2
Related Work

Our research spans two very different fields of computer science — *real-time physics simulation* and *artificial intelligence* — and then attempts to bring them together for a completely different purpose — *virtual character animation*. We will first look at each of these fields separately, and then present works on motion synthesis for virtual characters, which is the core subject of the present work.

2.1
Real-time Physics Simulation

In the context of computer science, *physics simulation* (also known as *dynamical simulation*) is the simulation of systems of objects that are free to move and rotate in two or three dimensions according to Newton's laws of dynamics. For a complex system, accurately determining the position and orientation of its objects involves a huge quantity of calculations, which usually prevents the employment of physics simulation in real-time applications[1]. Accuracy can be sacrificed, however, especially when it is not necessary for the position and orientation of the objects to be known with exact precision. *Real-time physics simulation* is possible when the problem is combined with *time integration methods*, allowing the calculation of *numerical solutions* to the problem. These calculations take place in small, discrete time steps, with small inaccuracies that are imperceptible to the user.

Figure 2.1 illustrates this concept. The ball is initially static and under the effect of gravity. After each step, the ball's vertical velocity increases and its position is updated accordingly. Between steps #3 and #4, the user interacts with the simulation, applying an oblique impulse to the ball. The ball's velocity is modified, but its effects will only become visible when the next step of the simulation is computed. If the time between steps is sufficiently small,

[1]The term *real-time application* or *interactive application* is used throughout this document to describe computer programs that give their users the illusion of immediate response to their actions. To achieve this illusion, the program must never spend more than a fraction of a second in its computations, or the user will notice that the program has "stalled" and the illusion of interactivity will be broken.

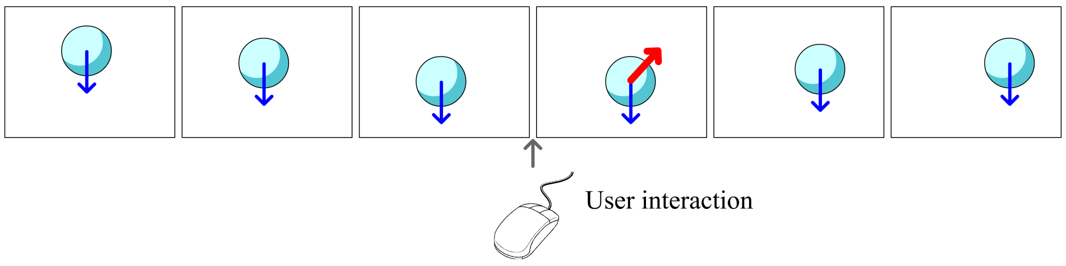

Figure 2.1: Example of real-time physics simulation.

however, the user will not realize this and will believe that his interaction had an immediate effect on the simulated object.

Physics simulation is a hot topic of research in the electronic entertainment industry. Ten years ago, it was enough for a video game to have great graphics, even if its environments were not fully interactive. The turn of the century, however, saw the release of two highly influential titles that raised the bar in terms of interactivity and immersion.

2.2(a): The player picks up and throws a barrel in *Half-Life 2*.

2.2(b): The main character in *Hitman: Codename 47* can grab unconscious people and drag them along the floor.

Figure 2.2: Examples of real-time interactive physics. It is difficult to convey the life-like motion of the simulated objects with static screenshots such as those above, but players and developers took notice.

Released in 2000 and developed by IO Interactive, *Hitman: Codename 47* was programmed with an innovative physics simulation system that made dead characters collapse on the floor in a realistic manner. The algorithm is deceptively simple and hardly innovative, as it had been developed many years prior to predict the motion of molecules, but Thomas Jakobsen was the first to realize it could be used in the context of real-time interactive physics. His paper [Jakobsen 2001] explains how to implement simple physics simulation for objects where accuracy is not a major concern, and we chose to use this technique in our research into 2D virtual characters.

Half-Life 2, produced by Valve Corporation and released in 2004, featured a weapon called “the gravity gun” that allowed players to pick up and throw loose objects in the virtual world. These objects would collide with the walls, floors, and each other, causing them to tumble and roll realistically. The technology behind the game was the Havok physics engine, which had been in development since the year 2000 and had been fine-tuned to perform complex mathematical computations without placing an excessive burden on the CPU.

The appearance of companies that specialize in physics simulation software (and, sometimes, physics simulation hardware) highlights its importance in modern electronic entertainment. When this project began, the PhysX SDK[2] was freely available from Nvidia's website [Nvidia 2008], and for that reason we chose to use it in our research into 3D virtual characters. (Unfortunately, as of this writing, Nvidia has restricted access to the SDK and only licensed developers can download it.)

2.2 Genetic Algorithms

A genetic algorithm is a search technique used to find optimal (or approximate) solutions to a problem based on a heuristic measure of the solution's quality. This technique was devised by observing biological processes — genetics, reproduction, and natural selection — and adapting them to work in a computational context. Genetic algorithms can be thought of as a *simulated evolution* of virtual species, in which each *individual* of the species is a possible solution to the problem [Davis and Mitchell 1991].

The core of a genetic algorithm is the genetic representation of its individuals, which consists of two parts:

- The **genotype** (or **chromosome**) is an abstract representation of the solution. Genotypes are typically simple data structures, such as strings or trees, that are not directly related to the solution being sought.
- The **phenotype** is a concrete representation of the solution, and is created by interpreting the genotype. Phenotypes can be thought of as the creatures whose characteristics are reflections of their genetic material.

The phenotype is evaluated by an *objective function*, which attributes a score to the individual. This score reflects the quality of the solution (as represented by the individual's phenotype), such that better solutions have a higher score (unless the optimization problem is attempting to minimize some value, in which case lower scores are better). Thinking of genetic algorithms as

[2]SDK: Software Development Kit

the simulated evolution of a species, the individual's score is called its *fitness*, and stands for its ability to survive in a harsh environment (see Figure 2.3).

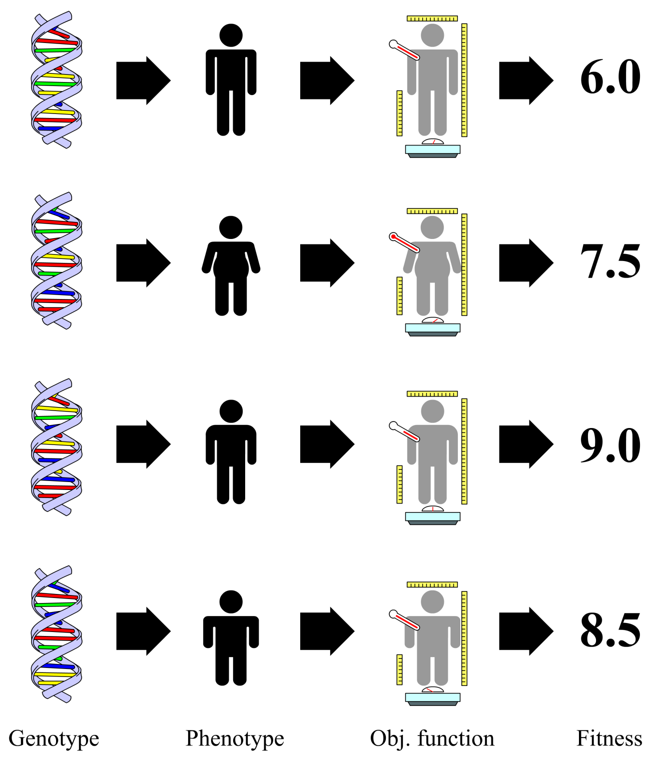

Figure 2.3: The phenotype is constructed from decoding the genotype. The objective function measures the desired characteristics of the phenotype, and calculates a fitness score for the individual.

The algorithm proceeds to apply the principle of *natural selection* to determine which individuals will be allowed to breed and pass their genetic material to the next generation. Individuals with a high fitness score are more likely to be selected, although this is a stochastic[3] process and even unfit individuals have a small chance of breeding. This ensures a certain degree of genetic diversity in the population and prevents the algorithm's early convergence to suboptimal solutions.

Breeding is carried out by the algorithm's *crossover operator*, where the chromosomes of two individuals (parents) are combined. This results in two new individuals (children) with genetic material that is related, but not identical, to the originals'. The children are then inserted into a new population of individuals, the "next generation" of solutions. The genetic algorithm then repeats the cycle — evaluation of chromosomes, selection of fit individuals, crossover of their genetic material, population of new generation — several times until a stopping condition is met. Typically, this is either a pre-determined fitness score or a pre-determined number of generations.

[3]We say that a process is *stochastic* if it is highly influenced by randomness.

Lastly, no discussion about genetic algorithms would be complete without mentioning the *mutation operator*. This operator works on a single individual, making random modifications to its genotype, and prevents the population from becoming excessively homogeneous. It is often the case that, after several generations, all individuals will have genotypes that are nearly equal to each other; mutation becomes necessary as a means of maintaining genetic diversity and exploring different areas of the fitness landscape.

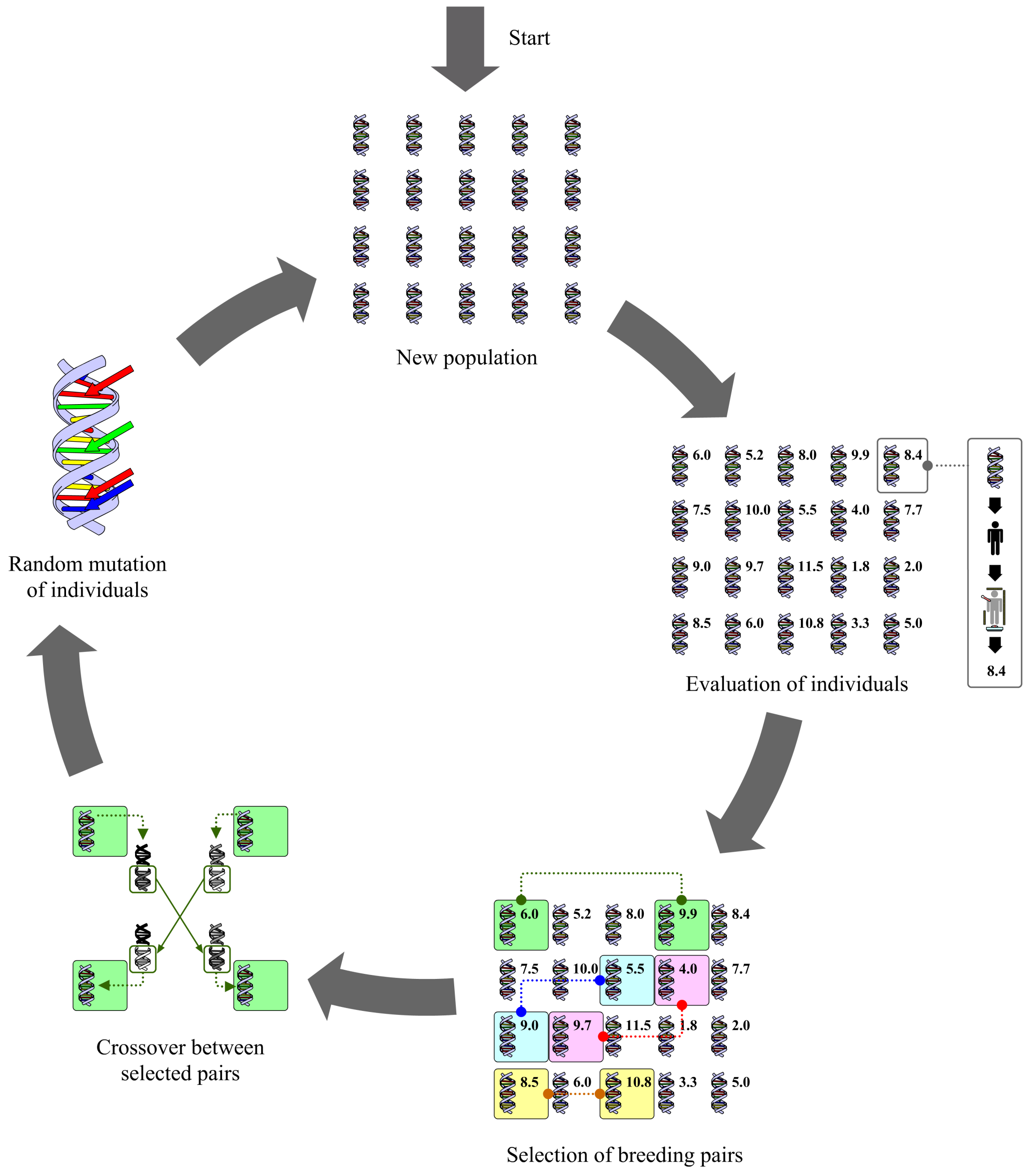
Start
New population
Evaluation of individuals
Selection of breeding pairs
Crossover between selected pairs
Random mutation of individuals

Figure 2.4: The cycle of a genetic algorithm.

Genetic algorithms have been used by [Machado and Cardoso 2002] and [Cope 2005] in the field of artificial creativity with remarkable success. While their work is not directly relevant to ours, they have shown that the pseudo-random nature of G.A.s can create aesthetically-pleasing works in a manner that traditional algorithms cannot, with surprising results that often startle the developers themselves.

2.3 Motion Synthesis for Virtual Characters

From the viewpoint of the optimal control theory [Kirk 2004], the problem tackled by this MSc dissertation is known as "synthesis of controllers", which applies to many areas, from robot locomotion [Tedrake 2004] to human movement synthesis problems [Pandy et al. 1992]. More specifically, in the present work, we have a controller synthesis problem for articulated figure movements. The control program proposed by this dissertation is a kind of motion controller that makes articulated figures move in a way that satisfies the animation goal, which is defined as an objective function of a genetic algorithm (Figure 1.2).

Most of the work on motion synthesis for articulated figures is concentrated in the 90's. We think that this concentration was motivated by the first constrained optimization technique for character animation (named "spacetime constraints") presented by [Witkin and Klass 1988] at Schlumberger Palo Alto Research in the late 80's. From the viewpoint of expanding the spacetime constraints paradigm or reacting against it, we can identify some lines of research on articulated figure motion. In the rest of this section, we present the research lines that are more aligned with the objectives of this dissertation.

The spacetime constraint formulation leads to a non-linear constrained variational problem that requires the reduction of the space of possible trajectories to those that can be represented by a linear combination of basis functions, such as cubic B-splines. [Cohen 1992] proposes an interactive method to use spacetime constraints through a spacetime window over cubic B-spline curves that represent the figure's DOF (degrees of freedom) functions. [Liu et al. 1994] extend this idea to hierarchical basis functions using a wavelet construction. These spacetime constraint approaches reflect the fact that full articulated figures (typically with 60 DOF) lead to a non-linear problem with no close form solution. In this case, only approximate methods that reduce complexity can succeed in practice. Currently, two major spacetime constraints research lines represent the forefront works on realistic articulated figures animation, both of which consider such approximate methods.

The first line uses dimensionality reduction techniques. For example, [Safonova et al. 2004] solve the optimization problem in a low-dimensional space by representing each frame of the desired motion as a linear combination of a few basis vector.

The other line proposes interactive techniques that explore how far we can go using only optimizations that can be computed rapidly enough to be used in place of traditional splines. This line of research is embodied by a new real-time technique called *Linear Spacetime Constraint Splines* or simply *wiggly splines* because of their predilection for oscillation. Wiggly splines are based on signal processing theory and have a number of attractive features: They can have unconditional stability, incorporate damping, and be computed in constant time per frame. This second line became a patent [Kass and Anderson 2009] assigned to Pixar, one of the most important animation studios nowadays.

Another problem with the spacetime constraints technique is that it uses local optimization to refine initial figure trajectories, which can lead to a local minimum that is undesirable and/or not reusable. If we are looking for a more *automated* controller synthesis (in contrast with the above-mentioned *interactive* spacetime constraints techniques), we should avoid techniques that use some kind of perturbational analysis to refine an initial trajectory. In other words, we should stay away from techniques that are local in nature. This approach gives rise to a line of research based on global search and optimization algorithms. The following works are representatives of this line of research: [Gritz and Hahn 1997], [van de Panne and Fiume 1993], [Ngo and Marks 1993], [Auslander et al. 1995], [Fukunaga et al. 1994], and [Sims 1994]. Some of these works have their roots in the spacetime constraints paradigm [Ngo and Marks 1993] [Auslander et al. 1995] and, consequently, are criticized because the existing literature has not made clear that energy is the best criterion to optimize.

Amongst the global search techniques, [Gritz and Hahn 1997] are the first authors to propose a general Genetic Programming (GP) evolution of controller synthesis. [Sims 1994] also describes the use of genetic programming to design articulated figures, but his work is not targeted towards the animation of predefined figures. Instead, his work deals with creatures whose topology evolves in an arbitrary way — which is great for artificial life games, but not for animation control. Some of the above-mentioned global techniques also use GP, but this is done for optimization tasks of fixed complexity. For example, [Ngo and Marks 1993] propose a search module that uses a genetic algorithm to choose values for the stimulus-response parameters defined by a table. Furthermore, these two authors work with walking gaits only and instructions of low specificity (like “walk forward”, instead of “walk to position X”).

The ideas underlying the present dissertation are drawn from the work by [Gritz and Hahn 1997]. We think that their work is more appropriate for automatic animation than the ones based on spacetime constraints or those based on global search of fixed complexity. However, some differences are worth to be noticed in relation to that work:

– We have games in mind and not animation for film and 3D cartoons. In this aspect, we propose a system in which the game character automatically performs an animation sequence after an event occurs. For example, the character falls after being shot and then stands up in a realistic way until a specific pose is reached. The learning process can take a long time but the repetition of the action should be in real time.

– We would like to develop a system that allows a game designer to design the creature (topology, geometry, masses, and springs) in a quick and easy way, and define general tasks (such as “move as fast as you can” or “stand up on your own”). In contrast, [Gritz and Hahn 1997] offers no facility to model arbitrary creatures and, in fact, only present results for a “jumping lamp” character.

– We would like to have a more flexible way of producing the control program. Moreover we propose more general operators for the expression trees. For instance, the expression trees proposed by [Gritz and Hahn 1997] only have “if less than zero then” conditionals.

We believe that the technique by [Gritz and Hahn 1997] has the same dependency drawback of the method proposed in this dissertation, that is: The dependency on the environment where the creature has learned how to move. However, they have not reported this particular issue.

Unfortunately, this dissertation work has not succeeded in evaluating all the proposed features. However, the results obtained so far indicate that we developed an efficient system and have started a promising research agenda.