THESIS REPORT ON:
Real-time Physics-based Animation of a Humanoid Swimmer

PREPARED BY: Jurgis Pamerneckas,
SUPERVISED BY: Arjan Egges and Nicolas Pronost.

Game and Media Technology,
Utrecht University, The Netherlands

August 31, 2014

Abstract

We present a real-time physics-based system for animating humanoid swimming. It includes a simplified fluid environment, a controlled virtual character, and a template for imitating different respiratory strategies. Swimmer actuation is carried out using a proportional derivative control strategy and a set of target poses. Balancing and interactive target tracking are achieved by generating additional torques in the controller feedback loop. Breathing patterns are modeled as a customizable piecewise linear function for torso density variation. After testing the system we find the optimal input parameters and investigate the effects of common breathing patterns on the performance of the swimmer. Our preliminary results suggest that buoyant properties using same breathing techniques might have slight effects on front-crawl and backstroke style performances, but are generally negligible.

Contents

1 Introduction **1**
- 1.1 Animating Swimmer Motions . . . 3
- 1.2 Framework for Swimmer Motion and Respiration Modeling . 4

2 Related work **5**
- 2.1 Overview . . . 5
- 2.2 Biomechanics of Swimming . . . 5
  - 2.2.1 Respiration . . . 6
- 2.3 Character Animation and Environments . . . 6
  - 2.3.1 Physics-based Character Control . . . 7
  - 2.3.2 Physics-based Fluid Animation . . . 9
  - 2.3.3 Articulated Creatures in a Fluid Environment . . . 10
- 2.4 Simplified Fluid Environment . . . 11
- 2.5 Summary . . . 12
- 2.6 Report Structure . . . 12

3 Model Design and Simulation **13**
- 3.1 Overview . . . 13
- 3.2 Virtual Swimmer Character . . . 13
  - 3.2.1 Body parts . . . 13
  - 3.2.2 Joints . . . 15
- 3.3 Simplified Aquatic Environment . . . 16
  - 3.3.1 Limb Subdivision . . . 16
  - 3.3.2 Collision Volume . . . 17
  - 3.3.3 Simulated Properties and Water Forces . . . 18
- 3.4 Motion Control . . . 20
  - 3.4.1 Styles and a Finite State Machine . . . 20
  - 3.4.2 Proportional-Derivative (PD) Controller . . . 21
  - 3.4.3 Balancing and Interactive Target Tracking . . . 22
- 3.5 Breathing Model . . . 23
  - 3.5.1 Breathing Patterns Design . . . 23

3.5.2 Piecewise Linear Function for Respiration Modeling . 25
3.6 Summary 25

4 Implementation and Analysis **26**
4.1 Overview 26
4.2 System Implementation 26
4.2.1 Software and Hardware 26
4.2.2 Software-specific Design 26
4.2.3 System Architecture 27
4.3 System Analysis 29
4.3.1 Motivation and Goals of the Analysis 29

5 Experiment **31**
5.1 Parameter Testing and Results 31

6 Conclusions, Limitations, and Future Work **37**
6.1 Conclusions 37
6.2 Limitations 38
6.3 Future Work 39

A Resulting animations **41**

References **44**

Chapter 1

Introduction

The interest among humans to represent motion dates back to about 30.000 years ago. Contemporary Palaeolithic researcher theories attempt to explain certain cave paintings as early forms of stop-motion pictures or even the origins of cinema [AR12]. Today, such interest does not seem to have diminished, but is advancing significantly. In particular, simulation of human motion evokes interest in disciplines such as computer animation, robotics, control theory and so on. Anthropomorphic characteristics are used to personalize creatures acting in various environments or scenarios. However, designing motion controllers and replicating human behaviors is still a challenging problem. One way of solving these issues is to incorporate an emerging concept known as *physics-based animation*.

A specific form of human motion comes into existence when immersed in an aquatic environment. Swimming, as a sport, can be considered as one of the most healthy and useful exercises for a human body and mind. The main advantages of swimming are that during strokes, almost all muscle groups are actively participating thus increasing cardiovascular fitness, core strength, posture and muscle definition. Also, the relatively little strained motions reduce the risk of muscle, joint injuries. In addition, psychosomatic research [BO83], [BGPB97] indicates, that swimming, if not abused, reduces tension, depression, anger, confusion and causes more vigor after exercising than before. Nevertheless, it can be a daunting task to teach a person to swim, because particular muscle groups have to be activated at the certain time. Dictating muscle activation time is not a trivial task for a brain that does not yet hold the automatic memory of such motions. The inability to see oneself from a third person perspective makes it more difficult to identify faults made during strokes, especially for novice swimmers. Therefore, it would be useful to have an interactive teaching tool, that would allow instructors to input swimmer motions so they could compare theirs with the

recommended ones[1]. Swimmers could then identify mistakes, correct them and eventually achieve better performance.

Figure 1.1: Hardware and software aids swimming coaches to provide swimmers with advices.

Next to teaching people how to swim, another motivation for the research could be to find out more about the efficiency of certain swimming styles, and also how that relates to physics properties such as the proportions of a person, how the person breathes, and efficiency of fast/slow swimming. Respiration is a vital physical activity and requires a lot of coordination to synchronize movements while maintaining a rhythmical breathing pattern. If respiration does have an effect on the performance of the swimmer, it may be desirable to include such feature into the design of the tool.

The swimming economy profile is a function of oxygen uptake [Tro99]. A special type of training, called "hypoxic", is employed by athletes to better cope with the regular demand for oxygen. During such training, depending

[1]Figure 1.1 courtesy: top image - QualisysAB http://www.qualisys.com/applications/biomechanics/swimming/, bottom images - Mar-Systems http://www.mar-systems.co.uk/swimming-cameras

on the style, instead of ventilating on every stroke cycle, breathing can be completed every second or third stroke cycle. Even though such training is known to minimize the energy expenditure and increase endurance, it is not yet clear whether the change in buoyant properties due to breathing affects a swimmer's performance. If this is the case, physics-based animation could provide a simulated environment to investigate such circumstances. Therefore, we attempt to simulate those conditions and seek to replicate respiratory patterns that swimmers make use of, in order to determine whether it has an effect on their performance.

There are several different types of professional swimming, e.g. water polo, synchronized, competitive swimming. In this work, we endeavor to model movements of a competitive swimmer, and make use of physically based animation to investigate the single most important physical change happening over the duration of the swim - breathing.

1.1 Animating Swimmer Motions

Scholars and game developers make use of physics-based animation systems that are able to reproduce various animal behaviors, simulate legged creatures that maintain balance, traverse, dance, use their body topology to maneuver, steer, avoid collisions and so on. Furthermore, computer animation can be used to simulate an extensive list of natural phenomena, such as fire, smoke, viscous liquids, snow, wind, dynamic earth terrain etc. In the context of simulating a swimming character several situations could potentially benefit from such system. Firstly, the inability to see oneself from a third person perspective makes it difficult to identify the mistakes made during strokes, especially for novice swimmers. Secondly, a swimming instructor observing drawbacks of the athlete needs to modify the motion immediately to be able to display it. Using accurate water simulation is still computationally expensive. In this case, a simplified real-time alternative solution, imitating basic physical properties could be useful. We propose a model, that could be used as a solution for both of these issues:

- A founding iteration for a serious game that could help swim-instructors or teachers to provide feedback for beginner swimmers. It could be used as a tool to point out the differences between the actual swimmer motion and the modeled one. In addition, physical properties of the character body allow us to model and highlight the importance of breathing patterns and possibly determine the optimal strategy.
- A simplified way to represent water environment, that should fit the real-time constraint. Submerged static objects would float/sink, depending on their physical properties, whereas articulated ones would

generate drag forces, granting characters the ability to swim. Moreover, it would not be limited to swimming characters, but could also provide this ability to traverse water for a wider variety of moving objects.

1.2 Framework for Swimmer Motion and Respiration Modeling

The primary objective of this work is to introduce an efficient way to simulate a swimming character, who is able to breathe. To satisfy this objective, the system should contain an aquatic environment, have a character control mechanism replicating competitive swim styles, and run at interactive rates. Swimmer body parts are created using physical properties, such as density and mass so that they can be used to model customizable breathing patterns. Finally, to represent system capabilities to work in real-time, some form of interaction needs to be incorporated, such as following the changing target position, specified by the user.

Chapter 2

Related work

2.1 Overview

Simulating a virtual swimmer in a physically based environment is a combination of at least two fields. Firstly, such an investigation is related to biomechanics of swimming, since the motions should resemble those of a real swimmer. Secondly, it accounts for physics based animation, which then branches into manipulation of virtual characters and simulation of their physical surroundings or used in combination of both. Existing research can be found in both of these areas and a related review will be presented in the following subsections.

2.2 Biomechanics of Swimming

A report by Barbosa et al. [BPC+09] points out that there was a consistent increase in number of papers per year on swimming "science" published within the span of 1971 to 2006 (ranging from 23 papers in 1971 to 145 manuscripts in 2006). The purpose of consulting such research is two-fold. Firstly, it would help to define the fluid and the ways it affects the swimmer. Secondly, it would aid in constructing the competitive swimming styles that would be performed by a humanoid character.

We start by looking at the way swimming science characterizes the activity. Maglischo [Mag03] describes competitive swimming as a unique sport, in which athletes move while suspended in a fluid medium and must propel their bodies forward by exerting forces against liquid rather than solid matter. Two major disadvantages are created in comparison to land sports. Firstly, water presents less resistance to propulsive efforts, than e.g. for runners, who can rely on ground contacts to immediately push against. Secondly, water has substantially higher density than air. Thus, compared to traversals surrounded by air, swimmers experience a lot more resistance.

Nevertheless, such conditions present very specific circumstances, when muscles have to perform in a smooth, continuous and non-disrupted way. This greatly reduces chances of muscle injuries and develops them in a gradual way.

According to Barbosa et al. [BMCS11], a large part of biomechanical analysis of competitive swimming is dedicated to the four competitive strokes: front crawl, backstroke, breaststroke, and butterfly. Athletes are allowed to compete in these styles, recognized by *FINA* (Fédération Internationale de Natation) or "International Swimming Federation", during the Summer Olympic Games. Barbosa et al. [BMCS11] perform a characterization of stroke mechanics for every style and report the relationships established between the strokes and the ways they might influence the performance.

2.2.1 Respiration

Breathing is an indispensable part of swimming and is a determining factor of whether a human floats or sinks. Since we want to create a respiration model for the swimmer, we should have numerical data to support this physical change. Bostanci et al. [BSS+13] conduct a study on 20 elite swimmers. They use stereological methods to measure the average lung volumes to be $7508.15 \pm 800.92 cm^3$ during expiration and $13292.11 \pm 1256.03 cm^3$ during inspiration. Clauser et al. [CMY69] measure anthropometric data of 14 segments of the body on 13 male cadavers and report a mean torso volume to be $32691 \pm 486 cm^3$. Wooten and Hodgins [WH96] simulate divers with a torso mass of $29kg$. We combine this data to extract the mean peak torso densities (i.e. $610 kg/m^3$ when inhaled and $710 kg/m^3$ when exhaled) of elite swimmers during breathing and use them in our simulation.

Furthermore, we want to model different breathing strategies and analyze their effects on swimming performance. A complete overview of such pattern effects on performance during different swim styles could not be found. However, Pedersen and Kjendlie [PK06] recommend to breathe as little as possible during 50m and no more than every 3rd stroke during a 100m front crawl race.

2.3 Character Animation and Environments

Real-time character animation can be achieved using several existing techniques. However, one shared aspect is that the virtual human is commonly represented by a model with an underlying *skeleton*. A skeleton is a

set of body segments connected through *joints*. These connections form a data hierarchy that comprise a *pose* of the character. We also employ this concept and will describe it in the corresponding subsection.

Van Welbergen et al. [VWVBE+10] study and report the trade-offs that various animation methods carry in terms of naturalness and the controllability. Reported techniques are responsible to articulate (or actuate) the skeleton and are using different concepts to do so. According to Van Welbergen et al. [VWVBE+10], character motions can be data-driven (e.g. motion-captured), based on a physical model, generated procedurally (e.g. using mathematic formulas to describe a kinematic motion). Depending on the desired results, combinations of methods are also possible. In section 2.3.1 we will overview some common physics-based character animation techniques and use one of them throughout the remainder of the work, mostly because it suits the purpose of modeling physical changes in human bodies during the swim, facilitates the need to incorporate aquatic forces required for locomotion, and can be performed in real-time.

The extensive field of research of fluid dynamics can be of help in understanding and address the objective of simulating an aquatic environment.

The extensive field of research of fluid dynamics can be of help in understanding and address the objective of simulating an aquatic environment. Given the wide practical use and many years of applied research, which still has open problems and is ongoing, it is considered among the most important problems in mathematics. There are various ways to model the dynamics of fluids, however all are still governed by the *Navier-Stokes* equations introduced several hundreds of years ago. These equations depict the fluid phenomena and numerical methods are typically only approximate solutions. A separate subject of *Computational Fluid Dynamics (CFD)* is established for discretizing the problem. The purpose of physically-based fluid animation differs from CFD in a way that it focuses on generating plausible visual effects and sacrifices accuracy for computational speed. In section 2.3.2 we will introduce some basic concepts of physically-based fluid animation and overview the advantages and disadvantages of commonly used methods merely as a mean to rapidly refer a reader, in case one wishes to extend this work and improve the naturalness of the fluid simulation, especially because real-time alternatives have recently been established.

2.3.1 Physics-based Character Control

From the data-driven animation research perspective, Pejsa and Pandzic [PP10] argue, that physics-based character animation approaches still do not depict the required expressiveness and naturalness of the motions. However, they agree that such methods allow more control over the motions

and produce much more realistic results in the cases of simulating *passive* phenomena, such as cloth, smoke, fire, water or rag doll physics in character unbalance or death situations. Hertzmann and Zordan [HZ11] provide reasons why physics based approaches are re-gaining popularity when used to actuate characters. Mostly it is due to the fact that the quality and flexibility of data-driven methods depends on the quality and quantity of recorded motions. Since capturing all combinations is impossible, it leads to unnatural behavior and/or reduced control. Interactive physics based character animation research is still approaching the visual quality standards set by example-based techniques and is not yet used commercially. Nevertheless, it is now becoming obvious that the increased researcher interest (as reported by Geijtenbeek and Pronost [GP12]) will have an influence on the way virtual characters act and behave and will most likely be utilized in video games.

In physics based character animation, motion control can be carried out using several techniques. Wrotek et al. [WJM06] use *external forces* in combination with motion capture data to increase stability and balance of fighting characters. Geijtenbeek and Pronost [GP12] describe *virtual force* actuation method as a control abstraction that works by computing external forces, which are realized through the application of internal joint torques. Coros et al. [CBvdP10] demonstrate this technique for various humanoid walking styles and present an interactive tool to easily customize character proportions and motions. *Muscle actuation* systems generate joint torques through tendons attached to the bones of the skeleton. An example of actuation method through muscle contractions is found in the work of Geijtenbeek et al. [GvdPvdS13]. Their controller is adopted to various bipedal character morphologies through evolutionary algorithms.

An *internal joint torque* method is the most straightforward among the above mentioned techniques. To generate torques and articulate character limbs, a controller is updated with information from a physical simulation (through *sensors*), which is then used together with desired pose information (through *actuators*). Such motor controllers at joint positions act as spring-damper systems and are usually referred to as *Proportional-Derivative (PD)* controllers. In one of the seminal works Yin et al. [YLvdP07] create balanced humanoid walking motions either from motion capture data or from a small set of input parameters. They use PD controllers together with a finite state machine and make them robust against various strength external pushes or terrain changes. Tan et al. [TLT11] predict the positions and velocities of the limbs at the next time frame and use them in combination with the current time step to minimize the oscillation of the controller errors.

2.3.2 Physics-based Fluid Animation

The enormous complexity of fluid dynamics makes it impossible for the animation artists to illustrate fluid motion frame by frame, which is why physics-based methods are becoming widely used techniques for generating realistic animations. Existing methods are most commonly distinguished to be either *Lagrangian* or *Eulerian*. We include a short description and comparison of these different approaches and mention some *coupling* techniques to animate interplay between solid objects and fluids.

Lagrangian method

Originally, Gingold and Monaghan [GM77] designed the Smooth Particle Hydrodynamics (SPH) method to simulate the flow of interstellar gas. In the SPH, fluid medium is represented by a set of particles that hold material properties and interact with each other within the range controlled by a radially symmetric smoothing kernel. The goal during a simulation step is to compute the new state of the fluid based on the current quantities, such as density, pressure or viscosity. Stam and Fiume [SF95] adopted this method for computer graphics. Müller et al. [MCG03] disposed of mass conservation equations, and convection terms, which allowed them to demonstrate SPH ability to perform at interactive rates. More recent works by Macklin and Müller [MM13], and Akinci et al. [AIA+12] showed great promise by improving realism and greatly reducing computational expenses.

Eulerian method

Harlow and Welch [HW65] investigated the dynamics of an incompressible viscous fluid and presented a time-dependent calculation method for a flow with a free surface. Their work inspired many further researches to design systems that sample the fluid domain space at fixed positions, and determine how the properties (e.g. velocities, density, temperature) change in time. In other words, governing equations are evaluated at discrete voxel grid positions, thus defining the state of the fluid. The idea was to store fluid scalar quantities (pressure, temperature) at the centers of the grid cells and vectors (velocity) at the edges. Alternatively, Stam [Sta03] improved efficiency by disregarding this distinction, and stored all information at the same grid cell node. Similarly to other Eulerian viewpoint researchers, Foster and Fedkiw [FF01] combined it with the level set tracking method to track and represent the surface.

Comparison

Lagrangian method has several advantages over the *Eulerian method*. Most notably, due to the existence of multiple particles it is straightforward

to conserve mass, depict splashes, bubbles and foams. Additionally, recent approximations have made it less computationally intense. Nevertheless, the complexity is still too sizable to be able to use it in games or other interactive systems, especially when there is a need to incorporate fluid-solid interactions. Another disadvantage is that it is necessary to choose a stable, accurate, fast or even multiple smoothing kernels. Moreover, due to the particle-based nature there is the problem of constructing a rendering surface, and it is not straightforward to enforce the fluid incompressibility. Alternatively, *Eulerian method* has the advantage of having smooth surfaces and the ability to perform well under large time steps. However, the disadvantages include the immense increase in computational cost due to finer grid resolutions, and aliasing artifacts at slanted solid object boundaries due to the axis aligned fluid grids.

Coupling

The interaction that exists between solid objects and fluids has been previously investigated. Three major coupling strategies exist: one-way solid-to-fluid coupling (as in the work by Enright et al. [EMF02]), one-way fluid-to-solid coupling (as in the work by Foster and Metaxas [FM96]), and two-way coupling (as in the work by Carlson et al. [CMT04]). The distinction between these methods can be identified just by reading into their names. In the first case a solid affects the fluid, but not vice-versa. In the second case a fluid affects the solid, but not vice-versa. Finally, in the last case both fluid and solid have an effect on each other.

2.3.3 Articulated Creatures in a Fluid Environment

In reality, moving objects exert forces enabling them to traverse aquatic environments. In this section, we review several works that animate either human, animal or imaginary creature locomotion in simulated fluid environments. Yang et al. [YLS04] focus on controlling a humanoid character to animate a few competitive swimming styles. Their breaststroke style

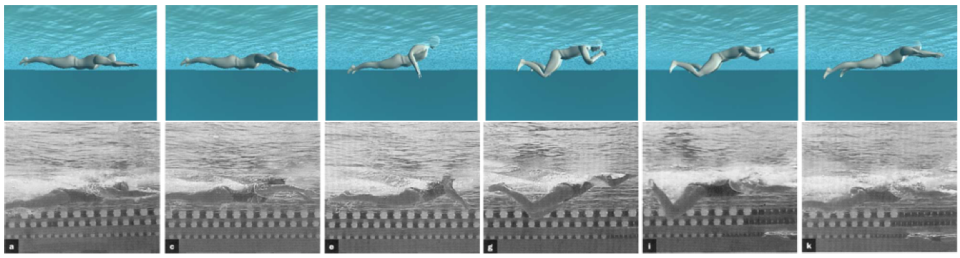

Figure 2.1: Yang et al. [YLS04] demonstrate a synthesized breast stroke compared to a real swim sequence by Maglisch [Mag03].

animation can be seen in figure 2.1. Authors use a layered control mechanism and achieve interactive rates by incorporating a simplified approach for simulating water forces. Our model is somewhat similar, because we also aim to achieve a system that is able to perform in real-time. However, we extend their approach by incorporating a breathing model and analyze the effects it has on the swimmer performance. In another work, Kwatra et al. [KWC+10] center their attention around an accurate fluid representation, but also include an articulated humanoid model. Human swimming motions are captured in dry laboratory conditions with the help of external support cords. Data is then used to drive a humanoid character, as depicted in figure 2.2. Finally, two-way coupling technique is used to simulate interac-

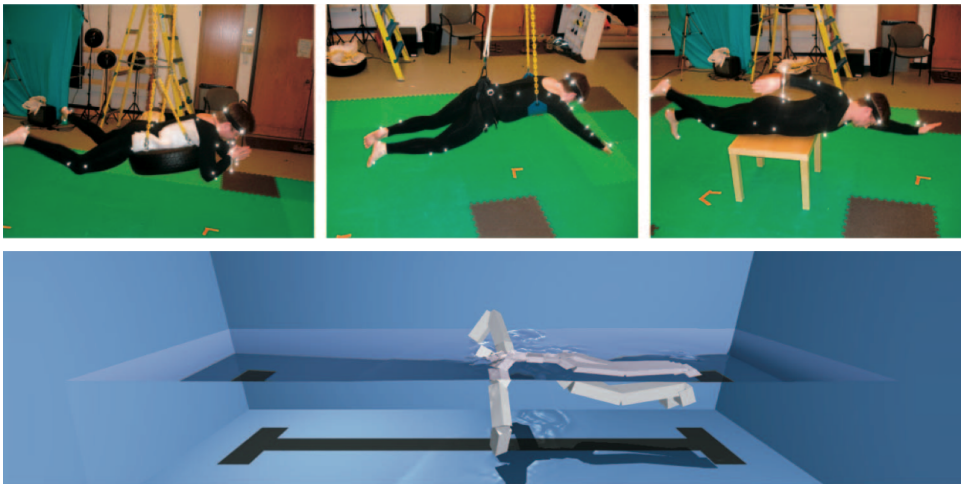

Figure 2.2: Kwatra et al. [KWC+10] motion capture swimmer motions and transfer them onto a virtual character.

tion between the solid and fluid. Lentine et al. [LGS+11] adopt an Eulerian viewpoint with a two-way coupled fluid system, together with PD-controlled reactive characters. They demonstrate creature ability to propel through fluids by exploiting minimization and maximization of the drag force. Tan et al. [TGTL11] also use eulerian approach with two-way coupling to animate a fish, a frog, a tortoise, and a manta ray. Creature control is maintained using periodic functions to determine joint torques. Offline optimization on a set of control functions is used to determine motions, that allow creatures to swim straight and are least energy consuming.

2.4 Simplified Fluid Environment

One of the goals of this thesis is to animate physical effects that water has on the swimmer body. However, we do not seek to animate any fluid motion (e.g. splashes, streams, turbulence) caused by the swimmer. There-

fore, we take a somewhat simplified inverted Eulerian viewpoint. Instead of discretizing the environment and evaluating fluid changes at those discrete positions, we subdivide virtual character body parts into volumetric components, centers of which will react to the water environment. Thus, in a way, we define a one-way fluid-to-solid interaction. It is important to incorporate a force to gradually suspend/stabilize the character at a certain height by counteracting the gravity force. In addition, it is necessary to include a force to demonstrate effective traversals in such environment. Therefore, these following environmental forces are taken into account: *buoyancy* (e.g. as in the work of van Rijn [vR90]) and *drag* (e.g. as in the work of Toussaint [Tou02] or Barbosa et al. [BMCS11]).

2.5 Summary

The goal of our work is to create a realistic virtual swimmer that can be used in real-time applications, because the existing techniques that attempt to do it are not suitable for games (eg. Kwatra et al. [KWC+10] or Tan et al. [TGTL11]). This allows us to address other issues, such as adding a breathing model, which is missing from the current state of the art. Nonetheless, it is likely that such approximations will reduce the realism of the simulation and may not reveal the breathing effects to their full extent. Yet, if the significant differences of respiratory technique influence on the swimming performance exist, our model should disclose these to a certain extent.

2.6 Report Structure

The remainder of this report is structured as follows. System modeling, design details and simulation pipeline are described in Chapter 3. Further, technical, implementation details and input parameters are discussed in Chapter 4. Then, a report on system performance under various parameter combinations is presented in Chapter 5. Lastly, some conclusions, limitations, and suggestions for further research avenues can be found in Chapter 6.

Chapter 3

Model Design and Simulation

In this chapter we present our physics-based animation framework and detail how each component contributes to display swimmer motions in aquatic environment.

3.1 Overview

The following sections 3.2 and 3.3 present the two major components of our physics-based framework: *a swimmer character* model and *a simplified aquatic environment*, respectively. Further, section 3.4 describes a chosen technique for *motion control* and some additional heuristics for keeping the character in balance. Moreover, section 3.5 reports how patterns for virtual swimmer breathing can be created and modified to imitate different respiratory strategies. Finally, section 3.6 summarizes the model design and simulation in a concise manner.

3.2 Virtual Swimmer Character

In automotive design, crash test dummies are employed as anthropomorphic objects to replicate reactions during various impacts. In virtual worlds, we can make use of rag-doll physics to simulate certain physical responses or behaviors. Therefore, we employ a similar actor concept for animation purposes.

3.2.1 Body parts

To begin with, we design a simplified version of a human body. Yang et al. [YLS04] model their swimmer using 16 segments. We use a similar

structure for our character. However, we exclude the neck region, as we seek to have a simplified model and it is not so significant when it comes to simulating propelling characters. Therefore, we end up reducing the number of segments to 15.

Shapes and Masses

The aim to create a system that could animate swimmer motions at interactive rates is one of the reasons behind a choice to use approximate body segment shapes. Hence, a character is constructed mostly using trapezoidal-prism shaped parts. Next, we need to establish some consistency of measurement units between a design and a physics engine. To determine the behavior, engine specifications require every "physical" object in the scene to be approximately the weight and size of his real-world counterpart. Thus, we first set the *mass* property for all rigid segments. Because swimmers generally do not seem to be neither significantly heavier nor lighter than average humans, we employ cadaver body segment mass data m_p (found in the work of Clauser et al. [CMY69]) to set-up our character weight.

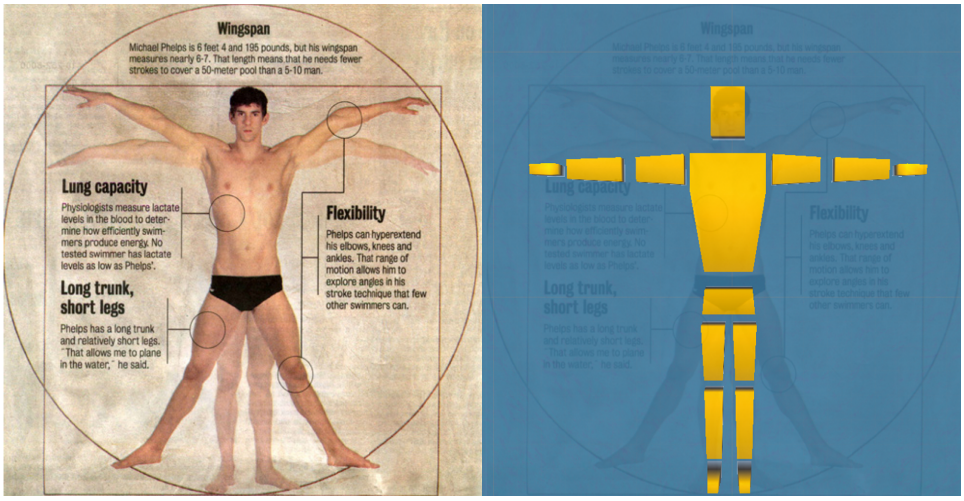

Figure 3.1: Michael Phelps and an approximate virtual swimmer model with their body part proportions.

Proportions

To take into account the proportions of our character, we assume an anthropometry of a competitive swimmer. Therefore, we adopt the average characteristics of 12 top-rank elite swimmers, found in McArdle et al. [MKK10]. To emphasize the possible effects on propulsion, we increase the size between the tips of swimmer arms by taking into account the arm-span measurements of 8 gold medal winner at 2008 Beijing Olympic games,

Michael Phelps. He weighs 88 kilograms and is 1.93 meters tall. As McArdle et al. [MKK10] clarify, Phelps' arm-span exceeds his stature by almost 10 centimeters and a large torso-leg ratio helps to explain the extra thrust generated while swimming.

Hierarchy of Transformations

To create a swimmer skeleton, we position the hip segment at the beginning of the world coordinate frame. Then, we arrange the remaining segments around it, such that it would resemble a complete body of an elite swimmer mannequin (see figure 3.1). Further, we form a kinematic chain of objects, such that the world coordinates of the ones appearing further away from the hip, would be transformed into the local coordinates of the rigid parts closer to it. This way, a hierarchy is created, so that every part, that is connected to pelvis, creates a local coordinate frame on its' own. By repeating this procedure all the way down to the palms and feet, we form a complete limb relationship representation. This is useful, because when a shoulder joint gets rotated, it affects both the forearm and the palm. It is important to assure, that scale and rotation values throughout the components are consistent, thus do not cause affine, skewed or other distortions upon transformations.

3.2.2 Joints

The next step is to present our transformation hierarchy to an empty physical environment, where the physical laws get applied. Much like human body segments, which are entwined together through tendons, ligaments, joints and muscles, virtual character body parts have their own alternatives. To describe how parts should move in space together, we specify their degrees of freedom by creating *kinematic pairs*. A kinematic pair is a connection between two bodies that imposes constraints on their local movements. Such connection is called a *joint* and can have various properties that define its behavior. This link is established by attaching a joint component and assigning a parent object to all body segments, excluding pelvis, which is at the top of the hierarchy and does not have a parent. One of the main properties is a *joint limit*, which constrains character limb rotations to approximate human motion. Limiting joint rotations prevents character limbs from rotating towards non-realistic orientations. By establishing all connections and setting all parameters, we create a character having 13 joints and a total of 27 degrees of freedom that connect all body parts together and define the appropriate spatial configurations of the swimmer figure. Table 3.1 presents a list of the joint limit parameters and Figure 3.2 shows the positioning of joints throughout the character.

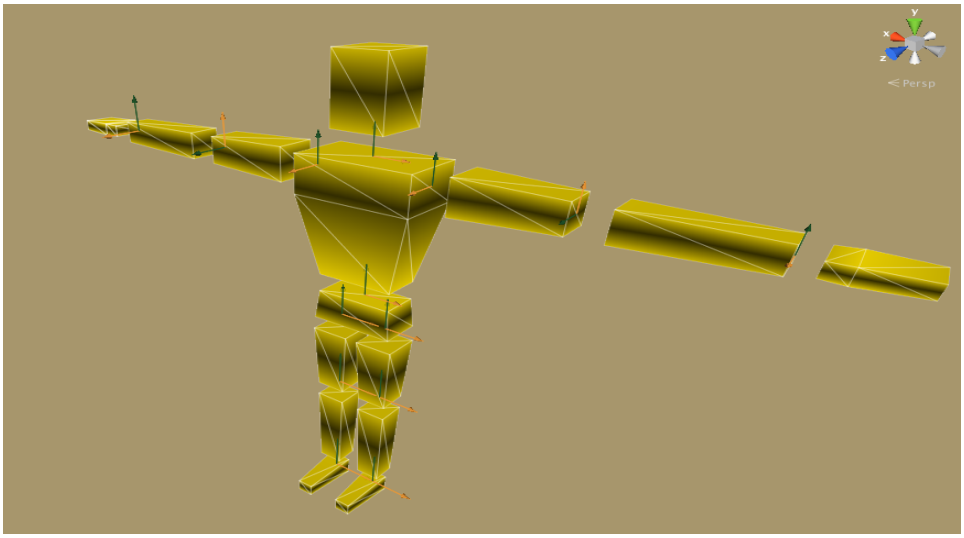

Figure 3.2: Arrows represent joint positions and axes of rotation.

Table 3.1: Joint rotation limits of the 3D model, similar to Faloutsos et al. [FvdPT01].

Joint	Axis of rotation	Lower limit	Upper limit
Waist	x/y/z	-45/-50/-70	90/50/70
Neck	x/y/z	-50/-80/-60	90/80/60
Left and Right shoulders	x/y	-90/-80	90/160
Left and Right elbows	x	0	120
Left and Right wrists	x/y	-45/-90	45/90
Left and Right hips	x/y	-120/-20	20/20
Left and Right knees	x	0	165
Left and Right ankles	x	-30	100

3.3 Simplified Aquatic Environment

To simulate a behavior of underwater objects, we make use of the physics engine concept, called a *collider*. A collider is a component that allows an object it is attached, to react to other colliders. The way system reacts on objects upon collisions defines our simplified aquatic environment.

3.3.1 Limb Subdivision

At this point, the character is fully defined and running the simulation would result in rag-doll physics behavior. Further, it is necessary to prepare character limbs for interaction with fluid environment. We do this by subdividing every limb into relatively small volumetric components (voxels),

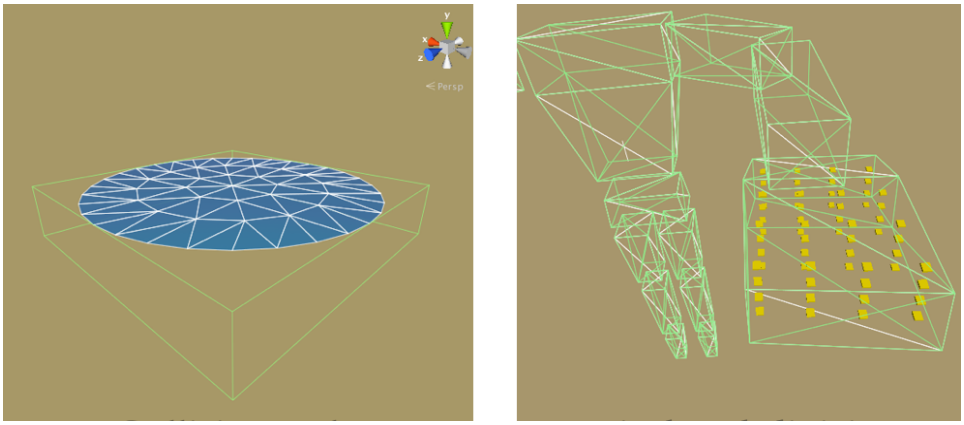

Figure 3.3: A "swimming pool" and the palm of the left hand, prepared for interaction with water.

each of which is responsible for keeping specific data (i.e. position, volume, linear velocity). A variable subdivision resolution is ensured by requiring a single user input parameter v to serve as a number of voxels per axis. Such voxel-based approach will in principle allow any convex body shape to be modeled, provided that the resolution is high enough. However, higher resolutions will impact the performance, as we will see later in section 5.1.

During the initialization, we divide the scale of the chunk on each axis by the number of voxels to determine their sizes. Then, voxels are positioned within the part, such that the consecutive ones are apart by the size of a single voxel on each axis. Yellow cubes on the right hand side of Figure 3.3 depict the centers of voxels. Because we want to simulate certain physical changes in our swimmer, we make sure that torso voxel volumes updated upon a density change. This way, altering densities will not change the visual appearance of the swimmer, but only the physical behavior.

3.3.2 Collision Volume

Fluid environment is modeled as a rectangular prism, which is enclosed by a collider (see left hand side of Figure 3.3). When a subdivided collider that is attached to character segment enters the fluid collider volume, it is considered to be partly under water. Further, we check whether the voxels within an underwater part are inside the fluid collider to determine how much of that part is immersed beneath the surface of the water. A special collision handling function is called and the water forces are approximated.

3.3.3 Simulated Properties and Water Forces

One way to simulate effective swimming in a fluid environment, is to compute the sum of two forces. Firstly, the *buoyancy force* comes from Archimedes principle and counteracts gravity. We will use it to create an illusion of floating objects. Secondly, the *drag force* is a type of friction, generated on objects moving through fluid medium, such as an arm of a swimmer performing a competitive style. In the following subsections we introduce the physical properties that are required to define these fluid forces. We explain how these forces are related to character physical appearance and motion.

Density and volume

To establish the environmental forces acting on the virtual swimmer during the swimming, we first model their dependency on the density of the body part and determine the volumes of each voxel.

So far, we have defined a mass m_p and the size/type of collider. We assign the densities ρ_p for each rigid part using cadaver mass and volume information. Having in mind that there are v voxels on every axis, we find the volume of each voxel V_{vox} within a rigid body part by

$$V_{vox} = \frac{\frac{m_p}{\rho_p}}{v^3}.$$

Buoyancy force

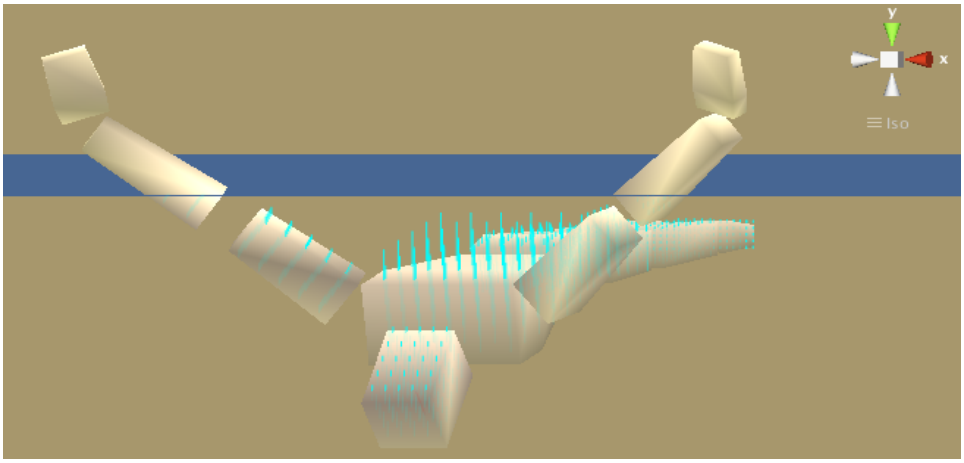

Figure 3.4: Cyan ray length is proportional to an upward buoyancy force. Swimmer torso is more buoyant because of the lower density due to "air" in the lungs.

Buoyancy is defined as an upward force exerted by a fluid that opposes the weight of an immersed object. See figure 3.4. Archimedes' principle states that such buoyant force F_b on a body is equal to the weight of the fluid that it displaces. We express the magnitude of such force as

$$F_b = \rho_f g V_{disp},$$

where ρ_f is the density of the water, g is the gravitational acceleration $9.80 m/s^2$ and V_{disp} is the volume of the displaced fluid. We consider the mandate for competitive swimming water temperature during Olympics. *FINA* (Fédération Internationale de Natation) requires it to be between 25°C to 28°C, yielding ρ_f values between $997.048 kg/m^3$ and $996.237 kg/m^3$. If the voxel is fully submerged, $V_{disp} = V_{vox}$. However, if it is not, $V_{disp} = kV_{vox}$, where $0 < k < 1$ indicates how much of that particular voxel is under water. Specifically, $k = 0$, when a single voxel volume is entirely out of water, and $k = 1$, when voxel is completely submerged.

Drag force

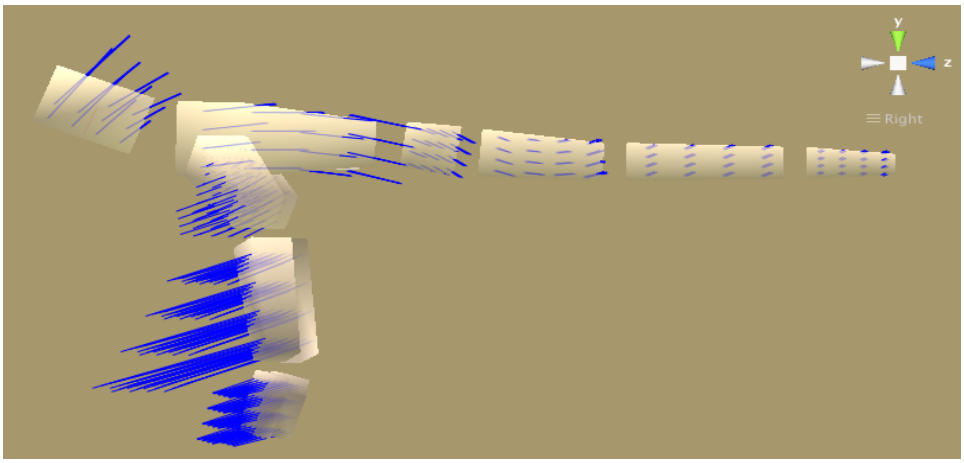

Figure 3.5: Blue ray length is proportional to a drag force, counteracting velocity. Faster moving voxels produce more drag.

After the introduction of the buoyancy force, objects become floaters or sinkers depending on their density property. To simulate propulsion, we need to introduce at least one more force that will act in the opposite direction to which the body part is moving. Assuming there is no fluid flow, a *hydrodynamic drag* force or *fluid resistance* is described as a function of object velocity relative to the object shape, fluid and its parameters. For example, the faster the object is or the greater the exposed surface is, the

larger the drag force. Sport biomechanics literature on swimming [Tou02], [BMCS11] define such force as

$$F_d = -\frac{1}{2}\rho_f v^2 C_d A_p,$$

where v^2 is the squared magnitude of voxel velocity. Sampling velocities at voxel positions yields larger values further away from the point about which the part is rotating, which in turn generates more drag and eventually propulsion. C_d is a constant value depending on the shape of the object. The lesser this value is, the more aerodynamic the shape is. Because we model rigid parts as trapezoidal prisms, we use an approximate drag coefficient (0.8) of an angled cube. However, if we were to incorporate an angle of attack influence on the drag force, we would have to dynamically adjust this value using a lookup table or a similar technique. A_p is the cross sectional area of the shape projected on a plane perpendicular to velocity vector. We approximate this area by multiplying two of the three measurements of a single voxel discarding the smallest value. In most of the cases, voxels are elongated, in which case the other two dimensions are quite similar, thus taking the largest values give us a reasonably good estimation of the area.

3.4 Motion Control

Similar to a human brain, a *closed loop control* contains the knowledge of a motion and dictates how character should behave during simulation. Figure 3.6 displays how controller combines the current situation data from physics simulation and the style knowledge extracted from [NM07]. Control scheme includes the following components: a description of several distinctive swimming styles, a state machine to organize style information, a PD controller to generate motion patterns, a balancing heuristics component to compute additional torques for global objectives, an engine that will advance a simulation one step forward, and a resulting state information.

3.4.1 Styles and a Finite State Machine

Animators use *key-frames* to define the start and the end for any smooth transition. We describe competitive swimming styles using a comparable term: *key-poses*. A key-pose is composed of local orientations for each degree of freedom of every joint. In our system, we assume six character poses per swimming style. This number could possibly be adjusted, since it depends on the motion complexity, length, etc. However, we found it was adequate enough to describe an elaborate, but smooth swimmer motion cycle. Pose information within a style will be used in the controller as desired states of the swimmer.

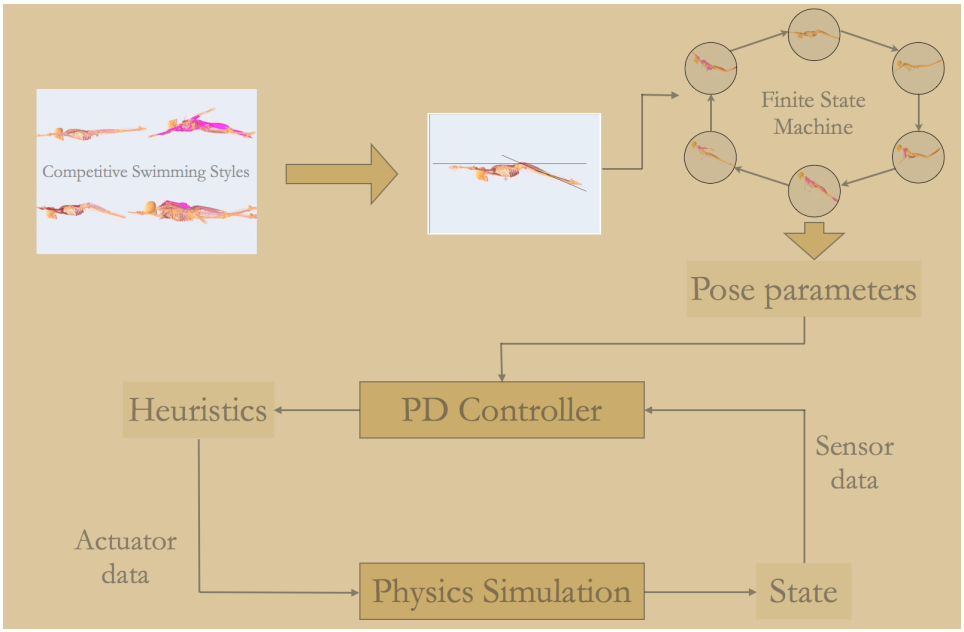

Figure 3.6: Closed loop control scheme of a virtual swimmer.

In our system, the *finite state machine (FSM)* is a computational model to link the execution timing to the desired pose. Using the FSM allows us to address key-poses, define transition order, and duration. The user is allowed to specify how long will it take to execute a single motion cycle. We call this time parameter a *motion cycle duration*. It is then divided by the number of poses to get the duration of each pose, ensuring none of the poses take longer than the others. After the time for a pose runs out, the FSM transitions to the next pose in the list making it the newly desired state of the target pose. However, this type of time-based transitioning does not ensure that the target poses will be achieved entirely. Usually, time runs out before the animation reaches the desired values and acts as a guide for orientation towards which the limbs should rotate. As an alternative to this approach, we attempted to develop a *pose metric* inside the state machine that would allow transitions only when the current and the target poses are similar enough. However, this method appeared to produce visual artifacts, especially because limb asymmetries due to global character orientation would prevent the finite state machine from transitioning and was disposed of.

3.4.2 Proportional-Derivative (PD) Controller

A PD controller is used to combine all the input information and output a torque that is required to advance a current pose of the swimmer towards the key-pose. The torque τ is evaluated on every frame for every joint and

is defined as

$$\tau = k_p(\theta_d - \theta) + k_v(\dot{\theta_d} - \dot{\theta}).$$

Evidently the controller has two terms. The first one accounts for the difference in orientations, whereas the second relates difference in angular velocities. θ_d is the key-pose, retrieved from the FSM and θ is the current pose, sampled from the physics simulation. θ_d changes every time the state machine transitions to the next state. Current angular velocity $\dot{\theta}$ is also obtained by sampling the physics simulation. Target angular velocity $\dot{\theta_d}$ is specified to be zero as this will attempt to reach target pose at lower speeds. Generally, the two controller gains k_p and k_v supervise the responsiveness of the limb behavior.

Finding appropriate values for k_p and k_v is not an effortless task and is commonly achieved through a manual *trial-and-error* process. If the gain values are set too low, the joint behavior appears slow and unresponsive. On the other hand, if they are set too high, motion becomes faster, but stiff and robotic. Gains can not only be low or high, but should also related to each other. Initially we used a recommended [GP12] relationship of $k_v = 2\sqrt{k_p}$, but the lower mass limbs (such as hands or feet) were oscillating rapidly. Consequently, we relate the gain values to the total masses that are attached to that joint. For example, a shoulder gain is associated to the mass of upper arm, forearm and hand segments, whereas knee joint gains depend on the masses of a calf and a foot, etc. This way, the more mass is connected to one particular joint, the more torque a controller is going to produce.

3.4.3 Balancing and Interactive Target Tracking

At this point, if we were to fix all global rotations of a root segment (i.e. pelvis), a character would appear to be swimming. However, disallowing these rotations is much like having a "hand of god" that is helping a character to maintain balance. Therefore, we need to incorporate a technique that would let us release these constrains. Additionally, even though competitive swimmers should only swim straight, we want to demonstrate the system ability to perform in real-time, hence we include an option to specify the target position that the character will follow.

Balanced swimming could be attained by continuously monitoring pelvis rotations and acting accordingly. The heuristic we apply here is that to stabilize themselves, real swimmers counterbalance the unwanted rotations by "turning" slightly at their waist. Similarly, we even out pelvic rotations on the "roll" axis by rotating the torso a little in the opposite direction, disallowing the swimmer to flip upside down. In particular,

$$rollErr_w = dot(vert_w, horiz_p),$$

where $rollErr_w$ defines how far the world vertical unit vector ($vert_w$) is apart from the local horizontal pelvis unit vector ($horiz_p$). If these vectors are perpendicular - no torques are generated.

Interactive target placement is achieved by casting a ray from the camera screen space towards the water surface. The position at which the ray hits water surface is considered as target position ($target_p$). The positions of the pelvis ($pelvis_p$) and the head ($head_p$) in world coordinates are used to obtain the direction towards the target ($target_{dir}$) and the direction to which the character is currently traveling ($overhead_{dir}$) as follows:

$$target_{dir} = \|target_p - pelvis_p\|,$$
$$overhead_{dir} = \|head_p - pelvis_p\|.$$

Next, we convert these directions into signed angles on the water plane and find the difference between them to determine how far apart the current heading direction is from the target direction in terms of angles as follows:

$$yawErr_w = atan2(target_{dir}.x, target_{dir}.z) -$$
$$atan2(overhead_{dir}.x, overhead_{dir}.z).$$

The resulting signed error ($yawErr_w$) is then used to generate torques on the "yaw" axis of the waist joint, making the character "lean" towards the required direction. This way, the virtual swimmer controller is extended with additional feedback torques that are exerted upon his waist joint, ensuring he maintains balance and interactively tracks a target placement.

3.5 Breathing Model

3.5.1 Breathing Patterns Design

The human lung volume depends on the amount of air that it holds inside. However, breathing in/out also affects the overall torso density and volume. For our model, we simplify this circumstance and simulate only the change in overall torso density, rather than the visible volume. Normally, lungs at maximum inspiration will force the whole body to float, whereas exhaled lungs will result in sinking, making torso density a decisive parameter of whether or not the body floats in water. We use varying torso densities as an input parameter that determines the volume of a single voxel, mentioned in section 3.3.3. Consequently, modifying the torso voxel volume property will result in differently buoyant body without causing changes in the visual

appearance of the torso.

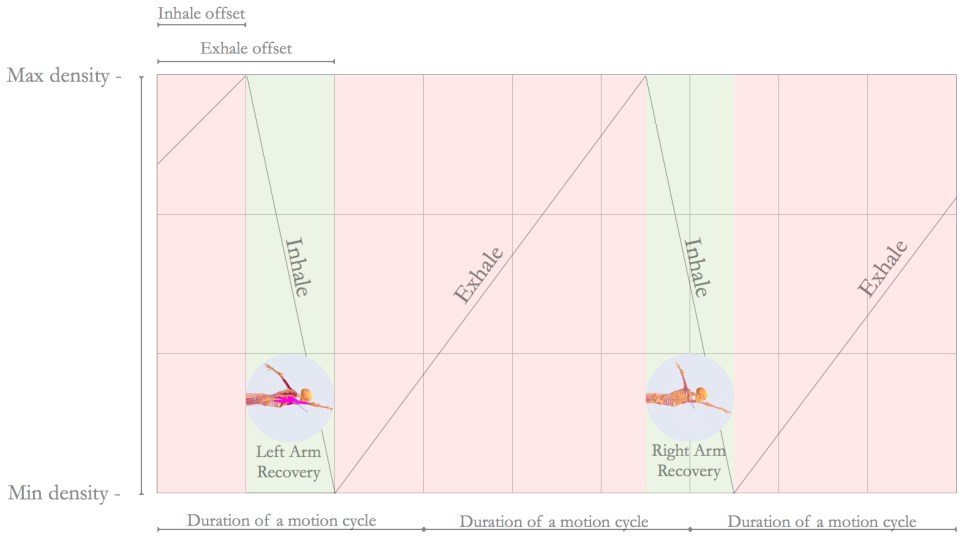

Figure 3.7: A blueprint of one full cycle of the *bilateral* front crawl breathing pattern as a piecewise linear function. Swimmer inhales only during the arm recovery states.

Respiratory strategies vary from style to style. For example, butterfly swimmers can inhale on each, every two, or only on every third stroke. Long distance front crawl swimmers usually inhale once every third stroke to each side, whereas sprinters do it every second stroke on the same side, etc. The chosen strategy can be modified during swimming based on the distance a swimmer has to overcome or adjusted on a per stroke basis, depending on fatigue. Since there are many ways to breathe and it is closely related to what part of the motion a swimmer is in, a function that models such patterns is made adjustable and customizable depending on what type of breathing is being represented. Input parameters for a respiration function are the states at which the transitions from inhaling to exhaling (or vice-versa) are made. In addition, the minimum and the maximum torso densities are taken into account to model the largest and the smallest oxygen uptakes at these transition moments. Additionally, strategies like breathing-in every third stroke to each side during front crawl stroke require three motion cycles to complete two breath-in-breath-out cycles (see figure 3.7), which means that such parameters should also be taken into consideration.

3.5.2 Piecewise Linear Function for Respiration Modeling

We model swimmer respiration patterns using a piecewise linear function, in which the horizontal axis represents time within a breath pattern, mapped to the torso density on the vertical axis. Conditions for switching between functions are specified by the two offsets, the first one being a number of states since the beginning of the motion style to the moment of first inhalation, while the second being the number of states to the moment of the first exhalation since the beginning of the motion style. Specification of the peak densities is related to elite swimmer lung volumes mentioned in section 2.2.1 of this thesis.

3.6 Summary

A real-time animation of a breathing humanoid swimmer in a physics based environment is accomplished by creating a virtual rag-doll, equipping it with kinematic constrains, attaching components for water sensing, a control scheme, a respiration function and submerging the system into a simplified fluid environment. To summarize in more concrete terms - a virtual rag-doll is made in accordance with an anthropometric data of elite swimmers. Character motions are limited to humans using kinematic constraints. Body segment colliders are subdivided into volumetric components for water sensing. A FSM contains definitions of swimming style key-poses, which are used in a PD-controller to actively actuate the swimmer. Additional torques are added on a waist joint to balance-out and assist the swimmer in following the target position. Breathing pattern is described through a customizable piecewise linear function. Finally, a large collision volume serves as an indication on when to evaluate governing fluid equations to produce water forces.

Chapter 4

Implementation and Analysis

4.1 Overview

Firstly, in section 4.2.1 we report on what kind of software and hardware was used to achieve our system. Further, in section 4.2.2 we describe the software specific setup and follow up with system architecture in section 4.2.3. Then, in section 4.3.1 we describe what are the input parameter types and how do they influence components behavior during the simulation.

4.2 System Implementation

4.2.1 Software and Hardware

The virtual swimmer shapes are drawn in 2D and later extruded to 3D using *Blender* v2.63 modeling software. The composition of objects and simulation itself are carried out using the *Unity* v4.3.3 game engine. Physics computations, required to advance the simulation forward, are executed through a built-in *PhysX* physics engine. Unity uses an implementation of the standard *Mono runtime* for scripting the behavior of components. We build an interface using Mono runtime to communicate with the game engine. Manipulation of the humanoid character and his interactions with the fluid environment are written using the *C#* programming language. All simulations were run on a 2.6 GHz Inter Core i7 processor Apple Macbook Pro with 8 GB 1600MHz DDR3 of RAM.

4.2.2 Software-specific Design

Firstly, we create a hierarchical representation of the swimmer body parts using Blender, as described in 3.2.1. After that, we import it into Unity as a parented combination of *game objects*. There, a game object is the most fundamental building block for any object existing in the scene.

By default, *a transform* component is assigned to every game object, which is responsible for positioning, orienting and scaling an object inside a 3D scene. We recommend to reset the scale of the objects back to "1" in the modeling software after creating the model, but before importing it into Unity, otherwise it will cause non-uniform scaling transformations on corresponding axes. Since we want our animation to be physics-based, we have to enable the physics engine by attaching *rigidbody* components to each of the character body segments and specify their mass property. Another component that we provide our objects with is *a collider*, which is an invisible shape around an object that is defined for the purpose of reacting to physical collisions between objects. Instead of using built-in collision handling functionality for solids, we exploit the colliders to create custom fluid behavior. To create kinematic pairs, all body segments, except for the root are equipped with *configurable joints*. A configurable joint function is two-fold. They can either restrict, or accelerate the movement/rotation of the attached game object. We only use the restriction functionality of the configurable joint and use a custom PD-controller to accelerate. Any linear motion between the two objects is constrained, whereas angular motion is specified by imposing joint limits, as described in table 3.1.

4.2.3 System Architecture

As previously mentioned, the behavior of objects in the scene is written using the *C#* programming language. An overview of our system architecture can be seen in a class diagram, depicted in figure 4.1.

In order to have a control interface with the simulation, we create a single class *"VirtualSwimming"*. There, we expose variables that are required to parametrize the whole system and to manage their changes throughout execution. Also, we retrieve the two objects of type "Rigidody" and "ConfigurableJoint" from the framework and pass them along to the *"Character"* class to setup the data related to swimmer body structure. Furthermore, we include a *"Logger"* class to record the simulation output to a file.

During the initialization, *"Character"* class is responsible for passing the parameters to their corresponding class objects. Then, during the simulation it is managing the order in which the class objects are going to be updated (e.g. we need to renew information after every physics step before computing torques and not vice-versa). Firstly, it updates the body segments with "newly arrived" information from the breathing module and the physics simulation. Then, it is used together with target pose data coming from the knowledge about swimming styles to feed the controller. Finally, *"Character"* class updates the style class object, consequently bringing the FSM up to date.

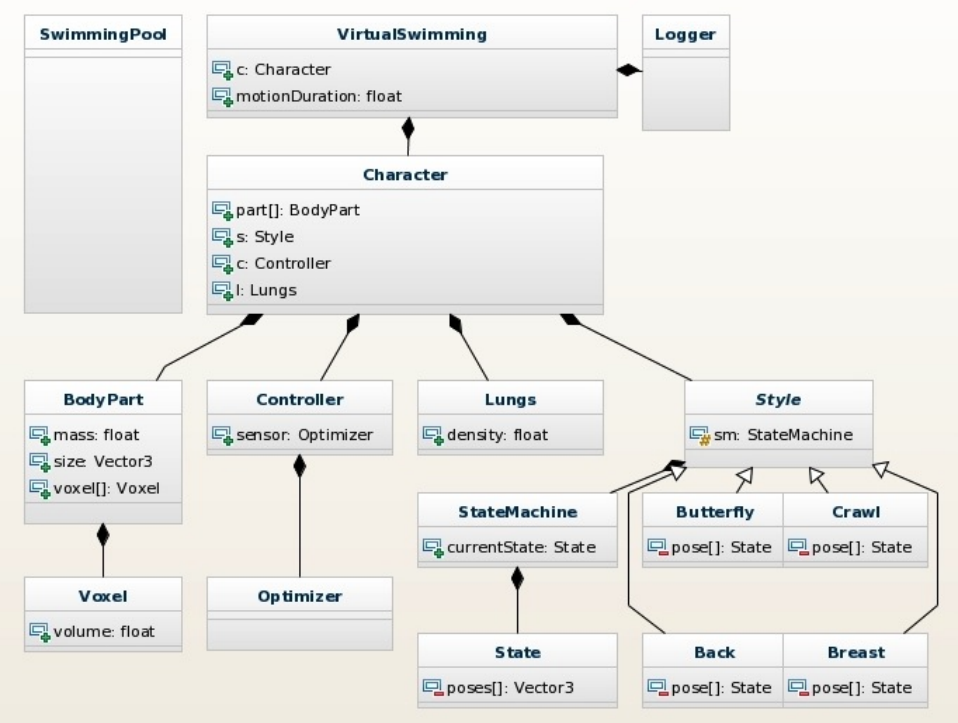

Figure 4.1: Class diagram of a framework for real-time physics-based animation of a humanoid swimmer.

The class *"BodyPart"* contains the anthropometric data, a list of *"Voxel"* class objects and determines the voxel arrangement within a single body segment. Furthermore, it updates the torso voxel volumes on every frame depending on the current density.

A *"Style"* is an abstract class, that defines data related to swimming style. In particular, any class implementing a *"Style"*, will have to initialize the state machine with all of the target pose information. Thus, a *"StateMachine"* class will sustain target poses during any moment of the simulation.

The *"Controller"* is responsible for putting together the incoming information regarding the current and the target states of the simulation. Further, it is made accountable for updating an *"Optimizer"* class object that computes additional torques. Those are required for maintaining balance and following the interactive world target placement.

4.3 System Analysis

To have a consistent way for testing parameters, we add an end condition to the execution of the simulation. For that reason, we place a spherical game object at the target position and attach a collider to it. Whenever the swimmer collides with it, we record the parameters indicating about swimmer performance, modify the ones that are being tested and restart the simulation.

4.3.1 Motivation and Goals of the Analysis

The system behavior can be parametrized in three major blocks. The first one defines the precision of the fluid simulation, the second one is related to motion specification, and the third one is associated with the breathing of the swimmer.

Fluid accuracy

The *numberOfVoxels* parameter defines the body segment subdivision resolution. More precisely, it sets how many voxels will be created on each local axis of the rigid part. For example, if we set this parameter to 4, then every rigid part will contain 64 voxels. During the development we observed that if this value is low, the swimmer carries out staggered motions and does not reach the target as fast as in the higher resolution cases. However, if we increase the resolution, motions seemed to increase in velocity. At some point the system becomes unable to render frames fast enough. Therefore, such investigation is based on the performance in terms of the speed of the swimmer and the system ability to deliver frames in real-time.

Style and motion duration

The *currentStyle* of the swimmer and the *cycleDuration* parameters are responsible for selecting the right collection of target poses for a particular swimming style and determining how long will the state machine loop last, respectively. As mentioned in 3.4.1, state machine transitioning between the target poses is based on the time parameter which does not ensure that the poses will be reached completely. If the time for the pose is not large enough, the states are transitioned rapidly, but the motion does not appear expressive. However, if the specified time is too large, the target poses become noticeable and the motion is no longer continuous. Additionally, both of the extreme cases cause our swimmer to traverse slower. Thus, the style and motion duration analysis is based on the swimmer performance in terms of velocity and the continuity of his motions. However, if the virtual

swimmer is performing at its' best - it is most likely that the motion will be continuous and should not require additional verification. For that reason, we only seek to find the *cycleDuration* parameter with which swimmer minimizes the time to reach a goal.

Breathing patterns

To create different types of breathing patterns we use two variables: *inhalesPerPattern* and *strokesPerPattern*. The first one specifies how many times should a swimmer inhale during one pattern. It also implies the number of exhales, since the piecewise linear function we have modeled is continuous (section 3.5.2). The second parameter defines how many strokes should a swimmer perform to complete one full breathing pattern. Altering these values allows us to simulate common breathing techniques, listed in table 4.1.

Table 4.1: Input parameter values representing common breathing patterns.

Style	*inhalesPerPattern*	*strokesPerPattern*	When to inhale
Butterfly	1	1	Every stroke
Butterfly	1	2	Every second stroke
Butterfly	1	3	Every third stroke
Front-crawl	1	1	Every second stroke
Front-crawl	1	2	Every fourth stroke
Front-crawl	2	3	Bilateral breathing (figure 3.7)
Backstroke	1	1	Every second stroke
Backstroke	1	2	Every third stroke

Chapter 5

Experiment

5.1 Parameter Testing and Results

In the following section we report what combinations of input parameters, described in section 4.3.1 have produced the best results in terms of computational and swimming performances. We also investigate whether the amount of air in the lungs due to different respiratory strategies has influence on the swimming speed.

numberOfVoxels

Firstly, we want to find the optimal variable for voxel resolution (i.e. the *numberOfVoxels*). To do that, we initialize *cycleDuration* and *defaultTorsoDensity* with fixed values. Also, we adjust the system, such that it would only increment the subdivisions when the swimmer reaches target position. After running series of tests, we observe differences in swimmer velocities upon different voxel subdivisions (see figure 5.1). It shows a negligible increase beyond number "8". If the swimmer velocity is no longer increasing by more than 1% due to the increased subdivision - we assume that further refinement is not necessary as it would not be optimal.

After that, we turn our attention to the last resolution value with which the system was able to render at least 30 frames per second. Figure 5.2 shows that increasing resolution above "10" voxels per axis drops the frame rate significantly, because the physics computation step consumes all the processing power. This allows us to conclude that with a fixed physics time step of 0.02(s) the optimal *numberOfVoxels* is between "8" and "10". After discovering these parameters, we can assume that in the context of our model, increasing resolution will no longer influence the speed of a swimmer significantly, nor will it render frames at real-time rates.

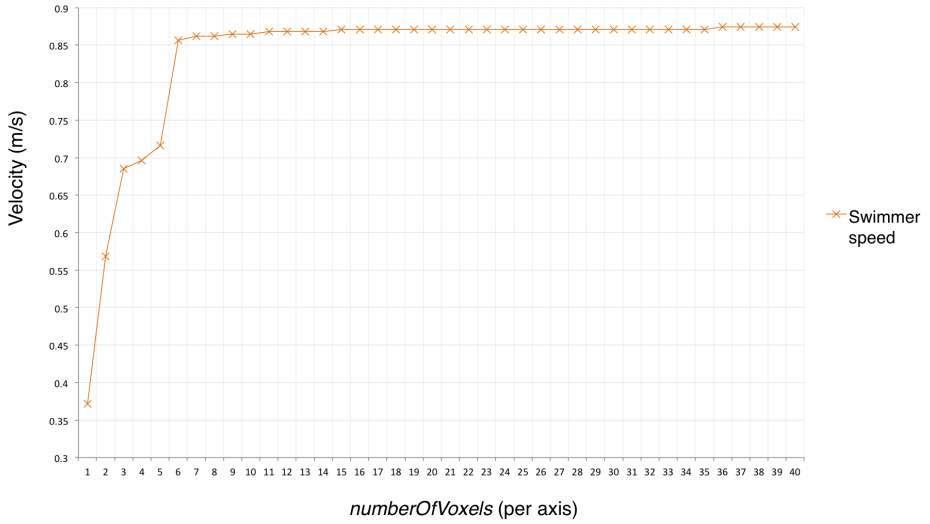

Figure 5.1: Swimmer speed does not significantly increase after "6" voxel per axis subdivision.

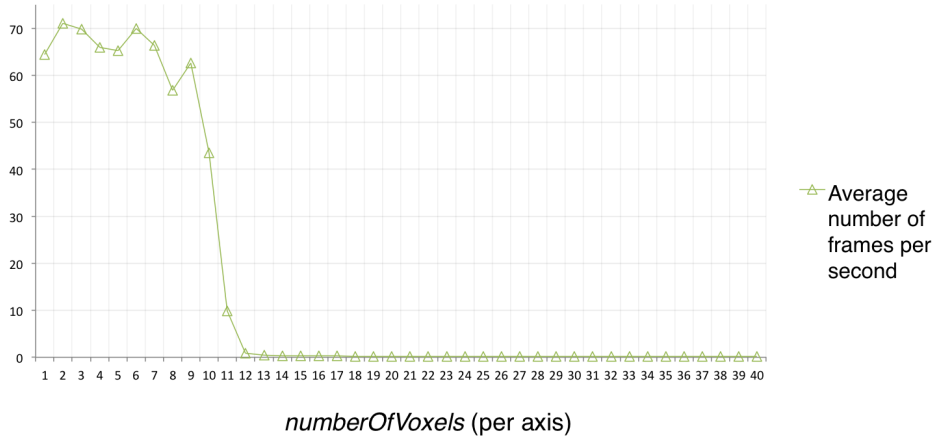

Figure 5.2: The system is able to render frames in real-time (i.e. less than 30 FPS) up until the resolution of "10" voxels per axis.

cycleDuration

The goal of a real competitive swimmer is to swim a given distance in as little time as possible. Thus, one of the aims for further examination becomes to find the best performing *cycleDuration* for every swimming style. In order to do that, we initialize the *cycleDuration* with a value that produces at least some form of locomotion, add a counter to measure how long did the simulation take and modify the simulation to increment it by 0.05

every time the swimmer reaches goal position. Such increment is precise enough to get an indication about the performance differences. In addition, we fix the subdivision resolution to "9" and test the implemented styles with two fixed distances (25 and 50 meters) away from the initial position.

Figure 5.3 shows the differences of how long did it take for the swimmer to perform a butterfly motion to reach the goal position. We notice that the simulation takes the least amount of time when the input parameter is between 1.45 and 1.55. Values to the left of these produce the aforementioned motions, when the swimmer is rapidly traversing butterfly poses, but does not generate enough propulsion due to "shortcuts" it must take to satisfy the time constraint. Values to the right create prolonged, slower strokes that do generate propulsion, but do not match the best ones due to lower limb velocities.

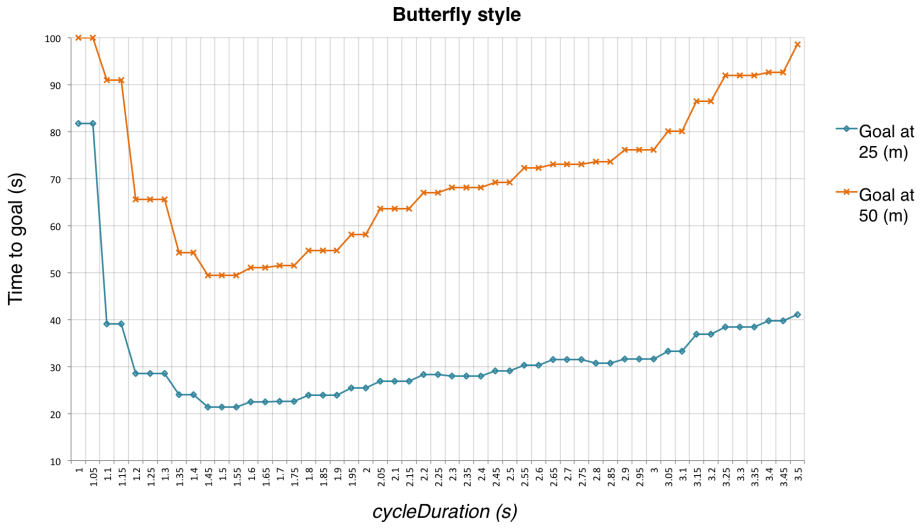

Figure 5.3: Swimmer performance in butterfly style is best, when the *cycleDuration* parameter is set between *1.45* and *1.55*.

Figures 5.4 and 5.5 display the same scenario for front-crawl and backstroke swimming styles, respectively. Even though the values are slightly different, we can still notice a few input parameters next to each other that produce the fastest performances. However, one difference for these styles is that we could not observe as many increased speed performances due to increasing *cycleDuration* as in the butterfly case (i.e. to the left of optimal value). Possibly this is due to the balancing heuristics that we apply on the waist joint. Usually, front-crawl and backstroke swimmers allow body roll rotations and maintain the direction of swim with the help of their hands.

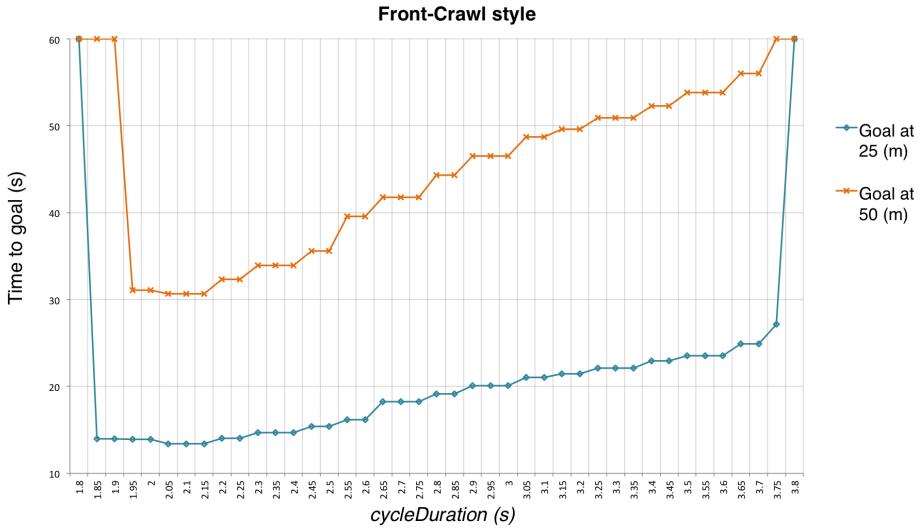

Figure 5.4: Swimmer performance in front-crawl style is best, when the *cycleDuration* parameter is set between *2.05* and *2.15*.

Our balancing heuristics try to keep the torso parallel to the water surface, which might cause him to flip over and prevent from reaching the target in cases when arm motion is too rapid. (i.e. with overly small *cycleDuration* values). Some stepped and erratic differences between the consecutive parameters are due to the number of strokes it took to reach a goal. For example, it may take two strokes less to reach a target with a single increment in *cycleDuration*. Increasing the analyzed parameter further makes the target poses visible until these discontinuities cause the swimmer to flip over again. One possible solution to both of these issues could be to introduce adoptive controller gains that depend on the motion input parameters. For example, the smaller the *cycleDuration*, the larger the controller output torques. This would make the motions more expressive with low time values, whereas high ones would cause less disruptive motions. Also, it would establish control over the swimming speed. In our case it is only possible through the direct manipulation of controller gains, which is not always straightforward.

Consequently, we find out that in the context of our model, there is a single *cycleDuration* input parameter for each swimming style that defines the best performing motion in terms of the time it takes to reach the goal.

Effects of different breathing patterns

To compare the effects of different respiration techniques on swimming performance we combine the data discovered so far with the torso variation

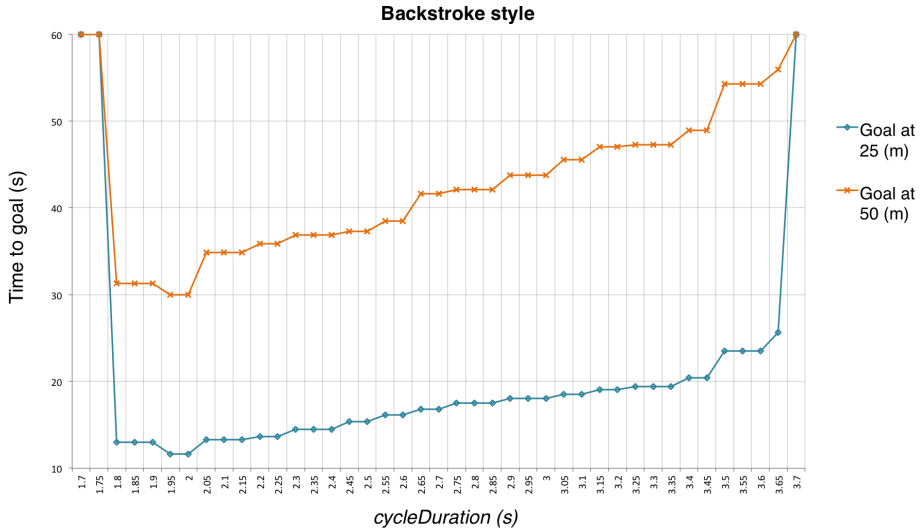

Figure 5.5: Swimmer performance in backstroke style is best, when the *cycleDuration* parameter is set between *1.95* and *2*.

functionality described in 3.5. For every implemented swimming style we make use of both optimal resolution and the best performing motion cycle duration parameter. Tests of fixed torso densities are also included to support the results. In addition, we increase the swimming distance to 100 meters to exaggerate possible effects. Figure 5.6 illustrates the differences of swimmer performances due to breathing patterns.

In their analysis, [PK06] notice a significant decrease of crawl-stroke velocity when breathing on every stroke compared to no breath. However, our data suggest hardly noticeable dissimilarities between these conditions. It may be due to the fact that real swimmers have to coordinate their strokes and maintain a streamlined body while breathing, whereas our simulation solely controls the buoyant properties of the torso. Therefore, breathing in on every stroke in their case could have created additional drag, which may have caused the significant decrease in velocity. On the other hand, our simulation does show that only minor performance differences would be applicable in case when body is consistently streamlined and not influenced by the stroke imperfections. Our data might also suggest that effects of breathing patterns have more influence on one style than the others. In the butterfly case, we do not observe a large change in time it took to reach the goal. In contrast, front-crawl performance was improved when breathing on every stroke. This may be due to the fact that on average swimmer torso remained buoyant more time than using other breathing techniques. On the contrary, this same breathing method has decreased the backstroke style

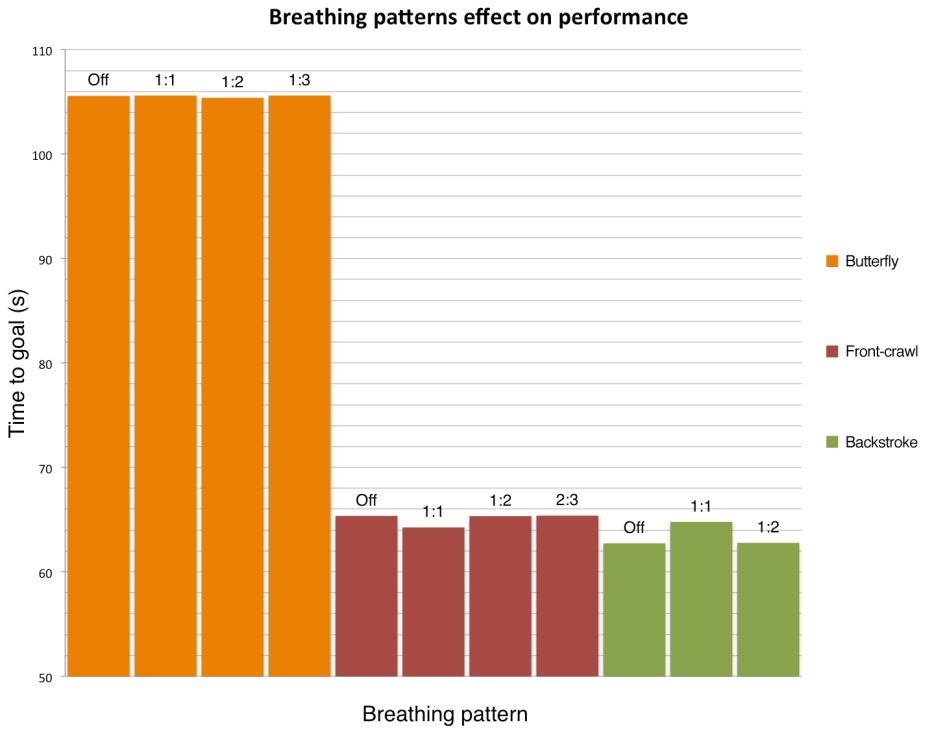

Figure 5.6: Swimmer performances using no or different breathing patterns (i.e. either "Off" or different "*inhalesPerPattern*:*strokesPerPattern*" ratio). Bar labels indicate which strategy was tested.

performance, which could possibly have been caused by an extra stroke that was needed to reach the target position.

For future reference, it would be interesting to incorporate an automatic optimization procedure into our controller. Such strategy would attempt to find best parameters related to breathing and motion timing; for example, removing restrictions of inhalation timing and trying to determine a breathing pattern that actual swimmers do not necessarily use, but that would produce better performances.

Chapter 6

Conclusions, Limitations, and Future Work

6.1 Conclusions

In this work, we have presented a real-time physics-based system for animating humanoid swimming. It includes a simplified fluid environment, and a controlled virtual character traversing it in three different swimming styles.

Character model.

Even though our virtual character model is only an approximation of how real swimmers look, we believe that with the current state of real-time physics-based animation such characters are quite promising and precise enough to allow at least some focus on other aspects. In other words, customizing a control scheme for a physics-based character is still a challenging problem, but the available tools and similar approximations allow us to shift focus on other indispensable physical features.

Breathing importance.

Our results have showed that simulated torso density change patterns do not affect the virtual character performance largely, but some common strategies have slightly different effects on the efficiency of the strokes. Besides, using our system to animate breathing patterns by drawing indicative visual cues of when to inhale and exhale is straightforward and might be helpful when teaching novice swimmers how to synchronize respiration with motions.

Time-based state transitions.

Results of the virtual swimmer performance using time-based transitioning method show an existence of a motion cycle duration input parameter that produces the best performance with a particular controller gain configuration. The main advantage of this method is that there is no need to worry about whether or not the target pose was achieved as long as it gives the required guidance for the upcoming poses and satisfies the time constraint. However, the disadvantage is that there is no control of the overall speed of the swimmer. This brings us to conclude that, if such control is required, either some form of adoptive controller gain technique or a shift to another method based on a pose similarity metric is necessary.

6.2 Limitations

Drag force.

An implemented version of voxel-based drag force does not take into account the angle of attack at which the body segment is traveling through water. In other words, there is no distinction whether the palm is moving through water vertically or horizontally - generated drag force will be the same.

Missing competitive style.

Breaststroke style was not incorporated due to the specifics of that motion and the above-mentioned drag force limitation. This particular style has a recovery motion that happens underwater while minimizing drag force. Since we do not simulate the influence of rigid segment angle of attack on water, we could not produce locomotion in breaststroke style. A more precise reference-area based drag force computation would have allowed to incorporate breaststroke style to the simulation.

Excessive pelvis rotations.

Balancing feedback loop torques produce some unwanted pelvis rotations on both pitch and roll axes, most likely because the upper body mass is greater than that of the lower. A possible solution would be not to model at the character with a single rigid component for torso. If we were to introduce several additional hinge joints in the backbone region and distribute the feedback torques over it, we could achieve more flexible and streamlined motions.

Non-realistic body parts.

Human beings do not look like our swimmer character. The geometry was chosen to be such, so that the simulation could run in real-time as it was not known if it would perform at these rates. However, after running at interactive rates, we found that this simplified fluid approach allows us to have more complex geometry.

6.3 Future Work

A commitment to this project has inspired to envisage some possible extensions and improvements of this work. In the following we present several ideas that could likely make the output of the system more credible and compelling.

Incorporating a skinned / textured 3D model for swimmer body representation would certainly improve the appearance of the animation. Also, reducing the controller input parameter space by exploiting symmetries that exist in swimming styles. For instance, to describe the butterfly or breaststroke styles it could be enough to specify one side of the swimmer pose. In addition, extending the system by including a simplistic interface to modify the swimming poses by "dragging-and-dropping" would allow swimming coaches to test out the system on a handheld device in the swimming pool.

Moreover, extending the breathing model by animating the fact that character has to turn his head in order to inhale and then comparing swimming performances could be an interesting work. It could also confirm our result that buoyancy variation patterns do not affect performance as significantly as the drag force that one creates during breathing. Another way to improve the accuracy of the breathing model would be to label voxels that occupy the volume of the lungs differently. Lungs are only a part of the torso, thus their corresponding voxel densities would be different. Also, it would be possible to animate the changes of the trunk size, when breathing. Extending our breathing model with an automatic optimization technique could find a breathing pattern that would actually improve swimming performance.

Furthermore, incorporating hydrodynamic lift force could extend our voxel-based simplified fluid approach by improving swimming efficiency. Another way would be implementing a Navier-Stokes equation solver that would allow the animation of water splashes or waves. Besides including automatic optimization for the breathing model, it would also be useful when finding more suitable controller gains or any other control parameter values. Next, using a genetic algorithm with a fitness function could spawn

an improved set of motions. It would be interesting to see if the generated motions, satisfying the fitness function still have the visual appearance of those based on biomechanical research. Another future work could be incorporating a different control method, such as the muscle actuation or state transitioning based on a pose distance to be able to control the speed of the motions (i.e. stroke rate).

Lastly, performing a qualitative research could be beneficial to understand and evaluate the naturalness of the produced swimming motions. Displaying animations to real swimmers and asking to answer Likert-scale type questions could provide some indication if the motions appear natural and what animation aspects could be improved. It could also reveal how much motion information do joint-space actuation approaches not include/subtract from original motion.

Appendix A

Resulting animations

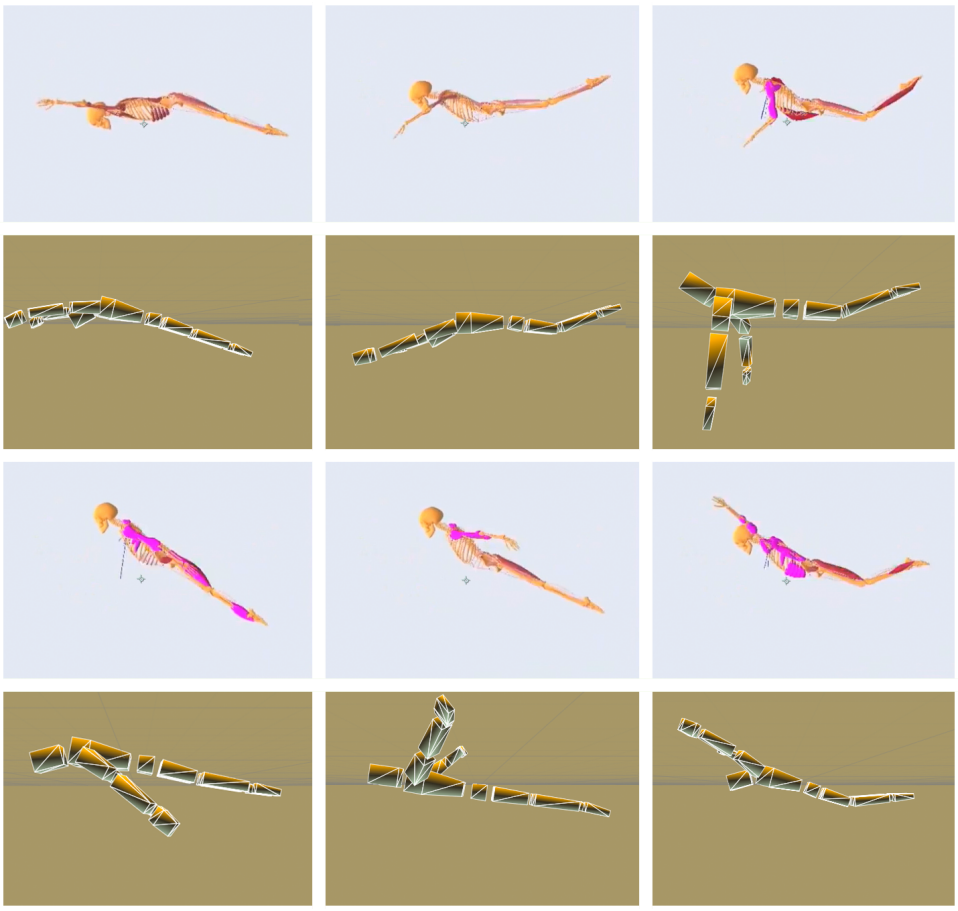

Figure A.1: Reference motion states compared to stills of our character performing the butterfly style.

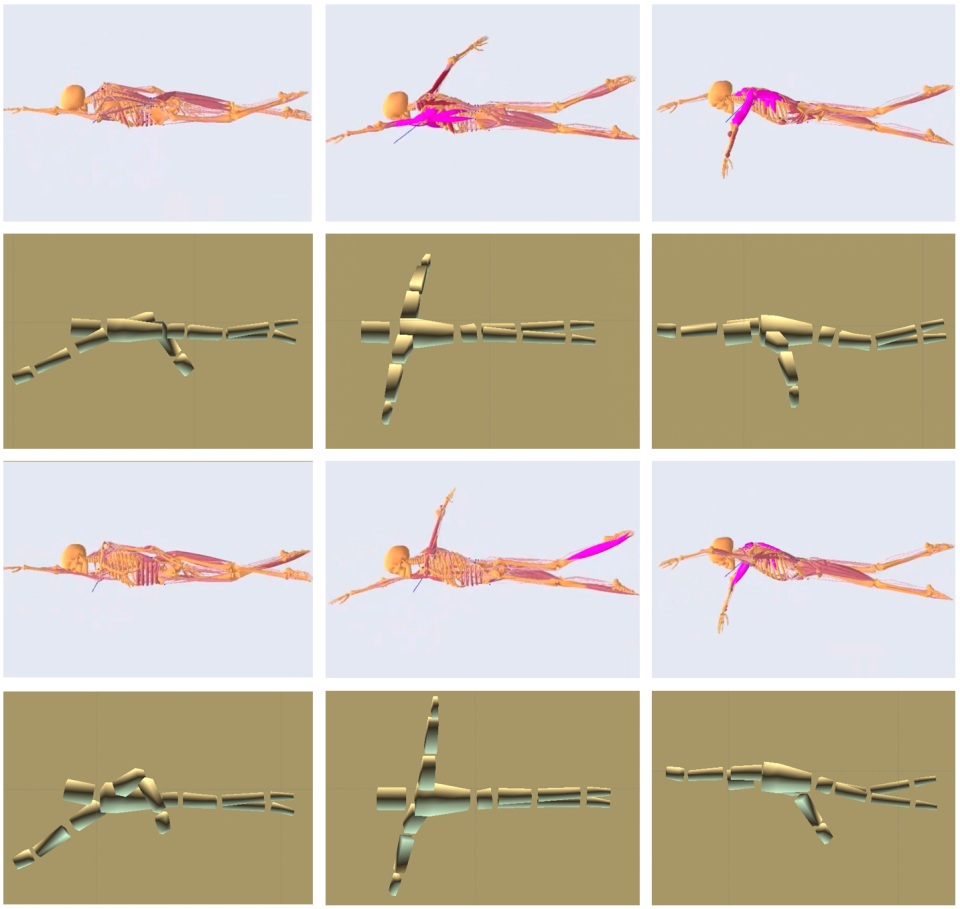

Figure A.2: Reference motion states compared to stills of our character performing the front-crawl style.

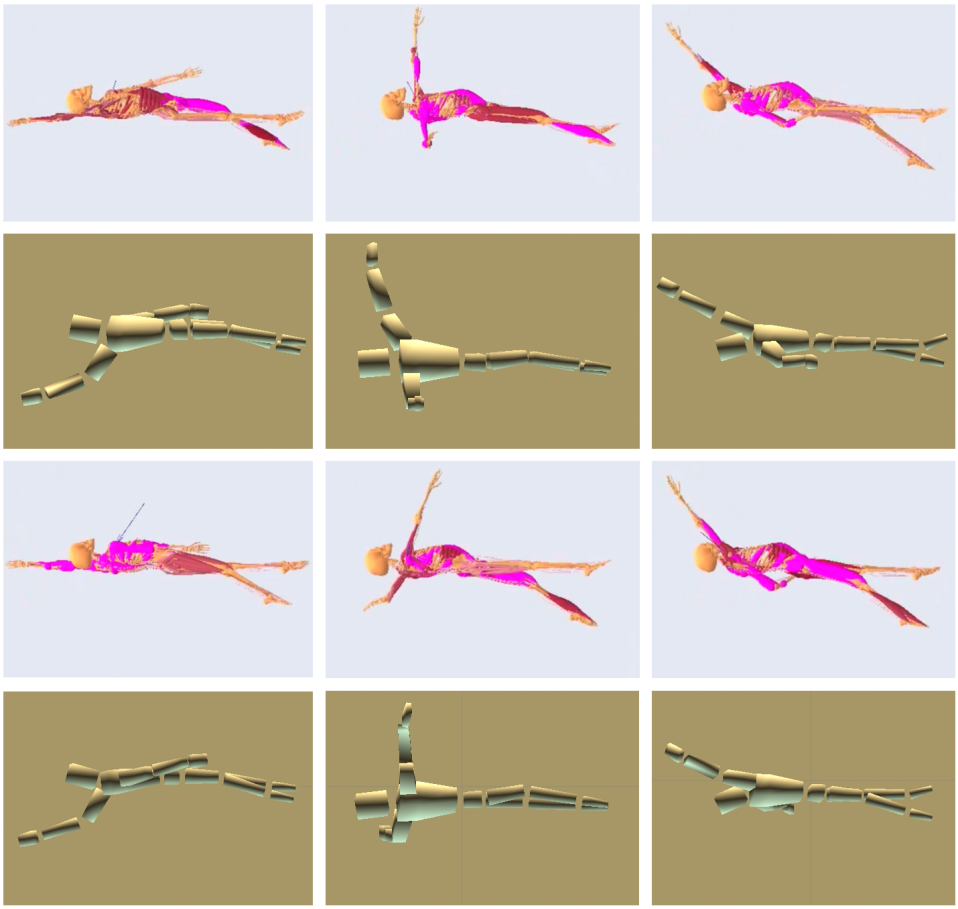

Figure A.3: Reference motion states compared to stills of our character performing the backstroke style.

Bibliography

[AIA+12] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph.*, 31(4):62:1–62:8, July 2012. 9

[AR12] Marc Azéma and Florent Rivère. Animation in palaeolithic art: a pre-echo of cinema. *Antiquity*, 86(332):316–324, June 2012. 1

[BGPB97] Bonnie G. Berger, J. Robert Grove, Harry Prapavessis, and Brian D. Butki. Relationship of swimming distance, eexpectancy, and performance to mood states of competitive athletes. *Perceptual and Motor Skills*, 84(3c):1199–1210, 2014/05/08 1997. 1

[BMCS11] Tiago M. Barbosa, Daniel A. Marinho, Mário J. Costa, and António J. Silva. Biomechanics of competitive swimming strokes, 09 2011. 6, 12, 20

[BO83] Bonnie G. Berger and David R. Owen. Mood alteration with swimming-swimmers really do "feel better". *Psychosomatic Medicine*, 45(5), 1983. 1

[BPC+09] T.M. Barbosa, E. Pinto, A.M. Cruz, D.A. Marinho, A.J. Silva, V.M. Reis, and T.M. Queirós. Evolution in swimming "science" research. *Motricidade 5(3)*, 2009. 5

[BSS+13] Ozgur Bostanci, Bilsen Sirmen, Bunyamin Sahin, Menderes Kabadayi, and Onder Daglioglu. Comparison of respiratory parameters with lung volume fractions determined by stereological methods in elite swimmers. *Australian Journal of Basic and Applied Sciences*, pages 201 – 208, 2013. 6

[CBvdP10] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Trans. Graph.*, 29(4):130:1–130:9, July 2010. 8

[CMT04] Mark Carlson, Peter J. Mucha, and Greg Turk. Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23(3):377–384, August 2004. 10

[CMY69] C.E. Clauser, J.T. McConville, and J.W. Young. *Weight, volume, and center of mass of segments of the human body.* Number v. 32, no. 3 in AMRL-TR. Aerospace Medical Research Laboratory, Aerospace Medical Division, Air Force Systems Command, 1969. 6, 14

[EMF02] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, July 2002. 10

[FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 23–30, New York, NY, USA, 2001. ACM. 9

[FM96] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical models and image processing*, 58(5):471–483, 1996. 10

[FvdPT01] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 251–260, New York, NY, USA, 2001. ACM. 16

[GM77] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389, November 1977. 9

[GP12] T. Geijtenbeek and N. Pronost. Interactive character animation using simulated physics: A state-of-the-art review. *Computer Graphics Forum*, 31(8):2492–2515, 2012. 8, 22

[GvdPvdS13] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Trans. Graph.*, 32(6):206:1–206:11, November 2013. 8

[HW65] F. H. Harlow and J. E. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids*, 8:2182–2189, December 1965. 9

[HZ11] Aaron Hertzmann and Victor Zordan. Physics-based characters. *IEEE Computer Graphics and Applications*, 31(4):20–21, 2011. 8

[KWC+10] Nipun Kwatra, Chris Wojtan, Mark Carlson, Irfan Essa, Peter J. Mucha, and Greg Turk. Fluid simulation with articulated bodies. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):70–80, 2010. 11, 12

[LGS+11] Michael Lentine, Jon Tomas Gretarsson, Craig Schroeder, Avi Robinson-Mosher, and Ronald Fedkiw. Creature control in a fluid environment. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):682–693, May 2011. 11

[Mag03] E.W. Maglischo. *Swimming Fastest*. Human Kinetics, 2003. 5, 10

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. 9

[MKK10] W.D. McArdle, F.I. Katch, and V.L. Katch. *Exercise Physiology: Nutrition, Energy, and Human Performance*, pages 761–763. Point (Lippincott Williams & Wilkins). Lippincott Williams & Wilkins, 2010. 14, 15

[MM13] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4):104:1–104:12, July 2013. 9

[NM07] M. Nakashima and Y. Motegi. Development of a full-body musculoskeletal simulator for swimming. In *International Symposium on Computer Simulation in Biomechanics (ISCSB2007)*, pages 59–60, 2007. 20

[PK06] Tommy Pedersen and Per-Ludvik Kjendlie. The effect of the breathing action on velocity in front crawl sprinting. *Portuguese Journal of Sport Science*, 6(supl 2):75–77, 2006. 6, 35

[PP10] T. Pejsa and I.S. Pandzic. State of the art in example-based motion synthesis for virtual characters in interactive applications. *Computer Graphics Forum*, 29(1):202–226, 2010. 7

[SF95] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 129–136, New York, NY, USA, 1995. ACM. 9

[Sta03] Jos Stam. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*, volume 18, page 25, 2003. 9

[TGTL11] Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. Articulated swimming creatures. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 58:1–58:12, New York, NY, USA, 2011. ACM. 11, 12

[TLT11] Jie Tan, Karen Liu, and Greg Turk. Stable proportional-derivative controllers. *IEEE Comput. Graph. Appl.*, 31(4):34–44, July 2011. 8

[Tou02] Huub M Toussaint. Biomechanics of propulsion and drag in front crawl swimming. In *International Symposium on Biomechanics in Sports*, 2002. 12, 20

[Tro99] John P. Troup. The physiology and biomechanics of competitive swimming. *Clinics in Sports Medicine*, 18(2):267 – 285, 1999. 2

[vR90] L.C. van Rijn. *Principles of fluid flow and surface waves in rivers, estuaries, seas and oceans.* Aqua Publications, 1990. 12

[VWVBE+10] H. Van Welbergen, B. J. H. Van Basten, A. Egges, Zs. M. Ruttkay, and M. H. Overmars. Real time animation of virtual humans: A trade-off between naturalness and control. *Computer Graphics Forum*, 29(8):2530—2554, 2010. 7

[WH96] Wayne L Wooten and Jessica K Hodgins. Animation of human diving. In *Computer Graphics Forum*, volume 15, pages 3–13. Wiley Online Library, 1996. 6

[WJM06] Pawel Wrotek, Odest Chadwicke Jenkins, and Morgan McGuire. Dynamo: Dynamic, data-driven character control with adjustable balance. In *Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames*, Sandbox '06, pages 61–70, New York, NY, USA, 2006. ACM. 8

[YLS04] Po-Feng Yang, Joe Laszlo, and Karan Singh. Layered dynamic control for interactive character swimming. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '04, pages 39–47, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. 10, 13

[YLvdP07] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3), July 2007. 8