Camera Interpolation using Screen-Space Constraints

Leon Barrett
University of California, Berkeley
and
Cindy Grimm
Washington University in St. Louis

This paper demonstrates the use of image-space constraints for key frame interpolation. Interpolating in image-space results in sequences with predictable and controlable image trajectories and projected size for selected objects, particularly in cases where the desired center of rotation is not fixed or when the key frames contain perspective distortion changes. Additionally, we provide the user with direct image-space control over *how* the key frames are interpolated by allowing them to directly edit the object's projected size and trajectory.

Image-space key frame interpolation requires solving the inverse camera problem over a sequence of point constraints. This is a variation of the standard camera pose problem, with the additional constraint that the sequence be visually smooth. We use image-space camera interpolation to globally control the projection, and traditional camera interpolation locally to avoid smoothness problems. We compare and contrast three different constraint-solving systems in terms of accuracy, speed, and stability. The first approach was originally developed to solve this problem [**?**]; we extend it to include internal camera parameter changes. The second approach uses a standard single-frame solver. The third approach is based on a novel camera formulation and we show that it is particularly suited to solving this problem.

Categories and Subject Descriptors: I.3.6 [**Computer Graphics**]: Methodologies and Techniques

General Terms: Measurement; Theory

Additional Key Words and Phrases: camera representation, keyframing, image-space constraints

1. INTRODUCTION

Key framing is an integral part of the animation making process. The user places the camera in a sequence of "key" positions, and the computer produces a set of intermediate camera locations that interpolates between these key frames. This approach has several benefits. First, the user only has to specify a small number of camera positions. Second, the interpolation produces smoother motions than hand-placement does. Third, the timing and number of output frames can be adjusted independently of the key frames themselves.

Camera key framing traditionally treats the camera as just another 3D object in the

Supported in part by NSF grants CCF-0238062 and REU-0138576.

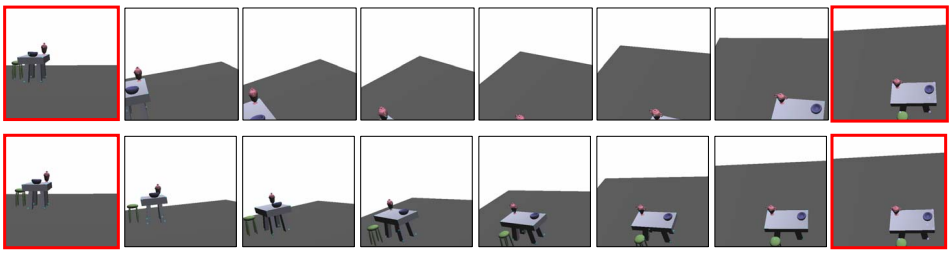

Fig. 1. Top row: Traditional interpolation interpolates the *camera's* 3D position and orientation to produce in-between frames. Bottom row: Screen-space interpolation interpolates an *object's* (in this case, the table's) 2D position and orientation in the image.

scene, with intermediate frames produced by interpolating the position and orientation of the camera in space. Unlike a 3D object, however, the camera's role is to *project* the 3D scene into 2D. The user indirectly controls the projection — how objects in the scene are placed in the 2D image — by adjusting the camera parameters for each key frame. Automatic 3D camera interpolation adds yet another layer of indirection. The net effect is that the user must solve a complicated inverse problem in order to move objects across the 2D scene in the desired manner. Figure 1 shows an example where the user wanted the table to move across the scene while moving the view to the top of the table. Traditional interpolation rotates the table out of the view.

Image-space constraints [Blinn 1988; Gleicher and Witkin 1992] were introduced as a solution to the inverse problem. The user specifies the desired image-space constraints (this object should be here) and the system solves the inverse problem to determine the correct camera path. Unfortunately, except for a few types of animations (flying around an object, panning across a scene), this approach is unstable and difficult to control. There are several reasons for this, but one of the primary problems is that there are multiple ways to move the camera in order to account for changes to the image-space constraints. If the image-space constraints are relatively simple (translation across the image) the system behaves well, but for complicated constraints, such as a rotation plus a translation, the resulting camera paths are jerky, or may pass through un-intuitive camera positions.

This paper proposes a hybrid approach that uses image-space constraints on user-selected scene points to control the *overall* interpolation between key frames, and traditional 3D interpolation *locally* to filter the sequences to produce smoother camera motions. Filtering has an added benefit of stabalizing the image-space constraint approach, as well as reducing the effect of local minima on the constraint solver.

The image-space constraint approach requires a solver — there is no known closed-form solution. We define and contrast three different constraint solver approaches, a Jacobian-based approach [Gleicher and Witkin 1992], a general-purpose simplex solver [Nelder and Mead 1965], and a two-step least-squares solver based on a novel Four-point camera representation. The Jacobian solver was originally developed to solve for image sequences and so produces nice camera paths; unfortunately, it also gets stuck in local minima. The simplex solver is representative of existing computer vision techniques that solve for camera position on a per-frame basis; this approach leads to very accurate constraint matching, but jerky camera motion. We show that the Four-point camera solver combines the accuracy of the second approach with the smoothness properties of the first one. Note that there are

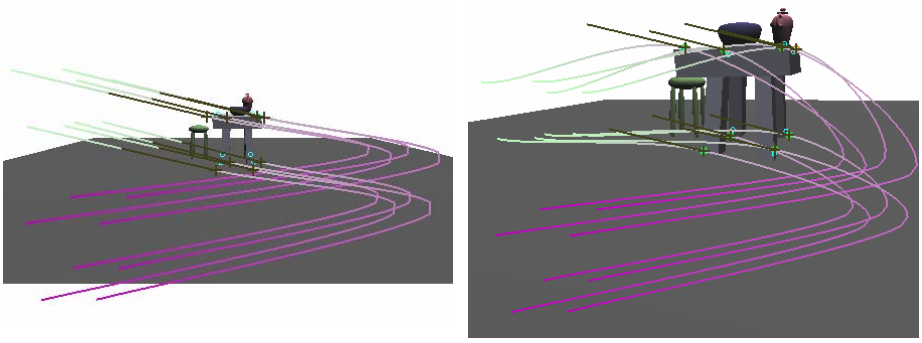

Fig. 2. Adjusting the paths the constrained points take. The points were shifted up and to the right, then scaled up in size.

eleven degrees of freedom needed to define a complete camera projection — six external (rotation and translation) and five internal (focal length, center of projection, aspect ratio, and skew). The Four-point camera model operates on all of the parameters, or just useful subsets, such as position, orientation, and focal length. We also describe modifications to the first two solvers to similarly support all, or subsets, of these parameters.

Interpolating in image-space results in camera sequences with predictable and controlable image trajectories for selected objects, particularly in cases where the desired center of rotation is not fixed or when the key frames contain perspective distortion changes. Traditional interpolation often fails in these cases, causing the object of interest to fly off of the screen (see accompanying videos). To better constrain the trajectory the user must introduce additional key frames, which is a time-consuming process and may produce unwanted wiggles in the camera path. Image-space constraints, however, allow the system to find a smooth interpolation between the two key frames while keeping the object on the desired path.

In addition to the above benefits, image-space interpolation provides a natural mechanism for adjusting both the object's image-space trajectory and projected size while still maintaining smooth camera paths. This is accomplished by allowing the user to edit the screen-space trajectories curves of the selected points. This allows the user, for example, to adjust the path to a more curved one without introducing new key frames (see Figure 2).

The hybrid approach extends to specifying the 3D key frames and the 2D interpolation curves. Our new camera model is both stable under movement of individual points and fast enough for real-time interaction. The user can change the 3D key frame cameras using traditional camera controls or image-space constraints. Similarly, the image-space paths can be adjusted by dragging them in 2D, or by re-attaching points on the 2D constraint paths to their corresponding model point. Changing the camera changes now changes the path points.

1.1 Overview

After covering previous work in Section 2, we describe how we incorporate image-space constraints into the key framing system (Section 3). In Section 4 we define the solvers themselves, including a new camera model. Section 5 covers results for both key framing

and screen-space camera manipulation. We conclude with remarks and future work.

2. BACKGROUND

Traditional 3D key framing requires interpolation of position, orientation, and the intrinsic camera parameters. The position and intrinsic camera parameters are easily interpolated using linear weights. Shoemake [Shoemake 1985] introduced quaternion interpolation for rotations. Barr *et. al.* [Barr et al. 1992] expanded on this idea to produce spline-style interpolations between two quaternions. Kem *et. al.* [Kim et al. 1995] describe how to blend between more than two quaternions, essentially treating each quaternion as a control point on a C^2 spline. Alexa [Alexa 2002] and Hofer and Pottman [Hofer and Pottmann 2004] both offer general-purpose methods for interpolating rigid body motions, although these have yet to be applied to camera motions.

Various systems have been proposed for semi-automatic or automatic control of the camera. Drucker and Zeltzer [Drucker and Zeltzer 1995] proposed a system, based on cinematic rules, for determining the sequence of camera shots in a virtual conversation. The individual camera shots had pre-defined locations for the camera; the system largely focused on rules for switching between them. He et. al. [wei He et al. 1996] expanded on this idea, allowing the individual camera shots to alter the 3D scene slightly to better frame shots. Tomlinson et. al. [Tomlinson et al. 2000] allow the characters themselves, and the evolving story line, to play a role in creating the camera shots. They model the camera as a 3D object attached to the characters via a system of springs and dampers — although the placement of the camera for a particular shot-type is still pre-defined, the dynamics allow some control over the "emotional" content in the camera motion. All of these systems rely on having a small set of (possibly parameterized) pre-defined camera shots to use as building blocks — they do not allow general camera placement. We are interested in creating arbitrary camera paths and rely on filtering and widely-spaced key frames to produce smooth camera sequences.

Halper and Olovier [Halper and Olivier 2000], Gooch et. al. [Gooch et al. 2001], and Bares et. al. [Bares et al. 1998; Bares et al. 2000; Bares et al. 2000] all propose positioning a camera using a set of image constraints. The set of constraints common to all three papers are an object's position, size, and orientation on the screen. Gooch added additional composition rules, such as the rule-of-thirds. Bares included field of view and occlusion rules. Halper used a genetic algorithm solver to search for the camera constraints; Bares and Gooch both took a hierarchical, brute force approach. In their most recent paper, Bares et. al. [Bares et al. 2000] apply a set of heuristic rules to "carve away" parts of the camera space which have no acceptable solution. They also provide a nice interface for specifying the constraints. None of these papers, however, have looked at using image-constraints to move the camera. Theoretically, our constraint-solver plus filter approach would work for these constraints and solvers as well. We have used the Simplex solver with object position, size, and orientation constraints with reasonable success, although these constraint types tend to produce substantial camera changes for even small changes to the constraints.

Blinn [Blinn 1988] pioneered the use of image constraints for camera fly-bys. His approach used the "look at" camera model and constraints that were natural in that model (*e.g.*, center the view on that planet). Gleicher [Gleicher and Witkin 1992] presented a more general approach based on existing Inverse Kinematics solvers. The examples in the paper were primarily under-constrained and used an additional set of soft constraints to

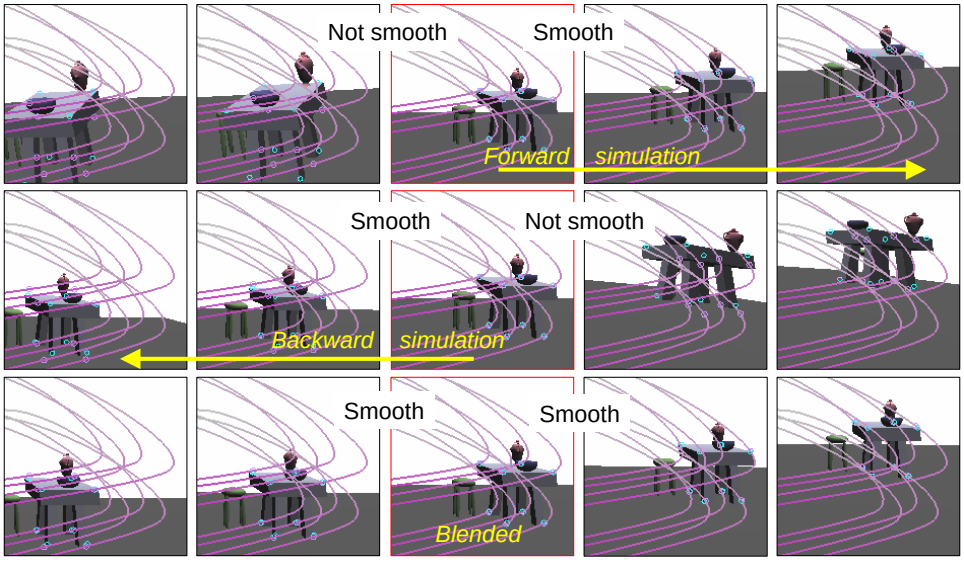

Fig. 3. Interpolating in the forward direction, the backward, then blending the results (Jacobian solver, rotation and translation). The middle frame is one of the key frames. Top row: The solver result matches the constraint curves well after the key frame (rightmost two images) but has drifted substantially by the end of the previous sequence (leftmost two images). Middle row: Running the solver in the reverse directions produces a good match for the end of the sequence (leftmost two images) but not the start (rightmost). Bottom: Blending between the results of the two sequences produces a good match at both ends.

minimize changes to the camera from frame to frame. In our experiments we found that this solver tends to get stuck in local minimum, especially when the constraints are not satisfiable. The overall error also tends to increase over time, especially if, at any point in the sequence, the constraints are not satisfiable (see Figure 3). We present two alternative solvers that overcome the limitations of this solver.

Computer vision researchers have developed many techniques for recovering camera parameters from projections of known points. Early work focused on planar [**?**] or conic [**?**; **?**] calibration patterns. More recently, linear and non-linear techniques for non-planar patterns have been developed [**?**; **?**]. In all of these approaches the intrinsic parameters are assumed to be fixed across the sequence, and only pose parameters (rotation and translation) are recovered per frame. These techniques are primarily concerned with stability (with respect to small perturbations of the tracked points) and accuracy of projection on a frame-by-frame basis; they are not concerned with ensuring visually smooth camera paths. We are interested in finding camera sequences which are smooth and where the internal parameters are allowed to change on a per-frame basis. Because of this, we have used the Simplex solver as a general stand-in for this type of approach. The Simplex solver can easily be adapted to solve for both intrinsic and extrinsic camera parameters, it provides some hysteresis because it can be initialized with the previous frame's solution, and it provides an upper-bound on solve times (*i.e.*, existing approaches in the computer vision literature are faster).

3. KEY FRAME INTERPOLATION

In this section we define what it means to interpolate in screen-space, and explain how we incorporate traditional 3D interpolation to reduce the problem of instability in the image-space constraint solvers. Implementation details of the solvers can be found in Section 4.

To produce image-space constraints the user provides not only the key frames, but also which scene points they wish to constrain. For example, the user might select the four corners of the table, the top and bottom points of an object, *etc.* There is no limit on the number of points, but in practice between four and twelve points works best. If the number of constraints drops below four than the system is under-constrained, which makes the image-space behaviour hard to control. More than twelve points means that any given point has little influence over the result, forcing the user to many points to effect changes.

The next step is to produce an image-space trajectory for each selected point. The system first projects the selected scene point using each key frame. These projected points become the end-points of 2D Hermite curves (see Figure 2); the tangents of the curves are chosen so as to minimize curvature in the trajectory.

Finally, the system finds a sequence of cameras that causes the projected points to follow the curves. This is an iterative process. First, two initial guesses are generated by running the solver forward along the curve, then backward. These two sequences are then blended together using traditional interpolation. The system then alternates between fitting the sequence to the curves and filtering the result, until the solution stabalizes.

More formally, suppose we are constructing n frames from two key frames C_1 and C_n. The user supplies these two key frames and m points P_j, $j \in [1,m]$, in the scene to use as pin points. The optimization proceeds as follows:

(1) Construct m image-space curves γ_j. The end-points of γ_j are C_1P_j and C_nP_j, *i.e.*, the projection of the pin points.

(2) Sample each of the image-space curves at $t_1,\ldots,t_n$, $t_i \in [0,1]$, to produce the image-space constraints for each intermediate frame.

(3) Linearly interpolate any camera parameters that the user has marked as not allowed to change.

(4) Create the forward camera sequence C_i^S. Set camera C_1^S to the starting key frame C_1. Camera C_{i+1}^S is found by running the image-space constraint solver starting at camera C_i^S and with the image-space constraints for intermediate frame $i+1$.

(5) Create the backward camera sequence C_i^E. Set camera C_n^E to the ending key frame C_n. Camera C_{i-1}^E is found by running the image-space constraint solver starting at camera C_i^E and with the image-space constraints for intermediate frame $i-1$.

(6) Create the blended camera sequence C_i. Parameter blend [1] between the two camera sequences to create $C = (1-\frac{i-1}{n-1})C_i^S + \frac{i-1}{n-1}C_i^E$.

This produces an initial camera sequence C_i which meets the image-space constraints, but may have a lot of jitter between frames. We next run the following fit and filter procedure:

(1) Run the image-space constraint solver on camera C_i with image-space constraints for intermediate frame i.

[1] For this paper we SLerp [Shoemake 1985] the rotation and linearly interpolate the remaining camera parameters.

(2) Filter across a small number of frames. Camera C_i is set to $\sum_{j\in(-k,k)} w_j C_{i+j}$.

These two blend and filter steps only need to be run a small number of times (two for the examples in this paper) before the camera sequence C_i stabalizes. The size of the filter controls the trade-off between smoothness and the constraint accuracy; the images in this paper used a filter with a support of approximately five frames.

3.1 Filtering

The filter weights w_j we use come from a Gaussian, clipped at blend values less than 0.001. To apply the filter we simply take a weighted average of the parameters for the corresponding frames, including the quaternion ones. We could use a more advanced method for blending the nearby frames [Hofer and Pottmann 2004; Kim et al. 1995], but have not found it necessary because the blended frames are fairly similar.

An added advantage of blending, then running the image-space constraint solver, is that the blending step can "bump" the constraint solver out of a local minimum.

3.2 Stability and number of frames

Image-space constraint solvers work best when solving for the camera parameters that produce a small change in the image-space constraints. There is also no guarantee that they will find an exact solution, even if one exists. For example, if we interpolate along the curves from one key frame to the next, it is unlikely that the last camera in the sequence will match the next key frame. Intuitively, the reason for this is that there can be multiple ways for the solver to minimize the constraints. The interpolated path may lead the solver into a part of the solution space where there is a local minima, and the global minima is "over the hill". Therefore, we run the constraint solver in *both* directions, then blend the cameras from the resulting sequences, weighting the blend by how close the camera is to the start of the sequence.

One drawback to image-space interpolation is that running the solver with a different number of in-between frames produces a slightly different result (although the results tend to be fairly similar because the points are moving on the same paths, just at different speeds). We can also run into problems if the desired frame spacing results in substantial changes in the constraints from one frame to the next. To address this problem, we choose the sequence t_i independently of the number of final frames, then just re-sample (using traditional interpolation) the camera constraint sequence C_i to get the desired number of frames. This also makes the solve time dependent upon the screen-space change of the constraints, not the final number of frames.

To choose the sequence t_i we ensure that the image-space constraint difference between any two constraint frames is (on average) less than some threshold (as measured in the film plane). Since we have image-space constraint curves, this is simply a matter of searching for a sequence of increasing t_i values such that, for all of the curves γ_j, $||\gamma_j(t_i) - \gamma_j(t_{i-1})||$ is bounded. For the examples in this paper we set a maximum allowable difference of eight pixels in a 512×512 screen, and used binary search to find each of the t_i values.

4. SOLVING CAMERA CONSTRAINTS

In this section we describe three different camera-constraint solvers. Each constraint solver is capable of operating on a subset of the camera parameters (*e.g.*, just rotation and translation) while holding the other parameters constant. The first two solvers operate on the

standard camera model [Foley et al. 1990] developed by Michener and Carlbom [Michener and Carlbom 1980]. The third solver operates on a novel camera model more suited to the image-space constraint problem; this novel model, called the *Four-Point camera*, can be constructed from the traditional model and vice-versa.

The first solver is an extension of Gleicher's [Gleicher and Witkin 1992] Jacobian-based approach to include all of the intrinsic camera parameters. The second solver employs the very general-purpose Simplex, or Amoeba [Nelder and Mead 1965] solver. The third solver uses a two-step approach very similar in spirit to fitting a plane to data points [Fowler and Bartels 1991].

Although all three solvers can handle more complex constraints, for this paper we only employ constraints on 2D point locations. Size, line, and orientation constraints can be phrased as point constraints, although size and orientation can not be specified separately in this case. The problem can be stated as follows. Given a set of 3D points on the model P_i and corresponding image-space points q_i, find a camera matrix C such that:

$$[uvdw]_i^T = CP_i \quad (1)$$

$$E = \sum_i ||q_i - (u_i/w_i, v_i/w_i)||^2 \quad (2)$$

where (u,v,d,w) is the homogeneous transformation of P_i. Note that it is the projection (divide by w) that makes this problem non-linear.

The depth of the projected points, d_i/w_i, does not factor into this error equation. However, we do require that d_i/w_i be between ε and one, *i.e.*, the point is projected in front of the `near` plane, represented by ε, and is not clipped by the `far` plane. This constraint is incorporated into each solver in a different way, as is discussed below.

4.1 Cameras with restricted parameters

We experiment with four different sets of camera parameters. The first set uses the entire (eleven) parameter set. The second set uses all of the parameters except aspect ratio and skew (nine). The third set drops the center of projection as well (seven parameters). The fourth set consists of just the position and orientation of the camera (six parameters).

4.2 Jacobian solver

Full details of the Jacobian approach can be found in Gleicher's paper [Gleicher and Witkin 1992]. The approach begins by calculating the Jacobian using the current camera parameters. The Jacobian is used to determine what *change* to apply to the camera parameters in order to reduce E in equation 2. This approach is identical to the one used in Inverse Kinematics to change joint angles to reach a target position.

We differ in three ways from the original paper. First, we use the full camera matrix (including center of projection, aspect ratio, and skew). Second, rather than calculate the derivatives explicitly using the chain rule, we use an automatic forward differentiation approach FAD [Stauning and Bendtsen]. FAD is a C++ template class that overloads standard mathematical operators; essentially, it computes the derivatives (using the chain rule) simultaneously with computing the equation. To prevent specific camera parameters from changing, we simply do not mark those parameters as differentiable. Third, on each simulation step we check to see that each projected point's depth value is between $[\varepsilon, 1]$; if the depth falls out of this range, we do not take the step. We have experimented with adding

depth constraints to prevent this from happening (forcing the depth toward the mid-depth value 0.5), but were not successful.

We use Runga-Kutta for the simulation step and no soft constraints since (in general) we have sufficient constraints to over-constrain the system, and the points move only a small amount per solve step. We use an adaptive simulation step-size which is reset whenever the image-space constraints change. Our initial step-size is 0.1, which was found (empirically) to be the largest step-size that consistently resulted in a valid step. We use a mass-matrix that weights focal length changes more than other ones.

4.3 Simplex solver

The Simplex solver [Nelder and Mead 1965] is a general-purpose, gradient descent-style solver. Given n parameters it first computes $n+1$ points in the parameter space (usually by jittering each of the parameters). This forms a simplex in n dimensional space. Then, the solver takes the point with the worst error and "flips" it around the simplex to create a new sample point. This continues until the system converges. The simplex solver is very good at both avoiding local minima and dealing with narrow valleys in the error space that arise when the parameters are poorly conditioned. One drawback to the simplex solver (in terms of visualization) is that it bounces around the solution space, so visualizing intermediate steps is not particularly useful.

From a practical standpoint, we provide a non-zero error function (equation 2) that computes the error for the given parameters. This error function is passed to the Simplex solver (the simplex solver does not use derivatives). Inside this error function we construct the camera matrix C, project each of the scene points P_i, then sum the distance to the desired screen point q_i. We augment E with an additional error term E_d that grows rapidly as the projection depth of CP_i approaches zero *i.e.*, the camera eye point moves past the 3D point P_i. ε is the minimum depth distance allowed (typically 0.001):

$$z_i = (CP_i)_d / (CP_i)_w \tag{3}$$

$$E_d = \sum_i \begin{cases} 0 & \varepsilon > z_i > 1-\varepsilon \\ 2(\varepsilon - z_i)^2 & z_i \leq \varepsilon \\ 2(z_i - (1-\varepsilon))^2 & z_i \geq 1-\varepsilon \end{cases} \tag{4}$$

To start the Simplex solver we need to choose initial jitter values for the parameters. Eye position, quaternion angle, alpha, and skew jitter values are 0.2. The quaternion vector jitter is 0.1. The center of projection jitter is 0.05.

4.4 Four-point camera

The Four-point camera was developed in order to overcome the local minima problems arising in the Jacobian solver, and the "jumpiness" of the Simplex solver. Like the Jacobian solver, the intermediate steps the Four-point solver takes "make sense", and can be used to animate the camera change (unlike the Simplex solver). Like the Simplex solver, the Four-point solver is fast and has far fewer local minima problems than the Jacobian solver.

The Four-point camera is motivated by the observation that in three dimensions there is no need to perform a perspective divide — the problem can be re-phrased as ensuring that rays from the eye-point pass through specific points on the film plane. By keeping the camera as a set of vectors we ensure that the influence of each of the camera parameters (the vector components) is much more uniform.

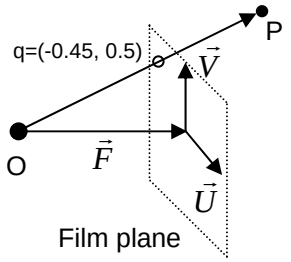

Fig. 4. The Four-point camera model is constructed from an eye point, a vector to the film plane, and two vectors which span the film plane. An image-space constraint q is converted to a 3D point on the film plane; the constraint is satisfied when a ray from the eye to the corresponding point P passes through the film plane at q's 3D location.

The traditional camera matrix is constructed from eleven camera parameters [Michener and Carlbom 1980]. There is no explicit film plane in this model, although it is possible to construct one by slicing a plane through the view volume. An alternative formulation [Eberly 1999] uses four of the five parameters to specify the boundaries of the film plane, the fifth parameter being the focal length. Note that these two camera models, as well as the Four-point camera, are interchangeable — it's simply the *meaning* of the parameters that changes.

In the Four-point camera, the eleven parameters represent the 3D geometry of the camera's eye-point and film plane as a point and three vectors, two for the film plane itself, and the third establishing the film plane's location with respect to the eye-point (see Figure 4.

In the next three sections we define the Four-point camera, how to convert to and from the traditional camera model, and finally, how to express and solve for point constraints.

4.4.1 *Definition.* The Four-point camera is comprised of a point and three vectors, for a total of twelve parameters. However, scaling all three vectors by the same amount results in the same image, so in effect there are only eleven parameters. This corresponds exactly to the fact that, in the traditional model, we can construct the film plane anywhere in the view volume.

The parameters are:

(1) O - the origin of the camera's coordinate system, or the pinhole of the camera.
(2) $\vec{F}$ - the vector from O to the center of the film plane.
(3) $\vec{U}$ - the film-plane vector in the x direction (right vector).
(4) $\vec{V}$ - the film-plane vector in the y direction (up vector).

We will refer to the plane formed by $O+\vec{F}, \vec{U}\times\vec{V}$ as the film plane. Note that the film plane is parameterized by camera coordinates, not screen-space coordinates, and is not pre-scaled to account for $W/H \neq 1$. If the screen is not square then we asymmetrically scale the desired points q_i so that whichever dimension is wider (W or H) maps to $\pm\vec{U}$ (or $\pm\vec{V}$). This places a rectangular viewport inside of the square viewport specified by $\vec{U}$ and $\vec{V}$. We also do not need near and far clipping planes, except for constructing the projection matrix.

$$(q_x, q_y) \;=\; \begin{cases} (q_x, q_y(H/W)) & W \geq H \\ (q_x(W/H), q_y) & H > W \end{cases} \tag{5}$$

Given a point $q = (q_x, q_y) \in [-1,1] \times [-1,1]$ in the image, we can construct the corresponding film plane point by $P = O + \vec{F} + q_x\vec{U} + q_y\vec{V}$. Similarly, given a point P we can find its image coordinates by intersecting the ray $P - O$ with the film plane:

$$P_{\vec{uv}} = O, (P \vec{-} O) \bigcap (O + \vec{F}), \vec{U} \times \vec{V} \tag{6}$$

$$q_x = \frac{(P_{\vec{uv}} - (O + \vec{F})) \cdot \vec{U}}{||\vec{U}||} \tag{7}$$

$$q_y = \frac{(P_{\vec{uv}} - (O + \vec{F})) \cdot \vec{V}}{||\vec{V}||} \tag{8}$$

The intrinsic camera parameters can be expressed as geometric constraints on the vectors. If $\vec{U} \times \vec{V}$ is parallel to $\vec{F}$ then there is no oblique projection, *i.e.*, the center of projection is the center of the film plane. If $\vec{U}$ and $\vec{V}$ are perpendicular to each other than there is no skew. If $\vec{U}$ and $\vec{V}$ are the same length, then there is no aspect ratio distortion. Changing the focal length involves changing the length of $\vec{F}$ while leaving the length of $\vec{U}$ and $\vec{V}$ fixed.

4.4.2 *Conversion to traditional model.* The eye-point of the camera is simply O. The `Look` vector, $\vec{L}$, of the camera is the component of $\vec{F}$ perpendicular to both $\vec{U}$ and $\vec{V}$, and is found by projecting $\vec{F}$ onto $\vec{U} \times \vec{V}$. The camera rotation matrix R is the matrix that takes $\vec{U}$ to the x-axis and $\vec{L}$ to the z-axis, with $\vec{V}$ pointing in the positive y-axis direction (if there is no skew then $R\vec{V}$ will map to the y-axis).

The intrinsic camera parameters are calculated from the geometric relationships between $\vec{U}$ and $\vec{V}$ and their rotated versions $R\vec{U}$ and $R\vec{V}$.

$$\alpha = \frac{||\vec{V} - \vec{U}((\vec{U} \cdot \vec{V})/||\vec{U}||)||}{||\vec{U}||} \quad \text{Aspect ratio: Length of rejection of } \vec{U} \text{ from } \vec{V} \text{ over length of } \vec{U}. \tag{9}$$

$$\vec{L} = \frac{\vec{U} \times \vec{V} \cdot f}{||f||} \tag{10}$$

$$f = \frac{||\vec{U}||}{||\vec{L}||} \quad \text{Focal length: Ratio of } \texttt{look} \text{ vector to } \vec{U}. \tag{11}$$

$$\gamma = -\frac{(R\vec{V})_x}{(R\vec{U})_x} \quad \text{Skew: Amount } \vec{V} \text{ is tilted towards } \vec{U}. \tag{12}$$

$$u_0 = -\frac{(R\vec{F})_y (R\vec{V})_x}{(R\vec{U})_x (R\vec{V})_y} + \frac{(R\vec{F})_x}{(R\vec{U})_x} \quad \text{C of P: Amount of } \vec{F} \text{ which is not perpendicular to the film plane.} \tag{13}$$

$$v_0 = \frac{(R\vec{F})_y}{(R\vec{V})_y} \tag{14}$$

To construct a Four-point camera from the traditional parameters, we construct three vectors from the intrinsic parameters then rotate them by R^T:

$$\left[\vec{F}|\vec{U}|\vec{V}\right] = R^T \begin{bmatrix} u_0/(f\alpha) - v_0 f\gamma & f & -\gamma f \\ v_0 f\alpha & 0 & f\alpha \\ -1 & 0 & 0 \end{bmatrix} \tag{15}$$

4.4.3 *Solving.* Solving for the Four-point camera using a set of image-space constraints is a two-step, iterative process. The key point is that we phrase the problem as a set of 3D geometric constraints. Each image-space constraint q is first converted to a 3D point Q on the film plane. We next find a scale factor d that scales the three vectors so that Q is as close to P as possible. Equation 2 is minimized when $P = O + (\vec{F} + \vec{U}q_x\vec{V}q_x)d$. This is a standard surface fitting problem, which can be solved as follows. First, calculate a scale value d_i for each constraint. Next, find the O, $\vec{F}$, $\vec{U}$, and $\vec{V}$ that minimize (in a least-squares sense)

$$\sum_i ||O + (\vec{F} + (q_i)_x\vec{U} + (q_i)_y\vec{V})d_i - P_i|| \tag{16}$$

Because the d_i are fixed, this can be expressed as a linear problem of the form $Ax = b$. Solve for the new point and vectors, the repeat until E does not change. The general format of the linear equation $Ax = b$ is (dividing both sides of Equation 16 by d_i):

$$\begin{bmatrix} I & (q_1)_x & (q_1)_y & 1/d_1 \\ \vdots & \vdots & \vdots & \vdots \\ I & (q_n)_x & (q_n)_y & 1/d_n \end{bmatrix} \begin{bmatrix} \vec{F} \\ \vec{U} \\ \vec{V} \\ O \end{bmatrix} = \begin{bmatrix} P_1/d_1 \\ \vdots \\ P_n/d_n \end{bmatrix} \tag{17}$$

where each element of A is a 3×3 diagonal matrix and x is a 12×1 matrix, *i.e.*, the vector components are stacked vertically. The d_i are found by solving the 3×1 matrix equation:

$$\left[\vec{F} + q_x\vec{U} + q_y\vec{V}\right]\left[d\right] = \left[P - O\right] \tag{18}$$

where again the vector components are stacked vertically. If we are solving for all eleven parameters then we do not need to stack the vectors, but can instead solve for a 4×3 matrix.

Because the Four-point camera is unique up to a scale, we always re-normalize each iteration to keep $\vec{F}$ unit length.

Equation 17 is a good approximation of $F(\rho)$ when the points Q and P are relatively close together. To ensure this, we find the current image-space projection of the points P_i, then linearly interpolate with the q_i to produce a sequence where the average screen-space distance is below some threshold, typically $1/50$th of the film plane. We fit with this sequence, then let the system converge. This results in no more than 2-8 extra iterations for the key framing case because we have already capped the image-space change per fit (see Section 3.2). Interactive manipulation, of course, provides a natural limit on the amount any given image-space constraint changes between calls to the solver.

4.4.4 *Fitting with linear approximations.* To fit with a subset of the eleven parameters we find a linear approximation that transforms the parameters to the full model. Let $\{\rho\}$

be the subset of parameters to solve for. Let $F(\rho)$ be a function taking ρ to the Four-point model. Then we want to solve:

$$AF(\rho) = b \tag{19}$$

with A and b the same as before. If F is nonlinear, then we can approximate it with a Taylor's series:

$$F(\rho) \approx M_0\rho_0 + M_1\Delta\rho + \ldots \tag{20}$$

where ρ_0 is the current set of camera parameters and $\Delta\rho = \rho - \rho_0$, where ρ is the new set of parameters we're solving for. M_0 and M_1 are both matrices. Replacing $F(\rho)$ with the linear approximation and rearranging Equation 19 so that the constants are on the right-hand side, we get:

$$AM_1\rho = b - AM_0\rho_0 + AM_1\rho_0 \tag{21}$$

This is again a linear equation, and we can solve it with the standard least-squares method. The answer won't be exact, but the errors will be second-order in $\Delta\rho = \rho - \rho_0$, so if the original camera parameters are roughly correct, the convergence should tend to be quadratic: each iteration will double the number of significant digits.

To restrict the parameters to just a rotation and translation, we solve for O and three Euler angles, which are used to rotate the three vectors from their current configuration. Initial values for the Euler angles (ρ_0) are always zero; this avoids the problem of Gimble lock [2]. $F(\rho)$ then constructs $\vec{F},\vec{U},\vec{V}$ by multiplying the current vectors by a linear approximation of the rotation matrix represented by the three Euler angles. To add in the focal length parameter, we simply multiply $\vec{F}$ by a fourth variable which scales $\vec{F}$; the initial value of this scale is one.

To solve for all of the parameters except aspect ratio and skew, recall that when $\vec{V}$ is orthogonal to $\vec{U}$ there is no skew, and if $\vec{V}$ and $\vec{U}$ are the same length then there is no aspect ratio. We can therefore solve for O, $\vec{F}$ and $\vec{U}$, and specify $\vec{V}$ as a rotation around the current film plane vector ($\vec{U} \times \vec{V}$) by 90 degrees.

In all cases, we use the FAD template classes [Stauning and Bendtsen] to calculate the derivatives.

Note that there are no step parameters to adjust with this method; the only simulation parameters needed are the stopping criteria. We stop when the delta change is below 0.001 of the best result so far, or when there have been three iterations with no improvement.

5. RESULTS

Examples are shown in Figure 10 and 1 and in the accompanying video. Each example consists of three evenly-spaced key frames with an extra curve segment placed between the last two key frames. The image-space constraints are placed at the eight outside corners of the table. We have deliberately chosen key frames that differ substantially in order to illustrate the use of image-space constraints.

[2] We also experimented with quaternions, but the derivative for the zero rotation is not well-defined.

In the first example the key frames primarily specify a translation; the curves are shifted to change the path of the table in the latter half of the sequence. The second example consists of rotations, translations, and a focal length change (Figure 1); the image constraint curves are adjusted to control the size of the table. Traditional key frame interpolation in this case causes the table to disappear off of the bottom of the screen. The third example demonstrates interpolation of key frames with center of projection changes. Note that changing the center of projection shifts objects on the screen by an amount related to their depth. This makes it extremely difficult to keep an object centered on the screen using traditional interpolation.

We interpolate the camera parameters that are not free to change, which allows us to restrict the free parameter set without loss of generality, *i.e.*, we can allow just the rotation and translation to change even on a sequence that includes focal length and center of projection changes in the key frames. Obviously, more parameters equates to better satisfaction of the image-space constraints.

Visually, the most noticeable difference is in the *quality* of the movement, not the interpolation (or lack there-of) of the constraints. Restricting the free parameters to just translation and rotation tends to produce blocky movement, with the object's image-space orientation changing fairly rapidly over a small number of frames. Adding in the focal length as a free parameter softens the transitions slightly, and greatly reduces the perspective distortion changes (since the solver can change the object's size in the image by changing either the focal length or changing the distance from the camera to the object). Allowing the center of projection to change dramatically smoothes out the orientation and size changes, at the cost of introducing slightly unusual perspective views in some places. Surprisingly, allowing the skew and aspect ratio to change does not, in general, produce strange results when there are more than four point constraints. This is because the image-space constraints do not (in general) skew or scale asymmetrically.

5.1 Comparison to traditional key framing

We compare our approach to traditional key framing in three ways. First, we take two example sequences which produce unacceptable camera sequences (Figure 1 and the top row of Figure 10) and determine the minimal number of additional key frames needed to produce motion which is visually acceptable. We experiment with both adding key frames by hand and automatically by selecting camera positions from the image-space interpolation sequence. Third, for the example sequence in Figure 10, middle row, we manually add key frames to the original three key frame sequence to create the desired change in the camera path.

Adding and adjusting a key frame takes between one and five minutes. The sequence in Figure 1 required only one additional frame to stabalize the first half of the sequence, but an additional three to keep the table on the screen in the second half. The resulting motion still has very visible "pulsing" artifacts. To determine if this was due to poor key frame placement, we "cheated" and set the key frames using the frames computed from the Four-point camera sequence. Placing key frames at every 8-9 frames was insufficient to smooth the pulsing out; the motion was only smooth with a key frame placed every 2-3 frames.

For the second sequence (Figure 10, top row) it took eight additional key frames to stabalize the trajectory of the table, and this motion again had perceptible artifacts. Again, the motion was smoothed out only by adding key frames every 2-3 frames (copied from

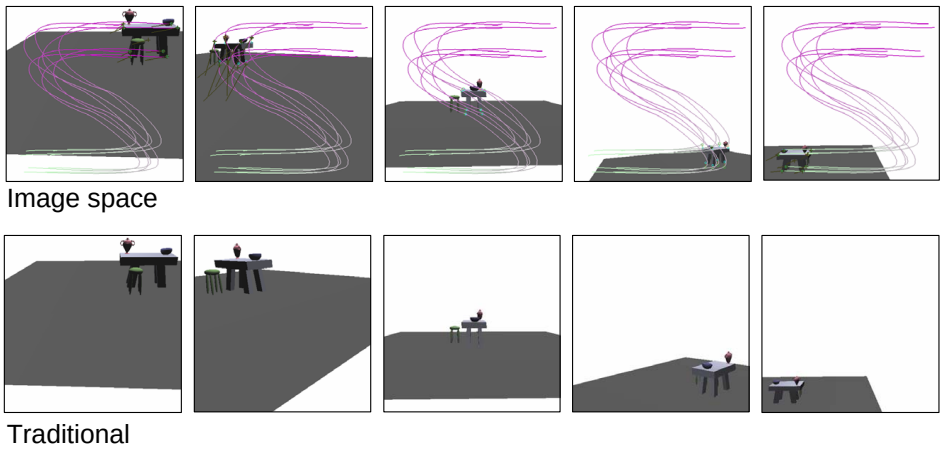

Fig. 5. Constructing similar sequences using image-space interpolation (top) and traditional interpolation (bottom). The table traces out an "S" on the screen, simultaneously rotating 180 degrees and shrinking. The image-space sequence uses three key frames (left, middle, and right) and two additional curve segments (remaining two frames) to constrain the movement. The traditional approach uses 17 key frames (every third shown).

the Four-point sequence).

To compare the overall process, including editing of curves, we compared using the traditional versus the image-space interpolation to construct a camera sequence where the table traces out an "S" while simultaneously rotating 180^{deg} and shrinking (Figure 5). The image-space sequence was created from three key frames (table upper right, middle, and lower left, with 90^{deg} rotations between frames) and two additional pin points (one in the first half of the sequence, one in the second). The curves were edited to produce an "S" shape and to adjust the scale. The entire process took about 10 minutes, about half of which was spent adjusting the tangents of the curves. The traditional sequence was created from 17 key frames, starting with the three original key frames and repeatedly subdividing the sequence and adjusting the new frames to align the key frame with the desired "S" shape. About 10 minutes was spent ironing out the resulting bobbles.

5.2 Key frame statistics

The Simplex solver is the most accurate solver, while the Jacobian is the least accurate (see Graph 6). All of the solvers are stopped when the delta change in the error function (Eq. 2) dropped below 0.001 of the lowest error value found so far. For all three solvers the error decreases as more parameters are allowed to change, which is not surprising.

Graph 7 shows the delta change in the camera's eye position, orientation, and focal length between successive frames, as measured after step five (before blending and filtering). This serves as a rough measure of the stability of the solver.

Running time depends on the cost of each iteration step, multiplied by the number of iterations. Running time increases as the number of parameters increases for both the Jacobian and Simplex approach; the FourPoint solver, however, runs faster because the number of derivative calculations decreases. The total compute time for the sequences shown (60-70 samples of the γ curves, 120 interpolated frames) for the Simplex and Four-point solvers is 1-2 seconds.

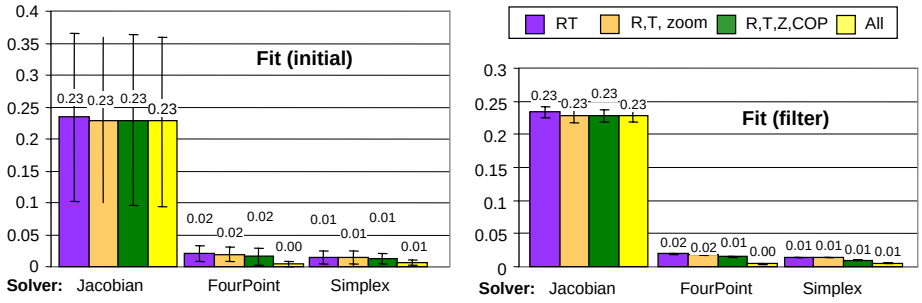

Fig. 6. Accuracy of solvers across all test cases with eight constrained points (Equation 2). Values represent average point distances in camera coordinates.

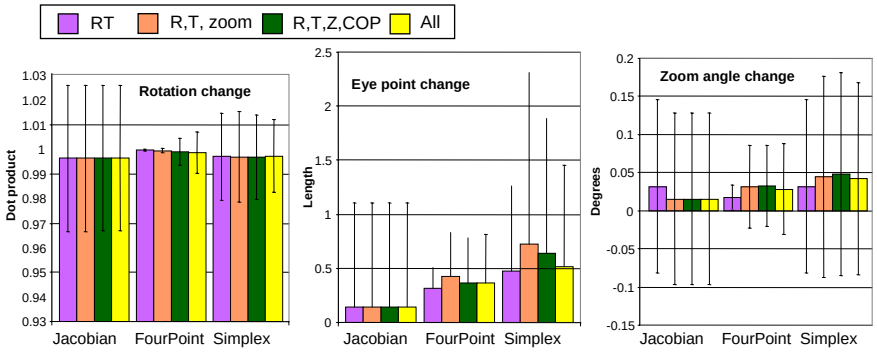

Fig. 7. Stability of the solvers across all test cases, as measured by the change in camera parameters between frames. Rotation is measured as the dot product of the quaternions (one being no change between frames).

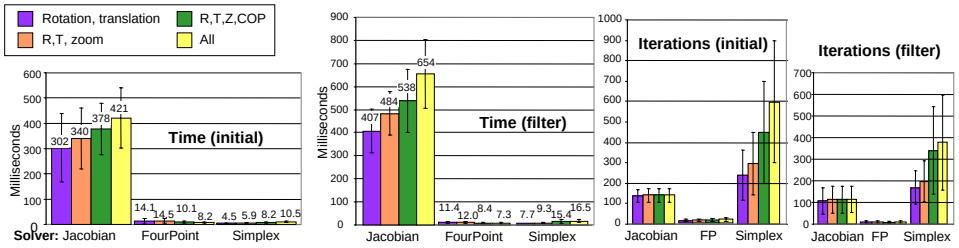

Fig. 8. Average time (in milliseconds) and number of iterations to run one solve step for one frame. Each sequence requires three initial solve steps (forward, backward, and after the blend) and one solve step for each filter step (typically two).

5.3 Editing single point constraints

As an alternative to positioning the camera by directly changing its parameters, we can use the constraint solver to change the camera while dragging the point constraints. We can also visualize the effect of changing the image-space paths by selecting a subset of frames and re-solving for the new image-constraints.

Figure 9 shows the constraint-solver behaviour when using eight points (the accompa-

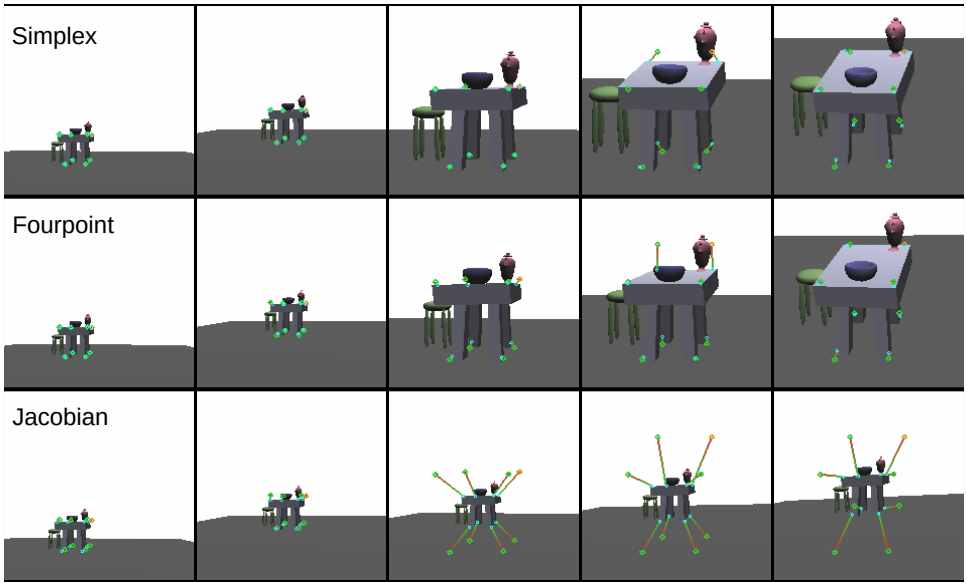

Fig. 9. Editing the image-space constraints. From top to bottom: Simplex, Fourpoint, Jacobian. All three allow rotation, translation, and focal length changes. From left to right: Translate, scale, moving top two points, moving bottom two points.

nying video shows four and eight points). The Jacobian solver works well when all of the points are simultaneously translated, but can get stuck when points are moved individually. It is also ten to twenty times slower than the Four-point and Simplex solvers, with running times increasing as the number of free variables increases.

The Four-point camera is very stable, especially when the center-of-projection is free to change. Solve times for the rotate-translate and rotate-translate plus zoom models are slower than for the other two models in part because of the derivative calculation. The number of iterations is fairly constant across all models, and is determined mostly by the division of the image-space constraints into small-sized steps (Section 4.4.3).

The Simplex solver results in the lowest error solutions, but tends to jump between local minima, resulting in substantial visual changes even with small constraint changes. Solve time increases with the number of variables, with the full, eleven parameter model taking approximately twice as long as the rotate-translate (six parameters) model. These times are comparable to the Four-point solve times.

Qualitatively, all three solvers are more likely to get “stuck”, and then “jump”, when using four or more constraints and restricting the camera parameters to just rotation, translation, and focal length. This is not too surprising; given four (or more) points there are more ways to place those points in 2D then there are ways to project them. The Jacobian solver has the most difficulty recovering from one of these infeasible states, the Simplex solver the least difficulty.

The Four-point and Simplex solvers run at interactive rates; the Jacobian solver is less-so, dropping to 3-10 frames per second.

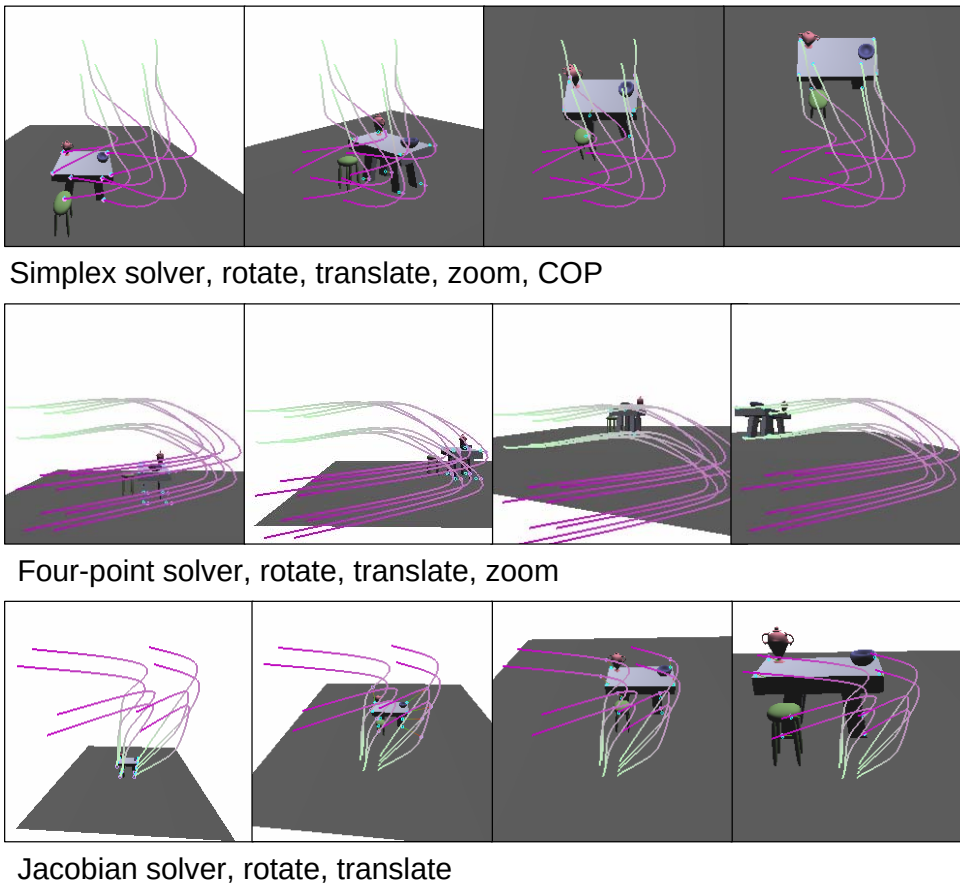

Fig. 10. Example sequences. Each sequence was constructed from three, evenly-spaced key frames. The curves show the desired image-space paths of the model points (shown as circles). Lines connect the desired image-space point with the actual, projected one. Top row: The Simplex solver with nine free parameters. Middle row: The Four-point solver with seven parameters. Bottom row: The Jacobian solver with six parameters.

6. REMARKS

If the desired camera motion is simple, such as a pan or a rotate around an object, then traditional key framing suffices. If the user desires a more complex motion that involves changing the object's orientation while simultaneously moving it across the screen, or incorporating center of projection changes, then image-space constraints can greatly reduce the number of key frames needed to produce the desired motion. This in turn produces a smoother camera path.

6.1 Solver issues

There are four problems that must be over come when using a solver that are not immediately obvious. First, there are *two* valid solutions — one with the 3D points projected into the view volume, the second one the reflected, upside-down image *behind* the eye point (see Figure 11). Both of these solutions minimize error equation 2. One method

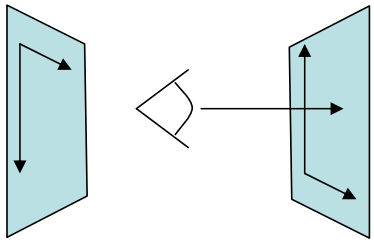

Fig. 11. There are two solutions that minimize image-space constraints, one of which is behind (and upside-down) with respect to the viewer.

of preventing this problem is to put constraints on the depth values as well as the image-space positions (Eq. 4). Unfortunately, this introduces steep regions in the error space and restricts the solution space, both of which can result in poor convergence behaviour.

The second problem is that not all of the values of the parameters are valid or unique — focal length, for example, should not go negative (or, usually, very small or very big) and quaternions are only unique up to a scale and so must be re-normalized between solve steps. Additional constraints can be used to keep the focal length reasonable, or the solver can simply ignore steps that take the parameters out of the valid ranges, but again these approaches affect convergence behaviour.

Third, the parameters are linked. For example, changing the focal length or the eye point's distance to the object both result in nearly identical changes to the image, with the latter introducing a perspective distortion change. To keep an object in the same place on the screen, pairs of parameters must be changed simultaneously. For example, a translation to offset an non-centered rotation, or a translation to offset a center of projection change.

Fourth, changing parameters by uniform amounts does not produce uniform changes in the image. For example, a translate in the viewing direction produces a less noticeable change than one in the perpendicular direction. Focal length, aspect ratio, skew and translation in the viewing direction all act as scales in the image-plane, while center of projection and translation perpendicular to the viewing direction look like image-space translations. Rotation can look like a rotation in the image-plane (spinning the camera around its optical axis), a sheer (rotation into or out of the film plane) or translation (non-centered rotation).

We believe that the Jacobian solver gets stuck in local minima because of these paired parameters and non-uniform effects — the gradient does not provide, for example, the necessary translation to offset a rotation. We have repeatedly seen the case where the following happens: translating the camera in (or out) of the film plane reduces the error slightly for the current frame; at the next frame the image-space constraints move slightly, but the only valid gradient causes the camera to move in (or out) again; over several frames the object will pass out of the view volume. This is especially problematic when the solver can also adjust the focal length, since that compensates for any change in image-space size.

The simplex solver easily finds simultaneous parameter changes that reduce the overall error, but it often jumps from one stable pairing to another. It is the most prone to "flipping" to the behind-the-eye solution and can, on occasion, go far enough down the error slope that it can no longer return to the "correct" solution. The projected-point depth constraint

(Equation 4) greatly reduces this problem.

The Four-point camera avoids these problems by using parameters that have roughly equal effects in image-space and are valid for all values. We deal with the flip problem by forcing all of the distances (Equation 16) to be positive; in practice, we have yet to come across a case where this was needed.

6.2 Comparison of solvers

Both the Simplex and the Four-point solver substantially out-perform the Jacobian solver in terms of speed and fit. The Simplex solver produces a better fit, but at the cost of substantial jitter in the results. Both the Jacobian and the Four-point solver are more stable. The Four-point solver is the most suitable for interactive manipulation because it results in gradual visual changes.

7. CONCLUSION

We have presented a novel camera model that is better suited to solving the image-space constraint problem than current techniques. We compare this model against two others with regards to accuracy, speed, and stability. We show that using local smoothing in combination with image-space constraints produces smooth camera paths even for very different key frames, and in particular, for key frames that have center of projection changes. The use of image-space constraints gives the user greater control over the image position and size of an object without introducing additional key frames.

REFERENCES

ALEXA, M. 2002. Linear combination of transformations. *ACM Transactions on Graphics 21,* 3 (July), 380–387.

BARES, W., GREGOIRE, J., AND LESTER, J. 1998. Realtime constraint-based cinematography for complex interactive 3d worlds. In *Tenth National Conference on Innovative Applications of Artificial Intelligence*. 1101–1106.

BARES, W., MCDERMOTT, S., BOUDREAUX, C., AND THAINIMIT, S. 2000. Virtual 3d camera composition from frame constraints. In *International Multimedia Conference*. 177–186.

BARES, W., THAINMIT, S., AND MCDERMOTT, S. 2000. A model for constraint-based camera planning. In *Smart Graphics: Papers from the AAAI Spring Symposium (Stanford, March 20-22, 2000)*. AAAI Press, 84–91.

BARR, A. H., CURRIN, B., GABRIEL, S., AND HUGHES, J. F. 1992. Smooth interpolation of orientations with angular velocity constraints using quaternions. In *Computer Graphics (Proceedings of SIGGRAPH 92)*. Vol. 26. 313–320.

BLINN, J. 1988. Where am I? What am I looking at? In *IEEE Computer Graphics and Applications*. Vol. 22. 179–188.

DRUCKER, S. M. AND ZELTZER, D. 1995. Camdroid: A system for implementing intelligent camera control. In *1995 Symposium on Interactive 3D Graphics*. ACM SIGGRAPH, 139–144. ISBN 0-89791-736-7.

EBERLY, D. H. 1999. *3D Game Engine Design*. Morgan Kaufmann.

FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1990. *Computer Graphics : Principles and Practice*. Addison Wesley.

FOWLER, B. AND BARTELS, R. H. 1991. Constraint based curve manipulation. *Siggraph course notes 25*.

GLEICHER, M. AND WITKIN, A. 1992. Through-the-lens camera control. In *Siggraph*, E. E. Catmull, Ed. Vol. 26. 331–340. ISBN 0-201-51585-7. Held in Chicago, Illinois.

GOOCH, B., REINHARD, E., MOULDING, C., AND SHIRLEY, P. 2001. Artistic composition for image creation. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*. 83–88.

HALPER, N. AND OLIVIER, P. 2000. Camplan: A camera planning agent. *Smart Graphics: Papers from the AAAI Spring Symposium (Stanford)*, 92—100.

HOFER, M. AND POTTMANN, H. 2004. Energy-minimizing splines in manifolds. *ACM Transactions on Graphics 23,* 3 (Aug.), 284–293.

KIM, M.-J., KIM, M.-S., AND SHIN, S. Y. 1995. A c2-continuous b-spline quaternion curve interpolating a given sequence of solid orientations. In *Computer Animation '95*.

MICHENER, J. C. AND CARLBOM, I. B. 1980. Natural and efficient viewing parameters. In *Computer Graphics (Proceedings of SIGGRAPH 80)*. Vol. 14. 238–245.

NELDER, J. A. AND MEAD, R. 1965. A simplex method for function minimization. In *Computer Journal*. Vol. 7. 308–313.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Computer Graphics (Proceedings of SIGGRAPH 85)*. Vol. 19. 245–254.

STAUNING, O. AND BENDTSEN, C. http://www.imm.dtu.dk/nag/proj_km/fadbad/.

TOMLINSON, B., BLUMBERG, B., AND NAIN, D. 2000. Expressive autonomous cinematography for interactive virtual environments. In *Proc. of the 4th International Conference on Autonomous Agents*. ACM Press, 317–324.

WEI HE, L., COHEN, M. F., AND SALESIN, D. H. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proceedings of SIGGRAPH 96*. Computer Graphics Proceedings, Annual Conference Series. 217–224.