c \nabla C_i(\mathbf{x}_i))$;

where ∇E_i and ∇C_i are the gradients of E_i and C_i, respectively (see Fig. 7), and α is a nonnegative scalar constant. The separation of the iteration into two loops increases the convergence rate of the trace curves. For each correc-

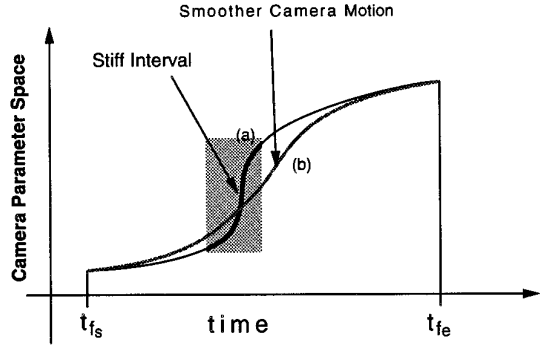

FIG. 6. Corrected sequences of camera parameters.

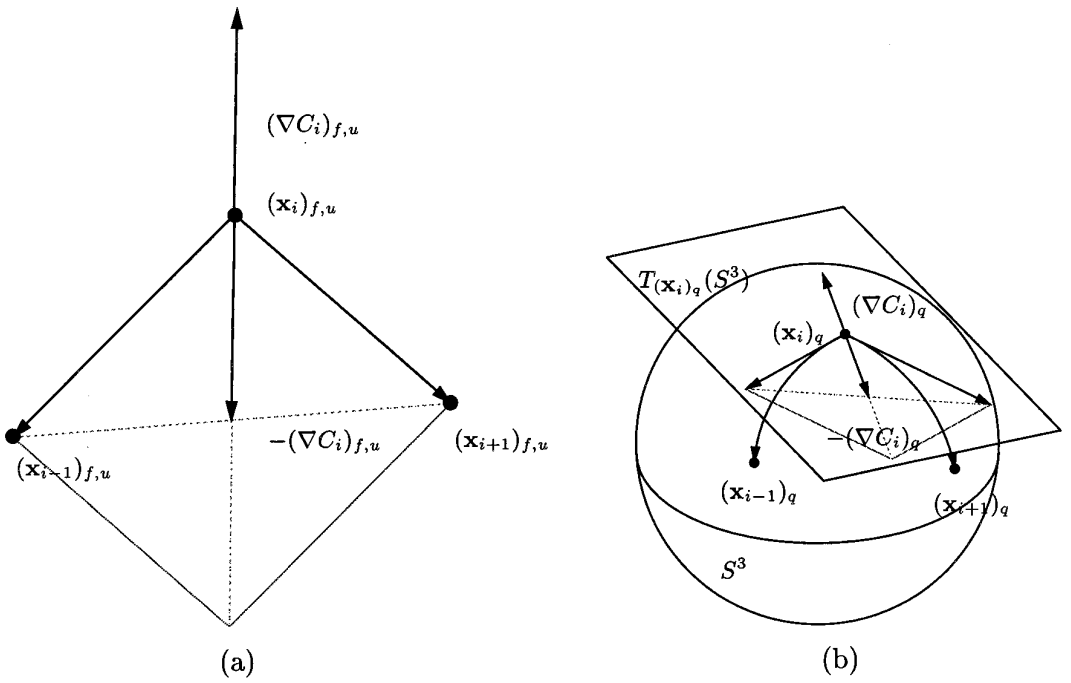

FIG. 7. Gradient of $C_i(\mathbf{x}_i)$ defined geometrically.

tion, we repeat the above two loops. The gradient ∇E_i is given by

$$\nabla E_i(\mathbf{x}_i) = \nabla\left(\frac{1}{2}\langle F_i(\mathbf{x}_i), F_i(\mathbf{x}_i)\rangle\right) = J(\mathbf{x}_i)^{\mathrm{T}} F_i(\mathbf{x}_i),$$

and the gradient $\nabla C(\mathbf{x}_i)$ is defined geometrically as

$$(\nabla C_i(\mathbf{x}_i))_f = -\frac{(f_{i-1} - f_i) + (f_{i+1} - f_i)}{2}$$

$$(\nabla C_i(\mathbf{x}_i))_u = -\frac{(u_{i-1} - u_i) + (u_{i+1} - u_i)}{2}$$

$$(\nabla C_i(\mathbf{x}_i))_q = -\frac{\log(q_{i-1}\cdot q_i^{-1}) + \log(q_{i+1}\cdot q_i^{-1})}{2},$$

where log is the logarithmic map defined on unit quaternions [7, 15, 16].

Figure 8 shows blending of forward and backward controls.

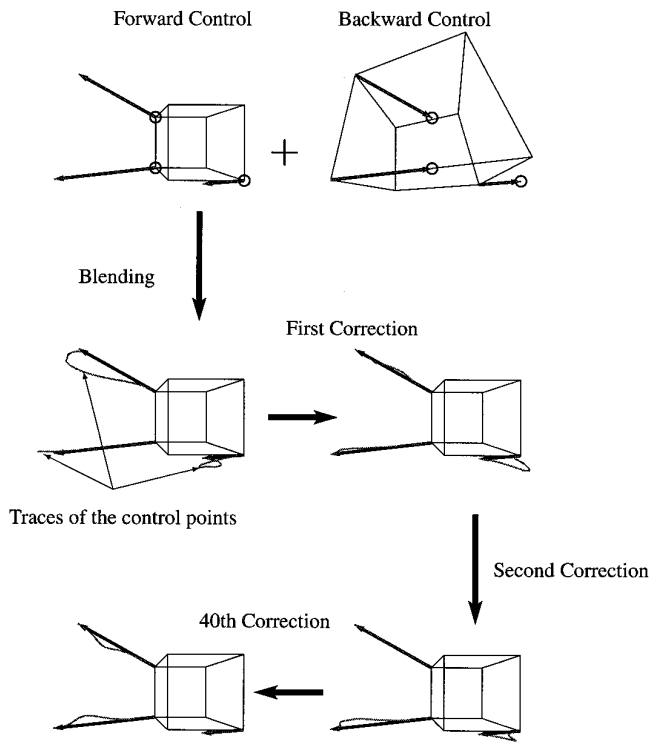

FIG. 8. Blending forward and backward controls.

7.3. *Degenerate Cases*

A serious problem in the integration of a tangent vector field is how to deal with the singularities of the vector field. Although virtual camera control is a relatively simple nonlinear inversion problem, we have observed some chaotic behaviors of the camera motion when the input speci-

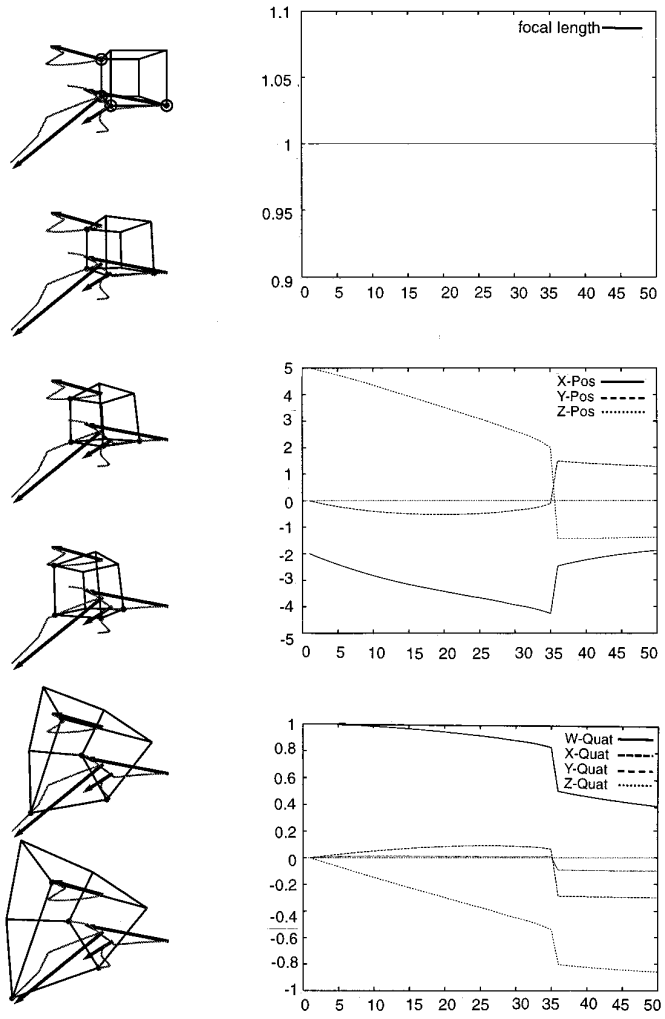

FIG. 9. Degenerate case with nonsmooth vector field.

fications are not given in proper way (see Fig. 9). In these degenerate cases, even the techniques discussed in Sections 7.1 and 7.2 do not produce smooth camera motions. The characterization of these phenomena may require more advanced mathematical tools from dynamical systems [14]. In Fig. 9, the gray curves show the actual trace curves of the image control points. There is an abrupt change of camera parameter in the middle of the control. We experimented with a fixed focal length. This is because the focal length easily diverges at singularity, which makes the visualization of the scene itself very difficult.

8. POSSIBLE EXTENSIONS

The basic result of this paper can be extended to other camera models and to other camera and motion control techniques. In this section, we briefly outline some possible extensions to these general camera models and motion control problems.

8.1. *Extensions to Other Camera Models*

There are three common representations for camera rotation:

1. the Euler angles modeled by R^3 or $S^1 \times S^1 \times S^1$,
2. the unit quaternions modeled by S^3, and
3. the Fibre bundle structure locally modeled by $S^2 \times S^1$ (see Shoemake [24]).

In general, each representation of camera rotation is modeled by a certain 3-dimensional manifold M and a parameterization map, F: $M \to SO(3)$, where $SO(3)$ is the 3D rotation group. Some basic requirements for the map F are:

- $F(M)$ covers the whole space $SO(3)$ (i.e., F is a surjective map),
- F is differentiable,
- F is locally invertible, and
- F preserves the metric properties of M to $SO(3)$, and vice versa (i.e., F is a local isometry).

Unfortunately, only the unit quaternion space S^3 satisfies all these criteria. The other two models R^3 and $S^2 \times S^1$ satisfy:

- only the first and second conditions perfectly,
- the third condition almost everywhere except some singularities, and
- the last condition only near the identify of M.

The singularities in the two representations, R^3 and $S^2 \times S^1$, have two different origins. The Euler angle representation has singularity called *gimbal lock* at which the mapping F is many-to-one, whereas the fiber bundle structure $S^2 \times S^1$ has singularity resulting from a certain limitation in the differential structure of S^2.

We discuss more details on the gimbal lock of Euler angles. Let ψ, θ, ϕ denote the pan, tilt, and roll angles, respectively, defined as follows (see Drucker [9]):

1. pan: Rotation of the camera about the vertical axis,
2. tilt: Rotation of the camera about the lateral axis, and
3. roll: Rotation of the camera about the viewing direction.

(These are cinematic terms [9]; in engineering, the pan, tilt, and roll angles are usually called yaw, pitch, and roll angles, respectively.) Then, $F(\psi, \theta, \phi)$ represents the resulting camera rotation which is obtained by applying the three component rotations: pan, tilt, and roll, in that order. The gimbal lock occurs when we have the tilt angle $\theta = \pm\pi/2$, that is, when the camera viewing direction is parallel/opposite to the global view-up vector. In this case, for a fixed value of α, the different Euler angles $(\psi, \pm\pi/2, \alpha - \psi)$, $-\pi < \psi \leq \pi$, represent the same camera rotation. These

Euler angles define a curve embedded in R^3 (equivalently, a great circle embedded in $S^1 \times S^1 \times S^1$):

$$C_\alpha(\psi) = (\psi, \pm\pi/2, \alpha - \psi), \quad \text{for } -\pi < \psi \leq \pi.$$

Then the reparameterization map, F: $R^3 \rightarrow SO(3)$, transforms the whole 1-dimensional curve C_α into a single point in $SO(3)$ (i.e., into an identical 3D rotation). Consequently, the linear differential

$$dF_{(\psi,\pm\pi/2,\alpha-\psi)}: R^3 \rightarrow T_{F(p)}(SO(3)),$$

has rank deficiency, which implies that the map F itself is locally non-invertible. Near such a singularity, the behavior of F becomes irregular.

The singularity in the fibre bundle structure $S^2 \times S^1$ results from the limitation in the differential structure of S^2. As we have discussed in Section 3.1, the Lie group structure of S^3 provides a canonical coordinate system on each tangent space $T_q(S^3)$ so that any tangent vector $v_q \in T_q(S^3)$ can be identified with a tangent vector $v_1 \in T_1(S^3)$ such that $v_q = v_1 \cdot q$ or $v_1 = v_q \cdot q^{-1}$. The spherical Lie groups S^1 and S^7 also have this property. However, in the case of S^2, it is quite well-known that there is no possible way to define a nowhere-vanishing smooth tangent vector field on S^2 [25]. This implies that it is impossible to define a coordinate system on each tangent plane of $T_p(S^2)$, where $p \in S^2$, so that the coordinate system changes smoothly on S^2. There is at least one singular point on S^2 at which the neighboring coordinate systems change quite dramatically.

The local fibre bundle structure $S^2 \times S^1$ represents the camera viewing direction by the spherical component S^2, and the rolling angle (i.e., the rotation about the camera viewing direction) by the circular component S^1. Unfortunately, the fibre bundle representation, when interpreted globally, has a singularity at $(-1, 0, 0) \in S^2$; that is, when the camera viewing direction is the opposite to that of the standard camera orientation, there is no well-defined representation for the camera orientation. For example, when the camera has an orientation corresponding to the Euler angle $(\psi, \theta, \phi) = (\pi, 0, \phi)$, for some angle ϕ, its fiber bundle representation has $(-1, 0, 0)$ as the spherical component; however, no matter what angle we specify for the circular component, there is no continuity with the fiber bundle representations of its neighboring points. This phenomenon is also closely related with the fact that there are infinitely many great circles which connect the two antipodal points $(1, 0, 0)$ and $(-1, 0, 0) \in S^2$.

A simple technique to deal with the singularities in the map, F: $M \rightarrow SO(3)$, is to use two maps, F_i: $M_i \rightarrow SO(3)$, for $i = 1, 2$, so that each singularity of F_1 is covered by a nonsingularity of the other map F_2, and vice versa. Except the regions near the singularities, each map F_i $(i = 1, 2)$ performs reasonably well. Although the Jacobian matrices are much more complex than the case of S^3, they are computable and locally invertible. Therefore, by switching between the two maps F_1 and F_2 if necessary, it is possible to extend our approach to the through-lens-camera control represented by other camera models such as the Euler angle and the fiber bundle structure.

8.2. *Extensions to Other Camera and Motion Control Techniques*

The camera control technique of Drucker [9] is formulated as a constrained nonlinear optimization problem, which is then solved by the Sequential Quadratic programming (SQP) technique (see [5]). In this section, we review the SQP formulation as an $(m + 7) \times (m + 7)$ square matrix equation, where m is the number of constraints. At the same time, we discuss some limitations of the SQP approach in the camera control problem and suggest a reformulation of the nonlinear optimization as a target tracking problem.

There is an important distinction between the two usages of the SQP formulation in Cohen [5] and Drucker [9]. In the space–time control of animation [5], the optimal solution obtained from the SQP formulation is a natural-looking animation which satisfies all the dynamic as well as kinematic constraints specified for the moving mechanism. That is, the final solution is a motion curve, not simply the final end point of the motion curve. The intermediate values of the state variable $\mathbf{x}$ in the optimization process are not directly related with the resulting motion; only the final optimal solution produces the motion. Therefore, in the middle of the optimization, the deviation from some constraints is acceptable as long as the final solution satisfies all the constraints specified. Cohen [5] approximates the motion curve by a piecewise cubic B-spline curve with a finite number (say, k) of B-spline control points. Therefore, the corresponding SQP formulation is given by an $(m + 7k) \times (m + 7k)$ square matrix equation.

In the camera control of Drucker [9], the objective functions are specified mainly for the desired final state of the optimization process, whereas the constraint equations are required to be satisfied all the time. The resulting camera motion is obtained as the sequence of intermediate values of the state variable $\mathbf{x}$ in the optimization process. Therefore, it is quite natural to interpret this solution process as a constrained target tracking procedure for the final goal. The constraint equations are required to be satisfied (or approximated in the least squares sense) all the time. In this respect, the SQP formulation of Cohen [5] is not quite appropriate for the solution process outlined in Drucker [9]. We discuss more details below.

Sequential Quadratic Programming. Given an objective function $f(\mathbf{x})$ and m constraints $C_i(\mathbf{x})$, the Lagrange equation is formulated as

$$\nabla f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \nabla C_i(\mathbf{x}) = 0, \tag{32}$$

for some Lagrange multipliers λ_i, for $i = 1, \ldots, m$. When we use the following (column) vector notations,

$$\lambda = (\lambda_1, \ldots, \lambda_m)^{\mathrm{T}}, \quad \text{and}$$
$$\nabla C(\mathbf{x}) = (\nabla C_1(\mathbf{x}), \ldots, \nabla C_m(\mathbf{x}))^{\mathrm{T}},$$

the summation term in Eq. (32) is simplified to

$$\sum_{i=1}^{m} \lambda_i \nabla C_i(\mathbf{x}) = \lambda^{\mathrm{T}} \cdot \nabla C(\mathbf{x}). \tag{33}$$

The constrained Lagrange equation is then formulated in the following compact vector equation:

$$\nabla L(\mathbf{x}, \lambda) = \nabla f(\mathbf{x}) + \lambda^{\mathrm{T}} \cdot \nabla C(\mathbf{x}) = 0, \tag{34}$$
$$\text{subject to} \quad C(\mathbf{x}) = 0. \tag{35}$$

While considering $\mathbf{x}$ and λ as independent variables, we can take the first derivative of Eq. (34) and obtain the equation

$$\begin{aligned} &\nabla^2 L(\mathbf{x}, \lambda) \cdot (\dot{\mathbf{x}}, \dot{\lambda})^{\mathrm{T}} \\ &\quad = \nabla^2 f(\mathbf{x}) \cdot \dot{\mathbf{x}} + \lambda^{\mathrm{T}} \cdot \nabla^2 C(\mathbf{x}) \cdot \dot{\mathbf{x}} + \nabla C(\mathbf{x})^{\mathrm{T}} \cdot \dot{\lambda} \\ &\quad = [\nabla^2 f(\mathbf{x}) + \lambda^{\mathrm{T}} \cdot \nabla^2 C(\mathbf{x}) \; \nabla C(\mathbf{x})^{\mathrm{T}}] \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\lambda} \end{bmatrix} = 0, \end{aligned} \tag{36}$$

where $\nabla^2 L(\mathbf{x}, \lambda)$ and $\nabla^2 f(\mathbf{x})$ are the Jacobian matrices of the vector valued nonlinear maps $\nabla L(\mathbf{x}, \lambda)$ and $\nabla f(\mathbf{x})$, respectively. Moreover, from Eq. (33), the term $\lambda^{\mathrm{T}} \cdot \nabla^2 C(\mathbf{x})$ is given by the summation

$$\lambda^{\mathrm{T}} \cdot \nabla^2 C(\mathbf{x}) = \sum_{i=1}^{m} \lambda_i \nabla^2 C_i(\mathbf{x}),$$

where $\nabla^2 C_i(\mathbf{x})$ is the Jacobian matrix of $\nabla C_i(\mathbf{x})$. The first derivative of Eq. (35) produces the following constraint equation for the first derivative vector $\dot{\mathbf{x}}$:

$$\nabla C(\mathbf{x})^{\mathrm{T}} \cdot \dot{\mathbf{x}} = 0. \tag{37}$$

This is equivalent to

$$\nabla C_i(\mathbf{x})^{\mathrm{T}} \cdot \dot{\mathbf{x}} = 0, \quad \text{for } i = 1, \ldots, m.$$

That is, the derivative vector $\dot{\mathbf{x}}$ is orthogonal to each gradient vector $\nabla C_i(\mathbf{x})$, for $i = 1, \ldots, m$.

Assume that the current camera parameter $\mathbf{x}$ satisfies the hard constraint $C(\mathbf{x}) = 0$, but not the necessary condition $\nabla L(\mathbf{x}, \lambda) = 0$, for the optimal solution of $f(\mathbf{x})$. For an optimal camera control, we want to move the current camera parameter $\mathbf{x}$ into the next one $\mathbf{x} + \dot{\mathbf{x}}$ so that the value of ∇L vanishes at $\mathbf{x} + \dot{\mathbf{x}}$:

$$\nabla L(\mathbf{x} + \dot{\mathbf{x}}, \lambda + \dot{\lambda}) \approx \nabla L(\mathbf{x}, \lambda) + \nabla^2 L(\mathbf{x}, \lambda)(\dot{\mathbf{x}}, \dot{\lambda})^{\mathrm{T}} = 0. \tag{38}$$

In case the hard constraint $C(\mathbf{x}) = 0$ is slightly deviated, we want to make the value of C also vanish at $\mathbf{x} + \dot{\mathbf{x}}$ as well:

$$C(\mathbf{x} + \dot{\mathbf{x}}) \approx C(\mathbf{x}) + \nabla C(\mathbf{x})^{\mathrm{T}} \cdot \dot{\mathbf{x}} = 0. \tag{39}$$

Note that, when the constraint $C(\mathbf{x}) = 0$ is satisfied perfectly, this equation reduces to that of Equation (37). By combining Eqs. (38)–(39), we can formulate the following $(m + 7) \times (m + 7)$ square matrix equation (see [5]):

$$\begin{bmatrix} \nabla^2 f(\mathbf{x}) + \lambda^{\mathrm{T}} \cdot \nabla^2 C(\mathbf{x}) & \nabla C(\mathbf{x})^{\mathrm{T}} \\ \nabla C(\mathbf{x})^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla f(\mathbf{x}) - \lambda^{\mathrm{T}} \cdot \nabla C(\mathbf{x}) \\ -C(\mathbf{x}) \end{bmatrix}. \tag{40}$$

Equivalently,

$$\begin{bmatrix} \nabla^2 f(\mathbf{x}) + \lambda^{\mathrm{T}} \cdot \nabla^2 C(\mathbf{x}) & \nabla C(\mathbf{x})^{\mathrm{T}} \\ \nabla C(\mathbf{x})^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}} \\ \lambda + \dot{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla f(\mathbf{x}) \\ -C(\mathbf{x}) \end{bmatrix}. \tag{41}$$

When we apply the space–time control technique of Cohen [5] to the camera control problem, the camera motion curve is approximated by a piecewise cubic B-spline curve with k control points. The state variable $\mathbf{x}$ is thus a k-tuple $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$, where each $\mathbf{x}_i$ is the camera configuration at time t_i, for $i = 1, \ldots, k$, and $t_1 < \cdots < t_k$. Therefore, Eq. (41) becomes an $(m + 7k) \times (m + 7k)$ square matrix equation. Since the camera motion curve $\mathbf{x}(t)$ is represented by a cubic B-spline curve with control points $\{\mathbf{x}_i\}$, its velocity curve $\dot{\mathbf{x}}(t)$ is given by a quadratic B-spline curve with the control points of the form $\{3(\mathbf{x}_{i+1} - \mathbf{x}_i)\}$, for $i = 1, \ldots, k - 1$. Therefore, we can specify dynamic as well as kinematic constraints using algebraic constraint equations: $C_i(\mathbf{x}) = 0$, for $i = 1, \ldots, m$.

The optimal solution of the SQP formulation would produce a natural-looking camera motion which satisfies all the dynamic as well as kinematic constraints specified. When there is no camera motion which satisfies all the constraints, the constraint deviation is formulated as a penalty to the objective function. The penalty function has a physical meaning as the external energy exerted to the camera system so that it could follow the given constraints

as closely as possible. Therefore, the resulting optimal solution provides a pseudo physical solution which is doing its best to follow the physical laws and other keyframe conditions as closely as possible.

Limitations of the SQP Formulation in Target Tracking. The SQP formulation and its solution process in Drucker [9] do not use the high-dimensional space-time approach of Cohen [5]. For efficiency reasons, Drucker [9] uses a 7-dimensional state variable $\mathbf{x}$ to represent the camera configuration at a variable time t. The camera motion is controlled by following the trace curve of $\mathbf{x}$ which is generated in the nonlinear optimization process, i.e., the sequence of approximate solutions of $\mathbf{x}$ in the Newton approximation of the Lagrange equation: $\nabla L(\mathbf{x}, \lambda) = 0$.

The SQP formulation has some shortcomings when it is used in the solution process of a constrained target tracking problem. First of all, the intermediate solutions of $\mathbf{x}$ may be allowed to have some deviations from the constraint equations: $C_i(\mathbf{x}) = 0$, for $i = 1, \ldots, m$. This is unacceptable for hard constraints. Therefore, at each step of the optimization, we first need to solve Eq. (39) and then proceed to solve Eq. (41) under the condition: $C(\mathbf{x}) = 0$. However, even this approach does not facilitate an efficient algorithm. We illustrate some more details below.

The matrix equation has second order partial derivatives of the objective function $f(\mathbf{x})$ and the constraint equations $C_i(\mathbf{x}) = 0$, for $i = 1, \ldots, m$. Since the overall computation is intended to solve the first derivative vector $\dot{\mathbf{x}}$, the second order derivative information is quite redundant. When the gradient vectors $\nabla C_i(\mathbf{x})$ and the Hessian matrices $\nabla^2 C_i(\mathbf{x})$ are available, for $i = 1, \ldots, m$, one should be able to locally approximate the constraint space $C(\mathbf{x}) = 0$ by a $(7 - m)$-dimensional parametric hypersurface in R^7. (For example, when two implicit surfaces are given in R^3, their intersection curve can be locally approximated by a cubic parametric curve. The curvature and torsion of the intersection curve can be evaluated based on the two surface gradient vectors and their Hessian matrices [2].) Subsequently, in the local neighborhood, the nonlinear optimization problem in R^7 with m constraints can be reduced to a nonlinear optimization problem in R^{7-m} with no constraint.

When we restrict the problem to the computation of $\dot{\mathbf{x}}$, we have to search the vector $\dot{\mathbf{x}}$ in the orthogonal space of the m vectors $\nabla C_i(\mathbf{x})$ in R^7, $i = 1, \ldots, m$. Therefore, the solution space of $\dot{\mathbf{x}}$ is constrained to R^{7-m} defined by the m constraints $C_i(\mathbf{x})^{\mathrm{T}} \cdot \dot{\mathbf{x}} = 0$, for $i = 1, \ldots, m$. However, the SQP formulation has an $(m + 7) \times (m + 7)$ square matrix equation, which becomes the larger as the more constraints are used. This is quite counter-intuitive to our interpretation of the constraint space as a $(7 - m)$-dimensional manifold.

In Equation (40), the vector

$$\lambda^{\mathrm{T}} \cdot \nabla C(\mathbf{x}) = \nabla C(\mathbf{x})^{\mathrm{T}} \cdot \lambda$$

is in the column space of the $7 \times m$ matrix $\nabla C(\mathbf{x})^{\mathrm{T}}$. Therefore, the column space projection of the vector in the right-hand side of Eq. (40) corresponds to that of Eq. (41), and vice versa. The solution spaces of Eq. (40)–(41) produce the same space for $(\dot{\mathbf{x}}, \dot{\lambda})$. However, due to the translation by $(0, \lambda)^{\mathrm{T}}$ in Eq. (41), the least-squares solutions in Eq. (40)–(41) may produce different solutions for $\dot{\mathbf{x}}$ and λ. When the magnitude of λ is large, this difference may influence the selection of $\dot{\mathbf{x}}$ quite significantly.

The Lagrangian in classical mechanics is given as a functional $L(\mathbf{x}, \dot{\mathbf{x}})$ of the state variable $\mathbf{x}$ and its time derivative $\dot{\mathbf{x}}$ (see [1]). There are two components which comprise the Lagrangian $L(\mathbf{x}, \dot{\mathbf{x}})$; one is the conservative energy term $U(\mathbf{x})$ (which is a function of $\mathbf{x}$) and the other is the kinetic energy term $T(\dot{\mathbf{x}}) = \frac{1}{2}\|\dot{\mathbf{x}}\|^2$ (which is a function of $\dot{\mathbf{x}}$). In contrast to this generic formulation of the total energy in a physical system, the Lagrangian $L(\mathbf{x}, \lambda)$ of the SQP formulation is given by

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^{\mathrm{T}} \cdot C(\mathbf{x}),$$

for which the physical meaning is not clear, especially due to the extra variable λ.

In the SQP formulation of Cohen [5], the physical meaning is specified as constraint equations. Therefore, as long as all the physical constraints are satisfied, the resulting motion (as the solution of the optimization process) follows the physical law, while minimizing the external energy specified as the objective function. In the space-time control of animation in Cohen [5], the intermediate approximate solutions in the nonlinear optimization are not directly related to the final animation. Therefore, the deviation from the constraint equations does not cause any problem as long as the deviation converges to 0 as the approximation approaches the final solution. However, in the constrained target tracking procedure such as the approach outlined in Drucker [9], the intermediate approximate solutions in the nonlinear optimization produce the resulting camera motion under consideration. Therefore, the hard constraints must be satisfied at each step of the optimization. This requirement also restricts the solution space of $\dot{\mathbf{x}}$ to a $(7 - m)$-dimensional subspace.

Target Tracking. The SQP formulation allows a single objective function, whereas Drucker [9] specifies multiple objective functions. Therefore, the optimization is solved as a minimax problem in which the maximum of the multiple objective functions is minimized. Under such a minimax strategy, a specific objective function comes into effect only when it has the largest value among many others; therefore, it is somewhat difficult to synchronize the interactions among different objective criteria. For this purpose of synchronization, we may formulate a single total objective function by a weighted summation of the multiple objective

functions. Different weighting factors have control over the relative contribution of each individual objective function. However, by combining different objective functions into one scalar valued objective function, it is quite difficult to resolve the conflicts among different objectives. To measure the influence of each individual objective function more precisely, we can map each objective function into a separate component axis in the range of the nonlinear map, $F: M \to R^n$. The corresponding Jacobian matrix represents the relationship between the camera motion velocity and the instantaneous changes of the objective functions. The question is how to specify the target vector. Differently from the through-the-lens control, the explicit target value may not be given as an explicit input in this approach.

In the core of every nonlinear optimization procedure is a target tracking algorithm which moves the current status into the next one so that the corresponding objective function changes to the negative direction. Therefore, in principle, it is possible to reformulate the nonlinear optimization problem of Drucker [9] into a target tracking problem by taking the negative objective direction as the target direction. It is also possible to generate other target directions corresponding to various constraints. We suggest a simple approach. More sophisticated formulations may be possible; however, the results would be highly tailored nonlinear optimization procedures, the developments of which are beyond the scope of this paper.

Given a camera model represented by a manifold M, the camera control problem can be represented by a nonlinear map, $F: M \to R^{m_1+m_2+m_3}$, where we assume m_1 objective functions, m_2 interval constraints, and m_3 hard constraint equations. We can solve the optimization problem by integrating a sequence of least-squares solutions, $\dot{\mathbf{x}} = (dF_{\mathbf{x}})^+ (v_{F(\mathbf{x})}) \in T_{\mathbf{x}}(M)$, where $v_{F(\mathbf{x})} \in T_{F(\mathbf{x})}$ $(R^{m_1+m_2+m_3})$ is the target vector. The mapping F is given as follows:

1. each objective function maps into R (one of the first m_1 components of $R^{m_1+m_2+m_3}$),

2. each interval or inequality constraint function maps into a bounded or semi-infinite interval of R (one of the second m_2 components of $R^{m_1+m_2+m_3}$), and

3. each hard constraint function maps into $\{0\}$ of R (one of the last m_3 components of $R^{m_1+m_2+m_3}$).

Given the current camera parameter $\mathbf{x} \in M$, we generate a tangent vector $v_{F(\mathbf{x})} = (\alpha_1, \ldots, \alpha_{m_1}, \beta_1, \ldots, \beta_{m_2}, \gamma_1, \ldots, \gamma_{m_3}) \in T_{F(\mathbf{x})}$ $(R^{m_1+m_2+m_3}) \equiv R^{m_1+m_2+m_3}$, where α_i is a negative value for each objective function, β_j is a value determined by the relative position of the value $F_{m_1+j}(\mathbf{x})$ in the constraint interval, and γ_k is $-F_{m_1+m_2+k}(\mathbf{x})$ for each hard constraint, where $F_l(\mathbf{x})$ is the lth component function of $F(\mathbf{x})$, for $l = 1, \ldots, m_1 + m_2 + m_3$. Then a tangent vector field $\dot{\mathbf{x}} \in T_{\mathbf{x}}(M)$ is generated by $\dot{\mathbf{x}} = (dF_{\mathbf{x}})^+ (v_{F(\mathbf{x})}) \in T_{\mathbf{x}}(M)$. In the environment of constrained optimization, the target vectors for the hard constraints are more important than other constraints. The row weighting scheme for the Jacobian matrix can be used to control the relative importance of each component of the target direction $v_{F(\mathbf{x})}$.

9. CONCLUSION

Through-the-lens camera control is a constrained nonlinear inversion problem. Given a nonlinear transformation, $F: M \to N$, where M and N are constraint spaces, the question is how to generate a tangent vector field on M. The integral curve of the vector field provides the required control. We used the pseudo inverse $(dF_p)^+$ to generate the vector field. The Lie group structure of S^3 greatly simplifies the Jacobian matrix representation of the linear differential, $dF_p: T_p(M) \to T_{F(p)}(N)$. The nonlinear inversion scheme presented here provides a general framework which is applicable to a wide variety of problems in computer animation and motion control in general. Therefore, the major contribution of this paper is in analyzing the mathematical structure of through-the-lens camera control. The current analysis, however, is quite elementary.

To interpolate a given sequence of discrete points on M, the integral curve approach using the tangent vector field provides only a C^0-continuous curve on M even if we use the curve blending technique on M. This is due to the singularities of the tangent vector field. Therefore, the problem essentially reduces to that of dynamical systems [14]. There are still many important open problems that must be solved in order to further develop the theory.

ACKNOWLEDGMENTS

The authors thank the anonymous referees for their careful reviews and constructive criticisms. Dr. Miyoung Lee of the Department of Mathematics at Purdue University informed the authors that, in practice, the conjugate gradient method usually works for positive semidefinite matrices even though, in theory, it is guaranteed to work only for positive definite matrices.

REFERENCES

1. V. I. Arnold, *Mathematical Methods of Classical Mechanics,* 2nd ed., Springer-Verlag, New York, 1989.
2. C. Bajaj, C. Hoffmann, R. Lynch, and J. Hopcroft, Tracing surface intersections, *Comput. Aided Geom. Design* **5,** 1988, 285–307.
3. J. Blinn, Where am I? What am I looking at?, *IEEE Comput. Graphics Appl.* **8**(7), 1988, 76–81.
4. R. Burden and J. D. Faires, *Numerical Analysis,* 4th ed., PWS-KENT, Boston, 1989.
5. M. Cohen, Interactive spacetime control of animation, *Comput. Graphics (Proc. of SIGGRAPH '92)* **26**(2), 1992, 293–302.
6. S. Conte and C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach,* 3rd ed., McGraw–Hill, Singapore, 1981.
7. M. Curtis, *Matrix Groups,* Springer-Verlag, New York, 1979.

8. M. do Carmo, *Differential Geometry of Curves and Surfaces,* Prentice–Hall, Englewood Cliffs, NJ, 1976.
9. S. Drucker, *Intelligent Camera Control for Graphical Environments,* Ph.D thesis, Program in Media Arts and Sciences, MIT, 1994.
10. J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice,* 2nd ed., Addison–Wesley, Reading, MA, 1990.
11. M. Gleicher, *A Differential Approach to Graphical Interaction,* Ph.D thesis, School of Computer Science, Carnegie Mellon Univ., 1994.
12. M. Gleicher and A. Witkin, Through-the-lens camera control, *Comput. Graphics (Proc. of SIGGRAPH '92)* **26**(2), 1992, 331–340.
13. G. Golub and C. Van Loan, *Matrix Computations,* Johns Hopkins Univ. Press, Baltimore, 1983.
14. M. Hirsh and S. Smale, *Differential Equations, Dynamical Systems, and Linear Algebra,* Academic Press, New York, 1974.
15. M.-J. Kim, M.-S. Kim, and S. Shin, A compact differential formula for the first derivative of a unit quaternion curve, *J. Visualization and Computer Animation,* **7**(1), 1996, 43–57.
16. M.-S. Kim and K.-W. Nam, Interpolating solid orientations with circular blending quaternion curves, *Comput. Aided Design* **27**(5), 1995, 385–398.
17. A. Lamouret, M. Gascuel, and J. Gascuel, Combining physically-based simulation of colliding objects with trajectory control," *J. Visualization and Computer Animation* **6**(2), 1995, 71–90.
18. J. H. Lee and M.-S. Kim, Pseudo dynamic keyframe animation with motion blending on the configuration space of a moving mechanism, in *Computer Graphics and Applications* (S. Y. Shin and T. L. Kunii, Eds.), pp. 118–132, World Scientific, Singapore, 1995. [*Proc. of Pacific Graphics '95,* Seoul, Korea, Aug. 21–24, 1995]
19. J. Marion and S. Thornton, *Classical Dynamics of Particles and Systems,* 3rd ed., Harcourt Brace Jovanovich, Orlando, FL, 1988.
20. J. Milnor, *Morse Theory,* Princeton Univ. Press, Princeton, NJ, 1963.
21. N. Papanikolopoulos, B. Nelson, and P. Khosla, Six degree-of-freedom hand/eye visual tracking with uncertain parameters, *IEEE Trans. Robotics and Automat.* **11**(5), 1995, 725–732.
22. W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes,* Cambridge Univ. Press, Cambridge, UK, 1986.
23. K. Shoemake, Animating rotation with quaternion curves, *Comput. Graphics (Proc. of SIGGRAPH '85)* **19**(3), 1985, 245–254.
24. K. Shoemake, Fibre bundle twist reduction, in *Graphics Gems IV* (P. Heckbert, Ed.), pp. 230–236, Academic Press, Boston, 1994.
25. G. Spivak, *A Comprehensive Introduction to Differential Geometry,* Vol. I, Publish or Perish, Berkeley, 1970.
26. G. Strang, *Linear Algebra and its Applications,* 3rd ed., Harcourt Brace Jovanovich, Orlando, FL, 1988.
27. S. Upstill, *The RenderMan Companion,* Addison–Wesley, Reading, MA, 1990.
28. J. Wittenburg, *Dynamics of Systems of Rigid Bodies,* Teubner, Stuttgart, 1977.