Visual Comput (2005) 21: 488–501
DOI 10.1007/s00371-005-0297-z

ORIGINAL ARTICLE

Adi Bar-Lev
Alfred M. Bruckstein
Gershon Elber

Virtual marionettes: a system and paradigm for real-time 3D animation

Published online: 19 July 2005
© Springer-Verlag 2005

A. Bar-Lev · A.M. Bruckstein (✉) ·
G. Elber
Computer Science Department, Technion,
I.I.T. 32000 Haifa, Israel
freddy@cs.technion.ac.il

Abstract This paper describes a computer graphics system that enables users to define virtual marionette puppets, operate them using relatively simple hardware input devices, and display the scene from a given viewpoint on the computer screen. This computerized marionette theater has the potential to become a computer game for children, an interaction tool over the Internet, enabling the creation of simultaneously viewed and operated marionette show by users on the World Wide Web, and, most importantly, a versatile and efficient professional animation system.

Keywords Computer graphics system · Virtual marionettes · Computer game · Animation system · RealTime physics-based animation

1 Introduction

Marionettes have always had a special place at puppet shows and are favorite toys for children of all ages [2]. The purpose of the system described herein is to unleash the phenomenal capabilities now widely available on personal computers, to allow natural and easy access to an imaginary puppet world of infinite variety: an imaginary world that allows the user to operate rather complex articulated objects and make them play on stages from a computer graphics wonderland with the help of some simple devices for geometric data input and high-resolution displays that can readily be projected on large screens.

2 Scientific background of the project

So far, most efforts in animation have gone into systems based on forward kinematics, where the animator can specify directly the positions and movements of all joints, or on inverse kinematics, where a certain movement goal is specified, and the positions and movements of joints are calculated to meet the required goal. Both approaches have found adherents, but the inverse kinematics approach is considered better for the design of user-friendly, goal-directed systems enabling the user to specify general motion requirements. Watt and Watt provide a good survey of these techniques in their book [20].

The marionette manipulation system that we present here differs from these approaches. It is a physics-based simulation program, calculating the movements of articulated objects in response to "virtual" locally applied forces and displacements. Recently considerable research effort has been devoted to the field of physically based modeling and simulation for a variety of computer graphics applications. In this context scientists have developed programs for simulating the motions of articulated rigid and even nonrigid bodies subject to various forces and constraints. Examples of such programs are described in [5, 6, 9, 22, 24] and are eminently suitable for marionette dynamics simulations and form the core of the system we have implemented.

In order to efficiently integrate the equations provided by the elegant theory of analytical dynamics due to Lagrange (see [13] and [12]), the system implemented herein combines the physics of basic particle systems, which considers forces and geometric and time constraints, with an adaptive time step integration process. A good reference for this theory can be found in [21, 22]. The physical theory is also supplemented with several heuristics because sometimes giving up physical behavior yields the possibility of having responses that "appear" physically correct and enable the real-time running of a rather complex virtual marionette show.

An issue that needed special attention was the topic of collisions with the floor and between the puppets, or the various parts of the same puppet. Here, too, several clever algorithmic shortcuts were imperatively needed to ensure physically plausible behavior, along with real-time system responses.

3 Virtual marionettes: general system description and functionality

The goal of the work presented here was to design a computer-graphics-based system that allows users to define a marionette as a general articulated object or structure, together with points of attachment for virtual operating strings. The operating strings are then conceptually attached to a virtual hanging frame—a user-defined geometric object, with easily modifiable modular movement functionality. The functions of this marionette control system are subsequently mapped to a mechanical/geometric data input device that might be viewed as a much simplified "Data Glove." The input device displacements are then transmitted via a communication link to a computer, where they are translated to movements of the virtual frame associated to the puppet, thereby controlling its movements via the virtual strings attached to it.

Using the concept of virtual marionettes, an operator/artist has access to a system for designing complete puppet shows with as many and as varied participants as his/her imagination desires, on stages that are designed with the use of recent advances in rendering methods, complete with illumination and color effects.

Note that in using our system, we do not aim to precisely reproduce the walk of animal or humanlike figures or to generate perfectly realistic motions. Rather, we let these aims be met by the skills of the human operator, using interactive feedback from the screen. In this sense our system is simpler than inverse-kinematics-based animation software programs. We claim that such a system, in the hands of a skilled operator, can effectively be used as a first-stage animation tool in the process of producing animated movies.

The system we implemented comprises three parts:

1. The physical simulation and animation program – *the brain*.
2. The movement control input devices – *the interface*.
3. The rendering engine and the graphics display – *the stage*.

Let us now describe the three components in detail.

3.1 The brain: a physics-based animation software program

The software system we implemented allows for the definition of articulated objects, the marionettes, by giving the user the possibility to select, modify, and combine given 3D graphics models. The 3D models can be created using any existing 3D software, such as the 3D Studio Max, and can then be loaded to the application and used as parts of the marionette. In more advanced versions of our software, nonrigid objects would be representable as well. After an articulated object has been defined, the user may choose the "anchors" for the virtual manipulation strings and their attachments to the control frame that can be suitably designed. From this point on, a physics-based animation software produces movements of puppets in response to the users' controls, transmitted via the software to the virtual manipulation strings. This way the system achieves fast and realistic responses, enabling the user to have real-time control over his puppets. We note that the user's voice can be integrated in the play as well (and one might even enhance the program with voice-controlled mouth-movement subroutines for each puppet!).

3.2 The interface: a movement-detection device

The inputs for marionette movement control are displacements and forces applied to the virtual strings. Since these virtual strings are pulled and moved around, we need to detect control signals for such actions coming from the user. For example, the strings may be controlled by the fingers of the operator, while global motion can be controlled by hand, head, or body motions.

Currently, we use a simple data-glove-type input device, which was designed and created as a part of this project by a team of two Technion undergraduate students (Fig. 1).

The input device transduces and transmits the stretch of several strings attached to the user's fingers. Each data glove transmits data for the control of a single marionette. The device we implemented has 5° of freedom. We chose to use them for the rotation axes and for setting the vertical and the forward/backward movement axes.

A project currently under development focuses on the use of digital video cameras as alternative input devices for the control of virtual marionette frames. The aim is to enable the detection and real-time computation of the spatial pose of a real "puppet control frame." The devices' position and orientation will then drive the virtual control frames for the marionettes defined in the system.

3.3 The stage: a computer-generated world

The virtual world in which the simulated marionettes perform can be as sophisticated as allowed by the latest developments in computer graphics. The stage could be simple,

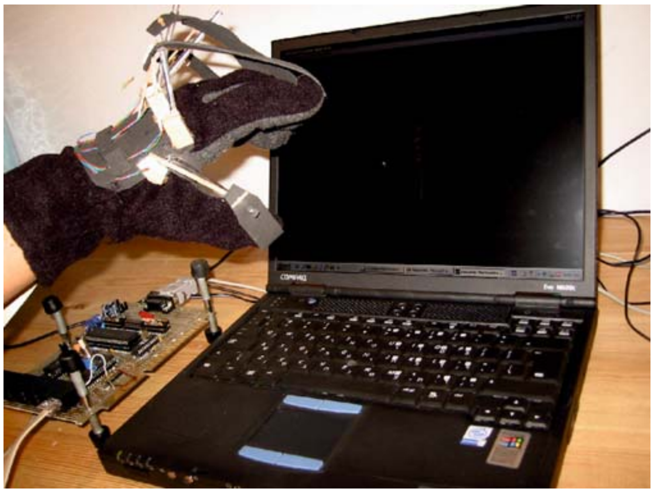


Fig. 1. A picture of one of the two data gloves that was designed and implemented as part of this project

with a few objects defined by the user, or it could become a fractal landscape modeling the surface of some distant planet. The stage, once defined, can also benefit from various lighting schemes, and a user can easily implement and control complex lighting and surface texture effects. Day can turn into night at the turn of a knob, and preprogrammed star constellations can appear on the simulated sky. Different events can be introduced into this imaginary stage, and changes in the scenery can be made almost as fast as computer memory can be accessed.

The combination between the simulation unit, the "brain," and the graphics engine that generates the "stage and the marionette-actors" ultimately determines the quality of the visual effects.

The display graphics engine itself supports three different modes:

(1) Particle display mode A 3D model is attached to each particle and the attached model is transformed according to the movement of the particle. This mode is used for simplified figures with symmetric models such as spheres.

(2) Link display mode Two particles can be used to display a 3D model according to their configuration. The model is automatically stretched, scaled, and positioned to fit the distance between the particles. The model orientation is set by the direction between the particles.

(3) Rigid group mode A rigid object model is linked to three given particles. The model is oriented either by the particles' configuration or by their initial orientation. During the simulation course, the model's orientation is determined for each frame by the configuration of the three particles.

(4) Freeform mode is in process of being implemented and will be used to represent deformable models.

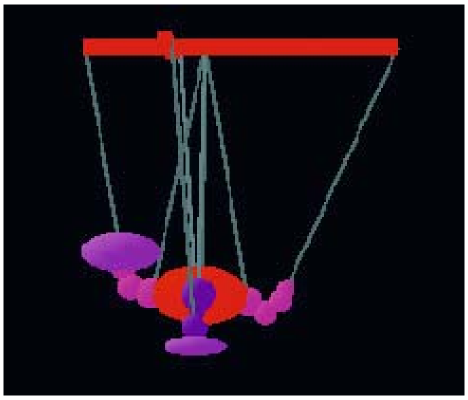


Fig. 2. A "dragon" marionette built with 3D models attached to particles (*particle* display mode)

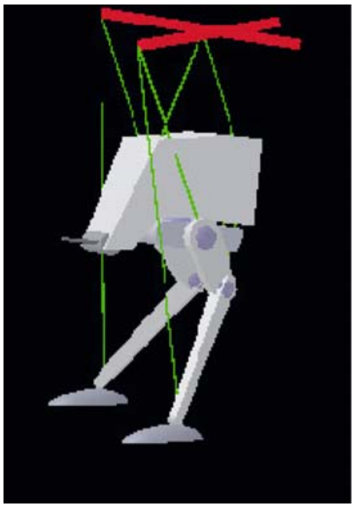


Fig. 3. The "IceTrooper" from the Star Wars movie. In order to display its legs, the *link* display mode was used, while for its head and body we used the *rigid group* display mode. The leg joints were displayed using the *particle* display mode

Note that 3D models can be loaded and set at any position, orientation, and scale during the initialization stage of the system. The model's material and texture are then applied and can readily be altered at any time during the simulation. Figures 2 and 3 demonstrate some of the modes described for puppet display.

4 System internals

4.1 Overview

We present below the working elements of the dynamic simulation system, the "brain," and the interactions among them. It is not our intention to get into the mathematical or physical aspects of the system yet, but rather to give the reader a global picture of our system and its structure. The system comprises the following building blocks:

Physical unit: This part of our system is based on the physical equation solver, following Witkin et al. [22]. The solver is given a physical configuration (configuration of masses, forces, velocities, locations, constraints, etc.) and computes the resulting overall forces that act in the scene during the subsequent time interval.

Mathematical equation solver: Based on the set of linear equations generated by the physical unit, the mathematical engine solves the equations by calling to action one of the many existing numerical methods that were implemented.

Collision detection unit: This unit detects and deals with the collisions between the marionettes and the other objects in a scene (as well as other marionettes).

Adaptive time step unit (ATS): Works together with the physical engine and decides the length of the next time interval based on previous sampling intervals and detected system deviations of the particles from their geometric constraints.

Frame display scheduler: Controls the rate (in frames per second) for the display device and synchronizes between the varying sizes of the time step intervals and the number of intervals that need to be calculated for each displayed frame.

In the following subsections, we briefly describe each of these elements and their combined actions and interactions. The complete structure of the system is presented in Fig. 4.

4.2 Physical unit

There are many approaches to simulating the dynamics of objects. Important research in this area includes the work by Barzel and Barr [5] on the behavior of rigid objects under constraints, the work of Cohen et al. [7, 8], which mostly handles simulation control and physical constraints, and the work by Witkin et al. [22, 24] focusing on spacetime constraints and simulation control. Other approaches to dynamic simulations exist, and several of them deal with emulating physical behavior without really solving all the correct physical equations involved. This is mostly done by finding behavioral functions that act "like the real thing," hence aim toward a visually good motion. One example along this line of thought is the work by Overveld [15].

In this work, we chose a different approach for handling the physical problem. Since our main aim is to manipulate marionettelike virtual devices, we want to have a system that is specially designed to deal with articulated objects with simple spherical joints (2/3 DOFs). Furthermore, we want to provide the user with an intuitive way of assembling and controlling these virtual marionettes. A method that nicely supports these specifications was first introduced by Witkin et al. [22] and is based on the classical physics of particle systems. The beauty of the method lies in the fact that the user does not have to specify an inertia matrices for the objects he works with, nor does he have to adjust the shape and mass distribution of any object to get a desired behavior. Instead, the user creates objects by simply specifying a group of points in space, each with a mass and location (***particle***). Objects can thus have physical behaviors that are simple to modify by moving particles or changing their mass at will. A significant advantage of this approach is the fact that we can separate between the objects' 3D models and the way they behave; thus we can get almost any desired behavior with an interface that is easy to understand and use.

Physical dynamic systems are based on the principle of continuous motion through time. This cannot be achieved when simulating such systems since the simulator works with discrete time steps instead of infinitesimally small steps as required for continuous integration in time. To this end, we use the *Runge–Kutta* extrapolation scheme for better prediction and reduction of the integration error. At each time interval the physical unit applies all known forces to the particles and calculates data needed from the constraints. At the end of the process, the physical unit forms a system of linear equations in which the unknowns are *Lagrange multipliers*. Once the equations are solved, the multipliers are assigned to the constraint equations, and by using a priori assumptions to be discussed later, we can obtain the constraint forces. Adding these forces to the particles, we obtain the new overall forces that act on each particle, and based on prior velocities and locations we extrapolate the system's state through the time interval.

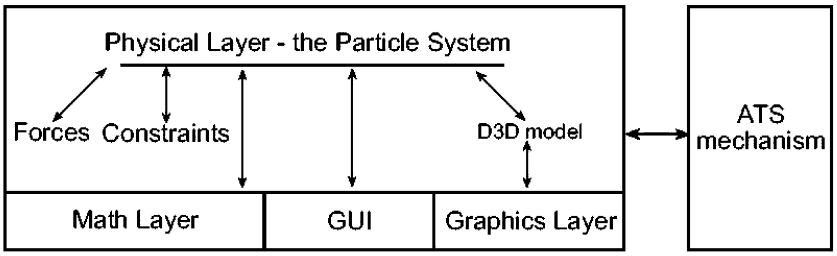


Fig. 4. The relation between the different parts of the articulated object simulation system. The physical layer is the hard core of the system, while the other mechanisms are attached to and through it. The frame display scheduler is omitted from the scheme but is basically attached to both the physical layer and the ATS mechanism

The final result of these calculations is a close approximation to the new locations of all the particles as well as their velocities and accelerations. Using the location of the particles, our system now calculates and displays the configuration of each of the 3D models in the scene.

4.3 Mathematics unit

The mathematics unit yields solutions to the linear equations generated by the physical unit. As we shall see, the solutions to these equations are constraint forces that were implicitly defined and hence previously unknown to the system. We tested several techniques for solving the system of equations and concluded that a *Gausss–Seidel* direct solver is suitable since our system has no more than several hundred sparse equations. However, as the systems become more complicated, and in light of the fact that system matrices tend to be sparse for most types of constraints, an iterative solver for sparse matrices can often be used more effectively. Furthermore, when using iterative methods, we can stop the solution process when reaching a given error threshold, thereby gaining time. Another advantage of an iterative solver stems from the fact that we can get good approximation with these methods even if the system has dependencies through the existing geometric constraints. This feature provides us with extra time to identify such ill-defined configurations and avoid, or even stop, the course of the simulation.

4.4 Collision detection unit

To enable as realistic a behavior as possible in our simulations, we enabled the system to deal with forces and momenta interactions. The simulated behavior does not seem realistic without reactions to collisions and contacts; hence we implemented a very basic collision and contact detection scheme between puppets and other objects in a scene.

We based our collision detection on two basic methods. Collision with planes such as the floor were detected by testing directly each of the particles representing a marionette, and collisions with more complex objects (such as the puppets themselves) were detected through the use of a collision sphere-tree. Although many other techniques exist and could be used here, we found these two techniques sufficient for demonstrating our concepts and obtaining highly realistic behavior. The sphere-tree collision technique and its modification were mainly inspired by Hubbard [16, 17] and is very effective in our case where the objects that comprise the puppet rapidly change their location and orientation. Some examples of the sphere-trees created for the marionettes are shown in Figs. 5 and 6.

4.5 Adaptive time step mechanism

A particle system has configurations determined by explicit behavioral forces (such as springs and dampers) and by constraints imposed on the system. These configurations correspond to different forces and momenta for the particles, and when simulating the movement during time intervals the particle system undergoes deviations due to varying constraint forces and magnitudes. As the time step increases, the extrapolation of these values becomes less stable; hence the entire system might become mathematically unstable, depending on the configuration. To overcome these undesirable phenomena, we can either use an adaptive time step (ATS) mechanism or we might always use sufficiently small time steps so that all integration errors will remain within a tolerable range. In most cases,

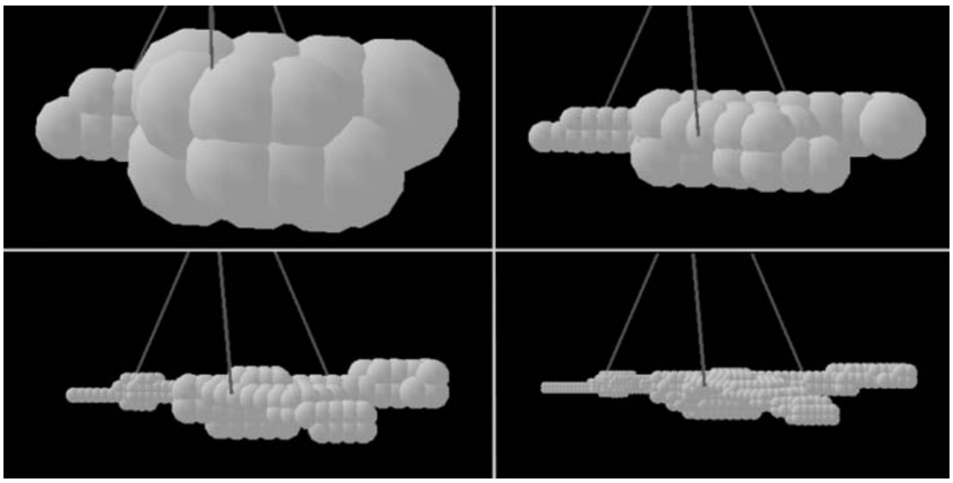


Fig. 5. The different levels of the collision sphere-tree, which represents a birdlike marionette

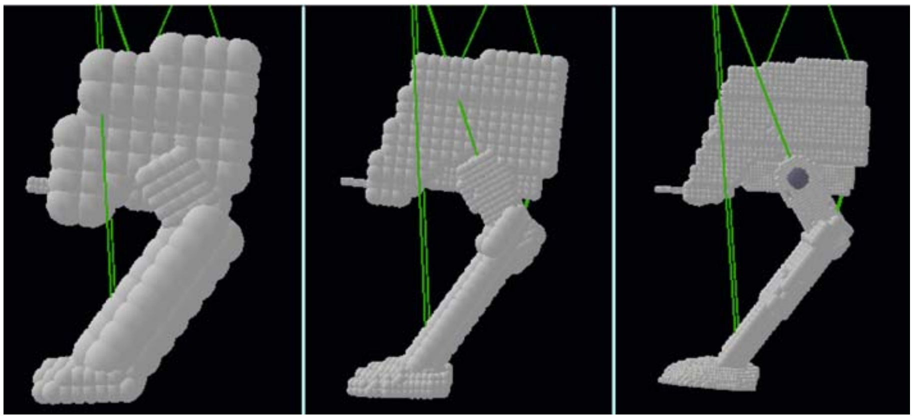


Fig. 6. The collision sphere-tree of the IceTrooper marionette

using the latter approach will impede running the simulation in real time.

In our system implementation, we introduced a relatively simple ATS mechanism. By monitoring all particle deviations at all times, we selected the current largest deviation from the valid configuration of the system. We then compared it to a predefined error threshold and decided whether the next time step should be increased or decreased and by what ratio. Default lower and upper time steps bounds, as well as predefined error thresholds, were implemented as well. These values can be set specifically for each system using a loaded configuration file.

Since the Runge–Kutta extrapolation scheme we implemented has a precision of order $O(h^4)$, the estimated time step is calculated as follows:

$$\Delta t_{new} = \Delta t_{old} \cdot time\_ratio$$

$$time\_ratio = \left(\frac{threshold\_error}{current\_error} . \right)^{\frac{1}{4}}$$

We note that the Runge–Kutta method engenders very small errors to begin with. Hence, starting with a reasonable time step (of about $\frac{1}{50}$ of a second), should be sufficient, and in practice the time step is not expected to vary by much. This is due to the fact that all ratios between the current error and the error thresholds are damped by the fourth root update formula. In addition, the system itself has a dynamic error correction mechanism; hence as long as the deviations from the valid states are kept within a reasonable range, they will tend to be reduced over time.

The ATS is also responsible for the time step reduction right after a collision occurs. In such a case, the time of the collision is calculated, the configuration of the system before the collision is restored, and a time step that matches the exact collision time is set. This must be done to prevent large deviations due to errors resulting from the inaccuracy in the collision location choice. After collision events are detected and calculated, the time step is restored by increasing (by a factor of two) each time interval of subsequent iterations until the allowable error threshold level is reached again.

4.6 Frame display scheduler

The ATS mechanism allows the simulations to run in real time while maintaining a correct and controlled speed of simulation and a constant number of frames per second (fps) for displaying the results. This mechanism is the trigger to both the calculation of each step and the display of the scene in real time. Using a simple configuration file, the user can set the system to output to the display at any desired frame rate within the capabilities of the CPU. During the execution of the simulation, the application's timer is activated every predefined interval of time (1/fps). At each such call, several iterations are carried out. The number of iterations varies depending on the deviation of the particle system from its physically valid configuration as well as on the amount of frames per second defined by the user (the deviation error is responsible for setting the length of the time interval for the system's integration step). At the end of each timer call, the marionette is actually rendered to the screen. In fact, the last frame is drawn at the start of each timer call, and then we carry out the next frame's iterations. This way, all frames are displayed without any delay due to the mathematical processing. However, the frames are shown with a delay of a single frame.

Since the application is adaptive with respect to both the time step and the deviations, if the complexity of the particle system exceeds the maximum that can be handled in real time, the frame display scheduler has no choice but to skip rendering frames until the previous frame completes its "dynamics" calculations. This is done in order to maintain the solution error within a reasonable range. When the dynamics iterations time interval reaches or passes the time limit allocated by the frame display, the

ATS mechanism reduces the last time step so that the display at the end of the frame will be accurate. This action also helps in reducing the amount of accumulated error through iterations.

5 Physical unit

5.1 Terminology and conventions

In this paragraph we first give a brief description of the terms used to describe the dynamic equations throughout this article. The physics and mathematics foundations and equations are only briefly reviewed here, and we assume that the reader is familiar with the field.

Particle An infinitesimally small object (a point in the 3D world) with the qualities of mass, location, and its derivatives (velocity and acceleration).

Constraint In this work we deal with two types of constraints: **geometric constraints**, which cause activation of forces in the system due to the need to maintain a predefined geometric configuration, and **time constraints**, which force objects to move along a predefined curve in time.

Force ($F = ma = m\ddot{x}$): Our system deals with two types of forces. The known forces, usually referred to as *applied forces*, are denoted by F_a, and the unknown forces, usually generated by the activation of constraints. These forces are denoted by F_c.

Momentum ($L = mv = m\dot{x}$): Since our approach uses particles, we refer only to linear momenta. Momenta in the system are considered mainly at collisions. We deal with complex systems of particles with geometric constraints; hence changing the velocity of single particles as a result of local operations cannot be applied because the system will deviate from its valid configuration. Therefore, momenta must be derived from the collision constraints in real time and be applied through an additional set of equations.

Rigid group A group of particles attached to each other by strict *static distance* constraints. The smallest group of that type that has a unique inertia matrix is a group of four particles in general positions, combined in a triangular pyramid-shaped clique (tetrahedron). In our system, we restrict the connection of each new particle to no more than three other nonplanar particles in order to prevent dependency in the equations.

An object in our system is thus comprised of particles, known forces due to *gravity, springs, dampers, drag*, etc., and constraints, such as *predetermined fixed distances* between particles. Using fixed-distance constraints we can construct *rigid groups* and display each such group using a chosen 3D model. We also have the freedom to form nonrigid (articulated and deformable) objects by connecting particles using the combination of spring/damper forces instead of connecting them by rigid geometric constraints.

5.2 Implicit constraint solver for forces

In our system we chose to use the so-called implicit approach [21], which solves for constraints represented in world coordinates. We begin with some basic relations and definitions. The reader will note that all the presented calculations from now on use vector and matrix functions and calculations in order to simplify the understanding of the overall method. Let q be a coordinate vector of size $3n$, n being the number of particles in the system, $C(q)$ the constraint equation vector of size k, and J the Jacobian matrix of size $3n \times k$. Then we have

$$\boldsymbol{J} = \frac{\partial C}{\partial q},$$

$$\dot{\boldsymbol{C}} = \frac{dC}{dt} = \frac{\partial C}{\partial q} \cdot \frac{dq}{dt} + \frac{\partial C}{\partial t} = J\dot{q} + \frac{\partial C}{\partial t} = 0,$$

$$\dot{\boldsymbol{J}} = \frac{dJ}{dt} = \frac{d}{dt}\left(\frac{\partial C}{\partial q}\right) = \frac{d}{dq}\frac{\partial C}{\partial q} \cdot \frac{dq}{dt} + \frac{\partial}{\partial t}\frac{\partial C}{\partial q}$$
$$= \frac{d}{dq}\left(J\dot{q} + \frac{\partial C}{\partial t}\right) = \frac{d}{dq}\dot{C} = \frac{d\dot{C}}{dq}.$$

Note that the immediate derivative of C by time can be omitted if no direct time dependency exists.

Next we use the law of action and reaction to derive the system of equations:

$$F_a + F_c = M\ddot{q}, \tag{1}$$

$$\ddot{C} = \dot{J}\dot{q} + J\ddot{q} + \frac{\partial^2 C}{\partial t^2} = 0. \tag{2}$$

M here represents the diagonal mass matrix of all the particles. Each particle's mass appears three times on the matrix's main diagonal. F_a represents the known forces applied on the system, and F_c are the unknown constraint forces. Here, too, the second immediate derivative of C by time may be omitted if t is not an explicit variable in the equation of the constraint. By setting both $\dot{C}$ and $\ddot{C}$ to zero, we ensure that constraints do not change over time, a fact that enables one to explicitly solve the system. Multiplying Eq. 1 by M^{-1} and J results in:

$$JM^{-1}(F_a + F_c) = J\ddot{q}. \tag{3}$$

Using the expression for $\ddot{q}$ from Eq. 2 we get:

$$JM^{-1}F_c = -\dot{J}\dot{q} - JM^{-1}F_a - \frac{\partial^2 C}{\partial t^2}. \tag{4}$$

Equation 4 is a linear system usually with more degrees of freedom than constraints. Assuming we have k constraints, transforming the system into the constraint space will give us exactly k unknowns and only one possible solution (assuming the constraints are independent). This method is known as the *Lagrange multipliers* transform method for linear systems. Applying the *principle of virtual work* to the system we can write $F_c \cdot dq = 0$, where dq is the virtual displacement. We know that the constraints forbid particles from moving anywhere but in the normal space of the constraint hyper surfaces; thus dq must be orthogonal to the hypersurfaces generated by the constraint directions. We also know that the Jacobian $J = \frac{\partial C}{\partial q}$ is orthogonal to these surfaces, and therefore we get the equality: $J \cdot dq = \frac{\partial C}{\partial q} \cdot dq = 0$. Reexamining the constraint forces, we can now use the above relation and write the constraint forces as follows:

$$F_c = \lambda \cdot J\,. \tag{5}$$

Next we substitute F_c into Eq. 4 and get:

$$JM^{-1}J^T\lambda = -\dot{J}\dot{q} - JM^{-1}F_a - \frac{\partial^2 C}{\partial t^2}\,. \tag{6}$$

The above equations represent a linear system with k unknowns. Solving the equations, we can easily find each of the unknown forces of the system by using Eq. 5.

We now add correcting forces in the direction of the constraint vectors and apply them directly to the system of equations. These forces should depend on the deviation from the correct configuration of the constraints, and thus we get:

$$K_s C + K_d \dot{C} = 0\,. \tag{7}$$

In the above equations, K_s and K_d are the spring stiffness coefficient vector and the damper coefficient vector, respectively. Notice here the similarity to forces of a spring and damper with equilibrium point at zero, action along the constraint's axes. Adding these correction "forces" to the equations system we get the final result:

$$JM^{-1}J^T\lambda = -\dot{J}\dot{q} - JM^{-1}F_a - K_s C - K_d\dot{C} - \frac{\partial^2 C}{\partial t^2}\,. \tag{8}$$

This equation describes the state of the system due to given applied forces F_a, and a set of geometric and time restrictions to obey, represented here by C.

5.3 Applying linear momentum

In some cases, we need to generate a very strong force that will act for a very short time in order to alter the velocity of the system at some location practically instantaneously. The most common example of such forces are those acting during collisions between objects. Such forces are referred to as ***Linear momenta***, and since the change in force appears for an infinitesimally short time δt, we have:

$$L = \int_t^{t+dt} F dt = \int_t^{t+dt} m\ddot{x} dt = m\dot{x}\,.$$

It is obvious that we cannot embed the momenta in the system as presented since the system was conceived to deal with forces. Moreover, we cannot simply add velocities to particles. Doing so will alter the validity of the configuration and violate constraints of the system, leading to a divergence of the entire system. The solution is to use the first-order derivative of the constraints w.r.t. time. So far we have dealt with forces; hence we used the second-order derivatives of the constraints. To use the first-order derivatives, we stop the derivation at the velocity equations (Witkin et al. pointed this out in a different context in [22, 24]). We start with the definition of momentum, where L_a is the applied momenta vector and L_c is the vector of momenta generated by the system in order to maintain the validity of the geometric constraints. Hence we can write:

$$L_a + L_c = M\dot{q} \qquad yielding: JM^{-1}L_c = -JM^{-1}L_a + J\dot{q}$$
$$\dot{C} = J\dot{q} + \frac{\partial C}{\partial t} = 0\,.$$

Combining these terms we get:

$$JM^{-1}L_c = -JM^{-1}L_a - \frac{\partial C}{\partial t}\,.$$

Here again we set the vector of each constraint momentum to be in the direction of the constraint in order to obtain again the Lagrange multipliers for the system, and add "correcting momenta"' as well:

$$JM^{-1}J^T\lambda = -JM^{-1}L_a - K_s C - K_d\dot{C} - \frac{\partial C}{\partial t}\,. \tag{9}$$

Here, L_a represents the applied linear momenta and not forces, as was the case in the previous section. Thus, no forces act in these equations at all. To advance the system through time in the presence of both forces and momenta, we must solve two systems of equations at each given time step—the system of forces and the system of momenta.

5.4 Collisions and contacts—conditions of occurrence

This paragraph relies on work done by Baraff [3, 4]. For simplicity, we assume that a collision always occurs between a plane of a polygon of one of the objects (object 1) and either a polygon or a vertex of the other (object 2). The direction of the collision impulse is in the direction of the plane's normal. We also assume, without loss of generality, that the normal direction ($\hat{n}$) points outward from object 1 and toward object 2. For collisions of singular points (two vertices, vertex and an edge, or two edges) we can generate a seminormal direction applying several known heuristics. In the following equations, p_1 and p_2 represent the collision locations on the surface of objects 1 and 2, respectively.

We start by defining a state function for the collision as follows:

$$\chi_{12} = (p_2 - p_1) \cdot \hat{n} \,. \tag{10}$$

The derivatives in time are:

$$\dot{\chi}_{12} = \frac{d\chi_{12}}{dt} = (p_2 - p_1) \cdot \dot{\hat{n}} + (\dot{p}_2 - \dot{p}_1) \cdot \hat{n} \,,$$
$$\ddot{\chi}_{12} = \frac{d^2\chi_{12}}{dt^2}$$
$$= (p_2 - p_1) \cdot \ddot{\hat{n}} + 2(\dot{p}_2 - \dot{p}_1) \cdot \dot{\hat{n}} + (\ddot{p}_2 - \ddot{p}_1) \cdot \hat{n} \,.$$

The derivative ($\dot{\chi}$) represents the relative velocity of the two colliding objects in the direction of the normal of the collision plane ($\hat{n}$). Differentiating the velocity function ($\ddot{\chi}$) we get an indication of the relative acceleration in the direction of the collision normal. During a collision the equality $p_1 = p_2$ must hold; therefore, we get:

$$\dot{\chi}_{12} = \frac{d\chi_{12}}{dt} = (\dot{p}_2 - \dot{p}_1) \cdot \hat{n} \,, \tag{11}$$
$$\ddot{\chi}_{12} = \frac{d^2\chi_{12}}{dt^2} = 2(\dot{p}_2 - \dot{p}_1) \cdot \dot{\hat{n}} + (\ddot{p}_2 - \ddot{p}_1) \cdot \hat{n} \,. \tag{12}$$

We now test the state functions for possible collisions. When a collision occurs, we are interested in only one of two scenarios. The first one is the case where the two bodies have negative relative velocity in the normal's direction. In other words, the objects move toward each other, hence a collision occurs that must be resolved. This is the state where an *impulse* should be introduced in order to change an object's velocity. We call this short impulse a **collision force** (but it actually is a change in linear momentum, hence an "impulse"). The collision is identified by the following state function:

$$\dot{\chi}_{12} = (\dot{p}_2 - \dot{p}_1) \cdot \hat{n} < 0 \,. \tag{13a}$$

The second scenario is when the relative velocity is zero and the relative acceleration is negative. Here we have a state of contact between the objects with no "real" collision. We still need to resolve this case because the bodies will continue to advance toward each other due to their relative acceleration toward each other. We refer to this state as a state of **resting contact**. Here the solution is to apply a **contact force** in order to eliminate the acceleration (since it is an acceleration and not a velocity, we must use a force and not a linear momentum). The state functions look as follows:

$$\dot{\chi}_{12} = 0 \quad \textit{and} \quad \ddot{\chi}_{12} = (\ddot{p}_2 - \ddot{p}_1) \cdot \hat{n} < 0 \,. \tag{13b}$$

After a collision is detected, we calculate the moment of collisions and recalculate the entire particle system state at that moment.

Resting contacts are treated in much the same manner, with one exception—there is no need to introduce a second equation system since the equations deal with forces.

6 Controlling the puppets

6.1 The problem

A real-life marionette is controlled by an overhead *control frame* (hanging frame) that connects to the puppet's *control joints* by strings (Figs. 2 and 3). An additional string is usually attached to the puppet's heaviest body part in proximity to the center of mass of the entire puppet. This string holds the puppet in its desired rest configuration. The two main problems of simulating this dynamic control model are the motions of the control frame and the behavior of the strings.

The first problem arises from the fact that we do not know the forces that are applied to the virtual control frame. We only have access to the location and orientation of the frame at each time interval; thus the data are not continuous due to the fact that time intervals have noninfinitesimal length. An approximation scheme must therefore be applied here.

The second problem is the choice of the strings. How do we simulate strings so that we get a behavior that is close enough to the behavior of real marionettes hanging by strings? We present several solutions to those problems, each with its advantages and disadvantages.

To generate the frame movement, we examined three approaches:

1. Moving the frame simply by setting its location to the new coordinates given by the user interface. Between each two time intervals the frame is placed in its most recent configuration.
2. Estimating the forces or momenta needed to move the frame's particles to the desired locations.
3. Setting time-dependent constraints that force the frame to move to the desired location while attaching the control points of the frame to the puppet by using a fixed distance constraints. In this context, each time-dependent constraint is actually a 3D curve that represents the change in location of a control point along the time parameter.

Depending on the above method, we tested techniques to simulate the strings. The techniques we applied were:

– Using sets of spring/damper force functuators as springs.
– Adding a virtual "nonphysical" force to the scene.
– Using geometric and time-dependent constraints to maintain a fixed distance from the frame's control points.

6.2 The method of choice

We found that a method based on the first technique above is best for our needs (a solution that is close to the real-life frame control and reaction). Using this technique we move the virtual frame to the exact location given by the input device, and then we apply the forces over the marionette by introducing virtual behavior forces such as springs. The main reason for choosing this method is that there is almost no delay in response time (almost, since springs do introduce some indirect delay while stretching). However, a very undesirable side effect implied in the use of springs is that the configuration of the whole puppet is changed from the desired rest configuration when running the simulation. This is caused by the fact that during the design stage, the marionette was not under the action of forces. The effect becomes more noticeable as the puppet's mass grows. This phenomenon also affects the control over the puppet, since now the joints are located in a different position relative to the control frame.

In order to compensate for these effects, we needed to introduce several new types of forces. The most important one is the **antigravity** force, a name that is justified by the force's action. (How nice it is to have a virtual world!) The new force is specified between two particles, where one of them is the acted upon particle and the other is the trigger particle. The force direction is opposite to the direction of the overall global force (gravity) in the scene.

The force as described helps the springs deal with the puppet's weight, yet it does not improve the response time by much. The result of applying these forces is that the new rest configuration of the system is close to the designed rest configuration. To improve the reaction time of the puppet, we add another feature to the force. This feature might be called a "triggered" action force. Whenever the length between the control particle and the trigger particle is changed, the generated force increases or decreases as a function of the distance between the two particles and the mass of the entire puppet.

This new feature allows very rapid movements yet at the same time appears natural. If configured correctly, the "antigravity" force only affects the particles near its location. Setting this force to act on all particles is useless because it will only reduce the overall gravity, hence the system will act as if under lower gravity (which will look as if the whole action happens on the moon!).

Applying only several of these forces to the main control points of the puppet causes each spring's pull to become faster. In addition, all the main strings that were created to hold the puppet's weight will now be much closer to their rest length, hence the whole system becomes much more stable.

Another type of force that was introduced to the system is a force based on the spring/damper set of forces, but with a different force activation function. While the spring force is linear with the change in distance, our new force resembles the behavior of a shifted sine function between 0° and 90°. The force reaches its maximum action level after a predefined distance and does not change unless the distance becomes shorter. This behavior prevents the introduction of very large forces to the system, hence the system is less exposed to large deviations in the legal configuration state. We therefore gain a reduction in system vibration due to the spring forces of the strings. Nevertheless, such phenomena do exist, but they can be visually reduced by introducing naive algorithms such as the inverse dynamics process, which iteratively shortens invalid string lengths until reaching a desired threshold [19]. The same principle can be applied for dampers in relation to particles velocities.

7 Results

All the simulation systems presented above were tested using the DirectX 5.0 3D environment on top of the Windows 2000 operating system. We ran the simulations on a 450 MHz Pentium 3 CPU, with 128 MB RAM and 512 KB cache memory. We tested more than six different complicated articulated systems (some of them are presented in the images of this paper) having various control frames, masses, and configurations.

Through all the tests, our system's settings were as follows: The system's lower time step bound was set to 0.01 s, and the upper bound was set to 0.05 s. We measured the deviations in our system by the largest amount of divergence from the given geometric constraints. By this definition, the minimal deviation threshold for time step adjustment was set to 0.01%, while the upper bound was set to 1.0% of deviation. The upper deviation threshold was needed so that time step changes due to the ATM would not be too drastic with respect to decreasing the time step interval. On average, all systems set the adaptive time step interval to about 0.02 s (above the minimal time step), and the more stable systems (such as the trooper and the humanoid puppet, which are presented in the following pages) remained fully stable at the upper time step bound through most of the movements.

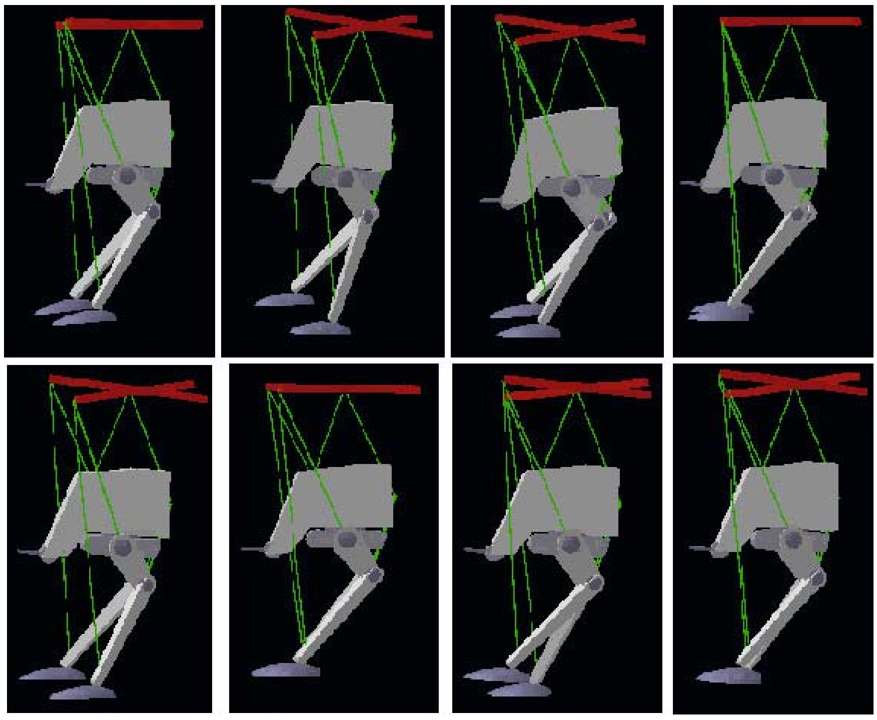


Fig. 7. The trooper from the movie “Return of the Jedi” in eight different walking states when constructed and controlled by our system as a marionette

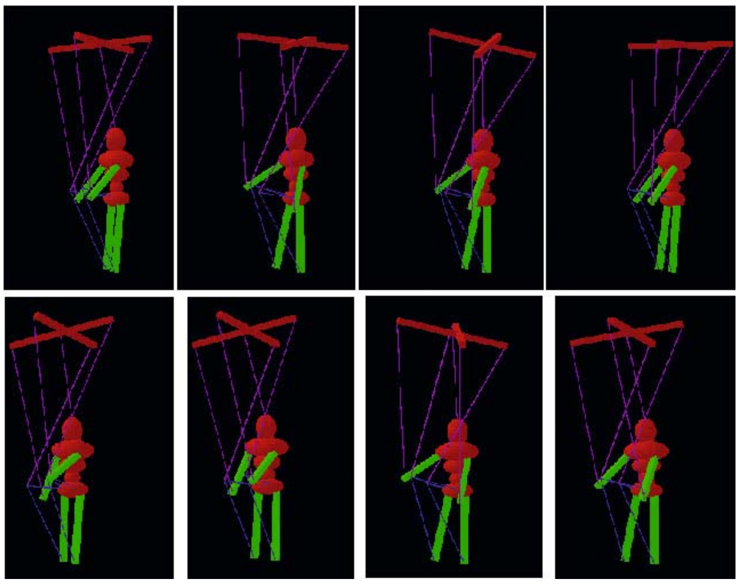


Fig. 8. Two steps made by a simple stick-figure marionette created to test and illustrate the degree of control over humanoid figures

The marionettes were tested for stability under drastic and rapid movement of the controls. When tested with added particle friction (which is our default mode for the marionette system due to its natural motions), the marionettes remained stable at all times, even through rapid changes that might have torn a real marionette. We also tested the system with no friction at all under extreme movement changes. Without any friction, the marionettes did get to a point where they either reached dependency in this system of equations or the deviation was too large and the system was forced to return to its starting configuration. Although these critical states can be avoided by allowing smaller time steps, this will be at the expense of the computation time needed for each frame to be calculated; thus the system will no longer be capable of reacting in real time.

Figures 7–10 are snapshot sequences taken from various scenes that were simulated using our system. We illustrate the behavior of the puppets with a sequence of images from each simulation. All sequences run

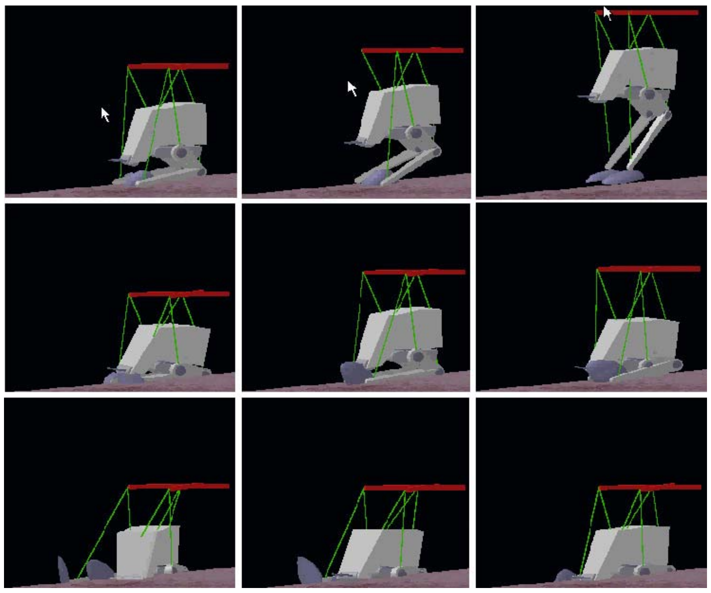


Fig. 9. Demonstration of marionette collision (*first five images*) and contact (*bottom row*). Note the bounce from the floor at the center image and the sliding ahead along the floor over the last three images

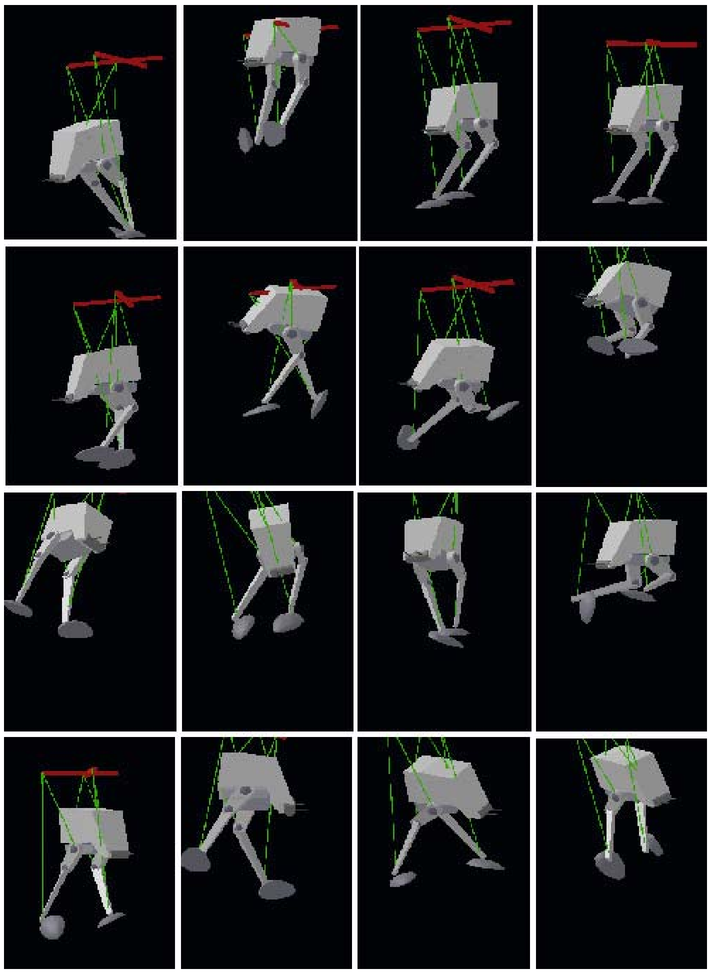


Fig. 10. Test of stability of the system against rapid movements of the control frame of the marionette. The images of the trooper marionette were taken approximately every four frames and near the peak of each oscillation

from top to bottom and at each line from right to left.

8 Conclusions

We presented a method to simulate and control the movements of virtual 3D articulated bodies (puppets) in real time. The system as presented can achieve a high degree of realism in the movement of 3D puppets. We estimate that, given a good input device, a professional puppeteer would be able to easily obtain a quality of movement close to what he/she could have achieved with a real marionette.

The method presented demonstrated the ability to simulate such systems in a manner that is not necessarily completely physical by introducing forces that do not really exist in order to achieve better looking behavior or faster control.

The results that we presented are mathematically based on the described implicit physical solver. We estimate that by using different solver approaches, such as the one described by Barzel et al. [5], could lead to similar results or perhaps better ones due to the handling of true rigid objects instead of particles. Also, better rendering methods such as skinned mesh techniques should increase the photo realism level of the entire system. Nevertheless, the main novelty in our work here lies in the approach to the control of the system, and the use of a physical solver and a rendering system was simply the means to achieve this goal.

Acknowledgement This research was partly supported by the Bar-Nir Bergreen Software Technology Center of Excellence and partly by the George Haber Research Foundation.

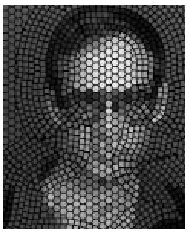
We would like to thank The Visual Computer's anonymous reviewers for their helpful suggestions and comments. We would also like to thank Shy Shalom and Amos Mosseri for their help in designing and implementing the glove input device.

References

1. Bar-Lev A (2003) Virtual marionettes: a system for real-time animation in 3D. MSc research thesis, Technion, Israel Institute of Technology
2. Mourey A (1993) Marionettes: Atelier et Creation, Editions Fleurus, Paris
3. Baraff D (1989) Analytical methods for dynamic simulation of non-penetrating rigid bodies. Comput Graph 23(3):223–232
4. Baraff D (1990) Curved surfaces and coherence for non-penetrating rigid body simulation. Comput Graph 24(4):19–28
5. Barzel R, Bar AH (1988) A modeling system based on dynamic constraints. Comput Graph 22(4):179–188
6. Lee D et al (2001) Reproducing works of Calder. J Visualizat Comput Animat 12:81–91
7. Isaacs PM, Cohen MF (1987) Controlling dynamic simulation with kinematics constraints. In: Proceedings of Siggraph '87, pp 215–224
8. Rose C, Guenter B, Bondenheimer B, Cohen MF (1996) Efficient generation of motion transition using space-time constraints. In: Proceedings of Siggraph '96, pp 147–154
9. van Overveld C (1994) A simple approximation to rigid body dynamics for computer animation. J Visualizat Comput Animat 5:17–36
10. Dahlquist G, Bjorck A (1974) Numerical methods. Prentice-Hall, Englewood Cliffs, NJ
11. Garcia AL (1994) Numerical methods for physics. Prentice-Hall, Englewood Cliffs, NJ
12. Goldstein H (1982) Classical mechanics. Addison-Wesley, Reading, MA
13. Greenwood D (1997) Classical dynamics. Dover, New York
14. Hemami H, Dinneen JH (1993) A marionette-based strategy for stable movement. IEEE Trans Syst Man Cybern 23(2):502–511
15. van Overveld CWAM (1994) A simple approximation to rigid body for computer animation. J Visualizat Comput Animat 5:17–36
16. Hubbard PM (1995) Collision detection for interactive graphics applications. IEEE Trans Visual Comput Graph 1(3):218–230
17. Hubbard PM (1996) Approximating polyhedra with spheres for time-critical collision detection. ACM Trans Graph 15(3):179–210
18. Meriam JL, Kraige LG (1993) Dynamics, 3rd edn. Wiley, New York
19. Desbrum M, Schroder P, Barr A (1999) Interactive animation of structured deformable objects. In: Proceedings of Graphics Interface, June 1999, pp 1–8
20. Watt A, Watt M (1992) Advanced animation and rendering techniques. Addison-Wesley/ACM Press, New York
21. Witkins A, Baraff D (1997) Physical modelling. Siggraph Short Course Notes
22. Witkins A, Gleicher M, Welch W (1990) Interactive dynamics. Comput Graph 24(2):11–21
23. Witkins A, Kass M (1988) Spacetime constraints. Comput Graph 22:159–168 Siggraph Proceedings '88
24. Witkins A, Welch W (1990) Fast animation and control of non rigid structures, Comput Graph 24(4):243–252. Proceedings of Siggraph '90

ADI BAR-LEV was born in Petach-Tikva, Israel, on 7 June 1969. He received both his B.A. and M.Sc. in computer science from the Technion, Israel Institute of Technology. His fields of research and interest are in advanced CG, games technology, animation and motion control, dynamics, and special effects. Currently he works at Samsung Telecom Research Israel (STRI).

ALFRED M. BRUCSKTEIN received his B.Sc. (honors) and M.Sc. in electrical engineering from the Technion, Israel Institute of Technology, Haifa, and his Ph.D. in electrical engineering from Stanford University, Stanford, CA, in 1977, 1980, and 1984, respectively. Since 1985 he has been a faculty member at the Technion, Israel Institute of Technology, where he is currently a full professor, holding the Ollendorff Chair in Science. During the summers from 1986 to 1995 and from 1998 to 2000 he was a visiting scientist at Bell Laboratories, Murray Hill, NJ, USA and in 2001–2002 a visiting chaired professor at Tsing-Hua University in Beijing, China. His research interests are in image and signal processing, computer vision, computer graphics, pattern recognition, robotics (especially ant robotics), applied geometry, estimation theory and inverse scattering, and neuronal encoding process modeling.
Professor Bruckstein is a member of SIAM, AMS, and AMM and is presently the dean of the Technion Graduate School.



GERSHON ELBER is an associate professor in the Computer Science Department, Technion, Israel. His research interests span computer-aided geometric design and computer graphics. Professor Elber received a B.S. in computer engineering and an M.S. in computer science from the Technion, Israel in 1986 and 1987, respectively, and a Ph.D. in computer science from the University of Utah, USA, in 1992. He is a member of the ACM and IEEE.
Professor Elber serves on the editorial board of Computer Aided Design, Computer Graphics Forum, and the International Journal of Computational Geometry and Applications and has served on many conference program committees including Solid Modeling, Pacific Graphics, Computer Graphics International, and Siggraph. Professor Elber was one of the paper chairs of Solid Modeling 2003 and Solid Modeling 2004.