Water with Character(s): Fluid Simulation with Articulated Bodies

Category: Research

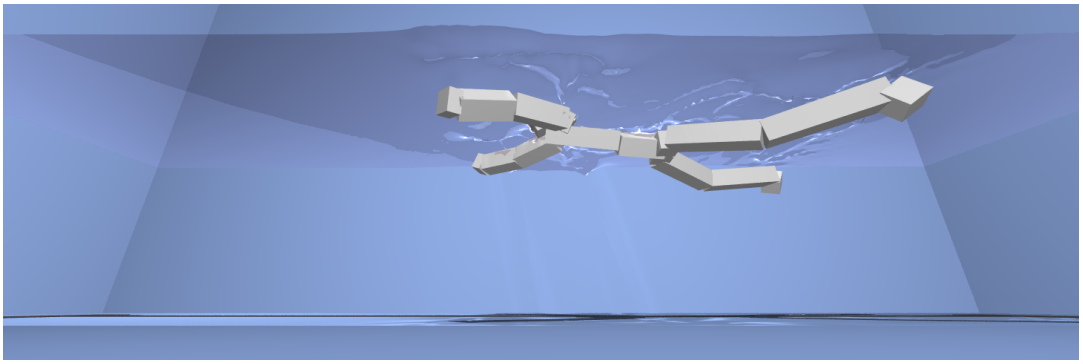

Figure 1: Motion capture of a swimmer shown interacting with a simulated fluid.

Abstract

We present an algorithm for creating realistic animations of characters that are interacting with fluids. Our approach combines dynamic simulation with data-driven (motion capture) and related kinematic motions to produce realistic animation in a fluid. The interaction of the articulated body with fluids is done by (a) incorporating joint constraints with rigid animation, and by (b) extending the *Rigid Fluid* method, which allows simulation of rigid bodies in a fluid, to deal with articulated chains. We introduce a constraint-based solver for coupling motion trajectories of articulated structures with fluid simulation. This solver takes as input the current state of the simulation and calculates the exact angular and linear accelerations of the connected bodies to incorporate motion trajectories into the simulation. These accelerations are used to estimate the forces and torques that are applied at each joint. These forces are used to conform the motion trajectories of the articulation data with the simulation of the fluid. We demonstrate results for animations such as swimming in water. The ability to have articulated bodies interact with fluids also allows us to generate simulations of simple water creatures that are driven by simple controllers.

Keywords: Physically Based Animation, Fluid Simulation, Motion Capture

1 Introduction

The field of computer graphics has recently witnessed significant maturity in the use of physics-based simulation and data-driven methods for generating realistic animations. In part, use of physical simulation has provided for some very good approaches for animating fluids (see [Enright et al. 2002], [Foster and Fedkiw 2001], [Carlson et al. 2004] for a review). Character animation either remains in the hands of a talented animators with good tools for keyframing or has found a new life in the form of data-driven animation. Motion capture technologies are now in wide use to capture data of performances to then animate characters, as it provides all the details and nuances of live motion. The usability of motion capture for animation has been further enhanced by some very practical approaches for the adaptation and reuse of already captured data (see [Bruderlin and Williams 1995] [Gleicher 1998], [Kovar et al. 2002], [Lee et al. 2002]). However only a few efforts make the data of character motion interact with the surrounding physical environment, especially a complex simulated environment with fluids. For example, it would be very interesting to take a kinematic motion of a fish and put it in simulated water to animate the effects on the water by the fish and conversely the effects on the fish from the simulated fluid. Similarly, it would be of great benefit for animators if motion capture data of a person performing swimming actions in the air could be put in simulated water to generate swimming animations, effecting both the character and the fluid.

To this end, we propose a method for simulation of the interaction of an articulated character (and its underlying motions and geometry) with a fluid simulation. Using our approach an animator can take any form of motion trajectory associated with a geometry, either in the form of a motion capture sequence or a related kinematic animation, insert it into a physical simulation of water, and animate the effects that one has on the other. In our method, we use the motion trajectories from an articulated character to calculate joint forces that closely mimic the input data when applied to the physical model. Once we have this *simulation* of the character, we make it interact with the fluid. To achieve this interaction, we turn to some of the recent efforts on simulating two-way interactions between rigid bodies and fluids. Specifically, we build on the *Rigid Fluid* method for full two-way coupling between the rigid bodies and fluids [Carlson et al. 2004]. As we are interested in articulated character animations, we need to extend the Rigid Fluid approach to incorporate rigid bodies with joints.

Our method consists of two main steps:

1. **Force/torque extraction:** Given motion capture data, we calculate the forces and torques that must be applied to our articulated body joints during simulation so that it follows the motion capture data as closely as possible. We use a constraint-based method discussed in Section 3, for this purpose.

2. **Fluid-Body coupling:** Once we have the joint forces and torques, we simulate the articulated body in the fluid environment using our extension of the *Rigid Fluid* method. This interaction changes the fluid environment, (e.g., creates splashes) and also affects the articulated body simulation (e.g., a person doing a swimming action is pushed forward).

The above two steps are repeated to form a feedback mechanism. The overall flow diagram of our approach is shown in Figure 2. The

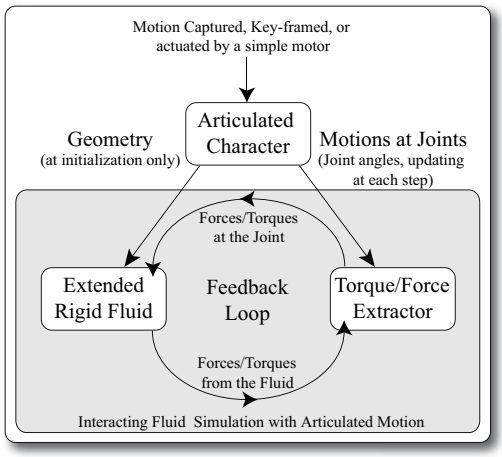

Figure 2: Flow diagram of our approach.

forces due to fluid-body coupling may cause the simulation to deviate from the motion capture data, but the force/torque extraction step tries to bring it back.

In this paper, we also further extend the issue of two-way coupling between objects and fluids. We have seen animations of a ball or feet, creating splashes of water[O'Brien et al. 2000; Enright et al. 2002; Foster and Fedkiw 2001], to simulations of buoyancy on objects due to a fluid [Foster and Metaxas 1996]. In the former case, the trajectory of the ball is not effected and in the latter the object does not cause any changes to the water. Only recently have we seen animations where the motion of the object affects the fluid and vice-versa [Carlson et al. 2004; Génevaux et al. 2003; Guendelman et al. 2005]. We use a full two-way coupling enabling us to demonstrate effects not viable before. For example, the two-way simulation allows us to simulate surface effects like splashing while impacting the motion of the object in the fluid. In addition, our articulated structures impart complex forces on the fluid and in return the fluid propels the structure in ways we expect. Using our approach, we are able to dynamically simulate characters swimming in water. Using a simple controller, we also validate a well-know theory of a Purcell swimmer [Purcell 1976].

2 Related Work

Most of the work in motion capture research is focused on reusing already captured data to (i) synthesize new ones by copying pieces from a database and reassembling them, or by (ii) generating new motions by using kinematics constraint satisfaction to search for new variations on existing data. Recently however, some researchers have tried to combine motion capture and simulation so that motion captured characters can dynamically interact with a physically simulated environment.

Oshita and Makinouchi [2001] use simulation to allow a character to respond to a mass being dropped on its back. Zordan and Hodgins [2002] create simulations from motion capture data that are able to hit and react to collisions. This work is in spirit closest to ours. They choose model motion capture data with proportional-derivative (PD) servos, and allow passive dynamics to take over during impacts. The stiffness parameters of the servo are interpolated to smoothly switch between tracking and passive control. Unlike this *transition* between tracking motion capture data and passive dynamics, as we are interested in interacting with fluids, we need to deal with the simulation continuously interacting with the physical environment at all times.

Recently Yang *et al.* [2004] developed a layered strategy to control swimming motions. In their approach, they employ a simple viscous drag model and do not simulate the fluid. They specifically state that "A full CFD [computational fluid dynamics] with unsteady flow is, however, necessary to truly capture the motion of swimming characters." [Yang et al. 2004].

Previous work on characters that interact with fluids use simple models for calculating forces between fluid and solids. Tu and Terzopoulos [1994] compute forces on a fish surface using the surface normal and relative velocity between the surface and water. Their simulation does not affect the water at all and hence surface phenomena like splashing could not be demonstrated. Along the same lines, Wu and Popovic [2003] use a simple aerodynamics to model the lift and drag forces on birds.

Several methods have been used in graphics to allow modeling of full two-way coupling between fluids and solids. Yngve *et al.* [2000] simulate destructive explosions with two-way coupling of breakable rigid bodies and compressible high speed fluid. Génevaux *et al.* [2003] implement two way coupling between incompressible fluid and deformable solids modeled by mass/spring systems. Takahashi *et al.* [2002; 2003] model the interaction of incompressible fluids and rigid bodies by setting the velocity of the rigid body as boundary conditions to the the fluid solver, and using the fluid's pressure to calculate a normal force acting on the surface of the rigid body. Guendelman *et al.* [2005] use a similar object velocity/fluid pressure coupling to animate the interaction of incompressible fluids with zero volume deformable and rigid shells. We chose to use the rigid fluid method from Carlson *et al.* [2004] to model our rigid bodies interacting with incompressible fluid, because it affords an easy translation of forces between the simulations, was straight-forward to implement. The method scales linearly, and it adds very little over-head compared to the cost of the fluid simulation.

In addition to above methods of simulating fluids, much work has also been fruitful for dynamics modeling of articulated figures. Methods for defining controllers for complex behaviors [Hodgins et al. 1995] to learning simple controllers [Faloutsos et al. 2001] have been proposed. However, most of these methods are more suitable for forward dynamic simulation [Redon et al. 2005] improving on efforts in dynamic robotic manipulation [Featherstone 1987].

Lastly, we recognize that other coupling techniques for solid/fluid interaction such as [Guendelman et al. 2005] may also be appropriate for our goals. We believe that our constraint-based solver will work equally well with the other approaches listed above.

3 Force/Torque Extraction

Our goal is to create a physical system modeled on the articulated data (and it's motion) so that the articulated character can exert and react to forces. For this purpose, we have devised an approach to compute joint forces and torques that allow the simulation to model (and mimic) the motion data as accurately as possible. Since our character simulation continuously interacts with the fluid, the forces applied by the fluid also need to be accounted for while calculating these joint forces and torques.

3.1 Constraint based Force/Torque Solver

For simplicity of exposition, consider the 2D case in which the orientation of a body can be represented by a scalar angle θ (e.g., counter-clockwise with respect to the x-axis). Details regarding the extension to the 3-D case, which requires dealing with quaternions is described in the next section (Section 3.2).

We model the motion data by constraining the angles of simulated joints to mimic with the corresponding angles in the mo-

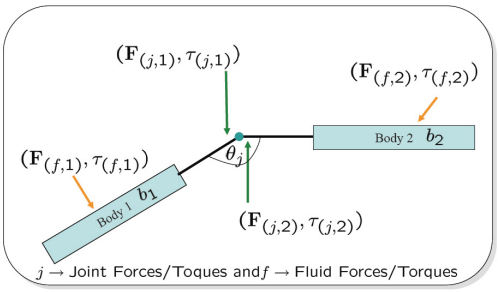

Figure 3: A simple 2 body articulated structure shown to clarify notation. The two bodies are acted upon by forces from the fluid and other forces, in addition to the internal forces due to the joint.

tion data. For instance, consider a joint j between bodies b_1 and b_2 at some time t, with angles $\theta_1(t)$, $\theta_2(t)$, angular velocities $\omega_1(t)$, $\omega_2(t)$, and angular accelerations $\alpha_1(t), \alpha_2(t)$, respectively, as in Figure 3. The angle of the joint $\theta_{(j,1-2)}(t)$ [for notational simplicity, we refer to $\theta_{(j,1-2)}(t)$ as $\theta_j(t)$] can then be defined as the angle of b_2 as seen by b_1, i.e.,

$$\theta_j(t) = \theta_2(t) - \theta_1(t) \tag{1}$$

The equation of evolution of the joint angle can then be written as:

$$\frac{d}{dt}\theta_j(t) = \frac{d}{dt}\theta_2(t) - \frac{d}{dt}\theta_1(t) = \omega_2(t) - \omega_1(t) = \omega_j(t) \tag{2}$$

where $\omega_j(t) = \omega_2(t) - \omega_1(t)$ is defined to be the angular velocity of the joint j at time t, which is also same as the angular velocity of b_2 as seen by b_1. The equation for the evolution of $\omega_j(t)$ can be written similarly

$$\frac{d}{dt}\omega_j(t) = \alpha_2(t) - \alpha_1(t) = \alpha_j(t) \tag{3}$$

where $\alpha_j(t) = \alpha_2(t) - \alpha_1(t)$ is defined to be the angular accelerations of the joint j at time t.

We want to apply forces and torques on the bodies such that the joint angles of our simulation will closely match the motion data joint angles at some future time $\tilde{t}$. The forces/torques we apply to the bodies at time t will affect the angular accelerations at that time. Using a simple explicit forward Euler integrator, the angular acceleration at time t will affect the angular velocities at time $t+\Delta t$, which in turn will affect the joint angles θ_j at time $t+2\Delta t$. Thus in our explicit forward Euler approximation, the forces and torques applied now will affect the joint angle θ_j two time steps later:

$$\theta_j(t+2\Delta t) = \theta_j(t+\Delta t) + \omega_j(t+\Delta t)\Delta t \tag{4}$$

$$\text{where } \theta_j(t+\Delta t) = \theta_j(t) + \omega_j(t)\Delta t\,.$$

Since we want our simulation to model the motion data, we set $\theta_j(t+2\Delta t)$ above to be equal to the desired joint angle $\hat{\theta}_j(t+2\Delta t)$ as calculated from the motion capture data; with this substitution, equation (4) gives

$$\omega_j(t+\Delta t) = \frac{\hat{\theta}_j(t+2\Delta t) - \theta_j(t+\Delta t)}{\Delta t}\,. \tag{5}$$

That is, given the current state of the simulation $(\theta_j(t), \omega_j(t))$, we are able to calculate the desired angular velocity $\omega_j(t+\Delta t)$ at the next time step. This gives us an estimate of the desired angular acceleration $\alpha_j(t)$:

$$\alpha_j(t) = \frac{\omega_j(t+\Delta t) - \omega_j(t)}{\Delta t}\,. \tag{6}$$

With the desired $\alpha_j(t)$'s, the forces/torques to be applied at each joint can be calculated using the following three sets of equations:

1. Equations relating angular acceleration of joint j to that of its constituting bodies b_1, b_2.

$$\alpha_j(t) = \alpha_2(t) - \alpha_1(t). \tag{7}$$

2. Newton's third law equations. Suppose a joint j between bodies b_1, b_2 applies forces $\mathbf{F}_{(j,1)}$, $\mathbf{F}_{(j,2)}$ and torques $\tau_{(j,1)}$, $\tau_{(j,2)}$ to the bodies b_1 and b_2, respectively. As these are internal forces (body b_1 applying force on b_2 and vice versa), they have to be equal and opposite, i.e.,

$$\mathbf{F}_{(j,1)} = -\mathbf{F}_{(j,2)}. \tag{8}$$

   A similar relation will hold true for the torque about a fixed internal point, say p_o, the origin for our global coordinate system, if we assume the *strong form* [Halliday and Resnick 1978][1] of Newton's third law.

$$\tau_{(j,1)} \mid_{p_o} = -\tau_{(j,2)} \mid_{p_o}\,. \tag{9}$$

3. Equation relating angular acceleration of a body, b_i to all the forces acting on the body. The forces/torques, $(\mathbf{F}_{b_i}, \tau_{b_i})$ acting on each body include all the joint forces/toques, fluid forces/torques and forces due to gravity and collisions.

$$I_{b_i}\alpha_i = \tau_{b_i} - \mathbf{r}_{b_i} \times \mathbf{F}_{b_i}, \tag{10}$$

   where I_{b_i} is the moment of inertia of the body and $\mathbf{r}_{b_i}$ is the vector to center of mass of the body b_i

   Using this formulation, our solution takes into account these external forces automatically, which allows us to follow the motion data more accurately. Since PD-Servos do not take these external forces/torques into consideration, using them to model motion data in a complex fluid environment is complicated.

3.2 Extending Constraint-based Solver to 3-D

Unlike the 2D case, the orientation of a body cannot be represented by a single angle in 3-D, so we use quaternions to represent the orientations of the bodies. A quaternion q is composed of a scalar/real part s and vector part $\mathbf{v}$, i.e. $q \equiv (s, \mathbf{v})$. All q quaternions below will be *unit* quaternions, since rotations and orientations correspond to unit quaternions. In order to provide a concise description of our constraint-based solver here, we assume a basic knowledge of quaternion algebra (see, e.g., [Shoemake 1985]).

We again consider the joint j between two bodies, b_1 and b_2, with their orientations now represented by quaternions $q_1(t)$ and $q_2(t)$, respectively, and the quaternion of the joint, $q_j(t)$, defined as the quaternion representing the orientation of b_2 as seen by b_1:

$$q_j(t) = q_1^{-1}(t)q_2(t) \tag{11}$$

where $q_1^{-1}(t)$ is the inverse of $q_1(t)$. We recall that the inverse of a quaternion $q \equiv (s, \mathbf{v})$ is given by $q^{-1} = q^*/N(q) = (s, -\mathbf{v})/N(q)$, where $N(q)$ is the squared norm of q, and so the inverse of a unit quaternion is equivalent to its conjugate.

[1] *Strong form* of Newton's Third Law requires that in addition to being equal and opposite, the forces must be directed along the line connecting the two particles

For a rigid body rotating with angular velocity $\omega(t)$, the time derivative of its quaternion $q(t)$ is given by

$$\frac{d}{dt}q(t) = \frac{1}{2}\omega^q(t)q(t) \tag{12}$$

where ω^q is the quaternion representation of the angular velocity vector ω, i.e., $\omega^q = (0,\omega)$. Now, given the angular velocities of the two bodies b_1 and b_2 constituting the joint j, our aim is to find the time derivative of the joint quaternion $q_j(t)$, defined in equation (11). If we define $\omega_j(t) = \omega_2(t) - \omega_1(t)$ to be the angular velocity of the joint j as we did in the 2-D case, it can be shown that

$$\frac{d}{dt}q_j(t) = \frac{d}{dt}q_1^{-1}(t)q_2(t) = \frac{1}{2}(q_1^{-1}\omega_j^q q_1)q_j\,. \tag{13}$$

That is, if we define $\Omega_j = q_1^{-1}\omega_j^q q_1$, then

$$\frac{d}{dt}q_j(t) = \frac{1}{2}\Omega_j q_j\,. \tag{14}$$

We note that $\Omega_j = q_1^{-1}\omega_j^q q_1$ is the quaternion representation of the vector ω_j in the reference frame fixed to b_1. Intuitively, equation (14) can then be thought of as the general equation (12) written in the reference frame fixed to b_1, where we replace q by the *relative* quaternion $q_j = q_1^{-1}q_2$ of b_2 as seen from b_1, and the angular velocity ω by the *relative* angular velocity Ω_j

In the 2D case, equation (4) described the desired orientation angle target after two time steps. To make the 3D simulation similarly follow the joint orientation data, we will set the joint quaternion after two time steps, $q_j(t+2\Delta t)$, to be equal to the desired joint quaternion determined from the motion capture data ($\hat{q_j}(t+2\Delta t)$):

$$\hat{q_j}(t+2\Delta t) = q_j(t+\Delta t) + \frac{1}{2}\Omega_j(t+\Delta t)q_j(t+\Delta t)\Delta t \tag{15}$$

$$\text{where}\;\; q_j(t+\Delta t) = q_j(t) + \frac{1}{2}\Omega_j(t)q_j(t)\Delta t$$

To simplify the notation of the above form, we replace $\hat{q_j}(t+2\Delta t)$ with $\hat{q}$, $q_j(t+\Delta t)$ with q, and $\Omega_j(t+\Delta t)$ with Ω to get

$$\hat{q} = q + \frac{1}{2}\Omega q\Delta t \tag{16}$$

We thus naïvely expect to solve for the unknown Ω in the above equation by simply post-multiplying both sides by q^{-1}. However, this entire operation appears problematic since we required orientations and rotations to be described by unit quaternions; while the differential equation (12) for q maintains constant $N(q) = 1$, our simple discretization in equation (16) violates the unit-quaternion condition. We could fix this by forcibly renormalizing the right side of equation (16) to have unit norm, but this leads to a complex quadratic equation for Ω. We instead fix this problem by introducing an additional variable s describing the non-unit norm resulting from the discretization error,

$$sq^d = q + 1/2\Omega q\Delta t\,, \tag{17}$$

and appeal to the requirement that the real part of Ω must be zero; that is, Ω must be the quaternion representation of a vector. Multiplying both sides of equation (17) by q^{-1}, rearrangement yields

$$\Omega = (sq^d q^{-1} - 1)(2/\Delta t), \tag{18}$$

with the requirement that $real(sq^d q^{-1}) = 1$. This system can now be easily solved for the desired $\Omega = \Omega_j(t+\Delta t)$ angular velocity at the next time step. This $\Omega = \Omega_j(t+\Delta t)$ plays the same role in 3D that the desired $\omega_j(t+\Delta t)$ in equation (6) played in 2D: the desired Ω at the next time step determines the angular accelerations that must be applied at the current time by the forward Euler relations, and those angular accelerations give the conditions for solving for the forces and torques imposed at the joints.

The above considerations give an under-constrained system of linear equations for the unknown joint forces $\mathbf{F}_{(j,1)}$, $\mathbf{F}_{(j,2)}$ and torques $\tau_{(j,1)}$, $\tau_{(j,2)}$, because we have not yet included any conditions on the relative distances of the connected bodies. These additional constraints complete the system of the equations and are discussed next as we discuss the simulation method for fluids.

4 Incorporating the Joint Constraints in Rigid Fluid Method

A joint is a relationship that is enforced between two bodies so that they can only have certain positions and orientations relative to each other. For example a *ball* and *socket* joint constrains a point (the "ball") of one body to be in the same location as a point (the "socket") of another body. A hinge joint on the other hand constrains the two parts of the hinge to be in the same location and to line up along the hinge axle.

When two bodies connected through a joint are subjected to external forces and torques, the linear and angular accelerations thus caused may try to violate the joint constraints. To counter this, the bodies apply forces and torques on each other in such a way that the net resulting acceleration does not violate the constraints.

4.1 Solving Joint Constraints

We now describe how we solve for these internal forces and torques to satisfy the joint constraints. We are given as input the *external* forces and torques acting on each of the bodies and we want to calculate the *internal* forces/torques due to the joints which will prevent the violation of joint constraints. We form three sets of equations, quite similar to the ones discussed in Section 3.1.

1. **Newton's Second Law equations:** Each body b may be connected to multiple joints. Each of these joints will apply an *internal* force and torque on the body. In addition there will be other external forces due to gravity, collision, fluid coupling etc. Let these forces and torques be $\mathbf{F}_b = \{F_{b_1}, F_{b_2}, \ldots, F_{b_n}\}$ and $\tau_b = \{\tau_{b_1}, \tau_{b_2}, \ldots, \tau_{b_n}\}$. The linear acceleration of the center of mass of the body can then be written as:

$$m_b\mathbf{a} = \sum_k \mathbf{F}_{b_k}, \tag{19}$$

   where m_b is the mass of the body b. The equation for angular acceleration is the same as equation (10). This gives us equations in terms of the unknown internal forces and torques. Such equations are constructed for each body in the system.

2. **Joint constraint equations:** Given the acceleration a_c of the center of mass m_c of a body, its angular velocity ω and angular acceleration α, the linear acceleration a_p of any point m_p on the body can be obtained as:

$$\mathbf{a}_p = \mathbf{a}_c + \alpha \times \mathbf{r}_{pc} + \omega \times (\omega \times \mathbf{r}_{pc}) \tag{20}$$

   Now, by imposing joint constraints on these accelerations (e.g., the acceleration of the *ball* in body b_1 should be same as that of the *socket* in body b_2, as they are connected through a ball and socket joint), gives us another set of equations. Such an equation is constructed for each of the joints.

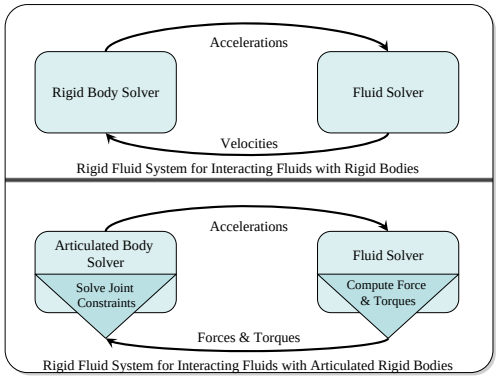

Figure 4: A simple schematic highlighting the Fluid solver that works with rigid bodies and its extension to work with articulated rigid bodies.

3. **Newton's Third Law equations:** This set of equations correspond exactly to the equations (8) and (9) in Section 3.1.

This gives us a system of linear equations in terms of the internal joint forces and torques. However this system is also under constrained. The missing constraint is provided by the physical properties of the joints. For instance, in a *ball* and *socket* joint, the torque at the joint, due to internal forces is zero as the joint allows for complete freedom to rotate in all directions. Conversely, in the case of a *hinge* joint the torque about the hinge axle should be zero, with constraints preventing motion on the other axis. In the case of limbs with hinge joints (e.g., elbows) torques are applied to create motion (in case of arms, via muscles).

As we have the data from our motion trajectories, our goal is to use the kinematic data, as constraints on the motion, to compute the forces. For this we turn to equations we discussed in Section 3, specifically equations (8) and (9) establishing the Newton's third law and equation (10) for Newton's second law. Equation (7) provides an additional constraint relating the difference in angular accelerations for bodies connected by a joint. This gives us a complete system of equations which can now be solved for the internal joint forces and torques, which will not only satisfy the joint constraints, but also help the simulation to model the motion data accurately.

Now we discuss how to integrate these joint constraints into the *Rigid Fluid* framework to allow for two-way coupling between articulated bodies and fluids. Before doing this let us summarize, in brief, the important aspects of the *Rigid Fluid* method to understand the integration more clearly.

4.2 Rigid Fluid, a brief summary

The Rigid Fluid method uses the Navier-Stokes equations to solve for the fluid and employs a rigid projection step to model the two-way coupling between rigid bodies and fluid (see Figure 4 (top)). The computational domain is divided into two parts - the part containing only the fluid is called $\mathscr{F}$ and the part occupied by the rigid bodies is called the solid domain $\mathscr{R}$. The *Rigid Fluid* method essentially consists of two main subsystems:

1. **Rigid Body Solver:** During each time step of the simulation, the rigid body solver uses the initial velocity gathered from the fluid solver, applies forces due to collision, etc., to the solid objects, and updates their positions.

   To properly transfer momentum between the solid and fluid domains due to these forces, the solver also maintains a running sum of the accelerations created on each body over the time step. Similarly the angular accelerations about each body's center of mass are calculated. These linear and angular accelerations are then passed as input to the fluid solver.

2. **Modified Fluid Solver:** The modified fluid solver consists of three main steps:

   (a) **Solving Navier-Stokes Equations:** During this step, the rigid objects in contact with fluid are treated *exactly as if they were fluid*. The Navier-Stokes equations are solved in the entire domain $\mathscr{C} = \mathscr{F} \cup \mathscr{R}$ in the usual fashion, using a semi-Lagrangian technique [Stam 1999] and pressure projection. We have also extended this to use semi-Lagrangian with Back and Forth Error Compensation and Correction (BFECC) [Kim et al. 2005].

   (b) **Incorporating Rigid Body Forces:** The rigid body forces due to collisions, etc., that were calculated in the rigid body solver are accounted for in this step. The accumulated linear and angular accelerations from the rigid body solver are used to change the velocities in the solid domain $\mathscr{R}$. The buoyant forces arising from *relative density* are also considered in this step.

   (c) **Enforcing Rigid Motion:** This step enforces rigidity in the solid domain $\mathscr{R}$. The velocities in $\mathscr{R}$ are gathered together to find the velocity of the center of mass and the angular velocity of the rigid body. This is then projected back on the grid cells in $\mathscr{R}$, so that every point in the rigid body obeys the same rigid body motion.

4.3 Extensions to Rigid Fluid

The rigid body solver influences the fluid solver by providing to it the accumulated linear and angular accelerations due to the forces from collisions, etc., acting on the rigid bodies. This is the top loop in the Figure 4. The fluid solver, on the other hand, affects the rigid body solver by changing the rigid body velocities due to a combined effect of the Navier-Stokes solution, buoyancy forces and rigidity enforcement. To replace the rigid body solver with an articulated body solver, we incorporate a joint constraints solver, which requires us to change the communication between these two solvers.

In our approach the articulated rigid body solver takes into account the joint constraints as explained in Section Section 4.1. The resulting forces and torques due to solving the joint constraints are accumulated into the linear and angular accelerations, so that momentum is properly transferred between the solid and fluid domains. Since the fluid solver changes the linear and angular velocities of the rigid bodies, this needs to be reflected in the rigid solver so that the joint constraints can take that fact into account. To incorporate this we calculate the difference in each rigid body's linear and angular velocities caused by the fluid solver at each time step. This difference is multiplied by the corresponding mass and moment of inertia of the rigid body to obtain the resultant force and torque, respectively. These are then passed to the rigid body solver as *external* forces and torques, which in turn are applied to the rigid body. The rigid body solver uses this information to solve the joint constraints. The bottom of part Figure 4 shows the additional modules added for this purpose.

5 Results

We have undertaken various simulations using (1) simulation of simple water creatures and mechanisms actuated by hand written controllers, and (2) motion capture of subjects trying to mimic swimming actions in air and putting these models in the water. We describe and discuss these results in this section.

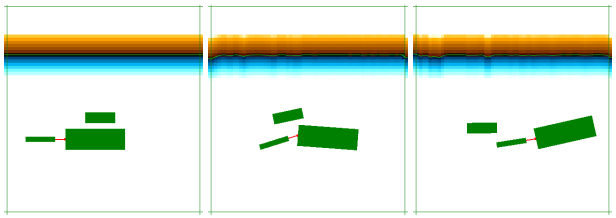

Figure 5: A freely floating body alongside a ***flipper*** in low viscosity fluid

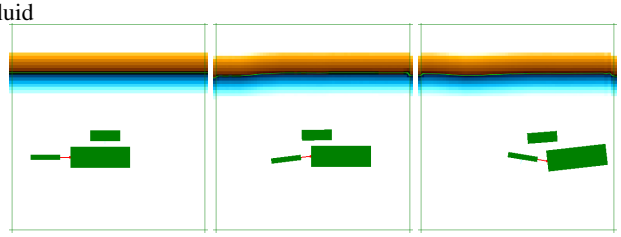

Figure 6: A freely floating body alongside a ***flipper*** in high viscosity fluid

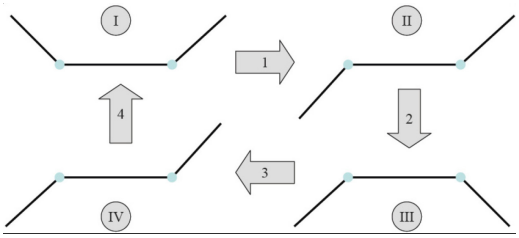

Figure 7: The configuration cycle followed by a Purcell swimmer[Purcell 1976].

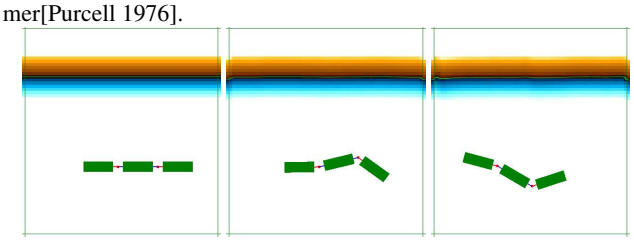

Figure 8: A ***Purcell swimmer*** moving to the left.

5.1 Simple underwater creatures: Flipper and Purcell swimmer

Using the coupling between articulated bodies and fluids we create a simulation of simple water creatures using hand-written controllers.

Figures 5 and 6 shows a simple ***flipper*** in which a large body is pushed forward by a small tail that is attached to the body. All body parts here are neutrally buoyant (relative density 1). A simple controller at the joint applies torques on the two parts to create a flipping motion. The sequence shows the flipper moving correctly under the influence of forces due to interaction with the fluid.

Since we perform a full two-way coupling between the articulated body and fluids, we are able to both demonstrate effects like viscous drag and explore their variation across different viscosities (equivalently speaking non-dimensionally, across different Reynolds numbers). Figure 5 shows a body freely floating alongside a ***flipper***. In this example we have kept the viscosity of the fluid low to minimize viscous effects. As the flipper pushes the water back to move forward, the freely floating body is also pushed back along with the backward flow of the fluid. In Figure 6 we have increased the viscosity of the fluid to make the viscous effects more pronounced. As one can see, the freely floating body now moves along with the flipper and the nearby fluid is viscously dragged along with the body. Note that these effects can only be demonstrated and distinguished if we perform a full two-way coupling between the fluid and the bodies. For comparison, simply modeling an everywhere-uniform drag force would fail to capture such interactions between different bodies in fluids.

Further demonstrating the effects of different viscosities on swimming motions, we find it useful to consider the case of the ***Purcell swimmer***. E.M. Purcell identified in 1976 that the simplest "animal" that can swim at zero Reynolds numbers is one with three straight elements connected by two hinges. A single-hinged creature like our simple flipper above makes little headway at small Reynolds number because inertia is unavailable to break the time reversibility of the different elements of the swimming cycle. In contrast, Purcell proposed a sequence of configurations (shown in Figure 7) which enables the three-link ***Purcell swimmer*** to propel itself in an irreversible way, even at zero Reynolds number. We wrote controllers to apply forces at the two joints in such a way that the swimmer follows the desired configurations. Figure 8 shows frames from an animation of the swimmer. As is predicted in [Becker et al. 2003], our swimmer swims in the left direction due to the fluid forces. To our knowledge, this is the first CFD simulation of this mechanism.

We note that understanding viscosity effects in swimming is a subtle matter, and that it can be puzzling to unravel the conditions on which decreasing viscosity alternatively impedes or aids a particular swimming motion. On the one hand, increased viscosity increases drag on the swimming object; but this increased viscosity simultaneously gives a swimmer a firmer material with which to push off of. Cussler and Gettelfinger [2004] recently established that these two effects effectively cancel each other out for human swimmers. Their experiment was so humorously dramatic, including competitive collegiate swimmers in a pool of guar, that it netted a 2005 IgNobel Prize for Chemistry. For our purposes, it demonstrates the complexities involved in human swimming.

5.2 Mocap Swimming

Now we will briefly discuss our experiments with motion capture data and how it interacts with fluids. For these animations, we captured a variety of swimming strokes like the ***crawl***, ***breaststroke*** and ***backstroke***. Motions like swimming strokes are very hard to mocap. Figure 9 shows three different ways our subjects were asked to perform the required motions. Firstly, these are hard motions to perform, as most of the real dynamics are only possible when you one is immersed in water, and secondly, all of these capture setups are extremely cumbersome. Nonetheless, we were able to capture several instances of ***crawl***, ***breaststroke***, and ***backstroke***. Correct looking butterfly stroke was just not possible as none of our subjects could move their bodies in the pre-requisite manner required for that stroke, while suspended or supported from below. In the next few paragraphs, we discuss some of the details and images of these animations. The included video is the best way to evaluate these results. In addition to simulating these animations with different configurations, we also recorded a video of a person swimming for visual comparison. The subject in the video tape is not the same as the one motion captured and we do not attempt any form of synchronization between the video and the animation.

The input to our system is cleaned up motion capture data. Joint angles of the various joints are then extracted to give the desired angle term $\hat{\theta}$ used in Equation (5).

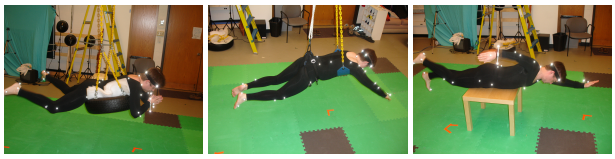

Figure 9: Pictures of a mocap session. Capturing mocap of a user doing motions like swimming is not that easy. Here are shown three different configurations.

For simplicity we modeled each part of the human articulated body as a cuboid with a predefined thickness and length corresponding to the limb length in the motion capture data. Using full meshes to model the articulated bodies is also possible. In most cases the relative density of the bodies was kept close to one, however we lowered the value a little when we wanted the swimmer to stay close to the surface, in particular for the crawl and backstroke.

Another problem we had was that of control. When left entirely on its own, our simulated swimmer moved in all directions. Though the simulated motion looked completely natural, the usefulness of the method was heavily compromised. To avoid this problem we applied small external torques on the torso of the body so that it maintains a fixed orientation in space. This is in essence similar to a simple balance controller used in dynamics walking simulations.

Figure 10 shows a view of the swimmer performing a ***breaststroke***. We experimented with the initial positions and relative density of the swimmers for the all our animations. In Figure 10 and Figure 11 the relative density was set to 1. In the latter example, where we started the swimmer underwater, the swimmer came up entirely due to the hand and leg motions. Figure 13, 14 show ***crawl*** motion and ***backstroke*** motion respectively. We also show the ***butterfly stroke*** in Figure 12.

Our approach is implemented in $C++$ and uses Open Dynamics Engine (ODE) to simulate the rigid body dynamics. The final frames were rendered using POVray. One second of simulation took approximately 30 to 40 minutes on a dual pentium 2.5 GHz machine.

6 Summary & Future Work

We have presented an algorithm for creating realistic animation of characters interacting with fluids, from motion capture data. The ability for articulated bodies to interact with fluids also allows interesting simulations of simple water creatures, by writing simple controllers. We can easily extend this framework to key-frame data in which the input joint angles will be given by key framing, instead of motion capture.

Currently the algorithm only allows *global* changes in the character simulation, due to interaction with fluids, i.e., , only the global position/rotation of the character changes but the relative orientations of different bodies w.r.t each other remains same. We would wish to explore the possibility of changing the relative orientations as well, which would allow creating more interesting simulations. For example, if we have a motion capture data of a person walking and we want to simulate it walking in knee deep water, it would be interesting if we could make the simulated person take higher steps like an actual human would do.

References

BECKER, L. E., KOEHLER, S. A., AND STONE, H. A. 2003. On self-propulsion of micro-machines at low reynolds number: Purcell's three-link swimmer. *Journal of fluid mechanics 2*.

BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 97–104.

CARLSON, M., MUCHA, P. J., AND TURK, G. 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. 23*, 3, 377–384.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 736–744.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 251–260.

FEATHERSTONE, R. 1987. *Robot Dynamics Algorithm*. Kluwer Academic Publishers, Norwell, MA, USA. Manufactured By-Kluwer Academic Publishers.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models Image Process. 58*, 5, 471–483.

GÉNEVAUX, O., HABIBI, A., AND DISCHLER, J.-M. 2003. Simulating fluid-solid interaction. In *Graphics Interface*, A K Peters, CIPS, Canadian Human-Computer Commnication Society, 31–38. ISBN 1-56881-207-8, ISSN 0713-5424.

GETTELFINGER, B., AND CUSSLER, E. L. 2004. Will humans swim faster or slower in syrup? *AIChE Journal 50*, 2646–2647.

GLEICHER, M. 1998. Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 33–42.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. *ACM Trans. Graph. 24*, 3, 973–981.

HALLIDAY, D., AND RESNICK, R. 1978. *Physics : parts I and II*. New York : Wiley.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 71–78.

KIM, B., LIU, Y., LLAMAS, I., AND ROSSIGNAC, J. 2005. Flowfixer: Using bfecc for fluid simulation. In *Eurographics Workshop on Natural Phenomena*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 473–482.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 491–500.

O'BRIEN, J. F., ZORDAN, V. B., AND HODGINS, J. K. 2000. Combining active and passive simulations for secondary motion. *IEEE Computer Graphics & Applications 20*, 4.

OSHITA, M., AND MAKINOUCHI, A. 2001. A dynamic motion control technique for human-like articulated figures. *Comput. Graph. Forum 20*, 3.

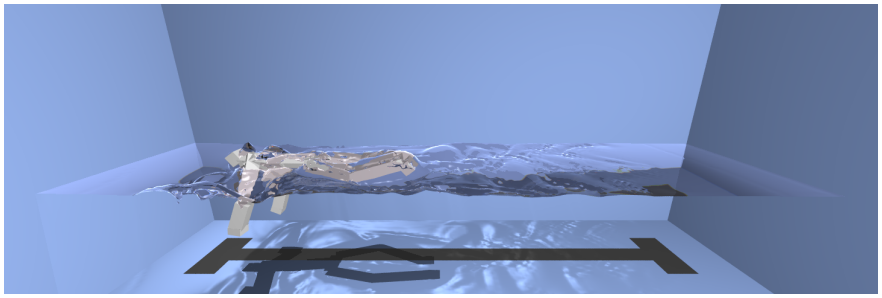

Figure 10: ***Backstroke***.See submitted videos.

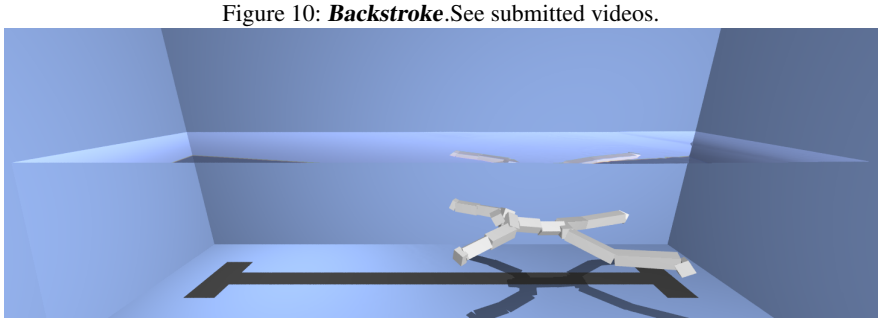

Figure 11: ***Backstroke***.See submitted videos.

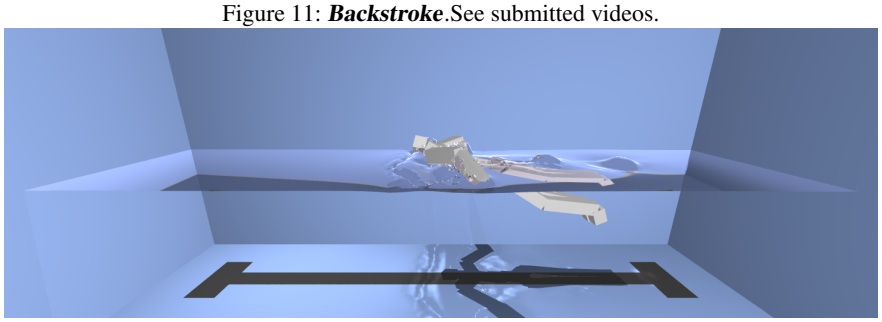

Figure 12: ***Butterfly***.See submitted videos.

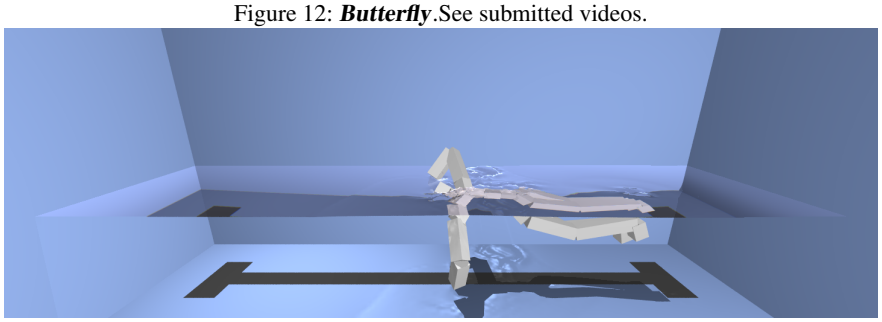

Figure 13: ***crawl***.See submitted videos.

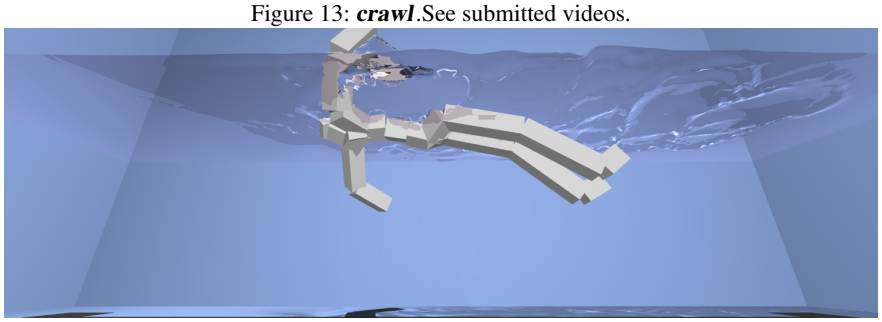

Figure 14: ***Backstroke***. See submitted videos.

PURCELL, E. 1976. Life at low reynolds number. *American Journal of Physics 45*, 3–11.

REDON, S., GALOPPO, N., AND LIN, M. C. 2005. Adaptive dynamics of articulated bodies. *ACM Trans. Graph. 24*, 3, 936–945.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 245–254.

STAM, J. 1999. Stable fluids. *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 121–128.

TAKAHASHI, T., HEIHACHI, U., AND KUNIMATSU, A. 2002. The simulation of fluid-rigid body interaction. *SIGGRAPH 2002: Setches & Applications*, 266.

TAKAHASHI, T., FUJII, H., KUNIMATSU, A., HIWADA, K., SAITO, T., TANAKA, K., AND UEKI, H. 2003. Realistic animation of fluid with splash and foam. *Comput. Graph. Forum 22*, 3, 391–400.

TU, X., AND TERZOPOULOS, D. 1994. Artificial fishes: physics, locomotion, perception, behavior. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 43–50.

WU, J.-C., AND POPOVIC, Z. 2003. Realistic modeling of bird flight animations. *ACM Trans. Graph. 22*, 3, 888–895.

YANG, P.-F., LASZLO, J., AND SINGH, K. 2004. Layered dynamic control for interactive character swimming. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 39–47.

YNGVE, G. D., O'BRIEN, J. F., AND HODGINS, J. K. 2000. Animating explosions. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 29–36.

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 89–96.