Motion Sketching for Control of Rigid-Body Simulations

JOVAN POPOVIĆ
Massachusetts Institute of Technology
STEVEN M. SEITZ
University of Washington
and
MICHAEL ERDMANN
Carnegie Mellon University

Motion sketching is an approach for creating realistic rigid-body motion. In this approach, an animator sketches how objects should move and the system computes a physically plausible motion that best fits the sketch. The sketch is specified with a mouse-based interface or with hand-gestures, which move instrumented objects in the real world to act out the desired behaviors. The sketches may be imprecise, may be physically infeasible, or may have incorrect timing. A multiple-shooting optimization estimates the parameters of a rigid-body simulation needed to simulate an animation that matches the sketch with physically plausible timing and motion. This technique applies to physical simulations of multiple colliding rigid bodies possibly connected with joints in a tree (open-loop) topology.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction techniques*; G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations—*Boundary value problems*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Physically based animation, animation with constraints, user interface design

1. INTRODUCTION

An animation process begins with a vision of how objects should behave. To transform that vision into a realistic animation, the animator specifies desired behavior precisely and constructs motions that appear realistic. Physical simulation techniques excel at generating realistic motion and have become widely adopted in the computer animation industry [Robertson 1998, 1999, 2001a, 2001b]. The problem of specifying desired behavior, however, remains a difficult and painstaking task. To take advantage of simulation methods, an animator must translate her or his vision of motion into a precise set of forces and torques, each defined at specific instants in time.

In this paper, we propose a sketching interface for specifying rigid-body motions, and develop a robust method for estimating simulation parameters from the sketch. In the sketching paradigm, an animator

This research was sponsored by the National Science Foundation, the Microsoft Corporation, and the Siebel Scholars Program.
Authors' addresses: J. Popović, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139; email: jovan@lcs.mit.edu; S. M. Seitz, Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98195; email: seitz@cs.washington.edu; M. Erdmann, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburg, PA 15213; email: me@cs.cmu.edu.

describes the motion roughly using a mouse or a three-dimensional (3D) input device. Once the sketch is complete, the estimation method converts a sketch into a realistic motion that conforms to physical laws. For instance, an animation of a pencil tumbling in the air and landing in a mug may be sketched by picking up a real pencil with an attached motion sensor and acting out the pencil's behavior in slow motion. Based on this sketch, the parameters of a rigid-body simulation are estimated automatically to construct an animation that moves the pencil in a similar manner, but with physically correct timing and motion. Motion sketching provides an intuitive mechanism for rapidly prototyping animations, but it does not take away control from the animator. A sketch can also specify required goals—for example, a sketch may specify that the pencil must land in the mug. Motion sketching benefits both inexperienced and experienced animators. For inexperienced animators, sketching is an automatic tool for generating rigid-body animations without having to tune the simulation parameters. For animation experts, sketching is a physically based key-frame interpolator and a tool for rapid prototyping.

Internally, our motion-sketching system solves a constrained optimization, which computes the parameters of the rigid-body simulation that matches the sketch. A simulated rigid-body motion depends on many parameters such as initial positions, velocities, and elasticities of each object and surface normals at each collision. A simulator may enhance realism by varying these parameters stochastically to model inhomogeneity of everyday objects [Barzel et al. 1996]. Our method varies these parameters to control simulated motion. In our system, the animator selects the variable parameters and specifies the bounds on acceptable variations. The resulting animations are realistic because the simulator solves the equations of motion that model the physical behavior.

The remainder of the paper is structured as follows. Section 2 provides the background and compares our approach to related techniques. Section 3 provides a mathematical framework for motion sketching. Sections 4 and 5 describe the parameter estimation on continuous and discrete optimization domain, respectively. Section 6 presents the two sketching interfaces and describes motion-sketching examples. Section 7 concludes with a discussion of limitations in our approach and suggests interesting directions for future work.

2. BACKGROUND

The dynamics and control of rigid-body motion permeates fields of computer graphics, biomechanics, robotics, and numerical analysis. This section summarizes previous work and relates it to the motion-sketching problem.

2.1 Rigid-Body Simulation and Control

The motion of rigid bodies is realistically modeled by ordinary differential equations, which are described in standard texts on classical dynamics [Symon 1971]. Applying physical simulation in a computer graphics environment introduces an array of difficult problems, in particular collision detection and collision response [Barzel 1992; Baraff 1992; Mirtich 1996]. In computer animation today, physical simulators integrate differential equations, detect collisions, and apply impulses or contact forces to prevent interpenetration.

Physical simulation alone is not effective for designing animations unless it can be made to attain specific animation goals. Several techniques addressed this problem with force-based constraints and inverse dynamics [Isaacs and Cohen 1987; Barzel and Barr 1988; Platt and Barr 1988]. Because these techniques apply artificial constraint forces to accomplish animation tasks, a rigid body would move as if it were propelled by an attached engine. This behavior is undesirable for many animation tasks such as when creating an animation of a bouncing die or other examples given in this paper.

2.2 Control of Actuated Rigid Bodies

Human and animal locomotion are of great interest to computer animation. Locomotive processes are approximated by rigid-body dynamics of a skeleton which relies on its own muscles for locomotion. Optimal control theory provides a framework for controlling such dynamic systems [Stengel 1994; Brotman and Netravali 1988]. The solution of an optimal control problem maximizes the performance of an evolving dynamic system by computing the control forces and the resulting motion trajectories. Spacetime constraints is an optimal control formulation. It computes realistic motion of a virtual actor by minimizing the power used by the muscles that propel the actor [Witkin and Kass 1988]. Spacetime techniques compute the motion trajectories and the applied control (exerted muscle forces) by solving constrained optimization problems. This general optimization approach permits additional optimization constraints that can specify explicit animation goals. The original spacetime technique discretized both the motion trajectories and the control functions with finite differences [Witkin and Kass 1988]. More recent approaches use polynomial [Cohen 1992] or hierarchical [Liu et al. 1994] basis functions.

Controller approaches have been more successfully adapted to locomotive tasks such as legged locomotion [Raibert and Hodgins 1991], human athletics [Hodgins et al. 1995] and swimming fish [Grzeszczuk and Terzopoulos 1995]. In contrast to spacetime techniques, the controller methods do not discretize motion trajectories. Instead, these techniques devise feedback controllers that affect the motion during a simulation. Driven by the feedback control, motion trajectories are physically simulated by integrating equations of motion. Designing the motion controllers is difficult. Most techniques use biomechanics data to design controllers by hand [Raibert and Hodgins 1991]. Other approaches parameterize control functions and use simulated annealing or genetic algorithms to compute controllers automatically [Hodgins et al. 1995; Ngo and Marks 1993; Grzeszczuk and Terzopoulos 1995]. These techniques are essentially parameter-estimation problems that optimize the parameters of control functions to maximize a locomotion metric such as walking without falling or swimming farther and faster [Pandy et al. 1992; Goh and Teo 1988]. Each technique solves the parameter-estimation problem by single shooting because it computes the motion trajectories throughout the simulation by a single long integration. These applications avoid the pitfalls of single shooting by optimizing control functions during short simulation trials. The controllers are then extended to longer time intervals by cyclic repetition or by transitioning between multiple controllers.

Both simulated annealing and genetic algorithms have slow convergence properties because they sample the optimization space instead of computing mathematical derivatives of the objective function. Gradient-based techniques have also been employed to study human jumps [Pandy et al. 1992], but the derivatives were numerically approximated with divided-difference formulas. Another approach approximates equations of motion with neural networks because the derivatives of a neural network can be computed efficiently and accurately [Grzeszczuk et al. 1998]. The results in this paper suggest that automatic differentiation might be a promising alternative because it computes the derivatives of functions with better accuracy.

2.3 Control of Rigid-Body Simulations

Control of rigid-body simulations without persistent controllers requires different techniques. The control parameters for a rigid body without any self-propelling forces may consist of initial position and velocity only. More control parameters could be added at each impact because small variations in elasticities and surface normals maintain visual plausibility and in some cases enhance realism [Barzel et al. 1996]. Because we are not aware of a perceptual study that identifies permissible variations, our approach allows the animator to specify the acceptable range of variations. With fewer opportunities to control the motion, control techniques must anticipate the effects of rigid-body dynamics. One

approach is to search the space of control parameters with a genetic algorithm [Tang et al. 1995] or a backward search from desired animation goals [Barzel et al. 1996]. The Monte Carlo technique is a more general method that excels at computing the parameters of chaotic rigid-body simulations [Chenney and Forsyth 2000]. This approach may require long running times to find the parameters that yield the desired motion. Interactive control is possible with a gradient-descent method but it requires underdetermined control problems with fewer constraints than degrees of freedom [Popović et al. 2000]. Furthermore, because interaction requires a sequence of physically feasible edits, many animations cannot be easily constructed from scratch. Because each of these techniques computes motion trajectories by a single integration throughout the entire simulation, each has the same drawbacks as all single-shooting methods. Furthermore, prior techniques may not scale as well to simulations of linked rigid-bodies which have high-dimensional state spaces.

2.4 Motion Transformation

Although the spacetime technique does not construct motions to match sketches, it does identify a problem similar to that of motion sketching: given a concise description of animation constraints, construct a physical motion that matches the constraints. Recent spacetime methods propose different optimization objectives and different constraint formulations. Similar to the motion-sketching goal of constructing the motion that best matches the sketch, three recent spacetime methods [Gleicher 1997; 1998; Popović and Witkin 1999a] transform motion-capture data to construct new, distinctly different, motion that preserves the style of the original. These methods rely on motion-capture data that is both realistic and accurate. The realism in the data permits the methods to ignore or to approximate the laws of dynamics and to still yield convincing realistic motions. Dynamic filtering is another technique for correcting physically inconsistent motion, but it requires a motion-captured (physically correct) motion as a reference for the locally optimizing feedback controllers [Yamane and Nakamura 2000].

3. MOTION SKETCHING: PROBLEM DEFINITION

We formulate motion sketching as a parameter-estimation problem that computes the simulation parameters $\mathbf{u}$ and the optimal timing t_i of each sketch sample $\mathbf{s}_i$:

$$\min_{\mathbf{u},t_1,\ldots,t_n} \quad \sum_{i=1}^{n} \left\| \mathbf{p}_i\left(\mathcal{S}_{t_i}(\mathbf{u})\right) - \mathbf{s}_i \right\|^2 \tag{1}$$

$$\text{subject to} \quad \underline{\mathbf{b}} \le \mathbf{c}_s(\mathbf{u}) \le \overline{\mathbf{b}},$$

where the projection function $\mathbf{p}_i(\cdot)$ extracts sketched quantities from the simulation function $\mathcal{S}_{t_i}(\mathbf{u})$, which describes the state of the rigid-body system as a piecewise smooth function of simulation parameters. The vector of simulation parameters $\mathbf{u}$ consists of quantities that affect the motion of each body such as the initial state $\mathbf{q}_0$, external forces $\mathbf{F}$, coefficient of restitution k^e, and surface normals at each collision $\mathbf{n}$. The animator specifies permissible parameter variations with constant upper $\overline{\mathbf{b}}$ and lower $\underline{\mathbf{b}}$ bounds. In general, the hard inequality constraints[1] $\mathbf{c}_s(\mathbf{u})$ also enforce explicit animation goals such as the mug landing of the pencil shown in Figure 3.

If a simulation parameter is not allowed to vary, it can be removed from the vector $\mathbf{u}$ and embedded in the simulation function $\mathcal{S}(\mathbf{u})$. Our current implementation supports variations of the initial state, coefficients of restitution, and surface normals at each collision. The terms in Equation (1) are described in more detail in the following subsections. Sections 4 and 5 describe the multiple-shooting method for solving this estimation problem.

[1]These inequality constraints express equalities with equal upper and lower bounds: $\underline{\mathbf{b}} = \overline{\mathbf{b}}$.

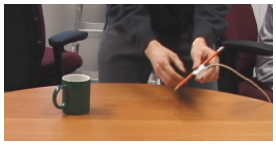

Fig. 1. The movement of an instrumented pencil acts out a sketch.

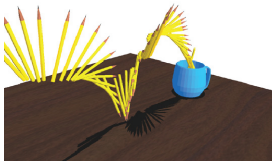

Fig. 2. A rendering of the acted motion applied to a computer-generated pencil.

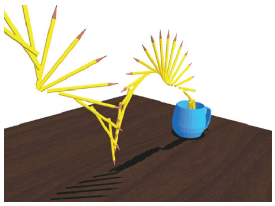

Fig. 3. The physical motion that matches the sketch.

3.1 Motion Sketch

A sketch specifies desired trajectories for animated objects. One way of generating a sketch is to have the animator pick up and move real objects with attached motion sensors. For example, Figure 1 shows an animator moving an instrumented pencil to act out an animation of a pencil bouncing, twirling in the air, and landing in a mug. The trajectories specify soft constraints that describe how the pencil should move. A sketch may also designate hard kinematic constraints. In this example, the animator requires that the pencil lands at a precise location, or that it collides with another object. The sketched motion need not be physically consistent. For example, the animator may move the pencil in slow motion. The motion, shown in Figure 3, is the result of a rigid-body simulation with the appropriate simulation parameters. The parameters are estimated to minimize the least-squares error between the sketch samples, shown in Figure 2, subject to also satisfying the hard landing constraint.

A sketch is represented by a sequence of vectors $\mathbf{s}_1, \ldots, \mathbf{s}_n$, called *sketch samples*. Each sketch sample $\mathbf{s}_i$ could specify the desired generalized state completely. More generally, it specifies a function $\mathbf{p}_i(\mathbf{q})$ of the generalized state. For example, a sketch that describes the center-of-mass trajectory would use a projection function that computes the location of the center-of-mass from the components of a generalized state. Each sketch sample may optionally specify the time t_i at which it occurs. Our method infers the timing for all other sketch samples, while maintaining the temporal order implied by the sequence of sketch samples.

3.2 Rigid-Body Simulation

A system of one or more rigid bodies is described by a generalized state vector $\mathbf{q}$ whose components are the generalized coordinates and velocities of the bodies. The behavior of a system is described by its equations of motion [Symon 1971], which we write in vector form as a coupled first-order differential

equation,

$$\frac{d}{dt}\mathbf{q}(t) = \mathbf{f}(t, \mathbf{q}(t)), \tag{2}$$

where the generalized force $\mathbf{f}(t, \mathbf{q}(t))$ is derived from Newton's law. For example, the equations of motion for a single rigid body with mass m and inertia tensor $\mathbf{I}(t)$ are

$$\frac{d}{dt}\begin{pmatrix} \mathbf{x}(t) \\ \mathbf{r}(t) \\ m\mathbf{v}(t) \\ \mathbf{I}(t)\boldsymbol{\omega}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{v}(t) \\ \frac{1}{2}\boldsymbol{\omega}(t)\mathbf{r}(t) \\ \mathbf{F}(t) \\ \boldsymbol{\tau}(t) \end{pmatrix}.$$

In this example, the generalized state consists of a vector $\mathbf{x}(t)$ and a unit quaternion $\mathbf{r}(t)$ describing the body's position and orientation, and vectors $\mathbf{v}(t)$ and $\boldsymbol{\omega}(t)$ describing the body's linear and angular velocity. The generalized force includes the external linear force $\mathbf{F}(t)$ and torque $\boldsymbol{\tau}(t)$. The same Equation (2) can describe the dynamics of any mechanical system that can be modeled as a set of rigid bodies connected with joints. In that case, the generalized coordinates represent the available degrees of freedom (the reduced coordinates) in the mechanical system.

The equations of motion describe the mechanical system in free-flight, when there are no collisions. The complex physics of a collision are approximated in computer animation with a simple collision model that relates relative velocities before and after the impact [Moore and Wilhelms 1988]. The model describes the elastic and inelastic impacts by applying instantaneous impulses to each colliding body. The impact model computes the generalized state $\mathbf{q}^+$ an instant after the collision by adding the impulse $\mathbf{J}$ to the generalized state $\mathbf{q}^-$ an instant before the collision:

$$\mathbf{q}^+ = \mathbf{q}^- + \mathbf{J}(\mathbf{q}^-). \tag{3}$$

The applied collision impulse $\mathbf{J}$ depends on the velocities before the collision. The impulse also depends on the intrinsic properties of the colliding body: the surface normal affects the direction of the impulse, the surface roughness and pliancy determine the friction and the elasticity of the collision, and the inertial properties determine the magnitude of the impulse.

The rigid-body simulator integrates equations of motion, detects collisions, and applies impulses to compute the physical motion $\mathbf{q}(t)$. The simulation function $\mathcal{S}$ describes this computation as a function of the simulation parameters $\mathbf{u}$:

$$\mathbf{q}(t) = \mathcal{S}(t, \mathbf{u}). \tag{4}$$

The vector of simulation parameters $\mathbf{u}$ consists of all parameters that affect the simulated motion, such as initial positions and velocities, external forces, coefficients of elasticity and friction, and inertial body properties. The simulation function $\mathcal{S}$ maps the simulation parameters $\mathbf{u}$ into a function that describes the state as a function of time $\mathbf{q}(t)$. To simplify notation, we also introduce shorthands $\mathcal{S}_{t_f}$ and $\mathbf{q}_f$, which denote the respective simulation function and state at a specific time instant t_f:

$$\mathbf{q}_f = \mathcal{S}_{t_f}(\mathbf{u}) = \mathcal{S}(t_f, \mathbf{u}). \tag{5}$$

In principle, the animator could tune the parameters $\mathbf{u}$ until the simulation produces the desired motion. Although most parameters have an intuitive physical meaning (mass, initial velocity, etc.), tuning parameters by hand is difficult. Because the simulation function is discontinuous and nonlinear, in most cases a parameter such as the initial velocity has an unpredictable effect on the simulated motion. Instead, our method estimates the simulation parameters automatically. The current implementation estimates any subset of parameters including initial positions, initial velocities, coefficients of restitution, and perturbations of surface normals at each collision.

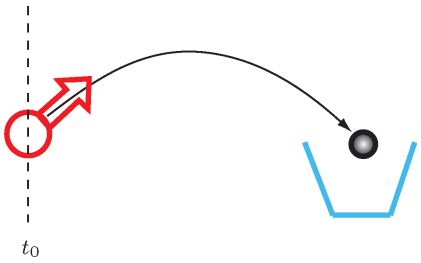

Fig. 4. Single shooting represents the entire motion with the initial conditions at the beginning of the simulation. In this example, the motion of a particle is represented by its initial position and velocity.

3.3 Parameter Estimation

Parameter estimation is a class of techniques for identifying unknown parameters of a mathematical model that best fit observed real-world measurements [Stengel 1994]. In the context of motion sketching, parameter estimation identifies the simulation parameters $\mathbf{u}$ that fit the simulated motion, $\mathbf{q}(t) = \mathcal{S}(t, \mathbf{u})$, to a motion sketch. Our approach could use several metrics to measure how well the motion $\mathbf{q}(t)$ fits the sketch. We use the least-squares metric because it is a maximum-likelihood estimator[2] with nice numerical properties [Stengel 1994]. Alternatively, the estimation could use a weighted least-squares metric without any performance penalty. A weighted metric could better combine the errors in position and orientation, which are measured in different units. The metric matches the motion to sketch samples, which describe objectives for the animation. For example, if the sketch samples describe that the pencil should twirl in the air, the least-squares metric is minimized by a motion of a pencil twirling in the air. The hard sketch constraints, such as a requirement that the pencil lands in a mug, are not enforced by the objective function. Instead, the parameter estimation problem is further constrained to enforce the hard sketch constraints.

This parameter-estimation formulation is closely related to optimal-control methods. For example, a spacetime technique [Witkin and Kass 1988] parameterizes the actuating control curves $\mathbf{u}(t)$ with polynomial basis functions. If the coefficients of the basis functions are also added to the finite-dimensional control vector $\mathbf{u}$ then solving the parameter-estimation problem is equivalent to solving an optimal control problem. Pandy and colleagues [1992] adapted this approach to estimate biomechanical parameters of a human jump. We favor casting the motion-sketching problem as a parameter-estimation problem, instead of optimal control, to emphasize its finite nature: parameter estimation computes the finite-dimensional vector of simulation parameters $\mathbf{u}$ that produce a motion matching the sketch.

4. PARAMETER ESTIMATION WITH SHOOTING

Parameter estimation is a variational problem that optimizes motion trajectories. A shooting method parameterizes entire motion trajectory with a finite set of parameters. For example, single shooting, illustrated in Figure 4, parameterizes the motion of a particle with its generalized state (position and velocity) at the initial time. A particle is then shot from its initial state to compute the subsequent trajectory. Single shooting is intuitive and easy to implement, but it has two significant drawbacks. First, parameter estimation is unstable because single shooting uses large integration intervals to compute trajectories. Second, the estimation is difficult to initialize—a critical step for success of any nonlinear optimization—because a single initial state describes the entire trajectory. Imagine trying

[2]Least-squares fitting is a maximum likelihood estimator if the sketch samples are independently distributed according to a normal (Gaussian) distribution of constant standard deviation.

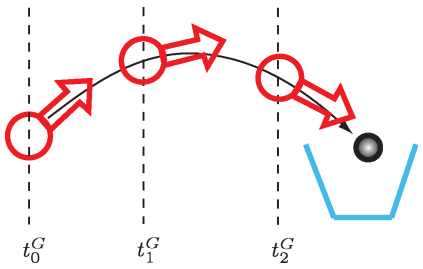

Fig. 5. Multiple shooting splits the motion into smaller integration intervals. In this example, the motion of a particle is represented by its initial position and velocity at the beginning of each shooting segment.

to guess the parameters require to make a die bounce twice across a table and land with a desired outcome.

Multiple shooting (Figure 5) is a numerically stable and robust technique for parameter estimation. It eliminates long integration intervals by splitting the simulation into a series of arbitrarily short intervals, and it simplifies parameter initialization by exposing the generalized state at more than just the initial time. Multiple shooting is well suited for mixed, continuous and discrete, optimization domains. This section describes the standard multiple-shooting formulation, which assumes a continuous optimization domain and fixed shooting intervals. In Section 5, we describe extensions required to adapt this approach to control the motion of colliding bodies.

4.1 Standard Multiple Shooting

A shooting technique for parameter estimation computes the motion $\mathbf{q}(t)$ of a rigid body system by evaluating the simulation function [Bock 1983]. The simulation function $\mathcal{S}(t, \mathbf{u})$, defined in Section 3.2, integrates equations of motion to parameterize the motion $\mathbf{q}(t)$ with simulation parameters $\mathbf{u}$. Multiple shooting partitions the integration interval $[t_0, t_f]$ to compute the motion on each subinterval separately. A time grid,

$$t_0 = t_0^G < t_1^G < \cdots < t_m^G = t_f,$$

defines m subintervals. Although partitioning is arbitrary, smaller intervals improve stability at the cost of increasing the size of the optimization problem. The animator should also be allowed to partition the motion at arbitrary times to ease the initialization of parameters.

After partitioning, the motion $\mathbf{q}(t)$ is defined by m functions:

$$\mathbf{q}(t) \begin{cases} \mathcal{S}(t, \mathbf{u}, \mathbf{q}_0) & \text{if} \quad t \in \left[t_0^G, t_1^G\right), \\ \qquad\qquad\vdots & \\ \mathcal{S}(t, \mathbf{u}, \mathbf{q}_{m-1}) & \text{if} \quad t \in \left[t_{m-1}^G, t_m^G\right]. \end{cases}$$

Each function is a solution to an initial value problem with initial conditions that specify the simulation parameters $\mathbf{u}$ and the initial state $\mathbf{q}_i = \mathbf{q}(t_i^G)$. Given the simulation parameters $\mathbf{u}$ and m initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$, the rigid-body simulator computes the simulation function that specifies the state of the mechanical system at every point in time:

$$\mathbf{q}(t) = \mathcal{S}(t, \mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}), \quad t \in [t_0, t_f].$$

In general, motions $\mathbf{q}(t)$ computed in this manner are not continuous. The sequence $t_1^G, \ldots, t_{m-1}^G$ marks the times at which the neighboring trajectories may not match up. Continuity constraints $\mathbf{c}_c$ are added to enforce continuity between motion segments, ensuring that the motion $\mathbf{q}(t)$ is continuous in

both position and velocity[3]:

$$\mathbf{c}_c(\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}) \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{q}_1 - \mathcal{S}_{t_1^G}(\mathbf{u}, \mathbf{q}_0) \\ \vdots \\ \mathbf{q}_{m-1} - \mathcal{S}_{t_{m-1}^G}(\mathbf{u}, \mathbf{q}_{m-2}) \end{pmatrix} = 0.$$

In this formulation, the simulation parameters $\mathbf{u}$ and initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$ are the unknowns. The parameter estimation computes their values to satisfy the sketch constraints $\mathbf{c}_s$ and to enforce the continuity constraints $\mathbf{c}_c$:

$$\min_{\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}} \sum_{i=1}^{n} \left\| \mathbf{p}_i\left(\mathcal{S}_{t_i}(\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1})\right) - \mathbf{s}_i \right\|^2 \tag{6}$$

$$\text{subject to} \quad \begin{cases} \underline{\mathbf{b}} \leq \mathbf{c}_s(\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}) \leq \overline{\mathbf{b}}, \\ \mathbf{c}_c(\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}) = \mathbf{0}. \end{cases}$$

If the simulation function $\mathcal{S}(t, \mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1})$ is continuous with respect to the simulation parameters $\mathbf{u}$ and the initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$, the multiple-shooting formulation yields a constrained nonlinear optimization problem on a continuous domain which can be solved efficiently with an iterative optimization method [Gill et al. 1989].

4.2 Multiple-Shooting Optimization

Our implementation uses sequential quadratic programming (SQP) to solve the parameter-estimation problem. SQP methods solve constrained nonlinear optimization problems by generating a sequence of iterates that converge to a solution point satisfying the Karush-Kuhn-Tucker conditions of optimality [Gill et al. 1997]. Each iterate is the result of a quadratic programming subproblem, which is derived from the original nonlinear optimization. Simulated annealing and genetic algorithms are two alternative techniques that have been applied previously for single-shooting variants [Hodgins et al. 1995; Ngo and Marks 1993; Grzeszczuk and Terzopoulos 1995]. Despite their success for single shooting, these techniques are inappropriate for multiple shooting because they cannot enforce continuous state trajectories with continuity constraints.

Initialization. The shooting time grid exposes initial conditions, which describe the position, orientation, and velocity of each body. In contrast to single shooting, which offers no help with parameter initialization, the animator can use these initial conditions to convert a rough animation concepts into a good initial approximation. For highly nonlinear constrained optimization problems, a good initial approximation is a key requirement for convergence. Parameter-estimation problems have many optimal solutions and a good initialization will lead to a locally optimal solution that is also the global optimum. Section 6 describes the initialization specifics for each sketching interface.

Derivatives. SQP methods solve the parameter-estimation problem efficiently but require accurate mathematical derivatives of the constraints and the least-squares objective function. With shooting methods, these derivatives depend directly on the derivative of the simulation function (see Equation (8) for an example). Because the simulation function is extraordinarily complex for most mechanical systems, previous shooting techniques either abandon SQP methods in favor of simulated annealing and genetic algorithms [Hodgins et al. 1995; Ngo and Marks 1993; Grzeszczuk and Terzopoulos 1995] or

[3]Velocity is not continuous for colliding bodies. In this case, the states must still match up through the impulse Equation (3).

approximate the derivatives with divided differences [Pandy et al. 1992]. Our system computes derivatives with automatic differentiation, which mechanically breaks down the simulation function into elementary operations (addition, multiplication, etc.) and composes their derivatives by repeatedly applying the chain rule [Griewank and Corliss 1991]. Automatic differentiation is a powerful technique that can compute derivatives of computer programs with branches, loops, and subroutines [Bischof et al. 1996]. The examples in this paper demonstrate that it remains practical for motions of complex mechanical systems such as rigid bodies constrained by joints and hinges. The original spacetime method also used a form of automatic differentiation [Kass 1992], but in that formulation the derivatives were not numerically integrated. In contrast to divided differences, automatic differentiation computes the derivatives with machine precision accuracy. This added accuracy is critical because SQP methods use the derivatives to estimate Hessians, the second-order derivatives of the constraints and the objective function [Gill et al. 1989].

Automatic differentiation may not generalize to some important cases such as motions with resting (sustained) frictional contact. For these motions, the linear complementarity formulation appears to be the most robust framework for computing the contact forces and impulses [Baraff 1994]. In general, linear-complementarity problems are best solved with iterative optimization procedures, which may be too complex for automatic differentiation. In these cases, the derivatives could be approximated with divided differences during contact and with automatic differentiation everywhere else. Furthermore, linear complementarity is only one of many possible formulations. The standard computer animation practice is to employ simpler, automatically differentiable, formulations for modeling sustained frictional contacts such as traction for automobiles [Grzeszczuk et al. 1998] and snakes [Miller 1988; Grzeszczuk and Terzopoulos 1995].

Sparsity. Multiple shooting increases the size of optimization problems by adding the initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$ to the optimization unknowns. Sparse optimization solvers can combat the added complexity by exploiting the sparse structure in Jacobian matrices. The Jacobian matrices are always sparse because an initial state at the beginning of a segment does not affect the motion trajectories in preceding segments. Frequently (see Section 5 for an exception), the succeeding segments are also unaffected and the Jacobian matrix of the continuity constraints $\mathbf{c}_c$ with respect to initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$ is a banded matrix:

$$\begin{pmatrix} -\frac{\partial \mathcal{S}_{t_1^G}}{\partial \mathbf{q}_0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\frac{\partial \mathcal{S}_{t_2^G}}{\partial \mathbf{q}_1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -\frac{\partial \mathcal{S}_{t_{m-1}^G}}{\partial \mathbf{q}_{m-2}} & \mathbf{1} \end{pmatrix},$$

where matrices $\mathbf{0}, \mathbf{1}$, are zero and identity matrices.

4.3 Time Assignment

The animator can specify the timing required for any sketch sample. For other sketch samples without the explicit timing information our system automatically computes the timing information t_i from sketched trajectories. The iterative algorithm, shown in Figure 6, alternates between the two steps: the assignment step and the optimization step. First, the assignment step estimates the timing information for each sketch sample t_i by pairing the sample $\mathbf{s}_i$ with the closest state $\mathbf{q}(t_i)$. Second, given the sample timing t_i, the optimization step solves the parameter-estimation problem described in Section 4.1.

compute initial approximation $\mathbf{q}(t)$
repeat
  /* *Assignment Step* */
  for $i = 1$ to n
    $t_i = \arg\min_{t_i} \|\mathbf{p}_i(\mathbf{q}(t_i)) - \mathbf{s}_i\|^2$
      such that $t_1 < \cdots < t_i$
  end for
  /* *Optimization Step* */
  estimate parameters as in Section 4.1
until sample times $t_1, \ldots, t_n$ converge

Fig. 6. The iterative algorithm assigns timing information to each sketch sample before solving the parameter estimation problem.

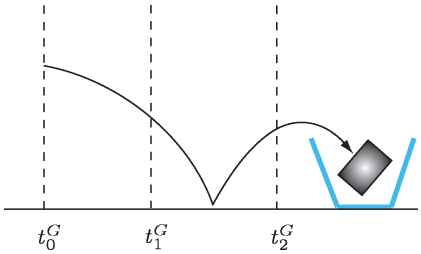

Fig. 7. A standard time grid subdivides the motion arbitrarily. The motion in the second interval $[t_1^G, t_2^G)$ may not be continuous with respect to the initial state $\mathbf{q}_1 = \mathbf{q}(t_1^G)$ because a small change in the state might switch the colliding vertex and thus abruptly change subsequent motion.

For efficiency, the optimization step can run for few iterations before the assignment step reestimates the timing. After several repetitions the timing estimates change little with each repetition and the parameter estimation continues until optimal parameters are identified. This iterative assignment-optimization algorithm is similar in spirit to a method for registering 3D shapes from multiple views [Besl and McKay 1992]. Registration aligns the shapes, initially expressed in multiple coordinate systems, to express the data in a single object-coordinate system. For registration, this method alternates pairing the points in overlapping shape regions and optimizing the coordinate transformations for the current point pairing.

5. MULTIPLE SHOOTING WITH DISCONTINUITIES

Standard multiple shooting, described in the previous section, works for continuous optimization problems. In this section, we extend the standard approach with a sliding time grid to abate the problems with collisions, which introduce discontinuities in the simulation function (Figure 7). In our approach, some points on the new time grid correspond to collision times. Because collisions times change with the parameter values, we generalize the standard shooting technique with a sliding time grid that adapts to changes in collision times.

5.1 Time Grid with Collisions

The alignment step (Figure 8) ensures that each collision corresponds to a point on the time grid. This alignment adds a shooting segment for each collision-free motion. Collision-free segments could also be further segmented to simplify parameter initialization or to shorten integration intervals for better numerical stability. Because rigid-body simulations model collisions with instantaneous impacts,

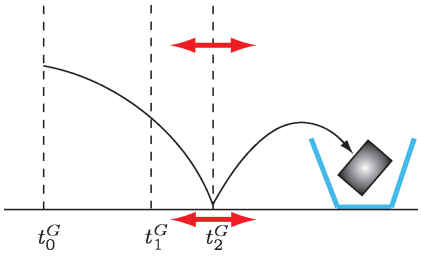

Fig. 8. An aligned time grid places a time point at each collision to partition the entire motion into collision-free shooting segments. The motion during each interval is continuous with respect to its initial state. Because collision times are not fixed, the points on the aligned time grid must slide with each collision.

a single collision time marks the end of one segment and the beginning of another. After alignment, the simulation function within each shooting segment is a continuous function of segment's initial state, which allows the continuous optimization from Section 4 to estimate the optimal parameters needed to fit the sketched motion.

The alignment also simplifies parameter initialization: each collision corresponds to some initial state, which identifies the collision point, the colliding bodies, and their positions and orientations. If an animator wishes to animate a pen bouncing on the table and landing in a mug, she or he may specify desired number of bounces, the preferred location of each bounce, or the pencil's contact point.

Rather than specify exact details of all collisions, an animator may prefer to have the optimizer infer contact points on each body automatically. An estimation procedure performs this task:

$$\min_{\mathbf{u},\mathbf{q}_0,\ldots,\mathbf{q}_{m-1}} \sum_{i=1}^{n} \left\| \mathbf{p}_i\left(\mathcal{S}_{t_i}(\mathbf{u},\mathbf{q}_0,\ldots,\mathbf{q}_{m-1})\right) - \mathbf{s}_i \right\|^2$$

$$\text{subject to} \quad \begin{cases} \underline{\mathbf{b}} \le \mathbf{c}_s(\mathbf{u},\mathbf{q}_0,\ldots,\mathbf{q}_{m-1}) \le \overline{\mathbf{b}}, \\ \mathbf{c}_r(\mathbf{u},\mathbf{q}_0,\ldots,\mathbf{q}_{m-1}) = \mathbf{0}, \end{cases}$$

with relaxed continuity constraints $\mathbf{c}_r$ that enforce the continuity of generalized coordinates but not the continuity of generalized velocities. This optimization uses the aligned time grid, but otherwise disregards collisions to allow interpenetration between colliding bodies. The result is a continuous but nonphysical motion with the appropriate collision sequence. Within each collision-free segment the motion obeys the equations of motion and complies with sketch constraints, but at each collision it contains physically inaccurate velocities. These physical inaccuracies are removed with the second estimation step described in the next subsection. What is more important at this stage are the initial positions for each shooting segment, which describe the contact points for each collision.

Another search procedure can identify the collision sequence that produces the optimal behavior. Given a specified number of desired collisions, an exhaustive discrete search could enumerate all possible collision sequences. The continuous optimization would then compute the optimal parameters for each collision sequence and select the one that yielded the best fit. We have implemented this exhaustive method and found it to be effective for simple polyhedral objects. Because the number of collision sequences grows exponentially with each collision, however, this simple approach does not scale for many practical problems.

5.2 Sliding Time Grid

The standard multiple-shooting formulation described in Section 4.1 uses a fixed time grid. After alignment, this standard formulation would require predetermined collision times. Although a similar

restriction also inhibits published spacetime techniques [Witkin and Kass 1988; Cohen 1992; Liu et al. 1994; Gleicher 1998; Popović and Witkin 1999b], we chose to address this problem by adapting the time grid to sliding collision times. In this formulation an interior grid point that corresponds to a collision time depends on the initial state of the preceding segment. For the example shown in Figure 4, the sliding time grid satisfies the following inequalities:

$$t_0 = t_0^G < t_1^G < t_2^G(\mathbf{q}_1) < t_3^G = t_f. \tag{7}$$

An optimization that solves the general estimation problem, defined in Equation (6), with the sliding time grid eliminates interpenetration, adjusts collision times, and corrects velocities at each collision.

The sliding time grid introduces two technical issues not encountered with a fixed time grid. First, as the optimization adjusts interior grid times, some sketch samples $\mathbf{s}_i$ will shift between shooting segments. These discrete events introduce small discontinuities into the least-squares objective function and sketch constraints $\mathbf{c}_s$. In practice, these discontinuities are removed by ignoring the subset of samples near each collision. Because the motion sketches are densely sampled, ignoring a few samples near boundaries has little measurable effect. Second, the sliding time grid may violate the total ordering in Equation (7). This violation is a strong indication that, for the given collision sequence, a physically correct simulation cannot fit the sketch. For example, a physically realistic motion cannot match physically infeasible trajectories. In these cases, the animator can relax the laws of physics or propose a different collision sequence.

6. SKETCHING INTERFACES AND EXAMPLES

A sketch interface combines the user interface, which abstracts the details of the parameter solver, and the initialization mechanism, which transforms the sketch into initial parameter values. The goal of the interface is to enable intuitive sketching process. Parameter initialization is the key component of the interface because parameter estimation is a difficult nonlinear and nonconvex optimization problem with many locally optimal solutions. This section describes two of many possible interfaces and presents several motion-sketching examples.

6.1 Editing Interface

The editing interface allows the animator to sketch the motion by dragging a rigid body, at any point in time, to a desired location. The editing interface is similar to the interactive single-shooting method for editing rigid-body simulations [Popović et al. 2000]. In contrast to that work, the editing interface divides the motion into collision-free segments. The animator can add or remove segments to change the intended number of collisions. A motion sketch is assembled by editing each collision-free segment independently. As described in Sections 4 and 5, multiple segments improve the stability of the estimation because the simulation need only integrate over short time intervals and need not compute the gradients of motion through collisions.

6.1.1 *Initialization.* Sketches designed with the editing interface consist of several collision-free segments. The motion within each collision-free segment is parameterized by the initial state. As the animator constructs the sketch, the interface adjusts the initial states appropriately. In effect, the interface provides the animator with direct intuitive control of the initial approximation. When the animator adds a segment, the interface creates the appropriate initial state; when the segment is removed, the interface deletes the initial state. Once the sketch is complete, the resulting initial states describe the initial approximation.

6.1.2 *Estimation.* After the editing, the sketched trajectories contain gaps between all collision-free segments. For example, edits in the second collision-free segment adjust the initial state at the

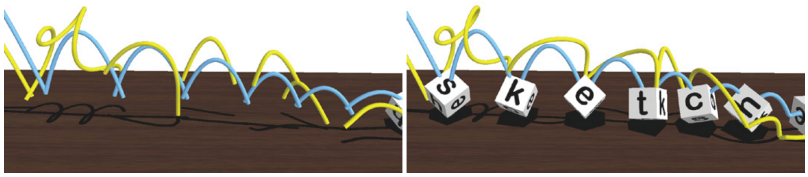

Fig. 9. On the left, the sketched trajectories of the center of mass (blue curve) and the die corner (yellow curve) are discontinuous. The figure illustrates the gaps in position and orientation at each collision. Gaps in velocities also exist but are not illustrated in the figure. On the right, parameter estimation removes the gaps in position and orientation to create continuous motion trajectories. Velocity gaps are also reduced but cannot be eliminated completely because the task is overconstrained.

beginning of that segment but do not affect the motion trajectories in the preceding and succeeding segments. The parameter estimation procedure removes these gaps by enforcing physically correct position, orientation, and velocity.

6.1.3 *Example.* The die example is an overconstrained motion-sketching task. The die must spell a word, bounce across the table in a straight line, and have the correct orientation at each impact so that the appropriate letter is visible. The bouncing constraints are enforced with hard sketch constraints: six-dimensional position and orientation constraints for each of the six bounces constitute 36 optimization constraints. The animator sketches each free-flight segment independently. The resulting sketch describes the desired trajectories and encodes the sequence of spelled letters. Figure 9 illustrates the gaps in position and orientation. Gaps in velocities also exist but are not illustrated in the figure. Parameter estimation computes the 24 simulation parameters consisting of the initial position (six parameters), initial velocity (six parameters), and the surface normals (two parameters) at each impact. The parameters are optimized to improve physical accuracy by reducing the gaps in the sketch. The constant bound optimization constraints are also introduced to ensure that the adjusted surface normals are within 6° from the actual surface normals. As a result, our system computes a physically plausible motion that accomplishes the animation tasks. At each collision, the gaps in position and orientation are imperceptible. The gaps in linear and angular velocity are larger but still yield a physically plausible animation. Because the task is overconstrained (36 constraints and 24 parameters), the velocity gaps cannot be removed completely. The entire parameter estimation required 182.45 s of optimization time on a single MIPS R10000 (195-Mhz) processor.

6.2 Acting Interface

The acting interface is a simple and intuitive interface particularly suited for inexperienced animators. The animator sketches the animation by acting out the motion with instrumented real-world objects. The sensor data encodes motion trajectories for each object. The trajectories are imprecise and have arbitrary timing (they may be in slow motion or faster than real time) but are sufficient to convey the essence of the motion. Parameter estimation transforms the acted sketch into a physical motion with physically correct timing.

6.2.1 *Initialization.* The acting interface converts sketched trajectories into initial parameter values. This initial guess defines the initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$ at the beginning of each collision-free segment. The interface recovers the initial states from the sketch by solving a sequence of simplified parameter-estimation problems. The optimizations transform each collision-free segment independently. Because the sketch trajectories are described with n sketch samples, the animator can identify the collision-free segments by identifying the indices $l_1, \ldots, l_{m-1}$ of a sketch sample near each collision. Because collisions denote the end and the beginning of a collision-free segment, the interface represents

m segments with $m+1$ indices:

$$1 = l_0 < l_1 < \cdots < l_{m-1} < l_m = n.$$

The interior indices $l_1, \ldots, l_{m-1}$ mark sketch samples for all sketch collisions. For example, samples $\{\mathbf{s}_{l_1}, \ldots, \mathbf{s}_{l_2} - 1\}$ describe sketch trajectories of the second collision-free segment, between the first and second collisions.

A simplified estimation procedure computes the timing of sketch samples and the center-of-mass trajectories simultaneously. The center-of-mass trajectories for each body are computed as functions $\mathbf{p}^{\mathbf{q}}_{\text{com}}(t, \mathbf{q}_i)$ of the appropriate initial state. The interface optimizes the least-squares fit to the sketched center of mass within each collision-free segment:

$$\begin{aligned} \min_{\mathbf{q}_i,\, t_{l_i}, \ldots, t_{l_{i+1}-1}} \quad & \sum_{j=l_i}^{l_{i+1}-1} \left\| \mathbf{p}^{\mathbf{q}}_{\text{com}}(t_j, \mathbf{q}_i) - \mathbf{p}^{\mathbf{s}}_{\text{com}}(\mathbf{s}_j) \right\|^2 \\ \text{subject to} \quad & 0 \le t_j < t_{j+1}, \quad \forall j \in [l_i, l_{i+1} - 1], \end{aligned}$$

where $\mathbf{p}^{\mathbf{s}}_{\text{com}}(\cdot)$ is the projection operator that extracts the center-of-mass coordinates from a sketch sample. The optimization constraint ensures that the total ordering of sketch samples is preserved: the succeeding sample must be at a later time than its predecessor.[4] Note that the function $\mathbf{p}^{\mathbf{q}}_{\text{com}}(t, \mathbf{q}_i)$ is considerably simpler than the simulation function $\mathcal{S}(t, \mathbf{q}_i)$. If the gravitational force is aligned with a second coordinate axis, the center-of-mass trajectory of a single 3D rigid body in free flight is described by a quadratic function and two linear functions:

$$\mathbf{p}^{\mathbf{q}}_{\text{com}}(t, \mathbf{q}_i) \stackrel{\text{def}}{=} \mathbf{x}_{\text{com}}(\mathbf{q}_i) + \mathbf{v}_{\text{com}}(\mathbf{q}_i)t + \begin{pmatrix} 0 \\ -\frac{1}{2}gt^2 \\ 0 \end{pmatrix},$$

where the projection functions $\mathbf{x}_{\text{com}}(\cdot)$ and $\mathbf{v}_{\text{com}}(\cdot)$ extract the position and the linear velocity of the center of mass.

After m optimizations, one for each free-flight segment, the interface computes m initial states $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$. Although these initial states define an initial approximation, the approximation can be improved by matching the orientation of samples in the sketch. The optimization described in Section 5.2 performs this task by solving a relaxed parameter-estimation problem:

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}} \quad & \sum_{i=1}^{n} \left\| \mathbf{p}_i\left(\mathcal{S}_{t_i}(\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1})\right) - \mathbf{s}_i \right\|^2 \\ \text{subject to} \quad & \underline{\mathbf{b}} \le \mathbf{c}_s(\mathbf{u}, \mathbf{q}_0, \ldots, \mathbf{q}_{m-1}) \le \overline{\mathbf{b}}. \end{aligned}$$

6.2.2 *Estimation.* Figure 1 shows the animator sketching a pencil bouncing, twirling in the air, and landing in a mug. The sketched trajectories describe the position $\mathbf{x}_s(t)$ and orientation $\mathbf{r}_s(t)$ of the pencil. Our system discretizes the continuous sketch trajectories with n sketch samples $\mathbf{s}_i \in \mathbf{R}^3 \times \text{SO}(3)$:

$$\mathbf{s}_i \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{x}_s(t_i) \\ \mathbf{r}_s(t_i) \end{pmatrix}, \quad i \in [1, n].$$

Although, in general, the sketch samples do not contain the timing information t_i, this section assumes that the timing information is provided. Section 4.3 describes the general setting, in which the timing

[4]Our implementation reformulates the problem slightly to solve for intersample durations d_i instead of explicit times t_j. With this reformulation $t_j = \sum_i^j d_i$, and the linear constraints $0 \le t_j < t_{j+1}$ are simplified with constant bound constraints $0 < d_i$.

information is not associated with each sketch sample. For each sketch sample, the projection functions $\mathbf{p}_i(\mathbf{q}(t_i))$ extract the position and orientation of the pencil from the generalized state:

$$\mathbf{p}_i \begin{pmatrix} \mathbf{x}(t_i) \\ \mathbf{r}(t_i) \\ \mathbf{v}(t_i) \\ \boldsymbol{\omega}(t_i) \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{x}(t_i) \\ \mathbf{r}(t_i) \end{pmatrix}, \quad i \in [1, n].$$

The sketched trajectories describe the preferred path for the pencil. The animator can also introduce a hard motion constraint to ensure that the pencil lands in the mug. In this case, the last sketch sample $\mathbf{s}_n$ corresponds to pencil's location inside the mug. The hard sketch constraint $\mathbf{c}_s$ would equate the final position and orientation of the pencil $\mathbf{p}_n(\mathbf{q}(t_n))$ with the last sketch sample:

$$\mathbf{c}_s = \mathbf{p}_n(\mathbf{q}(t_n)) - \mathbf{s}_n = 0.$$

The time grid for multiple shooting can partition the simulation into many subintervals. Here, we'll segment the simulation into two intervals and choose the time of collision as the splitting point t_1^G. As discussed in Section 4.1, the motion is computed within the two shooting segments independently:

$$\mathbf{q}(t) \stackrel{\text{def}}{=} \begin{cases} \mathcal{S}(t, \mathbf{u}, \mathbf{q}_0) & \text{if } t \in [t_0^G, t_1^G], \\ \mathcal{S}(t, \mathbf{u}, \mathbf{q}_1) & \text{if } t \in [t_1^G, t_2^G]. \end{cases}$$

The initial states $\mathbf{q}_0$ and $\mathbf{q}_1$ denote the generalized state of the pencil at the beginning of each interval. Parameter estimation computes the values of the initial states and the simulation parameters $\mathbf{u}$ that minimize the least-squares metric subject to satisfying the sketch constraint $\mathbf{c}_s$.

The continuity constraint $\mathbf{c}_c$ ensures that the initial state after the collision $\mathbf{q}_1$ matches the ending of the shooting segment prior to the collision:

$$\mathbf{c}_c(\mathbf{u}, \mathbf{q}_0, \mathbf{q}_1) \stackrel{\text{def}}{=} \mathbf{q}_1 - \mathcal{S}_{t_1^G}(\mathbf{u}, \mathbf{q}_0) = 0.$$

The continuity constraint enforces the continuity of the generalized state, which includes the positions and velocities of the mechanical system. Because the initial state $\mathbf{q}_1$ occurs an instant after the bounce, the simulation function $\mathcal{S}_{t_1^G}(\mathbf{u}, \mathbf{q}_0)$ evaluates the pre-collision free-flight motion and applies the collision impulse to compute the appropriate postcollision state.

Our system matches the sketch with a constrained optimization that solves the following estimation problem:

$$\min_{\mathbf{u}, \mathbf{q}_0, \mathbf{q}_1} \quad \sum_{i=1}^{n} \left\| \mathbf{p}_i\left(\mathcal{S}_{t_i}(\mathbf{u}, \mathbf{q}_0, \mathbf{q}_1)\right) - \mathbf{s}_i \right\|^2 \tag{8}$$

$$\text{subject to} \begin{cases} \mathbf{0} \le \mathbf{p}_n\left(\mathcal{S}_{t_n}(\mathbf{u}, \mathbf{q}_1)\right) - \mathbf{s}_n \le \mathbf{0} \\ \mathbf{q}_1 - \mathcal{S}_{t_1^G}(\mathbf{u}, \mathbf{q}_0) = \mathbf{0}. \end{cases}$$

The optimization computes the simulation parameters $\mathbf{u}$ and the two 12-dimensional state vectors $\mathbf{q}_0, \mathbf{q}_1$.

6.2.3 *Examples.* In the first motion-sketching example, the animator performs an animation of a pencil bouncing off a table and landing in a mug. This performance sketches the desired motion, by roughly describing desired trajectories. The animator also defines the collision-free segments and the hard sketch constraints that enforce the landing in the mug. The two collision-free segments are defined with an annotated sketch sample near the pencil's collision with the table and the landing is enforced

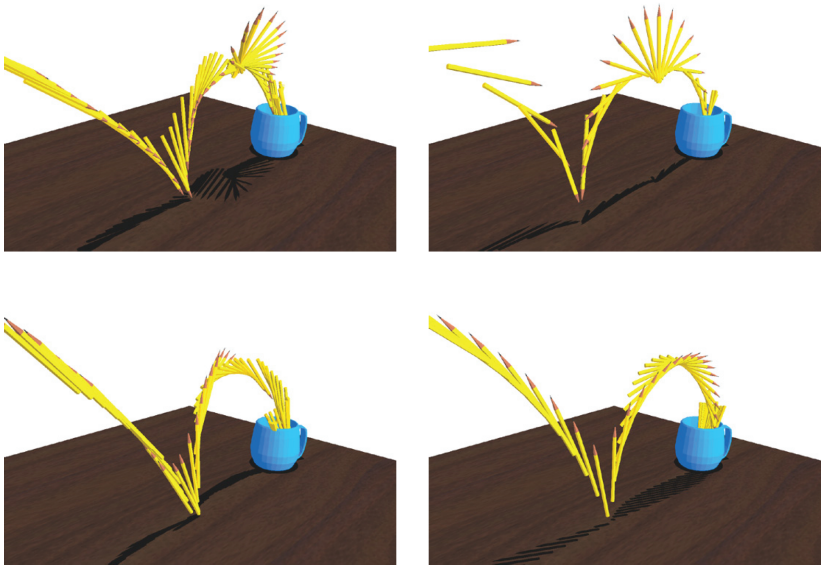

Fig. 10. Our system converts a sketch (left) into a physical motion (right). The two animations capture the variations in performed sketches.

with a six-dimensional hard constraint as described in the previous subsection. The interface constructs the initial approximation and the estimation optimizes 12 simulation parameters (initial position and velocity) to produce a physical motion that matches the sketch. The collision normal and elasticity coefficients were obtained from the physical simulation and not adjusted by the optimization. Figure 10 shows two physical animations with style variations. In the first animation, the pencil tumbles in the air after colliding, tip-first, with the table. In the second, the pencil bounces with the eraser side and lands in the mug after only a half-tumble. The average parameter estimation required 102.12 s of optimization time on a single MIPS R10000 (195-Mhz) processor. In our animations, the pencil skitters in the mug before coming to rest. Except for the constraint that the pencil lands in the mug, the skittering is not defined by the sketch. The motion is the result of extending the physical simulation.

The second motion-sketching task involves a sketch with two objects, a hat and a box, that collide in midair and fall on the table with the box inside the hat. Instead of moving real-world objects, the animator uses a head-mounted display to sketch the motion in a virtual world. The motion itself is specified by gesturing with hand-held motion sensors. The animator also specifies hard sketch constraints: inequality position constraints that enforce the midair collision and equality position constraints that force the box to land within the hat. These constraints result in a total of 15 optimization constraints. The parameter estimation computes 28 simulation parameters: the initial states of both objects (24 parameters), the surface normal at the impact (two parameters), and the elasticity of both objects (two parameters). The constant bound optimization constraints are introduced to ensure that the adjusted surface normals are within 6° from the actual surface normals and that the adjusted elasticities are within 10% of the chosen elasticity of 0.1 for the cube and the hat. The optimization required 300.35 s of time on a single MIPS R10000 (195-Mhz) processor. Figure 11 shows the resulting motion. In this example, the motion is physically plausible but not physically correct, due to the apparent infeasibility of the two constraints—in order for the box to land in the hat, the collision must cancel the opposite linear momentum of the two bodies. This is not possible without adjusting the angular momentum of the two bodies, which would, in turn, violate the orientations specified in the sketch. Instead, the solver

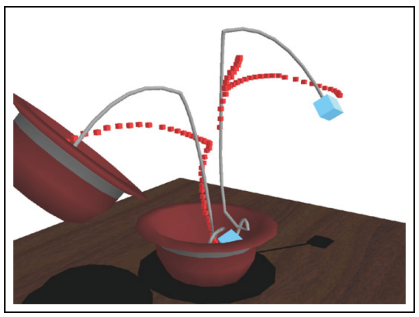

Fig. 11. An illustration of the physically plausible motion derived from a sketched motion. The computed center-of-mass trajectories for the hat and the box are shown in gray lines. The sketch samples are indicated with red dots.

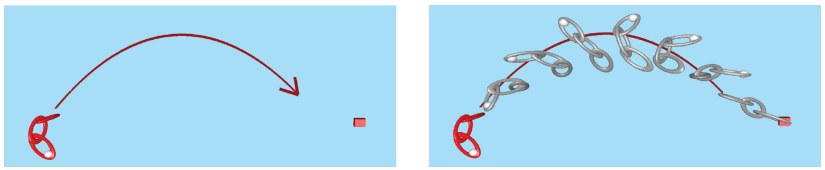

Fig. 12. A sketch of the chain motion is converted automatically into a physical motion of a chain flying across and reaching the red box with a white ball. The chain is modeled as a series of three rigid links connected by two universal joints.

minimizes physical violations at the impact to construct a plausible motion. Note that the midair collision in the solution has been automatically raised to a higher point than was specified in the sketch. Raising the collision point has the effect of decreasing the velocities of the two objects when they collide, making it easier to satisfy the constraints.

6.3 Constrained Rigid Bodies

In the last example, shown in Figure 12, our system constructs a motion of a three-link chain flying across the figure to reach a hook (the red box in the figure) on the right. The chain is modeled with three rigid bodies linked with two universal joints (u-joints). A u-joint has 2° of freedom to mimic connections between links in a real chain (the links in a chain accept little or no twist). With unit-quaternion parameterization of the six-dimensional position and orientation, the generalized state vector consists of 11 generalized coordinates and 10 generalized velocities.

The sketch consists of two hard constraints that describe the required initial and final position for the chain and soft constraints that indicate the preferred path. The first hard constraint specifies the initial configuration for the chain with all 11 generalized coordinates. The second hard constraint specifies the final 3D location for an end-point on the bottom-most link (the white ball in the figure). Similarly, the soft constraints specify the preferred trajectory for a center-point of the top-most link.

Our system estimates the 21 parameters of the generalized state needed for a simulation that satisfies the 14 hard optimization constraints and minimizes the least-squares fit to soft constraints. The optimization required 354.08 s on a single Pentium III (1.7-GHz) processor. In this example, because the first constraint specifies all generalized coordinates, an optimization could be specialized to estimate only the generalized velocities. Our system does not make this simplification and still converges robustly. In our experiments, the system also converges without this first hard constraint or with inequality constraints that restrict the initial configuration to an arbitrary region.

In this example, we used a multiple-shooting grid that splits the motion into five shooting segments. In general the number of shooting segments should increase with longer or more complex simulations. Unlike single shooting, which does not converge robustly as predicted by theoretical analysis, our experiments show that our multiple-shooting method converges for a range of mechanical systems such as clamped or free-falling multilink pendulums with pin, universal, or ball joints.

7. CONCLUSION

Motion sketching is a paradigm for creating physical animations automatically. As demonstrated, our motion-sketching technique is reliable and robust for a wide-range of animation tasks. The method relies on mathematical derivatives for fast convergence. For the same reason, it should not be applied to construct motions with large number of collisions because in those cases the optimized functions are largely discontinuous and mathematical derivatives carry little information. For these motions, sampling approaches appear to be the only solution, but will have tremendously slow convergence in high-dimensional optimization spaces. A hybrid discrete and continuous approach could combine a heuristic search, which would propose a collision sequence, and our estimation procedure, which would use the proposed sequence as a seed for finding the optimal motion. In contrast to single-shooting variants such as Monte Carlo method [Chenney and Forsyth 2000] and genetic algorithms [Tang et al. 1995], the hybrid approach would use the discrete search only where it is necessary, but would otherwise use a continuous optimization technique.

In our experience, the method is unsuccessful only for very infeasible or very imprecise sketches. Both cases lead to poor initial approximations for which the optimization cannot converge to an appropriate solution. Our die and hat examples show the effect of infeasible (overconstrained) sketches on the resulting motion. Although a quality of the initial sketch is difficult to formalize, in practice the motion-sketching approach converges even for sketches that would be thought of as poorly keyframed motions. A skilled animator could correct these problems by improving the sketch.

The acting and editing techniques are appealing interfaces for sketching motions. A camera-based interface could make the acting process more natural. For example, the animator might simply record her hand gestures with a video camera. A drawing-based interface could allow the animator to describe motions with paper and pencil.

ACKNOWLEDGMENTS

Sebastian Grassia contributed with his own robust code for parts of the user interface and for the computation of exponential map gradients. Dennis Cosgrove created the interface for sketching with motion sensors. The paper benefited from valuable comments of numerous anonymous reviewers. We are particularly grateful to one ACM TOG reviewer whose detailed comments significantly improved the organization of this paper.

REFERENCES

BARAFF, D. 1992. Dynamic simulation of non-penetrating rigid bodies. Ph.D. thesis, Cornell University, Ithaca, NY.

BARAFF, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. In *Computer Graphics (Proceedings of SIGGRAPH 94)*. Annual Conference Series. ACM SIGGRAPH, 23–34.

BARZEL, R. 1992. *Physically-Based Modeling for Computer Graphics: A Structured Approach*. Academic Press, San Diego, CA.

BARZEL, R. AND BARR, A. H. 1988. A modeling system based on dynamic constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*. Annual Conference Series. ACM SIGGRAPH, 179–188.

BARZEL, R., HUGHES, J. F., AND WOOD, D. N. 1996. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation '96*. Proceedings of the Eurographics Workshop. Poitiers, France, 184–197.

BESL, P. J. AND MCKAY, N. D. 1992. A method for registration of 3D shapes. *IEEE Trans. Patt. Anal. Machine Intell. 14*, 2, 239–256.

BISCHOF, C., CARLE, A., KHADEMI, P., AND MAUER, A. 1996. ADIFOR 2.0: Automatic differentiation of fortran 77 programs. *IEEE Computat. Sci. Eng. 3*, 3 (Fall), 18–32.

BOCK, H. G. 1983. Recent advances in parameter identification techniques for ordinary differential equations. In *Numerical Treatment of Inverse Problems in Differential and Integral Equations (Proceedings of an International Workshop, Heidelberg)*, P. Deuflhard and E. Hairer, Eds. Birkhäuser, Boston, MA, 95–121.

BROTMAN, L. S. AND NETRAVALI, A. N. 1988. Motion interpolation by optimal control. In *Computer Graphics (Proceedings of SIGGRAPH 88)*. Annual Conference Series. ACM SIGGRAPH, 309–315.

CHENNEY, S. AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH 2000*. Computer Graphics Proceedings, Annual Conference Series. ACM Press/ACM SIGGRAPH, 219–228.

COHEN, M. F. 1992. Interactive spacetime control for animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)*. Annual Conference Series. ACM SIGGRAPH, 293–302.

GILL, P. E., MURRAY, W., AND SAUNDERS, M. A. 1997. User's guide for SNOPT 5.3: A Fortran package for large-scale nonlinear programming. Tech. rep. NA 97–5, University of California, San Diego, San Diego, CA.

GILL, P. E., MURRAY, W., AND WRIGHT, M. H. 1989. *Practical Optimization*. Academic Press, London, U.K.

GLEICHER, M. 1997. Motion editing with spacetime constraints. In *1997 Symposium on Interactive 3D Graphics*. 139–148.

GLEICHER, M. 1998. Retargetting motion to new characters. In *Computer Graphics (Proceedings of SIGGRAPH 98)*. Annual Conference Series. ACM SIGGRAPH, 33–42.

GOH, C. J. AND TEO, K. L. 1988. Control parametrization: a unified approach to optimal control problems with general constraints. *Automatica 24*, 1, 3–18.

GRIEWANK, A. AND CORLISS, G. 1991. *Automatic Differentiation of Algorithms*. SIAM, Philadelphia, PA.

GRZESZCZUK, R. AND TERZOPOULOS, D. 1995. Automated learning of muscle-actuated locomotion through control abstraction. In *Computer Graphics (Proceedings of SIGGRAPH 95)*. Annual Conference Series. ACM SIGGRAPH, 63–70.

GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. 1998. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Computer Graphics (Proceedings of SIGGRAPH 98)*. Annual Conference Series. ACM SIGGRAPH, 9–20.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Computer Graphics (Proceedings of SIGGRAPH 95)*. Annual Conference Series. ACM SIGGRAPH, 71–78.

ISAACS, P. M. AND COHEN, M. F. 1987. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In *Computer Graphics (Proceedings of SIGGRAPH 87)*. Annual Conference Series. ACM SIGGRAPH, 215–224.

KASS, M. 1992. CONDOR: Constraint-based dataflow. In *Computer Graphics (Proceedings of SIGGRAPH 92)*. Annual Conference Series. ACM SIGGRAPH, 321–330.

LIU, Z., GORTLER, S. J., AND COHEN, M. F. 1994. Hierarchical spacetime control. In *Computer Graphics (Proceedings of SIGGRAPH 94)*. Annual Conference Series. ACM SIGGRAPH, 35–42.

MILLER, G. S. P. 1988. The motion dynamics of snakes and worms. In *Computer Graphics (Proceedings of SIGGRAPH 88)*. Annual Conference Series. ACM SIGGRAPH, 169–177.

MIRTICH, B. 1996. Impulse-based dynamic simulation of rigid body systems. Ph.D. thesis, University of California, Berkeley, Berkeley, CA.

MOORE, M. AND WILHELMS, J. 1988. Collision detection and response for computer animation. In *Computer Graphics (Proceedings of SIGGRAPH 88)*. Annual Conference Series. ACM SIGGRAPH, 289–298.

NGO, J. T. AND MARKS, J. 1993. Spacetime constraints revisited. In *Computer Graphics (Proceedings of SIGGRAPH 93)*. Annual Conference Series. ACM SIGGRAPH, 343–350.

PANDY, M. G., ANDERSON, F. C., AND HULL, D. G. 1992. A parameter optimization approach for the optimal control of large-scale musculoskeletal systems. *Trans. ASME. J. Biomech. Eng. 114*, 4, 450–460.

PLATT, J. C. AND BARR, A. H. 1988. Constraint method for flexible models. In *Computer Graphics (Proceedings of SIGGRAPH 88)*. Annual Conference Series. ACM SIGGRAPH, 279–288.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive manipulation of rigid body simulations. In *Computer Graphics (Proceedings of SIGGRAPH 2000)*. Annual Conference Series. ACM SIGGRAPH, 209–218.

POPOVIĆ, Z. AND WITKIN, A. 1999a. Physically based motion transformation. In *Computer Graphics (Proceedings of SIGGRAPH 99)*. Annual Conference Series. ACM SIGGRAPH, 11–20.

POPOVIĆ, Z. AND WITKIN, A. 1999b. Physically based motion transformation. In *Computer Graphics (Proceedings of SIGGRAPH 99)*. Annual Conference Series. ACM SIGGRAPH, 11–20.

RAIBERT, M. H. AND HODGINS, J. K. 1991. Animation of dynamic legged locomotion. In *Computer Graphics (Proceedings of SIGGRAPH 91)*. Annual Conference Series. ACM SIGGRAPH, 349–358.

ROBERTSON, B. 1998. Meet geri: The new face of animation. *Computer Graphics World* (www.cgw.com).

ROBERTSON, B. 1999. Antz-piration. *Computer Graphics World* (www.cgw.com).

ROBERTSON, B. 2001a. Medieval magic. *Computer Graphics World* (www.cgw.com).

ROBERTSON, B. 2001b. Monster mash. *Computer Graphics World* (www.cgw.com).

STENGEL, R. F. 1994. *Optimal Control and Estimation*. Dover Books on Advanced Mathematics, New York, NY.

SYMON, K. R. 1971. *Mechanics, 3rd ed*. Addison-Wesley, Reading, MA.

TANG, D., NGO, J. T., AND MARKS, J. 1995. N-body spacetime constraints. *J. Vis. Comput. Animation 6*, 143–154.

WITKIN, A. AND KASS, M. 1988. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*. Annual Conference Series. ACM SIGGRAPH, 159–168.

YAMANE, K. AND NAKAMURA, Y. 2000. Dynamics filter—concept and implementation of on-line motion generator for human figures. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 688–694.

Received April 2002; revised March 2003; accepted March 2003