Fluid Simulation with Articulated Bodies

Nipun Kwatra
Stanford University

Chris Wojtan
Georgia Institute of Technology

Mark Carlson
DreamWorks Animation

Irfan Essa
Georgia Institute of Technology

Peter J. Mucha
UNC Chapel Hill

Greg Turk
Georgia Institute of Technology

Abstract—We present an algorithm for creating realistic animations of characters that are swimming through fluids. Our approach combines dynamic simulation with data-driven kinematic motions (motion capture data) to produce realistic animation in a fluid. The interaction of the articulated body with the fluid is performed by incorporating joint constraints with rigid animation and by extending a solid/fluid coupling method to handle articulated chains. Our solver takes as input the current state of the simulation and calculates the angular and linear accelerations of the connected bodies needed to match a particular motion sequence for the articulated body. These accelerations are used to estimate the forces and torques that are then applied to each joint. Based on this approach, we demonstrate simulated swimming results for a variety of different strokes, including crawl, backstroke, breaststroke and butterfly. 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.

Index Terms—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 [1], [2], [3], [4] for a review). Character animation either remains in the hands of talented animators with good tools for key-framing or is performed using data-driven animation. Motion capture technologies are now in wide use to capture data of performances to animate characters. A major benefit of motion capture is that it provides many of the details and nuances of live motion. The usability of motion capture for animation has been further enhanced by some practical approaches for the adaptation and reuse of already captured data (see [5] [6], [7], [8]). 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 interesting to take a kinematic motion of a fish and put it in simulated water to animate the effects that the fish and the water have on each other. 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, affecting both the character and the fluid.

We propose a method for simulating the interaction between an articulated character and a simulated fluid. 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 they have on each 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 [9]. 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 1. The 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 the articulated figure back into step with the data. Thus, the swimmer closely follows the motion capture data in terms of the relative joint angles, but at the same time is given a global translation and rotational motion that represents the net force and torque from the fluid to the swimmer. This completes the two-way coupling of the swimmer with the fluid environment.

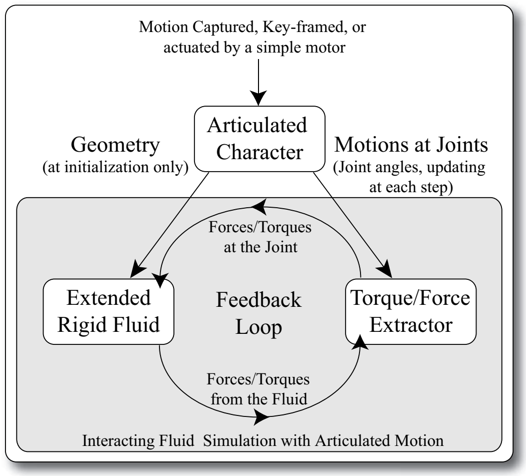

Motion Captured, Key-framed, or actuated by a simple motor
Articulated Character
Geometry (at initialization only)
Motions at Joints (Joint angles, updating at each step)
Forces/Torques at the Joint
Extended Rigid Fluid
Feedback Loop
Torque/Force Extractor
Forces/Torques from the Fluid
Interacting Fluid Simulation with Articulated Motion

Fig. 1. Flow diagram of our approach.

Our work broadens the range of objects that can interact with simulated fluids. In prior work, examples have been created in which a ball or feet have caused splashes in fluids [1], [2], [10]. In such cases, the trajectory of the ball is not affected by the fluid. Examples have also been created that simulate buoyant objects that are floating on the surface of the water [11]. In these situations 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 [3], [9], [12]–[17]. We use a full two-way solid-fluid coupling enabling us to demonstrate effects that were 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. Our articulated structures impart complex forces on the fluid and in return the fluid propels the structure in the ways that we expect. Using our approach, we are able to dynamically simulate characters swimming in water. Using a simple controller, we also demonstrate the validity of a result due to Purcell about simple swimmers at low Reynolds numbers [18].

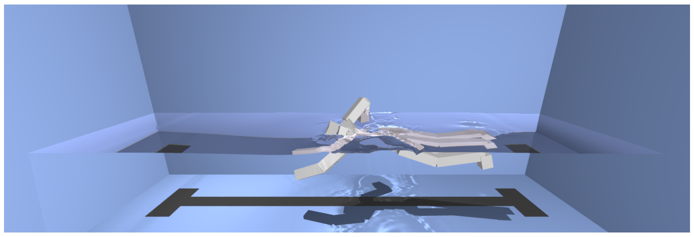


Fig. 2. A simulated swimmer that is driven by motion capture data. The forward motion of the swimmer is due to the interactions between the figure and the simulated fluid.

2 Related Work

Most of the work in motion capture research is focused on reusing already captured data either to synthesize new animation by copying pieces from a database and reassembling them, or to generate 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 [19] use simulation to allow a character to respond to a mass being dropped on its back. Zordan and Hodgins [20] create simulations from motion capture data that are able to hit and react to collisions. This work is closest to ours in spirit. They choose to 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, since we are interested in interacting with fluids, our articulated figure is continuously interacting with the physical environment at all times.

Recently Yang *et al.* [21] developed a layered strategy to control swimming motions. In their approach, they employ a simple viscous drag model and do not perform a full fluid simulation. They specifically state that "A full CFD [computational fluid dynamics] with unsteady flow is, however, necessary to truly capture the motion of swimming characters." [21]. [22] demonstrate coupling of the fluid with flexible bodies, with details to be published, including results for simulating free swimming motion of fish in a viscous fluid.

Previous work on characters that interact with fluids use simple models for calculating forces between fluid and solids. Tu and Terzopoulos [23] 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 [24] use a simple model of aerodynamics to create 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.* [25] simulate destructive explosions with two-way coupling of breakable rigid bodies and compressible high speed fluid. Génevaux *et al.* [12] implement two way coupling between incompressible fluid and deformable solids modeled by mass/spring systems. Takahashi *et al.* [26], [27] 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.* [13] use a similar object velocity/fluid pressure coupling to animate the interaction of incompressible fluids with zero volume deformable and rigid shells. [3], [15]–[17] solve simultaneosly for the fluid pressure and solid velocity to obtain a fully implicit two-way solid-fluid coupling. We chose to use the Rigid Fluid method from Carlson *et al.* [9] to model our rigid bodies interacting with incompressible fluid. This method affords an easy translation of forces between the simulations and was straightforward to implement. The method scales linearly, and it adds very little over-head com-

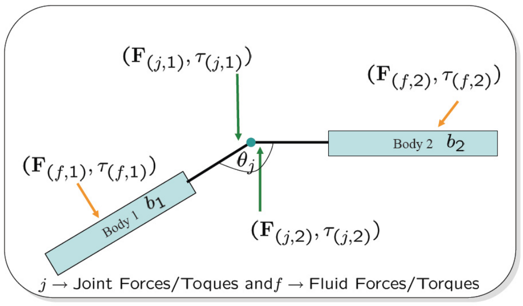

$(\mathbf{F}_{(j,1)}, \tau_{(j,1)})$
$(\mathbf{F}_{(f,2)}, \tau_{(f,2)})$
$(\mathbf{F}_{(f,1)}, \tau_{(f,1)})$
Body 2 b_2
θ_j
Body 1 b_1
$(\mathbf{F}_{(j,2)}, \tau_{(j,2)})$
$j \rightarrow$ Joint Forces/Toques and $f \rightarrow$ Fluid Forces/Torques

Fig. 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.

pared to the cost of the fluid simulation. We recognize that other coupling techniques for solid/fluid interaction such as [3] may also be appropriate for our goals. We believe that our constraint-based solver will work equally well with the other approaches listed above.

Much attention has been given to modeling the dynamics of articulated figures. Methods for defining controllers for complex behaviors [28] to learning simple controllers [29] have been proposed. However, most of these methods are more suitable for forward dynamic simulation [30] improving on efforts in dynamic robotic manipulation [31].

3 Force/Torque Extraction

Our goal is to create a physical system that is modeled on the articulated data (and its 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 (measured counter-clockwise with respect to the x-axis). Details regarding the extension to the 3D case, which requires dealing with quaternions, are described in the next section (Section 3.2).

We model the motion data by constraining the angles of the simulated joints to closely follow the corresponding angles in the motion 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:

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

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. Thus in our explicit forward Euler approximation, the forces and torques applied now will affect the joint angle θ_j two time steps later:

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

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

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

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

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) Equations due to Newton's third law. 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:

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

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* [32][1] of Newton's third law.

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

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

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.

Note that if the current state of the articulated body differs considerably from the desired state, then the controller might cause very large torques. This could be easily fixed by clamping the forces/torques applied to some threshold. Since all of our swimmer examples start from a good state and because our controller is able to follow the mocap trajectory closely, we did not need to perform any such clamping in our simulations.

3.2 Extending Constraint-based Solver to 3D

Unlike the 2D case, the orientation of a body cannot be represented by a single angle in 3D, 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. Here we assume a basic knowledge of quaternion algebra (see, e.g., [33]).

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 $q_1(t)$. Recall that the inverse of a unit quaternion is equivalent to its conjugate.

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., $\omega^q = (0, \omega)$. 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 2D 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 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

To make the 3D simulation 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 ($\hat{q}_j(t + 2\Delta t)$):

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

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

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 solve 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.

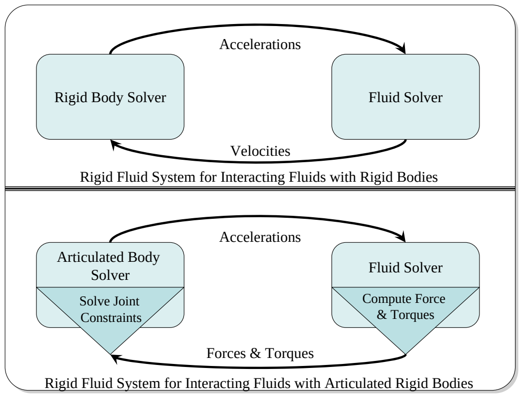


Fig. 4. A simple schematic highlighting the Fluid solver that works with rigid bodies and its extension to work with articulated rigid bodies.

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 that 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 in order 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 additional accelerations created on each body over the time step. Similarly the additional 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 [34] and pressure projection. We have also extended this to use a modification of the standard semi-Lagrangian approach that is known as Back and Forth Error Compensation and Correction (BFECC) [35], which is second-order accurate.
   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 forces such as collisions that act 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 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 that are needed for this purpose.

Note that the stability characteristics of our simulations with this coupling method are essentially the same as that of the underlying rigid fluid method, and we did not encounter any instability issues while running our simulations.

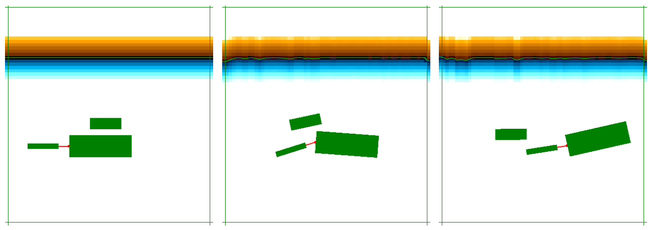


Fig. 5. A freely floating body alongside a ***flipper*** in low viscosity fluid

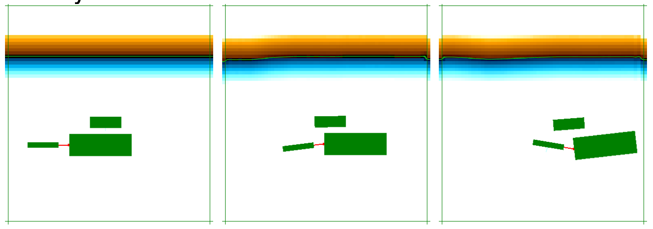


Fig. 6. A freely floating body alongside a ***flipper*** in high viscosity fluid

5 Results

We have undertaken two kinds of simulations using our two-way coupled simulation system. First, we have simulated simple water creatures that are actuated by hand-written controllers. Second, we used motion capture of subjects who were trying to mimic swimming actions in air and we have put the resulting articulated models into a simulated water environment. We describe each of these results in this section.

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 show a simple ***flipper*** in which a large body is pushed forward by a smaller 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 its 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. The effects of the fluid on a swimmer's global orientation can be seen when the flipper reaches the end of the fluid domain and begins turning upwards.

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 these desired configurations. Figure 8 shows frames from an animation of the swimmer. It requires subtle analysis to determine which direction such a simple swimmer will move; as is predicted in [36], 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 [37] 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.

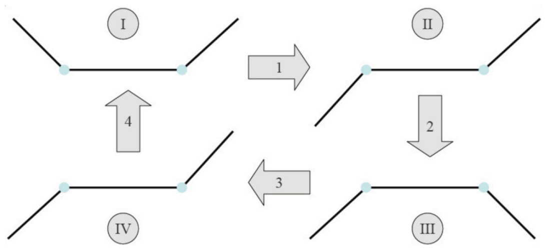


Fig. 7. The configuration cycle followed by a Purcell swimmer [18].

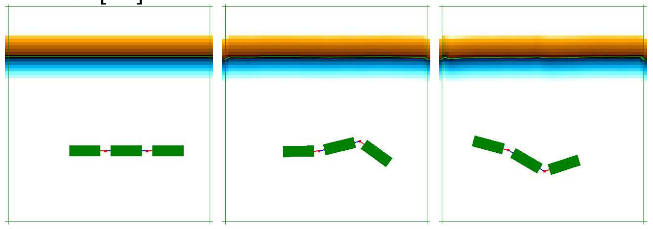


Fig. 8. A ***Purcell swimmer*** moving to the left.

5.2 Mocap Swimming

Now we will describe 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 these swimming strokes are very hard to mocap. Figure 14 shows three different ways our subjects were asked to perform the required motions. These are hard motions to perform, as most of the real dynamics are only possible when one is immersed in water, and moreover, all of these capture setups are extremely cumbersome. Nonetheless, we were able to capture several instances of ***crawl***, ***breaststroke***, and ***backstroke***. A correct looking butterfly stroke was just not possible, as none of our subjects could move their bodies in the manner required for that stroke while being 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.

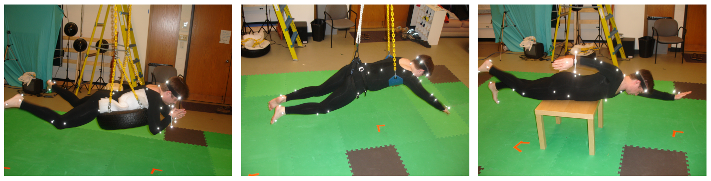


Fig. 14. Pictures of a motion capture session. Capturing mocap of a user who is performing motions to mimic swimming is not easy. Three different configurations are shown here.

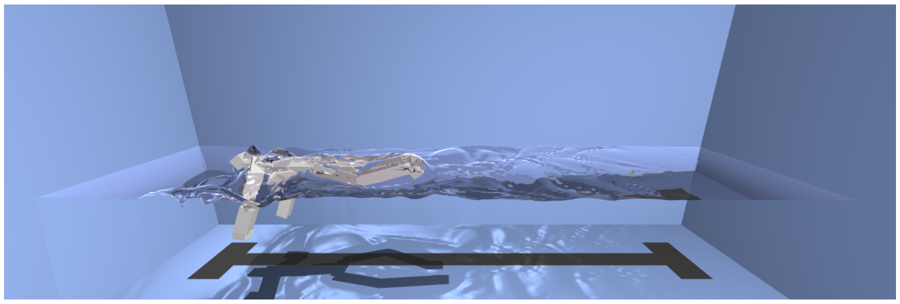


Fig. 9. ***Breaststroke***

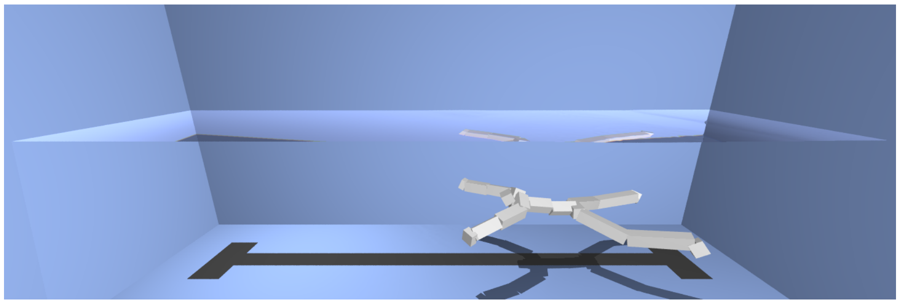


Fig. 10. ***Under-water Breaststroke***

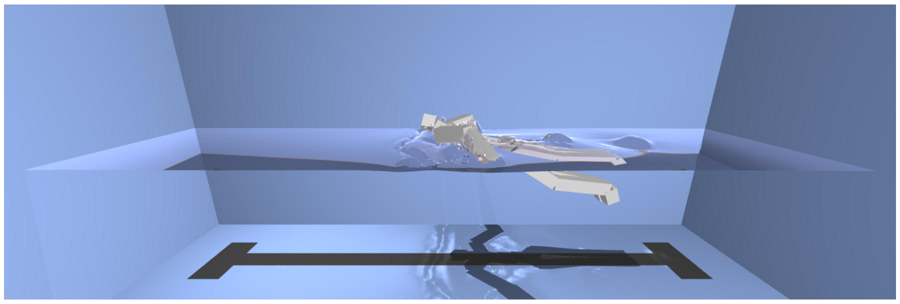


Fig. 11. ***Butterfly***

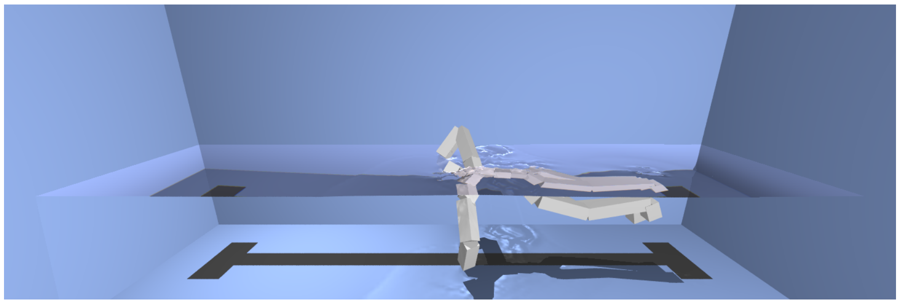


Fig. 12. ***Crawl***

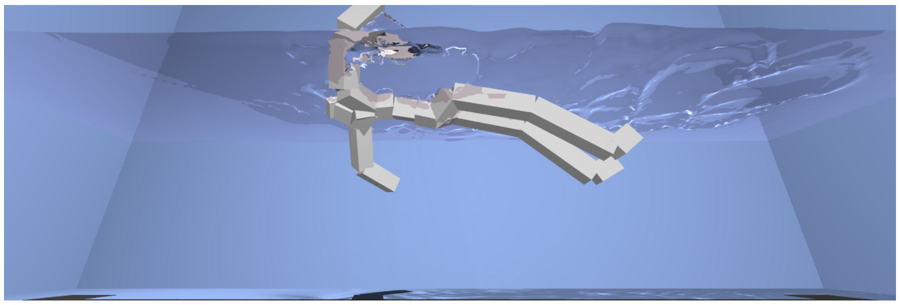


Fig. 13. ***Backstroke***

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).

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 would also be possible, but is left as future work. 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 9 shows a view of the swimmer performing a ***breaststroke***. We experimented with the initial positions and relative density of the swimmers for all our animations. In Figure 9 and Figure 10 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. The breaststroke swimmer demonstrates that the forward motion of the swimmer is not pre-scripted. The swimmer rapidly thrusts forward when the swimmer arms are moved backwards, but slows down a little due to drag when the arms move forward. Figure 12, 13 show ***crawl*** motion and ***backstroke*** motion respectively. We also show the ***butterfly stroke*** in Figure 11.

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. See the project website at http://www.cc.gatech.edu/cpl/projects/swimmer/ for videos and other related material.

6 Limitations

Our method allows us to simulate two-way coupling between articulated figures and fluids, yet as it stands there are some limitations to our approach. One limitation of our method is that the quality of swimming animation is directly related to the quality of motion capture data. These motions are hard to perform in air, as most of the real dynamics are only possible when one is immersed in water. Moreover, the data capture setup is extremely cumbersome. Improving motion capture for swimming is a possible future direction for research in motion capture.

Another current limitation of our system is that the controller samples the motion capture data only at the rate of the time steps of the simulation. Since we would like to take large time steps to keep the simulation times reasonable, we could miss high frequency motions in the captured data. This could be easily fixed by clamping the time step size by the motion capture rate, albeit at the cost of higher simulation time.

7 Summary & Future Work

We have presented an algorithm for creating realistic animation of characters that are interacting with fluids. This method allows us to create simulated swimmers based on 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 are given by key framing, instead of motion capture.

Our work has concentrated on the coupling between the articulated character and the fluid. Though the relative joint angles of our character closely follow the motion capture data, our method imparts a global translation and rotation to the character that represents the net force and torque form the fluid to the character. This is very important to get correct looking interactions of the character with fluid. For example, during the breaststroke, the simulated swimmer gets a sharp forward push when its arms move backward, while it slows down a little due to drag when the arms move forward. Similarly, our simulator allows us to predict the direction of motion of simple characters like the Purcell swimmer. The method can also be used to simulate phenomena like characters swinging against waves and one swimmer affected by the wake of another.

One exciting avenue for future work is to create controllers that alter behavior based on the character's motion through the water. This would allow a simulated charater to modify its motion to alter its swimming direction by varying the form of its stroke. If we had motion capture data of a person walking and we wanted to simulate 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. Reactive controllers would also allow other complex behaviours such as attack and escape motions for simulated aquatic creatures. Another extension of the current work would be to add skinning which would enhance both simulation accuracy and visual fidelity. Finally we would like to do an analysis of the motion of simulated swimmers against real swimmers in water. However this would require better motion capture of swimming (probably inside water), which is a good area for future study.

8 Acknowledgments

We would like to thank the members of Georgia Tech's Computational Perception Laboratory and Graphics Group for help and support. A special thanks to Vivek Kwatra for useful discussions and suggestions. Thanks to Matt Flagg for helping with motion capture and Sumit Jain for helping with video editing. Thanks also to Jessica Hodgins and her group at Carnegie Mellon University, specifically Moshe Mahler and Justin Macey, for providing motion capture data and underwater swimming video sequences.

This work was supported in part by NSF grants CCF-0625264, CCF-0811485 and CCF-0625190 and funding from Microsoft Research.

References

[1] D. Enright, S. Marschner, and R. Fedkiw, "Animation and rendering of complex water surfaces," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2002, pp. 736–744.

[2] N. Foster and R. Fedkiw, "Practical animation of liquids," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2001, pp. 23–30.

[3] A. Robinson-Mosher, T. Shinar, J. Gretarsson, J. Su, and R. Fedkiw, "Two-way coupling of fluids to rigid and deformable solids and shells," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*. New York, NY, USA: ACM, 2008, pp. 1–9.

[4] R. Bridson and M. Müller-Fischer, "Fluid simulation: Siggraph 2007 course notes," in *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*. New York, NY, USA: ACM, 2007, pp. 1–81.

[5] A. Bruderlin and L. Williams, "Motion signal processing," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1995, pp. 97–104.

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

[7] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2002, pp. 473–482.

[8] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," in *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2002, pp. 491–500.

[9] M. Carlson, P. J. Mucha, and G. Turk, "Rigid fluid: animating the interplay between rigid bodies and fluid," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 377–384, 2004.

[10] J. F. O'Brien, V. B. Zordan, and J. K. Hodgins, "Combining active and passive simulations for secondary motion," *IEEE Computer Graphics & Applications*, vol. 20, no. 4, 2000.

[11] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graph. Models Image Process.*, vol. 58, no. 5, pp. 471–483, 1996.

[12] O. Génevaux, A. Habibi, and J.-M. Dischler, "Simulating fluid-solid interaction," in *Graphics Interface*, CIPS, Canadian Human-Computer Commnication Society. A K Peters, June 2003, pp. 31–38, iSBN 1-56881-207-8, ISSN 0713-5424.

[13] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw, "Coupling water and smoke to thin deformable and rigid shells," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 973–981, 2005.

[14] F. Losasso, G. Irving, E. Guendelman, and R. Fedkiw, "Melting and burning solids into liquids and gases," *IEEE Trans. on Vis. and Comput. Graph.*, vol. 12, no. 3, pp. 343–352, 2006.

[15] B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O'Brien, "Fluid animation with dynamic meshes," in *ACM Trans. Graph. (SIGGRAPH Proc.)*, vol. 25, no. 3, Aug. 2006, pp. 820–825.

[16] N. Chentanez, T. G. Goktekin, B. Feldman, and J. O'Brien, "Simultaneous coupling of fluids and deformable bodies," in *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2006, pp. 325–333.

[17] C. Batty, F. Bertails, and R. Bridson, "A fast variational framework for accurate solid-fluid coupling," *ACM Trans. Graph. (SIGGRAPH Proc.)*, vol. 26, no. 3, p. 100, 2007.

[18] E. Purcell, "Life at low reynolds number," *American Journal of Physics*, vol. 45, pp. 3–11, 1976.

[19] M. Oshita and A. Makinouchi, "A dynamic motion control technique for human-like articulated figures." *Comput. Graph. Forum*, vol. 20, no. 3, 2001.

[20] V. B. Zordan and J. K. Hodgins, "Motion capture-driven simulations that hit and react," *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 89–96, 2002.

[21] P.-F. Yang, J. Laszlo, and K. Singh, "Layered dynamic control for interactive character swimming," in *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*. New York, NY, USA: ACM Press, 2004, pp. 39–47.

[22] M. A. MacIver, N. A. Patankar, B. Gooch, V. Oza, S. Lee, and O. Curet, "Numerical simulation of freely swimming fish."

[23] X. Tu and D. Terzopoulos, "Artificial fishes: physics, locomotion, perception, behavior," in *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1994, pp. 43–50.

[24] J.-C. Wu and Z. Popovic, "Realistic modeling of bird flight animations," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 888–895, 2003.

[25] G. D. Yngve, J. F. O'Brien, and J. K. Hodgins, "Animating explosions," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 29–36.

[26] T. Takahashi, U. Heihachi, and A. Kunimatsu, "The simulation of fluid-rigid body interaction," *SIGGRAPH 2002: Setches & Applications*, p. 266, 2002.

[27] T. Takahashi, H. Fujii, A. Kunimatsu, K. Hiwada, T. Saito, K. Tanaka, and H. Ueki, "Realistic animation of fluid with splash and foam." *Comput. Graph. Forum*, vol. 22, no. 3, pp. 391–400, 2003.

[28] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien, "Animating human athletics," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1995, pp. 71–78.

[29] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2001, pp. 251–260.

[30] S. Redon, N. Galoppo, and M. C. Lin, "Adaptive dynamics of articulated bodies," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 936–945, 2005.

[31] R. Featherstone, *Robot Dynamics Algorithm*. Norwell, MA, USA: Kluwer Academic Publishers, 1987, manufactured By-Kluwer Academic Publishers.

[32] D. Halliday and R. Resnick, *Physics : parts I and II*. New York : Wiley, 1978.

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

[34] J. Stam, "Stable fluids," *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

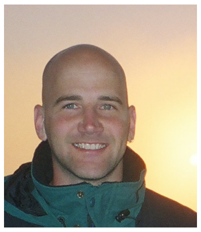
[35] B. Kim, Y. Liu, I. Llamas, and J. Rossignac, "Flowfixer: Using bfecc for fluid simulation," in *Eurographics Workshop on Natural Phenomena*, 2005.

[36] L. E. Becker, S. A. Koehler, and H. A. Stone, "On self-propulsion of micro-machines at low reynolds number: Purcell's three-link swimmer," *Journal of fluid mechanics*, vol. 2, 2003.

[37] B. Gettelfinger and E. L. Cussler, "Will humans swim faster or slower in syrup?" *AIChE Journal*, vol. 50, pp. 2646–2647, 2004.

Nipun Kwatra is a PhD candidate at Stanford University, under the guidance of Prof. Ronald Fedkiw. He received a BTech in computer science from the Indian Institute of Technology, Delhi in 2004 and an MS from Georgia Institute of Technology in 2006. His research is focused on computer graphics and computational physics, particularly physically based simulation of natural phenomena.

Chris Wojtan is a PhD student in the College of Computing at the Georgia Institute of Technology. He received a BS in computer science from the University of Illinois in Urbana-Champaign in 2004 and received a National Science Foundation Graduate Fellowship in 2005. His interests include physics-based animation, numerical methods, and applied geometry.



Mark Carlson received a BS in computer science, with a minor in creative writing, from the University of Central Florida in 1997, and earned a PhD in computer science, with a minor in computational physics, from Georgia Institute of Technology, in 2004. Dr. Carlson wrote a fluid simulator for DNA Productions where he also worked as a visual effects artist for the movie ”Ant Bully.” He was a Sr. Software Developer at Walt Disney Animation Studios, and is currently in the FX R&D group at DreamWorks Animation.

Irfan Essa Irfan Essa is a Professor in the School of Interactive Computing (SIC) of the College of Computing (CoC), at Georgia Institute of Technology, in Atlanta, Georgia, USA. At GA Tech, he is primarily affiliated with two interdepartmental centers; the Robotics & Machine Intelligence (RIM@GT) Center and the GVU Center. Irfan Essa works in the areas of Computer Vision, Computer Graphics, Computational Perception, Robotics and Computer Animation, with potential impact on Video Analysis and Production (e.g., Computational Photography, Image-based Modeling and Rendering, etc.) Human Computer Interaction, and Artificial Intelligence research. He has published over a 150 scholarly articles in leading journals and conference venues on these topics. He has been awarded the NSF CAREER Award and within GA Tech, he has won the College of Computings Junior and Senior Research Faculty Awards, Outstanding Teacher Award, Institutes Educational Innovation Award, and the Deans Award. He joined GA Tech Faculty in 1996 after earning his MS (1990), Ph.D. (1994), and holding a research faculty position at the Massachusetts Institute of Technology (Media Lab) [1988-1996].

Peter J. Mucha received a Ph.D. in Applied and Computational Mathematics in 1998 from Princeton University. He was an Applied Mathematics Instructor at MIT for three years, followed by four years as an Assistant Professor in Mathematics at Georgia Institute of Technology. He is currently an Associate Professor at the University of North Carolina at Chapel Hill, where he is a member of the Department of Mathematics and the Institute for Advanced Materials, Nanoscience & Technology.

Greg Turk received a Ph.D. in computer science in 1992 from the University of North Carolina at Chapel Hill. He was a postdoctoral researcher at Stanford University for two years, followed by two years as a research scientist at UNC Chapel Hill. He is currently a professor at the Georgia Institute of Technology, where he is a member of the School of Interactive Computing and the Graphics, Visualization and Usability Center. His research interests include computer graphics, scientific visualization and computer vision. In 2008 he was the Technical Papers Chair for ACM SIGGRAPH. He is a member of the IEEE.