# Through-the-Lens Camera Control

Michael Gleicher and Andrew Witkin
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
{gleicher|witkin}@cs.cmu.edu

## Abstract

In this paper we introduce *through-the-lens camera control,* a body of techniques that permit a user to manipulate a virtual camera by controlling and constraining features in the image seen through its lens. Rather than solving for camera parameters directly, constrained optimization is used to compute their time derivatives based on desired changes in user-defined controls. This effectively permits new controls to be defined independent of the underlying parameterization. The controls can also serve as constraints, maintaining their values as others are changed. We describe the techniques in general and work through a detailed example of a specific camera model. Our implementation demonstrates a gallery of useful controls and constraints and provides some examples of how these may be used in composing images and animations.

**Keywords:** camera control, constrained optimization, interaction techniques

## 1 Introduction

Camera placement and control play an important role in image composition and computer animation. Consequently, considerable effort has been devoted to the development of computer graphics camera models. Most camera formulations are built on a common underlying model for perspective projection under which any 3-D view is fully specified by giving the center of projection, the view plane, and the clipping volume. Within this framework, camera models differ in the way the view specification is parameterized. Not all formulations are equivalent—some allow arbitrary viewing geometries, while others impose restrictions. Even so, alternative models can be viewed to a great degree as alternative slicings of the same projective pie.

How important is the choice of the camera model's parameterization? Very important, if the parameters are to serve directly as the controls for interaction and keyframe interpolation. For example, the popular LOOKAT-/LOOKFROM/VUP parameterization makes it easy to hold a world-space point centered in the image as the camera moves without tilting. To do the same by manually controlling generic translation/rotation parameters would be all but hopeless in practice, although possible in principle.

The difficulty with using camera parameters directly as controls is that no single parameterization can be expected to serve all needs. For example, sometimes it is more convenient to express camera orientation in terms of azimuth, elevation and tilt, or in terms of a direction vector. These particular alternatives are common enough to be standardly available, but others are not. A good example involves the problem, addressed by Jim Blinn[3] of portraying a spacecraft flying by a planet. Blinn derives several special-purpose transformations that allow the image-space positions of the spacecraft and planet to be specified and solves for the camera position. The need for this kind of specialized control arises frequently, but we would rather not face the prospect of deriving and coding specialized transformations each time they do.

In short, camera models are inflexible. To change the controls, one must either select a different pre-existing model or derive and implement a new one. If this inflexibility could be removed, the effort devoted to camera control could be reduced and the quality of the result enhanced.

In this paper, we present a body of techniques, which we call *through-the-lens camera control,* that offer a general solution to this problem. Instead of a fixed set, the user is given a palette of interactive image-space and world-space controls that can be applied "on the fly," in any combination. For example, the image-space position of an arbitrary world-space point can be controlled by interactive dragging, or pinned while other points are moved. Image-space distances, sizes, and directions can also be

Computer Graphics 26(2), July, 1992.
pages 331–340.
Proceedings SIGGRAPH '92.

controlled. Points can be constrained to remain within the image or within a specified sub-region. These and other image-space controls can be freely combined with direct world-space controls on camera position and orientation. The set of controls is extensible, with a general procedure for adding new ones.

Using through-the-lens control, the spacecraft/planet problem, and others of its kind, could be solved immediately and interactively: a point on the planet would be grabbed, dragged to its target location, then left pinned at that image point. The spacecraft would be similarly positioned and pinned. Residual degrees of freedom could be fixed by dragging additional points, or manipulating the camera in world space, while both image points remained nailed. All the while, the required camera motions would be computed automatically at interactive speed.

The principal technical obstacle to achieving this kind of control lies in the nonlinearity of the relationship between the desired controls and the underlying view specification. No general guaranteed procedure exists for solving nonlinear algebraic systems; in fact there may often exist no solution, or many. The direct approach—solving numerically for the camera parameters given the controls—is therefore unlikely to succeed.

The key to our approach is that we instead formulate the problem differentially—solving for the *time derivatives* of the camera parameters, given the *time derivatives* of the controls. For example, a point is dragged by specifying its velocity from moment to moment, rather than giving a final target position. When the camera is under interactive control, it falls to the user interface to convert user actions, such as pointer motions, into suitable velocity signals. In keyframe control, the velocity is calculated by taking the time derivative of the interpolating function. The use of differential control does not allow us to directly position the camera in global leaps, but instead provides a robust and accurate means of translating continuous adjustments of the controls into continuous motions of the camera. We are primarily interested in interactive control by grabbing and dragging, for which continuous motion is desirable, and with low-level camera control for animation, for which continuous motion is sufficient.

We formulate the problem of computing time derivatives of camera parameters as a simple constrained optimization. Once the derivatives have been computed, using them to update the camera's state over time reduces to the standard problem of solving a first-order ordinary differential equation from an initial value, for which good numerical methods abound (see Press *et. al.* [20] for a good practical introduction.)

An interesting feature of through-the-lens control is the new role in which it places the camera parameterization. Although we retain a fixed underlying camera model, the model's parameters no longer bear the burden of serving as user-level controls. In fact, the parameters may be completely hidden from the user. This leaves us free to choose a camera model on the basis of numerical well-behavedness and implementation convenience. Our preferred formulation, based on quaternion rotations, is a case in point: it is a very poor model by conventional criteria, since the four components of the quaternion would be exceedingly difficult to control directly. Yet it provides an ideal substrate for through-the-lens control because it allows free camera rotations without singularities or other artifacts. We avoid the well-known difficulties involved with interpolating quaternions for animation[22] by interpolating the controls instead.

The remainder of the paper is organized as follows: Following a discussion of related work, we develop the machinery of through-the-lens control in terms of a generic camera model, starting with the problem of controlling the image-space velocity of a single point, then generalizing to the full solution. We then present complete through-the-lens equations for our simple quaternion-based camera model. We describe our implementation and present examples, then conclude with a discussion of future work.

## 2 Related Work

As we noted in Section 1, standard computer graphics camera models are based on specialized transformations that specify the view as a function of parameters that are useful for interactive, procedural, or keyframed control. Earlier we discussed the standard LOOKAT/LOOKFROM model. An example of a more general viewing model currently in wide use is the PHIGS+ model[6]. In addition, a variety of special-purpose models such as Blinn's spacecraft flyby transformations [3] have been developed. Issues involved in using the LOOKAT/LOOKFROM model to navigate virtual spaces are considered by [16]. In [7], the LOOKAT/LOOKFROM model is embedded in a procedural language for specifying camera motions.

Much of the work on interactive camera placement in computer graphics has been concerned with direct control of the camera's position and orientation. The problem of developing intuitive controls for 3-D rotations is a difficult one[5], particularly when the input device is two-dimensional. Several researchers have addressed the problem through the use of use of 3-D interfaces, including six degree-of-freedom pointing devices[26, 25, 1] and more specialized devices such as steerable treadmills[4].

Problems involving the recovery of camera parameters from image measurements have been addressed in photogrammetry[1], computer vision, and robotics. All of these are concerned with the recovery of parameter values, rather than time derivatives. Algebraic solutions to specific problems of this kind are given in [18] and [9], while numerical solutions are discussed in [15, 10, 17]. In [24], constrained optimization is employed to position a

[1] Also see chapter 6 of [21] for amazing mechanical solutions to photogrammetry problems.

real camera, mounted on a robot arm, for the purpose of object recognition. Factors considered in the optimization include depth of field, occlusion, and image resolution.

Optimization techniques have been applied to the related problem of object placement in computer graphics. In [1] articulated figures are posed using penalty methods to meet positional goals. In [28], similar methods are employed for general object placement and control. The use of these methods for camera placement in animation is described in [30].

The differential control methods employed in this paper are formally more closely allied to the methods of constrained dynamic simulation described in [29, 2, 19, 31, 23] than to the positional optimization methods cited above. Some of the issues involved in adapting these methods to differential kinematic control are addressed in [13], while [27] considers their application to the design of free form surfaces, and [12] illustrates their use in a constraint-based drawing program.

# 3 The Machinery

In this section we introduce the basic mechanisms that support through-the-lens control, employing a simple constrained optimization formulation. Assuming that we have chosen a camera model to provide the fixed, underlying parameterization, we solve for the time derivatives of the parameters such that their mean squared deviation from a desired value is minimized, subject to the constraints imposed by the image-space controls. Setting the default values to zero yields a solution that minimizes the mean squared rate of change of camera parameters. Non-zero values can be used to support interactive dragging subject to the constraints imposed by other controls.

We begin by giving the relationship between a world-space point and its image-space counterpart, which we express in terms of a generic camera model. A specific quaternion-based model will be fully described in section 4. We give the coordinates of an image point $\mathbf{p}$ as

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

where $\mathbf{x}$ is the world-space point that projects to $\mathbf{p}$, $\mathbf{V}$ is a homogeneous matrix representing the combined projection and viewing transformations, and $\mathbf{h}$ is a function that converts homogeneous coordinates into 2-D image coordinates, defined by

$$\mathbf{h}(\mathbf{x}) = \left[\frac{x_1}{x_4}, \frac{x_2}{x_4}\right],$$

where the $x_i$'s are components of homogeneous point $\mathbf{x}$. The matrix $\mathbf{V}$ is some (for now unspecified) function of the camera model parameters, which we denote by a length-$n$ vector $\mathbf{q}$. In practice, $\mathbf{V}$ would usually be computed as the product of several matrices, each a function of one or more of the parameters.

## 3.1 Camera motions and image point velocities.

Assuming for now that the world space point $\mathbf{x}$ is fixed, the image point $\mathbf{p}$ is entirely a function of the camera parameters $\mathbf{q}$. This is a nonlinear relationship because $\mathbf{h}$ is nonlinear, as in general is $\mathbf{V}(\mathbf{q})$. We obtain the expression for the image velocity $\dot{\mathbf{p}}$ by applying the chain rule:

$$\dot{\mathbf{p}} = \mathbf{h}'(\mathbf{V}\mathbf{x})\left(\frac{\partial(\mathbf{V}\mathbf{x})}{\partial\mathbf{q}}\right)\dot{\mathbf{q}}, \tag{2}$$

where $\mathbf{h}'(\mathbf{x})$ is the matrix representing the derivative of $\mathbf{h}(\mathbf{x})$, given by

$$\mathbf{h}'(\mathbf{x}) = \begin{bmatrix} \frac{1}{x_4} & 0 \\ 0 & \frac{1}{x_4} \\ 0 & 0 \\ -\frac{x_1}{x_4{}^2} & -\frac{x_2}{x_4{}^2} \end{bmatrix}, \tag{3}$$

$\dot{\mathbf{q}}$ is the time derivative of $\mathbf{q}$, and $\partial(\mathbf{V}\mathbf{x})/\partial\mathbf{q}$ is the $4 \times n$ matrix representing the derivative of the transformed point $\mathbf{V}\mathbf{x}$ with respect to $\mathbf{q}$. We differentiate the point $\mathbf{V}\mathbf{x}$ rather than the matrix $\mathbf{V}$ to avoid differentiating a matrix with respect to a vector, which would give rise to a rank-3 tensor. In section 4, we give an example of how this derivative matrix can be computed.

For notational compactness we will define the $2 \times n$ matrix

$$\mathbf{J} = \mathbf{h}'(\mathbf{V}\mathbf{x})\frac{\partial(\mathbf{V}\mathbf{x})}{\partial\mathbf{q}}, \tag{4}$$

so that

$$\dot{\mathbf{p}} = \mathbf{J}\dot{\mathbf{q}}. \tag{5}$$

Notice that equation 5 gives $\dot{\mathbf{p}}$ as a *linear* function of $\dot{\mathbf{q}}$, even though $\mathbf{p}$ is a nonlinear function of $\mathbf{q}$.

## 3.2 Controlling a single point

Having obtained $\dot{\mathbf{p}}$ as a function of $\dot{\mathbf{q}}$, we next consider the problem of controlling a single image point, i.e. solving for a value of $\dot{\mathbf{q}}$ that makes the image point assume a given velocity $\dot{\mathbf{p}} = \dot{\mathbf{p}}_0$. In practice the value for $\dot{\mathbf{p}}_0$ might be supplied by the user interface or might indicate the velocity on a keyframed motion path. Although the relation between $\dot{\mathbf{p}}$ and $\dot{\mathbf{q}}$ is linear, we cannot simply solve $\dot{\mathbf{p}}_0 = \mathbf{J}\dot{\mathbf{q}}$ for $\dot{\mathbf{q}}$ unless matrix $\mathbf{J}$ is square and of full rank, which in general it will not be.

The singularity of the matrix $\mathbf{J}$ reflects the fact that many distinct camera motions can cause a single point to move in the same way. One way to solve the problem might be to require the user to control enough points or other features to yield a square matrix. We choose a different option that offers far more flexibility: subject to the constraint that $\dot{\mathbf{p}} = \dot{\mathbf{p}}_0$, we minimize the magnitude of $\dot{\mathbf{q}}$'s deviation from a specified value $\dot{\mathbf{q}}_0$. Letting $\dot{\mathbf{q}}_0 = 0$ imposes a criterion of minimal change in the camera parameters. As we shall see later, the ability to choose other values makes it possible, for example, to drag image points and other features *subject to* the hard constraints.

The problem we now wish to solve is:

$$\text{minimize } E = \frac{(\dot{\mathbf{q}} - \dot{\mathbf{q}}_0) \cdot (\dot{\mathbf{q}} - \dot{\mathbf{q}}_0)}{2} \text{ subject to } \dot{\mathbf{p}} - \dot{\mathbf{p}}_0 = 0. \tag{6}$$

To qualify as a constrained minimum, $\dot{\mathbf{q}}$ must satisfy several conditions. First, of course, the constraint must be met, i.e.

$$\dot{\mathbf{p}}_0 = \mathbf{J}\dot{\mathbf{q}}.$$

At an unconstrained minimum, we would require that the gradient $dE/d\dot{\mathbf{q}}$ vanish. Instead, we require that it point in a direction in which displacements are prohibited by the constraints. This condition is expressed by requiring that

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

for some value of the 2-vector $\lambda$ of *Lagrange multipliers.* This equation simply states that the gradient of $E$ must be a linear combination of the gradients of the constraints. Combining the two conditions gives

$$\mathbf{J}\mathbf{J}^T\lambda = \dot{\mathbf{p}}_0 - \mathbf{J}\dot{\mathbf{q}}_0, \tag{7}$$

which is a matrix equation to be solved for $\lambda$. Then the camera parameter derivatives are given by

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}_0 + \mathbf{J}^T\lambda. \tag{8}$$

Finally, we must use the computed value of $\dot{\mathbf{q}}$ to update the camera state $\mathbf{q}$, a standard initial value problem. See Press *et. al.* [20] for a discussion of the issues and a good assortment of numerical methods for ordinary differential equations. The very simplest method, *Euler's method,* employs the update formula

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

Although easy to implement, Euler's method is notoriously unstable and inaccurate. Use it at your own risk! In the interactive loop of through-the-lens control, drawing and input are interleaved with solver steps.

### 3.3 General quadratic objective functions

The restricted form of the objective function given in equation 6 is often adequate, but can cause problems: when the controls do not fully determine the camera's state, the task of accounting for the remaining degrees of freedom falls to the objective function. For example, if the camera is able to respond to the motion of a controlled point by a combination of tracking and panning, the objective function determines how much of each will take place. Because the error norm of equation 6 is the Euclidean distance in the camera's parameter space, rather than being intrinsic to the world-space camera motion, the behavior depends in a somewhat haphazard way on the choice of camera parameterization, and could even depend, for example, on the choice of linear and angular units of measure!

To allow such behavior to be controlled in a more rational way we make a reasonably straightforward generalization, allowing $E$ to be any quadratic function of $\dot{\mathbf{q}}$, having the form

$$E = \frac{1}{2}\dot{\mathbf{q}}\mathbf{M}\dot{\mathbf{q}} + \mathbf{b} \cdot \dot{\mathbf{q}} + c,$$

where $\mathbf{M}$ is a matrix, typically symmetric and positive-definite, $\mathbf{b}$ is a vector, and $c$ is a scalar, none of them depending on $\dot{\mathbf{q}}$. Since $E$ is quadratic, the problem remains linear, although the matrix equation to be solved becomes a bit more complex. The gradient of $\mathbf{E}$ becomes

$$\frac{\partial E}{\partial \dot{\mathbf{q}}} = \mathbf{M}\dot{\mathbf{q}} + \mathbf{b}.$$

Denoting the inverse of $\mathbf{M}$ by $\mathbf{W} = \mathbf{M}^{-1}$, equation 7 assumes the form

$$\dot{\mathbf{p}}_0 = \mathbf{J}\mathbf{W}\mathbf{J}^T\lambda - \mathbf{J}\mathbf{W}\mathbf{b}. \tag{9}$$

It is also possible to solve for $\lambda$ without obtaining the explicit inverse for $\mathbf{M}$ by forming a larger linear system (see [8].)

Under this general linear/quadratic formulation, the camera's response to controls can be decoupled from the parameterization, for instance by letting $\mathbf{M}$ be a *mass matrix* for the camera[13, 29, 31].

### 3.4 Multiple Points and Other Functions

Controlling more than one point involves a simple extension to the foregoing derivation. The matrix $\mathbf{J}$ depends on $\mathbf{x}$, so each point being controlled yields a distinct version of equation 5. We combine the $m$ equations into a single one by concatenating the derivative matrices to form a $2m \times n$ matrix, and concatenating the image velocities to form a $2m$-long vector. From that point on, the derivation proceeds as above, to the solution of equation 7 for $\lambda$, which is now also a vector of length $2m$.

In addition to controlling image points directly, we would like to control functions of one or more points, such as image distance or orientation. In fact, to mix image-space and world-space controls we may want to control other functions of $\mathbf{q}$ that do not involve the image at all, such as object-to-camera distance. Conceptually, this is not a difficult generalization to make: in equation 5, we simply interpret $\mathbf{p}$ not as a literal point, but as the vector of quantities we wish to control. Matrix $\mathbf{J}$ must then give the derivative of each controlled quantity with respect to each camera parameter. In practice, performing the derivative evaluations, indexing and other bookkeeping, etc., can become quite complex. See [14, 29] for general-purpose schemes that facilitate the handling of this kind of matrix-assembly problem. Although our own implementations are based on such a scheme, the camera control problem is sufficiently restricted in scope that this certainly is not necessary.

Many through-the-lens controls, such as point-to-point distance, can be expressed as functions of several image points' positions. The labor involved in implementing such controls can be greatly reduced through through the use of the chain rule. For instance, consider a scalar function of two image points $f(\mathbf{p}_1, \mathbf{p}_2)$. The derivative of $f$ with respect to $\dot{\mathbf{q}}$ is

$$\frac{df}{d\dot{\mathbf{q}}} = \frac{\partial f}{\partial \mathbf{p}_1}\mathbf{J}_1 + \frac{\partial f}{\partial \mathbf{p}_2}\mathbf{J}_2,$$

where $\mathbf{J}_1$ and $\mathbf{J}_2$ are the derivative matrices for $\mathbf{p}_1$ and $\mathbf{p}_2$, computed according to equation 4. The code that evaluates $\mathbf{J}$ for image points need only be implemented once. Thereafter, just derivatives with respect to image points need be treated anew for each control. These tend to be simple, and as an added advantage, they are independent of the choice of the underlying camera parameterization.

### 3.5 Constrained Dragging and Soft Controls

When controls are added dynamically by the user, it is entirely possible for inconsistencies to arise, either because the degrees of control exceed the camera's degrees of freedom, or because some controls are in conflict, e.g. trying to move one point in two directions. These problems can be handled gracefully by employing a least-squares method to solve the matrix equation—see for example the conjugate gradient solver described in [20]—so that the error due to the inconsistency is distributed uniformly over the controls, in a least-squares sense.

Although the least-squares solution avoids disaster when conflicts arise, we have found that it is very helpful to permit the user to drag points and other features *subject to* the constraints imposed by existing controls, so that conflicts can never arise. We achieve this behavior by incorporating the dragged point's desired behavior into the objective function, rather than using a "hard" constraint to control it. The constrained optimization solution then resolves any conflicts strictly in favor of the hard constraints. Thus, for example, a point whose range of motion is restricted by the controls will move freely up to the limit of its travel, but no further. A simple way to implement such "soft" controls is to specify the desired camera motion $\dot{\mathbf{q}}_0$ according to the formula

$$\dot{\mathbf{q}}_0 = k_c \mathbf{J}^T(\mathbf{p}_c - \mathbf{p}), \qquad (10)$$

where $\mathbf{k}_c$ is a constant and $\mathbf{p}_c$ is the position of the cursor in image coordinates. Using this value to drive the system is similar to attaching a rubber band between $\mathbf{p}_c$ and $\mathbf{p}$, inducing camera motion that causes $\mathbf{p}$ to "chase" $\mathbf{p}_c$. Inserting this value of $\mathbf{q}_0$ into equation 7 minimizes the mean squared difference between $\dot{\mathbf{q}}$ and $\dot{\mathbf{q}}_0$, subject to the constraints. Soft controls can be implemented more accurately, at the expense of greater complexity, by minimizing the squared difference between $\dot{\mathbf{p}}$ and a desired value $\dot{\mathbf{p}}_0$, subject to the constraints. To express this objective function, the general form given in equation 9 must be used.

A greatly simplified though much less powerful version of through-the-lens control is obtained by using soft controls only. Then, the constrained optimization of equation 6 collapses into an unconstrained optimization. For example, an image point could be dragged by using equation 10 directly to determine $\dot{\mathbf{q}}$.

### 3.6 Position Feedback

So far, we have cast the problem in terms of velocity control. The velocity signals that drive the control process may come from several sources. For example, during interactive dragging of a controlled image point, the velocity may represent an estimate of mouse velocity. In keyframing, the velocity represents the derivative of a known trajectory curve $\mathbf{p}_0(t)$. In both cases, position as well as velocity information is available. This extra information can be used to greatly improve tracking accuracy by preventing error accumulation and drift as velocity is integrated over time. We do this by the addition of a simple linear feedback term to our initial statement of the control requirement:

$$\dot{\mathbf{p}} = \dot{\mathbf{p}}_0 - k_f(\mathbf{p} - \mathbf{p}_0),$$

where $k_f$ is a feedback constant, and $\mathbf{p}_0$ is the desired position for $\mathbf{p}$ at the current time. When $\mathbf{p}$ is on target, the feedback term vanishes, but if positional error exists, the velocity is biased in a direction that reduces the error. The feedback term carries straight through the derivation, leading to the following modified form for equation 7:

$$\mathbf{J}\mathbf{J}^T\lambda = \dot{\mathbf{p}}_0 + k_f(\mathbf{p}_0 - \mathbf{p}) - \mathbf{J}\dot{\mathbf{q}}_0. \qquad (11)$$

### 3.7 Tracking a moving point

Until now, we have assumed that the world-space point $\mathbf{x}$ is stationary. A small generalization makes it possible to accurately track a moving point. In keyframe animation, for example, this would allow moving points on objects to be tracked automatically. To make the generalization, we assume that the world-space point moves according to a known function $\mathbf{x}(t)$. In practice, we need only know the point's current position $\mathbf{x}$ and velocity $\dot{\mathbf{x}}$. Since $\mathbf{x}$ now depends on time, an additional term appears in equation 2, the chain-rule expression for $\dot{\mathbf{p}}$, accounting for the part of $\mathbf{x}$'s image velocity due to the motion of $\mathbf{x}$ itself:

$$\dot{\mathbf{p}} = \mathbf{h}'(\mathbf{V}\mathbf{x})\frac{\partial(\mathbf{V}\mathbf{x})}{\partial \mathbf{q}}\dot{\mathbf{q}} + \mathbf{h}'(\mathbf{V}\mathbf{x})\mathbf{V}\dot{\mathbf{x}} \qquad (12)$$

As before, the extra term carries through, adding an additional correction factor to the right hand side of equation 7, yielding

$$\mathbf{J}\mathbf{J}^T\lambda = \dot{\mathbf{p}}_0 + k_f(\mathbf{p}_0 - \mathbf{p}) - \mathbf{h}'(\mathbf{V}\mathbf{x})\mathbf{V}\dot{\mathbf{x}} - \mathbf{J}\dot{\mathbf{q}}_0. \quad (13)$$

This formulation makes it possible to control the *image-space* motion of a point independently of its *world-space* motion. If the image point is pinned, the camera will move as necessary to maintain its position. Both the image point and the world point can be keyframed independently: the camera will move as required to achieve the desired image motion, regardless of the world-space motion of the point.

## 4 A Quaternion Camera

Having developed the through-the-lens equations in generic form, it remains to fill in the blanks. In the equations of the last section, the camera transformation was described in terms of an anonymous matrix $\mathbf{V}$ depending on an anonymous parameter vector $\mathbf{q}$. To proceed, we must say what the function $\mathbf{V}(\mathbf{q})$ actually is. Then we must formulate the equations that are required to evaluate the image point derivative matrix $\mathbf{J}$. If we limit ourselves to image-space controls that can be expressed purely as functions of point positions, then the matrices $\mathbf{V}$ and $\mathbf{J}$ tell us everything we need to know about the camera.

As we noted in section 1, through-the-lens control hides the underlying camera parameterization from the user, so that most of the criteria by which a conventional camera model would be judged do not apply. The model we present in this section is unusual in that a quaternion is used to represent the camera's orientation; we choose it because of the quaternion's ability to represent arbitrary rotations free of singularities and other artifacts. The equations of section 3 are compatible with any camera model. If you prefer another one, the derivation in this section can still serve as a template for the general procedure.

### 4.1 The View Matrix

Our model employs a translation to specify the Lookfrom point and a quaternion to specify orientation. The view matrix $\mathbf{V}$ called for by equation 1 is given by the matrix product

$$\mathbf{V} = \mathbf{P}(f)\mathbf{T}(t_x, t_y, t_z)\mathbf{Q}(q_w, q_x, q_y, q_z), \tag{14}$$

where $\mathbf{P}$ is a matrix for perspective projection with focal length $f$, $\mathbf{T}$ is the matrix for translation by $[t_x, t_y, t_z]$, and $\mathbf{Q}$ is a *quaternion rotation matrix,* performing the rotation specified by the quaternion $\mathbf{q}$, with scalar part $q_w$ and vector part $[q_x, q_y, q_z]$. The camera parameter vector $\mathbf{q}$ is the length-8 vector formed by concatenating the transformation parameters, $[f, t_x, t_y, t_z, q_w, q_x, q_y, q_z]$.

The perspective matrix is a simple one, placing the focal point at the origin and the image plane at distance $f$ from the origin along the $z$-axis, lying parallel to the $xy$-plane:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix}.$$

The translation matrix is the standard one:

$$\mathbf{T}(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The quaternion rotation matrix is a bit more complex. The form given in [22],

$$\mathbf{Q} = 2\begin{bmatrix} \frac{1}{2} - q_y{}^2 - q_z{}^2 & q_x q_y + q_w q_z & q_x q_z - q_w q_y & 0 \\ q_x q_y - q_w q_z & \frac{1}{2} - q_x{}^2 - q_z{}^2 & q_w q_x + q_y q_z & 0 \\ q_w q_y + q_x q_z & q_y q_z - q_w q_x & \frac{1}{2} - q_x{}^2 - q_y{}^2 & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}, \tag{15}$$

assumes that the quaternion has unit magnitude, i.e. that

$$|\mathbf{q}| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1.$$

Otherwise, $\mathbf{Q}$ is not a pure rotation, and shapes will be distorted. This constraint on $|\mathbf{q}|$ means that the camera has only seven true degrees of freedom. To enforce the constraint, it is not sufficient simply to normalize $\mathbf{Q}$ between iterations: in that case, the derivative matrix wouldn't "know" about the constraint, and the control solution would be incorrect. While it would be possible to add the constraint, in differential form, to the control solution, there is a much simpler alternative: in place of equation 15, we express $\mathbf{Q}$ in a form that incorporates the normalization, so that quaternions $\mathbf{q}$ and $\alpha\mathbf{q}$ specify the same transformation, for any scalar $\alpha$. Under this scheme, we must still normalize $\mathbf{q}$ from time to time to prevent the accumulation of numerical errors. The modified version of $\mathbf{Q}$ is most simply expressed as the product

$$\mathbf{Q}_n = \frac{1}{|\mathbf{q}|^2}\hat{\mathbf{Q}},$$

where

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

### 4.2 Evaluating J

Employing the notation of section 3, the image coordinates corresponding to world point $\mathbf{x}$ are given by

$$\mathbf{p} = \mathbf{h}(\mathbf{V}\mathbf{x}) = \mathbf{h}(\mathbf{P}\mathbf{Q}_n\mathbf{T}\mathbf{x}).$$

The rows of $\mathbf{J}$ are formed by differentiating this expression with respect to each camera parameter in turn. To perform the differentiations, we note that each camera parameter influences exactly one matrix in the chain. Therefore, using the rule for differentiation of a product, the derivative of the chain with respect to a parameter is another chain, obtained by replacing the appropriate matrix by its element-by-element derivative. Thus, for example,

$$\frac{\partial \mathbf{V}\mathbf{x}}{\partial t_x} = \mathbf{P}\frac{\partial \mathbf{T}}{\partial t_x}\mathbf{Q}_n\mathbf{x},$$

and we obtain the row of $\mathbf{J}$ corresponding to $t_x$ from

$$\frac{\partial \mathbf{p}}{\partial t_x} = \mathbf{h}'(\mathbf{V}\mathbf{x})\frac{\partial \mathbf{V}}{\partial t_x},$$

where $\mathbf{h}'(\mathbf{V}\mathbf{x})$ is as defined in equation 2, and where

$$\frac{\partial \mathbf{T}}{\partial t_x} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Differentiating each matrix with respect to each parameter on which it depends yields eight matrices in all. The matrix for $\partial \mathbf{T}/\partial \mathbf{t}_x$ is given above—the other two derivatives of $\mathbf{T}$ are likewise trivial. The derivative of $\mathbf{P}$ with respect to its only parameter, $f$, is

$$\frac{\partial \mathbf{P}}{\partial f} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/f^2 & 0 \end{bmatrix}.$$

The four derivatives of $\mathbf{Q}_n$ may be expressed compactly as

$$\frac{\partial \mathbf{Q}_n}{\partial q_w} = \frac{-2q_w}{|\mathbf{q}|^4}\hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2}\begin{bmatrix} q_w & q_z & -q_y & 0 \\ -q_z & q_w & q_x & 0 \\ q_y & -q_x & q_w & 0 \\ 0 & 0 & 0 & q_w \end{bmatrix},$$

$$\frac{\partial \mathbf{Q}_n}{\partial q_x} = \frac{-2q_x}{|\mathbf{q}|^4}\hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2}\begin{bmatrix} q_x & q_y & q_z & 0 \\ q_y & -q_x & q_w & 0 \\ q_z & q_w & -q_x & 0 \\ 0 & 0 & 0 & q_x \end{bmatrix},$$

$$\frac{\partial \mathbf{Q}_n}{\partial q_y} = \frac{-2q_y}{|\mathbf{q}|^4}\hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2}\begin{bmatrix} -q_y & q_x & -q_w & 0 \\ q_x & q_y & q_z & 0 \\ q_w & q_z & -q_y & 0 \\ 0 & 0 & 0 & q_y \end{bmatrix},$$

and

$$\frac{\partial \mathbf{Q}_n}{\partial q_z} = \frac{-2q_z}{|\mathbf{q}|^4}\hat{\mathbf{Q}} + \frac{2}{|\mathbf{q}|^2}\begin{bmatrix} -q_z & q_w & q_x & 0 \\ -q_w & -q_z & q_y & 0 \\ q_x & q_y & q_z & 0 \\ 0 & 0 & 0 & q_z \end{bmatrix}.$$

To evaluate $\mathbf{J}$, we need only implement functions that calculate each of these eight derivative matrices, along with the function that calculates $\mathbf{h}'(\mathbf{x})$. Standard matrix/vector operations are then used to produce the eight rows of $\mathbf{J}$.

## 5 Implementation and Examples

We have implemented through-the-lens control as part of a multi-view, direct manipulation testbed. The program is written in C++ on a Silicon Graphics Iris workstation and uses a toolkit which permits rapid evaluation of dynamically composed functions and their derivatives[14]. All of the examples in this paper can be specified interactively and run at interactive rates on a Silicon Graphics IRIS 4D/210 GTX.

We have experimented with a wide variety of through-the-lens controls including

- the position of a point on the screen,
- the distance between two points on the screen,
- the orientation of two points in the image,
- the ratio between two screen space distances.

All can be interactively specified and connected to vertices in the scene, can be made into hard or soft controls, can serve as constraints, and can be keyframed. Controls that do not have an obvious geometric method for direct manipulation, such as the last three on the list, can be connected to sliders.

The architecture of our system makes it easy to define new types of controls, although this must be done at compile time. Unlike finding new transforms, which entails solving systems of non-linear equations, defining new controls is easy to automate in a general and guaranteed manner since the only required mathematical manipulation is differentiation. We have built automatic code generation tools that facilitate defining new types of controls.

By adding the ability to place boundaries on the values of a control, we have been able to create several interesting through-the-lens features in our system, such as

- bounding a point within a region of the image,
- ensuring that an object does not become larger or smaller than a certain size,
- preventing an object from becoming too much bigger or smaller than another.

We use an active set technique[11] to extend the methods of section 3 to provide the capability of inequality constraints.

These through-the-lens controls work in concert with a variety of world-space controls. Because a camera is a first class object in our system, these controls can be applied to them as well as other objects in the scene. Multiple windows with cameras dynamically assigned to them make it easy to use world and image space controls together in composing an image.

Building on top of a general purpose facility for composing derivatives permits our implementation to exercise the full generality of the methods in section 3 by allowing us to solve simultaneously for camera and object parameters. Through-the-lens controls can therefore affect other objects in addition to the camera. Although removing the restriction that $\mathbf{x}$ does not depend on $\mathbf{q}$ does not require any change to the techniques presented, the pragmatic issues that arise in including parameters of objects other than the camera in $\mathbf{q}$ are beyond the scope of this paper. These issues are discussed in [14, 13, 29]. They permit a unified approach to controlling and constraining all objects, including cameras.

When the state vector includes objects besides the camera, through-the-lens controls provide a way to couple the camera and scene objects. If a point on an object is pinned to a particular place in the image, as the object moves the camera will also change to maintain the constraint. Changing the camera will similarly alter the object. If the camera is locked in place, the object is restricted to locations where its image satisfies the through-the-lens requirements. Adjusting a through-the-lens control can cause both the camera and the scene objects to change,

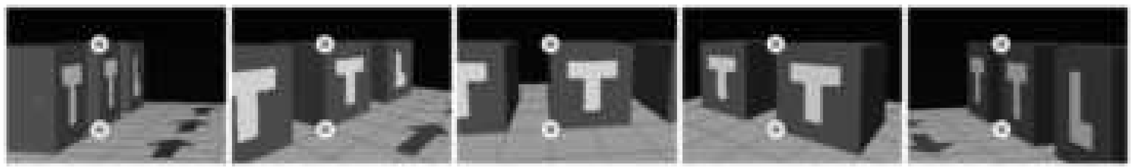

**Figure 1:** Multiple through-the-lens constraints: Multiple through-the-Lens point controls fixate two corners of the center cube. As the camera is translated along the faces of the cube (following the arrows on the floor), it rotates and zooms to maintain the constraints.

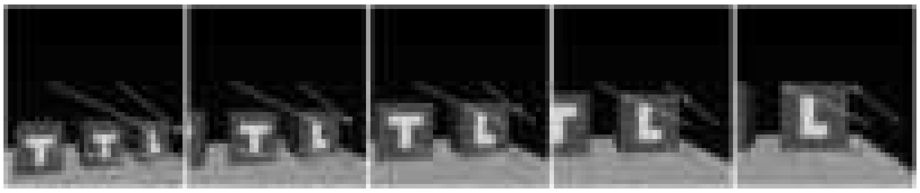

**Figure 2:** Through-the-lens keyframing: Through-the-lens controls are moved along keyframe paths. Each arrow grabs a corner of the cube and pulls it along a path in the image.

such controls permit manipulation of an object in terms of how it appears in the image. In a highly constrained environment, this can make it easy to achieve a desired effect when it is unclear how to do it by controlling the camera and other objects independently.

As a simple example of what through-the-lens controls can do, consider the role of the standard LOOKAT/-LOOKFROM/VUP camera model in our system. The ability to place points in the image and specify the orientations of line segments subsumes the need for this camera model. Although L/L/V is one of several camera models we have coded into our system[2], we typically prefer to use representations like the quaternion-based one in section 4 for their well-behavedness, using through-the-lens controls to point the camera. Even if the L/L/V representation is employed, the user is not restricted to specifying the view using these parameters.

The spacecraft example from the introduction exemplifies the use of through the lens controls to compose an image (Figure 3). Continuing with the example, the constraints used to position the spacecraft and planet can be maintained as the spacecraft flies past the planet to create a fly-by animation, either by coupling the state variables or using the tracking techniques of section 3.7. If the geometry of the scene isn't predetermined, through-the-lens control can help specify it. For example, consider creating a picture of the spacecraft flying by the planet and its moons. If we free the position of the craft, through-the-lens controls can move it so that its position in the image is maintained as we move the camera to find a view which shows the planets and the moons in a desirable manner.

Another use of through-the-lens controls is registering 3D models with photographs. This can be done by displaying a real image as a backdrop and pinning points on the synthesized image to their corresponding locations. Using a least squares technique for overdetermined matrices can allow several points to be specified: the system will move towards the best fit (Figure 4). A viewing transform can be derived for registering 3 points[18], but through-the-lens techniques provide a general method for performing these manipulations.

## 6 Conclusion

As we gain more experience using through-the-lens control, we find more interesting controls and constraints to aid in the process of composing pictures and manipulating scenes. Other additions to through-the-lens manipulation might include using optimization and constraints to help compose images, developing an interface that makes it easier to specify both through-the-lens and world-space controls, inferring constraints to make manipulation easier, and providing a method of detecting and preventing unwanted occlusions. We are beginning to explore using through-the-lens techniques to connect 3D models to real photographs and live video. We are also considering how to use through-the-lens techniques to address issues in planning good camera motions for animations.

Through-the-lens techniques provide a method for manipulating the virtual camera by controlling and constraining image attributes. Interactive control techniques permit the user to control the virtual camera by directly manipulating the image as seen through the lens. The control techniques make it easy to enforce constraints on attributes of the image and scene. The techniques make it simple to implement a wide variety of constraints and controls.


### Acknowledgements

This research was funded in part by Apple Computer, a fellowship from the Schlumberger Foundation, and an equipment grant from Silicon Graphics. We would like to thank Pete Wyckoff for the fractal planets, James Rehg and Michael Kass for providing some references, Steven


[2]Finding the derivatives of this matrix is not for the faint of heart — don't try it without a symbolic mathematics program.

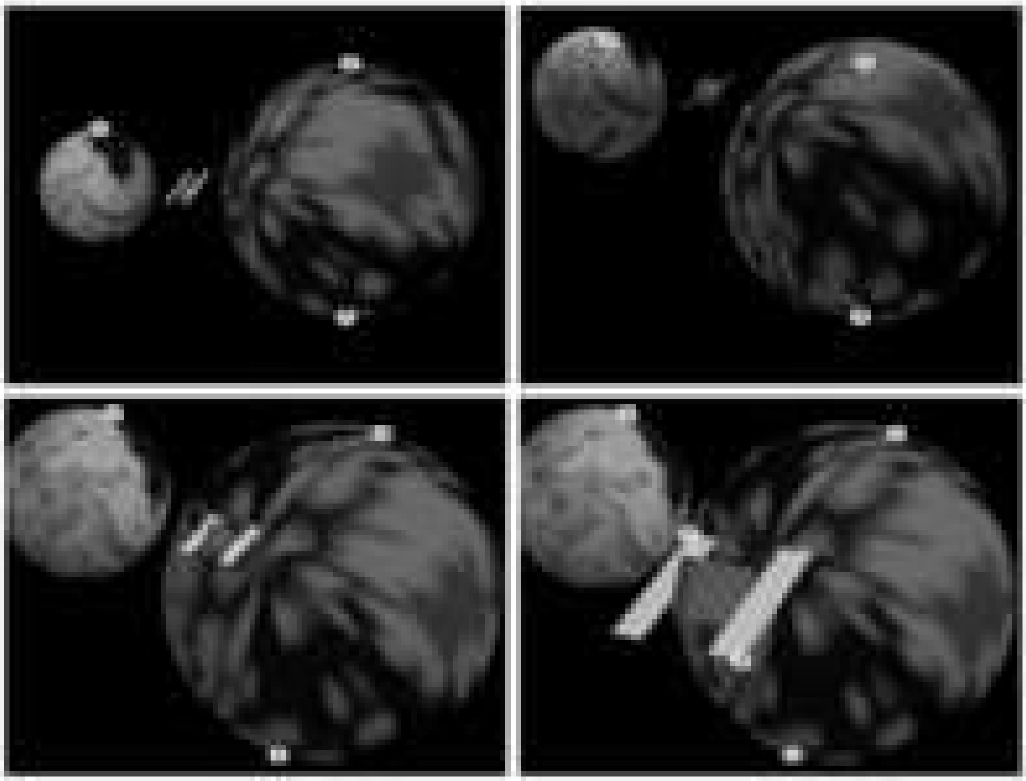

**Figure 3:** Composing an image with Though-the-Lens controls. Through-the-lens point controls are placed at the poles of the planet and the moon (upper left). The leftmost control is moved (upper right). The controls are adjusted further (lower left). After positioning the camera, through the lens controls are used to position the spacecraft (lower right).

**Figure 4:** Decorating the lab with Through-the-Lens controls: The rectangular polygon has the same dimensions as the tabletop. By interactively positioning the vertices to correspond with their respective corners, the virtual camera models the real camera.

Drucker and Tinsley Gaylean for helping renew our interest in camera placement, and Will Welch for providing caffeine and conversation.

## References


[1] Norman Badler, Kamran Manoocherhri, and David Baraff. Multi-dimentional input techniques and articulated figure positioning by multiple constraints. In *Proceedings of the 1986 Workshop on Interactive 3d Graphics*, pages 151–170, October 1986.

[2] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constaints. *Computer Graphics*, 22:179–188, 1988. Proceedings SIGGRAPH '88.

[3] Jim Blinn. Where am I? What am I looking at? *IEEE Computer Graphics and Applications*, pages 76–81, July 1988.

[4] Frederick Brooks. Walkthough – a dynamic graphics environment for simulating virtual buildings. In *Proceedings of the 1986 Workshop on Interactive 3d Graphics*, pages 9–22, October 1986.

[5] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3d rotation using 2d input devices. *Computer Graphics*, 22(4):121–130, August 1988. Proceedings SIGGRAPH '88.

[6] PHIGS+ Committee. Phigs+ functional description, revision 3.0. *Computer Graphics*, 22(3):125–215, 1988.

[7] Steven Drucker, Tinsley Gaylean, and David Zeltzer. CINEMA: a system for procedural camera movements. In *Proceedings of the 1992 Symposium on Interactive Computer Graphics*, pages 67–70, 1992.

[8] Roger Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, 1987.

[9] Sundaram Ganapathy. Decomposition of transformation matrices for robot vision. In *International Conference on Robotics*, pages 130–139, March 1984.

[10] Donald Gennery. Stereo-camera calibration. In *Proc. DARPA Image Understanding Workshop*, pages 101–107, 1979.

[11] Phillip Gill, Walter Murray, and Margret Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.

[12] Michael Gleicher. Briar - a constraint-based drawing program. In *CHI '92 Formal Video Program*, 1992. SIGGRAPH video review, in press.

[13] Michael Gleicher and Andrew Witkin. Differential manipulation. *Graphics Interface*, pages 61–67, June 1991.

[14] Michael Gleicher and Andrew Witkin. Snap together mathematics. In Edwin Blake and Peter Weisskirchen, editors, *Advances in Object Oriented Graphics 1: Proceedings of the 1990 Eurographics Workshop on Object Oriented Graphics*. Springer Verlag, 1991. Also appears as CMU School of Computer Science Technical Report CMU-CS-90-164.

[15] David Lowe. Solving for the parameters of object models from image descriptions. In *Proc. DARPA Image Understanding Workshop*, pages 121–127, 1980.

[16] Jock Mackinlay, Stuart Card, and George Robertson. Rapid controlled movement through a virtual 3d workspace. *Computer Graphics*, 24(4):171–176, August 1990.

[17] Chris McGlone. Automated image-map registration using active contour models and photogrammetric techniques. In *Proceedings of the SPIE, Volume 1070*, January 1989.

[18] Francis H. Moffitt. *Photogrammetry*. International Textbook Company, 1959.

[19] John Platt and Alan Barr. Constraint methods for flexible models. *Computer Graphics*, 22:279–288, 1988. Proceedings SIGGRAPH '88.

[20] William Press, Brian Flannery, Saul Teukolsky, and William Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1986.

[21] K. Schwidefsky. *An Outline of Photogrammetry*. Pitman Publishing Corporation, first english edition, 1959.

[22] Ken Shoemake. Animating rotations with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985.

[23] Mark Surles. Interactive modeling enhanced with constraints and physics – with applications in molecular modeling. In *Proceedings of the 1992 Symposium on Interactive Computer Graphics*, pages 175–182, March 1992.

[24] Konstantinos Tarabamis, Roger Tsai, and Peter Allen. Automated senor planning for robotic vision tasks. In *Proceedings of th 1991 IEEE International Conference on Robotics and Automation*, pages 76–82, April 1991.

[25] Russell Turner, Francis Balaguer, Enrico Gobbetti, and Daniel Thalmann. Physically-based interactive camera motion using 3d input devices. In N. M. Patrikalakis, editor, *Scientific Visualiztion of Physical Phenomena: Proceedings of CG International 1991*, pages 135–145, Tokyo, 1991. Springer-Verlag.

[26] Colin Ware and Steven Osborne. Exploration of virtual camera control in virtual three dimensional environments. *Computer Graphics*, 24(2):175–184, March 1990. Proceedings 1990 Symposium on Interactive 3D Graphics.

[27] William Welch, Michael Gleicher, and Andrew Witkin. Manipulating surfaces differentially. In *Proceedings, Compugraphics '91*, September 1991. Also appears as CMU School of Computer Science Technical Report CMU-CS-91-175.

[28] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(4):225–232, July 1987.

[29] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. *Computer Graphics*, 24(2):11–21, March 1990. Proceedings 1990 Symposium on Interactive 3D Graphics.

[30] Andrew Witkin, Michael Kass, Demetri Terzopoulos, and Kurt Fleischer. Physically based modeling for vision and graphics. In *Proc. DARPA Image Understanding Workshop*, pages 254–278, 1988.

[31] Andrew Witkin and William Welch. Fast animation and control of non-rigid structures. *Computer Graphics*, 24(4):243–252, August 1990. Proceedings SIGGRAPH '90.