Real-time locomotion with character-fluid interactions

Luis Bermudez
Clemson University
lbermud@clemson.edu

Jerry Tessendorf
Clemson University
jtessen@clemson.edu

Daniel Zimmermann
Studio Gobo
daniel.zimmermann@studiogobo.com

Victor Zordan
Clemson University
vbz@clemson.edu

Figure 1: A sample animation of a character in a game-like setting. As the player moves the character through the body of water the proposed system automatically modifies the animation by transitioning from a jog to a walk and adapting the motion to account for the drag of the water.

ABSTRACT

This paper proposes a real-time approach to animate a character walking in different environments for a video game setting. The technique combines a physics-based model with procedural animation and motion editing. Highlighting environmental interaction in fluids, the paper leverages simplified drag forces to drive realistic changes from existing locomotion data. To demonstrate generalizability of the effort, we also generate forces from impulse felt from interactions in the environment.

CCS CONCEPTS

• **Computing Methodologies** → **Animation**; *Physical Simulation*; *Procedural Animation*; *Motion Processing*;

KEYWORDS

physics-based animation; character locomotion

ACM Reference Format:
Luis Bermudez, Jerry Tessendorf, Daniel Zimmermann, and Victor Zordan. 2018. Real-time locomotion with character-fluid interactions. In *MIG '18: Motion, Interaction and Games (MIG '18), November 8–10, 2018, Limassol, Cyprus.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3274247.3274515

MIG '18, November 8–10, 2018, Limassol, Cyprus

ACM ISBN 978-1-4503-6015-9/18/11…$15.00
https://doi.org/10.1145/3274247.3274515

1 INTRODUCTION

The use of data-driven animation to control characters has become standard for most 3D video games. Regardless of whether the motion is from human motion capture or manually created, the overhead of producing such motion is high. In this paper, our goal is to maximize the reuse of locomotion data specifically by modifying existing motion to be appropriate in more settings than originally intended – for example, automatically editing the motion of walking to produce a character wading through water. A further benefit of this reuse is the reduction of unnecessary duplication in storage, allowing the animated character to move within different scenarios and environments using an existing, fixed-size database.

The technique we explore is a two-step approach that employs a simple abstract physical model to determine plausible interaction with the environment. The physical response is used in turn to modify the original motion using procedural animation and motion editing. For the latter, we propose a specialized *step controller* to construct new motion based on the interaction. The step controller is coupled with customized motion-editing rules for additional effects, such as the arms lifting above the water in our wading example. The power of this technique includes the ability to support multiple forms of interaction, including changing environments and impulses for direct interaction.

We focus on the effect of character locomotion through fluids, especially how drag forces would affect the character in various settings (See Figure 1). To this end, drag forces are computed using a calculation of drag adapted from fluid mechanics and applied to our abstract model. This approach computes drag forces very quickly, and sidesteps the need for a full-blown fluid simulation of coupling as proposed in other techniques [Si et al. 2014; Tan et al. 2011]. While the drag is approximated, it includes reasonable detail about the interaction. Specifically, in addition to information about

the medium such as density and relative flow velocity, the approximation accounts for the cross-sectional area and limb velocity of the character's locomotion – tightly coupling the motion and the effect of drag.

While we limit our examples to mostly fluids, the technique we propose can be generalized to other settings, for example, climbing slopes, walking with additional weight/encumbrances, and so on.

2 RELATED WORK

Over the past decade, researchers have proposed ample approaches to control physics-based characters that can locomote, [Coros et al. 2010; da Silva et al. 2008; de Lasa et al. 2010; Kwon and Hodgins 2010; Lee et al. 2010; Muico et al. 2009; Peng et al. 2016, 2017; Sok et al. 2007; Wampler and Popović 2009; Yin et al. 2007], adapt to new environments [Coros et al. 2009, 2008; Mordatch et al. 2010; Wu and Popović 2010; Ye and Liu 2010; Zimmermann et al. 2015], and respond in a variety of situations [Abe et al. 2007; Macchietto et al. 2009; Wu and Zordan 2010; Ye and Liu 2008; Yin et al. 2008]. However, we have not yet seen adoption of these techniques in video games. Likely reasons are that the published approaches are too slow, not robust enough for their needs, and do not fit well within existing pipelines.

Instead of full simulation, the approach of utilizing physical attributes to modify motion capture has been investigated in a wide variety of forms [Abe et al. 2004; Arikan et al. 2005; Bruderlin and Calvert 1989; Girard and Maciejewski 1985; Han et al. 2016; Tak and Ko 2005; Yin et al. 2005], among others. The work presented here belongs to a specific growing sub-area within this topic — so-called *hybrid* techniques that directly combine kinematic and dynamic models for character animation [Majkowska and Faloutsos 2007; Nguyen et al. 2012; Shapiro et al. 2003; Ye and Liu 2008; Zordan et al. 2005]. Among these, approaches such as [Shapiro et al. 2003] and [Zordan et al. 2005] modulate between the use of kinematics and dynamics along the time axis, employing each as necessary based on the conditions of the character. In contrast other hybrid approaches, such as [Ye and Liu 2008] and [Nguyen et al. 2012], splice the kinematics and dynamics across the body at a given instant in time. Still other techniques, such as Ishigaki et al [2009] , layer a kinematic model over a dynamic one. This work is similar to our own in that we propose a simple dynamic model and layer an articulated kinematic model on top of this one. However, our focus is unique in our attempt to approximate the effect of interaction with fluids.

A main focus of our effort is to simulate the effect of character-fluid animation. Others have approached this for fish and humanoids through fluid coupling with full physical simulations [Grzeszczuk and Terzopoulos 1995; Si et al. 2014; Tan et al. 2011; Tu and Terzopoulos 1994]. Our goal of real-time animation, with a motion capture-driven (kinematic) controller motivates our approach of simplified interaction. While approximation is a simplification from full simulation, we do draw from fluid mechanics for our drag force model [Cengel and Cimbala 2006].

3 ABSTRACT MODEL

Our technique uses a simple physics-based simulation to represent the character. Abstractly, this is a rigid body that holds a position and orientation plus derivatives. This abstract model allows response to external stimuli through input from forces and torques (generalized forces). Such abstract models have been proposed for a variety of applications [Ishigaki et al. 2009; Nguyen et al. 2012; Ye and Liu 2010]. We also add *behavior* controllers to maintain the velocity of the center of mass and the aggregate yaw and pitch rotation of the character. (We opt not to allow any rotation about the roll axis to maintain believable *upright* locomotion.) The output of the abstract model is used to inform a kinematic animation system (Section 4).

Conceptually, the controllers help to give the character the appearance of an active response that is physically drawn from ground reaction forces. While we sidestep the full simulation of such, the purpose of the controllers is both to create a purposeful response to general force input, and subsequently to return the character to the original (unperturbed) motion when the effect of the input dissipates.

3.1 Velocity Controller

When external disturbance force $\boldsymbol{F}_{\text{ext}}$ occurs, the velocity of the abstract model $\boldsymbol{v}$ is affected by the acceleration, $\dot{\boldsymbol{v}} = \boldsymbol{F}_{\text{ext}}/\boldsymbol{m}$. Our velocity controller dissipates the effect and returns the velocity to the desired, game engine-driven velocity, $\boldsymbol{v_d}$. We treat this as a physical, *character-driven* response to the external disturbance, computed by a force-based PD (proportional-derivative) servo with its output, $\boldsymbol{F}$, applied to the abstract model.

Specifically, the PD servo computes forces based on the velocity error and current acceleration as

$$\boldsymbol{F} = k(\boldsymbol{v_d} - \boldsymbol{v}) - b(\dot{\boldsymbol{v}}) \tag{1}$$

which corrects for external perturbations based on the gain value k and damping value b. Note, the control force is computed based on its desired velocity set by the game engine which, for locomotion, is the user-specified velocity for the original motion data.

We tune this system to produce the desired qualitative effect of this controller, namely, that the character appears to respond in a timely manner to external disturbances.

3.2 Orientation Controller

The pitch/yaw rotation of the abstract model is likewise affected by torque-based disturbances and we add an orientation controller to maintain upright, forward-facing locomotion. Similar effects are created by other researchers through direct and indirect control, for example [Han et al. 2016; Komura et al. 2004].

We employ a PD servo with torque, $\boldsymbol{\tau}$, that is applied to the abstract model. This torque changes the orientation, θ, based on the desired rotation, θ_d.

$$\boldsymbol{\tau} = k_\theta(\theta_d - \theta)\hat{\boldsymbol{r}}_{\boldsymbol{\theta}} - b_\theta\dot{\theta}\hat{\boldsymbol{r}}_{\dot{\boldsymbol{\theta}}} \tag{2}$$

which corrects for disturbances based on the gain value k_θ and damping b_θ as well as the orientation error $(\theta_d - \theta)$ and the angular velocity $\dot{\theta}$ in the appropriate directions of $\hat{\boldsymbol{r}}$, where $\hat{}$ denotes a normalized vector.

The torque is computed based on the desired orientation set by the game engine. For our examples, the desired pitch rotation is set to zero, so that the character attempts not to pitch. We set yaw

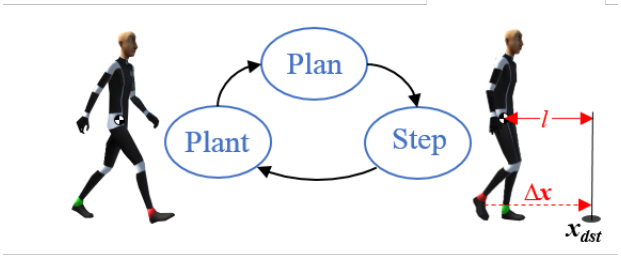

Figure 2: Step control state machine. Starting from planning, the controller computes the location of the new step and releases the back foot off the ground. During stepping, the controller moves the swing foot and keeps track of the remaining distance to complete the intended step. The step is planted once the foot has reach the distance to complete the step.

rotation to zero degrees so the character is also always attempting to face the forward walking direction.

4 KINEMATIC CONTROL

From the abstract model's state, we construct the skeletal motion for the character kinematically. To this end, we employ a step controller to guide the footsteps and legs, and derive the remaining motion from motion capture playback that is synced with the step controller.

4.1 Step Controller

Step controllers with and without motion capture have been employed extensively to produce locomotion animation that is controlled and responsive [Agrawal and van de Panne 2016; Boulic et al. 2016; Girard and Maciejewski 1985; Mordatch et al. 2010; Wu et al. 2008; Wu and Zordan 2010; Wu and Popović 2010]. Our kinematic step controller is footstep-driven and employs inverse kinematics (IK) to move the legs consistently with the desired foot position and the phase of the locomotion cycle.

The step controller follows the layout shown in Figure 2. At the start of each step, during the **plan** state, the step controller decides where to place the step, $\boldsymbol{x}_{\text{dst}}$, according to given conditions. Specifically, the foot is to be placed at the nominal length, l, away from the character at $\boldsymbol{x}_{\text{root}}$ in the direction of the current velocity, $\hat{\boldsymbol{v}}$. We also add a constant value to the relative left or right ($\hat{\boldsymbol{v}}_{\perp}$) depending on which foot is stepping, as

$$\boldsymbol{x}_{\text{dst}} = \boldsymbol{x}_{\text{root}} + l\hat{\boldsymbol{v}} + w\hat{\boldsymbol{v}}_{\perp} \tag{3}$$

where w is the lateral width of the steps from the centerline of the body. All the calculations and values in the step controller are projected onto the ground plane. Thus, the calculations in this section assume that each variable is projected onto the two dimensional ground plane.

The footstep destination allows us to compute the actual distance the foot must travel, from its current position, $\boldsymbol{x}_{\text{src}}$, to $\boldsymbol{x}_{\text{dst}}$, as

$$\Delta\boldsymbol{x} = \|\boldsymbol{x}_{\text{dst}} - \boldsymbol{x}_{\text{src}}\|. \tag{4}$$

The length, l, is a user-specified measure for the given locomotion. For clarity, we define l as the distance from the character's root to the desired stepping foot destination, projected onto the character's normalized velocity (See Figure 2, right). The width value w is negative for the left foot, and positive for the right for our coordinate set up. Note, the foot may not necessarily land at the intended location if there is a disturbance, but it serves as the goal for planning.

The pacing of the footstep happens during the **step** state and is carefully scheduled to assure the footstep is complete (just) before the root reaches the desired foot location $\boldsymbol{x}_{\text{dst}}$. When no disturbance is present, the calculation for the timing of the foot is

$$t_{\text{step}} = \frac{\|\boldsymbol{x}_{\text{twilight}} - \boldsymbol{x}_{\text{root}}\|}{\|\boldsymbol{v}_{\boldsymbol{d}}\|} \tag{5}$$

where $\boldsymbol{x}_{\text{twilight}} = \boldsymbol{x}_{\text{dst}} - \boldsymbol{\epsilon}$. The user-specified vector $\boldsymbol{\epsilon}$ has a small value in the direction of the stepping and ensures the step will complete just shy of the root hitting the to-be planted foot. The step needs to be completed by the time that the character reaches the intended step destination as this allows the character to continue to be balanced and supported by the feet.

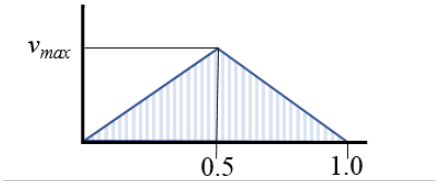

Figure 3: The step velocity maximum, $v_{\max}$, is computed for each footstep. This plot shows the time-normalized schedule for the foot's velocity during a step.

Using the step velocity schedule shown in Figure 3, we set the velocity for the stepping-foot handle which acts as the IK end effector for the stepping leg. From Equations 4 and 5, we know the average velocity of the step must be $\boldsymbol{v}_{\text{avg}} = \Delta\boldsymbol{x}/t_{\text{step}}$. Our scheduler starts and finishes with a speed of zero during each step and uses linear interpolation to reach a maximum of two times this time-normalized average. This maintains the overall average with a speed up and slow down during each step.

For animation, during each frame the IK "handle" is advanced along the path toward the destination the distance specified according to the current scheduled velocity and the time elapsed. As the foot reaches its destination or as the scheduled time ends (whichever is first), the stepper enters the foot **plant** state and the IK handle is held fixed at the new location. During foot plant, the root progresses and the IK solver continues to compute the updated leg animation. To complete the cycle, the **plan** state is triggered once again when the character root position is less than the nominal length, l, from $\boldsymbol{x}_{\text{dst}}$. Then, the cycle repeats.

Beyond the leg IK solver, the remainder of the animation is handled by the motion editing module which accounts for any further abstract adjustments (Section 4.2). One exception is the yaw control. The yaw affects the direction chosen at the beginning of each step. That is, the direction of the step is always correlated with the yaw

orientation of the abstract model. In this way, the direction of the step can change at any time step due to disturbances. However, the intended size of the step does not change and otherwise follows the same procedure as described.

4.2 Motion Editing

To augment the step controller, we use motion capture data of a walk cycle to animate the remaining joints of the character. The motion capture data of walking is played in sync with the stepping controller walk cycle by aligning the phase. Specifically, offline we segment two time syncs at the start of each the left and right step and, online, the system computes the frame (or interpolated partial frame) for the motion data based on the percentage completion for the active foot's step. Along with the given frame, the character is positioned and rotated according to the abstract model's state. The leg animation from the motion capture is the start point for the step controller IK which modifies the motion according to the footstep plan. The foot height for the IK is extracted from the given frame of motion capture for the walk cycle.

Note, without the step controller, the edited motion playback has severe foot sliding during disturbances as the abstract model changes the character speed and body orientation. It is the step controller that corrects footskate and creates the appearance of clean, crisp foot contact. However, when the character is wading through a fluid, which obstructs the view of the feet, the step controller can be turned off. The result is a character that slows down and leans due to the abstract model, but otherwise follows the standard animation pipeline for the character.

When the character is submerged in fluids, the hands are also edited kinematically. The hands (and thus arms) are kept above the fluid surface using IK. This minimizes the amount of the body submerged in the water which leads to less area, and thus less drag force (Section 5). More on this in the next section. As the results and accompanying video demonstrate, keeping the hands (and arms) raised makes a significant contribution to decreasing the drag when locomoting in deep water.

5 DRAG FORCES

When the character is walking through fluids, we use a drag model to approximate the forces applied onto the character. This effectively slows down the character based on the influences of the fluid and the character's motion. The fluid also imparts a torque onto the character. Qualitatively, a human rotates to compensate, for example, pitching forward while wading through water. Empirically, we can generate this effect by applying a torque to our abstract model and allowing the rotation controller (Section 3.2) to compensate for its influence.

5.1 Fluid Model

The model that we employ to compute drag forces comes from fluid mechanics [Cengel and Cimbala 2006], as follows

$$\boldsymbol{F} = C_{\mathrm{D}} \rho A (\boldsymbol{u} \cdot \boldsymbol{u}) \hat{\boldsymbol{u}} \tag{6}$$

where C_{D} is the coefficient of drag, which accounts for the character's geometry. ρ is the mass density of the fluid. A is the area projection of the character's body orthogonal to the direction of the water velocity. This area consists of only the submerged portion of the character facing the water velocity. $\boldsymbol{u}$ is the flow velocity relative to the body. More precisely, $\boldsymbol{u} = (\boldsymbol{u_f} - \boldsymbol{v})$ where $\boldsymbol{u_f}$ is the velocity of the fluid and $\boldsymbol{v}$ is the velocity of the character. Note, the drag force is proportional to the square of the relative flow velocity and thus most sensitive to this value.

To precisely approximate the drag using this model, we compute the drag force for each relevant polygon in the character mesh. This accounts for the character's movement through the water, including the velocity changes due to the stepping. We define a "relevant" polygon as having two attributes: 1) the polygon needs to be below the water depth (at least the centroid of the polygon is below water depth); and 2) the polygon's normal needs to be facing opposite the fluid velocity, i.e. the dot product of the polygon normal and the fluid velocity vector needs to be negative. We sum the relevant-polygon drag forces together to form the total drag force for the character at each time step.

We compute the global velocity of each relevant polygon by finite differencing with the current and previous time step. We then project each polygon onto the plane normal to the relative flow velocity and calculate the projected area of each polygon. Using these velocity and area values, each polygon drag force is computed from Equation 6. We aggregate the per-polygon drag into the total drag force and apply this force to the abstract model.

We also apply a torque to the character based on the total force to account for their effect on the character's rotation. Our torque computation is derived from the total drag force and its moment arm. To compute the torque on the abstract model we compute the the moment arm as the vector from the Center of Mass (C_{M}) to the Center of Pressure (C_{P}). We approximate the COM as the average of the hip joints and compute the C_{P} from the force-weighted average position.

$$\boldsymbol{\tau} = (\boldsymbol{C}_{\mathrm{P}} - \boldsymbol{C}_{\mathrm{M}}) \times \boldsymbol{F} \tag{7}$$

where $\boldsymbol{F}$ is the total drag force from all the polygons. Drag torque is calculated and applied to the abstract model at each time step.

5.2 Fluid Parameters

The mass density and coefficient of drag from Equation 6 depend on the fluid, be it water, oil, mud, or otherwise. The mass density is a straightforward constant that is defined for each fluid. For example, taken from a standard text, the mass density for water is $1000 kg/m^3$ while the mass density for standing mud is $1840 kg/m^3$.

In contrast, the C_{D} term is an aggregate term that is influenced by a variety of factors. Notably, C_{D} changes based on the turbulence and relative flow velocity. It also depends on the shape and surface of the drag object. C_{D} further changes with the type of fluid or mass density. While in fluid mechanics, C_{D} is often approximated through experimental testing, for our purposes we propose a structure that is fast to compute and easily tunable.

Specifically, we model the coefficient of drag as a function of mass density, projected area, and relative flow velocity of the following form.

$$C_{\mathrm{D}} = C_{\mathrm{D0}} + \frac{B}{\rho A (\boldsymbol{u} \cdot \boldsymbol{u})} \tag{8}$$

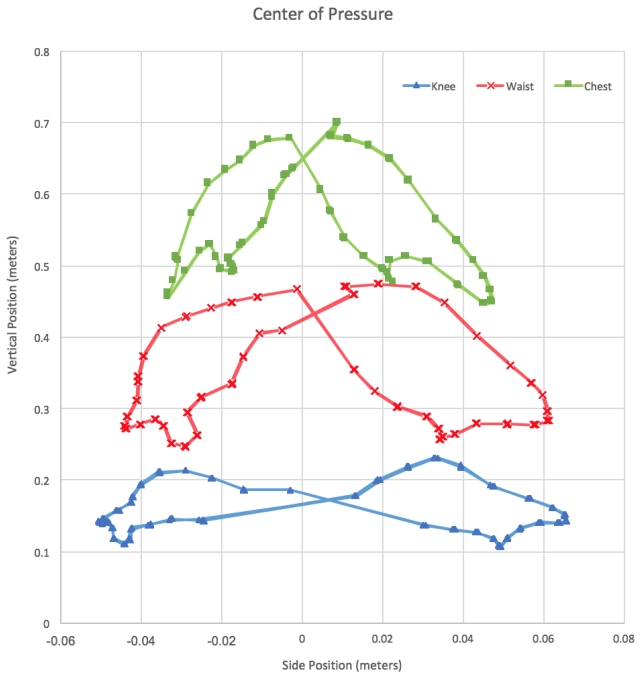

Figure 4: Center of pressure (C_P) phase plot for various water levels. This graph shows a number of cycles (each) for changing water height indicating how the torque will vary as the character moves with each locomotion cycle.

Table 1: Fluid Material Parameters. Data extracted from waist-height fluid of various types. The so-named goo material was constructed to test the system with extreme parameters, selected to significantly increase drag. The last two columns are aggregate terms that help summarize the impact on the character.

Fluid Type	density	Median C_D	Ave v drop
Water	1000	1.08	72 %
Mud	1840	38.9	60 %
Goo	2750	370	38 %

C_{D0} is the base value which we build as a function of mass density, and B is an offset term that acts as a force magnitude. For ease of tuning, C_{D0} and B are both sigmoid functions, specifically, $a \Big/ 1 + e^{-b(\rho - c)}$ where a, b, c are tuned constants that make C_{D0} a range of values that works for values between air and water densities and B a range of values that works for more dense fluids, e.g. mud or liquid cement. See Table 1 for precomputed C_D values.

6 RESULTS

We implement our technique within Unity3D [Unity3D 2018] which supports the physics for the abstract model, the IK solver, and the motion capture playback. To showcase the power of the approach, we place the character in a number of different scenarios. While the bulk of the experiments show the character interacting with fluid, we also include some physical interactions with basic impulses in isolation. The accompanying video highlights the animations that are possible with our system. Figures 4- 8 show some of the analytics associated with the animations.

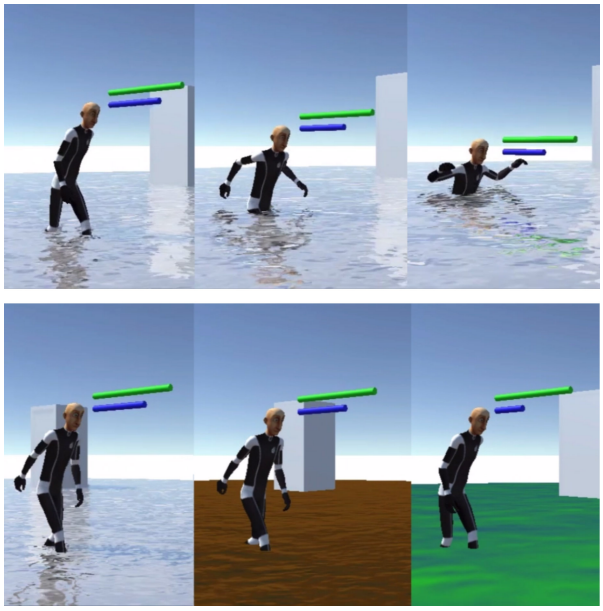

Figure 5: Top row shows results for differing water depths, while the bottom row shows a set of different material properties. Note the differences that appear in the velocity (blue bar) as well as the body pose variations.

As seen in Figure 5, we make assessments of the response of the system by changing the fluid depth (top) and changing the fluid material properties (bottom) for character-fluid interaction. Figure 4 shows how the C_P changes as the water level increases. The variation over the cycle has a characteristic shape but morphs slowly as the depth increase. The higher water yields higher drag forces, decreasing character velocity as seen in Figure 6. Further, the deeper water results in vertically increased centers of pressure, which yield lower moment arms. However, when computing the torques, as the moment arm decreases the drag force increases and the resulting torque does not change significantly between water levels.

In Figure 7, the effect of different fluid parameters can be seen. The differences are summarized in Table 1. Not surprisingly, denser fluids significantly slow down the character. For example, mud creates a higher drag force than water. And as such, the amount of torque is also increased with denser fluids (in direct proportion with the force). The percentage drop from the original walking velocity for each fluid type appears in the right column of the table. For the "goo" example, the assumptions built into the kinematic model start to breakdown as the character begins to look less realistic due to the slowdown.

Finally, the impact of the arm IK is shown in Figure 8. The hands controller significantly reduces the amount of drag force applied

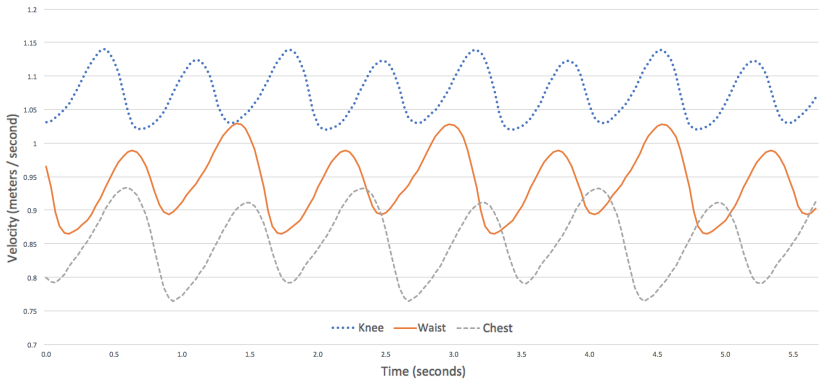

Figure 6: Character velocity output for walk cycles at various water depths. All plots use the fluid parameters for water. As the fluid depth increase, the drag increases and the overall speed drops

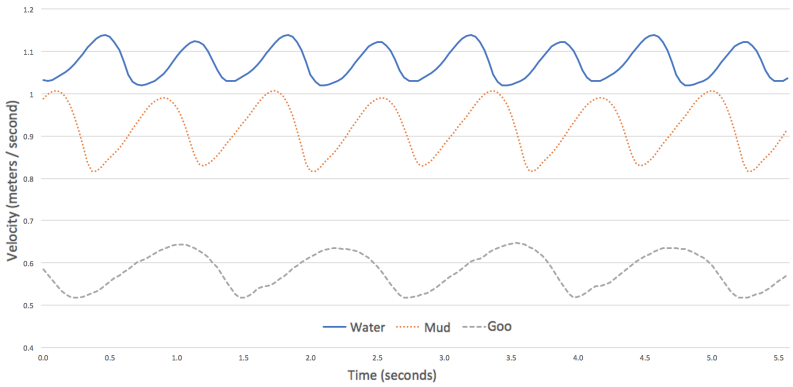

Figure 7: Character velocity output for walk cycles for knee depth water using three distinct fluid parameter sets. As the fluid density increases, the drag increases and the overall speed drops.

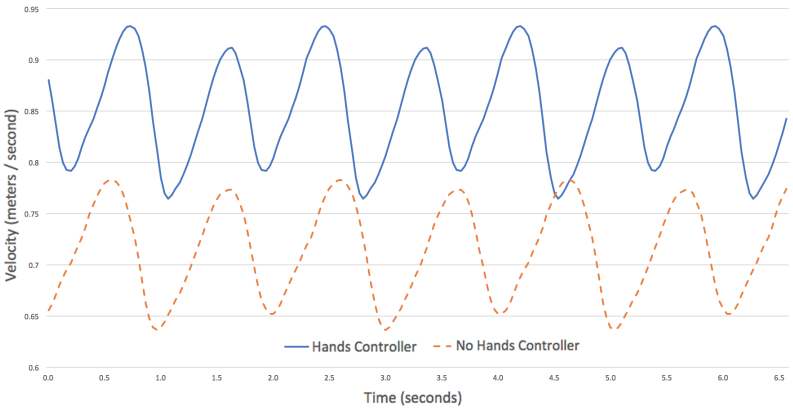

Figure 8: Character velocity output for walk cycles at chest depth water. The blue/solid graph shows the effect of raising the hands/arms on the output motion. Without raising the arms, the drag is increased.

Figure 9: Our method can drive high fidelity water simulators.

onto the character over time. This is due to a decrease in surface area when the hands are outside the water. While our approach is quite simple, it motivates a plausible theory that suggests that humans lift their arms, not only to keep their hands dry, but also to make it easier to walk when wading in water.

The drag forces from the jogging animation (Figure 1) are higher than the drag forces from walking. The character can transition from jogging to walking to decrease the amount of drag forces that impact the character. Note, the character velocity resulting from drag forces is higher for jogging compared to walking. These results are due to the higher velocity of jogging compared to walking. At knee level water, both jogging and running in water lead to about a fourth loss in velocity. The absolute velocities increase with jogging, but the relative velocity changes remain similar to walking.

Finally, the supplemental material also includes a separate example of water simulation driven by the character's walking as seen in Figure 9. This simulation uses an existing simulator [Tessendorf 2004, 2014] to demonstrate how the interaction would look with the water reacting. Note, this simulation inputs motion from the technique described, the water simulation only responds to the modified motion in the animation.

7 DISCUSSION

The abstract model provides a generic procedure for inputting disturbances. While the system is very simple, it affords a complex set of interactions, including the combination of impulse forces and persistent drag forces as shown in the video. Combined with the control, the physical model can "act" to resume its behavior following a disturbance. While we sidestep the method for computing control from the ground reaction force via full simulation, we assume that such efforts are plausible and focus on making the system tunable and fast. To this end, our system has a finite number of parameters and runs at 60 fps (with Unity3D's renderer, without screen recording) on a standard desktop.

There are several subtleties to the approach we propose. For example, there is a nuanced coupling between the impact of the drag forces and the step controller. Our step controller drives the step speed, which directly impacts the drag force. The faster the step speed, the more drag force is generated. However, as the drag increases the character velocity drops and thus the step speed. In sum, this feedback loop leads to an oscillatory drag effect that settles into a stable non-linear rhythm under persistent conditions (Figures 6- 8), such as wading through fixed depth water.

Similarly, the cyclic nature of the step speed results in the center of pressure (C_P) moving left and right between each stepping leg. The C_P is closer to the stepping foot because it has a higher speed than the standing foot. This cyclic translation of the C_P results in a cyclic moment arm and torque, causing a small yaw rotation back and forth for each step. Similarly, the pitch rotation is affected as the C_P moves slightly up and down in its cycle. While these motions are small, the visual quality of the result is perceptible and gives the impression that the character is immersed in the fluid.

In the results, we include examples without the step controller (just the motion data) - both to highlight the differences and expose the option of foregoing the step control. There is a trade-off, as the step control has both pros and cons. Namely, the step controller offers greater control and has more options than editing motion data directly. However, this level of control can lead to animations that look more robotic when parameters are not tuned properly. On the other hand, the step control also leads to a more pronounced visual effect of the drag forces, as seen in the results. Additionally, the step controller removes issues with foot skating and allows for a side-stepping behavior in reaction to a disturbance that comes from the lateral direction.

Modifying the hand/arm animation accomplishes the cursory goal of keeping the hands and arms outside of the water as possible. It also leads to an incidental lowering of the surface area submerged in the water for a given water height. The decreased surface area leads to less drag forces. Likewise, we hypothesize that a *high step* controller for low water (for example, a flamingo-like walk) would significantly decreases the drag forces, again due to lower surface area when taking the foot out of the water. More so, keeping the hands out of the water significantly decreases the drag forces, since the hands are kept out of the water all the time instead of only during a step.

Finally, we point out that due to the hybrid nature of our system, certain aspects do not parallel physical counterparts. For example, the drag forces are computed at each polygon, and the aggregate drag force is applied at the center of pressure instead of each drag force being applied at each polygon. Our model applies the aggregate drag force to character, which slows down the character velocity. The decrease in character velocity leads to a decrease in the step speed, via the step controller. This is the opposite of what would happen in reality where the legs would be slowed down by the drag forces which would *then* lead to a slow down in the character velocity.

8 CONCLUSION

We present a system where real-time motion-capture locomotion is made to respond to discrete and continuous physical disturbances. This is accomplished by applying external forces to a simple representation of the character. The physically-based model reacts to the forces as it incorporates a response from a behavior control

routine. The physics serves as input for a procedurally generated, responsive step controller which is synced to motion play back.

There are several areas of future work we wish to investigate, including variety in water flow, for example turbulence and ocean-like waves. We also seek to investigate how buoyancy forces and surface tension affects the character. Furthermore, we are interested in experimenting with leg collision avoidance as well as uneven terrain surfaces. Finally, we would like to look at increasing the reaction time between the change in character velocity and the response change in step speed, to better observe balance and stutter or partial steps.

In summary, we show that we can construct complex locomotion reactions to physical disturbances from a finite database of motion captured animation. This method is less costly than full simulation and sidesteps the need for producing a spectrum of animations for environments with character-fluid interactions.

ACKNOWLEDGMENTS

The authors thank Ioannis Karamouzas as well as our reviewers for their helpful and constructive feedback.

REFERENCES

Y. Abe, M. DaSilva, and J. Popović. 2007. Multiobjective control with frictional contacts. *ACM SIGGRAPH/Eurographics symposium on Computer animation* (2007), 249–258.

Y. Abe, C. Karen Liu, and Zoran Popović. 2004. Momentum-based parameterization of dynamic character motion. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 173–182.

Shailen Agrawal and Michiel van de Panne. 2016. Task-based Locomotion. *ACM Trans. Graph.* 35, 4, Article 82 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925893

O. Arikan, D. A. Forsyth, and J. F. O'Brien. 2005. Pushing People Around. In *Symposium on Computer Animation*. 59–66.

Ronan Boulic, Utku Evci, Eray Molla, and Phanindra Pisupati. 2016. One Step from the Locomotion to the Stepping Pattern. *Proceedings of the 29th International Conference on Computer Animation and Social Agents - CASA '16* (2016), 165–168.

A. Bruderlin and T. W. Calvert. 1989. Goal-directed, Dynamic Animation of Human Walking. *SIGGRAPH Comput. Graph.* 23, 3 (July 1989), 233–242. https://doi.org/10.1145/74334.74357

Yunus Cengel and John Cimbala. 2006. Fluid Mechanics: Fundamentals and Applications. McGraw Hill, New York, NY, USA.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust task-based control policies for physics-based characters. *ACM Trans. Graph.* 28, 5 (2009), Article 170.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized biped walking control. *ACM Trans. Graph.* 29, 4 (2010), Article 130.

Stelian Coros, Philippe Beaudoin, Kang Kang Yin, and Michiel van de Pann. 2008. Synthesis of constrained walking skills. *ACM Trans. Graph.* 27, 5 (2008), Article 113.

Marco da Silva, Yeuhi Abe, and Jovan Popović. 2008. Interactive simulation of stylized human locomotion. *ACM Trans. Graph.* 27, 3 (2008), Article 82.

Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. 2010. Feature-based locomotion controllers. *ACM Trans. Graph.* 29, 4 (2010), Article 131.

Michael Girard and A. A. Maciejewski. 1985. Computational Modeling for the Computer Animation of Legged Figures. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*. ACM, New York, NY, USA, 263–270. https://doi.org/10.1145/325334.325244

Radek Grzeszczuk and Demetri Terzopoulos. 1995. Automated Learning of Muscle-actuated Locomotion Through Control Abstraction. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 63–70. https://doi.org/10.1145/218380.218411

Daseong Han, Seokpyo Hong, Junyong Noh, Xiaogang Jin, and Joseph S Shin. 2016. Online real-time locomotive motion transformation based on biomechanical observations. *Computer Animation and Virtual Worlds* 27, 3-4 (2016), 378–384.

Satoru Ishigaki, Timothy White, Victor B. Zordan, and C. Karen Liu. 2009. Performance-based Control Interface for Character Animation. *ACM Trans. Graph.* 28, 3, Article 61 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531367

T. Komura, H. Leung, and J. Kuffner. 2004. Animating Reactive Motions for Biped Locomotion. In *ACM Symp. on Virtual Reality Software and Technology*. 32–40.

Taesoo Kwon and Jessica Hodgins. 2010. Control Systems for Human Running using an Inverted Pendulum Model and a Reference Motion Capture Sequence. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*. 129–138.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven biped control. *ACM Trans. Graph.* 29, 4 (2010), Article 129.

Adriano Macchietto, Victor Zordan, and Christian R. Shelton. 2009. Momentum control for balance. *ACM Trans. Graph.* 28, 3 (2009), Article 80.

A. Majkowska and P. Faloutsos. 2007. Flipping with physics: motion editing for acrobatics. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*. 35–44.

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. 2010. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph.* 29, 4 (2010), Article 71.

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.* 28, 3 (2009).

Nam H. Nguyen, Raul Arista, C. Karen Liu, and Victor Zordan. 2012. Adaptive Dynamics with Hybrid Response. In *SIGGRAPH Asia 2012 Technical Briefs (SA '12)*. ACM, New York, NY, USA, Article 5, 4 pages. https://doi.org/10.1145/2407746.2407751

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Trans. Graph.* 35, 4, Article 81 (July 2016), 12 pages. https://doi.org/10.1145/2897824.2925881

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.* 36, 4, Article 41 (July 2017), 13 pages. https://doi.org/10.1145/3072959.3073602

Ari Shapiro, Fred Pighin, and Petros Faloutsos. 2003. Hybrid Control for Interactive Character Animation. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications (PG '03)*. IEEE Computer Society, Washington, DC, USA, 455–. http://dl.acm.org/citation.cfm?id=946250.946930

Weiguang Si, Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. 2014. Realistic Biomechanical Simulation and Control of Human Swimming. *ACM Trans. Graph.* 34, 1, Article 10 (Dec. 2014), 15 pages. https://doi.org/10.1145/2626346

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph.* 26, 3 (2007), Article 107.

Seyoon Tak and Hyeong-Seok Ko. 2005. A physically-based motion retargeting filter. *ACM Transactions on Graphics (TOG)* 24, 1 (2005), 98–117.

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. 2011. Articulated Swimming Creatures. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. ACM, New York, NY, USA, Article 58, 12 pages. https://doi.org/10.1145/1964921.1964953

Jerry Tessendorf. 2004. Interactive Water Surfaces. *Game Programming Gems 4, Charles River Media* (2004).

Jerry Tessendorf. 2014. eWave: Using an Exponential Solver on the iWave Problem. *Technical Report, Clemson University* (2014).

Xiaoyuan Tu and Demetri Terzopoulos. 1994. Perceptual Modeling for the Behavioral Animation of Fishes. In *Proceedings of the Second Pacific Conference on Fundamentals of Computer Graphics (Pacific Graphics '94)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 185–200. http://dl.acm.org/citation.cfm?id=212531.212550

Unity3D. 2018. http://www.unity3d.com/.

Kevin Wampler and Zoran Popović. 2009. Optimal Gait and Form for Animal Locomotion. *ACM Transactions on Graphics* 28, 3 (2009), Article 60.

Chun-Chih Wu, Jose Medina, and Victor B. Zordan. 2008. Simple Steps for Simply Stepping. In *Proceedings of the 4th International Symposium on Advances in Visual Computing (ISVC '08)*. Springer-Verlag, Berlin, Heidelberg, 97–106. https://doi.org/10.1007/978-3-540-89639-5_10

Chun-Chih Wu and Victor Zordan. 2010. Goal-directed stepping with momentum control. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 113–118.

Jia-Chi Wu and Zoran Popović. 2010. Terrain-Adaptive Bipedal Locomotion Control. *ACM Trans. Graph* 29, 4 (2010), Article 72.

Yuting Ye and C. Karen Liu. 2008. Responsive Characters with Dynamic Constraints in Near-Unactuated Coordinates. *ACM Transactions on Graphics* 27, 5 (2008), Article 112.

Yuting Ye and C. Karen Liu. 2010. Optimal Feedback Control for Character Animation Using an Abstract Model. *ACM Trans. Graph.* 29, 4 (2010), Article 74.

KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2008. Continuation Methods for Adapting Simulated Skills. *ACM Trans. Graph.* 27, 3 (2008), Article 81.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. *ACM Trans. Graph.* 26, 3 (2007), Article 105.

K. Yin, D. K. Pai, and M. van de Panne. 2005. Data-driven Interactive Balancing Behaviors. In *PG'05: Proceedings of the 13th Pacific Conference on Computer Graphics and Applications*. 118–121.

Daniel Zimmermann, Stelian Coros, Yuting Ye, Robert W. Sumner, and Markus Gross. 2015. Hierarchical Planning and Control for Complex Motor Tasks. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15)*. ACM, New York, NY, USA, 73–81. https://doi.org/10.1145/2786784.2786795

V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast. 2005. Dynamic response for motion capture animation. *ACM Trans. Graph.* 24, 3 (2005), 697–701.