Creature Control in a Fluid Environment

Michael Lentine, Jón Tómas Grétarsson, Craig Schroeder, Avi Robinson-Mosher, Ronald Fedkiw

Abstract—In this paper, we propose a method designed to allow creatures to actively respond to a fluid environment. We explore various objective functions in order to determine ways to direct the behavior of our creatures. Our proposed method works in conjunction with generalized body forces as well as both one-way and two-way coupled fluid forces. As one might imagine, interesting behaviors can be derived from minimizing and maximizing both drag and lift as well as minimizing the effort that a creature's internal actuators exert. A major application for our work is the automatic specification of secondary motions, for example, certain joints can be animated while others are automatically solved for in order to satisfy the objective function.

Index Terms—Computer Graphics, Animation, Motion Control, Creatures, Fluids

1 Introduction

A Number of graphics researchers have created computer generated creatures. [1] simulated snakes and worms, [2] used genetic algorithms to create creatures that interact with their environment, [3] simulated fish using a simple hydrodynamics propulsion model (see also [4]), and [5] simulated a bird in flight with a simplified aerodynamics model. We also make note of the layered rigid deformable creatures simulated by [6] and [7].

One creature that many researchers have dealt with is the human being. Various authors have researched the interactions of humans with their environments, with the seminal motivating work being [8], which illustrated the need to mix animations of humans with environmental constraints in the context of sporting events. Whereas earlier work [9] had shown that minimizing a creature's internal effort gives viable animation, this space-time optimization was too expensive to be applied to humans, and so [10], [11] devised ways to use reduced models.

There are numerous ways of intermixing the ideas of animation and simulation. For example, many authors have considered hybridizing an animation cycle or state controller with the secondary passive effects a character should exhibit when hit by an object, another character or by an external force [12]–[17]. Different types of interactions between animation and a physical environment are illustrated by carrying a briefcase [18], pushing a heavy object on the ground [19], moving or shaking the ground underneath the character [20], etc. [21], [22] worked to separate the forces a character needs to apply to interact with the environment from those the character would use to obtain a goal or express a sense of style.

Whereas most authors treat forces as a sequence of simple contacts or collisions, our focus is on the interactions that occur when a creature is immersed in a fluid environment. This is motivated by swimming fish [3], flying birds [5], swimming humans [23], etc. These papers all use simplified models of the fluid to calculate lift drag propulsion and other properties. Our goal is to instead use the full Navier-Stokes equations and simulate the creature in a fully two-way coupled manner with its environment. [24] recently showed key benefits to using the full Navier-Stokes equations over simplified fluid models. However, they do not implement a controller that is able to react to the environment and instead use a predefined motion to drive the character joints. This means that if the fluid flow changes, the character is not able to react.

Using the full Navier-Stokes equations permits our technique to be used in interesting complex fluid flows (consider highly turbulent smoke, ocean waves, etc). If one tries to simulate or animate a creature without incorporating these flows and then place it into a complex, fully dynamic fluid environment, the result is a creature whose relative joint motions seem wildly out of place with the surrounding fluid. There are essentially two ways to handle this. The first is to have the creature kinematically one-way coupled to the fluid, which is unrealistic as it assumes the creature has infinite strength and ability to control its environment – for example, a swimmer would swim the same way in giant waves as it would in a still ocean. The alternative is to allow the fluid to somehow affect the animation, which can arbitrarily modify joint angles and motion in a way that removes a creature's ability to obtain its goals – for example, a bird tipped sideways by the flow has no way of recovering its balance. One could try to build an animation that is robust to the subsequent situation by using simplified and approximate fluid models. Unfortunately, the Navier-Stokes equations are so complex that one cannot predict all nonlinearities and even a simple vortex shedding upstream in the flow can produce forces that are impossible to predict using simplified fluid models. Note that one can use hand animation to achieve visibly pleasing results, but this takes a large amount of resources and can be done much more quickly if automated.

Several complications arise as a result of coupling together dynamic simulation of the Navier-Stokes fluid flow, PD control, rigid bodies, articulated bodies, de-

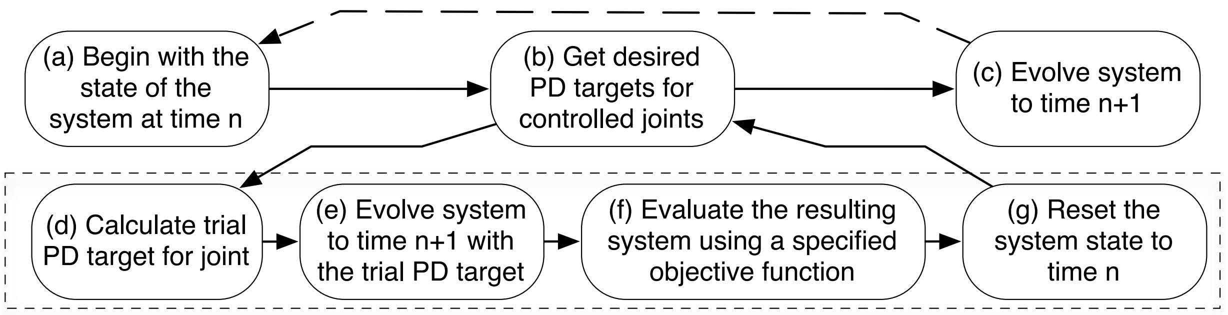
(a) Begin with the state of the system at time n
(b) Get desired PD targets for controlled joints
(c) Evolve system to time n+1
(d) Calculate trial PD target for joint
(e) Evolve system to time n+1 with the trial PD target
(f) Evaluate the resulting system using a specified objective function
(g) Reset the system state to time n

Fig. 1. A flow chart of our system. In a typical solid-fluid coupling system, only steps (a) and (c) are executed each time step. Our system adds step (b) (which applies (d-g) on each joint). Note that these steps can be handled in parallel for each joint.

formable bodies, contact and collisions. For example, PD controllers do not typically account for forces resulting from physical phenomena such as solid/fluid coupling or external forces. We address this by extending the post-stabilization projections of [7] and [22] to properly account for these phenomena. In addition, the implicit system of [25] does not produce velocities compatible with contact constraints, which we address by introducing projections that properly constrain the solid velocities. We also extend this coupled solve to account for kinematic (i.e. infinite mass) bodies.

We specify objective functions which can be used to minimize or maximize quantities such as drag and lift, as well as more complex functions such as the ability of a character to minimize the internal actuator force needed to fight the external environment, and we actuate the character's internal degrees of freedom based on a gradient descent on this objective function. Once this controller decides what the internal actuation should be, we evolve the full system forward in time, solving for all relevant forces and degrees of freedom. These include fluid dynamics forces arising from the Navier-Stokes equations; the rigid, articulated, and deformable degrees of freedom of the creature; internal actuations (torques); and coupling forces. This evolves the creature one step forward in time, and the process repeats, allowing the creature to locally optimize towards its objective.

Our paper is organized as follows: we propose a simple scalable methodology and show that the method provides reasonable results for simple problems with known solutions. Next we extend this method to take into account dynamic fluid forces and again demonstrate the viability of our method. In order to demonstrate the scalability of the method we then apply it to multiple joints of a human skeleton. Finally we apply our method to more complex fully two-way coupled dynamic examples and show that this gives visually pleasing results.

2 Creature Controller

Real creatures adapt their locomotion to best suit the environment in which they live. For creatures that must support their own weight, it is quite beneficial to move in ways that minimize the total energy or force required to remain upright. With this in mind, we treat the problem of motion control as an optimization problem where creatures move by seeking (locally) optimal solutions to objective functions.

The problem of nonlinear optimization has received significant attention (see for example [26]). Because our objective function depends on the current state of the creature and the environment with which it interacts, the optimum changes with time. Further, it is not appropriate to make large changes to the system in a single time step (due to limits on a creatures strength), which is what might happen if the local or global optimum of the objective function were simply computed and achieved at each step. For this reason, we choose to make small local changes based on the environment at the current time, and we use gradient descent. We illustrate our algorithm in Figure 1.

Our method locally optimizes a given objective function, with the intention that these objective functions, when constrained by the physical strength of the creature, give reasonable behaviors. Our simple, two-dimensional block examples demonstrate that when nothing else is happening in the vicinity of the block, an optimum solution is obtained; however, in general we do not expect (and do not desire) to reach an optimal solution. Even in the block example, it is creature strength which determines how quickly the optimum solution is reached. This is shown in Figure 2. Note that if we assume the creature has infinite strength we can

Simulation	Converged Frame
Navier-Stokes - Max Drag	56
Navier-Stokes - Min Drag	50
Navier-Stokes - Min Effort	30
Simple Fluid - Max Drag	42
Simple Fluid - Min Drag	68
Simple Fluid - Min Effort	51

Fig. 2. First frame at which a converged solution is reached.

perform multiple iterations of gradient descent to obtain the (locally) optimal configuration (with a static flow field) in one step.

2.1 Computing Derivatives

For a complex, dynamic, fully coupled system, derivatives with respect to a given degree of freedom are quite difficult if not impossible to compute analytically. This is a result of coupling with a fluid, as a local change in one degree of freedom changes the solution globally in a nonlinear way. This is further complicated by discontinuous, non-differentiable phenomena such as collisions and contacts. It is therefore desirable to compute the partial derivatives numerically, as this approach works regardless of complexity of the system. Note that if one does not use a Navier-Stokes fluid model, collisions, etc, it would be much simpler to analytically compute these derivatives. For robustness and accuracy, we choose a central differencing scheme to compute these derivatives, and our objective functions are differentiated with respect to the degrees of freedom at the controlled joints. For each degree of freedom, we first make a change in one direction, evaluate the objective function, then make the same change in the opposite direction and re-evaluate the objective function. The approximation for the partial derivative in that degree of freedom is taken to be the difference of the objective function evaluations divided by twice the magnitude of the change from the initial configuration in that degree of freedom. This process is illustrated in Figure 3 for a single degree of freedom. This process is inherently parallelizable as one can evaluate each partial derivative independently. Furthermore, if one wanted to reduce the cost of computing these partial derivatives, one could use a one sided differencing scheme, effectively reducing the computational expense by approximately a factor of two. However, the bias that this introduces made this approach impractical, and thus we used central differencing in our examples.

We use the partial derivatives to compute a search direction as defined by steepest decent in the objective function, but we still need to choose a distance to travel in that direction. There are many ways of doing this, one of the more robust being golden section search. For simple examples, this yields favorable results. However, the repeated evaluations of the objective function make it less suitable for more complex examples and we instead choose a fixed step size that reflects the strength of the creature. That is, the controller must decide what it will do in the next time step of the simulated creature, and the strength of the creature limits what it should accomplish in a unit of time equal to the size of the next time step. We note that this method achieves convergence over time if there is a steady state solution, as demonstrated in Figure 4. In the case where there is no steady state solution, we obtain convincing dynamic results as shown in Section 7. In order to speed up the examples, a creature can search even less frequently than it's strength allows. In these cases we only redetermine the direction once in a given number of time steps. Although this does work fine for some examples, it can result in the creature reacting rather slowly to the external environment.

Although this is a fairly simple way of computing the gradient, because of the complexity and dynamical nature of our system, we found this to be the most effictive way of computing the derivatives. One could apply machine learning to accelerate the process over time. We experimented with this and found that although this does improve the speed of some examples, the dynamical nature of the system quickly creates new configurations not covered by the space of the training set.

2.2 Proportional Derivative Control

Computing the numerical central difference approximations for our gradient descent optimization requires us to apply positional changes to our controlled degrees

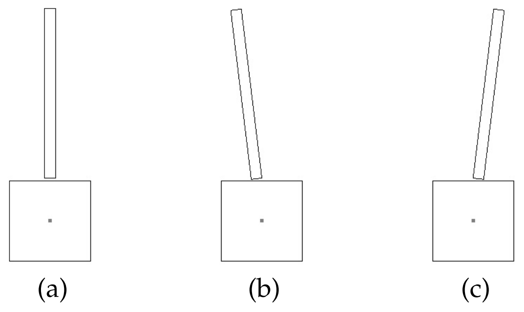
Fig. 3. (a) A single controlled joint connecting a static square to a block. (b) The resulting configuration of the block after taking a step to the left during our derivative calculation. (c) The analogous configuration to the right.

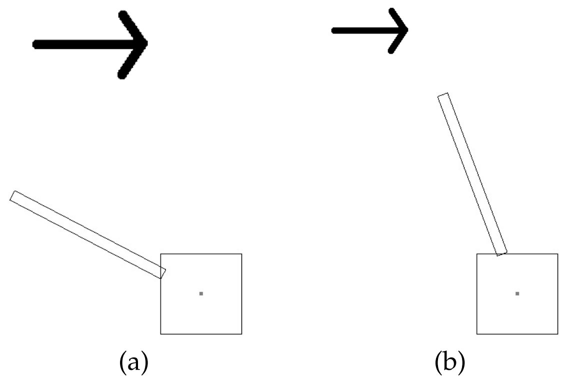
Fig. 4. (a) A single controlled joint connecting a static square to a block minimizing effort with gravity and a wind force to the right. (b) The same block minimizing effort with a weaker wind force.

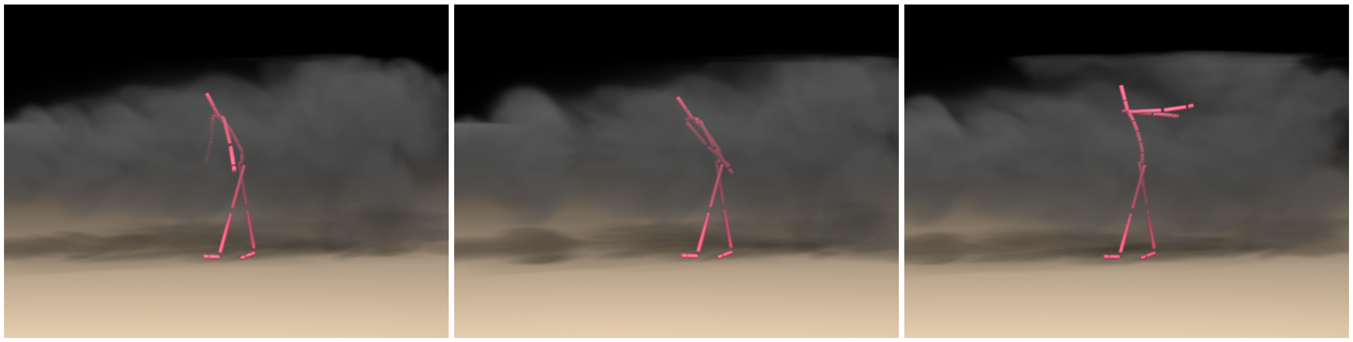

Fig. 5. A human undergoing both gravity and constant wind forces with kinematic legs and three controlled joints: one on each arm, and one on the lower back. (a) All joints minimize effort on the lower back. (b) The lower back joint minimizes its own effort while the arm joints minimize a weighted average of the effort on the lower back and their own effort. (c) The lower back joint minimizes its own effort while the arm joints minimize drag.

of freedom. There are many approaches that could be taken to achieve this. Many authors have addressed this problem with proportional derivative (PD) control. PD control has been shown to produce very desirable results, but it is known to have shortcomings in the presence of external forces. These retard the rate of progress towards the target and can even prevent the controller from achieving its target. For our application, this needs to be addressed so that we can achieve the desired perturbations in the position to compute gradients that are not subject to significant noise. We address issues related to outboard inertia tensors, gravity, fluid dynamics forces, and the resistant forces due to wrapping rigid bodies in deformable meshes, by utilizing the post-stabilization projections of [22] (in order to obtain the desired displacements when calculating our partial derivatives). This framework allows us to guarantee that the joint degree of freedom undergoes a position change irrespective of all external forces.

2.3 Optimizing Effort

The proportional derivative controller is used to evolve the system to a new configuration, and then we evaluate our objective function in this new configuration. Since [9], many researches have viewed effort minimization as a viable metric for determining realistic animation. Animation controllers apply the least effort when a configuration is reached where net external forces are as orthogonal as possible to these degrees of freedom. For example, Figure 4 shows the global minimum effort solution in a simple configuration, which our controller was able to find. Note that this shifts the external load off of the actuated degrees of freedom (which are weaker, for example supported by muscles) and onto dimensions that are not degrees of freedom (such as those that are bound, for example by compression of the joints). In particular, in Figure 4 at steady state the net force lies in the direction of the skinny block which is completely orthogonal to the actual degrees of freedom, and thus is resisted by the joint attachment itself.

We compute the minimum effort by finding the force to maintain our current velocities, thus canceling the acceleration and the net force. We again utilize the method of [22], solving the combined equations with the aim of using PD control to cancel all external forces in order to maintain the current velocities. The actuator forces that PD applies can then be interpreted as effort. In other words we calculate our objective function, the effort as

$$F_e = \mathbf{j}_\tau \tag{1}$$

where $\mathbf{j}_\tau$ are the angular impulses that PD applies to a joint.

3 Simplified Fluid Forces

As an initial step towards adding real fluid forces to our simulations, we begin with simplified fluid forces similar to those found in [3], [5], [23].

3.1 Simple Wind Forces

Given a velocity field, one can loosely approximate fluid pressure along the solid surface with a force that scales like the square of the relative velocity. Since we define the velocity on the entire grid, we can interpolate the fluid velocities to the center of a triangle. We then calculate the force acting on this triangle as

$$F_{di} = \text{sign}\left(\mathbf{u}_{rel}^T \mathbf{n}\right) A\rho \left(\mathbf{u}_{rel}^T \mathbf{n}\right)^2 \mathbf{n}, \tag{2}$$

where A is the triangle surface area, ρ is the fluid density, $\mathbf{u}_{rel}$ is the relative fluid velocity, and $\mathbf{n}$ is the inward-facing normal. F_{di} is then equally distributed to each of the triangle's nodes.

This simple force yields interesting results in simple examples, but cannot be used with a real fluid velocity field, as any consistent flow field requires that the fluid velocity won't flow through the solid (i.e. $\mathbf{u}_{rel}^T \mathbf{n} = 0$ at the surface of the solid).

3.2 Optimizing Simplified Drag

In a fluid environment, creatures worry less about supporting their own weight (thanks to buoyancy forces) and more about controlling how they are pushed around by the environment. Therefore, the creature should know about the lift and drag forces acting on it, giving it the opportunity to maximize or minimize drag.

Using simplified fluid forces, we can compute the wind drag as

$$F_d = \sum_i F_{di} \tag{3}$$

for all triangles i in the mesh and treat it as a body force. To illustrate how our optimization framework deals with this force, we submerge our single degree-of-freedom example into a constant rightward-moving velocity field, and compute the solution one obtains when minimizing and maximizing drag. According to the objective function, the controller maximizes and minimizes drag by increasing and decreasing the relative velocity. Thus, the controller will attempt to actuate the joints as fast as the creature's strength allows to push in a way that resists the flow direction when maximizing drag, and to move with the flow as best as possible when minimizing drag. Many creatures actually behave in a more complex way than this and optimize fluid forces in a given direction, for example up and down motion is extremely important when flying or when a creature is making its way towards the surface of a fluid, while lateral directions are important when attempting to achieve a locomotion target such as one that takes the creature closer to a food source. Thus, we can reformulate the objective function as

$$F_d = \sum_i \mathbf{w_i} \cdot F_{di} \tag{4}$$

such that the goal is to maximize or minimize drag in a given set of directions $\mathbf{w}$. As an example of this, suppose we had one direction we wished to maximize or minimize drag in, say left to right (i.e. $\mathbf{w} = \{1, 0\}$). The joint then actuates as fast as it can to the left to maximize drag, and as fast as it can to the right to minimize drag.

3.3 Relative Joint Velocities

The creature might also desire to achieve a steady state, such as a parachute-like shape for falling. Thus, another interesting formulation of the objective function projects out the relative velocities induced by joint actuation. We accomplish this by perturbing the object to the desired position for the partial derivative calculations and then setting the object velocities to zero before calculating F_d using equation (2). The results of this objective function are shown in Figure 6, where the drag is increased by maximizing the exposed surface area and decreased by minimizing the exposed surface area.

In our single degree of freedom example, the joint is rooted in a stationary object, and therefore the steady state calculation comes when all velocities and angular velocities of rigid bodies are zero. For a freely moving creature which is two-way coupled with its environment, setting all joint velocities to zero is nonsensical. Instead, the steady state calculation is obtained when the relative joint velocities are set to zero, thus freezing the shape of creature while still allowing it to be pushed around as it interacts with its environment. This is readily accomplished by treating the entire creature, or component of a creature (in the case that there are different dynamic components connected to a kinematic component) as a unified rigid body. That is, a number of rigid bodies that used to be moving relative to each are now treated as a single rigid body in a way that conserves both linear and angular momentum. We would like to stress that in our examples, we only used the steady state calculation in the simple two-dimensional cases, in order to verify the results of our technique.

4 Navier-Stokes Fluid Forces

There are many complex behaviors that arise in fluid flows, such as vortex shedding, viscosity, and turbulence. The simplifying assumptions made when deriving the forces discussed in the previous section ignore or discard these effects for the sake of computational efficiency. These effects however lead to the rise of more physically plausible motion as well as interesting secondary visual effects which greatly enhance the believability of a scene. With that in mind we implement a full fluid solver incorporating state-of-the art methods to capture realistic fluid effects and phenomena and pair the fluid with our creatures in a fully-coupled fashion, allowing them to respond to fluid forces.

4.1 Navier-Stokes Equations

We simulate fluid on a uniform grid using the inviscid, incompressible form of the Navier-Stokes equations, given by

$$\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \mathbf{f} \tag{5}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{6}$$

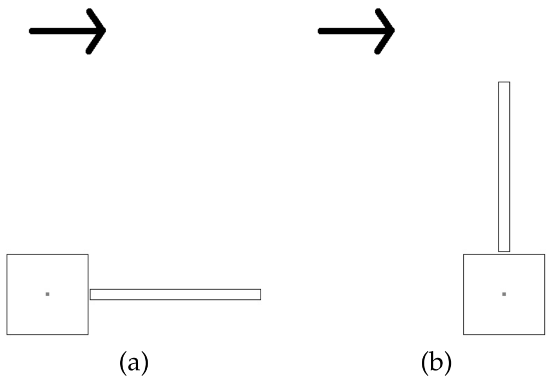

Fig. 6. (a) A single controlled joint connecting a static square to a block minimizing drag with a wind force to the right. (b) The same block maximizing drag.

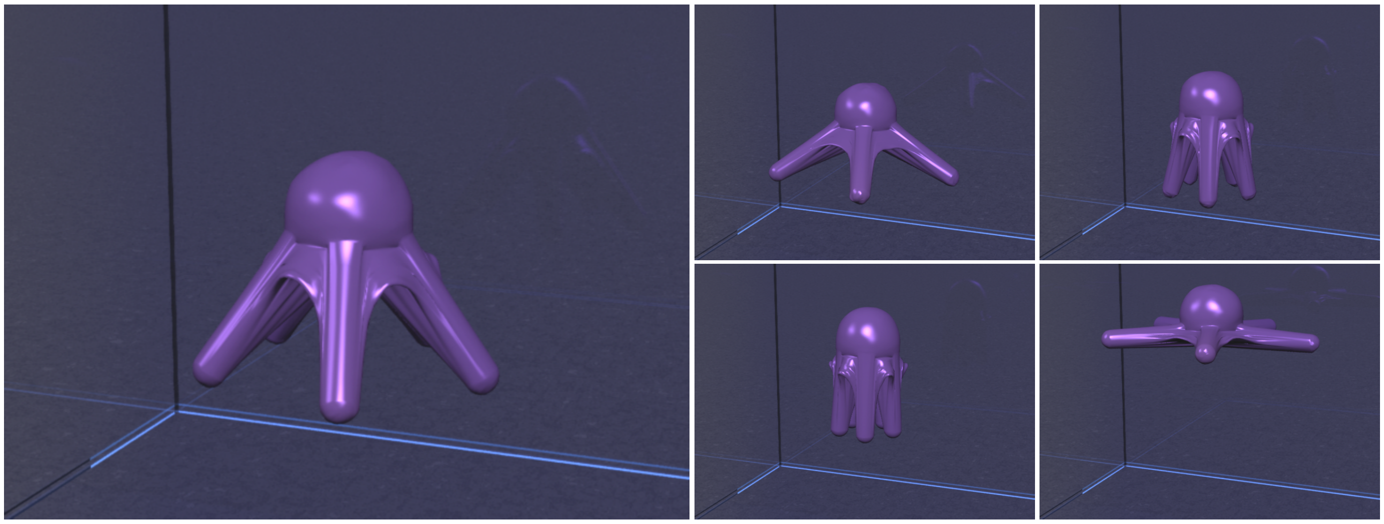

Fig. 7. A swimmer with five joints wrapped in flesh that propels itself through a fluid environment while alternating the minimization and maximization of drag.

where $\mathbf{u}$ is the velocity field of the fluid, ρ is the density of the fluid, $\mathbf{f}$ are any external forces (such as gravity), and p is the fluid pressure. We solve these equations by first calculating an intermediate velocity field $\mathbf{u}^\star$ using

$$\frac{\mathbf{u}^\star - \mathbf{u}^n}{\Delta t} + \mathbf{u}^n \cdot \nabla \mathbf{u}^n = \mathbf{f}$$

and subsequently adding in the pressure forces via

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^\star}{\Delta t} = -\frac{1}{\rho}\nabla p, \tag{7}$$

where the pressure is calculated by solving a Poisson equation of the form

$$\nabla \cdot \frac{1}{\rho}\nabla \hat{p} = \nabla \cdot \mathbf{u}^\star, \tag{8}$$

where $\hat{p} = p\Delta t$.

4.2 Two-way Coupling

The traditional method for coupling solids to fluids consists of prescribing fluid velocities to be equal to the solid velocities at the interface and then integrating pressure to get force boundary conditions on the solid. We use a method similar to [25] for the two-way coupling, although other methods such as [27]–[29] could also be applied since the part of this that is important to us is the ability to determine the coupling forces between the solid and fluid, and all of these schemes make this information readily available. As in [25], we solve the following symmetric indefinite system:

$$\begin{pmatrix} VG^T\frac{1}{\rho}G & VJ \\ VJ^T & -\tilde{M}_S + \Delta t D \end{pmatrix} \begin{pmatrix} \hat{p} \\ V_S^{n+1} \end{pmatrix} = \begin{pmatrix} VG^T\mathbf{u}^\star \\ -M_S V_S^\star - W^T M_F \mathbf{u}^\star \end{pmatrix}, \tag{9}$$

where G is the discretized gradient operator, V is the volume of a grid cell, V_S are the solid velocity degrees of freedom, M_S is the solid mass matrix, D is the damping matrix, and M_F is the fluid mass matrix. This matrix is of size $(m+n)\times(m+n)$, where there are m pressure degrees of freedom, and n solid velocity degrees of freedom. Note that the second equation is written as a change in momentum, meaning that $VJ^T\hat{p}$ is an impulse, and therefore VJ^Tp is the force the fluid applies to the solid during the two-way coupled solve. Although this is an indefinite system and can be rather slow, one can speed up these solves by using a SPD system such as the one proposed in [30].

4.3 Force Calculations

When we calculate partial derivatives for use in determining search directions for the controller, we actuate solid degrees of freedom to new positions and calculate forces in those new positions. Since the motions of the solid will affect the fluid, and vice versa, it is necessary to use a stable time integration when changing the configuration of the solids in order to evaluate these derivatives. Moreover, avoiding noise in the calculation typically means non-negligible changes in the configuration which in turn requires a few time steps to achieve in a stable and robust manner. After achieving this positional change, we calculate a pseudo-velocity equal to the difference in position over the difference in time and use this as the solids' average velocity for evaluating forces for the controller. The objective function (drag force) is then taken to be

$$F_d = \int_{\partial\Omega} p\mathbf{w} \cdot d\mathbf{A} \tag{10}$$

where $\mathbf{w}$ are the directions we want to optimize in and $\partial\Omega$ is the surface of the creature.

This produces good results but is difficult to validate due to the complexities of the resulting fluid. In order to

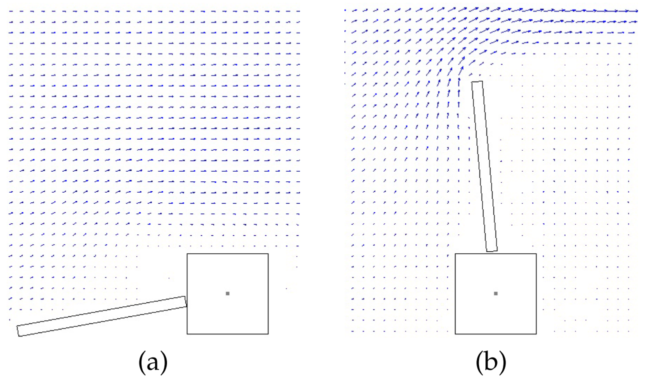

Fig. 8. (a) A single controlled joint connecting a static square to a block minimizing drag with a fluid force to the right. (b) The same block maximizing drag.

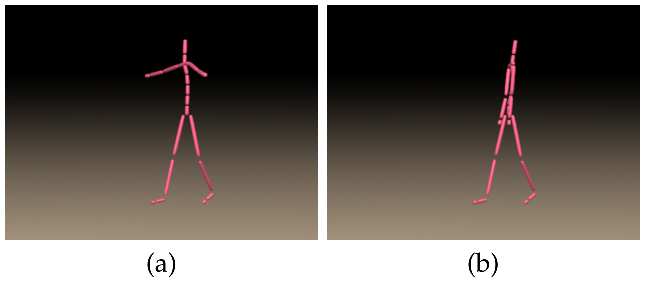

Fig. 9. A human undergoing gravity forces with kinematic legs and three controlled joints: one on each arm, and on the lower back. (a) All joints minimize effort on the lower back. (b) The lower back joint minimizes its own effort while the arm joints minimize a weighted average of the effort on the lower back and their own effort.

validate our method, we desire a simple flow field that has a known solution. In order to achieve this, we want to damp out the complexities such as vorticity which can be accomplished by simulating a few time steps of fluid evolution after we fix the relative orientation, allowing the flow field to respond to the creature's new configuration. This permits the disturbances created by the new configuration to be swept downstream. The resulting flow field is simple enough to verify visually, as shown in Figure 8. Note that although the results here are not perfectly horizontal or vertical they accurately minimize drag in the presence of a dynamical fluid flow. For example, in minimizing drag, it is beneficial to have a stagnant pocket of fluid below and in front of the dynamic component, as it reduces the drag felt on the object. We would also like to stress that we only damp out the velocity field in these simple cases so that we can more accurately evaluate our method. For our more complex examples this is not done.

5 Multiple Joints

Many creatures use more than one joint to accomplish a specific objective or set of objectives, and this needs to be incorporated into our objective function. Although one could explore the entire space of motions that can be achieved by looking at all combinations of joints, it is much more straightforward and standard to use independent calculations for each derivative. This makes our control algorithm linear in the number of degrees of freedom. Moreover, because these calculations are independent, they are trivially parallelizable making our algorithm for multiple degrees of freedom almost as wall clock time efficient as that for a single degree of freedom meaning that our algorithm runs almost as quickly when controlling n degrees of freedom (with $\geq n$ processors) as it does with 1 degree of freedom as seen in Figure 10.

We have also found that one can achieve interesting effects by allowing for a weighted average of objective functions at each degree of freedom. That is, instead of each joint attempting to minimize its own stress, certain joints can be actuated to minimize the stress on others. For example, in the case of a walking human, it might be more important to minimize the joint stress on the lower back than it is on the shoulder or elbow. Figure 9 (a) shows an example of a human walking under the influence of gravity where the lower back and arm joints are minimizing effort on the lower back. Figure 9 (b) shows a similar example except that the arms are minimizing their own effort while also working to minimize the effort of the lower back. Note that we can combine objectives with differing units such as an impulse (F_d) and an angular impulse (F_e). This is because we are taking a weighted average and can incorporate unit conversions into the weights.

When dealing with multiple joints, we have found that a straightforward calculation of partial derivatives via actuating a particular joint to its new position can lead to erroneous results. For example, consider the fact that bending the elbow while leaving the wrist loose will cause the hand to flex backward. This is undesirable, and instead we compute positional changes where the wrist joint is instead kept stiff. We accomplish this by rigidifying all bodies outward from the joint whose partial derivative is being calculated. This is calculated in the usual fashion by clustering these rigid bodies into a single rigid body with a unified mass, center of mass, velocity, and angular velocity.

Number of Joints	Time (Serial)	Time (Parallel)
1	$14s$	$14s$
2	$29s$	$15s$
3	$43s$	$15s$
4	$58s$	$15s$

Fig. 10. Timing information for a variation of our simple 2D block example using the full Navier-Stokes equations. All timings are given in seconds per frame. The parallel case used four processors.

6 Evolution

In order to accurately model a realistic environment we must model rigid bodies, deformable objects, and fluids in the presence of constrains such as PD targets and contacts. Our method evolves the solids forward in time using a modified Newmark method similar to the one proposed in [7] which allows us to accurately handle contact, collisions, and friction. For PD control, we integrate this with [22] and for solid-fluid coupling, we combine this with the scheme proposed in [25] (that provides the major high level steps which our evolution follows). We must also of course integrate our controller into the time integration scheme. The resulting full time integration scheme used for evolution proceeds as follows:

1: Calculate controller direction and magnitude as discussed in Section 2 and set PD targets from the results.

2: Use all non-pressure based and non-advection based fluid forces (i.e. external forces and viscosity) to advance the fluid velocity to time $t^{n+1/2}$, $\mathbf{u}_F^{n+1/2} = \mathbf{u}_F^n + (\Delta t/2)(\mathbf{f} + \nu\Delta\mathbf{u}_F^{n+1/2})$.

3: Evolve the solid positions forward in time. We do this by first integrating all explicit solid forces to time $t^{n+1/2}$. We then solve the coupled system to get $\boldsymbol{v}^{n+1/2}$. We then modify the resulting velocity with collisions. Next we apply post-stabilization to the velocity field followed by applying PD impulses to the velocity field. Note that the PD targets can either come from our controller or can be pre-specified without using our controller. Using this resulting velocity field we then update positions as $\boldsymbol{x}^{n+1} = \boldsymbol{x}^n + \Delta t \boldsymbol{v}^{n+1/2}$. Finally we resolve the resulting contacts using $\boldsymbol{v}^{n+1/2}$ and $\boldsymbol{x}^{n+1}$. To prevent competition between our PD controller and external fluid and deformable forces, we extend the post-stabilization projections of [7] to include our actuated degrees of freedom and apply the PD of [22] as boundary conditions for those degrees of freedom. Because projections are removed during the force application following the second CG solve in [7], we store and reapply PD boundary conditions after that step.

4: We prevent leaking by forcing the fluid to move with the solid effective velocity, calculated as $(\boldsymbol{x}^{n+1} - \boldsymbol{x}^n)/\Delta t$. A standard fluid Poisson equation is solved using the solid effective velocity mapped onto the Eulerian grid by W as Neumann boundary conditions to project $\mathbf{u}_F^n$. The resulting projected velocity is our leak-proof advection velocity $\mathbf{u}_{ADV}$.

5: We calculate the intermediate fluid velocity via $\frac{(\mathbf{u}_F^\star - \mathbf{u}_F^n)}{\Delta t} + \mathbf{u}_{ADV} \cdot \nabla \mathbf{u}_F^n = \mathbf{f} + \nu\Delta\mathbf{u}_F^\star$. Note that the advection velocity $\mathbf{u}_{ADV}$ is used to formulate the rays in the typical semi-Lagrangian scheme [31], but the advected quantity is the actual fluid velocity $\mathbf{u}_F^n$. $\mathbf{u}_{ADV}$ is also used to advect all other fluid scalar quantities to time t^{n+1}. Since this advection velocity exactly conforms to the effective velocity of the solid, it prevents leaking.

6: Evolve the solid velocities forward in time. We do this by first integrating all explicit solid forces to time t^{n+1}. We then solve the coupled system to obtain $\boldsymbol{v}^{n+1}$ and p. We modify the equations of [25], to properly account for kinematic bodies; we evolve them forward in time first, compute their solid effective velocity, and then add this into the right hand side of the coupled system. We also apply contact projections similar to those found in [7] during each iteration of the implicit solve to properly enforce contact constraints. We then apply post-stabilization using this new velocity and then update the velocity using PD targets. Following this, we apply contact using this new velocity field. To be safe we once again apply post-stabilization to make sure the joint constraints are not violated after PD targets and contacts are satisfied. Note that we do not necessarily need to apply post-stabilization but do so every time the velocity is changed because it is fairly cheap to do.

7: Using the fluid pressure from 6, project the intermediate fluid velocity $\mathbf{u}_F^\star$ to be divergence free, $\mathbf{u}_F^\star \rightarrow \mathbf{u}_F^{n+1}$.

The resulting velocity field is then used as an initial condition for the controller, for the next time step.

7 Examples

We generated a large number of examples with different numbers of joints, forces, and objective functions. As discussed earlier, we ran a number of single joint examples to demonstrate the viability of our algorithm (see Figures 3, 4, 6, and 8). We also ran a number of examples of a human with kinematically controlled legs optimizing various objective functions. The human contains a skeleton composed of twenty three rigid bodies and the kinematic motion is specified with motion capture data that was taken using a standard optical motion capture system. Figure 9 demonstrates examples that are run with only a gravity force. The joints in Figure 9 (a) optimize the effort (F_e) on the back joint by moving in such a way that makes the center of mass of the upper half of the body straight above the joint, meaning that the force gravity exerts on the bodies is cancelled by the compression force of the joint. In Figure 9 (b), the arms are also minimizing their own effort resulting in the arms hanging down, which cancels the gravity force with the arm joints while still minimizing the effort on the lower back by keeping the center of mass above that joint. Note that this example also demonstrates our algorithm in the presence of hard constraints. The collisions between the

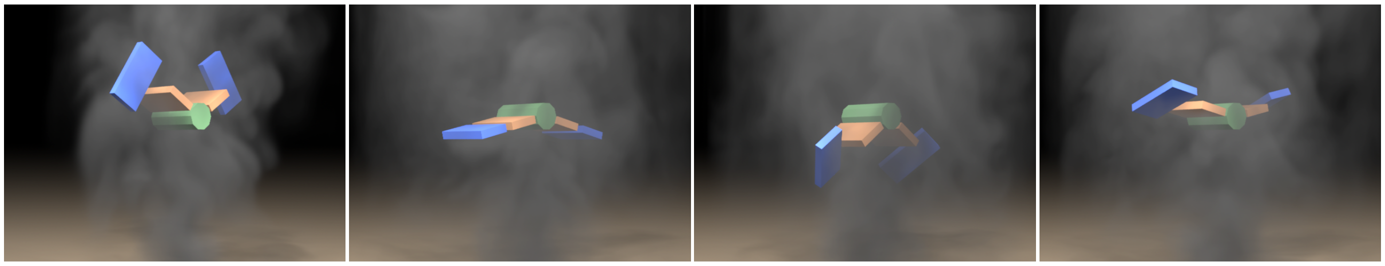

Fig. 11. A driven flyer with four joints (two on each wing) in a fluid environment. The inner most joints have pre-specified PD targets while the outer ones are fully controlled.

left leg and the left arm force the arm to go further forward than it would otherwise have. Our controller compensates for this by rotating the body to keep the arms down and the center of mass above the lower back joint.

This twisting motion minimizes the total effort on the system as any alternative would require more effort, such as is required to re-balance the center of mass over the lower back joint. For example, the human could lift its left arm to avoid contact with its leg, but this strains the lower back and shoulder. Other configurations, such as lifting both arms (in order to minimize effort on the lower back), also require additional total effort to maintain. We would like to stress that the human examples in our paper do not have the same constraints as a real human would (for example there is no concept of joint limits or pain) and are shown for illustrative purposes only.

Figure 5 shows examples with simplified fluid forces modeled as a constant wind field. Figure 5 (a) shows a human minimizing effort (F_e) on the back joint, and as a result, it leans into the wind force, canceling out the combined external forces of gravity, wind, and the back joint. Figure 5 (b) shows a human minimizing effort on the lower back as well as the arms. As expected, the human leans into the wind while positioning his arms in the direction of the combined external forces allowing for a cancellation of forces. With a wind force we can also optimize drag (F_d). Figure 5 (c) shows a human minimizing effort on the back joint and drag on the arms. The result is that it leans into the wind while holding its arms back. Note that we do not apply smoothing as a post-process for illustration, but one could easily do so if it was desired.

We rendered smoke for the purposes of visualization but stress that the actual fluid used was significantly heavier (similar to water) making the human lean forward more than visually expected in smoke. Conversely, we could have used lower density smoke but with a significantly higher velocity to get the same degree of lean but this would make the visualization of the flow field more difficult.

7.1 Driven Flyer

We have also used our algorithm with more complex and dynamic fluid flows. In Figure 12 we show the configuration for a simple creature with three rigid bodies whose wings are driven to flap up and down via PD control, with fluid that is injected from below. In order to enhance the appearance of this creature, we give it longer wings and add more joints that are actuated by the controller, which dynamically determines their optimal positions with regard to lift and drag. As the kinematically controlled inner wing joints push down, the outer joints governed by the controller maximize drag (F_d) on the body, and when the inner kinematic wing joints are moving upwards the controller works to minimize drag. Figure 11 shows the resulting animation for when these joints are controlled about the twisting axis. In a more realistic creature, the joints of actuation are more likely to be bending joints, as shown in Figure 13. We have also added flesh to the creature in the form of a deformable mesh that is bound to the rigid body skeleton, using fully implicit springs similar to those discussed in [32], [33] for stability. The fluid boundary is open at the top and closed elsewhere with a source term at the floor forcing fluid flow upwards. Although the model is simplistic, our resulting animation closely matches that of many flying creatures. At the down-stroke, a turbulent flow field is generated and the optimum lift and drag configurations vary rapidly with time. As a result of this turbulent behavior, the flyer's animation in this phase

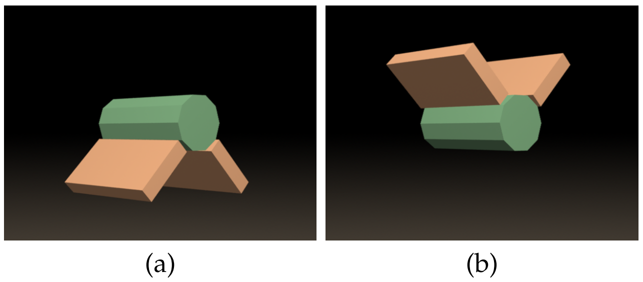

Fig. 12. The base setup for the driven flyer with no controlled joints. (a) The furthest down the flyer reaches and (b) the furthest up the flyer reaches.

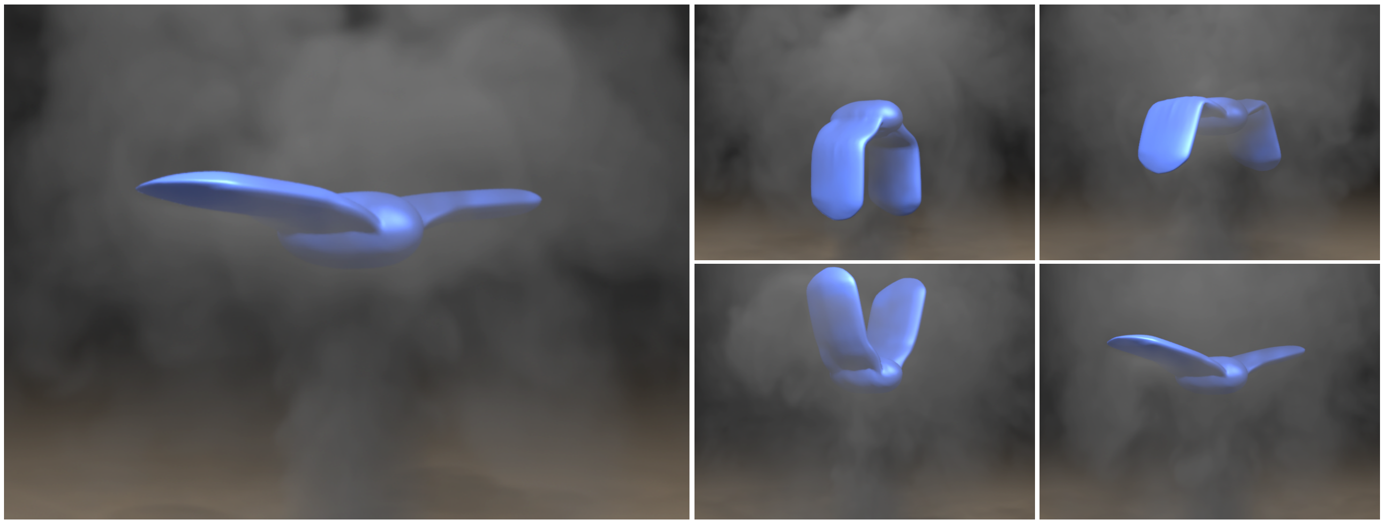
Fig. 13. A driven flyer with four joints (two on each wing) wrapped in flesh and submersed in a fluid environment. The inner-most joints have pre-specified PD targets while the outer ones are fully controlled.

can appear somewhat unnatural, despite satisfying the objective function. This accurately captures the dynamic nature of the flow field, even though the flow field may be atypical.

7.2 Aqueous Swimmer

In Figure 7 we present a fully two-way coupled swimming creature that interacts with its environment in order to swim upwards. The creature consists of a central spherical body with five legs (each of which are dynamically controlled), all of which are wrapped in flesh. The creature uses a propulsion system similar to those found in several common underwater species, expelling water in order to propel itself forward—this is implemented by enforcing a non-zero flux across the "mouth" (located on the bottom) of the creature. This creature alternates pulling and pushing water in three phases. In the first phase the creature minimizes drag (F_d) upward while drawing in water. This results in the creature pushing its arms downward as the water is drawn in. In the second phase, it propels water outward while continuing to minimize drag, which results in the creature maintaining a streamline position. In the third phase it maximizes drag without propulsion, resulting in the creature pulling its arms back up to remain steady in the fluid. The propulsion alone is not enough to move the creature significantly forward, but when used in conjunction with our controller we are able to achieve significant locomotion in a realistic way. Note that the swimmer would not be able to achieve this motion using our simplified forces. One could potentially create this animation using specialized forces but new specialized forces would then have to be created for every example. It is also interesting to note that other secondary phenomena arise as a result of the two-way coupling in this simulation, for example the flesh of the creature deforms as it moves through the fluid, and the creature sways through the flow as a result of the vortices it sheds. These realistic secondary phenomena are completely missed by methods which do not utilize two-way coupling. The fluid boundary is open at the top and closed elsewhere. As with our flyer, the model we use is fairly simple, our resulting animation again closely match that of a squid-like creature.

7.3 Timings

Figure 14 shows the timings of each of our simulations. As mentioned above, the simple examples were in 2

Example	Time
Block minimizing effort - Simple Forces	$0.03s$
Block minimizing drag - Simple Forces	$0.03s$
Block maximizing drag - Simple Forces	$0.03s$
Block minimizing effort - Navier-Stokes	$15s$
Block minimizing drag - Navier-Stokes	$14s$
Block maximizing drag - Navier-Stokes	$14s$
Human minimizing effort on back	$0.26s$
Human minimizing effort on back and arms	$0.63s$
Human minimizing effort on back	$0.44s$
Human minimizing effort on back and arms	$0.81s$
Human minimizing effort on back, drag on arms	$0.79s$
Driven Flyer without flesh	$3.2s$
Driven Flyer with flesh	$34s$
Aqueous swimmer with flesh	$83s$

Fig. 14. Timing information for our examples. The first group are the simple 2D examples with one joint. The second group are the human examples with only gravity. The third group are the human examples with gravity and wind. The final group are the more complex creatures. All Timing information is given in number of seconds per frame.

dimensions, the humans and one of the driven flyers were ran without a deformable mesh (which significantly speeds up the simulation) while the other flyer and the swimmer used a deformable mesh as skin. The 2D Navier-Stokes examples were run with our steady-state calculation and thus take considerably longer. Note that all these examples were run without our parallelized controller.

8 Discussion

Unlike previous creature controllers which have focused on generating animations in a simple environment either with no external forces or with only simple ones, we have explored using a controller that can function well in a complex fluid environment. We employ a local optimization method using gradient descent, using a modified form of proportional derivative control and a clustering system resulting in a more accurate solution and a more natural looking animation. This framework allows us to easily integrate with both simple fluid forces and Navier-Stokes fluids in order to generate high quality animations. We also explored various objective functions that can be optimized to give realistic animations for a variety of creatures.

Although our creature controller is both versatile and robust, there are some limitations. Due to our gradient descent approach and the complex fluid environment, we must numerically calculate our derivatives requiring a large number of evaluations of our objective function. This can become costly, especially with many controller degrees of freedom, but can be mitigated by parallelizing each degree of freedom. We could also reduce this by placing the solid in the new position instead of taking a full time step. This, however, makes the resulting calculations of our derivatives less accurate. We can also simplify our system by using lower resolution meshes and fluid grids, or fast approximations to physical phenomena such as the fluid environment found in [34]. These techniques would increase the speed of our algorithms at the cost of visual fidelity.

Our method can have difficulty finding an optimal solution if there are too many unspecified directly connected degrees of freedom. This is because each degree of freedom finds a local solution without any knowledge of the actions of other degrees of freedom. This can be addressed by performing a global optimization, however, this is prohibitively expensive in the presence of a complex fluid environment. An alternative approach is to use an iterative method such as the one found in [35] which would be significantly faster than solving a global optimization but would still greatly increase the cost.

As we demonstrated in our examples, we can easily integrate our technique with other motions such as a specified animation or a motion capture sequence. This allows animators to specify a vague overview of an animation as well as a reasonable objective function and generate a realistic looking animation that both adheres to the input motion and appropriately reacts to the environment. One interesting avenue of future work is to take this further and use our controller to modify the input animation with an objective function that balances staying close to the input with trying to accomplish a separate task. Similarly, our technique can take as input an animation specified by another controller such as [11] which specifies a global, low dimensional approximation. This will give both the benefits of the global overall behavior and the local corrections under the influence of a complex environment. However, this can only be used for creatures where such input can be provided (e.g. a human). We can also use our own controller to specify the animation in a multi-pass approach. This avoids the need for solving many joints simultaneously, and allows certain joints to be specified as constraints while others are solved for via a controller.

We explored different objective functions including minimizing effort, minimizing and maximizing drag, and minimizing and maximizing lift. Although these objective functions worked well for our examples, they are by no means the only objective functions that can be used. In particular, we found that some objective functions were wholly inappropriate for a given scenario; minimizing effort for a human with no initial motion, for example, gives a human which stands still. A penalty function could be added to force the human to move forward. An interesting direction for future work would be to determine other objective functions that creatures use when performing locomotion.

Although it is quite easy to add an objective function to our framework, there are certain problems that are difficult to quantify with an objective function. For example, human location is a complex dynamical behavior. One way to specify an objective function is to use a much more complex human model that models elements such as muscle strain. This would allow for the use of an objective function that minimizes the effort exerted by the muscles. Another option is to simply use motion capture data and specify an objective function that stays close to the input motion capture data. This is an interesting avenue for future work.

9 Conclusion

In summary, our technique addresses a difficult problem that has not yet been addressed by the computer graphics community. When exploring a very hard problem, it is often best to begin with a simple, yet promising, approach. We do this by first combining many complex physical systems including, rigid body simulation, deformable object simulation, collisions, contact, articulation, PD control, and fluid simulation. We believe that this paper describes a simple, scalable (to large numbers of degrees of freedom that are not directly connected) methodology and we show that the method provides reasonable results for simple problems whose analytical results are easy to verify. This paper also explores more complex scenarios, for example our largest example has 15 degrees of freedom. However, this paper is not meant to be the ultimate solution to this difficult problem and we encourage others to expand on our work by creating more complex models and controllers that can provide difficult, realistic results such as convincing human locomotion.

Acknowledgements

The authors would like to acknowledge Sean Meador for his help with modeling. Research supported in part by a Packard Foundation Fellowship, an Okawa Foundation Research Grant, ONR N0014-06-1-0393, ONR N00014-02-1-0720, ONR N00014-05-1-0479 for a computing cluster, NIH U54-GM072970, NSF IIS-0326388, NSF ITR-0205671 and NSF CCF-0541148.

References

[1] G. S. P. Miller, "The motion dynamics of snakes and worms," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 169–173, 1988.

[2] K. Sims, "Evolving virtual creatures," in *Proc. SIGGRAPH 94*, 1994, pp. 15–22.

[3] 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*, 1994, pp. 43–50.

[4] R. Grzeszczuk and D. Terzopoulos, "Automated learning of muscle-actuated locomotion through control abstraction," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995, pp. 63–70.

[5] J. Wu and Z. Popović, "Realistic modeling of bird flight animations," in *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, 2003, pp. 888–895.

[6] N. Galoppo, M. Otaduy, S. Tekin, M. Gross, and M. C. Lin, "Soft articulated characters with fast contact handling," *Comput. Graph. Forum (Proc. Eurographics)*, vol. 26, no. 3, pp. 243–253, 2007.

[7] T. Shinar, C. Schroeder, and R. Fedkiw, "Two-way coupling of rigid and deformable bodies," in *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2008, pp. 95–103.

[8] J. Hodgins, W. Wooten, D. Brogan, and J. O'Brien, "Animating human athletics," in *Proc. of SIGGRAPH '95*, 1995, pp. 71–78.

[9] A. Witkin and M. Kass, "Spacetime constraints," in *Comput. Graph. (Proc. SIGGRAPH '88)*, vol. 22, 1988, pp. 159–168.

[10] Z. Popović and A. Witkin, "Physically based motion transformation," in *Comput. Graph. (Proc. SIGGRAPH '99)*, 1999, pp. 11–20.

[11] A. Safonova, J. K. Hodgins, and N. S. Pollard, "Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 514–521, 2004.

[12] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *ACM Trans. Graph. (SIGGRAPH Proc.)*, 2001, pp. 251–260.

[13] V. Zordan and J. Hodgins, "Motion capture-driven simulations that hit and react," in *Proc. ACM SIGGRAPH Symp. on Comput. Anim.*, 2002, pp. 89–96.

[14] O. Arikan, D. A. Forsyth, and J. F. O'Brien, "Pushing people around," in *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2005, pp. 59–66.

[15] V. Zordan, A. Majkowska, B. Chiu, and M. Fast, "Dynamic response for motion capture animation," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 697–701, 2005.

[16] Y. Ye and C. K. Liu, "Animating responsive characters with dynamic constraints in near-unactuated coordinates," in *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, 2008, pp. 1–5.

[17] R. Metoyer, V. Zordan, B. Hermens, C.-C. Wu, and M. Soriano, "Psychologically inspired anticipation and dynamic response for impacts to the head and upper body," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 1, pp. 173–185, 2008.

[18] C. Liu, A. Hertzmann, and Z. Popović, "Learning physics-based motion style with nonlinear inverse optimization," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1071–1081, 2005.

[19] K. Yin, S. Coros, P. Beaudoin, and M. van de Panne, "Continuation methods for adapting simulated skills," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 2008, pp. 1–7.

[20] Y. Abe, M. da Silva, and J. Popović, "Multiobjective control with frictional contacts," in *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2007, pp. 249–258.

[21] M. Neff and E. Fiume, "Modeling tension and relaxation for computer animation," in *Proc. ACM SIGGRAPH Symp. on Comput. Anim.*, 2002, pp. 77–80.

[22] R. Weinstein, E. Guendelman, and R. Fedkiw, "Impulse-based control of joints and muscles," *IEEE Trans. on Vis. and Comput. Graph.*, vol. 14, no. 1, pp. 37–46, 2008.

[23] 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*, 2004, pp. 39–47.

[24] N. Kwatra, C. Wojtan, M. Carlson, I. Essa, P. J. Mucha, and G. Turk, "Fluid simulation with articulated bodies," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 1, pp. 70–80, 2010.

[25] A. Robinson-Mosher, T. Shinar, J. Grétarsson, J. Su, and R. Fedkiw, "Two-way coupling of fluids to rigid and deformable solids and shells," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 46:1–46:9, Aug. 2008.

[26] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. San Diego, USA: Academic Press, 1981.

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

[28] 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.

[29] 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.

[30] A. Robinson-Mosher, C. Schroeder, and R. Fedkiw, "A symmetric positive definite formulation for monolithic fluid structure interaction," *J. Comput. Phys.*, 2010, in review.

[31] J. Stam, "Stable fluids," in *Proc. of SIGGRAPH 99*, 1999, pp. 121–128.

[32] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," *Comput. Graph. (SIGGRAPH Proc.)*, pp. 317–324, 1999.

[33] A. Selle, M. Lentine, and R. Fedkiw, "A mass spring model for hair simulation," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 64.1–64.11, Aug. 2008.

[34] A. Treuille, A. Lewis, and Z. Popović, "Model reduction for real-time fluids," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 826–834, 2006.

[35] C. K. Liu, A. Hertzmann, and Z. Popović, "Composition of complex optimal multi-character motions," in *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2006, pp. 215–222.

Michael Lentine received his B.S. in Computer Science from Carnegie Mellon University in 2007. While there, he was an undergraduate research assistant working on computer animation research. He is currently pursuing his Ph.D. at Stanford University where he was awarded an Intel Graduate Fellowship. He has also been a consultant for Industrial Light + Magic for the last two years working in the research and development group on physical simulation.

Jón Tómas Grétarsson received his B.S. in Computer Science from Worcester Polytechnic Institute in 2006, and is currently pursuing his Ph.D. at Stanford University.

Craig Schroeder received his B.S. in computer science and mathematics and his M.S. in computer science at Drexel University in 2006. He is currently pursuing his Ph.D. in computer science at Stanford University, where he was awarded a Stanford Graduate Fellowship. He also works at Pixar Animation Studios and has received a screen credit on "Up."

Avi Robinson-Mosher received his B.S. in computer science and physics at Yale University in 2004, his M.Sc. in Philosophy, Policy and Social Value at the London School of Economics in 2005, and his M.S. and Ph.D. in computer science at Stanford University in 2010. He is entering a postdoctoral position at the Wyss Institute for Biologically Inspired Engineering at Harvard University.

Ron Fedkiw received his Ph.D. in Mathematics from UCLA in 1996 and did postdoctoral studies both at UCLA in Mathematics and at Caltech in Aeronautics before joining the Stanford Computer Science Department. He was awarded an Academy Award from The Academy of Motion Picture Arts and Sciences, the National Academy of Science Award for Initiatives in Research, a Packard Foundation Fellowship, a Presidential Early Career Award for Scientists and Engineers (PECASE), a Sloan Research Fellowship, the ACM Siggraph Significant New Researcher Award, an Office of Naval Research Young Investigator Program Award (ONR YIP), the Okawa Foundation Research Grant, the Robert Bosch Faculty Scholarship, the Robert N. Noyce Family Faculty Scholarship, two distinguished teaching awards, etc. Currently he is on the editorial board of the Journal of Computational Physics, Journal of Scientific Computing, SIAM Journal on Imaging Sciences, and Communications in Mathematical Sciences, and he participates in the reviewing process of a number of journals and funding agencies. He has published over 80 research papers in computational physics, computer graphics and vision, as well as a book on level set methods. For the past seven years, he has been a consultant with Industrial Light + Magic. He received screen credits for his work on "Terminator 3: Rise of the Machines", "Star Wars: Episode III - Revenge of the Sith", "Poseidon" and "Evan Almighty."