In *Proc. of Graphics Interface '95, pp. 171–178.*

Through-the-Lens Camera Control with a Simple Jacobian Matrix

Min-Ho Kyung, Myung-Soo Kim*,○, and Sung Je Hong**

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

○ Dept. of Computer Science, Purdue University, W. Lafayatte, IN47907, USA

Abstract

This paper improves both the computational efficiency and numerical stability of the through-the-lens camera control [5]. A simple $2m \times 7$ Jacobian matrix is derived. The matrix equation is then solved using an efficient weighted least squares method while employing the singular value decomposition method for numerical stability.

Keywords: virtual camera control, quaternion calculus, Jacobian matrix

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-from/view-up represent one of the most popular virtual camera models [4]. The camera status is usually described by three parameters: focus, position, and orientation. The focus is represented by a single value, i.e., the focal length. Each position and rotation has three degrees of freedom (DOF). Then the user controls the virtual camera with seven DOFs. However, it is not easy to control all the seven camera parameters simultaneously; most user interfaces (e.g., mouse) do not support all the required seven DOFs at the same time.

Gleicher and Witkin [5] suggested the *through-the-lens camera control* scheme to provide a general solution to the virtual camera control problem. 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 in the way the user has specified on the screen. That is, when the user selects 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 into the new 2D points. In Figure 1, the virtual camera is in the position Eye1 and the given 3D point is projected into 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 into the new position B with the new virtual camera at the position Eye2.

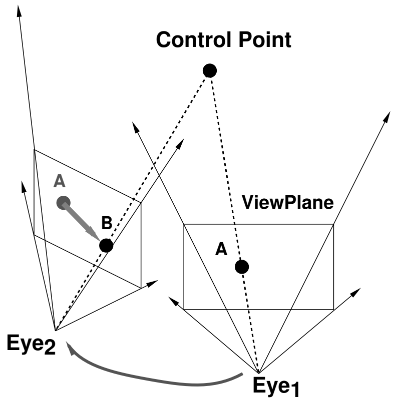

Figure 1: Through-the-lens Camera Control

The through-the-lens camera control provides a very powerful user interface to the virtual camera control. However, the constrained nonlinear optimization technique of Gleicher and Witkin [5] has limitations in both computational efficiency and numerical stability. For an overconstrained case with m image control points (with $m \geq 4$), the Lagrange equation is formulated as a $2m \times 2m$ square matrix equation, which takes $O(m^3)$ time to be solved. (The square matrix is also singular when $m \geq 4$.)

In this paper, we suggest some improvements. First of all, a simple $2m \times 7$ Jacobian matrix is derived using the quaternion calculus [7, 8, 11]. Instead of using a nonlinear optimization technique, we use an efficient weighted least squares method while employing the singular value decomposition for numerically stability [6, 10, 13]. The time complexity grows only linearly, i.e., $O(m)$ time for m control points.

The rest of this paper is organized as follows. In Section 2, we review the previous method [5]. Section 3 introduces a simple Jacobian matrix. In Section 4, matrix equations are derived for the through-the-lens camera control. They are solved in Section 5 using the weighted least squares method. The implementation details and experimental results are discussed in Section 6. Finally, Section 7 concludes this paper.

2 Previous Work

2.1 Review on the Previous Work

Most virtual camera models have seven degrees of freedom: i.e., one for the focal length, and three for each position and orientation. In representing the orientations, the unit quaternions are quite useful since they are free of singularities such as gimbal lock [8, 11, 14]. Each unit quaternion consists of four parameters (q_w, q_x, q_y, q_z) with the constraint: $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$. When quaternions are used, a total of eight parameters instead of seven parameters are required to represent the status of a virtual camera.

Given m points $p_1, \ldots, p_m \in R^3$ and eight camera parameters $(f, t_x, t_y, t_z, q_w, q_x, q_y, q_z) \in R^8$, the perspective projection in the viewing transformation for the m points, $V : R^8 \to R^{2m}$, is defined by:

$$V(f, t_x, t_y, t_z, q_w, q_x, q_y, q_z) = (h_1, \ldots, h_m) \in R^{2m},$$

where each $h_i = (x_i, y_i) \in R^2$ $(i = 1, \ldots, m)$ is the perspective projection of p_i onto the 2D display screen. The perspective projection V produces a nonlinear relationship between the camera control parameters and the projected 2D image points. Thus, it is very difficult to construct even a local inverse map from the image space to the camera parameter space.

Gleicher and Witkin [5] solved this inverse problem by approximating the nonlinear inverse problem with a sequence of linear inverse problems. Each linear equation is obtained by differentiating the nonlinear equation between the camera parameters and the 2D image points; the linear equation gives the relationship between the velocities of the camera parameters and the velocities of the control points in the image space. That is, let J be the $2m \times 8$ Jacobian matrix of the perspective transformation V, and furthermore, let $\mathbf{x} = (f, t_x, t_y, t_z, q_w, q_x, q_y, q_z) \in R^8$, and $\mathbf{h} = (x_1, y_1, \ldots, x_m, y_m) \in R^{2m}$. The relationship between $\dot{\mathbf{h}}$ and $\dot{\mathbf{x}}$ can now be represented by a simple matrix equation

$$\dot{\mathbf{h}} = J\dot{\mathbf{x}}$$

Given an initial velocity $\dot{\mathbf{h}}_0 \in R^{2m}$ for the m control points in the image space and a specified value of $\dot{\mathbf{x}}_0 \in R^8$, Gleicher and Witkin [5] solved the nonlinear optimization problem which minimizes the quadratic energy function:

$$E = \frac{1}{2}(\dot{\mathbf{x}} - \dot{\mathbf{x}}_0) \cdot (\dot{\mathbf{x}} - \dot{\mathbf{x}}_0) \tag{1}$$

subject to the linear constraint:

$$\dot{\mathbf{h}}_0 = J\dot{\mathbf{x}} \tag{2}$$

That is, the problem is converted into a Lagrange equation:

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

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

$$JJ^T\lambda = \dot{\mathbf{h}}_0 - J\dot{\mathbf{x}}_0, \tag{3}$$

and this matrix equation is solved for the value of λ. The value of $\dot{\mathbf{x}}$ is obtained by

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

It is then used to update the virtual camera parameters $\mathbf{x}$. For example, using the Euler's method, $\mathbf{x}$ is updated as follows:

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

The Jacobian matrix J of the perspective transformation V plays an important role in the computation of the Lagrange equations; it is the most critical factor that determines the overall performance of the whole algorithm. The Jacobian matrix J should be re-evaluated each time the value of $\mathbf{x}$ is updated. When the Jacobian matrix J is very complex, it would disimprove the overall performance of the algorithm. In this paper, we present a simple $2m \times 7$ Jacobian matrix, which can be easily derived by a technique based on the quaternion calculus [7, 8, 9].

2.2 Rank Deficiency of the Jacobian Matrix

In computing the Jacobian matrix J for the perspective transformation V, the parameters (f, t_x, t_y, t_z) are free variables. This means that they can be differentiated without considering any constraints. However, the unit quaternions (q_w, q_x, q_y, q_z) have a constraint for the unit length (i.e., $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$) and it is somewhat difficult to differentiate the unit quaternions with the constraint. To eliminate this constraint, Gleicher and Witkin [5] derived the rotation matrix R by using $q/|q|$ instead of a unit quaternion, and then computed the differential of this rotation matrix. That is, for a unit quaternion $q = (q_w, q_x, q_y, q_z) \in S^3$, the rotation matrix is given by (see [11]):

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

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 [5] obtained a new rotation matrix:

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

This method defines a rotation matrix for any nonzero quaternion and eliminates the constraint for a unit quaternion. However, this rotation matrix has a problem: the Jacobian matrix is rank deficient.

For a given nonzero quaternion q $(\neq 0)$, the quaternions αq represent the same rotation matrix for $\alpha \in R$, i.e.,

$$R_{\alpha q} = R_q, \quad \text{for } \alpha \in R.$$

For given m points $p_1, \ldots, p_m \in R^3$ and a quaternion q, consider the transformation (for $i = 1, 2, \ldots, m$):

$$\begin{array}{rccc} T_i: & R^4 \setminus \{0\} & \longrightarrow & R^3. \\ & q & \longmapsto & R_q(p_i) \end{array}$$

The differential of T_i is a linear transformation: $d(T_i)_q : R^4 \longrightarrow R^3$, which can be represented by a 3×4 matrix [3]. Let $\hat{p}_i(t) \in R^3$ be the curve generated by the point p_i under the rotation of $q(t) \in R^4$, for $t \in R$, that is,

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

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

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

That is, the curve $\hat{p}_i(t)$ is a constant curve. The differential at $t = 1$ is a zero vector:

$$d(T_i)_q(q) = 0.$$

This means that the three rows (which are 4D vectors) of the Jacobian matrix of $d(T_i)_q$ are all orthogonal to the 4D vector q. Thus, $d(T_i)_q$ has rank 3, for $i = 1, \ldots, m$. Furthermore, for the transformation

$$\begin{array}{rccc} T: & R^4 & \longrightarrow & R^{3m} \\ & q & \longmapsto & (R_q(p_1), \ldots, R_q(p_m)), \end{array}$$

the differential dT_q is represented by a $3m \times 4$ Jacobian matrix and all the $3m$ rows of dT_q are orthogonal to the 4D vector q. Thus, dT_q has rank 3. In the formulation of the Jacobian matrix J of [5], the complexity and rank deficiency of the Jacobian matrix J essentially result from those of the Jacobian matrix dT_q for the rotational degrees of freedom.

3 A Simple Jacobian Matrix

3.1 Quaternion Calculus

To remedy all the complications of the Jacobian matrices dT_q and J, we take a different approach in representing the rotational degrees of freedom. Instead of starting from the quaternions, we start from the angular velocities and represent the quaternions by taking the integrals of the angular velocities. For a given unit quaternion curve $q(t) \in S^3$, $t \in R$, the differential $q'(t) \in T_{q(t)}(S^3) \subset R^4$ is given by:

$$q'(t) = \frac{1}{2}[0, \omega(t)] \cdot q(t), \tag{4}$$

for some $\omega(t) \in R^3$, where $\cdot$ is the quaternion multiplication, $T_{q(t)}(S^3)$ is the tangent space of S^3 at $q(t) \in S^3$, and $q'(t)$ is orthogonal to $q(t)$ as a 4D vector in R^4. (The details on the derivation of Equation (4) are described in [9]. Also see [7, 8, 11] for quaternions.) Given fixed 3D points $p_i \in R^3$ (for

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

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

(See [9] for more details on the derivation. When ω is interpreted as the angular velocity, this equation is exactly the same as the formula given in classical dynamics [14].) Consequently, for the transformation

$$\begin{array}{rccl} T_i: & S^3 & \longrightarrow & R^3 \\ & q & \longmapsto & R_q(p_i) = \hat{p}_i \end{array}$$

the differential $d(T_i)_q$ is given by

$$\begin{array}{rccl} d(T_i)_q: & T_q(S^3) & \longrightarrow & R^3 \\ & q' & \longmapsto & \omega \times \hat{p}_i \end{array}$$

Since the isomorphism

$$\begin{array}{rccl} F: & T_q(S^3) & \longrightarrow & R^3 \\ & q' = \frac{1}{2}\omega \cdot q & \longmapsto & \omega \end{array}$$

identifies the tangent space $T_q(S^3)$ with the 3D Euclidean space R^3, we may interpret the differential $d(T_i)_q$ as

$$\begin{array}{rccl} d(T_i)_q: & R^3 & \longrightarrow & R^3 \\ & \omega & \longmapsto & \omega \times \hat{p}_i \end{array}$$

The Jacobian matrix of $d(T_i)_q$ can be represented by a simple 3×3 square matrix:

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

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 the relation

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

where Δt is the time interval for the integration, the operation $\cdot$ is the quaternion multiplication, and the transformation exp is the exponential map. (See [2, 7, 8] for more details on the exponential map.)

3.2 The Jacobian Matrix for a Viewing Transformation

A virtual camera can be specified by a perspective viewing transformation which projects 3D points onto the 2D viewing plane. Let the position and orientation of the virtual camera at time t be given by $-t_{(x,y,z)}(t) = (-t_x(t), -t_y(t), -t_z(t)) \in R^3$ and $q^{-1}(t) = \overline{q}(t) = (q_w(t), -q_{(x,y,z)}(t)) = (q_w(t), -q_x(t), -q_y(t), -q_z(t)) \in S^3$, where $\overline{q}(t)$ is the quaternion conjugate of $q(t)$. For a given fixed 3D point $p = (x, y, z) \in R^3$ in the world coordinate system, the projected 2D image point $h(t) \in R^2$ can be represented by:

$$\begin{aligned} h(t) &= V_p(f(t), t_{(x,y,z)}(t), q(t)) \\ &= P_{f(t)} \circ Q_{q(t)} \circ T_{t_{(x,y,z)}(t)}(p) \end{aligned} \tag{5}$$

where $P_{f(t)}$ is the perspective projection with a focal length $f(t)$, $T_{t_{(x,y,z)}(t)}$ is the translation by $t_{(x,y,z)}(t) \in R^3$, $q(t) = (q_w(t), q_{(x,y,z)}(t)) \in S^3$, and $Q_{q(t)}$ is the rotation about the axis $q_{(x,y,z)}(t) \in R^3$ by an angle $2\theta(t)$, where $\cos\theta(t) = q_w(t)$.

The 3D rigid transformation: $Q \circ T(p) = \hat{p} = (\hat{x}, \hat{y}, \hat{z}) \in R^3$ is simply given by

$$\hat{p}(t) = Q_{q(t)} \circ T_{t_{(x,y,z)}(t)}(p)$$

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

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

To derive the Jacobian matrix J of the viewing transformation V_p, we differentiate Equation (5):

$$\frac{dh}{dt} = \frac{dV_p}{dt} = \frac{d}{dt} P_{f(t)} \circ Q_{q(t)} \circ T_{t_{(x,y,z)}(t)}(p)$$

By applying the chain rule to this equation, we obtain

$$\frac{dh}{dt} = J \begin{pmatrix} f' & t_x' & t_y' & t_z' & \omega_x & \omega_y & \omega_z \end{pmatrix}^T$$

with

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

where r_{ij} is the ij-th component of the 3×3 rotation matrix $R_{q(t)}$ of the unit quaternion $q(t)$. (See [9] for more details on the derivation.) This Jacobian matrix J is much simpler than the one given in Gleicher and Witkin [5].

4 Camera Control by Moving Image Control Points

4.1 Moving a Single Image Point

In through-the-lens camera control, the virtual camera placement is automatically determined by solving the following equation:

$$h_0 = V_p(f, t_x, t_y, t_z, q_w, q_x, q_y, q_z) \tag{6}$$

where $p \in R^3$ is the given 3D point, and h_0 is the 2D point onto which the point p is to be projected.

Gleicher and Witkin [5] approximated the solution by using a constrained nonlinear optimization technique and solving a series of Lagrange equations. For an underconstrained system with many possible solutions, the solution of Equation (3) provides an optimal solution with respect to the objective function of Equation (1). For a system with no solution (e.g., an overconstrained system), an approximate solution may be computed using the projection method to Equation (2) (see [13] and Section 5.1). When the vector $\dot{\mathbf{h}}_0$ is projected into the column space of J in Equation (1), there is now at least one solution of Equation (2). (Note that, when the projection method is applied to Equation (3), it is not clear what geometric meaning the approximate solution has.)

The projection process enforces us to give up some hard constraints. After then, we believe the optimization process does not make much sense. In this paper, we rather concentrate on how to control the projection in a user-controllable way (see Section 5.2). We approximate the solution of Equation (6) using the Newton method [1]. 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, $\Delta\mathbf{x} = (f', t'_{(x,y,z)}, \omega) \in R^4 \times T_q(S^3)$, where $\mathbf{x} = (f, t_{(x,y,z)}, q) \in R^4 \times S^3$, and $q \in S^3$. By integrating the velocities, we can approximate the solution of Equation (6). Let $F : R^4 \times S^3 \to R^2$, be defined by:

$$F(\mathbf{x}) = V_p(\mathbf{x}) - h_0. \tag{7}$$

Given

$$\mathbf{x}_k = (f_k, t_{(x,y,z),k}, q_k), \text{ and}$$
$$\Delta\mathbf{x}_k = (\Delta f_k, \Delta t_{(x,y,z),k}, \omega_k),$$

with $q_k \in S^3$ and $\omega_k \in T_{q_k}(S^3)$, let

$$\mathbf{x}_{k+1} = (f_k + \Delta f_k, t_{(x,y,z),k} + \Delta t_{(x,y,z),k}, \exp(\frac{\omega_k}{2}) \cdot q_k).$$

The Taylor series expansion of F at $\mathbf{x}_{k+1}$ gives:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k) + dF_{\mathbf{x}_k}(\Delta\mathbf{x}_k) + o(|\Delta\mathbf{x}_k|^2),$$

where $dF_{\mathbf{x}_k} : R^4 \times T_q(S^3) \to R^2$ is the differential of F at $\mathbf{x}_k$ [3]. We approximate $\mathbf{x}_{k+1}$ so that

$$F(\mathbf{x}_{k+1}) = 0.$$

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

$$0 = F(\mathbf{x}_k) + dF_{\mathbf{x}_k}(\Delta\mathbf{x}_k).$$

Thus we solve for $\Delta\mathbf{x}_k \in R^4 \times T_{q_k}S^3$ in the following linear system:

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

where the Jacobian matrix $J(\mathbf{x}_k)$ is the matrix representation of the differential $dF_{\mathbf{x}_k}$. (Note that this matrix equation is the same as Equation (2).) Here, $J(\mathbf{x}_k)$ is not a square matrix; thus it is not invertible. Weighted least squares method will be used in Section 5 to approximate the solution.

4.2 Moving Multiple Image Points

The linear system for a single control point has been derived as a 2×7 matrix in Section 4.1. For multiple 2D image control points, Equation (7) can be generalized as follows:

$$F(f, t_x, t_y, t_z, q_w, q_x, q_y, q_z) = \begin{pmatrix} \frac{f(\hat{p}_1)_x}{(\hat{p}_1)_z} - (h_1)_x \\ \frac{f(\hat{p}_1)_y}{(\hat{p}_1)_z} - (h_1)_y \\ \vdots \\ \frac{f(\hat{p}_m)_x}{(\hat{p}_m)_z} - (h_m)_x \\ \frac{f(\hat{p}_m)_y}{(\hat{p}_m)_z} - (h_m)_y \end{pmatrix}$$

As the function F is extended to $2m$-dimension, the Jacobian matrix $J(\mathbf{x}_k)$ now becomes a $2m \times 7$ matrix. The linear equation $J(\mathbf{x}_k)\Delta\mathbf{x}_k = -F(\mathbf{x}_k)$ may have many solutions or no solution at all depending on the rank of $J(\mathbf{x}_k)$ and the value of $F(\mathbf{x}_k)$.

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 take this fact into account in controlling the virtual camera parameters. For the moving 3D points $p_i(t)$'s, the derivative for the 3D rotation and translation of the virtual camera can be computed as follows (see [9]):

$$\begin{aligned} & \frac{d}{dt} Q_{q(t)} \circ T_{t_{(x,y,z)}(t)}(p(t)) \\ = \;& \omega(t) \times \hat{p}(t) + Q_{q(t)}(p'(t) + t'_{(x,y,z)}(t)). \end{aligned}$$

Thus, for the perspective viewing transformation

$$h(t) = V_{p(t)}(f(t), t_{(x,y,z)}(t), q(t)),$$

we have

$$\frac{dh}{dt} = J(\mathbf{x}_k) \begin{pmatrix} f' \\ t'_x + p'_x \\ t'_y + p'_y \\ t'_z + p'_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}.$$

The linear system to be solved is:

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

5 Solving Linear System

5.1 Computing Pseudo Inverse

When there are m control points in the image space, the system to be solved is a $2m \times 7$ linear system:

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

where $\Delta\mathbf{x} = (f', t'_x, t'_y, t'_z, \omega_x, \omega_y, \omega_z) \in R^4 \times T_q(S^3)$. Since the non-square matrix $J(\mathbf{x})$ is not invertible, a solution should be chosen from the possible solutions of $\Delta\mathbf{x}$'s so that it is optimal to some given criteria. We suggest the following least squares method for the selection of a solution:

1. For the case of $2m < 7$, $\Delta\mathbf{x}$ with the minimal norm is selected from the solutions for $J(\mathbf{x})\Delta\mathbf{x} = -F(\mathbf{x})$.
2. For the case of $2m > 7$, $\Delta\mathbf{x}$ is chosen so that it minimizes $|J(\mathbf{x})\Delta\mathbf{x} + F(\mathbf{x})|$.

To compute the least squares solution in a numerically stable way, we use the Singular Value Decomposition (SVD) and the pseudo inverse of the Jacobian matrix $J(\mathbf{x})$ [6, 10, 13]. There are various efficient and stable methods [6, 10] to decompose a matrix A into the form UWV^T, where W is a diagonal matrix. After decomposing the Jacobian matrix J into UWV^T, its pseudoinverse J^+ is computed as:

$$J^+ = VW^{-1}U^T$$

where

$$(W^{-1})_{ij} = \begin{cases} 0 & \text{if } i \neq j \text{ or } (W)_{ij} = 0 \\ 1/(W)_{ij} & \text{if } i = j \text{ and } (W)_{ij} \neq 0 \end{cases}$$

The numerical stability of the SVD method is unsurpassed by any other methods, especially when the matrix J is almost singular. The above pseudoinverse matrix always produces the solution with the minimal norm while satisfying the condition.

For the case of $2m > 7$, the pseudoinverse would require the SVD of a large $2m \times 7$ matrix as m increases. In this case, it is more efficient to use the projection method [13] which produces the solution that minimizes the quantity $|J\Delta\mathbf{x} + F|$, that is,

$$\Delta\mathbf{x} = -J^+F$$

where

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

But, the pseudoinverse J^+ is not defined when the square matrix J^TJ is singular, i.e., when the column vectors of J are not linearly independent. Since the SVD method can detect the singularity of a matrix to be decomposed, we compute $(J^TJ)^{-1}$ by using the SVD method. When the square matrix J^TJ turns out to be singular, we go back to the previous method of computing the pseudo inverse: $J^+ = VW^{-1}U^T$ based on the SVD decomposition of $J = UWV^T$. The construction of the 7×7 square matrix J^TJ takes $O(m)$ time and the inverse operation for J^TJ takes constant time.

5.2 Weighted Least Squares

For an underconstrained system, the least squares method gives the solution which minimizes the magnitude $|\Delta\mathbf{x}|$. However, sometimes other solutions may be required. For example, when the user wants to move the camera with little change of focus and/or camera rotation, the solution should be skewed from the least squares solution. That is, higher weights should be given to the parameters for less 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 condition is to scale the camera parameter space in different ratios. This can be done easily by scaling each column of the matrix; that is, by solving $AWx = b$ instead of $Ax = b$, where W is a diagonal matrix.

When the linear system is overconstrained, that is, the number of control points is more than 3, a solution dominated by certain 2 or 3 control points may be desired. To control the contribution of each control point, we suggest the row-weighting method. A given linear system $Ax = b$ is changed into a new form $WAx = Wb$, where W is a diagonal matrix. The row and column weightings may be combined together, reducing the linear system $Ax = b$ into a general form of $W_1 A W_2 x = W_1 b$, where W_1 and W_2 are diagonal matrices.

The row-weighting method may have useful applications in computer animation. For an animation movie with dramatic scene changes, it is not enough to have only a few control points. The control points appropriate 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 as

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

where p_i is the i-th 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 $t = e_i$, the row-weighting function $w_i(t)$ for $p_i \in A(t)$ is defined as a non-negative smooth function with $w_i(t) = 0$ for $t \le s_i$ or $t \ge e_i$.

6 Implementation and Results

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

Camera-Control($f_x, f_e, \mathbf{x}_{f_s}, H_{f_e}$)
Input:
  f_s, f_e : the start and end frames;
  $\mathbf{x}_{f_s}$: the start camera parameters;
  H_{f_s}, H_{f_e}: the start and end positions;
Output:
  $\mathbf{x}_{f_s}, \ldots, \mathbf{x}_{f_e}$: a sequence of camera parameters;
begin
  $\Delta H := \frac{1}{f_e - f_s}(H_{f_e} - H_{f_s})$;
  for $j := f_s + 1$ **to** f_e **do begin**
    $H_j := H_{j-1} + \Delta H$;
    $\mathbf{x}_j :=$ Newton$(\mathbf{x}_{j-1}, H_{j-1}, H_j)$;
    end
end

Newton($\mathbf{x}^{(0)}, H^{(0)}, H_0$)
Input:
  $\mathbf{x}^{(0)}$: the initial camera parameters;
  $H^{(0)} = (h_1^{(0)}, ..., h_m^{(0)})$: the start positions;
  H_0: the destination positions;
Output:
  $\mathbf{x}^{(i+1)}$: the approximate solution of $V_P(\mathbf{x}) = H_0$;
begin
  /* $H^{(i)} = (h_1^{(i)}, ..., h_m^{(i)})$: positions at i-th step */
  for $i = 0$ **to** *MAX-ITERATION* **do begin**
(1) $F(\mathbf{x}^{(i)}) := H^{(i)} - H_0$;
(2)   Construct the Jacobian matrix: $J(\mathbf{x}^{(i)})$;
(3)   Solve for $\Delta\mathbf{x}^{(i)}$ in the matrix equation:
      $J(\mathbf{x}^{(i)})\Delta\mathbf{x}^{(i)} = -F(\mathbf{x}^{(i)})$;
    /* $\mathbf{x}^{(i)} = (f^{(i)}, t^{(i)}_{(x,y,z)}, q^{(i)})$,
      $\Delta\mathbf{x}^{(i)} = (\Delta f^{(i)}, \Delta t^{(i)}_{(x,y,z)}, \omega^{(i)})$, and
      Δt is the time step */
(4) $\mathbf{x}^{(i+1)} = (f^{(i)} + \Delta t \ \Delta f^{(i)}$,
      $t^{(i)}_{(x,y,z)} + \Delta t \ \Delta t^{(i)}_{(x,y,z)}, \exp(\frac{\Delta t \ \omega^{(i)}}{2}) \cdot q^{(i)})$;
(5) $H^{(i+1)} = V_P(\mathbf{x}^{(i+1)})$;
(6)   **if** $\left| \|H^{(i+1)} - H_0\| - \|H^{(i)} - H_0\| \right| < \epsilon$ **then**
      return $(\mathbf{x}^{(i+1)})$;
  end
end

The most time-consuming is Step (3), which solves for $\Delta\mathbf{x}$ in the linear system $J(\mathbf{x})\Delta\mathbf{x} = -F(\mathbf{x})$ with the least squares method. The main subroutine for this step is the SVD, which takes $O(2rc^2 + 4c^3)$ time, for an $r \times c$ matrix with $c \le r$. When the linear system is underconstrained, i.e., $m \le 3$, the whole computation takes constant time. For an overconstrained system with $m > 3$, the SVD of $J(\mathbf{x})$ takes $O(2 \cdot (2m) \cdot 7^2 + 4 \cdot 7^3) = O(m)$ time. This is a promising result since the time complexity grows only linearly as the number of control points increases.

Three experimental results are demonstrated in Figure 2. Examples of controlling three and four image points are shown in Figure 2(a) and (b), respectively. In these two cases, the 3D points are stationary points. In Figure 2(c), a more general case is shown for three moving 3D points. The numerical approximations up to three control points are

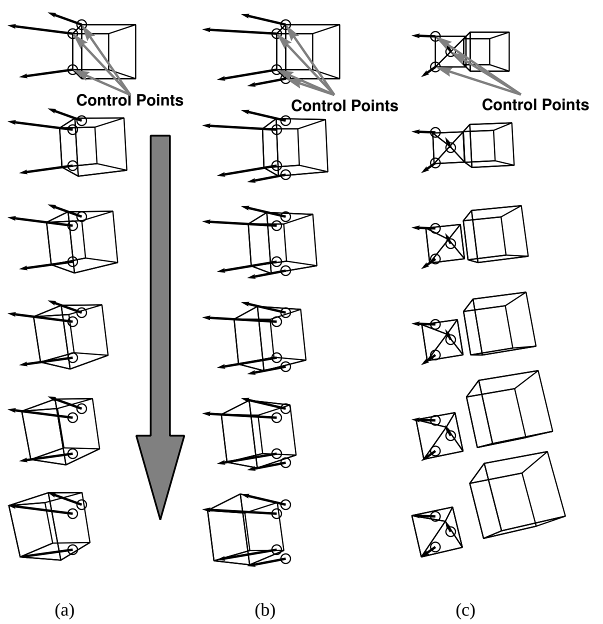

Figure 2: Experimental Results

accurate as shown in Figure 2(a) and (c). However, in the overconstrained case of controlling more than three points, we have experienced large approximation errors as shown in Figure 2(b).

7 Conclusion

The Jacobian matrix plays an important role in the through-the-lens camera control. The computational efficiency and numerical stability of the overall algorithm mainly depend on the simplicity of the Jacobian matrix. The SVD method makes the Newton approximation numerically stable. The weight least squares method provides a convenient way of controlling optimization criteria.

The simplicity of the Jacobian matrix J is due to the fact that the unit quaternion space S^3 is a Lie group [2]. This implies that $q'(t) = \frac{1}{2}[0, w(t)] \cdot q(t) \in T_{q(t)}(S^3)$, for some tangent vector $w(t) \in T_1(S^3) \equiv R^3$ at the identity element 1 of S^3. The differential calculus becomes much more complex on other virtual camera model spaces when they are not Lie groups. Nevertheless, the tangent space and the exponential map are also defined in any differentiable manifold [3], and our technique is extendible to the general case as long as the exponential map has an explicit formula. For example, consider the fibre bundle structure $S^2 \times S^1$ of Shoemake [12] for the control of camera rotations. Applying the chain rule to $T_i \circ F : S^2 \times S^1 \to R^3$, where $F : S^2 \times S^1 \to S^3$ is a diffeomorphism, we can compute the Jacobian matrix for the camera model. The exponential map also has a relatively simple explicit formula in this case.

References

[1] Conte, S., and de Boor, C., *Elementary Numerical Analysis: An Algorithmic Approach*, 3rd Ed., McGraw-Hill, Singapore, 1981.

[2] Curtis, M., *Matrix Groups*, Springer-Verlag, New York, 1979.

[3] do Carmo, M., *Differential Geometry of Curves and Surfaces*, Prentice Hall, Englewood Cliffs, New Jersey, 1976.

[4] Foley, J., van Dam, A., Feiner, S., and Hughes, J., *Computer Graphics, Principles and Practice*, 2nd Ed., Addison-Wesley, Reading, Mass., 1990.

[5] Gleicher, M., and Witkin, A., "Through-the-Lens Camera Control," *Computer Graphics*, Vol. 26, No. 2, 1992, pp. 331–340,

[6] Golub, G., and Van Loan, C., *Matrix Computations*, Johns Hopkins University Press, 1983.

[7] Kim, M.-J., Kim, M.-S., and Shin, S., "A Compact Differential Formula for the First Derivative of a Unit Quaternion Curve," Technical Report CS-CG-94-005, Dept. of Computer Science, POSTECH, 1994.

[8] Kim, M.-S., and Nam, K.-W., "Interpolating Solid Orientations with Circular Blending Quaternion Curves," to appear in *Computer-Aided Design*.

[9] Kyung, M.-H., Kim, M.-S., and Hong, S.J., "Through-the-Lens Camera Control with a Simple Jacobian Matrix," Technical Report CS-CG-94-006, Dept. of Computer Science, POSTECH, 1994.

[10] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W., *Numerical Recipes*, Cambridge University Press, 1986.

[11] Shoemake, K., "Animating Rotation with Quaternion Curves," *Computer Graphics (Proc. of SIGGRAPH '85)*, Vol. 19, No. 3, 1985, pp. 245–254.

[12] Shoemake, K., "Fibre Bundle Twist Reduction," *Graphics Gems IV*, Heckbert, P., (Ed.), Academic Press, Boston, 1994, pp. 230–236.

[13] Strang, G., *Linear Algebra and its Applications*, 3rd Ed., Harcourt Brace Jovanovich, Pub., Orlando, Florida, 1988.

[14] Wittenburg, J., *Dynamics of Systems of Rigid Bodies*, B.G. Teubner, Stuttgart, 1977.