Supplementing Frequency Domain Interpolation Methods for Character Animation

Michael Robert Leopold Molnos
School of Computing Sciences
University of East Anglia

A thesis submitted for the degree of
Doctor of Philosophy

October 2012

Supplementing Frequency Domain Interpolation Methods for Character Animation

Michael Robert Leopold Molnos

Abstract

The animation of human characters entails difficulties exceeding those met simulating objects, machines or plants. A person's gait is a product of nature affected by mood and physical condition. Small deviations from natural movement are perceived with ease by an unforgiving audience.

Motion capture technology is frequently employed to record human movement. Subsequent playback on a skeleton underlying the character being animated conveys many of the subtleties of the original motion. Played-back recordings are of limited value, however, when integration in a virtual environment requires movements beyond those in the motion library, creating a need for the synthesis of new motion from pre-recorded sequences. An existing approach involves interpolation between motions in the frequency domain, with a blending space defined by a triangle network whose vertices represent input motions. It is this branch of character animation which is supplemented by the methods presented in this thesis, with work undertaken in three distinct areas.

The first is a streamlined approach to previous work. It provides benefits including an efficiency gain in certain contexts, and a very different perspective on triangle network construction in which they become adjustable and intuitive user-interface devices with an increased flexibility allowing a greater range of motions to be blended than was possible with previous networks.

Interpolation-based synthesis can never exhibit the same motion variety as can animation methods based on the playback of rearranged frame sequences. Limitations such as this were addressed by the second phase of work, with the creation of *hybrid networks*. These novel structures use properties of frequency domain triangle blending networks to seamlessly integrate playback-based animation within them.

The third area focussed on was distortion found in both frequency- and time-domain blending. A new technique, *single-source harmonic switching*, was devised which greatly reduces it, and adds to the benefits of blending in the frequency domain.

Acknowledgements

I would like to thank my primary supervisor, Professor Andy Day, who suggested doctoral studies and offered steadfast support throughout. Similarly, I am very grateful to co-supervisor Dr Stephen Laycock, whose continued help and positive outlook I found invaluable. Considerable thanks extend also to Dr Robert Laycock, co-supervisor during the earlier part of my studies.

I owe much to my family as well as to friends, whose ongoing support was a mainstay in the success of this undertaking.

I would also like to express my appreciation of the helpful feedback provided by Professor Ken Brodlie and Dr Richard Harvey, external and internal examiners respectively, from whose experience both my thesis, and I, were fortunate to benefit.

Contents

1

Introduction

1.1 Synthetic Environments and Virtual Humans

Digital virtual environments are a growing part of everyday life [Fre08, WOR, LoCIoI10]. While initially familiar through video games, television and cinema, the widespread availability of high-speed internet has seen the emergence of online environments, with increasingly popular virtual worlds for social interaction [Incb, Ope, Inca] being one such example. Further uses of synthetic environments include virtual tourism [Boo03], architectural visualisation [NHAH03], product marketing [KR07], virtual heritage [Add01], rehabilitation [Los06] and urban planning [DRRT07]. Digital worlds also serve as educational tools, for distance learning [BG04], or for teaching children by harnessing their age-old fondness of imaginary places and characters [Pej10], while military training in dangerous environments becomes safe, and less costly, when replaced by a simulated reality [ZLUR+12].

The above are but some facets of the wide spectrum of virtual existence, whose various synthetic environments will, in most cases, require animated humans within them. Thus while a flight simulator might not model people, many virtual environments do benefit from them, like the proposed work of an architect or a virtual tourism site. Furthermore, many games and virtual worlds could simply not function without avatar

interaction.

It is this abundance of synthetic environments, expanding in scope and commercial importance [Fre08, WOR, LoCIoI10], which provides the context for character animation, and thereby for a number of important research topics in the field of computer graphics [Sch10b].

1.2 The Nature of Character Animation

The need to *animate* characters sets them apart from static virtual objects such as buildings and roads. Moreover, virtual humans should be differentiated from moving objects like cars, clothing or robots, for which precise physical simulation would suffice for convincing results. Human motion is especially hard to animate, [Stu98a, AF02, PB00, LWS02], and is guided by psychological factors which physics cannot model, such as mood and intention [MFCD99, UAT95, Sch10b]. Instinct and reflexes also play a role, as does physical state such as tiredness or illness. Character animation should thus extend beyond physical realism and, ideally, convey the natural quality of human motion. Compounding this difficulty is the expertise we all share in reading our fellow humans [HWBO95, Tro02, HOB98], being able to detect someone's frame of mind, and sometimes even their gender or identity from their gait alone [Vas02, KC77a, KC77b].

1.3 Skeletal Animation

Complex animated characters are typically constructed around a central articulated skeleton, comprising rigid segments connected by joints. Human skeletal anatomy is merely approximated thereby, with a far simpler structure which, from project to project, can vary both in segment connectivity and in the number of degrees of freedom (DOFs). The skeleton is surrounded by one or more layers employing, for example, primitives to emulate the volume of muscle and fatty tissue, encapsulated by a mesh

Figure 1.1: Victorian parlour toy discs. (Left top and bottom) While inscribed "zoopraxiscope", the precise origin and usage of this disc remain open, as discs projected by that device were made of glass and without radial slits [Bra10]. (Right top and bottom) Slits, as found in the phenakistoscope disc, served as a shutter through which successive phases of the motion sequence were viewed [Les93]. (Image source: Library of Congress Prints and Photographs Division, Washington, D.C. 20540-4730, USA.)

to simulate skin, or by a combined layer for skin and clothing [BP07, CHP89, CH01, MFCD99]. Constraints between layers ensure that animating the skeleton controls the surrounding layers, bringing the outermost to life in the eyes of the user. The scope of this thesis and associated implementation work is the animation of the skeleton which drives the character.

1.4 Historical Context and Classification

This section presents a classification of the branches of character animation, after first touching upon the historical background. While the focus is on computer-based meth-

ods, it does extend, as relevant, to the manual traditional craft.

The simplest forms of character animation may, arguably, date back to prehistoric cave paintings in which animals were depicted with multiple heads, legs and tails, apparently intended to give the illusion of motion [Mar08, BES02, Bra10]. Moving images relying on persistence of vision became widespread in Victorian times as evidenced by a plethora of optical devices (Figure 1.1) [Kra04, Baz72, Les93, Jon88], while the end of this era saw the birth of film-based animation [Kra04]. Lee Harrison III pioneered the early days of *computer-based* character animation in 1961 [Stu98a, Har61, Stu98b], by patching together analogue circuitry including signal generators and summing amplifiers, to generate output on a cathode ray tube. The project evolved into Scanimate, a commercially successful system [Stu98b], whose earliest blueprints date back to 1969 [Stu98a, Sie98]. The many developments which followed created a broad field able to be categorised in various ways. The chosen classification is enumerated below, divided into mainstream general approaches and auxiliary methods which play an important role in character animation. Further clarification is given in the literature review in the following three chapters.

1.4.1 Mainstream Approaches

Computer keyframing. Keyframing, as implied by the name, involves the setting up of *key* motion frames. Also known as 'extremes' [Tho58], these frames specify spatial configurations for the character which are deemed pivotal in defining the motion being created. Keyframes are then bridged by sequences of suitably-timed [Las87, TJ95] "inbetween" frames [Las01, Cat78], whose character poses change gradually from one extreme to the other, resulting in a smooth and meaningful overall sequence. With computer-based methods, the user can specify keyframes by sketching a two-dimensional image [BW75, Kor02, Ree81], or by

setting the pose of a skeletal character [GMWC06, GCFD07], with so-called 'in-betweening' performed algorithmically, following such timing of the action as specified by the user [TM04].

Traditional animation. Keyframing originated from hand-drawn animation [Stu98a, GMWC06, Las01, Neb99], where, as set out in Chapter 2, it was one of two general approaches to creating motion sequences [TJ95, RP12]. Computer keyframing systems thus inevitably adopted some parallels with the manual process, including the inherent difficulty in timing motion to best effect [TM04, Las87, RP12], and the relevance of most of the principles of traditional animation [Las87, TJ95, Las01]. Furthermore, these principles occasionally emerge in seemingly unrelated computer science research [SPCM97, GLD08], including papers covering physics-based animation, [WK88] and signal processing techniques [BW95], quite apart from those on computer keyframing itself [Las87, Pix86]. It is this overlap between traditional and computer-based keyframing which makes the manual art a relevant inclusion.

Forward kinematics. Setting the position of a skeletal limb, or the entire character pose, by the application of given joint angles, is a straightforward and unambiguous process known as 'forward kinematics' [GMPO00, MFCD99, LBJK09]. However, in the absence of pre-determined joint angles it is not an intuitive way to adjust keyframe postures [HOB98, ZB94, Tra94, Wel93], just as people do not consider the angles subtended by their bones when stepping on, or reaching for something, but instead consider the position of hand or foot. Nevertheless, forward kinematics is widely used, even if often not specified, as, for one thing, almost *all* approaches to character animation ultimately determine joint angle trajectories, which, when applied to the skeleton, enact a progression of poses.

Inverse kinematics. Given the required location (or location and orientation) for the mobile end of an articulated arm, such as the end effector – the gripper or tool – of a robotic manipulator, inverse kinematics is the process of determining such joint angles as would position the end effector in the manner requested [Gle98, Stu98a, GMWC06]. Often referred to as the "inverse kinematics *problem*" [BC89, RGBC96, CH07], it must also contend with kinematic redundancy giving multiple, or infinite solutions [CB97], and with singularities [Wel93], where the relationship between end effector placement and joint angles breaks down. Having obtained the joint angles, it is forward kinematics which performs the actual positioning.

Notwithstanding the increased complexity, inverse kinematics makes keyframing systems palpably more user-friendly [HOB98, ZB94, Tra94, Wel93], by allowing the setting of character poses by positioning the hands and feet, with all intervening joint angles calculated automatically. It also serves as an often-used tool in animation synthesis of multifarious kinds [HWBO95, RCB98, SLSG01], able to maintain constraints such as footplants or handholds, and used for the correction of violations such as footskate (Chapter 4, Section 4.12) [LMT07, KSG02].

Physical simulation – forward dynamics. Unlike kinematics, which considers articulated structures and their motion from a purely geometrical perspective [GMWC06, Aba01, GMPO00], dynamics applies to physics-based simulation, in which geometrical configurations are controlled only as a consequence of the prior application of forces and torques (henceforth referred to as 'forces' for brevity) [WMC11, CHP89].

In character animation, *forward* (or *direct* [Ott03]) dynamics updates the character pose in response to *pre-determined* forces [MFCD99, Wel93, GMWC06]

applied to the individual skeletal segments, whose every position and orientation are calculated, at each time step, by integrating the equations of motions of the system (Chapter 3, Section 3.4) [BC89, HWBO95]. Constraint forces and torques within the joints (henceforth simply 'constraint forces') need also to have been established [BB88, Wel93, ESHD05], as these hold the skeletal segments together, by emulating, or approximating, the action of real-world joints.

Physical simulation – inverse dynamics. Calculating the above constraint forces, based on known *external* forces, positions and motions, is a problem of *inverse* dynamics [Wel93, WMC11], the handle to the solution of which lies in that joints, like other constraints, can be defined as constraint equations (or accounted for by terms within a system-wide description) [ESHD05, BB88, Ott03]. (The Newton-Euler method is considered here – alternative approaches to multibody dynamics do exist).

In the animation of jointed multibody structures, the internal constraint forces require solving at every frame before being combined with known external forces for use in forward dynamics, in turn bringing about the geometric configuration of the articulated structure, while maintaining joint coherence [HWBO95, MFCD99]. The inverse dynamics phase, however, is very expensive for complex systems [Ott03, GMPO00, Wel93].

Physical simulation – spacetime constraints. In contrast with dynamics simulations, which for each individual frame, repeatedly perform all the calculations needed to update the skeleton's state, spacetime techniques simultaneously compute the *entire* motion sequence [LMT07, GMWC06, AW01, GMPO00].

The problem is one of trajectory optimisation, which, posed over the sequence duration instead of a single frame, seeks that motion which best fits a specified set

of constraints [MTT96, LP02, MFCD99]. Although not bound to do so [Gle98, MCC09, CH07], the spacetime framework usually includes Newtonian physics constraints, ensuring compliance with dynamics laws [RGBC96, WK88], or at least an attempt to respect them [LP02].

Motion capture and playback Motion capture refers to the recording of frame-by-frame measurements of the movements of a real actor, the post-processing of raw data, and the computation of such joint angles and root (pelvis) position values, as allow a synthetic character of similar structure and dimensions to enact those very movements [Stu94, WF02, Fur99]. While the file thus created [Lan98, Mad01, Ger04] could simply be played back on a suitable skeleton, as indeed is sometimes useful [MLD+08], the motion thereby enacted would be limited to a copy of the original performance. The utility of captured motion is increased, however, by using it as source data for the synthesis of new motion, for which research has developed a number of general approaches, now described below.

Synthesis by concatenation. Also known as 'sequence rearrangement' or 'pose rearrangement', this method generates motion streams by concatenating individual clips extracted from a motion capture library, while applying seamless transitions to ensure continuity [AF02, AFO03, LL06]. Fluid joins demand compatible clips, and legal connectivity amongst those in the motion database is often expressed as a graph structure created in preprocessing [KGP02, LCL06, LCR+02]. During synthesis, suitable clips are selected and pieced together, with reference to permissible connectivity as well as to user instructions, thereby producing a longer sequence conforming to the specified task, such as following a sketched path. Smooth transitions are created by easing out the motion at the end of one clip,

while concurrently easing in the beginning of the next, exemplifying a common use of motion interpolation (below).

Motion Interpolation. Also known as 'blending', this method interpolates between two or more of the captured sequences [UAT95, RCB98, MLD10], with the contribution from each defined by blending weights, which may in turn be controlled via parameters more meaningful to the user [WH97, HG07, PSS02], like the required speed or rate of turn of the output motion. While interpolation *merges* the inputs thereby creating a blend, an appropriate choice of blending weights will induce extrapolation, a similar tool which creates motion lying outside that achievable by a simple mix of inputs. A degree of extrapolation can be useful, though beyond that the animation may look cartoon-like, and, if extended further still, the motion will quickly fail.

Frequency domain blending [UAT95, BW95, PL06] is a subset of motion interpolation, and is the context for the methods developed and presented in this thesis [MLD10]. In the frequency domain, or 'Fourier domain' [KG03, UAT95, PTVF92], the data represents spectral content – values, such as phase angle, attributed to the component frequencies which make up the motion – as compared to the time domain with its more familiar time-varying quantities such as joint angle trajectories. Interpolation, extrapolation and various motion adjustments can be performed in the frequency domain, before returning to the time domain, and thus converting the data to that form required to animate the character.

Statistical methods. Statistical methods learn the idiosyncrasies of motion capture data, which they encompass in compact mathematical form, allowing them, to some extent, to generalise and synthesise characteristically similar motion [LWS02,

PB00, MCC09], even using it to fill gaps where data was missing in the source motion [CC10].

Motion warping and other motion editing. Such methods, as used, for example, in [LCR+02, UAT95], modify a captured or keyframed sequence, by the adjustment of motion amplitudes and timing, and may also include blending. A straightforward example is that of a tennis shot, in which the racket is moved higher or lower by editing the joint angle trajectories, as demonstrated in the seminal paper by Witkin and Popović [WP95] who introduced the term 'motion warping', an expression since become common parlance.

Dynamic timewarping [BW95, KG03, SO06], extensively explored in speech recognition [Sen08, BW95] and found in numerous fields besides [Sen08], is used in character animation to distort skeletal trajectories along the time axis, such that events in each motion with synonymous meaning occur at identical times. The mutual alignment of timewarped motion ensures synchronicity of movement prior to blending, making the technique an important element within some mainstream approaches.

Procedural methods. As the name suggests, procedural methods methodically execute a prescription for motion creation [HOB98]. Unlike the approaches above, which process ready-made data such as captured or keyframed motion, procedural methods dispense joint angles to the skeleton based on biomechanical or empirical knowledge of human locomotion [MFCD99], a simple example of which is the breakdown of the human walk cycle into phases, as shown in Figure 1.2 and universally relevant to character animation. The procedural category is not strictly defined [GBT06], sometimes even said (implausibly, perhaps) to encompass the fields of dynamics or spacetime methods per se [GMWC06, HOB98]. It

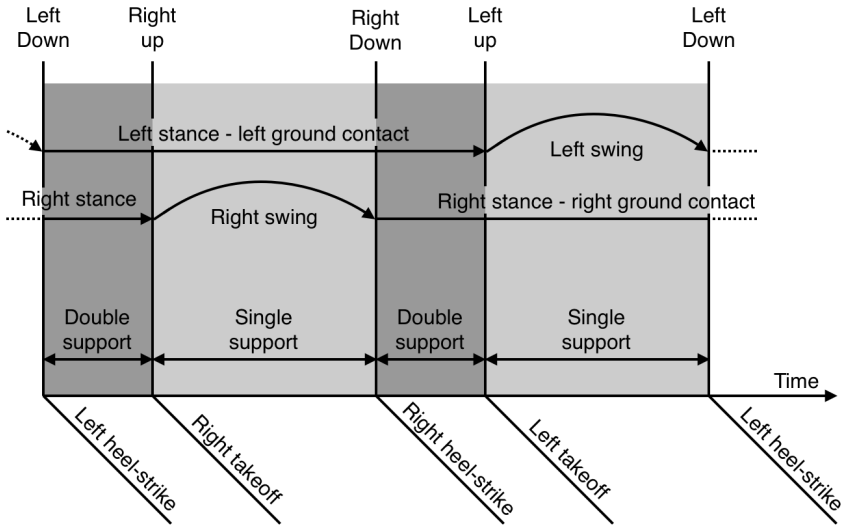

Figure 1.2: Simple breakdown of a complete walk cycle (two steps) into its constituent phases, foot strikes and takeoffs (toe-offs [BC89, RCB98, GBT06]). (Image based on [BC89, MFCD99, GBT04a, GBT04b, AW01].)

is certainly based to a great extent on kinematics as concurred by [PSS02]. To avoid nebulosity and reduce overlap between categories, the classification used in this thesis does not include an explicit procedural method category, instead allocating such techniques to whichever other category their attributes most clearly befit.

1.4.2 Auxiliary Methods

The methods below, while auxiliary, are nevertheless essential in character animation and the focus of dedicated research papers, just as applies to individual techniques within the general approaches above.

Mapping marker data to DOF values. Raw marker data obtained from motion capture recording sessions differs from, and needs converting to, the joint angle and root trajectories typically found in motion files and employed by researchers

and end-users. Mapping data from an optical capture set-up, for example, uses recorded and post-processed streams of marker position data to animate a virtual skeleton of size and proportions similar to the real-life actor, as in [ZVDH03], or by the use of commercial software such as Autodesk MotionBuilder [Aut], and it is this animated skeleton which then yields the DOF values to be saved to file in a standard format such as BVH [Mad01, Lan98, Sch10a].

Retargeting. The DOF values in a motion file created and stored as above, apply only to skeletons of the same size, proportions and skeletal structure as that used for the mapping process from which the file arose. Playing back this data on a skeleton of different size or morphology will generate motion which no longer reflects that of the original human actor, and which, typically, will be beset with artefacts such as footskate or ground penetration [PSS02, MLD$^+$08, LMT07]. To use the existing motion file on a different kind of skeleton, retargeting is performed, as in [SLSG01, Gle98], a process which attempts to adapt the motion defined for one skeleton, for use on another, without unwanted distortion.

Character rigging. Rigging aims to ensure proper deformation of the skin when using the layered approach (Section 1.3) [BP07, WH97, CH01] to character modelling. It is mentioned here for the sake of completeness, but not further discussed in this thesis, which, instead, focusses only on the internal skeleton like almost all work in character animation.

Rigging involves complex mapping from the articulated skeleton within, to the character surface model, and has become widespread practice [CHP89, PCLS05]. A common approach uses skinning algorithms with per-vertex weighted combinations of skeleton joint transformations, which, however, suffers many disagreeable artefacts [MG03, LCF00] and does not model muscle bulging [PCLS05]. One way

to obtain a higher quality result is by simulating internal musculature for more natural-looking surface deformations [PCLS05, CHP89, MG03]. Automated solutions to both of these time-consuming tasks can be found in [BP07] and [MG03] respectively.

Cyclification. Many human motions such as walking or running are cyclic in nature, repeating similar movements (usually two steps) over and over. Approaches to locomotion synthesis sometimes exploit this, by modifying individual cycles extracted from source data, in such a way that they can be "wrapped-around" and played back endlessly without discontinuity. The prerequisite modification, dubbed 'cyclification' [AMH03, RGBC96, GBT04b], is the process of adjusting the motion cycle, notably matching its endpoints, so it becomes a continuous periodic sequence which when concatenated yields smooth-looking motion.

Footskate correction and other post-processing. With character animation the subject of ongoing research, it is evident that perfect motion, fulfilling all requirements, cannot at present be synthesised. One common failure is the production of unwanted artefacts, of which footskate – the undesirable sliding of the foot during ground contact when it should remain planted – is perhaps the most mentioned [GBT04b, CH07, AF02]. Others include foot-ground penetration, hovering feet which fail to be firmly planted, and analogous aberrations with handholds or reaching actions. Jerkiness is another, whereby even slight discontinuities in the path or velocity of just part of a character immediately stand out. In the literature, the correction of aberrations is often delegated to post-processing [BW95, GBT06, ABC96]. Examples of footskate remedies are given by [LMT07, KSG02].

Much work in character animation is hybrid in nature, belonging to more than one class. Furthermore, methods shown to use motion capture as source data will often work equally well with other types of input, such as keyframed motion. Moreover, intended objectives, such as smoothness of motion or correction of footskate, are not necessarily fully achieved by the methods intended to do so. The above classification thus presumes flexibility of interpretation.

1.5 Research Motivation

1.5.1 Frequency Domain Methods

As outlined in the previous section, animation synthesis is usually performed in the time domain, thus involving the manipulation of motion expressed as time-varying data values. The frequency domain provides an alternative representation, in which motion is described by parameters pertaining to the specific frequency content of a motion sequence. Although comparatively scarce, frequency domain methods are an established subfield of character animation. It is the potential control available through manipulations of individual frequencies within character motion that motivated research into frequency domain methods.

1.5.2 Chapter-specific Research Problems

The implementation work presented in this thesis, covers three different areas each with its own more tightly focussed underlying motivation, as detailed in the relevant chapters (5, 6 and 7) and reiterated while concluding in Chapter 8. Concise, preliminary descriptions of these chapter-specific motivations follow below.

Chapter 5 – motivation. Investigation of the existing literature revealed interesting work in the frequency domain as detailed in Chapter 4, but it showed limited util-

ity for the important context of real time user-driven character navigation in virtual environments. The most pertinent work, that of Pettré and Laumond [PL06], defined the range of motions able to be synthesised in the frequency domain by a triangle-based network as detailed in Chapters 4 and 5. This network was well suited to the motion planning focus of their work. It was, however, far less suitable for user-control in interactive applications, with limited capability for user-friendly character-navigation, and a significant restriction in the range of motions it could be used to synthesise. These deficiencies are addressed by the more generalised triangle networks presented in the method of Chapter 5. Dubbed the *streamlined approach*, it has further benefits, including lower runtime costs than the method of [PL06], which provided additional incentives for the work of that chapter.

Chapter 6 – motivation. The streamlined approach, like much of the previous frequency domain work [UAT95, PL06, BW95] employed blending-based synthesis. Motion blending, whether in the time or the frequency domain, benefits from precise control enabling any of a continuous range of motions to be animated. The variety of motion within this spectrum, however, is potentially lower than is achievable by concatenation synthesis (Section 1.4.1, above), as further explained in Chapters 6 and 8. This inspired the solution presented in Chapter 6, a hybrid method which complements the frequency domain blending of the streamlined approach with seamlessly integrated time-domain motion sequences, resulting in benefits including a considerable increase in motion diversity.

Chapter 7 – motivation. Experiencing and correcting unwanted artefacts is to be expected when developing character animation methods, as indeed was the case during practical work for the above-mentioned chapters. One aberration, how-

ever, occasionally observed while motion blending, had so far remained unsolved. Taking the form of intermittently jerky motion, investigation showed it to be equally present in time domain blending. This motivated, in Chapter 7, a novel frequency domain method termed *SSH switching*, which is shown to successfully counteract the offending jerkiness while avoiding brute sacrifice of upper-harmonic content.

1.6 Research Contribution

The practical research work associated with this thesis builds upon previous frequency domain methods, in which motion captured data was processed by interpolation, extrapolation and by editing of various kinds. The work presented here breaks down into three distinct phases. The most significant contributions are collated below, with full clarification given in Chapters 5, 6 and 7.

Phase 1 – Streamlined approach. The first area, covered in Chapter 5, consolidates and extends existing ideas from frequency domain methods, presenting a novel package [MLD10] especially, (though not exclusively), useful for games or other virtual environments with user-driven characters.

It introduces the interface-orientated triangle network (Chapter 5) to guide interpolation in the frequency domain. Benefits include greater user-friendliness than in previous frequency domain methods [UAT95, PL06, BW95], an ability to blend a greater range of motions than in similar recent work [PL06], and further potential which was later exploited in the second research phase (hybrid networks).

The method is also significantly cheaper than that of [PL06] in certain contexts outlined in Chapter 5. While this mostly results from using a cheaper formula

for Fourier synthesis, Unuma et al. [UAT95] had done so too, and the contribution lies not in this trivial point, but instead in using the correct approach to phase angle blending, omitted by [UAT95], which, when applying this formula, is imperative for the consistent synthesis of properly blended motion.

The steps to implement this streamlined approach are presented in a methodical fashion, effectively as an instruction manual, to assist anyone wishing to use it.

Phase 2 – Hybrid networks. Interpolation synthesis, as used in a variety of differing methods [HG07, BW95, KG03] including the streamlined approach to frequency domain blending [MLD10], above, allows precise continuous changes in the style of the motion, and thus fine control over qualities like rate of turn or speed. Such blending-based methods cannot, however, exhibit the same *variety* of motion as the simple playback of motion clips used in synthesis by concatenation [AFO03, KGP02, LL06], for reasons explained in Chapter 6 which, as the second research focus, addresses this limitation with the development of *hybrid networks*. These novel structures exploit the mechanism underlying the interface-orientated triangle networks devised in the initial research phase [MLD10], to seamlessly integrate within them, varied clips for motion capture playback, thus amalgamating two quite distinct synthesis methods, interpolation-based and motion playback.

Hybrid networks extend previous frequency domain interpolation methods as follows:

- The inclusion of clips enacting any motion type, of any length, allows the addition of great variety to the inherent near-homogeneity of motion in blending-based approaches.
- The bridging of disparate blending networks with motion clips is a preferable, sometimes necessary alternative to bridging by interpolation-based transi-

tions, when the latter would either look poor or would fail altogether, as further detailed in Chapter 6 Section 6.8.1.

- In addition to the above, as described in Chapter 6, hybrid networks show improvements over previous approaches, notably greater user-friendliness.

Phase 3 – SSH switching. Perhaps the greatest challenge in character animation, and an inconceivably time-consuming one, is the quest for high quality output. The third research phase addresses one facet of this challenge, investigating a motion aberration which is shown in Chapter 7 to affect both frequency- and time-domain blending. Analysis reveals its nature and source, prompting the name 'upper-harmonic distortion' (UHD). A procedure, termed 'single-source harmonic switching' (SSH switching) is then formulated which greatly reduced it. SSH switching, expounded in Chapter 7, adds tangible benefit to the field of frequency domain blending while even lowering runtime costs.

All three phases are accompanied by demonstration videos, for which download links are given in Chapters 5, 6 and 7, as well as in Appendix E. All demonstrated motion is synthesised in real time and shown with no post-processing corrections of any sort.

1.7 Literature-based Implementations and Videos

To ensure and demonstrate a proper understanding of selected previous work [BB88, WB97, Wel93, ESHD05], additional implementations were carried out. Videos are provided for demonstration purposes with download links given in the relevant literature review chapters, and in Appendix E. Inverse kinematics is demonstrated, as is inverse dynamics simulation. (Videos of motion capture playback and of frequency domain blending were superseded by demonstrations accompanying research presented in Chapters 5, 6 and 7, and are thus excluded as no longer relevant).

1.8 Literature Review – Comparison of Performance

Computational expense is an important consideration when comparing previous work. However, frame rate and timing figures are sometimes missing, while qualitative statements on performance appear volatile, and quickly supplanted by different views [LCL06, PSS02] in following publications. Claims of 'interactive' performance are frequent, but do not imply any specific time values and in the absence of further information [KG03, MCC09] are highly subjective, and even the expression 'real time', while more precise and currently meaning at least 30 frames per second, has been used for lower rates in less recent papers [UAT95, LCR+02]. Furthermore, while Moore's law [Moo65] famously predicted a doubling of the number of transistors on a chip about every two years, computing power itself has doubled every year [Ott03] even *before* accounting for growing memory bandwidth or increasingly powerful graphics processing units (GPUs) [NVI12]. The methods proposed in less recent past work would thus clearly run faster on modern hardware, but how much faster is hard to evaluate, especially as some may be suitable for parallel processing on the GPU, and others not [NVI12, ND10].

For the above reasons, no global comparison of the performance of existing methods could be given when reviewing the literature. Instead, when describing past work, selected results are illustrated and put in context by highlighting the publication date.

1.9 Thesis Structure

Computer-generated character animation has grown into a broad field since the early 60s [Stu98a, Har61, Stu98b], with research providing contributions from multiple disciplines including physics [HWBO95], signal processing [BW95], robotics [Wel93], biomechanics [LP02], traditional animation [Las87], control theory [TLP07], artificial intelligence [LL06], speech recognition [KG03], and of course computer graphics and math-

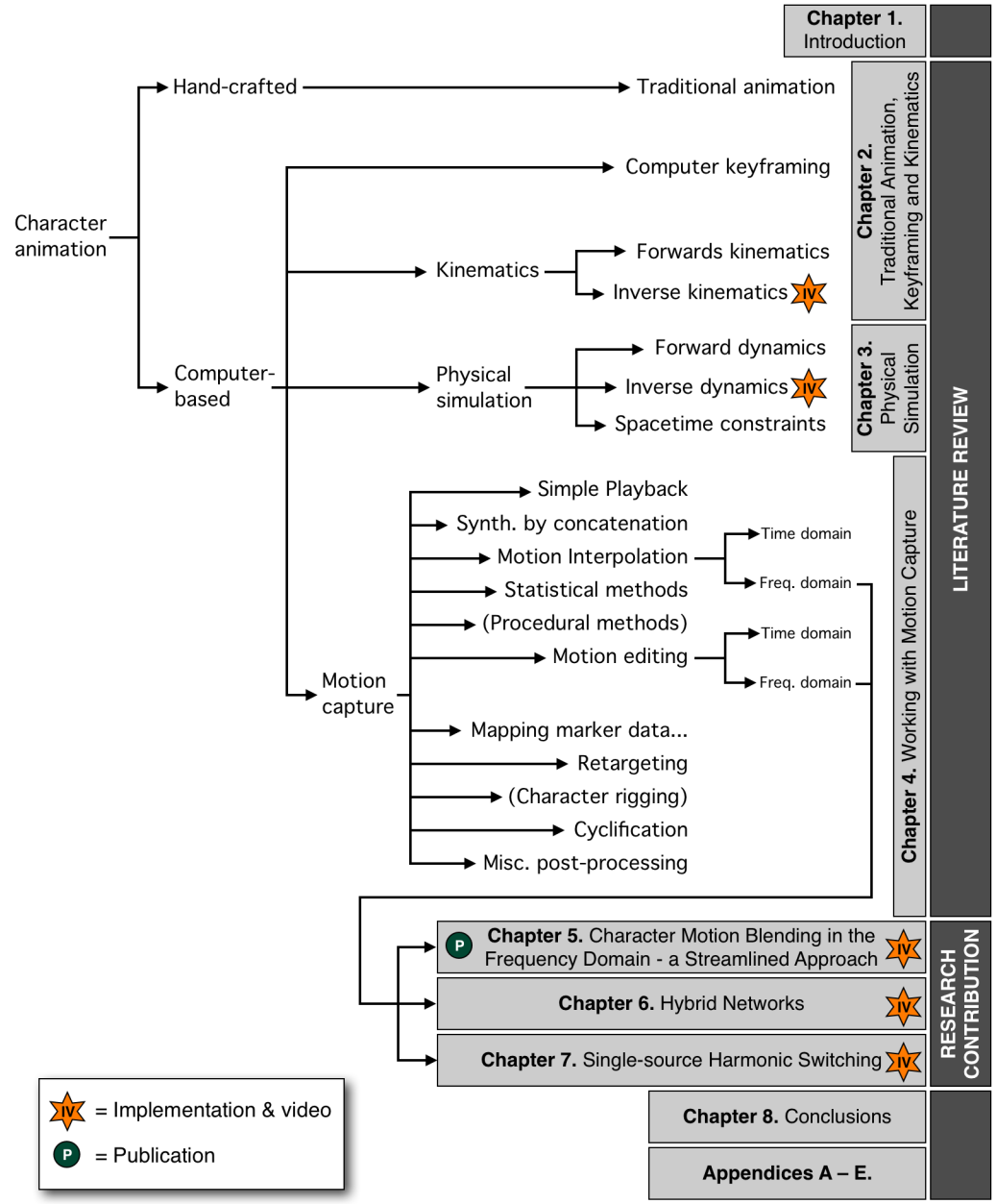

Figure 1.3: Thesis structure, demarcating literature chapters from research contribution. The relation with the classification of animation techniques described in Section 1.4 is shown, while the research contribution is seen to lie within the context of frequency domain methods. Published work is highlighted as are demonstration videos of implementations which, as shown, were also undertaken to consolidate the literature section.

ematics, while even touching upon psychology [MLD+08] and acting [BD09]. Selected research from this vast body of literature is reviewed in Chapters 2, 3 and 4, weighted towards seminal papers and considering individual works in depth, to properly convey their true nature and contribution.

Chapter 2 covers computer keyframing and hand-drawn animation which it is partly derived from. Forward and inverse kinematics are included too, and relevant to keyframing as well as serving as a general tool used in many other methods. The following chapter considers physics-based simulation comprising forwards dynamics, inverse dynamics and spacetime constraints. Working with motion capture, an extensive field, is then described in Chapter 4. Chapters 5, 6 and 7 present novel work, the streamlined approach to frequency domain blending [MLD10], hybrid networks and SSH switching respectively. The research contribution thereby put forward is outlined in Section 1.6 above. Chapter 8 provides concluding remarks, consolidating previous content. Appendices then follow, which include background information into the discrete Fourier transform and Fourier synthesis, and derivations of a formula presented in Chapter 5.

The structure of this thesis is set out in Figure 1.3, which furthermore highlights the following.

- The relationship between the literature review of Chapters 2, 3 and 4 and the classification of character animation techniques described in Section 1.4 (above).
- The delineation of chapters presenting research contributions, and those reviewing literature.
- The context of the research contribution, shown as strictly limited to the subfield of frequency domain methods.

- Demonstration videos for practical work, seen to accompany each of Chapters 5, 6 and 7, as well as previous work in the literature review on inverse kinematics and inverse dynamics. Publication is indicated also.

2

Traditional Animation, Keyframing and Kinematics

2.1 Introduction

Computer-based character animation followed simpler devices for creating the impression of motion, like the parlour toys of Victorian times [Kra04, Baz72, Les93, Bra10], a prehistoric thaumatrope, perhaps, [DFD08] and film-based animation for public viewing, with the single-user kinetoscope presented in 1893 [Tal12] and, for larger audiences, projections from the cinématographe of the Lumière brothers in 1895 [Cou06]. It is the motion picture industry, albeit keyframing as used by the artists of animated films, which computer-based animation developed from in the early 1980s [Stu98a, Las87], though exceptions exist such as [Har61, Sie98, Cen07]. This chapter thus includes traditional animation, notably principles born from it which may surface – even unintentionally [WK88] – in computer animation too, before describing algorithmic approaches to what previously was achieved with pencil and brush.

Figure 2.1: Parietal art from the Upper Palaeolithic. (Left) eight-legged bison dated ca. 31000BP [Val03] from the cave of Chauvet-Pont-d'Arc in France, representing prehistoric animation, according to Azéma [Mar08]. (Right) drawing of eight-legged boar by Abbé Henri Breuille published in Illustrated London News in 1912, depicting the original found in Altamira cave, Spain, for which artwork was, at the time, dated 20000–30000BP. (Source, left image: Ministère de la Culture et de la Communication, Direction Régionale des Affaires Culturelles de Rhône-Alpes, Service Régional de l'Archéologie. Source, right image: Library of Congress Prints and Photographs Division, Washington, D.C. 20540-4730, USA.)

2.2 Classical Animation

The origin of hand-drawn animation may, possibly, be traced to pre-history. In his PhD thesis on the representation of movement in the Upper Palaeolithic era, Marc Azéma [Mar08] analysed parietal art at 141 sites in France, of which 140 were caves or painted rock shelters, as well as one single open-air engraved rock face. Of 3763 zoologically determined animals, 41.1% were found to depict "animation concerning the body of the animal as a whole or one of its parts" with such claims (from earlier publications) being in part concurred by Brulé et al. [BES02]. By the "decomposition [breaking up of movement] by superimposition of successive images" the Palaeolithic artists are said, by Azéma [Mar08], to "seek to formulate graphically the fourth dimension, that is, time". Figure 2.1, left, gives an example of this technique from the Chauvet cave in France portraying a bison with eight legs. A differently-styled eight-legged boar, right, is a further case from Altamira cave in Spain (outside the research

Table 2.1: Five of the principles of animation enumerated in [Las87, TJ95, Las01, RP12], found most relevant to this thesis. The descriptions are expanded in the following text.

Animation Principle	Description in Brief
Squash and stretch	Shape deformations maintaining volume
Anticipation	Anticipatory move building audience expectation of what follows
Follow through	Continuation of action beyond termination point
Timing	Modulates speed of action and thus its meaning
Slow in and slow out	Adjustment of inbetween density, emphasising extremes and adding zest

scope of [Mar08]).

More recent techniques for hand-drawn animation are enumerated in 'Disney Animation: The Illusion of Life' (or its reversed-titled reworking 'The Illusion of Life: Disney Animation' [TJ95]). Often referred to as the animation bible [RP12], it lays down twelve principles of traditional animation of which most apply to computer animation too, occasionally surfacing in computer research even when not by design [WK88, BW95], and, employed, famously, in the short film *Luxo Jr.* [Pix86] which, having drawn on these principles, received a standing ovation at SIGGRAPH '87 even before screening concluded. In [Las87], John Lasseter, the creator of Luxo Jr., an animated baby anglepoise lamp, describes the application of eleven of these principles of 2D hand-animation to its 3D computerised counterpart (eleven only, presumably, since much of the principle of solid drawing – conveying the volume, solidity, weight and three-dimensionality of a character – is performed by the graphics software already).

Table 2.1 lists selected principles which, [Las87] aside, emerge in research papers discussed in this thesis, or which otherwise seem especially pertinent. See [Las87, TJ95, Las01, RP12] – on which both the table and following details are based – for the full picture.

Squash and stretch. Seen as the most important of all principles and employed, or

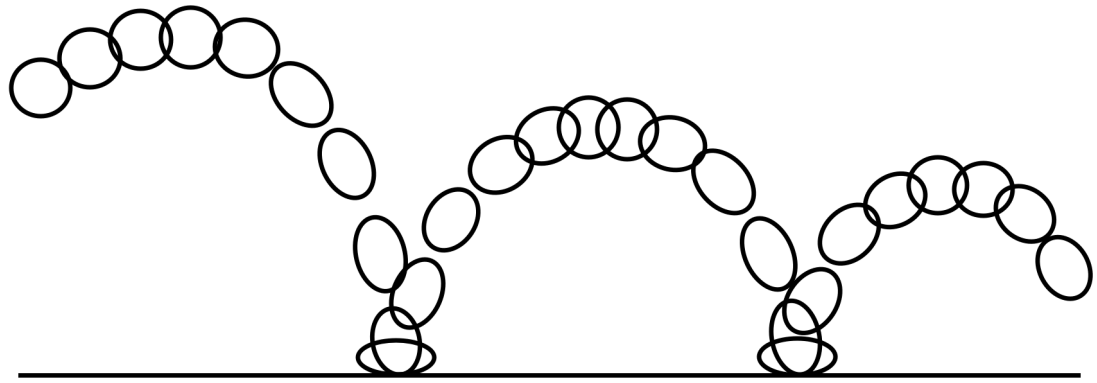

Figure 2.2: Classical bouncing ball example of the principle of squash and stretch, conveying the springy resilience of the ball itself, as well as its varying speed and the force of impact upon hitting the ground.

found to emerge, in [SPCM97, GLD08, WK88, BW95]. It shows deformation of shape during motion, as with muscle bulging or deformation of the cheeks when smiling, the extent of which indicates plasticity or rigidity of substance, conveying, for example, the fleshy quality of the body or its parts, or the solidity of many an object. A classical, and elementary demonstration, is the bouncing ball shown in Figure 2.2, where, moreover, the deformations emphasise the impact of the bounce and the heightened speed either side of it. Another, quite different use of this principle, is the stretching out of a fast-moving object so its positions overlap from frame to frame, helping reduce the strobe-like perception of separate images in succession instead of one in continuous motion.

Anticipation. Referred to in the research of [BW95], it is the preparation for an action which is thereby anticipated, as shown, for example, by body stance before throwing an object, or by a cartoon character's gearing up before rushing off at speed. Figure 2.3, right, gives a simple illustration. Anticipation can be natural, dictated by anatomy and physics, as when a cat prepares to pounce, though such

Figure 2.3: (Right) Squash and stretch, and anticipation – springy quadruped prepares to pounce. (Left) Follow through – braking hard to a standstill, eyebrows fly past, before returning.

action, in animation, is exaggerated for effect. It serves also to prime the audience, focussing attention upon imminent action, perhaps making them aware of events off-screen, as when a character suddenly looks sideways into the distance, ensuring the viewer awaits an impending action, and notices it all the more when it enters the frame, even if at speed.

Follow through. Akin to the familiar term from golf or tennis, where the action, in this case striking the ball, does not cease abruptly upon termination, but continues beyond that point. An animated example might be a dancing character whose long dress continues to move when her twirling motion stops. Follow through can also be used to convey a sense of the "weight" (mass). If an animated character staggers backwards upon catching a large object thrown for him to catch, the extended follow through conveys that the object was heavy, while the absence of follow through if catching a beach ball would show its weight to be trivial. Follow through is mentioned in physics and signal processing papers [WK88, BW95], and illustrated in Figure 2.3, left.

Timing. The timing of frames in an animated sequence controls the speed of the action, greatly affecting the message it conveys, such as emotional content. A hand withdrawn quickly from an object it just touched might indicate irritation,

or an unpleasant shock like burning, but the same action performed slowly could convey appreciation or wonder. Manipulation of timing can also secure audience attentiveness and their understanding of the scene portrayed.

Slow in and slow out. Traditional animation can be drawn 'straight ahead', where the artist creates each frame in succession from the start to the end of the scene. Another approach is 'pose to pose', drawing keyframes at key moments called 'extremes', with an assistant (or the animator) creating frames – 'inbetweens' – to span the gaps between keyframes. Computerised keyframing systems use the latter approach, inbetweening automatically as per user instructions. Slow in and slow out adjusts inbetween-frame density to be higher near keyframes and lower when midway between them, thus drawing attention to carefully crafted extreme poses, and moreover, enlivening the animation with the sprint from keyframe to keyframe. Were it not for slow in and slow out, one option for inbetweening would be spherical linear interpolation (SLERP) of joint rotations [Blo04]. However, to allow for easing in and out inbetweening is usually controlled by user-adjustable splines. In practice separate splines can be used to control the *spatial* positioning between extremes, of characters or their parts, and to control the *temporal* spread of keyframes. Furthermore, different keyframes might be used for different aspects of a character model. When Luxo Jr. jumped forward with a hop, only two keyframes – where he takes off and lands – were used to define x-translation. But animating the anticipation, squash and stretch and follow through, required constant adjustments to Jr.'s anglepoise-lamp arms and keyframes at just about every frame.

2.3 Computer Keyframing

Terra and Metoyer [TM04] addressed the difficulty for the novice in timing sequences correctly during computer keyframe animation. (The importance and impact of timing were discussed in Section 2.2). Their motivation arose from noting that while simply setting up keyframe poses themselves caused no difficulties, the proper spacing of these keyframes in time did. Nevertheless, they believed that the user could, in fact, visualise the correctly-timed action and could easily have acted this out with his hands, hence that the difficulty lay only in conveying this information via the provided keyframing interface. Their work thus lay in the creation of an intuitive tool to replace it, explicitly targeted at keyframe *timing* only, while leaving spatial editing intentionally untouched.

Their implementation took the form of a plug-in for the Maya modelling and animation package [May], and was used to specify the timing for an already-keyframed sequence. Driven by a 2D device like a mouse or graphics-tablet, a sketching interface allowed the user to draw directly within the animation window, using any convenient camera viewpoint. A point on the character – or alternatively an inverse kinematic handle in space treated as rigidly connected to the character – was chosen as the animation target. This target point traced a path during animation, highlighted within the interface. The user then traced a rough likeness to this path on screen, which by the *speed* of his motion acted out the desired timing.

The correlation between the user-traced path and that of the keyframed object was then established by one of three methods. The first was based on finding local maxima and minima in the horizontal and vertical directions, as seen from the currently selected viewpoint. The second was a discrete dynamic programming curve matching [SB94] method, which while more robust, was also more expensive. A third option allowed manual editing of feature correspondence as this could not always be identi-

fied automatically. Having spatially mapped one motion to the other, the animation keyframe timings were adjusted as defined by the time-stamped samples composing the user-sketched path.

It was, however, only the *keyframes* in the original animation whose timing was adjusted by the method, not the inbetween frames. The latter were simply linearly interpolated between the repositioned keys. This is surely one reason why, while the demonstration videos did show the method to function per se, and the interface did seem easy to use and hence successful, the created animations themselves were not fully convincing. A comparison with the difficult task of keyframe timing using Maya alone was not provided, however, and would probably have revealed greater value in the method of Terra and Metoyer [TM04].

Instead of the timing of keyframed motion, Gunawardane et al. [GCFD07] addressed the spatial positioning of characters in software packages such as Maya, for which "due to the sheer number of adjustable parameters and settings in these interfaces, skills of a master animator are required to create satisfactory results". Gunawardane et al. [GCFD07] bypassed complex interfaces, by instead allowing the user to specify keyframes by setting the pose of an artist's doll.

The doll had joints of different colours for easy identification in software, which determined their 3D locations by processing images from a stereo camera set-up. A process of linking keyframes to motion capture data (Chapter 4) to enhance naturalness was also planned, but not further described.

The software detected poses by searching for the largest connected components of each colour in each image from the cameras, the centres of which were used to calculate the joint positions by stereo triangulation. Joint locations in combination with knowledge of the hierarchical structure of the articulated figure allowed the 3D poses to be extracted.

The method of Gunawardane et al. [GCFD07] was used to generate 3D poses as sequential keyframes in Maya, and, though few results were given, it does appear intuitive and potentially useful. A drawback is the need for additional hardware, which, however, was far cheaper than the $14500 (as per 1995 publication) required for a similar method of Esposito and Paley [EPO95] using a robot-like structure with potentiometers at each joint. Furthermore, although the idea of using motion capture data to assist keyframing was mentioned only as future work, it appears, as a general idea, one worthy of pursuing.

The above methods aimed at distancing the user from complex and unintuitive interfaces for keyframe specification. An even more direct input method is provided by performance animation, where, by means of motion capture technology, the user literally acts out the required motion. Also known as computer puppetry [SLSG01, Stu98b], the method is described in Chapter 4, Sections 4.1 and 4.10.

2.4 Forward and Inverse Kinematics

Forward and inverse kinematics are concerned with jointed chains, such as robot arms – known as manipulators – or the limbs of an animated character. The joints are usually revolute (rotational) joints for skeletal chains, but also include prismatic (translational) joints for robot manipulators. The term 'end effector' is used to describe the end of the chain, and is given somewhat different meanings in the literature. For the purpose of this review the end effector will simply refer to the last link in the kinematic chain, such as a gripping tool at the end of a manipulator or a hand at the end of a skeletal chain.

An example of the use of kinematics in character animation might be to create a pose while keyframing. Doing so by directly adjusting the joint parameters is referred to as forward kinematics [Aba01, ZB94, GMPO00]. While it offers full control over

the articulated limbs, it is not an intuitive approach. Inverse kinematics makes this easier [HOB98, Wel93, Tra94], requiring only that the user specify the desired location, or both location and orientation of the end effector, via the user interface [Gle98, Stu98a, GMWC06]. The inverse kinematic algorithm then computes all the joint angles necessary to adjust the kinematic chain, such that the end effector becomes positioned as previously specified. In a similar manner, the foot of a walking character could be constrained to a specified spot on the ground [RCB98, HWBO95], while angles for the hip, knee and ankle which maintain it there are calculated using inverse kinematics.

The inverse kinematic problem is solved iteratively, based on the following approximation.

$$\Delta\mathbf{q} \approx \mathbf{J}^{-1}\Delta\mathbf{x} \tag{2.1}$$

The vector $\Delta\mathbf{x}$ is a desired change in position, and possibly orientation, of the end effector (hence a 3 or 6-vector). $\mathbf{J}$ is a Jacobian matrix of partial derivatives of end effector position (and orientation) with respect to each of the joint angles in the chain. A vector $\Delta\mathbf{q}$, of increments to the joint angles results, which approximately brings about the intended change in the end effector variables. The process then repeats, until the end effector is within a specified tolerance of the intended goal.

Inverting $\mathbf{J}$ is not possible for kinematically redundant chains, that is, those with more degrees of freedom than are necessary to meet the goal. This is because the Jacobian matrix is not square in such cases, but even when it is, the matrix may not be invertible. The pseudo-inverse can then be used instead. However, calculating either type of inverse is an expensive operation which must be repeated each time the joint angles are incremented. Accordingly, techniques exist to avoid calculating the inverse altogether.

Welman's thesis [Wel93] states, rather surprisingly, that the Jacobian transpose can

be used instead of its inverse, explaining it with reference to the equation

$$\boldsymbol{\tau} = \mathbf{J}^T\mathbf{F} \tag{2.2}$$

whose derivation is based on the principle of virtual work. $\mathbf{F}$ is a composite external force, comprising force and torque components, applied to the end effector. The composite force arises by considering the error between the end effector's current location (and possibly orientation) and its desired goal position, as a force pulling it towards that goal. $\boldsymbol{\tau}$ is a vector of internal generalised forces which Welman states can be considered to be the joint angle accelerations, but which he, in the interest of simplicity, considers instead to correspond to joint velocities. This is akin to the first-order physics simplification, as used, for example, in [GW91], in which $\mathbf{F} = m\mathbf{v}$ instead of $\mathbf{F} = m\mathbf{a}$, whereby m is the mass of a single particle or rigid body, $\mathbf{F}$ the applied force (or the total force if many are applied), and $\mathbf{v}$ and $\mathbf{a}$ refer to linear velocity and acceleration respectively. The aforementioned joint angle velocities allow the joint angles to be updated by means of a single integration step. The process then repeats, with the difference between the end effector's current and goal states giving rise to a new composite force, allowing the end effector to approach the goal with successive iterations. A scaling matrix can be added to Equation 2.2 to adjust the stiffness of individual joints.

Welman [Wel93] also presents a second inverse kinematic method which, like the Jacobian transpose method, dispenses with matrix inversion. Cyclic coordinate descent (CCD) is a technique whereby each iteration involves considering the degrees of freedom in the articulated chain separately and in sequence, from the most distal joint to the first. Each joint variable is adjusted in turn so as to reduce the magnitude of an objective function whose value is calculated from both the positional and rotational end effector-to-goal errors. The problem to solve in respect of each joint variable is simple, and can be treated analytically and hence quickly. Once the jointed chain

has been traversed, the process repeats, until the end effector is within an acceptable tolerance from the goal.

Kinematic singularities are configurations in which the end effector loses one or more degrees of freedom of motion. This can be illustrated by considering the kinematic chain for a fully outstretched arm, where small joint angle changes are seen to move the end effector vertically or side-to-side, but no longer along the axis of the outstretched arm which has thus lost that degree of freedom. In the neighbourhood of a singularity changes in end effector position can be associated with large increments in joint angles, leading to oscillations about the singular configuration. The cyclic coordinate descent method is an exception to this, which obviates the need for extra measures to achieve numerical stability such as using a smaller integration step size or clamping joint velocities to some maximum bounds before integrating.

Inverse kinematics in character animation will often involve the explicit maintaining of constraints, such as ensuring feet do not slide or sink into the ground, and that joint angles do not exceed realistic values. In respect of the latter, Welman, [Wel93], states that – although often not recommended – simply clamping joint values to their limits appears to suffice in practice. However, experience (of thesis author) does seem to suggest that clamping works only in limited cases and can lead to instability in the kinematic chain.

Welman also describes how constraints can be used with the above-described Jacobian transpose and CCD methods. The latter case simply involves an enhancement of the CCD method, which allows branching of kinematic chains into subtrees thereby allowing multiple end effectors which can be individually clamped. This might be useful, for example, when creating a pose for a skeleton, as it would allow constraints during editing which prevent previously positioned body parts from moving. A recursive scheme is employed whereby a joint is adjusted only once all its children have been

incremented. Joint updates minimise the sum of the position and orientation errors over all end effectors distal to that joint.

In some cases this extension to the CCD method only approximately enforces constraints however, unlike a more comprehensive approach Welman elaborates on which is based on the Jacobian transpose method described above. The idea, taken from existing work in constrained dynamics within physical simulations, is to assume the constraints are already met, and simply need to be maintained. Thus the task reduces to that of ensuring changes to the system do not violate the constraints.

If $\mathbf{C}$ is a vector of scalar values indicating the extent to which the constraints in the system are satisfied, such that, for example, a position constraint on the end effector would contribute three rows, and if $\mathbf{q}$ is the vector of values giving the system state (the joint variables) we have

$$\mathbf{C} = f(\mathbf{q}) \tag{2.3}$$

However, since the only requirement is that changes to the system do not violate the constraints, the constraint vector should remain unchanged, thus

$$\dot{\mathbf{C}} = \frac{\partial \mathbf{C}}{\partial \mathbf{q}}\dot{\mathbf{q}} = 0 \tag{2.4}$$

Given that the transpose method considers $\boldsymbol{\tau}$, the internal vector of generalised forces of Equation 2.2, to correspond to joint velocities, it can be substituted for $\dot{\mathbf{q}}$ in Equation 2.4 which leads to

$$\mathbf{J_c K g_c} = -\mathbf{J_c K g_a} \tag{2.5}$$

where the generalised forces have been broken down into an applied component $\mathbf{g_a}$ and one due to constraining forces, $\mathbf{g_c}$, which prevent the applied component from violating

the constraints. $\mathbf{J_c}$ is the constraint Jacobian $\frac{\partial \mathbf{C}}{\partial \mathbf{q}}$ and $\mathbf{K}$ is the same scaling matrix which, as mentioned previously, can be added to Equation 2.2. The only unknown in this equation is the vector of generalised constraint forces $\mathbf{g_c}$.

However, in general, there are many solutions for $\mathbf{g_c}$. One way to remove the ambiguity is to insist that the constraint force must lie in a direction in which the system may not move. It can be shown, using singular value decomposition, that this implies that the constraint force must lie within the *range* of the constraint Jacobian matrix $\mathbf{J_c}$ which in turn implies $\mathbf{g_c} = \boldsymbol{\lambda}\mathbf{J_c}$ for some vector $\boldsymbol{\lambda}$.

Equation 2.5 can then be rewritten

$$\mathbf{J_c K J_c^T} \lambda = -\mathbf{J_c K g_a} \tag{2.6}$$

allowing the vector of Lagrange multipliers $\boldsymbol{\lambda}$ to be solved for, which in turn gives $\mathbf{g_c}$. The disambiguation of $\mathbf{g_c}$ arises from the fact that by expressing it in terms of $\mathbf{J_c}$, a restriction has been imposed which limits the values available to $\boldsymbol{\lambda}$, and hence to $\mathbf{g_c}$.

$\mathbf{g_c}$ is then substituted into a simple equation of motion which is integrated to provide updates to the joint angles, whereby the updated angles respect the constraints imposed on the system.

Finally, as previously mentioned, Welman's more comprehensive approach to constraints in inverse kinematics assumes the constraints are met from the start, and simply need be maintained. Clearly, this may not be the case in practice, and even if all constraints were initially satisfied, they might drift and become violated due to numerical inaccuracies. This is compensated for, by, in effect, using a simple penalty-based spring feedback mechanism.

Before inverse kinematics can be performed a suitable representation for the articulated chain is needed. The paired coordinate method [ESHD05] does this by using

three local reference frames for each link. In comparison, Denavit-Hartenberg representation [ESHD05] offers a lightweight alternative commonly used in computer animation. It employs only one reference frame per link, with transformations between adjacent frames being fully defined by four quantities known as the link length, link twist, link offset, and joint angle. Each of these describes one simple link-to-link translation or rotation. Joints have one degree of freedom but can be overlapped to effectively produce more complex joints.

To consolidate the knowledge acquired from the literature, a Denavit-Hartenberg-based program was written forming the first of the implementations indicated in Figure 1.3 of Chapter 1. It employs the basic inverse kinematic approach described by Equation 2.1 to drive a robot manipulator. A demonstration video can be downloaded from `http://www.urbanmodellinggroup.co.uk/IK_Lit_Impl.mp4.zip` (link duplicated in Appendix E). The manipulator is seen to adjust the position and orientation of its end effector to match those of a target object (the traditional graphics teapot), whose position is repeatedly changing in a random fashion. Screenshots of the manipulator reaching for the target are shown in Figure 2.4.

Zhao and Badler [ZB94] addressed the calculation of joint angles required to hold a desired character pose in the $Jack^{TM}$ human figure simulation software. It was designed to improve on existing interactive animation systems seen to fail either in generality or in performance when the object being manipulated was highly articulated as applies to realistic human figures. The animator requested a character posture by specifying a set of one or more positioning constraints, upon which the system either computed the joint angles forming a pose which satisfies them, or, in the event of untenable user-demands, such angles as provide the optimal solution. Constraints could be of various types, dictating position and/or orientation of the end effector, or directing it to aim at

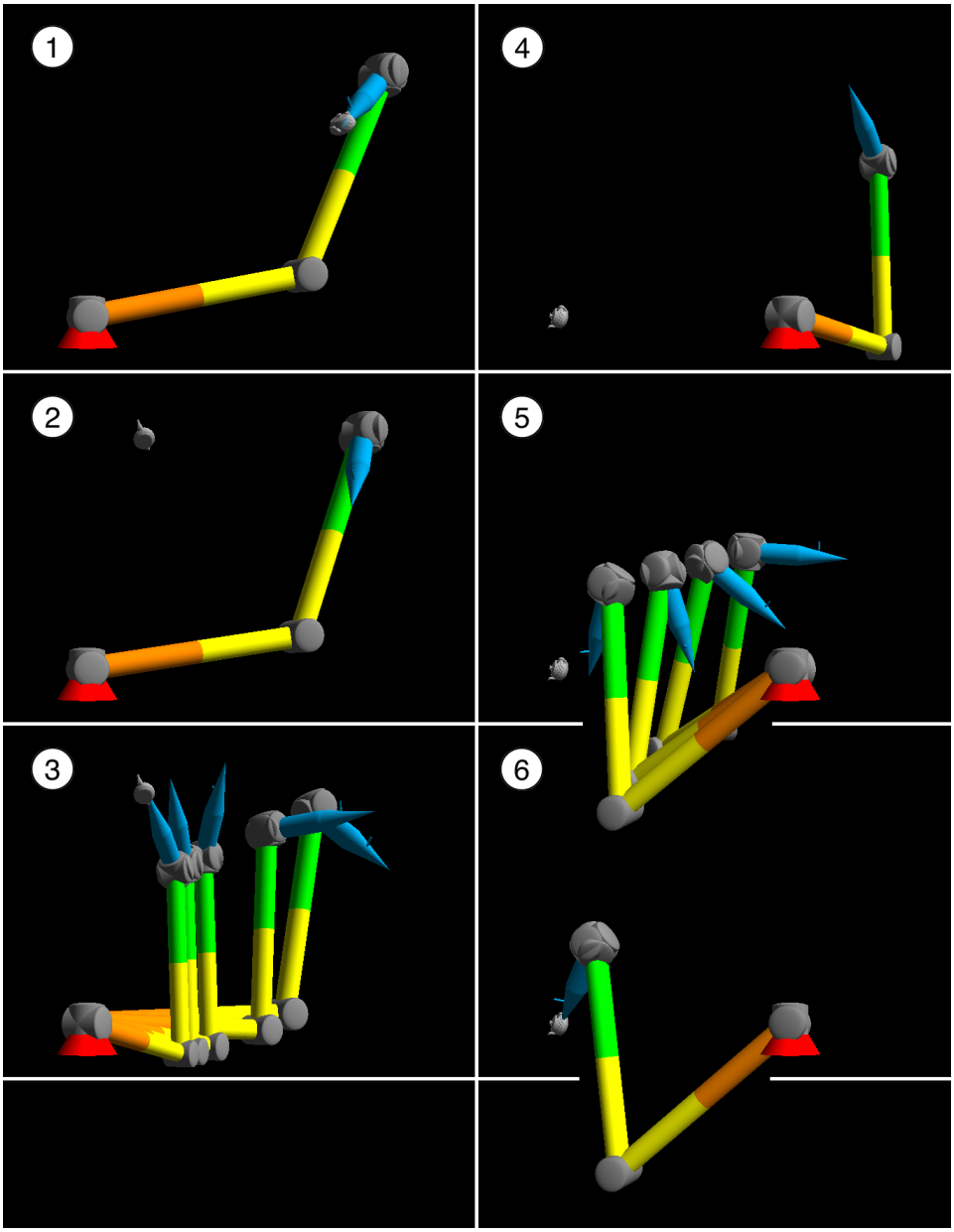

Figure 2.4: Screenshots from Denavit-Hartenberg inverse kinematics program written by thesis author based on [Wel93, ESHD05]. (1) End effector is located at target object (teapot) and matches its orientation. (2) Target randomly relocated and rotated. (3) Manipulator tracks and reaches target. (4, 5 and 6) Target randomly positioned again, and caught up by end effector, matching it in all 6-dimensions. Small protuberance on end effector shows its orientation roll.

Table 2.2: Optimisation, or mathematical programming, typically involves maximising or minimising the value of a function known as the 'objective function', by manipulation of the values of a collection of variables, given a set of constraints which restrict the values that these variables can take. Listed are further details for linear programming as used by Zhao and Badler [ZB94] and for other types found in past work discussed in later chapters.

Optimisation type	Description
Linear programming	Objective funct.: linear (no higher powers, roots, trig. functions etc.) Constraints: linear equalities and inequalities
Nonlinear programming	Objective f. or some constraints nonlinear (e.g. incl. trig. functions)
Dynamic programming	Problem recursively broken down into nested subproblems
Quadratic programming	Objective function: quadratic terms allowed Constraints: linear equalities and inequalities
Sequential quadratic programming	Solves nonlinear optimisation problem by solving quadratic programming subproblems at each iteration

a certain point, they could also require it be placed anywhere on a given line or plane, or within a specified half-space.

Unlike [Wel93] and [ESHD05] above, Zhao and Badler [ZB94] solved the inverse kinematics problem using nonlinear programming techniques. "Programming" as meant here refers to mathematical optimisation, a broad range of interdisciplinary methods of which varied subfields surface in a number of character animation papers including [WK88, CH07, AFO03]. Table 2.2 provides a brief overview of optimisation as encountered both here and later on in this thesis.

For a single constraint the nonlinear programming problem of [ZB94] can be formulated as

$$\begin{cases} \text{minimise } G(\boldsymbol{\theta}) \\ \text{subject to } \boldsymbol{a_i}^T\boldsymbol{\theta} = b_i \quad i = 1,\ 2,\ \ldots\ l \\ \qquad\qquad \boldsymbol{a_i}^T\boldsymbol{\theta} \leq b_i \quad i = l+1,\ l+2,\ \ldots\ l+k \end{cases} \tag{2.7}$$

where scalar function $G(\boldsymbol{\theta})$ gives the "error" or "distance" between the end effector's

actual position and/or orientation, and the goal required for constraint satisfaction. $\boldsymbol{\theta}$ is the column vector of n joint angles, θ^1, θ^2, ... θ^n in the chain under consideration from its root to the end effector, while $\boldsymbol{a_i}$ is a column vector with the same dimensions as $\boldsymbol{\theta}$. In addition to the k inequalities which allow for the specification of joint limits, l equalities are also included to allow linear relationships to be specified between the joint angles.

The solution to Problem 2.7 above is performed iteratively by a solver which requests (from another module) the evaluation, at each step, of both $G(\boldsymbol{\theta})$ and its gradient $\nabla_{\boldsymbol{\theta}} G$, the vector of partial derivatives with respect to each joint angle in the chain, where the gradient operator, $\nabla_{\boldsymbol{\theta}}$, is defined as

$$\nabla_{\boldsymbol{\theta}} = \left(\frac{\partial}{\partial \theta^1} + \frac{\partial}{\partial \theta^2} \; ... \; \frac{\partial}{\partial \theta^n} \right)^T \tag{2.8}$$

To describe how $G(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}} G$ are obtained it is expedient to consider the example of a simple constraint type in which a point $\boldsymbol{r}$ on the end effector is constrained to a point $\boldsymbol{p}$ in 3D space. The end effector vector $\boldsymbol{e}(\boldsymbol{\theta})$, which can potentially hold both position and orientation data, is, for this constraint type, chosen to comprise only the former whereby $\boldsymbol{e}(\boldsymbol{\theta}) = \boldsymbol{r}$. The end effector-to-goal distance can be expressed in terms of this end effector vector, and is defined as

$$P\left(\boldsymbol{e}(\boldsymbol{\theta})\right) = P(\boldsymbol{r}) = (\boldsymbol{p} - \boldsymbol{r})^2 \tag{2.9}$$

The end effector location $\boldsymbol{r}$ is easily obtained by forward kinematics and $G(\boldsymbol{\theta}) = P(\boldsymbol{r})$ then follows.

As mentioned above, the solver requires not only the objective function $G(\boldsymbol{\theta})$ at each iteration, but also its gradient, $\nabla_{\boldsymbol{\theta}} G$, which can be rewritten

$$\nabla_{\boldsymbol{\theta}} G = \left(\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} \right)^T \nabla_{\boldsymbol{r}} P(\boldsymbol{r}) \tag{2.10}$$

$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}}$ is the Jacobian matrix, populated by considering that for the i^{th} joint angle with rotation axis $\boldsymbol{u}$, $\frac{\partial \boldsymbol{r}}{\partial \theta^i} = \boldsymbol{u} \times (\boldsymbol{r} - \boldsymbol{r}_i)$ where $\boldsymbol{r}_i$ is any point on that axis (a derivation is given in [WB97]). The final element required to evaluate $\nabla_{\boldsymbol{\theta}} G$ is $\nabla_{\boldsymbol{r}} P(\boldsymbol{r})$, the gradient of Equation 2.9, thus

$$\nabla_{\boldsymbol{r}} P(\boldsymbol{r}) = 2(\boldsymbol{r} - \boldsymbol{p}) \tag{2.11}$$

Similar principles to those applied above for a simple position constraint were used in the $Jack^{TM}$ system for all constraint types, yielding, at each iteration, the objective function and its gradient with respect to each angle in the joint chain, as required by the solver to perform the optimisation.

While the system eventually met the optimal solution, its path thereto within the configuration space was not constrained. Furthermore, the high redundancy across multiple degrees of freedom allowed an infinite number of poses, and, for example, "in constraining the hand to the goal, the elbow might result in an undesired position" [ZB94], so that "an additional constraint for the elbow could be necessary for a satisfactory posture". The ability to process multiple constraints was thus essential, and achieved using an overall objective function comprising a weighted sum of the objective functions for each individual constraint.

Presented screenshots did look robotic, however, and sometimes showed "irregular joint angle distribution along the spinal joints" [ZB94], even when "grouping these joints together" which was said to produce more natural images. A more natural-looking output might have resulted by using biomechanical data as a basis for linear

relationships among the joint angles which the system was expressly capable of. However, the authors further stated "it is beyond the scope of this paper to find realistic spinal joint angle distributions".

The optimisation algorithm used to solve problem formulation 2.7, above, homed in on a *local* minimum only, instead of the globally optimum solution, which, however, was said to be commonplace in efficient nonlinear programming algorithms. Unwanted local minima where not of major concern, since "if they do occur during interactive manipulation, users can easily perturb the figure configuration slightly to get around the local minima" [ZB94].

Response times of between 2 and 15 seconds were reported for the mere creation of one single character posture, but this must be seen in light of the work's 1994 publication date and the system's usage which never was intended for real time games or simulations. The purpose, instead, was to allow interactive manipulation of a highly articulated skeleton via the definition of spatial constraints, thus indirectly specifying individual postures upon which the $Jack^{TM}$ software successfully computed the corresponding joint angles, while honouring skeletal joint limits.

2.5 Conclusion

Computer-based keyframing evolved from traditional animation, providing a degree of automation. The danger exists, however, of simply relying on technology, as "these systems will [yield good results but] also enable people to produce more bad computer animation" [Las87]. The skills of the user remain paramount, like those of John Lasseter, a classically trained animator, whose short film *Luxo Jr.* "sent shock waves through the entire industry to all corners of computer and traditional animation" [Cat98]. Entertaining characters require distinct believable personalities, whose very

thinking is conveyed by their actions [Las87, Las01], from whence the traditional view of the animator as an "actor with a pencil" [BD09].

Patience and application are surely just as important, however, and easier to neglect in the digital workflow, whose raison d'être is – in part at least – speed and efficiency. Traditional animation was the result of discussions and experimentation [TJ95, Joh02], and a time-consuming production process. Rough sketches were made, line work cleaned up and inbetweens drawn, inking or Xeroxing used to transfer drawings to cels [Ste79] (transparent sheets of cellulose nitrate, later cellulose acetate) which were subsequently coloured by painting, then stacked – with an elaborate background cel at the bottom – to build up, in layers, the final scene to be photographed [Joh02]. Such involvement and attention to detail are naturally reflected in the final creation.

The yardstick for superior animation seems indeed that it be instilled with life [TJ95], as defined by the word 'animation' itself. What the twelve principles and undeniable mastery of Disney animation create, however, is a *style*, which while aptly befitting its goal of entertainment, should not to be confused with a *universal* reference standard. Application of the principles does emulate nature but only to a limited degree [Las87], and even then in an exaggerated fashion – realism, clearly, was not the objective. In contrast, maximum realism within given constraints, may indeed at times be the goal of animation, as in architectural visualisations or virtual tourism. The digital medium caters for both types of animation, the realistic and the traditional. It does include new challenges, as in avoiding "that well-known 3D computer-look, now generally considered as bland and lifeless" [Hod09] and the *uncanny valley*, "the eerie feeling a viewer can experience when encountering almost-human robots" [Hod09, SN07] or avatars in virtual reality [SN07], extending even to "photorealistic human characters where the uncanny valley phenomenon is strongly perceptible" [BD09]. Despite presenting its own difficulties, however, computer animation has ample potential for artistic expression,

including hand-draw colour textures, the medium's "extraordinary variety of styles" [Hod09] and of course the very art of keyframing itself.

Inverse kinematics greatly simplifies the spatial positioning of keyframes, though it also finds use, quite apart from keyframing, in varied papers on character animation [BC89, Gle98, KSG02]. Underlying theory has been illustrated including work involving the $Jack^{TM}$ system, which saw further evolution, and is still in use today [BEL02, Sie]. The use of inverse kinematics in character animation can be seen in the 1989 film *Eurythmy* [AG90], which, while rudimentary by today's standards, does not betray the robotics origin [Wel93] of inverse kinematics.

3

Physics-based Methods

3.1 Introduction

Much of the motion found in nature is governed largely, sometimes entirely, by the laws of physics, and can thus be aptly described by them. From colossal heavenly bodies which follow predictable paths, to raindrops as they fall and land in a pool of water, physical modelling can mirror the natural movement of many a real-world happening. The same applies to the simulation of man-made objects like vehicles and aircraft, whose realism is limited only by the accuracy of the model, rather than any fundamental ill-suitedness of the physics-based approach.

In contrast, the activities of animals and people clearly depend on more than the physical laws, as both are affected by psychology and the extent of physical well-being [Tro02, MFCD99, Sch10b]. Instinctive behaviour and that learned from experience can also be instrumental, as when putting an arm out to prevent injury, in the reflex action of breaking a fall. The enthusiasm in earlier papers [BC89, HWBO95, WK88] for physics as a means to realistic-looking human motion appeared more reserved in later work, with the concept of 'naturalness' gaining importance, as expressed by Yan et al. [GMWC06] "dynamic simulation often produces motion that lacks important features of natural human motion" and by Liu and Popović [LP02] "to appear realistic,

a character motion needs to satisfy the laws of physics, and stay within the space of naturally occurring movements".

The move towards motion capture, now ever-present in character animation, further attests to the limitations of physics. Nevertheless, used within its actual scope, physics provides a powerful tool as a component of wider approaches to character motion synthesis, and remains in use in more recent work, as, for example, in the hybrid physics-based statistical method of Wei et al. [WMC11] (Chapter 4, Section 4.9). Potentially, physics-based methods have the additional merit of empowering characters with the ability to adapt to their environment, automatically changing their gait when carrying a load or when pushed by the wind, or adjusting their motion when tackling varying terrain geometry [MFCD99, HOB98, Pop00].

3.2 Controllers

In character animation the term 'physics-based' is somewhat of a misnomer, as the motion is not the result of physics alone but needs guiding by control software. This is required, as unlike passive systems like clothing or hair, virtual humans have an internal source of energy, and activate their muscles of their own simulated volition [HOB98], embodied in the controller. Controllers come in two main categories, adaptive, and non-adaptive [Nik94], the latter being employed by Bruderlin and Calvert [BC89] and Hodgins et al. [HWBO95]. Non-adaptive controllers incorporate state machines and are designed for a given behaviour, based on biomechanical or empirical data, but may necessitate a long period of interactive tuning. Their open-loop control strategy prevents them from being simply transferred to another character with a different morphology [MFCD99].

While adaptive controllers might simply adjust parameters which mirror those of their non-adaptive counterparts, thus increasing performance over time, other ap-

proaches include neural networks, fuzzy logic and evolutionary computing [Nik94]. The adaptive control process might merely be given the definition of an articulated figure, and a task to optimise, upon which it automatically optimises a set of controllers [MFCD99]. A disadvantage for character animation, however, is that the automatically learned motion might perform the specified task well, yet look far from human-like [HWBO95]. Furthermore, the number of degrees of freedom (DOF) needed for a plausible model of the human body was, at least in 1995, beyond the scope of automatic techniques, according to Hodgins et al. [HWBO95].

3.3 Forward and Inverse Dynamics

Physics-based approaches, unlike kinematics, account for the forces and torques – here jointly referred to as 'forces' for convenience – acting on the masses and moments of inertia tensor of the rigid bodies in the simulation [GMWC06, GMPO00]. Dynamic simulation is a physics-based method which comes in two variants. Forward dynamics updates the positions, orientations, and linear and angular velocities of the bodies, by first finding their accelerations, resulting from *known* forces applied to them [Sha01, Wel93, Ott03]. These may be either external to the skeleton, such as reaction forces preventing a foot from penetrating the ground, or internal, as applies to the constraint forces within the joints which maintain the interconnectedness of a character's skeletal structure. Solving for such forces as will maintain joint satisfaction, is a complex and expensive problem [MFCD99, Ott03] known as inverse dynamics, a variant of which is comprehensively explained in [BB88], with further work described in [ESHD05] and [Sha01]. More generally, given the trajectories of all the degrees of freedom, inverse dynamics finds the forces which create that motion by solving a system of algebraic equations [Sha01].

In character animation the rigid bodies are the skeletal links. While all such segments might be modelled, as in [HWBO95], due to its complexity, physical modelling might also be limited to part of the skeleton, or merely applied to a simplified model upon which the skeleton is later superimposed, as in the simplified leg-only model used in [BC89]. The dynamic simulation of multibody systems is often a two-part process [ESHD05, Sha01], as found in [BB88]. The constraint forces within the joints are first computed using inverse dynamics, whereby evaluation of any single force takes account of the entire interconnected structure. Having found the constraint forces, as well as the external forces acting on the skeleton, the second stage uses forward dynamics to apply them to the rigid bodies which are thereby repositioned accordingly.

3.4 Equations of Motion

Fundamental to updating the simulation state in forward dynamics, is the numerical integration of ordinary differential equations known as the laws of motion, of which one form is shown below [ESHD05, WB97].

$$\dot{\mathbf{P}}(t) = \mathbf{F}(t) \tag{3.1}$$

$$\dot{\mathbf{x}}(t) = \mathbf{v}(t) \tag{3.2}$$

$$\dot{\mathbf{L}}(t) = \boldsymbol{\tau}(t) \tag{3.3}$$

$$\dot{\mathbf{q}}(t) = \frac{1}{2}\boldsymbol{\omega}_{\boldsymbol{q}}(t)\mathbf{q}(t) \quad \text{where } \boldsymbol{\omega}_{\boldsymbol{q}} = \big[0, \boldsymbol{\omega}(t)\big] \tag{3.4}$$

In the above equations $\mathbf{F}(t)$ is the total force applied to the rigid body, which is identical to $\dot{\mathbf{P}}(t)$, the rate of change of linear momentum with respect to time, as stated by Newton's second law in its original form. Numerical integration over the interval of one time step yields the *change* in linear momentum occurring during that

interval, allowing the rigid body's momentum to be updated. Dividing by mass gives the linear velocity of the centre of mass, $\mathbf{v}(t)$, which upon further integration allows its position to be updated for the next displayed frame. Equations 3.3 and 3.4 relate to the rotational equivalent of Equations 3.1 and 3.2 with $\boldsymbol{\tau}(t)$, the total applied torque, equating to the time derivative of angular momentum $\dot{\mathbf{L}}(t)$, from which the angular momentum can be updated as above in the linear case. From this can be obtained the angular velocity, $\boldsymbol{\omega}(t)$, by pre-multiplication with the inverse inertia tensor in world coordinates, a quantity which, unlike mass in the linear case, needs to be re-computed at each time step. $\mathbf{q}(t)$, the orientation quaternion of the rigid body can then be updated, after integration of its time derivative.

A number of methods exist for performing the above numerical integration. Integration as used in simulation gives the area under some curve during a single time step. Euler integration, the simplest and cheapest method, approximates this area rather poorly with a rectangular shape [ESHD05]. Far greater accuracy is provided by the popular Runge-Kutta method of order 4 (RK4), which, being based on a higher-order Taylor series expansion, [CDH00, WB97], calculates the area of a shape which more exactly fits under the curve [ESHD05, Sha01], but at around four times the cost. As often demonstrated, a simulated pendulum using an Euler integrator is unstable, developing ever growing swings [KYT$^+$06] – an effect not observed in practice under RK4. However, such accuracy may not be needed, as, for example, in a driving game where user input automatically corrects for small inaccuracies in the vehicle's velocity or direction. Euler integration would allow time steps one quarter the size at the same cost, and while still less accurate than RK4, this does allow far greater spring forces to be used without the simulation becoming catastrophically unstable, explaining, perhaps, why it remains popular in the simulation community.

Constructing the equations of motion for complex objects can be difficult, in which case the Lagrangian formulation is preferred, [WK88], which expresses the motion of a mechanism in terms of its degrees of freedom, and results in fewer equations than the Newton-Euler equations described above [Ott03]. The Lagrange equations for a system with n degrees of freedom can be written

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_r}\right) - \frac{\partial L}{\partial q_r} = F_{q_r} \qquad r = 1, 2...n \tag{3.5}$$

where L is known as the Lagrangian, equal to the difference between kinetic and potential energy, q_r is the generalised coordinate used for degree of freedom r, and F_{q_r} is the generalised force in the direction of coordinate q_r, [BC89]. Substitution of object-specific expressions into Equation 3.5 yields the equations of motion actually used in the simulation. Commercially available software packages exist for the automatic generation of these equations, as employed by [HWBO95].

Further details are given in Section 3.6 which describes a sample of the existing past work in dynamic simulation.

3.5 Spacetime Optimisation

The above-mentioned simulation methods, whether forward dynamics only or preceded by inverse dynamics, perform all the calculations needed to refresh the simulation state by one time step, thus updating the rigid bodies' coordinates to create a single frame of animation. The spacetime constraint method, [WK88, LP02, Gle98], differs from the above approach in that the animation for the entire sequence of frames is computed simultaneously [MFCD99, GMPO00]. The process is one of optimisation, where an objective function reflecting one or more quantities such as energy use, is minimised over the entire animation sequence [GMWC06, RGBC96]. Various constraints may be

specified, such as a path to follow, a goal position to reach, or an obstacle to jump over with the aim being to minimise the objective function, while honouring hard constraints and *trying* to satisfy any soft constraints [LP02].

Spacetime constraints are expanded on in Section 3.7 giving a more in-depth treatment of some of the literature.

3.6 Dynamic Simulation

Excellent treatment of the basics of rigid-body dynamics in computer simulation is provided by the SIGGRAPH course notes of Witkin and Baraff [WB97], which include related topics such as differential equations, methods of integration, particle systems, constrained and unconstrained dynamics, and impulse modelling. The material is directly relevant to a number of character animation papers, and helps elucidate works such as [BB88] below. To illustrate its great utility, Figure 3.1 displays views from a physics-based car simulator whose implementation (by the author of this thesis) was based to a great extent on the work of Witkin and Baraff [WB97].[1]

A frequently referenced work in character animation papers [GMPO00, CHP89, MFCD99, Aba01] is that of Barzel and Barr [BB88], although it of itself does not mention such animation at all. It describes, in comprehensive detail, an approach to using constraints in physically-based modelling. It is the presence of physics [WK88, HWBO95, WMC11] as one of the many disciplines in character animation research, which gives relevance to their work. Presented is a technique for interconnecting simple rigid bodies using constraints to form joints, thereby creating multibody constructions. Constraints include the point-to-point form which acts as a ball joint between two rigid

[1] A video download link is omitted as the associated implementation *preceded*, and is hence external to, the work of this thesis.

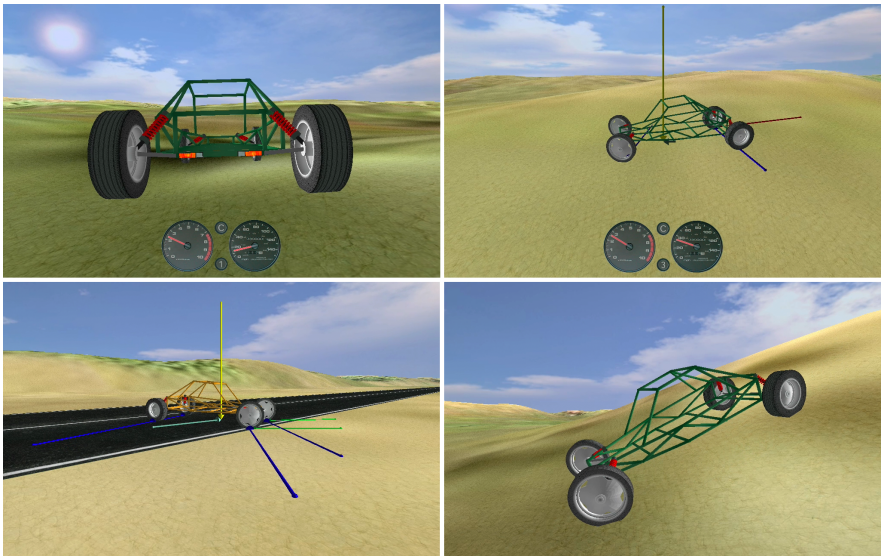

Figure 3.1: Physics-based car simulator implemented by thesis author prior to commencing character animation work. It illustrates the merit of Witkin and Baraff's extensive SIGGRAPH '97 course notes [WB97] which it was mostly based on, and which are equally valid for rigid body simulation within animated characters. Coloured arrows depict forces.

bodies, and the point-to-nail variety which impels a point on a body to a position in space, while leaving this body free to swivel about the location to which it is nailed.

As indicated in Chapter 1, the background research for this thesis included practical implementation work, the aim of which was to ensure, and demonstrate, a proper understanding of selected contributions taken from the existing literature. The dynamic constraints method of Barzel and Barr [BB88] was included in this, with a multibody construct – which can be thought of as simulated chain mail – using 88 joints and 64 rigid bodies being created and animated. A demonstration video can be downloaded from `http://www.urbanmodellinggroup.co.uk/BB88_Lit_Impl.mp4.zip` (link duplicated in Appendix E). It echoes Barzel and Barr's results showing a self-assembling structure which swings naturally once assembled. Screenshots are shown in Figure 3.2.

Due to the highly mathematical and physics-oriented nature of Barzel and Barr's

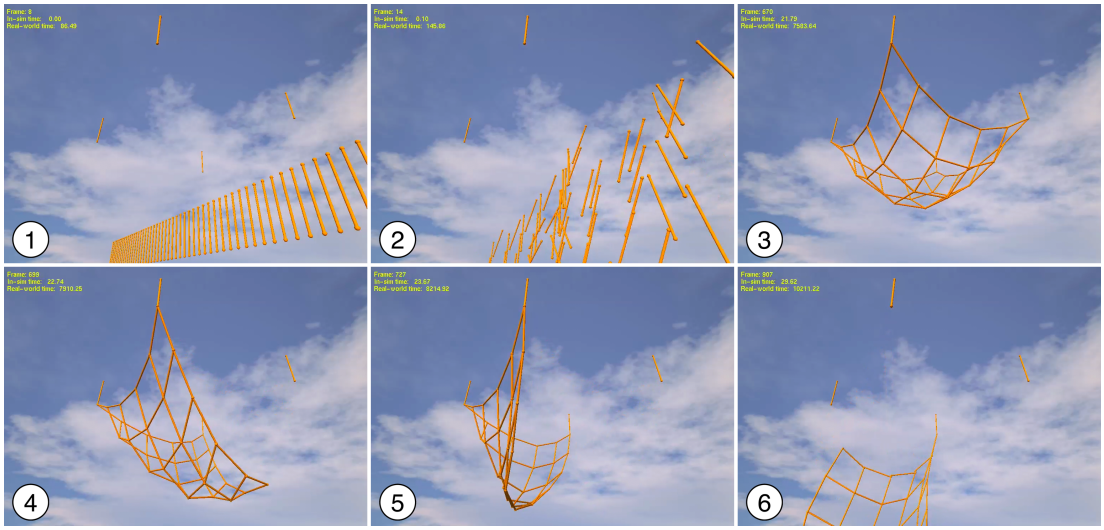

Figure 3.2: Implementation by thesis author of Barzel and Barr's dynamic constraint modelling system. The rigid bodies (1) fly together (2) and self-assemble into a structure (3) hanging from four point-to-nail constraints, while swinging naturally, as described in [BB88]. The realistic swinging and waves rippling within the chain mail-like structure continue, as it is gradually released (4,5) and falls away (6). The four static segments serve to highlight the point-to-nail constraint locations.

paper, this review has a similar style. Central to their approach, is the following second-order differential equation which dictates how the constraints become satisfied over a period of time, the plot of which is shown in Figure 3.3.

$$\frac{d^2D}{dt^2} + \frac{2}{\tau}\frac{dD}{dt} + \frac{1}{\tau^2}D = 0 \quad t \geq t_0 \tag{3.6}$$

where D is the constraint deviation and τ is an arbitrary time constant which controls the rate at which D decays with time (Figure 3.3).

The formula defines how the deviation of a *single* constraint approaches zero with time – whereby a constraint deviation of zero would indicate the constraint being fully met. At the start of the simulation the rigid bodies move and come together, joining into a multibody object as the constraint deviations fall to zero at a rate determined by

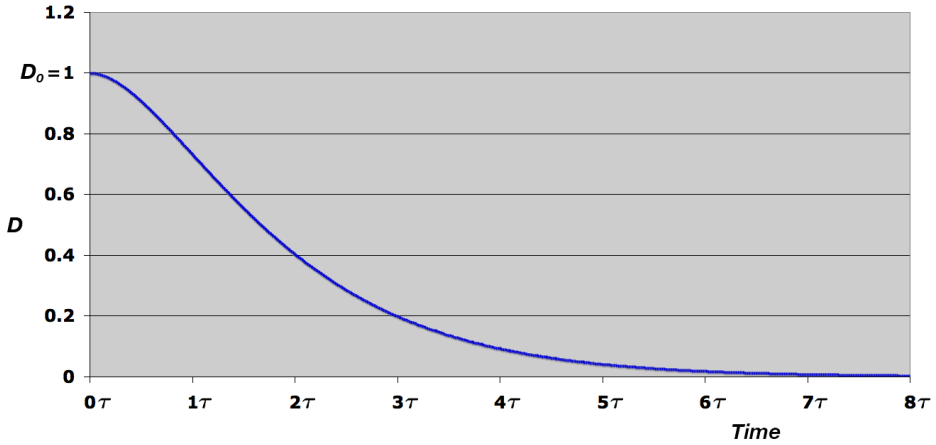

Figure 3.3: Constraint deviation D relative to its original value D_0 as it evolves over time based on Equation 3.6, central to the method of Barzel and Barr [BB88]. Zero deviation, corresponding to a fully met constraint, is approached asymptotically and closely approximated after the elapsing of such time as corresponds to eight time constants (8τ).

the time constant τ. Such self-assembly can be seen at the start of the above-mentioned video. The time constant τ controls the rate at which the components assemble and can be changed – within limits, as explained below.

The key to Barzel and Barr's approach is that the above formula, describing the desired deviation, can be rewritten in terms of both the unknown constraint forces and constraint-specific expressions for deviation, velocity and acceleration. The latter replace the generic terms D, $\frac{dD}{dt}$ and $\frac{d^2D}{dt^2}$ in Equation 3.6. (In the interest of brevity the term 'constraint force' is used here in a colloquial manner which is not strictly correct, as did Barzel and Barr, by their own admission, in much of their paper). The solution to this equation, calculated once for each frame of the animation, gives the time-varying constraint force which would cause the constraint type in question (such as point-to-point), to modulate its deviation so as to follow the plot in Figure 3.3, thus ensuring the constraint becomes met over time. At this stage what has been obtained is the solution to an inverse dynamics problem.

The equation set up in respect of a single constraint includes terms for *all* the constraint forces. Thus to solve for these forces, an entire system of simultaneous equations needs to be set up, comprising one equation for each constraint. Barzel and Barr collect the individual constraint equations into a multidimensional vector equation, which they solve by means of singular-value decomposition.

Having found the constraint forces, the final calculations for the frame are a simple application of Newtonian dynamics. The constraint forces are combined with the external forces and torques acting on the rigid bodies, and the simulation moved forward by one step in time using the equations of motion of the rigid bodies. This final stage is thus an application of forward dynamics.

All the above, and notably, the solving of the linear system, needs to be performed for each frame in the animation, as the dynamic constraint forces continually change to maintain the constraints. When using anything more than a small number of constraints and rigid bodies, the system of equations becomes large, and very expensive to solve. Barzel and Barr [BB88] themselves acknowledge that while their approach, building from modular constraints and rigid bodies, makes multibody objects easier to construct and animate, it does nothing to address the cost of physical simulation. This often prohibitive expense appears to be a serious drawback of their proposed approach, even outside the context of real time animation.

A further limitation, is that the constraints are only met over the course of time. Thus once the multibody objects have assembled themselves, they do remain so, but may partly disassemble if large external forces tend to make the component rigid bodies move apart. Thus gaps may appear, momentarily, in the model under the influence of external forces, which then close again and disappear in the ensuing animation frames, as the constraints become satisfied in accordance with the plot of Figure 3.3. With

careful observation, this can be seen in the above-mentioned video, merely indicating, however, a characteristic of the faithfully implemented method.

A solution might seem to be to make the time constant so small that any gaps close fast enough to be of no significance. However, this is not possible, as a very low time constant generates correspondingly high constraint forces, which if great enough, in programmer parlance, make the simulation "explode".

Although the time constant can only be modified within certain limits, it is, nevertheless, arbitrary. It is interesting to note that this also applies to the equation for the motion of a critically damped spring (Equation 3.7), where the undamped angular frequency can similarly be chosen arbitrarily. This makes Equation 3.7 for critical damping, and Equation 3.6 used by Barzel and Barr, identical.

The formula for critically damped motion [ESHD05] is

$$\frac{d^2x}{dt^2} + 2\omega_0 \frac{dx}{dt} + \omega_0^2 x = 0 \tag{3.7}$$

where ω_0 is the undamped angular frequency, and x the deviation of the spring from its resting position.

Thus, although Barzel and Barr make no mention of this, the constraint forces they calculate ensure the constraint gaps close as if controlled by critically damped springs. This would explain why the gaps in the joints close so smoothly and free of unwanted oscillations. It should be noted that any gaps in the constraints always close in time, even when considerable steady-state forces are exerted pulling two bodies apart, which is in contrast to penalty-based systems where a gap would remain. Furthermore, the behaviour of each component rigid body is calculated taking account of all other bodies as well as all constraints in the system. The resulting multibody structures thus exhibit considerable realism and fluidity. For example, in the above-mentioned video,

ripples are seen moving through the chain mail, and both its inertia and weight are clearly sensed by the viewer, confirming Barzel and Barr's claim that (in the context of inanimate objects) their method produces natural-looking motion.

More directly connected with character animation is Bruderlin and Calvert's [BC89] hybrid approach to animating human walking which allowed higher-level goals to be specified, such as speed and step length, yet employed dynamics to control most of the motion by means of applied forces and torques. The higher-level inputs, intended for convenient specification relieving the animator of tedious work, reduced the amount of detail needed to define a motion. The use of physics promised realistic motion – a partly superseded view which was prevalent at that time.

Upper-level inputs lead to motion generation by a hierarchical process. Rules were used to convert them into step constraints, the latter being specific quantities such as the angles of the legs or the durations of the various phases of a walk cycle. The rules used for this conversion included the empirically derived relationship between step length and step frequency, and knowledge of the anatomy of a locomotion cycle.

The middle stage of the hierarchy acted like a finite state machine, providing motion co-ordination and guiding the level below it.

The bottom of the hierarchy served to generate the angle values which drive the character's skeletal degrees of freedom. It calculated the forces and torques required to make a highly simplified model of the walking legs conform to the step constraints. More precisely, equations of motion were set up using the method of Lagrange, their use being made easier by constraining (simplifying) them in manners appropriate to the various subphases of the locomotion cycle's stance and swing phases. The equations were used in an iterative integration process which converged to an approximate solution, giving *those* forces and torques which allowed, for example, the swing leg to move forward in the exact time demanded by the step constraints, and heel-strike to occur with precisely

the desired hip angle. The stance leg of the simplified model was a variable-length telescopic structure, and having created a generic walking pattern for the dynamic walking model, kinematic algorithms were used to superimpose a proper leg over the telescopic one, to produce the remaining body angles, and to bestow human appeal on the motion by adding user-selected characteristics known as the *determinants of gait*.

There is no doubt the synthesised motion from this early work would easily run in real time on modern hardware, though it merely entails a straight-ahead walk on flat terrain, and its quality was described by the authors as no more than "quite realistic". Truly high-level input (eg "walk from here to there avoiding that obstacle") was not catered for, nor was mouse-based control to navigate the character in a virtual environment.

Any naturalness will arguably stem more from the anthropomorphic and empirical data used to control and constrain the physics, than from the realism imparted by the physics itself. Values such as leg angles and the times at which they are to be reached, were based on user-prescribed specifications and empirical data from human locomotion studies. The physics was *constrained* to comply with these *output* values, which the final motion will have evidenced as a characteristic gait, but this would be witnessed *however* those values were imparted to the motion, be it via physics or not. Thus, while physics was used, its full potential was not. Surely if physics is to create realism it should more freely *guide* the motion, determining the output based both on physical conditions and on the guidance of a controller, and should not, as in Bruderlin and Calvert [BC89] be forced by a controller to mould the synthesised motion so as to exhibit pre-determined output values.

Unlike Brüderlin and Calvert, above, Hodgins et al. [HWBO95] applied dynamic simulation combined with control algorithms to the entire skeleton in their animation of physically realistic models. The models comprised rigid links connected by joints

and performed the activities of running, cycling and vaulting. Their paper's main contribution is the creation of control algorithms to use with the equations of motion. Biomechanical data and observations of humans performing the respective tasks were used to tune the algorithms to achieve realism, and to satisfy more exacting requirements on the style of the motion than had applied to previous work.

State machines activated control laws according to the current phase of the activity that was being performed, such as the flight stage or heel contact while running. Furthermore, emphasis was placed on synergies which synchronise the motions of several degrees of freedom in a manner which reduces disturbances to the system.

Low-level control was based on proportional-derivative control laws. Thus, for example, to steer a bicycle while maintaining balance, a control algorithm computed the desired angle for the fork (and hence that of the handlebars) based not only on the deviation of the bicycle's roll and yaw angles from their desired values, but also on the roll *velocity* and that of the yaw. Such use of the derivative provides stability through damping.

As is often the case in animations, inverse kinematics was also employed, and, continuing with the cycling example, it computed the desired shoulder and elbow angles that would position the hands on the newly-oriented handlebars. In keeping with the dynamic nature of the simulation, however, the desired joint angles were not directly enforced, but indirectly via a simple muscle model in the form of joint torques. These were computed using proportional-derivative servos, based on the error between the desired and actual value of the joint angles in question. At each simulation step, the above-mentioned internal joint torques were combined with the external forces and torques acting on animated characters, before integrating the equations of motion of the system forward in time.

Algorithms were also created to provide group behaviour, as well as secondary motion which was included in the form of a spring-mass based cloth model.

A limitation of their approach is that it works best when simulating activities with sufficient dynamic content to significantly constrain the task, as the controllers for the remaining content can generate unnatural-looking motion unless carefully designed and tuned.

Runtime performance (in 1995) was only described for the cycling, and said to run ten times slower than real time on a Silicon Graphics Indigo Computer with an R4400 processor.

3.7 Spacetime Constraints

Witkin and Kass, [WK88], presented an animation technique which involved numerically solving the entire motion sequence en bloc as a constrained optimisation problem, in contrast to the more common approach of computing a sequence of solutions one frame at a time. The single solution, spanning the entire sequence, thus extends through time and not merely space, giving the method its name, *spacetime constraints*.

Constraints included particular poses at specified frames or the requirement to jump over a hurdle or perform a soft landing. The models employed objective functions which calculated energy expenditure or fuel use – quantities to be minimised by the optimisation, while honouring the constraints.

A distinctive feature of the spacetime approach is that unlike the conventional practice of using the equations of motion to relate quantities such as force and acceleration, and then obtaining position updates via definite integration, the equations are instead reformulated and treated as physics *constraints* which must be enforced. When applied to the above example, the spacetime method would thus treat force and position as independent quantities, contrary to usual practice.

The authors aimed to reduce the need for keyframing, so an animator need only specify a small number of frames, and realistic motion would be automatically computed by applying the same physical laws that dictate the motion of real objects. To a degree this would appear to have succeeded. For example, simulations of Luxo, an articulated lamp character familiar from earlier work [Pix86, Las87], jumping as well as ski-jumping, were computed and found to exhibit realism including the physics-related traditional animation features of squash-and-stretch and follow-through (as described in Chapter 2, Section 2.2) [Las01, Las87, TJ95]. The former allowed Luxo to push against the floor and take off, while the latter allowed momentum to be absorbed on landing. Furthermore, being able to modify motion by specifying high-level requirements, for example, a soft-landing constraint, or a style of motion defined by a suitable objective function, seems an important benefit of spacetime constraints.

The system is said to be no harder for an animator to use than keyframing would be. However, this should possibly be treated with some caution as the process appears to still require significant mathematical knowledge, despite the authors having enhanced user-friendliness by automating much of the process. It also cannot be overlooked, that the selected motions – jumps and ski-jumps – lend themselves particularly well to physical simulation, which cannot be said of all motion, such as human locomotion which is governed by more than just physics. Furthermore, the most complex model simulated was a simple desk lamp, with no real-world living example to compare it to. Simulating realistic human behaviour using spacetime constraints would be a very much harder task which the paper did not address.

Performance was very slow (in 1988), with the small jump computed in "under 10 minutes", and the simple ski jump in 45. While motion creation, and editing, may be possible at *interactive* speeds on current hardware, *real time* animation looks unlikely.

Liu and Popović [LP02] similarly chose spacetime constraints for motion synthesis, but based it on a highly simplified animation, provided by the user as a rough guidance to specify the required motion. Intended constraints on the character, such as footplants, were extracted algorithmically from the input animation. Physics constraints, in the form of realistic linear and angular momentum patterns, were enforced during synthesis. They conferred natural-looking motion while avoiding the complexities of comprehensive character-dynamics approaches, and allowing muscle force calculations to be dispensed with altogether. While synthesis was by means of the spacetime constraints optimisation method introduced by Witkin and Kass, [WK88], the constraints did not include the equations of motion themselves, but instead the simpler momentum constraints described above. This avoided non-convergence issues otherwise present in spacetime constraints optimisation, which might have prevented a solution being found.

The input animation was referred to by the authors as a rough "sketch", which, it should be clarified, was not any form of *drawn* sketch. An example of its simplicity was conveyed during the creation of a hopscotch animation, where the character in the sketch held a neutral arms-at-side upper body stance during all frames, yet the synthesised motion displayed arms held out in the realistic fashions appropriate to the hops, jumps and spins the game consists of.

Footplants and other environmental constraints were detected automatically by an elaborate method, which forms a major part of Liu and Popović's method. Considering just one of the character's rigid-body segments at a time, and the transformations associated with its frame-to-frame movement, linear algebra revealed, at each frame, an associated *stationary* point, line or plane, in world coordinate space. To establish the point(s) which have *remained* stationary during a span of time, the intersection of the above-mentioned points, lines and planes over successive frames was found. In those cases where it fell on the body, a constraint was defined which fixed that part

of the body to the intersection location. Sliding constraints, which fix a body point to a line or plane, were obtained by a different, optimisation-based method, to find both the point on the character and the line or plane to which it was bound.

Having detected constraints in the input motion, it was divided into constrained and unconstrained phases. The distinction was necessary as the physics and biomechanics rules for a character constrained to the ground are different from those applicable when in flight. These distinct phases were separated by *transition* poses, which could be specified by the animator, or suggested by the system's trained pose estimator. Additionally, the momentum constraints, described above, were generated by extracting parameter values from the sketch to customise generic momentum patterns obtained from the biomechanics literature. The system would attempt to preserve these parameter values during animation. As a final stage prior to animation the objective function was set up, with three components designed to favour firstly minimum mass displacement which leads to natural joint movements, secondly minimum DOF velocity for smoothness of motion across frames, and thirdly, balance when stationary. The unknowns solved for during optimisation included the joint angles at each frame. The result was obtained using sequential quadratic programming, ie it involved the solving of a quadratic programming subproblem at each iteration.

The work of Liu and Popović demonstrated handspring jumping, high-bar gymnastics, ice-skating jumps and more, and is striking in that it created motion which looked not only physically correct, but even natural, based on little more than a rough description of the desired motion and generic momentum transfer rules. For sufficient data to be present in the sketch a small set of keyframes inevitably had to be specified by the animator. Yet the authors succeeded, it seems, in their intention of making the creation of realistic motion easier for non-skilled users, while catering for the artistic

expression of skilled animators who could modify the system-generated transition poses and add keyframes as desired.

As with Witkin and Kass's original work on spacetime constraints, [WK88], a limitation is that the method works best when synthesising highly dynamic motion, as such motion is mostly the result of compliance to physical laws. Synthesis was reported as taking "less than five minutes", thus precluding real time motion creation, as does the time-consuming nature of the required user-input.

Spacetime constraints was also the choice of Rose et al. [RGBC96] who used the method for full-skeleton motion, as part of an approach which combined spacetime optimisation with inverse kinematics, maintaining some basic dynamic properties of the motion as well as kinematic constraints. Its use was limited to short transitions between concatenated motion clips, which lessened the impact of the dimensional explosion otherwise preventing the use of the spacetime method with complex articulated figures. A fast dynamics formulation further mitigated optimisation costs, as did reducing the number of degrees of freedom found in motion capture data down to 44 – three DOFs per joint as in the original data having been seen as excessive for many joints.

Briefly, transition creation was by three distinct processes. Interpolation techniques returned the root trajectory between the end of the first motion and the start of the second. An optimisation procedure, which solved the inverse kinematics problem over the whole transition, instead of just one frame, was used to constrain support limbs. These were defined as the kinematic chain from the support point, such as a foot on the floor, back up the kinematic tree to the root. In the presented examples this meant legs having feet held planted to the ground during appropriate parts of the transition interval. The motion of all those limbs which do not support the body was determined using spacetime constraints. All motion curves were defined as B-splines and spacetime optimisation was performed over the values of the knots, aiming to

obtain a minimum energy solution by minimising joint torques while also maintaining joint angle constraints.

Methods which effectuate transitions by blending a simple linear combination of the two motions to be joined [BW95, UAT95, WP95] can result in a lack of realistic dynamic qualities, and sometimes also fail to meet kinematic or anthropomorphic constraints [RGBC96]. In contrast, the method of Rose et al. was said to provide seamless and invisible transitions satisfying both dynamic and kinematic constraints, though "quite realistic" and "quite good" [RGBC96] were more sober qualifiers also used. A demonstration video, had one been provided, would have allowed a more precise assessment and been especially useful as dynamics processing was only performed on a DOF subset, excluding the all-important support legs, thus limiting physical realism with a result which is difficult to imagine. Furthermore, optimisation is expensive and synthesis times were correspondingly lengthy, with 72 seconds reported for the creation of a 0.6 second full-skeleton transition, and 20 seconds for a mere 0.3 second hand gesture, transitioning from arm-at-side to military salute (or vice versa), and involving the DOFs of only one arm. An additional restriction on transition sequence length was confirmed by the authors stating that *successful* transitions are quite short (0.3 to 0.6 seconds, as above) and that "without a biomechanical model to guide a large motion, our minimal energy model will often prove insufficient". Nevertheless, the motion authoring system of Rose et al., was able, semi-automatically, to seamlessly join motion segments with "dynamically plausible" [RGBC96] transitions, and was a notable contribution to the utility of the method of spacetime constraints. Additional contributions included a proposed cyclification method (as defined in Section 1.4.2, Chapter 1) and a functional expression language with interactive interpreter, for representing and manipulating motions.

Further use of the space-time formulation is found in the physics-based statistical method of Wei et al. [WMC11], described in Section 4.9 of the following chapter.

3.8 Conclusion

This chapter discussed the field of physics as used in simulation, emphasising its ability to model many real-life processes but also its more limited scope in character animation. Dynamic simulation and the need for controllers, as well as spacetime constraints, were introduced prior to an in-depth description of selected work in those areas.

Physical simulation alone cannot synthesise human motion, and controllers are needed to guide it. This is analogous to a driving game where vehicle physics of itself is not sufficient, and the car must be guided by the driver, be it the human player or the artificial intelligence of a software module. Too much emphasis on control, however, can override the merits of physics, as exemplified by a car turning in accordance with the driver's control input, yet doing so *incorrectly*, by ignoring slippery virtual road conditions for which physics would dictate a skid along a straight-ahead course.

As long as control is not stifling – a possible shortcoming in the method of [BC89] – the ingredient added by physics to character animation is realism [WK88, LP02], albeit one *limited* to those aspects of human motion which, unlike those of psychological origin, can be modelled by physical laws. Synthesising natural walking motion is more difficult than applies to activities of a more ballistic nature, such as vaulting or skating. This limitation, as well as the cost of physics-based methods, is surely a factor in the widespread use of motion capture seen today. While enthusiasm for physics in character animation appears more measured [WMC11, GMWC06] than in the 80's and 90's [BC89, HWBO95, WK88], its deserved place in the animator's toolbox is nevertheless indisputable.

4

Working with Motion Capture

4.1 Introduction

An intractable problem for the methods covered so far, is that they lack the subtle detail expected by human viewers, viewers well practised in perceptual judgement [MCC09, Tro02, HOB98]. Only a *degree* of naturalness can be imparted by biomechanical data, empirical knowledge of human locomotion, and by the careful observations of people performing specific tasks, as used in [HWBO95, BC89, LP02]. Furthermore, although manual artwork and computerised keyframing show skilled animators as impressive indeed, their work does remain an interpretation [Tra94], and is even intended as such [CBC+97]. What the above methods lack is the objective expression of all the delicate nuances found in real human motion. A solution is afforded by motion capture, which by recording human movements for later playback, provides, within the limits of the technological accuracy, motion we interpret as anthropomorphically correct.

Extracting data from actual human motion is not new. In the late 1800s photography was used as an intermediary to analyse human movement for medical and military purposes [Tra94]. The invention by the Fleischer brothers of the rotoscope [CBC+97], patented in 1917 [Bec03], allowed animation to be traced over film footage of live actors playing out the scenes [Stu94]. The early 1960s saw Lee Harrison III's precursor

of the modern exoskeleton, converting body movements to control voltages by means of potentiometers built into a harness worn by an actor [Stu98a, Stu98b], though it was only in the 1980s that motion capture methods started to become markedly more widespread [Stu94]. These included an extension of rotoscoping whereby actors were filmed from multiple viewpoints with markers caught on film, to be converted to 3D coordinates by a manual process, although the late 1980s saw algorithmic processing of captured marker data akin to methods seen today [Tra94].

The simplest use of motion capture is straightforward playback of a recorded sequence on a synthetic character. Applications include miscellaneous endeavours like perceptual research and demonstration purposes [MLD+08, Sim12], as well as the motion picture industry (which also employs motion *editing* and *synthesis*). Motion capture in films provides not only convincing-looking foreground characters, but can also animate backgrounds with crowds of people [Sco03, GHS+02], by reusing the same motions on multiple characters. Furthermore, it allows the depiction of scenes which would be too impractical or dangerous for real actors, as used to simulate the unfortunate souls falling from ship in the film *Titanic* [GHS+02].

While simple playback is by definition constrained to pre-recorded motion, motion capture-based performance animation, or computer puppetry, by contrast, sees a live person animating a synthetic character in real time [Stu98a, Stu98b, Tra94]. The avatar becomes an interface between actor and viewer, offering a potentially interactive [SLSG01, Stu94, Tra94, WH97] experience, as sometimes seen at trade shows or press conferences. Furthermore, real time display of captured motion has been successful in television [SLSG01, Stu94, Stu98b, Tra94], allowing live broadcast, within a scene which is real, of a character who is not.

Times do exist, of course, demanding avatar-control without the complexity, and encumbrance, of digital puppetry. Video games provide a case in point, as aside from

showing extended sequences in cinematic fashion to good effect, they mostly require real time synthesis as guided either by the player, or, for autonomous characters, by program control. This requirement is not the sole remit of games either, since, as detailed in Chapter 1, many virtual environments exist, demanding the real time *creation* of new synthetic motion. Methods have thus been developed for the algorithmic modification of motion capture data, allowing the synthesis of motion which exceeds the bounds of the recorded actor's movements. Four general approaches have been developed for this, concatenation of short clips to be played in succession, blending between recorded motions, statistical approaches and editing of the motion captured data.

The concatenation approach, as in [KGP02, LCL06, AFO03], involves selecting those frame sequences from a database which, when played in succession, would best fit the requirements of the motion being created. It is similar to copying and pasting text in a document, in that the rearranged clips, like the text segments, remain unaltered bar transitions created at the sequence boundaries to ensure a smooth join. As such concatenation-based methods involve reorganisation as well as splicing, the terms 'frame rearrangement' or 'sequence rearrangement' are equally valid names. Blending, the second approach, is another term for interpolation, and collects a contribution from each of two or more motions, to create another, which, depending of the employed weightings, is generally different from any of the input motions. While frequently used in a peripheral manner, it is of central importance in interpolation synthesis as in [WH97, RCB98, MLD10], and in one of the four methods presented in [BW95]. In the third category, statistical methods learn from motion capture data and express it in compact mathematical form, from which can be synthesised new motion which has, however, a similar style and idiosyncrasies to that of the original motion data [LWS02, CH07, MCC09]. Finally, the editing of motion capture data, as in [WP95, BW95, UAT95], and as applies to a stage in the method of [LCR$^+$02], modifies a

single sequence to create a new one, while attempting to retain, as much as possible, the natural look inherent in the original. Past work, employing motion capture data within these four categories is covered in Sections 4.6 (synthesis by concatenation), 4.7 (motion interpolation), 4.9 (statistical methods) and 4.8 (motion editing).

4.2 Motion Capture Technologies

Motion capture sensing technologies exist in a number of variants, the most common being mechanical, acoustic, magnetic, optical and, more recently, inertial. An overview is given below, inspired by sources including [SLBR04, Tra94, Stu94, WF02, Fur99, Ger04, WH97, Sch10a], though it should be noted that hybrid technologies exist also, to overcome the limitations of any one approach.

Mechanical. As seen above, the exoskeleton, or external skeleton, stems back not only to the early days of motion capture, but to those of computerised animation itself [Stu98a]. Worn by an actor, it includes a collection of electromechanical transducers such as potentiometers or shaft encoders, to convert angular position to an analogue or digital signal, allowing recording of the current pose. While the device has to be physically worn, which inhibits performance, it does dispense with the occlusion problems of optical systems (described below) and so can be used in confined spaces, and this even when driving a car, as it does not suffer from magnetic (or optical) interference (below). Measurements are relative to the actor, however, so there is no awareness of absolute position, and hence none of the ground either, which precludes jumping and tends to generate footskate.

Acoustic. Acoustic systems use sound emitters fitted to the body, while the locale is equipped with monitoring receptors. However, the transmitters emit sound pulses sequentially, so a full sequence describing the performer's position will not

correspond to an instant in time, generating a slightly skewed representation. Occlusion problems exist to a lesser extent than with optical trackers as sound can pass through and around obstacles with relative ease. However, spurious reflections, external noise, wind (outdoors) and sensitivity to air temperature and humidity can be problematic. Furthermore, a limiting factor is the speed of sound itself, which, due to the sequential nature of the transmissions, limits the size of the capture area and the number of transmitters worn by the actor.

Magnetic. Magnetic systems use a single transmitter, with the receivers being worn by the actor. Orientation can be tracked using three orthogonally placed magnetic sensors in a single sensor unit. The non-existence of occlusion problems increases accuracy and ease of use, but nearby ferromagnetic objects as in iron or steel-reinforced concrete floors, and the presence of conductive material can distort the results, by affecting magnetic field shape. High processing speed can provide real-time feedback for actors and allows immediate broadcast. Compared to optical systems equipment cost is lower, but so is the sampling rate and data can be noisier. As with acoustic systems, cables are needed which hinder the actor, although the size of the user-worn component can be quite small.

Optical. Cabling is not required for optical systems, thus leaving the performer unhindered. Reflective marker spheres are worn and sensed by multiple cameras which emit infra-red light. The use of light-emitting diodes (LEDs) worn by the actor is an alternative method. The primary disadvantage of all optical systems is that there must be a clear line of sight between markers and the cameras acting as sensors. Occlusions, where markers are no longer seen by a sufficient number of cameras, are a common problem addressed by approximating the missing data during post-processing. Furthermore, confusion arises from the difficulty in dif-

ferentiating between tracking markers which have moved in close proximity with each other. Advantages are high sample rate, relatively clean data, the possibility of using a large number of reflectors and a large capture volume. Additional drawbacks include light interference and high equipment cost. Furthermore time-consuming post-processing is required of the captured data, although real time feedback has recently become possible [Sha10, Mota, Vic].

Inertial. Inertial sensing, commonplace in ships and aircraft in the 1950s, is the most recent capture technology used in computer graphics. Originally equipped with heavy gyroscopes on a gimballed platform equipped with accelerometers, position could be calculated by integration of the measured accelerations akin to the methods of Section 3.4. Gimbals became redundant with the advent of strap-down (fixed) inertial navigation systems, and microelectronic mechanical systems (MEMS), a technology which integrates mechanical and electronic elements between 1 to 100 micrometres in size on a silicon substrate, made possible small and lightweight sensors in the 1990s. With gyroscopes and accelerometers now available in chip form, inertial sensors can be worn for motion capture. They benefit from relatively high sample rates, an absence of occlusion issues, and do not suffer interference from light, unwanted sound or magnetic field distortions. The drawback of inertial trackers is drift, as tiny inaccuracies in the accelerometers or gyroscope orientations can lead to large accumulated errors in the calculated position. This major drawback is overcome, however, by using inertial capture in conjunction with other sensing technology [VAV+07].

4.3 Motion Capture File Structure

Motion capture files come in many formats, the most popular being BVH from Bio-Vision (now defunct) and Acclaim's ASF/AMC format [Lan98, Mad01]. Due to its straightforward structure, the Biovision Hierarchical Data or BVH file format was preferred for the practical work of Chapters 5, 6 and 7. BVH files comprise two distinct parts. The first details the skeletal hierarchy, describing topology and joint-to-joint offsets, thus all the information needed to construct the skeleton, with appropriate connectivity and bone lengths, and a single initial pose. The second part of the file describes the motion, giving the frame rate, and then, for each frame in the recorded sequence, the channel data for each degree of freedom, thus the x, y and z joint angle rotations expressed in terms of the parent frame, as well as the world-coordinate translation and rotation data for the skeleton's root node (usually the pelvis). The hierarchy section of the BVH file clarifies how to interpret the motion data, by specifying which x, y and z rotations and translations the channel data values are referring to [Lan98, Mad01]. Acclaim's ASF/AMC format is more complex, also comprising two parts, albeit split into separate files. The ASF (Acclaim Skeleton File) file describes the skeleton, with the motion data held in the AMC (Acclaim Motion Capture) file [Ger04, Lan98].

4.4 Data Acquisition

motion capture data can be sourced free of charge online, the Carnegie Melon University database [CMU] being a popular example, another being the Ohio State University Advanced Computing Centre for the Arts and Design [TOSU]. Nevertheless, when specific data is sought and recording oneself is not an option, purchasing becomes necessary as applied to the practical work of Chapter 6 and 7. Video recordings

of desired motions were sent to MotionCapture3D [Motc], a motion capture bureau offering an at-a-distance service for the creation of custom motion captured sequences[1]. Recordings are made using an actor who emulates the specified movements.

Creating motion capture files to match specific requirements is a time-consuming and demanding process. Quite apart from the obvious difficulty of properly acting out the desired motions in a confined capture volume, post-processing can be a drawn-out process. A brief glimpse of the task is hereby given, based on equipment found at the University of East Anglia. The system is an optical one, made by Motion Analysis Corporation [Motb], using cameras equipped with infrared emitters. A session starts with equipment calibration, after which the marker-clad actor is recorded, using, for example, Motion Analysis's own EVaRT software (now-superseded). Any errors due to marker occlusion will then need correcting in software, merely yielding, at this point, the coordinates of the markers for each frame of motion, thus, as yet, no angular data for the skeletal joints. Software such as Autodesk MotionBuilder [Aut] then uses these marker coordinates to animate a skeleton whose dimensions are those desired for the final animation. More specifically, *virtual* markers are placed on the skeleton in a manner emulating marker placement on the real actor, and, by using the recorded data stream, are then continuously displaced in a manner which echoes the movement of the real markers on the actor. Due to its linkage with the virtual markers within MotionBuilder, the skeleton is animated to mirror the real actor. Only at this point are the skeletal poses known, allowing the joint angles to be extracted and a motion file to be saved in BVH [Mad01, Lan98, Sch10a] or other choice format.

[1]Input data shortcomings stated in Chapter 6, Section 6.8.1, resulted from changing requirements inherent in the organic nature of research project development, and do not reflect on the company itself.

4.5 Mapping Motion Capture Data To Skeletal Motion

As described above, during optical motion capture, an actor is fitted with reflective markers which are tracked by specialised cameras, whose output is processed to generate Cartesian 3D marker coordinates. These coordinates do not correspond to the joint positions of a character's underlying skeleton, however, and a method is needed to convert this raw data to values describing a sequence of poses for whichever skeleton is to be used to drive the character. From this sequence can then be extracted the joint angle and root trajectories typically used as input when working with motion capture data. This issue, of mapping optical marker data coordinates to skeletal poses, (as performed internally in MotionBuilder), was addressed by Zordan and van der Horst [ZVDH03], by using a forward-dynamics physical model as an intermediary between raw data and skeleton.

The linkage between the two was performed by interactively attaching virtual markers adjacent to the skeleton, at positions which emulated those of the actual markers relative to the human actor. Virtual damped springs were then attached between the virtual markers, and the positions of the real markers specified in the raw data, thus applying external forces to the skeleton. Furthermore the skeletal joints were provided with internal resistive damped torques designed to give the skeleton a tendency to adopt a neutral arms-at-side standing pose.

Considering each frame of motion capture data in succession, the external spring and damping forces were allowed to act on the skeleton, counteracting the rigidity imposed on the skeleton by the internal torques, like a springy toy being bent and twisted from its preferred state to another shape. The damping allowed the skeleton to quickly reach a steady equilibrium posture, at which point the pose was recorded and the virtual external springs' forces were updated as governed by the next frame of

raw data. Thus, for every frame of motion, the skeleton was positioned in a manner analogous to a coach or teacher adjusting the stance of a student player or dancer, by applying a guiding force until the correct stance is achieved.

The system could be extended to provide additional forces which maintain constraints such as footplants and hand holds. Friction forces prevented feet from sliding while ground penetration was counteracted with reaction forces. Constraint satisfaction did not extend to the explicit honouring of joint limits however. Instead, the true human motion underlying the raw data which guided the skeleton, ensured avoidance of limit violations.

An advantage of their system was that it did not exhibit the side-effects of inverse-kinematic based methods, such as knees and elbows that never fully extend. It also dispensed with the need to give some markers a higher priority than others, such as those associated with the pelvis. The rate of 2–3 frames per second in 2003, suggests their method could run in real time on current hardware, and thus be used for performance animation [Stu98b, Tra94]. Digital puppetry, however, is a specialist application, and frame rate is not critical for mapping done offline.

Outside the scope of Zordan and van der Horst's work [ZVDH03] were the problem of retargeting [SLSG01, Gle98] to skeletons of different size or proportions (Section 4.10), and that of skeleton estimation. They simply used an average skeleton scaled to be the same height as the motion capture actor.

Although the system is said to *map* 3D marker position data to joint trajectories, the trajectories were actually obtained without establishing any explicit mapping, and this using a satisfyingly simple concept. However, it must be realised that joint angle generation is a subjective affair; there is no "correct" pose in an absolute sense for any given raw data. Thus while the generated skeletal motion was shown to be smooth, and was said to reliably produce crisp footplants and "reasonable joint angles", other

methods using alternative approaches to link raw data to skeleton, might generate different motion which could be deemed equally satisfactory.

It is after the mapping of marker data to DOF values, and subsequent storage to file, that various methods of motion capture-based synthesis can be performed, as are now described below.

4.6 Pose Rearrangement

4.6.1 Graph-based Pose Rearrangement

Kovar et al. [KGP02] introduced motion graphs, a method for representing the connections within a database of motion captured data, and of extracting motion sequences therefrom able to follow a user-specified path, as well as honouring any requirements stipulating the *type* of motion to be created.

The graph is created algorithmically, and has directed edges representing either original motion clips (or sections thereof), or automatically created transitions. Nodes merely represent the joining of edges. Frames can also be labelled – for example to indicate a walking or a sneaking motion – allowing such motion types to be requested when specifying the motion to be created.

Continuity between each pair of frames in the motion dataset was tested using an error metric based on the sum of square distances between the points in a downsampled version of the mesh attached to the skeleton (Chapter 1, Section 1.3). The metric was based not merely on the two frames in question, but on two *windows* of frames, one from each motion, allowing it to take account of continuity not only of position, but also of velocity and acceleration. By allocating each of the motions to an axis, error values for every frame pair could be plotted in a two-dimensional representation, whose local minima, if below a user-specified threshold, defined where transitions should occur between the two motions. The interpolation used to form them, could, however, lead to

footskate and other constraint violations. These were corrected in post-processing with reference to boolean constraint annotations added to the input motions, and carried over to the new motion while blending. Having built the transitions, the graph was pruned, which included removing dead end nodes and sinks (places from where it is only possible to travel to a small number of graph nodes).

Given a completed graph, motion creation at runtime involves graph-search techniques to select edges that satisfy user requirements and thereby build a graph *walk*, which corresponds to placing motion sequences selected from the database one after another. Search efficiency was increased using branch and bound. This allowed large subsets of the potential candidate solutions to be deemed fruitless and be discarded en masse, thus accelerating the search process. To find the suitability of a proposed walk, the error associated with appending each new edge was calculated. The choice of error function affects which path is selected in the graph, and it was chosen to correspond to the specific requirement for the motion being synthesised. Local search methods were used to generate a graph walk incrementally with the aim of producing a complete and satisfactory walk that minimises the error within a reasonable amount of search time, and not to find the globally optimal solution.

Graph search duration (in 2002) was no more than the playback times for the synthesised motion, but this should not be confused with real time animation. Kovar et al. [KGP02] gave no indication that the incremental nature of the graph search technique allowed playback to commence before the search process was complete. Another limitation was that the generated motion was limited to that found in the database, unlike the continuous motion spectrum which blending-based methods can create.

Arikan and Forsyth [AF02] also used directed graphs, and created motion which satisfied various constraints, by using a randomised search of a hierarchy of such graphs,

to select a string of motion clips (sequence fragments) from a database of input motion sequences.

For every graph in the hierarchy, the nodes represented the input motions. The edges, however, differed from level to level. At the lowest and most detailed one, edges were used to connect (with some exceptions) all pairs of frames for which a join with sufficient continuity – low enough cost – could be achieved. When creating higher level graphs the number of edges was reduced using k-means clustering, such that each graph summarised the one below it. At the top (coarsest) level each edge was the root of a binary tree, whose leaf nodes were the edges of the lowest-level graph.

Synthesising a motion involved searching for a sequence of clips in the database which fully satisfied hard constraints, and optimised both soft constraints and the continuity of motion. Efficiency was afforded by the hierarchical structure, whose higher coarser level could be searched quickly for approximate paths, with the better of these being refined by further searching down to lower levels. More specifically, initial seed paths were randomly selected for the top level and then mutated to see whether a better match could be found. One of the mutation types involved deleting edges in the path and replacing then with child edges in the underlying graph, allowing the search to proceed down the binary trees with successive iterations, and thus from the coarser representation of the motion database to its most detailed one. The search process avoided getting stuck on locally optimal paths by adding new random seed paths with each iteration, allowing substantially different portions of the graph to be searched. This strategy does not search the entire graph, and instead aims to find paths which are *close* to the global optimum.

The approach allowed motion authoring at interactive speeds. Although the graph hierarchy typically took five hours to construct, searching it for an acceptable 300 frame motion was typically performed in three to ten seconds (results published 2002).

(Jehee) Lee et al. [LCR+02] addressed the issue of real time animation sourced from a large database of behaviours, but with an emphasis on user-interface methods. Transitions added flexibility to the database where frame-to-frame continuity allowed, and efficient searching of the resulting graph was obtained by clustering. Three user input methods were adopted: selection from a set of displayed choices, path sketching and enacting of the desired motion in front of a video camera.

The data was stored as a two-level structure, whose lower layer was a graph formed by identifying every pair of frames with poses and joint angle velocities of sufficient compatibility to sustain a transition. Additionally, frame-pairs required identical contact states (determined automatically in preprocessing). For example, if one frame had a pose with the left foot about to leave the ground, the other frame had to do the same in order to match that contact state and allow a transition. This requirement is one of the ways the graph was pruned, two more being the elimination of similar transitions by selecting the best among them, and the avoidance of dead ends, by eliminating edges not fully contained in the graph's single largest connected component. Small discontinuities were smoothed using displacement mapping, as described in [WP95, BW95], and footskate was, where possible, corrected with a rule-based method which selected the foot trajectory from either one of the motions being blended, instead of blending the two.

The higher layer was a statistical model giving a generalised depiction of the input motion, obtained by clustering together similar motion frames, while retaining the same transitions used at the lower level. For each frame of input motion there existed a tree of clusters, rooted at the cluster to which the input frame belonged, and describing, up to a certain depth from that frame, the connectivity of the higher-level representation of the database. Each cluster in the tree consolidated similar motion states, and thus a type of action, so the tree itself indicated which actions could be performed when

starting from the frame to which the tree belonged. Associating a tree to each frame in the database resulted in a cluster forest – the high-level representation of the entire database. The compactness of this representation, and the allocation of only a small part of it to each motion frame, allowed efficient searching at runtime for behaviours which were available from the current frame and which satisfied user specifications.

Searching was not limited to the upper level, however – the usefulness of clustering depended on circumstances. It was found to be unnecessary for the sketch-based interfaces. These were used for navigating a maze as well as a rough terrain environment, and the motion best matching the sketch was found by merely searching the lower-layer graph. The choice-based interface was used to select the action to be performed in a virtual playground, and cluster trees provided a means to obtain a small set of typically well-dispersed actions for display in the interface. The vision-based interface allowed interaction with a virtual step stool, and clustering was required to efficiently match a silhouette of the user performing various actions to possible avatar actions, the cost saving resulting from having to consider only tens of paths in a cluster tree instead of millions in the lower layer graph.

The two-layer structure was thus able to offer rapid searching via clustering. Performance was called "real time", although merely 15fps, and the video based motion synthesis had an inherent lag of three seconds.

The lack of any need to manually process the database to any substantial extent was sold by the authors as a benefit of their method for the computer games industry. However, a considerable amount of time and money is invested in commercial game production, of which database creation is just one facet. Developers may thus prefer to ensure a high quality database by continuing to manually process the short, painstakingly planned and possibly annotated motion clips which the method of Lee et al. [LCR+02] is able to avoid.

(Kang Hoon) Lee et al. [LCL06] re-used recorded motion data in an extended virtual environment, but compared to (Jehee) Lee et al. [LCR+02] before him, achieved greater efficiency by storing motions in a number of smaller, directed graphs, appropriate only to their immediate virtual surroundings, and was able (in results given in 2006) to animate a thousand characters in real time at a high frame rate of 100 fps excluding rendering time.

The graphs were embedded in building blocks called *motion patches*, which could be distributed throughout a target environment. A patch contained motion recorded in a part of the source environment which exhibited a characteristic geometry, such as a desk and chair setting. It could then be used to play back motion in a larger virtual target environment at all locations exhibiting the same geometry. A given patch included many different actions which virtual characters could enact, thus, for example, the desk and chair patch included motions for sitting down and standing up, working or chatting.

The approach first involved motion capture in custom-build physical environments. An analysis of the source environment's geometry then served as a basis for building patches, which included embedding recorded motion data in a graph structure to later provide a choice of motions for virtual characters. Patches were then automatically fitted to the pre-existing target environment with appropriate transforms, and interconnected by blending between the embedded motions dependent upon satisfactory continuity. Connections were pruned as necessary to ensure the creation of a strongly connected graph which ensured characters could navigate between any two locations in the virtual environment.

Using small graphs on a per-patch basis, while fast to search, meant there was no explicit representation of the overall graph covering the entire environment, making path planning impossible. To rectify this, a second higher-level graph was used with a

resolution indicating the connectivity among individual patches. This could be used to plan a route prior to refining the search for the shortest path into the lower layer.

The modular approach of Lee et al. [LCL06] to data re-use simplifies the considerable task of recording motion capture data and its application to large virtual environments. It is processor and memory efficient, and scales well with the size of motion data and with the complexity of the virtual environment. However, it is only of practical use for environments exhibiting repeated regular geometric structures, and many motion patches in the virtual environment will be clones of the other patches, which significantly reduces variety in the rendered scene. (The perception of duplication in rendered crowds is the subject of [MLD+08] discussed below).

A limitation of graph structures encoding the connectivity of large databases is the runtime cost of searching them for motion which matches the current requirements, be they imposed by the user or specified autonomously. Some of the approaches used to accelerate this search were described above [KGP02, AF02, LCR+02, LCL06]. (Jehee) Lee et al. [LL06] (in subsequent work to theirs only just mentioned) addressed the problem by precomputing avatar runtime behaviour as a preprocessing step. The precomputation created a table whose values reflected the expected long-term rewards for executing an optimal *control policy*, and it is this table, and hence the control policy, which guided the character's behaviour at runtime, indicating how it should behave given various states of the simulation.

The synthesised animation in their work involved the sport of boxing, in which a boxer approaches and hits a target, this target being either a punching ball or an opponent's head. Both avatar and target were considered to have a discrete set of possible states, and these alone determined, at each decision-making step, which of a discrete set of actions should be taken by the boxer.

Interconnectivity within the motion database was represented as a directed graph, with the nodes corresponding to static poses which were the avatar states, and the edges being connecting transitions – the actions – between those states.

The possible *target* states were not part of the graph per se, instead being represented by a grid of locations, centred at the avatar itself. Dimensionality, resolution and extent of the grid depended on its application. For example the grid for potential target positions used when throwing punches, was designed to accommodate the three dimensional nature of the target positions, was within arm's length of the avatar, and was finely-spaced to allow for precise punching behaviour.

The paper's main contribution was the precomputation of a lookup table, which would indicate, at runtime, which of the currently available actions would lead to the most beneficial combination of avatar and target states. This avoided the runtime bottleneck of searching the state-action space for an appropriate sequence of actions to reach a desired goal state. An iterative reinforcement-learning process was used to populate and incrementally refine the table, such that its listed values converged to reflect, for each possible combination of avatar and target states, the *long-run* sum of future rewards expected for executing the optimal policy when starting from those states.

The state-action model thus involved not only states requiring actions, but equally, actions creating new avatar and target state combinations. During graph traversal, selection of actions simply involves choosing the one which brought about that combination of states, for which the table showed the highest long-term reward. Thus despite its greedy nature the runtime strategy ensured the optimal choice is made. A degree of randomness in the choice of actions was also added, to reduce any monotony arising from a homogeneous approach to the selection of actions.

Despite using dynamic programming to avoid redundancy and reduce cost, learning was a lengthy process. The punching behaviour, for example, took seven hours to learn. In contrast, its execution at runtime was highly efficient and allowed (in results presented in 2006) 30 boxing avatars to be animated at 100 frames per second, while interacting responsively with each other and with the user. However, although the resulting policy allowed the animation of boxing against an opponent, the motion capture data used for training was limited to sparring with a *static* target, as learning was not feasible in a higher-dimensional state-space. Learning higher quality policies thus remained difficult, and policy adjustment at runtime – which would be possible with graph search procedures – was precluded in the method of Lee et al. [LL06].

Shin and Oh [SO06] also used a graph structure to represent interconnectivity within a database. In this case, however, nodes represented collections of similar poses, and graphs could have as little as one or two nodes. Edges entering or leaving a node encapsulated not a single motion sequence but, instead, *groups* of similar motions all starting or ending with a pose characteristic of that node, hence the term 'fat edges' leading to the name 'fat graphs'. An example thereof would be nodes for the left and right ready stances in karate, and one fat edge for each style of kick, with each such edge conglomerating all the sequences depicting a kick of that type.

The motion segments within fat edges were blended to form the actual sequence played back during animation, thus providing a continuous spectrum of possible interpolated motions, which avoided the usual limitation of motion graphs only being able to rigidly play back clips found in the input data. Three facets of the motion were catered for during blending: position and orientation, joint angles, and timing information. The latter employed correspondence maps acquired by dynamic timewarping (as introduced in Chapter 1, Section 1.4.1) to ensure blending occurred between matching poses in the input motion sequences.

An indirect but intuitive approach to user-selection of blending weights was employed. It involved first selecting an instant on a timeline, then selecting a joint on the character, and finally repositioning that joint as desired, from which the system calculated the necessary weightings to create just that blended sequence, whose character pose at the specified point in time resulted in the joint under consideration being placed at the user-selected location. More specifically, at the time under consideration each sequence in the fat edge contributes one pose and one 3D position for the user-selected joint. Random weightings were used for which the resulting joint positions were projected onto a plane forming a two-dimensional parameter space. The mapping between the points on the plane and the weightings which led to them was thus readily available. Furthermore, Delaunay triangulation was used to join the points on the plane after which Barycentric coordinates led to an approximate mapping for any further points in the plane. In this manner, the weightings were calculated which corresponded to all the intersections of a regular 2D grid placed over the parameter space, which in turn allowed fast weighting calculations for any point in the plane at runtime by bilinear interpolation of the weightings for the nearby grid points. A 2D device such as a mouse was thus perfectly adequate to chose a blended position for the joint in question, from which the system calculated the corresponding weightings and generated the 3D motion sequence, which included a frame satisfying the joint-position requirement at the time under consideration.

Motion graph traversal typically involves graph-search techniques, [KGP02, AF02, LCR$^+$02, LCL06]. However, it was the user-selection of desired actions, and hence of fat edges, which determined the path followed during the traversal of fat-graphs. User input was also possible during fat graph construction, with an option to manually select the base pose around which nodes were centred, as well as control of graph construction

by adjusting thresholds affecting both node and fat-edge creation, with a consequent impact on connectivity and motion quality.

The salient contribution of Shin and Oh's [SO06] work was its hybrid nature. It merged the temporal rearrangement of motion clips available through motion graphs, with the continuous control offered by motion blending. It did, however, require that the user select a motion type each time a node was reached, and that he subsequently choose a skeletal joint and position it as desired to define the blending within the fat edge. The complexity of this repeatedly required user-interaction suggests fat graphs may be more suitable for interactive authoring of motions, to be saved and played back at a later time, and not suited to applications requiring a more direct and user-friendly control of characters in real time, such as video games. The latter is addressed by the hybrid networks proposed and demonstrated in Chapter 6.

4.6.2 Pose Rearrangement With No Graph Structure

Like many others, including [LCR+02, LCL06, LL06], Arikan et al. [AFO03] created animation by rearranging sequences from a database of captured motion, but their approach dispensed with any graph structure. Instead, dynamic programming optimisation was used to search the database for fixed-length *blocks* of motion, and to place them in succession so they matched *annotations* the user had specified on a timeline, while also ensuring continuity between adjacent blocks and satisfying additional user-specified constraints. Efficiency was afforded by initially using large blocks to produce motion of high granularity which only crudely matched the annotations and constraints, and later refining the motion to ensure a better fit, by means of repeated optimisation steps with decreasing block sizes.

The aim of their work was to provide an intuitive control method which involved the user painting annotations on the timeline, in a similar manner to a director guiding

an actor's performance during a rehearsal. Their annotations – reflecting a database of American football motions – included actions such as 'walk', 'run' or 'jump' and modifiers like 'reach', 'catch', 'carry', although the vocabulary could be freely chosen by the animator to suit any particular application. Composite annotations such as 'carry while running' added flexibility as did negative ones like 'don't run backwards'. Additionally, constraints could be specified requiring that the motion pass through a particular pose, or a particular position and orientation, at a specified time.

Despite dynamic programming, searching even a small database for suitable sequences to join would have been prohibitively expensive for interactive authoring. A hierarchical search algorithm was thus used, creating motion which *approximated* that obtained by comprehensive searching, while restricting cost to an acceptable amount.

At the highest level 32-frame blocks were combined to create a crude motion which only roughly met the user requirements. For efficiency, the blocks were selected from a small number representative of the entire database, these having been obtained by k-means clustering the larger collection the database comprised. The search was then refined using 16-, and later, 8-frame blocks. Again, clustering reduced the search space for faster optimisation, but clusters were this time centred around data present in the coarser motion created at the previous, higher level. The lower level searches thus fine-tuned the result, while the upper-level search ensured good database coverage.

By selecting from current and previously created motions, and concatenating the start of one motion with the end of another, new output motions could be synthesised each with a unique end-location for the character at the final frame. Optimisation was thus able to consider such concatenations and try to satisfy goal position constraints too. However, there was no guarantee a goal location would be reached, as this depended on the suitability of the motions available for concatenation.

To lessen the task of database annotation, much of it was performed algorithmically, using a process based on the use of support vector machine (SVM) classifiers. For each annotation, an SVM was trained to decide whether motions satisfy a given annotation or not, having learned to do so from a small manually annotated (or user-verified) training set. In this manner seven minutes of motion took "under an hour" to annotate. However, this does not appear strikingly faster than manual annotation.

While Arikan et al. [AFO03] succeeded in providing annotation-based control of character motion, its use is possibly limited. Many games and other applications portraying virtual environments require characters which are able to navigate their virtual environment while being driven by the user, or while following a predetermined path. The use of annotations, via a suitable interface, may benefit such applications, but the approach does not provide character navigation per se. Having specified a goal location, little control was available over the path followed to reach it. Worse still, as explained above, the goal may never be reached. Furthermore, while synthesis was fast enough for interactive authoring (in 2003), it may not suffice for real time animation in virtual environments.

Treuille et al. [TLP07] also dispensed with an explicit graph structure for their concatenative approach to animation, thereby avoiding the "great amount of skill and manual adjustment" [TLP07] graph construction entails. The presented approach used reinforcement learning to automatically compute kinematic controllers from given motion capture data, removing the costly need to search a graph structure at runtime, and generating fluid-looking real time character animation providing both interactive control and automated obstacle-avoidance.

The method used a two-layered approach. At the lower, a motion engine stitched together captured motion clips by blending in real time. The upper level executed a

near-optimal control policy (explained below) selecting the best sequences of clips to achieve some multivariate goal.

The controller must decide which sequence best achieves its goals quickly and naturally. To this end, goals were expressed by assigning a cost to each state and transition, which were inversely proportional to how well they fulfil the goal. For example, for navigation tasks, deviation from the desired path or proximity to an obstacle were penalised by raising the state-related cost. Similarly, transition costs ensured smooth character motion. Specifying goals in terms of numerical costs does not of itself specify *how* to achieve these goals, however. A greedy policy – choosing the clip yielding the immediately lowest state and transition costs – is simple, but is often inferior to a forward-looking policy able to sacrifice short-term objectives to achieve better results in the long run. The optimal policy looks ahead, accounting for delayed rewards and choses that sequence of clips which minimises the entire policy cost spanning into the future. This cost is defined by a value function (mapping states to expected cumulative future reward) [GE07] which in turn is sufficient to fully specify the optimal controller. At runtime, when a clip finishes playing, the controller can quickly select the best motion clip to transition to, as the clips themselves are also state variables. At the end of this subsequent clip the best next one is again chosen, a renewed choice being due as state variables such as the character's position and orientation will have changed.

Controllers were built automatically given source data and a set of controller objectives, using reinforcement learning, a method which previously (as in [MA95]) had required storing a table of samples to represent the optimal value function at each state [DF02], this being intractable for large-scale problems. A compact approximation of the value function using basis functions had later been presented by [DF02], thereby avoiding the need to store samples and the associated high controller memory costs. This method was adopted by Treuille et al. [TLP07] with the value function approx-

imated by a linear combination of n basis functions, requiring only a comparatively simple optimisation to solve for the n-dimensional vector of weightings instead of solving for the complete value function. The method is designated as *near-optimal* as an exact representation of the value function could not generally be achieved.

Treuille et al. [TLP07] also mitigated "the curse of dimensionality" [TLP07, DF02, MA95], in which "the size of a state space typically grows exponentially in the number of state variables" [DF02], by learning controllers only within certain subspaces, and parallelising some aspects of precomputation. Furthermore, a set of controllers could be constructed separately for different tasks, and later transitioned between as necessary, greatly reducing controller construction costs by using separate optimisation tasks before connecting the controllers together.

The motion model, at the lower of their two-level approach, avoided footskate during blending for some classes of motions like walking or running. It did so by specifying constraint frames during the middle of the ground contact phase, overlapping such frames from the two motions being stitched, and "re-rooting" [TLP07] the skeleton so the stance foot was treated as the skeletal root. Inevitably, however, when input motions had stance phases of different length, the outer limits of the blended stance phase could exhibit residual footskate.

The method's accompanying video lives up to the paper's claims of fluidity and aptly demonstrates the power of reinforcement learning, with – within limits – successful avoidance of moving and stationary obstacles, including small crowds shown (though not guaranteed to be) devoid of character collisions. With the forward-looking nature of the optimal policy sharp turns did appear much more convincing and had a faster response to user input, than those created with a greedy policy which was shown for comparison. The method, however, was "limited to simple environments described with a small set of parameters" and only *mitigated* the curse of dimensionally, with

Treuille et al. [TLP07] conceding that "the number of bases must generally still grow exponentially with the size of the state". Furthermore they required large amounts of manually categorised input data (384 clips), despite only synthesising three motion types – walking, running and twirling – indicating demanding source data requirements.

4.7 Motion Interpolation

Unuma et al. [UAT95] modelled locomotion with an emphasis on conveying human emotions. Fourier expansions of measured human motion data were used to express it in terms of its component frequencies. Both interpolation and extrapolation of motions could then be performed by blending the Fourier coefficients from two different motions, thus creating a hybrid motion in the frequency domain. After blending, the data was returned to time-domain representation by evaluating the hybrid Fourier expansion, using successive values for the time parameter t. Transitions, from one motion type to another, were also performed, and this by varying the blending weightings over time.

Additionally, the quality or mood of a motion could be extracted by subtracting the Fourier coefficients of one motion from another and superimposing the result over the coefficients of a third motion. Thus, for example, measured data for both a brisk and a normal walk was used to obtain the quality of 'briskness', which was then added to running motion data, leading to the synthesis of a brisk run.

Finally, Unuma et al. [UAT95] also performed motion *editing*, some of which was done in the frequency domain. Control over step-length and speed was performed by manipulation of the Fourier expansions, while the duration and height of the flight stage of running motions was controlled in the time domain, by means of two parameters which adjusted the vertical position of the character's hips.

Their work addressed the need to provide an easy way for animators to create

natural-looking movements, doing so by means of blending and the extraction of characteristics such as tiredness, from a small amount of experimental data.

It was, however, limited to periodic motion. Furthermore, it was purely geometrical, which could lead to the generation of physically incorrect behaviour. The inclusion of dynamics methods would address this issue, and might, of itself, serve as a basis for extending motion extrapolation beyond the capabilities already allowed by the method. While seemingly slow at about 10 frames per second the method, (presented in 1995), would easily exceed real time performance on modern hardware (as confirmed by measured results for the more complex frequency domain work of Molnos et al. [MLD10], discussed below and the focus of Chapter 5).

Bruderlin and Williams [BW95] used both image and signal processing techniques to create, modify, and blend animated motion. They presented four methods which are listed below.

- Multiresolution motion filtering
- Multitarget motion interpolation (including dynamic timewarping)
- Motion waveshaping
- Motion displacement mapping

The second method, clearly involving blending, used the output of the first, both of which will be described next. The third and fourth methods are motion editing techniques and involve no blending. For the sake of consistency these are described in Section 4.8 (Motion Editing).

Multiresolution filtering comes from image processing where it is applied to two-dimensional data. The original image is repeatedly sub-sampled with a low pass filter, which halves the number of pixels in each dimension at every step, ending with

a final image of just one pixel. With each step the drop in image resolution results in loss of detail, thus reducing the higher frequency content and acting as a low-pass filter, with the final image showing only the average or DC value. This produces a low-pass pyramid, so-called due to the appearance of the ever-smaller images when stacked on top of one another.

A second pyramid is then created known as the bandpass pyramid, whose layers are the *difference* between successive images in the low-pass pyramid (pixel-dimensions having been equalised prior to subtraction). The difference image thus contains frequencies present in the larger of the low-pass pyramid images which have been filtered out of the smaller. They thus contain the higher frequency content of the low-pass pyramid layers.

The original image can be reconstructed by summing all the bandpass layers plus the DC value.

Bruderlin and Williams applied the same methods to one-dimensional angular motion data, as well as to joint positions. The relevance of dividing motion data into into passbands is that low frequencies contain general, gross motion patterns, whereas high frequencies contain detail and subtleties. Particular frequency bands could then be amplified, attenuated or even made negative.

Various effects could be generated, such as movement with a nervous twitch, or qualities akin to anticipation and follow-through, as well as squash-and-stretch cartoon-like walks (Chapter 2, Section 2.2) [Las01, TJ95, RP12]. Furthermore they suggested creating motion sequences by starting with a generic motion pattern made of low frequencies, and fine tuning it by adding higher frequency refinements. However, the technique does not seem to be an intuitive one offering predictable control over the motion, and achieving a particular *desired* motion would appear to be difficult.

The method can lead to violation of joint limits as well as foot-floor penetrations, with Bruderlin and Williams' philosophy being to correct such artefacts after the general character of the motion has been defined.

The constituent motion components obtained via the bandpass pyramid can also be blended using multitarget interpolation, which combines contributions from two or more motions into one. The blending ratios determining the share of each motion to be added to the mix can be set differently for each of the bandpass frequency ranges, and can also be made to follow a trajectory in time.

To successfully blend walk cycles the steps must coincide so feet strike the ground at same time. Thus multitarget motion interpolation must include not only a blending function, but also a remapping of the motion data with respect to time, which is performed by dynamic timewarping.

Dynamic timewarping has been used in speech recognition [Sen08, BW95] to identify a combination of expansion and compression along the time axis which can warp two signals for best correspondence. When applied specifically to motion data [KG03, SO06], it first establishes a correspondence between samples of the discrete signals present in trajectories from two different motions. The second step then involves actually applying the warp as specified by these correspondences.

In the initial step, for each sample of one signal, a sample in the other signal is assigned such that a global cost function measuring the "difference" between the two signals is minimised. The difference is found by calculating how much work it takes to deform one signal into the other, with the total cost function consisting of the sum of the local stretching and bending work terms. The solution space can be represented as a two-dimensional grid, with the sample assignments forming a path within it. The initial and final samples of each signal are always made to correspond, so the path runs from grid reference $(0, 0)$ to (n, n), and a dynamic programming optimisation technique

is used to select that route which joins them at minimal cost. In the second step, the path on the grid is used to control the warping of one sequence of motion data to the other.

As mentioned above, Bruderlin and Williams [BW95] also presented *motion waveshaping* and *motion displacement mapping*. These, however, are motion editing techniques and correspondingly described in Section 4.8 below.

Wiley and Hahn [WH97] blended motion capture data to synthesise simple character motion, with the desired blend specified in a convenient space of parameterised motions (further explained below). Providing "inverse kinematics capability" [WH97] with no need for conventional IK, the method computed such joint angles as would position the hand or foot of an articulated figure at a desired location in space. Termed 'parameter space', the values it spanned (such as hand locations while reaching) were more meaningful to the user than the space of DOF-values otherwise defining the skeletal pose. A pose fulfilling the target-location request was obtained by blending the most closely complying poses in the input data which was achieved, however, while specifying pose requirements in one space and blending in another. The method is described below.

Pursuing the reaching example, the motion captured input was a collection of reaching actions, and a subset of its poses was sought whose hand positions were closest to the specified reach target, these forming a volume around it in parameter space. A lack of precision and orderliness in captured data, however, even in collections of a single type, often meant an exhaustive search was necessary to find this subset, a search deemed prohibitively expensive for data sets with hundreds of poses. To accelerate future searches, the input data was resampled so the hand locations of its reaching actions covered a regular grid in parameter space. During synthesis, the now orderly collection of grid-forming poses replaced the previous input motions. It allowed efficient

selection of that subset of frames which most closely enclose the target, with no need for a costly search.

Such subsets of poses were interpolated, twice, once during preprocessing to build, from the input motion, its grid-aligned substitute, and once at runtime, creating the user-requested pose by blending from a subset of the resampled now-aligned input frames. At runtime, blend weights were computed by considering the desired target reach-point and the hand positions of the reaching actions in the selected subset of grid-forming poses. Similarly, in preprocessing, weightings to synthesise poses whose hands line up in a grid pattern, were based on the location of the selected grid point, and that of the hands in the subset of input frames.

Subset interpolation was trilinear, a type of multitarget blending, which was applied independently to each degree of freedom of the skeletal data. The nonlinear relationship between skeletal DOF values, where blending took place, and parameters of interest in parameter space (such as hand location) precludes full compliance to specifications while blending. However, given sufficiently dense data, the generated pose does closely approximate the desired parameter value (and reaching, for example, can be said to attain to the goal). Furthermore, if required, higher accuracy can be obtained by an iterative optimisation process. Additional contributions included the positioning of feet in a bicycle riding motion by a procedure including a cylindrical coordinate system, and motion walking up a slope, with walk cycle duration and gait style continuously changing in response to variations in steepness.

At runtime the arm-reaching activity could accommodate continually varying parameters, entered interactively, resulting in motion described as smooth and continuous. Being a blend of motion capture data, subtleties such as knee bending, back bending, and motion of the other arm occurred during the reach operation, adding realism. Real time performance was claimed for all activities including cycling and running, but not

quantified. While the synthesised motion was simple with character navigation not featured at all, the *approach* to blending seems noteworthy, allowing the desired result to be specified not in terms of DOF values, but in a more meaningful parameter space, the utility of which is highlighted by similar parameterised spaces of motion following in later papers such as [RCB98, KG04, HG07]. The method, is, however, limited to a small number of parameters, of the order of ten [WH97], although the examples of Wiley and Hahn above used no more than three. Data requirements were a major drawback, at least doubling with every parameter added, this cost arising from the need to populate a higher-dimensional parameter space. Furthermore, as applies to blending in general, there was no guarantee of producing physically correct motion, nor of honouring kinematic constraints such as joint limits. Nevertheless, the method allowed motion to be smoothly controlled via meaningful parameters, it was (in 1997, already) suitable for simple interactive games and, considering the lightweight nature of its interpolation parameters, also, over a network, for avatars in multiplayer virtual worlds.

As with Wiley and Hahn [WH97], Rose et al. [RCB98] interpolated complex linked figures, but verbs and adverbs – the name of their approach – scaled better with the dimensionality of parameter space.

The verbs were parameterised motions, built from sets of distinct yet structurally similar examples obtained by capture or keyframing. The adverbs were the dimensions of the resulting parameterised space. Adverbs quantified emotional expressiveness such as happiness or sadness, and control behaviours like turning or walking up, or downhill. Additionally, verbs could be interconnected with smooth transitions between them. A pre-processed verb graph encapsulated those interconnections found feasible. At runtime, the overall structure provided twofold control. Firstly, the precise nature of verbs, thus the quality of the motion, could be tuned by the adjustment of interpolation

parameters (adverbs). Secondly, the graph allowed transitioning between verbs, thereby switching between fundamental motion types. The method was thus a forerunner not only of motion graphs, as later presented in the seminal work of Kovar et al. [KGP02], but also of hybrid approaches, as followed, for example, with Heck and Gleicher [HG07]. Furthermore, a fast inverse kinematics optimisation was presented, with uses including footskate avoidance for the support phases of locomotion cycles.

Wiley and Hahn [WH97], above, had used uniform time scaling to equate the lengths of motions in a blend, demanding, however, a structural similarity between inputs. In contrast, Rose et al. [RCB98] preceded blending by a non-uniform scaling, performed with reference to hand-annotated keytimes, defining frames such as foot-down events. Timewarping was thus performed, (though distinct from dynamic timewarping as, for example, in [BW95, SO06, KG03]). Keytimes were also used for constraint enforcement, by specifying, for example, the start and end of footplants. Adverb values, linking example motions to points in parameter space, were manually added too.

Interpolation in the method of Rose et al. [RCB98] was by a combination of radial basis functions and low order (linear) polynomials. These created the space of interpolations between example motions. Furthermore, for each example, each DOF was represented by a uniform cubic B-spline curve with associated control points. The spline control point for each *interpolated* DOF curve was then given by a weighted sum of radial basis functions plus a weighted sum of linear bases. Keytimes were interpolated in a similar manner. As with Wiley and Hahn [WH97] only *subsets* of the example motions contributed to blending, which was ensured in the verbs and adverbs approach [RCB98] by *compact support* of the basis functions (their being zero-valued beyond some distance from the example motion) limiting each example motion's influence to a local region of adverb space.

Quite distinct from the above is verb-verb transition interpolation, although both occur at runtime. Neither this nor offline graph construction, nor runtime graph searching are elucidated here, however, as the work of Rose et al. [RCB98] is marked mostly by its use of multidimensional radial basis interpolation for the synthesis of verbs qualified by adverbs.

A quintessential example of the multi-facetted nature of animation work, the hybrid system of Rose et al. [RCB98] combined continuous and responsive control of adverbs, with seamless constraint-honouring transitions between verbs, and was said, by the authors to "show richer content than existing techniques". Implemented walking, jogging, reaching, and idling motions ran in real time, requiring, at worst, ca. 5 milliseconds to evaluate a character pose (in results published 1998). Furthermore, data requirements were polynomial in the dimensionality of the control space and thus superior to the exponential costs of Wiley and Hahn [WH97]. Presented screenshots could appear cartoon-like, however, and some blatantly failed to show natural-looking motion – the greatest challenge in character animation. A sad and clueless walk, for example, was shown with the head dropping down to waist level, and a happy and knowledgeable one was leaning back to an extreme extent, though perhaps only to demonstrate parametric control efficacy. A clear drawback, however, was the constraint mechanism, which brusquely set the elevation of the foot to that of the terrain. Although claimed to be "subtle changes" only, the eye will pick up the slightest aberration.

Exemplifying a recurring, almost contradictory pattern in the literature, whereby methods seen to aptly fulfil requirements are deemed, shortly thereafter, to be significantly lacking, Park et al. [PSS02] addressed the issue that "previous methods can hardly generate the convincing locomotion of a character following a curved path with a desired speed and style". Their proposed remedy built upon the very similar "verbs and adverbs" approach of Rose et al. [RCB98], as well as its subsequent adaptation

to shape blending by Sloan et al. [SRC01]. The principle differences are enumerated below.

Instead of a verb graph as in [RCB98] interconnecting fundamental motion types such as running or walking, the method of Park et al. combined all verbs into one "large verb" [PSS02]. Speed, rate of turn, and locomotion type (running, walking, etc), were core dimensions in a single parameterised motion space said to offer easier runtime control than transitioning from verb to verb in a graph. Additional parameters, for emotional content or physical characteristics, akin to the adverbs of Rose et al. [RCB98] could be defined as before.

Motion blending at runtime comprised four steps: weight computation, timewarping, posture blending, and motion retargeting (Chapter1, Section 1.4.2).

Weight computation. The use of radial basis functions for scattered data interpolation by [RCB98] (described above) was modified by Sloan et al. [SRC01], who instead of interpolating the control points of DOF spline curves at every frame computed, by a similar method, the weights of the example motions. This lead to greatly improved efficiency as animations typically entail far fewer input motions than spline curve coefficients [SRC01]. It is this lower-cost method that was adopted by Park et al. [PSS02] for weight computation.

Timewarping. In the verbs and adverbs method (above) [RCB98], example motions were aligned by timewarping, expressing them in generic or canonical time for blending, after which an "untimewarp function" [RCB98] returned them to actual time. However, as stated by Park et al. [PSS02] "the range of speed over example motions may also be quite large", which, combined with changes in blending weight may "cause a blended actual time of the target motion to go reversely, that is, go back to the past, with respect to the generic time". The reason, very

briefly, is that example (ie input) motions of different speeds also have differing cycle lengths (in actual, non-timewarped, time). So weighting the output mostly on one input, and then more on another, changes the blended cycle length, which, especially when synthesising the end of the cycle, can shift the simulated time-point in actual time in a backward (or forward) direction. To avoid this, an *incremental timewarping* scheme was presented, which by blending the gradients of the strictly monotonic example motion timewarp curves, created a strictly monotonic *blended* timewarp function, thereby guaranteeing a forward-moving simulation time-point in the actual time, during the uniform progress of generic time.

Posture blending. "With Euler angles, it is non-trivial to ensure that similar poses use similar Euler angles" for which reason "conventional methods using Euler angles are required to preprocess the example motions" [PSS02]. To obviate such preprocessing Park et al. presented a scheme using unit quaternions, which guaranteed a similar representation for similar poses, and ensured consistent parameterisation over example motions.

Motion retargeting. The blended posture was retargeted, both to adapt it to the target character and to fit it to the environment, by, according to [PSS02], the method of Shin et al. [SLSG01] (Section 4.10). The latter, however, assumes level terrain, and explicitly states "The floor is modelled as a plane for all of the uses of our system to date" [SLSG01]. For this reason, and others, explanations of the adjustment of gait in [PSS02] to match the terrain, remain, it seems, inconclusive.

Results described walking and running actions, built from repeated short clips (two steps), and able to follow gently curved paths and undulating terrain in real time (1.2 milliseconds per frame without rendering, in 2002). The contribution of Park et

al. [PSS02], however, appears unclear and open to debate. To mention one point only in the interest of brevity, the value of having one large verb [PSS02] was not demonstrated. Using continuous blending to switch between basic motion types within a single verb, instead of transitioning between them by means of a graph, only seems feasible with similar and hence limited motion types, as indeed they were using. The approach thus limited motion variety, in contrast with synthesis by concatenation, which was markedly popular in much following work [KGP02, LCL06, TLP07].

Pettré and Laumond [PL06] built on the Fourier expansion work of Unuma et al. [UAT95], and the multitarget interpolation of [BW95], by using the discrete Fourier transform to blend between three selected motions, in the frequency domain. Their work focussed on creating a locomotion controller whose inputs specified the required linear and angular velocities for the motion to be created, and outputs provided time-varying values for the character's degrees of freedom. Control inputs could be continuously changing and either user-defined or automatically generated by a motion-planning module.

The presented approach comprised four stages, input motion analysis, representation in 2D control space, frequency domain blending, and extracting postures to create the output sequence. The steps are detailed below.

1. Motion capture data in a library was automatically analysed to determine the associated linear and angular velocities of the character's root.

2. The input motions were represented as points in a two-dimensional control space for which the vertical axis represented linear velocity, and the horizontal axis angular velocity. The points were then joined with a Delaunay triangulation.

3. Like the input motions, the motion to be synthesised was also represented as a point in the velocity space, this time specifying the *required* linear and angular

velocities. The vertices of the triangle it lay in specified three input motions from the motion library, to be blended into a new, hybrid motion, by means of interpolation in the frequency domain. More specifically, the frequency spectra – the Fourier series expansion coefficients[1] – of the character's degrees of freedom were obtained via the discrete Fourier transform for all input motions during library initialisation, and a blend created from the three selected input motions at runtime. Interpolation weights for coefficient blending were obtained by solving a simple linear system which presumably (though not strictly specified) yielded the barycentric coordinates of the desired motion's location within the triangle. The vertex lying closest to that location thus had the greatest influence on the output motion. Extrapolation, with the desired motion's coordinates lying outside the Delaunay triangle coverage, was possible albeit with potentially unnatural-looking results.

4. The hybrid cycle was expressed as a collection of Fourier synthesis formulae, one for each degree of freedom. They allowed motion representation to be returned to the time domain by *extracting postures*, a term by which Pettré and Laumond refer to the process of obtaining skeletal poses by evaluating the formulae at given points in time. This built up a sequence of frames to produce the animation. For steady controller inputs it merely involves substitution of the current time value into the formulae. However, postures can also be generated for changing controller inputs and to function properly in this case, extraction of postures had to account for the potentially evolving period of the output motion.

[1]The Fourier series, Fourier synthesis, the discrete Fourier transform and related matters are explained in Appendix A

Pettré and Laumond's contribution was to represent input and desired output motions within a 2D control space. This allowed automatic selection of input motions from the library and blending weight calculation, prior to synthesising a new motion which satisfied a given output velocity specification. The triangle network envelope could also be employed to indicate to a motion-planning module which motion *velocities* could be synthesised. Furthermore, the paper included an illustration of how the controller could be used in a collision-free motion-planning context.

Synthesis was in real time. Some timing data is to be found in Chapter 5 which presents the more efficient approach of Molnos et al. [MLD10] and compares it to [PL06]. Motion was said to be "realistic", though "unbelievable motion details (foot skates, limb interpenetrations, etc) can be observed in the resulting animations under certain conditions" [PL06]. Further limitations exist, of which some now follow. Pettré and Laumond synthesised walking motions only, although the method is, in principle, not limited in this way. Input motions were subject to restrictions such as needing to have constant velocities, and needing to be in phase, for example, by having each motion start with a left foot strike. Additionally, they required all input motions to have the same length, which when they did not, was corrected by resampling based on linear interpolation. The general concept, however, of frequency domain triangle network-based blending is one regarded with interest in this thesis, and shared by the practical work presented in Chapters 5, 6 and 7.

The work of Pettré and Laumond [PL06] was, three years later, streamlined by Molnos et al. [MLD10]. This work, and the differences with its predecessor, are treated comprehensively in Chapter 5, with only a brief overview given here to avoid duplication.

While the earlier work had focused on creating a controller particularly well suited to motion-planning, Molnos et al. [MLD10] concentrated on character navigation within

virtual environments, such as video games. To this end, the triangle network defining a continuous blending space was made more flexible, and instead of reflecting root velocities, was presented as an intuitive and adjustable interface element, which, furthermore, allowed motions to be blended which had not been possible with the method of Pettré and Laumond [PL06]. In addition to this, higher efficiency was obtained in a number of ways, notably by using a cheaper Fourier synthesis formula than had [PL06]. While the formula was nothing new [UAT95], the contribution lay in presenting a solution to the issue of phase angle blending, which thereby allowed this formula, unlike its use in [UAT95], to consistently generate correctly-formed animation poses, and to remain efficient by transferring associated calculations from runtime to preprocessing.

Weighting calculation, blending and Fourier synthesis of realistic-looking motion using five harmonics required (as published in 2010) 0.39µs per degree of freedom for each frame in the created sequence – a once-only cost incurred only when blending ratios changed. As shown in Chapter 5 this figure surpasses the efficiency of [PL06] (and could be improved further using proposed level-of-detail adjustments).

The streamlined approach of Molnos et al. [MLD10] is expounded in Chapter 5. It furthermore lays the foundation for *hybrid networks*, novel structures whose operation and great utility are detailed in Chapter 6, leading, in Chapter 7, to further network refinement with the innovative process of *SSH switching*. Further advantages, and potential, of the streamlined approach thus follow in later chapters.

In their presentation in Chapter 6, the above-mentioned hybrid networks are compared to the parametric motion graphs of Heck and Gleicher [HG07] on account of a degree of similarity shared by both methods. The latter, is now described. Parametric motion graphs [HG07], like fat graphs [SO06] before them, combined blending-based synthesis with synthesis by concatenation. They united, to an extent detailed below,

the accurate parametric control of the former with the ability to switch from one parametric motion space to another using linear blend transitions.

Collections of short clips enacting similar activities, such as taking two walking steps, or cartwheeling, were grouped together, with each such grouping represented as a graph node. Motions within nodes were interpolated to generate the output animation, as specified by one or more user-guided, or algorithmically-specified parameters. A node thus represented a continuous space of motions instead of a collection of discrete clips. The contribution of [HG07] was the description of a technique said to create transitions between such continuous motion spaces, hence the name *parametric* motion graphs, though, as discussed below, both the name and the claims, while technically correct, perhaps suggested more than these useful structures were capable of.

Establishing a transition between two continuous spaces relied on earlier offline computations whose results were stored within graph edges. Edges were built using the distance metric introduced by [KGP02] and entailed consideration of the $n_s n_t$ potentially viable transitions between a list L^s of n_s *randomly sampled parametrically derived* (thus blended) motions in the source node, and n_t such motions in the target node (as distinct from the usual approach of transitioning between the *original* motion clips). For each parametric motion sample in the source node a set of motion samples in the target node was identified to which transitioning was feasible. The latter were mutually similar, hence specified by similar parameter values, and they defined a continuous *sub*space in the target node motion space, within which any parametrically synthesised motion could also be transitioned to from the source motion sample under consideration. A single edge linked source and target node, and described, for each source node sample, the associated target node subspace in the form of the dimensions and position of the smallest possible axis-aligned bounding box able to enclose it.

At runtime, prior to node-node transition, the k nearest neighbours to the particular motion currently being synthesised were selected from L^s, the previously randomly sampled motion clips in the source node (k normally being one more than the number of dimensions in the source space). For each of these k samples, the subspace bounding box details whose description had been stored in the graph edge in preprocessing was extracted and contributed to a weighted average, blending the sample-specific box dimensions. It is the resulting box, thus obtained by interpolation at runtime, which specified the target node subspace of mutually similar motions to which the motion currently being synthesised in the source node could smoothly transition. Further details required for transition creation, such as timing information, were similarly found stored in the graph edge.

While, as described above, parametric motion graphs did (in a sense) transition from the continuous motion space in the source node to that in the target node, this claim of Heck and Gleicher [HG07] may have been somewhat optimistic, one reason being that it was only a *subspace* of the target node that could act as destination for transitions. The typical size of this subspace was not quantified, but as stated by [HG07], the *similarity* of motions within the subspaces whose dimensions were stored in graph edges was fundamental to graph creation, suggesting the runtime-blended subspace was significantly less than the full target node motion space. This was also suggested, by the similarity which existed between the motion being synthesised in the source node and the range of continuous motions in the target subspace, a similarity which originated from the distance metric of [KGP02] used in preprocessing. This means that while transitioning *from any* part of the source space was possible, transitioning *to any* part of the target node space was not, so transitioning between continuous motion spaces, the very purport of parametric motion graphs, was only possible in limited sense.

This was aggravated by the fact that, upon reaching the target subspace, parametric adjustment to move out of this space was not possible, as the authors [HG07] themselves stated "we do not adjust the parameter vector while generating a motion", so having transitioned to the limited target subspace, parametric motion adjustment was apparently not possible until the next transition, at which point a change was again only possible to a chosen point in the limited subspace of the new target box. By comparison, in the transitions of the hybrid networks presented in Chapter 6, which anyway can do more than merely join blending spaces, the concatenation they perform can, with minor restrictions explained in that chapter, be set up to be available *from any* part of a target blending space *to any* in the destination space. Furthermore, unlike the method of [HG07], which failed to build a graph edge between nodes if even one single sample in L^s could not be assigned a target subspace bounding box, the hybrid networks of Chapter 6 can join *any* two motions, no matter how disparate.

Quite apart from the limited ability of parametric motion graphs to reach throughout the target node space, the latency issue described above was a problem in itself and again confirmed by Heck and Gleicher [HG07], attesting that "for motion spaces that represent long motions, it may take time for the character to react to user requests", which was apparently because the currently playing synthesised motion had to finish playing before the program reacted to user input upon the next transition, a delay which surely applied to shorter clips too, though less severely. This surprising inability to adjust parametric synthesis during motion generation appears to make the claim that transitions connect *parametric* motion spaces overstated. In contrast, in the analogous blending space of the hybrid networks of Chapter 6, the avatar responds *immediately* to user input during interpolation anywhere within a blending space, allowing, importantly, *continuously responsive* latency-free smooth control of character navigation while blending.

Another issue with parametric motion graphs, is that to be truly in control when performing node-node transitions, the user driving the animation would have needed to know the *extent* of the target subspace which was repeatedly changing in size *and* in position within the target node. Heck and Gleicher worked around this by *overriding* user input to lie in the target bounding box, but added that "by limiting the transitions to good ones, our characters occasionally miss targets". In comparison, when blending with the method proposed in Chapter 6, the user is always aware of the entire space of available motion to select from and the character will always respond as instructed.

Further comparison is found in Chapter 6. The above-mentioned drawbacks should not detract, however, from the important contribution made by Heck and Gleicher [HG07]. Implementation issues aside, the seamless interconnecting of blending spaces is indeed a useful approach in character animation.

Kovar and Gleicher [KG03] presented an approach to blending itself (thus only to a *subset* of the greater problem of motion synthesis). It encapsulated pre-existing techniques within a new data structure designed to expand the class of motions that could be successfully blended without manual intervention. Given the name 'registration curve', it was created automatically and described relationships involving the timing of events, local coordinate frames, and constraints, of an arbitrary number of input motions.

Registration curves were used with linear blending – a standard blending method based on weighted averages – of the skeletal parameters of each input motion. However, blending was enhanced with the aid of information in part automatically extracted from the input motions, which determined which frames to combine, how to position and orient them prior to averaging, and what constraints should exist on the synthesised motion. This information made possible the creation of three data structures: a time-warp curve, a coordinate frame alignment curve, and a set of constraint matches. These

were merged into the final structure, the registration curve, whose three components are detailed below.

Timewarp curve. The first component addressed the fact that blending may yield poor results when motions are not synchronised, ie when corresponding events – like a left foot strike – occur at different absolute times. It made use of dynamic timewarping to distort input motions along the time axis and ensure events with identical meaning occur simultaneously. Dynamic timewarping [BW95, Sen08], (as explained, for example, in Chapter 1, Section 1.4.1), creates a dense set of frame correspondences between input motions, which, in the method of [KG03], was based on the heuristic that frames are more likely to correspond if they are geometrically similar. Kovar and Gleicher [KG03] extended this, however, by passing through these frame correspondences a uniform quadratic B-spline, which, while initially only *approximately* strictly increasing, was then adjusted to be quite strictly so, giving the final timewarp curve, $S(u)$. It had the characteristic that an increase in the parameter u was always associated with an increase in the frame index of the input motions.

The timewarp curve had as many dimensions as there were input motions, and it associated every point on $S(u)$ with a set of frames – one frame from each input motion. Being strictly increasing, the curve described a one-to-one correspondence between input motion frames and parameter u, ensuring an inverse function existed, such that for a given frame of one input motion, the parameter u could be found, from which the corresponding frame of any another input motion could be obtained.

Unlike two-dimensional timewarp curves, those linking a greater number of input motions were prohibitively expensive to build, as the underlying dynamic time-

warping step scaled exponentially with the number of motions. To avoid this, an efficient algorithm was presented which combined multiple curves of two dimensions. More specifically, one of the input motions was selected as a reference motion, and paired with each other input motion, establishing frame correspondences between that reference motion and every other, and thereby indirectly between all the input motions. This generated an approximation of the *exact* results which would be obtained by the costly generalisation of the two-dimensional timewarp method to a greater number of motions. Having found the correspondence between all the frames for k input motions, a k-dimensional quadratic spline was then fitted and, as described above, adjusted to be monotonic and strictly increasing.

Coordinate frame alignment. Another consideration was the alignment of skeletons prior to their blending, which was accounted for by the second function of registration curves: coordinate frame alignment. The process rotated the character in each motion about the vertical axis, and translated it in the ground plane so as to make, for the current frame (ie time-point) in the blend, each motion as similar as possible. Such alignment was one step in a character-positioning process which avoided failures sometimes found when using linear blending directly[1]. The input motions were left unspoilt as the rigid 2D transformation did not change their nature, but merely moved their local coordinate frames. The current coordinate frames of each motion were aligned such that their corresponding character depictions reached maximum similarity in a least-squares sense, after which was

[1]Figure 5.4 of Chapter 5 illustrates what was stated in [KG03] to be a failure of linear blending. This failure, however, is entirely precluded from the linear blending of the methods presented in Chapters 5, 6 and 7.

applied a final transformation orienting and translating the coordinate frames by identical amounts which depended on the blending weights. Registration curves assisted the initial transformation stage, by means of alignment curves. An alignment curve was a function $A(u)$ that gave a set of transformations which, at each point on the timewarp curve, aligned input motion coordinate frames as described above.

The alignment curve was built by employing the frame (ie time-index) correspondences in the timewarp curve. It specified, for the two-motion case, a rigid 2D transformation $\{\theta_1, x_{0_i}, z_{0_i}\}$ that aligned one motion frame to the other. A 3D quadratic spline was then fitted to these transformations, yielding an alignment curve $A(u)$. Constructing the *alignment* curve $A(u)$ for more than two motions was analogous to the extension of *timewarp* curve construction to a greater number of input motions, and so will not be further detailed here.

Constraint matches. The third and final component of a registration curve was a set of constraint matches. Each constraint match contained a set of related constraints, one from each motion. As a prerequisite to collating them it was assumed the input motions were annotated with constraint information, and Kovar and Gleicher [KG03] did this using established automated methods with subsequent tuning of the results by hand.

Identifying *matching* constraints in the different motions was based on the heuristic that corresponding constraints ought to occur at similar points in the motions, thus constraints were gathered from the input motions at frames which were nearby with respect to the global time parameter u. Furthermore, to qualify as a constraint *match* a set of constraints had to include exactly one from each motion, and the constraint intervals of each motion had to overlap such that their union

formed a single continuous interval. Additionally, where appropriate, rules were used to eliminate certain constraints in the input motions or alternatively, to split them into two shorter constraints by the addition of a gap, prior to identifying a constraint match for inclusion in the registration curve. Having accumulated all constraint matches within a registration curve, each individual match, grouping constraints from every input motion, was merged into a single constraint interval for the synthesised output, by treating the start and end points of the constraint intervals of each input motion as parameters which may themselves be blended.

Conglomeration of the three above-described components into a registration curve involved first constructing the timewarp curve $S(u)$, which was then used to build the alignment curve $A(u)$, and finally to map constraint intervals into a standard time frame to allow constraint match identification. Central to constructing both the timewarp curve and the alignment curve, was a distance function $D(F1, F2)$ which simultaneously determined the distance (similarity) between two frames of motion, and gave the parameter values for such a rigid 2D transformation as would align the frames so as to minimise that distance. The distance function was the same as that employed by Kovar et al., [KGP02] (detailed in Section 4.6).

Having built the registration curve, creating a single frame of blended motion involved four steps:

1. Determining a position on the timewarp curve and the corresponding frames (ie points in time) in the input motions.

2. Aligning the input motion coordinate frames so as to best overlap their character motions and set the final position and orientation of the skeleton on the ground plane.

3. Interpolating the motion frames (ie the poses) based on the blending weights.

4. Determining the constraints on the resulting motion-sequence frame.

The steps were then repeated while moving along the timewarp curve in a manner dependent upon whether the aim was to create a transition going from one motion to another, or an interpolation giving a weighted combination of a number of motions.

Kovar and Gleicher's contribution was a method which packages together already-known techniques, automatically constructing a single unit which once created, did in an equally automatic manner, improve the quality of blended motions and allowed blends that were previously beyond the reach of automatic methods. However, they did require input data to be annotated, a process which may require manual intervention. Furthermore, some, if not all of the individual techniques involved in the construction and application of registration curves, would be used anyway by practitioners in the field where necessary, and this with some degree of automation, as well as, arguably, lower implementation complexity. The novelty offered by registration curves lay in their solid approach to marrying these techniques in an automatic fashion.

While registration curves did expand, as intended, the class of motions which could be successfully blended using *automatic* methods, it should be noted that the extension to the range of blendable motions was limited, and there were cases where registration curves were likely to fail. Their construction assumed that logically related parts of motions look more similar than unrelated parts, but this assumption is not always valid, and, by the authors' [KG03] own admission, in such a case the timewarp curve generated by the presented approach would not be nearly as accurate as manually labelling correspondences.

Constructing a registration curve took 3.43 seconds for two 20-second motions at 30 frames per second (in 2003), but they only needed to be created once and were

then stored for future use. Far more important is the runtime cost of subsequently using these structures, and performance was – rather inconclusively – said to provide "interactive rates". Of itself, this does not indicate real time performance.

Motion capture data used by practitioners or in research is sometimes sourced from large motion libraries. Selection of suitable clips can be a lengthy process, even when the data is categorised by movement type. Without automated methods the data must be painstakingly played back and visually assessed. Specific sought-after motions might only surface after sifting through clips of varied content, and final selection may require repeated comparisons of similar-looking sequences. It is to automate this process that Kovar and Gleicher [KG04] proposed a method for extracting logically similar sequences from large motion databases. Furthermore, from each resulting collection of semantically similar clips, a continuous parameterised *space* of motions was automatically created. Parameterised spaces allow interpolation guided by user-requested parameters (motion properties) which the synthesised motion then honours, such as the target of a karate kick or the elevation of the foot when stepping onto a platform. Extraction and parameterisation, each useful in their own right, are described in succession below.

The extraction of logically similar motions, is made difficult because such motions are not necessarily numerically similar. To identify them as belonging together an iterative multi-step search approach was used. The search first returned matches found numerically similar to an initial *query* motion, and then repeatedly sought motions similar to these matches, iterating until new matches ceased to be found. To enable fast processing of search queries at every step, a *match web* was created in preprocessing – an efficiently searchable representation of all similar motions in the data under consideration.

The evaluation of numerical similarity – required for matchweb construction – took account of time correspondences in the motion sequences, doing so in two distinct ways. Firstly, similarity was considered in respect of *corresponding* frames. These were established by a dynamic programming technique which time-warps the motions to minimise the total distance (ie total dissimilarity) between matched frames, based on the distance metric previously used by the authors in [KGP02]. Secondly, minimising *total* cost does not imply *local* optimality of the time alignment. To require this as well, a threshold was imposed demanding strong local frame correspondences between the two motions along their entire lengths before declaring a match.

Match webs, like dynamic timewarping grids (as used, for example, in [LCR+02, BW95, SO06]), have a row for each frame of one motion sequence and a column for each frame of another, but instead of a single globally-optimum time alignment path the grid shows time correspondences for *all* numerically similar motion segments within the two sequences, albeit after further processing. This final processing step considered only those portions of the time alignments which happened to be locally optimal (in contrast to near-optimal as enforced by the above optimality threshold) and joined them with near-optimal *bridges* to complete the match web. The resulting structure allowed, given a query sequence, a fast lookup returning all similar sequences matching it numerically, and was thus well-suited for multi-step searching.

The second stage in this two-part work of Kovar and Gleicher [KG04], had an entirely different focus, parametric motion space creation for each collection of semantically related motion, as outlined below.

The proper time alignment of extracted clips, required for successful blending, was first established. Specifically, the collection of motions were registered in time through a continuous timewarp curve as used in [KG03], where, given N example motions, each point on the curve is an N-dimensional vector specifying a set of corresponding

frame times whose character poses can be successfully blended. However, the method of [KG03] fails for numerically distant motions even when these are logically similar, as applies, for example, to reaching motions which target very different locations.

To overcome this limitation the timewarp curve was instead created using a *match graph*, in which the matches obtained in a multi-step match web search, as well as the query motion itself, were shown as nodes, with *similar* nodes connected by edges. The match graph, like the multi-step search results it represents, links a query motion to logically related distant motions via intermediary "in between" [KG04] sequences. By allocating a two-motion timewarp curve to each edge (after additional steps not elaborated here), a chain of such structures becomes available between the query motion and any other motion in the graph. Consequently, given a frame in the query motion, the corresponding frame in any other motion could be obtained, by walking the path between them and using consecutive timewarp curves to find corresponding frames from node to node. In this manner, a direct time correspondence was obtained between query and any match in the graph, allowing the construction of a single global timewarp curve encompassing the entire set of logically related motions under consideration.

Parameterised motion spaces enable blending with direct user-access to motion properties, unlike control via blending weights. Mapping blend weights to parameters of the motion they generate is straightforward, but the inverse function, providing blend weights given parameter values is not. Scattered data interpolation is a technique which allows an approximation of the inverse function to be built.

Given a set of discrete example motions, each with associated parameter value(s), a new motion honouring user-specified parameter settings can be blended from a weighted average of nearby example motions, weighted by the distance in parameter space between these motions and the blend being sought. This provides an approximation of the inverse function, mapping parameter values to weightings, which increases in accuracy

the higher the density of example motions in parameter space. The parameter value(s) of the motion blended with these weightings can be precisely established, and stored as a sample bridging parameter and blend space, with a higher density of such samples increasing the accuracy of the inverse function as would a higher density of example motions.

For large motion collections, however, achieving high sample density in parameter space by densely sampling the space of blending weights is not possible, as the latter's high-dimensionality would bring about a prohibitive number of samples. This problem is avoided by selecting, for random points in parameter space, the $d+1$ example motions with the closest parameters, where d is the dimensionality of parameter space (thus selecting, for example, four samples in a 3D parameter space, sufficient samples to form a volume in this space). Then, ignoring all other example motions, the now low-dimensional blend weight space is densely sampled, with corresponding parameter values easily obtained, thus generating new samples in parameter space which become available for use in parameter-to-blend-weight approximations of now greater accuracy. The process is repeated until the accessible region of parameter space is covered, thereby discarding newly created samples which are too close to existing ones, to ensure a more uniform sampling.

At runtime the user supplies parameter(s) describing the desired motion. The k nearest neighbours in densely-sampled parameter space are found, and a weighted average of the blending weights associated with each neighbouring sample is established, whereby computing this average is based on distance in parameter space. This yields the weightings required to blend motion possessing an "accurate approximation" [KG04] of the user-specified motion requirements.

The *idea* of an algorithm able to build parameterised motion spaces from a large dataset may appear enticing. However, the time saved by avoiding manual sequence

selection must be weighed against the considerable time and difficulty associated with implementing the presented approach, which itself includes non-trivial algorithms embedded within it. Furthermore, the method sometimes returned spurious matches, at times it also failed to match logically related motions, and when processing some motion types using unlabelled data "we require users to independently confirm that matches have the correct meaning" [KG04]. Another limitation is that match webs, central to motion extraction, were only demonstrated on ten minutes' worth of data taking 50.2 minutes to create a 76.2 MB structure, and being $\mathcal{O}(n^2)$ in both time and storage space, they impose a limit on dataset size, as admitted by the authors.

Even regardless of the above limitations, exacting practitioners will most likely insist on a significant degree of manual selection and oversight at the very least. For one thing, they may wish to manually choose the *best looking* clip(s) from a collection of motion, which the presented system is unable to do, and may even prefer to oversee the entire motion selection process to ensure all required motions are available and to an appropriate standard of quality. Compatibility must also be visually assessed, as confirmed by the authors [KG04] who state that "motion sets found by our search engine are not guaranteed to be blendable". Though laborious, manual selection of input motions is a small task in the context of developing games or other animated programs, and Heck and Gleicher [HG07] confirm that despite research having presented automated ways of creating the motions in video games, methods used in practice require extensive manual work, thus indicating the choice of practitioners.

The method of [KG04] is appealing, however, and limitations are to be expected given the ambitious goal. The multi-step approach fares better than a single search with a lower similarity threshold. While seemingly simple, this cleverly exploits the tendency for "in-between" motions to exist between those that are logically similar but numerically distant, such intermediaries being less common between motions which are

both numerically and semantically distant. This does imply, however, that successful operation requires that the dataset contain those intermediate matches. Similarly, few "in-between" matches should be present between motions which are logically distant, something increasingly unlikely as the database grows in size and consequently in variety.

4.8 Motion Editing

Witkin and Popović [WP95] presented *motion warping*, a technique for editing motion obtained via motion capture or keyframing. It involved modifying one or more existing poses to define keyframe-like constraints at certain frames, and interpolating the *changes* thereby applied to the motion parameter curves over the span of frames between such keyframe constraints. (For the purpose of interpolation, the endpoints of the motion were also treated as keyframe constraints). The difference between motion warping and ordinary keyframing is that it is not the actual poses which are interpolated but the applied changes.

Motion warping involves either, or both, of two possible types of deformation, one in space and one in time. In each case, Witkin and Popović performed the associated interpolation using Cardinal splines. Spatial warping produces a curve given by

$$\theta'(t) = a(t)\theta(t) + b(t) \tag{4.1}$$

where $\theta(t)$ and $\theta'(t)$ are the original and warped values of the motion parameter curve respectively, and $a(t)$ and $b(t)$ are user-specified scaling and offset functions. Timewarping used timewarp constraint pairs (t'_j, t_j), where t'_j gave the time to which the motion originally at time t_j should be displaced. The timewarp function, which passed through the time constraints and interpolated between them, was written

$$t = g(t') \tag{4.2}$$

Parameter t was expressed in terms of t', instead of the other way round, as timewarping involved finding an actual value of $\theta'(t)$ in the untimewarped curve to be used at a known time t' in the synthesised timewarped sequence.

While timewarping of itself involves no blending, Witkin and Popović [WP95] also used the process to join captured motion clips by deforming the start of one, and the end of the other, prior to blending. However, while certainly useful, the presented timewarping was a rudimentary one, and should not be confused with the elaborate synchronisation offered by *dynamic* timewarping (as used, for example, by [KG04, LCR+02, SO06]).

Examples of motion warping included an ordinary walk transformed into one which stepped onto a block, or leant over to walk under a doorway, as well as on-the-fly motion synthesis of a tennis forehand shot warped to match the ball's trajectory.

Motion warping is intuitive due to its simplicity of concept, but also because it can be used with a keyframing interface as did Witkin and Popović. Furthermore, when working with motion captured data, the smooth deformations allow the motion to retain the fine structure of its high frequency content, preserving its naturalness and realistic appearance as long as extreme warps are avoided. The technique acquires much of its value from the fact that only a small number of frames (the authors used 1–5) need to be modified, making such work far less than that of creating new motions from scratch.

Limitations include the difficulty of enforcing geometric constraints between motion-warping keyframes, and the fact that the process is a purely geometric one and does not incorporate any deep understanding of a character's structure or the dynamics of its motion.

The work of Bruderlin and Williams [BW95] was partly discussed in Section 4.7 which considered motion interpolation. Of the four image and signal processing tech-

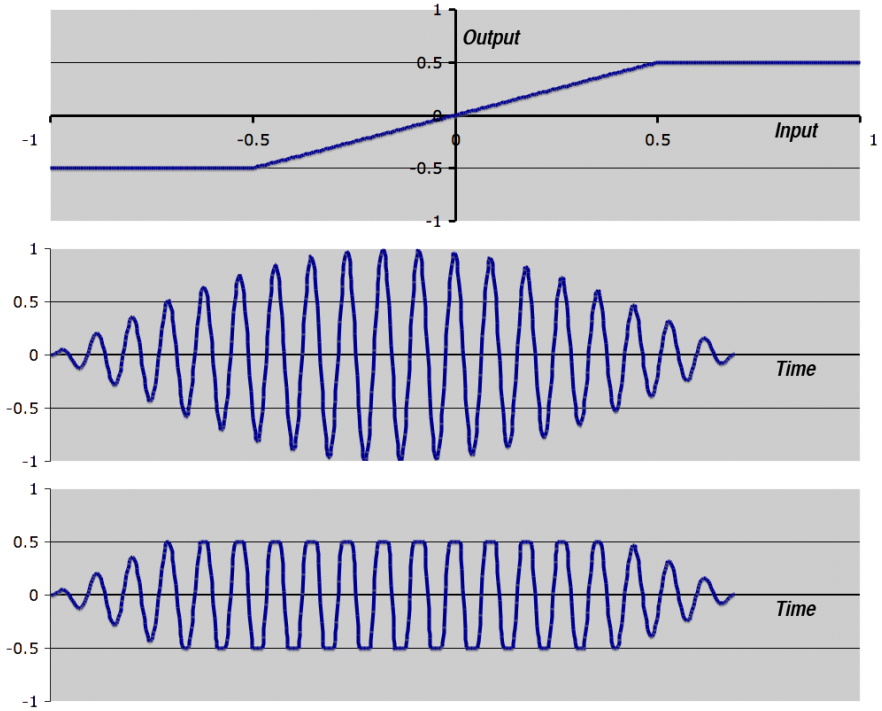

Figure 4.1: Motion waveshaping, as used by Bruderlin and Williams [BW95] for processing motion data. The shaping function (top) defines the impact (in this case clipping) on the input signal (centre) which it re-forms to create the output (bottom).

niques presented in [BW95], however, two, motion waveshaping and motion displacement mapping, were motion *editing* procedures, and accordingly, are now described below.

Motion waveshaping – based on digital waveshaping used in sound synthesis – employs a waveshaping function to define how to modify an input signal, and thereby produce an output. With references to Figure 4.1, top, if the horizontal axis quantifies possible input signal values, and the vertical axis those for the output signal, then the relationship between them is given by the waveshaping function, of which an example is plotted on those axes. The effect on a specific input signal, like that shown in the middle image, is the output shown below it.

Bruderlin and Williams illustrated waveshaping of motion data by using it to limit the joint angles for a motion sequence of an articulated figure waving. A simple waveshaping function made of three linear segments merely clipped the motion applying hard joint limits, but a more gradual smooth-curved function provided soft limits giving the motion a more pleasing appearance. Their implementation of the shaping function itself was based on interpolating cubic splines to produce a smooth shaping function.

The final technique presented by Bruderlin and Williams [BW95] is motion displacement mapping, a procedure apparently identical to the *motion warping* of Witkin and Popović [WP95] (above), when selecting, for the latter, the option to warp in space instead of time, and to use an offset function instead of a scaling function. The pose of an articulated figure is changed at a few keyframes, and the displacement resulting at those poses joined with a cubic spline curve, thus providing a smoothly varying *displacement* over a sequence of frames. The displacement is then added to the original motion, which is thus modified in a smooth manner while honouring the constraints defined by the newly defined keyframes.

An example of displacement mapping illustrated by Bruderlin and Williams, is the prevention of joint limit violations, as well as the correction of a case where the character's feet did not make proper contact with the ground.

4.9 Statistical Methods

The character animation literature has seen a proliferation of statistical-methods since the turn of the century. Statistical models are able to encapsulate information learned from source data and have become, in recent years, "a tool of choice for enclosing the motion specific information"[CC10]. Such methods mostly handle the spatial correlations and dynamical information contained within motions, and, to a degree, are able to generalise, even, in some cases, based on a single source motion. The models frequently

take the form of a collection of mathematical equations, or functions, that describe human motion with a limited number of parameters and associated probability distributions [LWC+11]. Models are often compact and can be used to synthesise animation that is not in the captured data [WMC11]. Capture- and video-based statistical models have been used to synthesise motion perturbed by random yet statistically reasonable variations [PB00], for example-based keyframing [LWS02], for motion authoring via direct manipulation interfaces, sketching and performance animation [CH07], for accelerating such authoring to interactive rates [MCC09], and for the incorporation of physics-based methods [WMC11], to mention but few. This representative sample is now detailed below.

Early work by Pullen and Bregler [PB00] was based on video recordings of a wallaby hopping on a treadmill. The subject was chosen because it hops with its legs together providing locomotion which can be represented in two dimensions, using only three angles (hip, knee and ankle) and two translations (x and y). Eight hops of input video were first transformed to DOF values by extending an established vision-based tracking system [BM00]. This many cycles were necessary to sample natural fluctuations in the motion, as the goal was to generate a set of DOF motion curves which, while not intended to copy the original data, were to preserve its characteristic features and reflect its variability.

Adding noise to motion to achieve subtle variations as seen in real-life was not new, as demonstrated by Perlin and Goldberg's [PG96] earlier inclusion of Perlin noise [Per85] during synthesis. The statistical approach of Pullen and Bregler [PB00], however, achieved greater realism by inserting variations previously extracted from, and hence aptly befitting, the source data itself. The resulting animation thus not only resembled the original data, but its random variations bore statistical resemblance too.

For each DOF the training data was separated into frequency bands or "levels" [PB00] via wavelet decomposition, similar to the multi-resolution layers in the bandpass pyramid of Bruderlin and Williams [BW95], (Section 4.7, above). During synthesis, the bands were used to create motion one level at a time.

The method accounted for strong correlations within the source data. The data value at a certain instant in time, for a specific joint and wavelet level, depended upon the value at nearby times, for that or for other joints, both at that or other wavelet levels. These dependencies were represented with kernel-based multivariate probability densities, with the kernel chosen large enough to generalise the data, but small enough to retain the shape of the distribution. The data was synthesised by a process which included sampling from these densities and thus from the motion data correlations therein.

Although no demonstration video was available, the artificial data synthesised with the method of Pullen and Bregler was said to look "as if it could have been real data" [PB00] despite, as intended, subsequent runs producing different results, thus confirming the realistic effect of the introduced fluctuations. However, results were implemented on "computer models of a creature with limb lengths that are proportional to that of the wallaby" which in fact bore no significant resemblance to a wallaby, but were quite imaginary characters. Such characters, as applied to the physics-based Luxo lamp [Pix86] of [WK88], are not only relatively simple, but have no analogues in real life to live up to, which, in the context of human motion synthesis, inevitably moderates the significance of such work. Furthermore, output motion was fixed in that no opportunity existed for user control such as character navigation, the work was limited to five DOFs and two-dimensional input data, and footskate was evidenced (receiving rudimentary correction). Nevertheless, the demonstrated principle represents a useful contribution. It illustrated the potential of statistical methods to generate new motion similar to

source data, allowed extended motion sequences with natural, subtle variations to be synthesised from limited input data, while furthermore providing a means to reduce the clone effect [MLD+08] in crowd scenes.

Li et al. [LWS02], working in three-dimensions, presented *motion textures*, two-level statistical models in which the lower level comprises a set of motion *textons*, and the higher describes their distribution. By texton, a term introduced two decades earlier in the context of texture perception [Jul81], was meant a repeated primitive motion pattern inherent in complex human motion, such as spinning or hopping while dancing – this being the motion type used by [LWS02] to demonstrate their method.

Motion textons were modelled by a linear dynamic system (LDS) to capture the local dynamics of the motion (whereby dynamics as used by [LWS02] refers to the *kinetic* aspects of the motion, not to force-based physical simulation as traditionally meant in the literature). At a higher level the nonlinear global dynamics of complex human motion were regarded as stochastic, with the texton distribution showing the computed transition likelihood between any pair of textons, for use "as the distance metric for determining whether those textons may be successfully stitched together" [FHP07]. Having learned textons and their distribution from captured data, new motion could be synthesised, and interactively edited, both at the texton and the distribution level.

Texton synthesis used the learned LDS as well as sampled noise, and in simple form tended to deviate from the original motion it was modelling as time progressed. However, all textons were constrained by an initial state, which in the context of motion textures refers to the first two frames, and if as a *further* constraint the textons last two frames were made to match the original motion, then all intermediate frames were automatically adjusted in such a manner as to produce animation, throughout the texton, with dynamics perceptually similar to those of original motion. More generally, interactive texton editing was possible anywhere within it, requiring minimal

pose adjustments, while noise added fine detail, with noise variability ensuring unique output even when based on an already-used texton.

With adjacent textons, setting the start frames of the second to match the end constraints of the first produced continuous transitions between them, allowing smooth concatenation of multiple textons. As well as user-selection of texton sequence, infinitely long chains of textons could be generated by randomly sampling the texton distribution, and, additionally, automatic texton path generation was able to link user-specified start and end frames.

Motion textures performed sequence rearrangement like motion graphs but as a significant improvement they also included a degree of lower-level motion editing. Although 20 minutes of dance motion took approximately 4 hours to learn (in results given 2002), synthesising textons (of 1 to 2.9 seconds duration) at runtime took only 25 to 35 ms, allowing motion synthesis in real time (unlike, ironically, more recent methods described below). By remaining statistically similar to, yet visually different from, the captured data, the method was said to generate realistic motion (though no video was available for examination). Limitations of motion textures included synthesised motion potentially lacking global variations given limited training data, no guarantee of physical correctness, no interaction with the environment, and contamination of texton quality should frame editing deviate too much from the original pose.

Chai and Hodgins [CH07] used a statistically based optimisation approach with some similarity to spacetime constraints, the physically based optimisation method of Witkin and Kass [WK88] (Section 3.7). Both consider the task as one of trajectory optimisation and compute the entire motion sequence simultaneously, but that of Chai and Hodgins, instead of using physics constraints to ensure physically valid motion, relied on statistical dynamic models to constrain the motion to the space of plausible human motions. The models were enforced as motion priors learned automatically

from the motion capture data, with the combination of motion priors and user-defined constraints forming the trajectory optimisation problem and providing sufficient information to produce motion which honours these constraints while retaining a natural appearance.

Motion capture data was preprocessed using principle component analysis (PCA) [Smi02] so as to reside in a reduced subspace, allowing a low-dimensional representation of the character configuration and hence the learning of an efficient and low-dimensional representation of human motion. As with a physical model, a statistical dynamic model can, given an initial system state, create animation sequentially by choosing an appropriate value for the control input vector, analogous to joint torques applied in a physical model. The lower dimensionality of control input compared to dynamics models, however, allows the optimisation to potentially achieve faster convergence while being less likely to encounter local minima [CH07].

The user could specify a variety of kinematic constraints, including keyframes, individual joint angle settings and positions and orientations for particular points on the character. The constraints could exist at isolated points in time or throughout the motion sequence, and could be fine-tuned incrementally. To a limited extent, the system could generalise and create motion not present in the motion capture database, being able, for example, to synthesise walking up a slope from a normal walking sequence. It was even flexible enough to animate characters whose skeleton was significantly different from that in the source data. It did, however, impose a notable restriction on the input data, as evidenced by it generating walking motion of lower quality when using a prior learned from a general locomotion database than from a walking-only source motion. Another limitation lay in constraint specification being *demanded*, not optional, with, for example, footskate present in a two-keyframes sequence, which only improved as further keyframes were added and numerical error dropped. Furthermore,

processing was expensive, with a 100-frame sequence taking 10–20 seconds to compute (in results published 2007), making it prohibitively expensive for games or real time simulation. Unsuitable also due to the interface, which included sketching the desired path of specific joints, and even performance animation (Section 4.1) [SLSG01, Stu98b], with Chai and Hodgins [CH07] acknowledging the need for future work to "design intuitive interfaces that allow the user to specify spatial-temporal constraints quickly and easily". This should be seen in the context of the methods implemented and demonstrated for this thesis, which, as detailed in Chapters 5, 6 and 7, synthesise motion markedly faster than real time, provide animation with immediate character response to user input, which can walk, run and more, footskate-free, without the necessary specification of constraints, and which use a simple interface allowing direct navigation in a virtual environment.

The method of Chai and Hodgins [CH07] appears well-suited to motion authoring for later playback, as long as the requirement for keyframes specification is not felt restrictive. It complements physically-based optimisation, is able to synthesise slow or stylised motions which are unsuitable for such physics-based approaches, and avoids the difficulty of building anatomically correct physical models.

Min et al. [MCC09] devised a statistical approach based on their 'deformable motion model' and generated walking, jumping, bowling and golf-swing motions from a large set of short, annotated, pre-captured clips. Geometric and timing information was extracted from the input data, and then adjusted continuously during optimisation by "deforming" two corresponding motion-model parameters, thereby controlling the animation so as to match a variety of user-specified constraint types.

Decomposition into kinematic and timing data was by a semi-automatic process, in part based on [KG03] (Section 4.7 above). Input clips were 'registered' to a reference motion by first timewarping (as in [BW95, KG03, SO06] above) to obtain time

warping functions, and then using these functions to warp each input motion into a new sequence, resulting in two datasets: geometric data in timewarped semantically registered input motions, and timing data in the corresponding timewarping functions (expressed as vectors). The geometric data was reduced in dimensionality using PCA and statistical analysis techniques, then applied to construct a deformable geometric model, deformable by modifying weightings in a vector which controlled the influence of motion components within the model, thus bearing a degree of resemblance to interpolation methods (Section 4.7). A similar process was applied to the timing data, with additional steps to avoid PCA violating the required strict monotonicity of the timewarping function data. Additionally, a joint probability distribution function was learnt from the two datasets, the distribution modelling the correlation between the geometric and timing variations. The deformable motion models, fed from a sampling of the distribution function, formed a generative model able to synthesise an infinite number of motion instances, albeit constrained by the probability distribution function, to stay close to the training examples, and from these motions were sought those that best matched the user-defined constraints.

The method of Min et al. [MCC09] ran (in 2009) approximately 100 times faster than that of Chai and Hodgins [CH07] before them, and improved on previous statistical approaches unable to support varied user constraints at interactive frame rates, "which significantly limits the impact and use of statistical motion models in graphics applications" [MCC09]. Furthermore, information on motion structure such as left-toe-down frame details and environmental contact events like foot-contact constraints were encoded in the model which used this information to avoid visual artefacts like footskate. It thus improved on Chai and Hodgins [CH07] whose continuous dynamic model and optimisation framework often produced poor results when using a sparse set of constraints.

The research goal of [MCC09] was "to develop an interactive system that allows novices to create realistic human animation quickly and easily" in effect identical to that of Witkin and Kass [WK88] whose spacetime constraint approach was to replace simple keyframing with a system which does "much of the work that the animator would otherwise be required to do, and that only a skilled animator can do". The method of [MCC09] generally succeeded in its goal as a motion authoring tool, providing both direct manipulation and sketching interfaces, and running at unspecified "interactive rates" (to be seen as distinct from real time animation with its greater demands). Drawbacks, however, include the inability to create longer sequences without first making shorter ones to be stitched together, the need for input data to be annotated, and for it to comprise large sets of structurally similar but distinctive motion examples. The quality of the animation depended – not surprisingly – on that of the prior knowledge embedded during training. It was also subject to appropriate constraints being specified by the user which seems somewhat in conflict, however, with the stated goal of quick and easy synthesis by novice users.

Wei et al. [WMC11] went a step further, constructing a statistical model which combined statistical motion priors with *physical* constraints. The resulting model generated animations which are not only natural-looking and which honour user-specified constraints, as had Chai and Hodgins [CH07] and Min et al. [MCC09], but which were also physically realistic. Unlike previous statistical models, theirs drew on Newtonian dynamics and contact mechanics to react to external forces and to physical quantities, like the mass and inertia of the rigid body segments used to approximate human motion. For example, when simulating walking while wearing a 2.5 kg shoe on the left foot, the character automatically lent to the right to offset the additional weight and maintain balance. Appropriate gait adjustments were also evidenced when simulating walking on

slippery surfaces (with reduced friction coefficient), or in lower gravity (set to $1.62\,\mathrm{m/s^2}$ as on the moon), or during resistance running (running against a restraining force).

Statistical modelling devoid of physics-constraints may lead to implausible visual artefacts like jerkiness, footskate or a character out of balance. Similarly, physically-based human motion optimisation taking no account of the subspace of natural motions faces a mathematically ill-posed problem, since there are many physically valid ways a character can perform a task such as walking, but not all will look natural. In the methods of [WK88] and [RGBC96] these are narrowed down to a subset of more natural looking motions, by a heuristic of "minimizing the power consumed by the muscles" [WK88], or the principle that "motion that minimizes energy looks natural" [RGBC96]. However, these strategies, while suitable for more dynamic motion, are less so for those of low energy or highly stylised human movements. These problems are avoided by using statistical priors which – having learned an appropriate performance criterion – enforce naturalness [WMC11].

The physics-based statistical framework of Wei et al. [WMC11] has similar limitations to the kinematic-only statistical method of Chai and Hodgins [CH07], in that constraints such as keyframes and footplants had to be defined by the user, and stylised walking had to be defined by the guidance of trajectory constraints, such tasks being "not trivial for a novice user" [WMC11]. This limitation seems severe, with the difficulty in creating trajectory and contact constraints emphasised by them instead being "either directly modified from reference motion data or rotoscoped from video streams" [WMC11]. In this *clearly limited* sense, little or no progress is to be seen compared to the work of Witkin and Kass 23 years earlier (1988), having achieved its aim of limited user-input on the level of start here and stop there, and a "how" criterion such as "don't waste energy" [WK88]. Furthermore, the method of Wei et al. [WMC11] (2011), is – not surprisingly given the task performed – exceedingly slow despite re-

cent hardware, synthesising (learning times aside) a 45 second walking sequence in 51 minutes, falling far short of the requirements of real time simulation like video games. These limitations aside, however, the achievement seems remarkable, generalising input motion to synthesise – given similar source data – new motion which not only honours user-requirements, but conforms to physical correctness while also remaining constrained to the subspace of natural motions.

4.10 Motion Retargeting

When the size or proportions of a synthetic character are different from those of the human performer, motion capture data cannot be applied directly to the target skeleton, as doing so may cause artefacts such as footskate, ground penetration, or reaching actions which fail to meet their goal [Gle98, MLD+08, PSS02]. The reason is made apparent by considering two animated figures with segments of different length. If both figures enact the same joint angle sequences, then, due to differing segment lengths, the end effector trajectories will be different between skeletons. Thus, for example, motion data befitting one skeleton perfectly, will, if its root translation and angle data is applied to another, result in feet following modified trajectories which no longer match the (unaltered) root translation data, and hence in feet that slide along the ground. Retargeting adapts the motion befitting one character for use on another [LMT07, SLSG01], while, as much as possible, avoiding such artefacts.

Motion retargeting was the focus of Gleicher [Gle98], whose work was primarily limited to target characters which had identical skeletal structures to those of the actor, but different segment lengths. (Retargeting to differently *structured* characters, thus to those with different limb connectivity and degrees of freedom, was, as an adjunct, merely touched upon). Retargeting was based on spacetime optimisation (Section 3.7) which considers the entire motion simultaneously, providing a global view which ensured

choices made regarding one frame were based not only on that one, but also on those which preceded and followed. Thus, for example, by effectively looking ahead, each footplant in a series of steps took account of the final destination, providing superior results to approaches that consider each frame independently.

Emphasised in [Gle98] are the choices, limitations and compromise underlying retargeting. Should a tall person's gait, applied to a child, walk in the style of an adult or a child? And if a high-level quality such as 'grace' should be preserved, how should it first be defined mathematically? Furthermore, "good adaptations preserve important aspects of the motion by altering less important ones" [Gle98], so prioritising of qualities is needed.

To reduce such difficulties Gleicher [Gle98] opted for a pragmatic approach, avoiding any explicit definition of motion qualities except for lower-level readily encodable features like the need for walking feet to touch the ground, a strategy said to "generally preserve the desirable characteristics of a motion". Retargeting thus had the two-fold aim of retaining simple features encoded as constraints (in the wake of which were preserved additional, undefined, desirable qualities), and of avoiding uncharacteristic changes to the motion signal itself, as further described below.

Maintaining constraints. Constraints were maintained by the spacetime framework, which poses a problem over the full sequence duration, seeking that motion which best fits the specified constraints. Constraints were geometric, setting out, for example, that two points on a character be a certain distance apart (useful for carrying a fixed-sized object), or that a parameter value be inside a limited range (useful for joint limits). They did not include physical laws as found in the original method of [WK88], however.

Avoiding signal degradation. The objective function represented not the target

motion, but instead the displacement signal which, when added to the original motion, produced the retargeted motion. To preserve the qualities of the original motion, the magnitude of the changes between source and target frames was minimised during optimisation. Furthermore, inverse-kinematics-based retargeting, a quite different approach which poses the problem on a single frame only, was shown to generate discontinuities of motion, "jerkiness" [Gle98], distorting the high frequency content of the signal. To avoid any such changes between source and target motions, the displacement signal was further restricted during optimisation, limiting high frequency content by representing it with a cubic B-spline, whose number of control points was intentionally limited.

The constrained optimisation process thus addressed both parts of the retargeting problem, the maintaining of constraints and the minimising of changes vis-à-vis the original motion. The spacing of the displacement curve's B-spline control points determined the frequency content of the motion adaptation. Although not defined explicitly as such, the capped frequency response of the retargeting process was then itself a constraint and one which limited the optimisation such that there might be no solution to the given constraints. The objective thus only attempted to *minimise* the constraint residual.

Above-described is the core of Gleicher's retargeting method [Gle98], which also included, amongst other things, methods for retargeting to morphing characters whose segment lengths were changing, and methods for characters whose skeletal structure was different from that originally used to create the motion data.

Results were numerous and varied, reflecting the difficulty of motion retargeting, in which "our approach makes many sacrifices to achieve practicality" [Gle98]. One limitation is the absence of physics constraints, which can lead to unrealistic poses, and

another is the method's complexity, whereby, for example, even when using pre-defined constraint types "augmenting our characters and motions with constraints does require some additional work", and defining *new* types of constraints required programming. Furthermore, the best spacing for the B-spline control points (viz. every 2, 4 or 8 frames) required confirmation by visual examination of the retargeted motion, as an automatically made choice based on the frequency decomposition of the original motion, while included, was not infallible. Processing times were given only for the optimisation itself, and depended heavily on factors such as the desired stopping tolerance and the choice of solver used, but to give just one example it was under 10 seconds to process a walking motion of 3.7 seconds. Also retargeted was a ladder-climbing motion for which, however, the result did in part look "slightly unnatural" [Gle98], and a couple swing dancing, where, through being connected by holding hands, the resizing of one character required simultaneous retargeting of each character's dance motion.

While the presented method of Gleicher [Gle98] is neither user-friendly nor a complete and polished solution, it is surely, in light of the size of the challenge, a notable contribution.

Gleicher later addressed the issue of *real time* retargeting, as co-author in the work of Shin et al. [SLSG01], whose context was computer puppetry [SLSG01, Stu98b, Tra94]. Digital puppetry or performance animation, maps the motion of a performer to an animated character in an online real time manner, thereby reflecting the relation between puppeteer and traditional puppet. Shin et al. [SLSG01] worked only with humanoid virtual characters of identical connectivity and degrees of freedom as the skeletal structure underlying the live capture data – data which (in this case) took the form of joint angles. The size and proportions of the character differed from those of the actor, however, so the problem itself was similar to the motion capture file-based retargeting work of [Gle98] described above, albeit with the immediacy of live

performance adding to the challenge. Coining the notion of the dynamic *importance* of an end effector, used to determine which end effector trajectories in the original performance the system should endeavour to keep while retargeting, the approach of Shin et al. [SLSG01] exemplified *importance-based* retargeting, which, more generally, prioritises aspects of the source motion, thereby specifying those it is desirable to retain and others to be allowed to change.

The method comprised three steps, as described below.

Source data filtering. Captured motion data is generally noisy, and was especially so with regard to sensors required by computer puppetry. Although optical systems can currently stream and map or retarget in real time [Mota, Vic], it was magnetic systems that were widely used for real time performance in 2001 [Stu94, Fur99], and these suffered from interference and always exhibited jitter [WF02, SLSG01]. Shin et al. [SLSG01] thus first removed capture technology artefacts, and did so with a real time smoothing technique that was more efficient than previous methods.

Dynamic importance measure. To determine the importance value of every end effector in relation to its environment, the filtered motion was evaluated with the dynamic importance metric. The importance of an end effector was deemed high if it was interacting with some object (such as the floor), or if it was about to do so. Importance thus increased both with proximity to objects in the environment, and with the rate of change of proximity. The values thus computed were weightings, establishing the degree to which end effector trajectories in the original performance were to be reproduced in the target motion, as opposed to merely copying joint angles to the limb in question.

Inverse kinematics. A fast IK solver, incorporating both analytic and numerical methods, computed the target pose in real time, recreating as many as possible of those aspects deemed important in the source motion. Furthermore, as the notion of importance took account of whether an end effector was heading to interact with an object in near future, the solver acquired a degree of look-ahead capability helping to reduce any jerkiness of motion, in contrast with the motion discontinuities illustrated in [Gle98] as otherwise being associated with IK-based retargeting.

The inverse kinematics algorithm was a hybrid process, combining inexpensive closed-form analytic methods with more costly numerical optimisation whereby the latter, however, was used only when strictly necessary. The process comprised the three stages below.

Root offset. An attempt was made to satisfy the constraints as much as possible by translating the root position. This step broke down into two phases. To clarify the first, the trivial case of a single end effector and its specified goal position is now considered, which illustrates how it is the vector between them that defines the translation of the root which would allow the constraint to be fully met. Shin et al. [SLSG01] had to contend with the *multiple* end effectors, however, and a weighted average of the displacement vectors was used to give the root offset, with the importance values providing the weightings. This initial root offset was then further refined in a second phase which aimed to place the root so all goals were reachable. The required adjustments for this were based on consideration of the reach of the arms and legs as determined by computational geometry, while, as before, again accounting for the importance of the end effectors.

Body posture adjustment. Should the refined translation of the root not succeed in making all goals reachable, the body posture itself – consisting of root position, root orientation and the posture of the upper body – was optimised, minimising an objective function whose value was greater when postures made goals harder to reach. Here too importance weightings played a role, ranging from the highest values attempting to enforce goal reachability and the meeting of constraints, to zero-importance values for which end effector positions were simply ignored in favour of joint angles from the original performance. While ordinarily prohibitively expensive for real time use, confining optimisation to the upper body lessened the cost, and the process, importantly, was rarely needed, requiring few iterations when it was.

Limb positioning. Having set the body posture, that of the limbs was computed, efficiently, with an analytic IK solver specialised for articulated human-like characters. The calculated limb postures were then blended with those in the captured data, using, as weightings, the importance values of the end effectors. In this way, those end effectors of high importance retained the positional constraints required by their close association with the environment, and limbs whose end effectors were unimportant and thus without constraints to maintain, took on the joint angles found in the source data, fulfilling a secondary aim of avoiding degradation of the original motion.

Results included successful performance animation of virtual characters on KBS, the national Korean television network. One character, a virtual news reporter, gave interim election results thereby showcasing the retargeting method's suitability for time-critical situations. Further tests with different characters returned total retargeting times, excluding rendering, of 5.4 ms per frame on average (in 2001), for walking, throwing,

jumping, dancing and handstand motions, again showing the method to qualify as 'real time'. Although no video was provided still images suggest motion quality to be, as expected, less continuous than that obtained with the all-out spacetime method of [Gle98] (above), but otherwise rather similar, and achieving this, in real time, is where the contribution lies. A weakness of the importance-based approach of [SLSG01], though, is that "it requires much time to compute"[SLSG01] the distances between character end effectors and objects in the environment, and that "to achieve a real time performance, we need to minimize the number of possible objects that interact with each end effector in accordance with an animation context". Further drawbacks are interpenetration between end effectors themselves, or end effectors and body, and that as with the method of [Gle98], no algorithmic feature is included which incorporates movements idiosyncratic to the target character.

4.11 Cyclification

Many human motions such as walking, running or swimming are cyclic in nature, and longer sequences can be generated by the concatenation of individual motion cycles. Locomotion cycles enacted by real-life actors, as underlies motion captured sequences, will, however, always exhibit endpoints which do not fully match up. This generates visible discontinuities – jerkiness – as well as emphasising that the animation repeats the same cycle. Cyclification is the process of adjusting a motion cycle, and most notably its endpoints, such that concatenation yields smooth-looking animation [AMH03, RGBC96, PB00].

Three simple methods were presented by Ahmed et al. [AMH03]. One was for use when more than a single cycle of motion data was available. It involved replacing one end of the motion cycle with a blend created by merging the frames being replaced

with others taken from a cycle in the motion data immediately preceding or following that being smoothed.

The other two methods required for their operation only the cycle being cyclified. Of these methods, one was concerned with motions for which the second half of a cycle resembles the first half reversed, such as kicking motions. The other was for activities like walking and running, where the second half-cycle resembles a mirror image of the first. Cyclified motion was created by reversing, or alternatively mirroring the data of selected half cycles, and blending these amongst themselves, or with untouched data from the original cycle, and then concatenating the resulting blends either amongst themselves or with unaltered original data from the motion cycle.

In all three methods, the blending operations used weightings controlled in a such a manner that the motion cycle finally resulting from the procedures outlined above had endpoints which match up without geometric discontinuity.

4.12 Footskate Correction

Motion capture files created by mapping raw data to a skeleton can be directly used to build and animate that skeleton, as both its dimensions and DOF trajectories are written within them. Animators using such files, however, may wish to employ a skeleton with different proportions, requiring retargeting of the motion data to fit the new skeleton. While not discussing retargeting itself, Lyard and Magnenat-Thalmann [LMT07] addressed the issue of *footskate* generated as a side-effect of retargeting, specifically that created by retargeting after a user selects from a database of 3D bodies in virtual reality applications, or, alternatively, where the system generates entirely new bodies following user-specification. Footskate is the condition where feet slide along the ground when they should remain planted, and is a common problem in animation synthesis

[GBT04b, GBT06, CH07]. It is likely in this wider context that the corrective measure of [LMT07] outlined below can be applied.

For increased accuracy, Lyard and Magnenat-Thalmann took account of the character's skin, thus of the actual point of contact with the terrain, instead of the skeleton. Also, rather than repositioning the feet to comply with defined constraints and using inverse kinematics to adjust the legs as did, for example, Glardon et al. [GBT06], the correction was instead applied only to the root node, thus translating the entire skeleton to counteract footskate, while, however, modifying its path quite significantly.

The method first detected which foot should be planted by comparing the root translation in a given time step to the change in world-coordinate root position relative to certain vertices in the mesh of the soles of the feet during that time step. Having established the planted foot, the vertex with the least world-coordinate motion was chosen as "the static one" [LMT07]. Considering only one vertex static may be unrealistic but this vertex did (often) change from frame to frame during the walk cycle, and such changing reflects reality where the heel is seen to touch the ground at the start of the contact phase, moving to the ball of the foot at its end.

The precise time at which the static vertex changed when going from one frame to the next was computed, and was in turn used to obtain an accurate measure of the frame-to-frame root translation relative to the deemed-static foot vertices under consideration, thus of that root translation which would eliminate the unwanted footskate. The method thus intentionally overwrote the root translation with another, introducing drift in the character's position to correct the footskate present in the original animation. The correction was limited to the floor plane, as found inaccurate in respect of vertical translation. A separate vertical adjustment was then applied to remedy improper elevation of the foot.

Disappointingly, while the above method did reduce footskate it did not do so entirely, with the authors stating "the feet may well slide a bit between two frames". It was also limited to walking motions only. In comparison, the work presented in Chapters 5, 6 and 7 uses foot constraints to remove all footskate in walking, running and other motions.

Footskate correction is also addressed by the inverse kinematics-based method of Kovar et al. [KSG02], "useful because it increases the utility of other editing operations which destroy footplants", and the statistical method of [MCC09] (Section 4.9) which "can be applied for [...] detecting and removing foot-sliding artifacts of input walking sequences".

4.13 Clone Perception

As stated in Section 4.1, one use of motion capture is the populating of a virtual environment with crowds of people, by reusing a limited amount of motion capture data. Rachel McDonnell et al., [MLD+08], investigated the extent to which clones of characters can be used in crowd simulations, without being perceived as such. Cloning allows a reduction in the computational expense associated with simulating large numbers of characters, but can impinge upon the perceived heterogeneity of crowds. Their work dealt with both appearance clones and motion clones, whereby the former employ the same character although colour and motion may be different, while the latter refers to gait duplication and is the type most relevant to research in skeletal motion synthesis.

Many of the results of [MLD+08] relate merely to character appearance clones, but the key findings relevant to character motion are listed below.

- Motion clones are hard to detect, harder than detecting appearance clones.

- The perception of cloned motion is quite independent of the appearance of the characters (including any applied colour modulation, or the actual model of person used).
- Captured motion should be free of artefacts, such as a limp, as this makes recognition of cloned motion easier. Similarly, clones of distinctive motions such as very ladylike ones are easier to detect.
- The detection of appearance clones is affected by character motion, so in this manner motion can indirectly affect the perception of cloning. Appearance clones are harder to detect when motion is out of step, or when characters are given random motions instead of motions characteristic of the type of person represented by the model.

Thus the effect of motion on clone detection appears to be a satisfyingly simple one, in which duplicate motions do not stand out prominently, and motion can even be used to reduce already-existing clone recognition rates.

Quantitative results were also given. These tabulated the number of clones (of both types) that would, on average, be imperceptible when viewed for specified time durations (5 sec, 10 sec etc). Caution is due, however, as the thresholds applicable to different situations may vary.

Although the paper does not propose any new techniques, the presented results can be used for fine-tuning the cost/realism balance for simulated crowd scenes to enhance their visual appeal.

4.14 Conclusion

The Fleischer brothers knew that rotoscoping of itself produced smooth yet sterile motion, so used artists for their most famous cartoon characters like Popeye. In a

break from this trend, the hero in Gulliver's Travels was rotoscoped and subsequently found by many critics to be lifeless and uninspired, [CBC+97].

Unsurprisingly, the introduction of motion capture to computerised animation was not welcomed by all, 'the devil's rotoscope' being a favourite term among traditional animators [BD09, Fur99, CBC+97]. Yet its use is now widespread, with motion capture seen as bestowing upon synthesised motion, a natural, and hence life-like quality, which the methods of previous chapters could not.

Realistically-moving characters benefit the motion picture industry in many a way, creating convincing characters be they human or imaginary-looking. A hero's daring moves may not even be those of a stuntman, having instead taken place in the comfort of a studio, while a depicted throng of people perhaps never existed. But such techniques are potentially simple, as they may not require adapting captured motion to a new environment, and to circumstances different from those present while recording. In contrast, working with capture data to create new motion is frequently required in games and other virtual environments (Chapter 1, Section 1.1), and methods to achieve this have been described in this chapter.

It was shown that playing back carefully joined sequences from a motion database, can be an effective, sometimes very efficient way of creating motion, [LCL06]. The main challenge is finding a way to rapidly chain together clips from a large – and thus potentially higher quality – database of input motion at runtime. An alternative method, as in [UAT95, PL06, MLD10], and in one of several approaches given in [BW95], was shown to be the blending of motions to create one which (bar fully one-sided blends) is not, at any given instant in time, a copy of any of the input motions. This allows a synthesised motion to exhibit subtle variations not found in the input data. Fat graphs [SO06] and parametric motion graphs [HG07], interestingly, offered a hybrid approach combining both of these disparate methods. More recently, statistical

approaches have become popular. These learn *about* the input motions, and generate new output with similar characteristics, yet without directly cloning the idiosyncrasies they learnt from. Finally, it was seen that synthesis is also possible without having to combine motions at all, by instead editing a single one in isolation, using signal processing techniques, [BW95], geometric warping, [WP95], or manipulation in the frequency domain [UAT95]. As long as such editing remains moderate, these methods can produce useful results, retaining the motion's natural quality.

It should be realised that many of the above methods are not limited to creating motion from captured data, and would work equally well with sequences generated by keyframing, kinematics or physics-based methods. However, capture technology is in widespread use, and does appear unrivalled as a recipe for natural-looking motion. Nevertheless, a little caution may be in order, as upon witnessing the artistic expression and emotions in hand-drawn and keyframed animation, those distant cries against the devil's rotoscope do inspire a considerable measure of sympathy.

5

Character Motion Blending in the Frequency Domain – a Streamlined Approach

5.1 Introduction

Individual characters in a virtual environment should behave differently, walking at varying speeds with differing styles of motion, while navigating a course befitting their particular location and goal. Achieving this by naively playing back sequences from a large database of motions comprising many thousands of actions and styles is clearly not practical, so research has sought ways of modifying existing data to re-use it, thus *creating new* motion, instead of retrieving from storage what was previously sourced. Furthermore, fine control is required to enable characters to interact convincingly with each other and with their surroundings, enacting precise movements like the most subtle of turns or gradual changes in speed and associated gait geometry. Motion interpolation is a means to attain such delicate control by merging two or more sequences to create another, as guided by continuously variable blend weights. It exists in two variants, one working with DOF values, blending their trajectories as functions of time, and the other merging selected spectral content in the frequency domain. Motion blending and

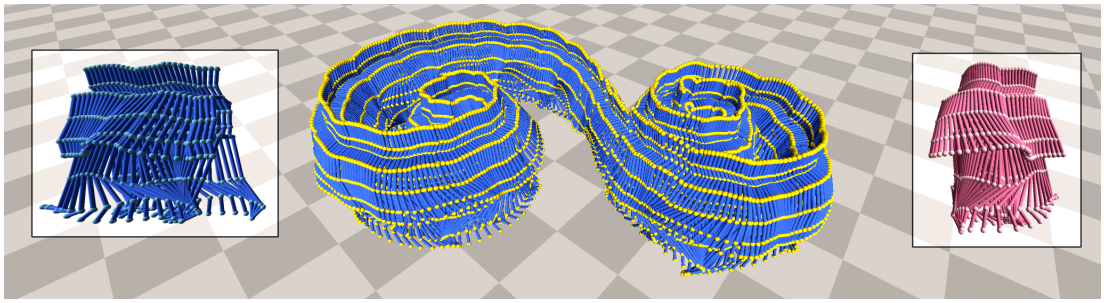

Figure 5.1: Successive animation frames from the implementation accompanying this chapter, showing a skeleton performing a continuous range of turns from sharp right, to less sharp, straight, left, then sharp left, created in real time from two short pre-processed input sequences (inset).

manipulation in the frequency domain is an established, if sparsely populated, subfield of character animation, to which this chapter[1] adds an approach which both consolidates and augments existing methods [UAT95, BW95, PL06], while, as illustrated in Figure 5.1, amply fulfilling the above-stated requirement of creating new motion from limited input.

The method presented here is an alternative to previous work on frequency domain blending, and is primarily contrasted with the more recent contribution of Pettré and Laumond [PL06], described in Section 4.7, over which it shows advantages in the context of interactive virtual environments such as video games, as well as in the animation of autonomous crowd scenes. The proposed approach, like that of [PL06], uses the discrete Fourier transform (DFT) and Fourier synthesis (Appendix A), to enable motion interpolation in the Fourier domain, but does so in a more flexible manner which is also

[1] Associated publication:
[MLD10] Michael Molnos, Stephen Laycock, and Andy Day. Using the Discrete Fourier Transform for Character Motion Blending and Manipulation – a Streamlined Approach. In *EG UK Theory and Practice of Computer Graphics 2010*, pages 207–214, Sheffield University, United Kingdom, 2010. Eurographics Association.

more efficient in certain contexts (clarified in Sections 5.7 and 5.8), while dispensing with requirements like the need to resample input data.

Efficiency is increased by using a more compact formula for Fourier synthesis, though the associated contribution lies not in the use of this formula, but in the proper approach to the associated problem of phase angle blending. Only this way can the cheaper formula *consistently* generate correctly-formed character poses, unlike its use with the naive blending of phase angles observed in [UAT95].

Moreover, Pettré and Laumond [PL06] had represented input motions as points in a two-dimensional space whose axes indicated linear and angular velocities of the skeletal root node. The points were joined by a Delaunay triangulation, defining a blending space, whereby for every location within it, blending weights were determined by the distance from that location to the vertices of the enclosing triangle. In contrast, the vertices of triangle networks in the hereby-presented approach are not constrained in this manner, and as a result these networks can, as will be shown, blend motions which the method of [PL06] could not. Furthermore, this completely different perspective in viewing blending triangle networks allows them to be formed as intuitive interface devices, appropriate to the intended focus on games and other user-driven character animations. (Interface-style networks also have further potential exploited in Chapter 6).

The measures described in this chapter thus provide a *streamlined approach* to motion blending in the frequency domain, a designation which, for convenience, is used throughout this chapter to differentiate the presented work from previous approaches. The proposed methods are put firmly in context by their inclusion in an implementation-level description of a Fourier-blending system which incorporates them, providing a detailed depiction of many of the steps involved in its construction, as well

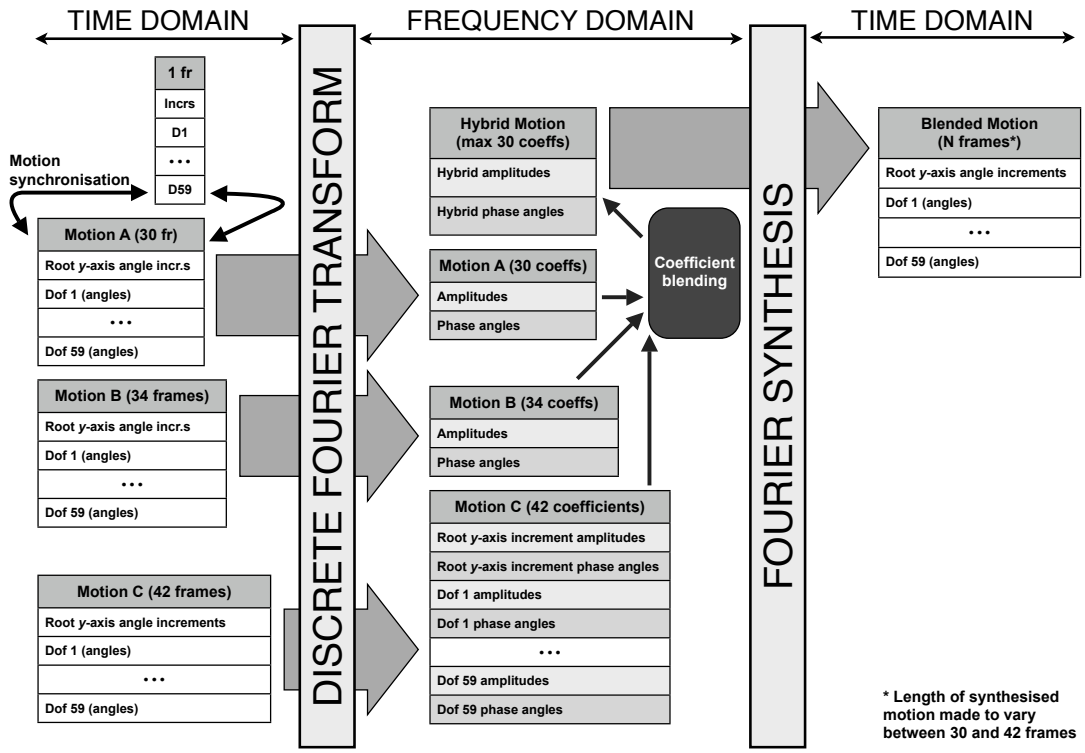

Figure 5.2: High-level view of the implemented system, shown simplified and referring to a single blending triangle. Root y-axis angle increments and motion synchronisation are described in Sections 5.4.4 and 5.4.6 respectively.

as their requisite order, with the aim of assisting anyone wishing to emulate the presented work.

5.2 Synthesis Overview – Context for Streamlining

A concise high-level view is first given, of the motion synthesis system to which the streamlining measures – described *elsewhere* in this chapter – are applied. Similarity with previous work is to be expected at this level.

Motion captured input data is pre-processed as described in Section 5.4, which includes conversion to smooth periodic sequences (cyclification) and subsequent representation in the frequency domain via the DFT. A continuous blending space is defined by considering input motion data to be positioned at the vertices of a triangle network.

At runtime the location of a blending point in this space specifies an interpolation to be performed on the frequency domain data of three of the input motions, with blending weights defined by the distance from the blending point to the vertices of the enclosing triangle. Synthesised motion is obtained by then returning the blended data to the time domain using Fourier synthesis. The runtime procedures are described in Section 5.5. A simplified overview of the system implemented for this chapter is given in Figure 5.2.

5.3 Fourier Series Representation and Phase Angle Interpolation

Several parts of this chapter make reference to the representation of Fourier series expansions employed in this thesis, and the associated issue of blending phase angles. A precursory clarification is thus due. Fourier series are further explained in Appendix A which gives general background theory on Fourier-related matters.

The Fourier series expansion given in [PL06] is

$$m(t) \quad = \quad \frac{\alpha_0}{2} + \sum_{k=1}^{N} \alpha_k \cos\left(\frac{k\pi t}{T}\right) + \beta_k \sin\left(\frac{k\pi t}{T}\right) \tag{5.1}$$

where α_0, α_k and β_k are magnitude coefficients obtained via the DFT. The formula is shown only to illustrate how it involves twice as many trigonometric functions per harmonic, k, as that used in the streamlined approach proposed in this chapter (Equation 5.10). The other terms are not relevant to the current treatment and hence not further explained. By comparison, Equation 5.10, as employed in the streamlined approach, is just as able to represent or synthesise waveforms despite having only half as many trigonometric functions[1]. With this equation, the coefficients from the DFT

[1] As explained in Section 5.5.2 the DC term $\frac{m_0}{N}\cos p_0$ of Equation 5.10 is replaced in practice by a low-cost blend of three scalar values, circumventing the use of m_0, p_0 and the trigonometric function.

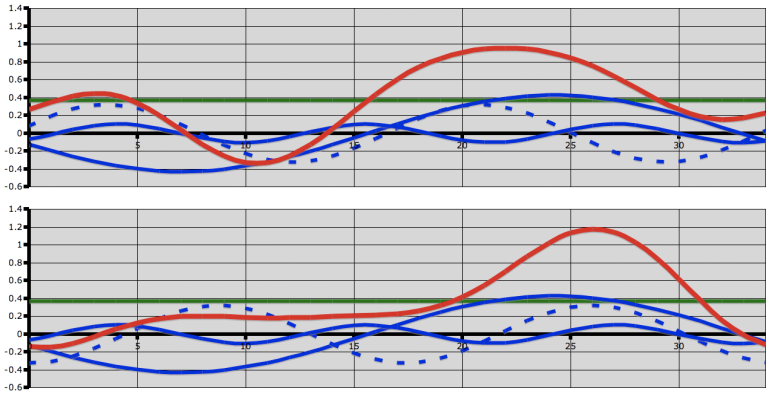

Figure 5.3: The significance of Fourier phase angle coefficients illustrated by their effect during Fourier synthesis. A waveform is constructed by the summation of a steady-state "DC" value (green) and – in this example – three harmonics (blue). Both plots are identical bar an increment in the phase angle of the second harmonic (dotted line), which is seen to modify the nature of the resulting waveform (red) while leaving the period unaffected. Since each skeletal DOF expresses one such (red) waveform, phase angle coefficients change the *style* of skeletal motion.

are given by m_k and p_k representing both magnitudes and phase angles, which in turn creates the need to blend phase angles. A comprehensive description of the equation and its symbols is given later in Section 5.5.2.

Blending is explained in Section 5.5.1 but, in brief, consists of taking a weighted average of the coefficients from each motion in the blend. While blending α_0, α_k and β_k, or m_k in this manner is problem-free, this is not the case for the phase angles, p_k.

A phase angle represents a time-shift in one of the sinusoidal components (harmonics) inherent in a single DOF's motion and affects the shape of the overall waveform associated with that DOF while, predictably, leaving the period unchanged. Consequently, phase angles underlying skeletal motion affect motion *style*. This is illustrated in Figure 5.3. While naively blending them may succeed in some cases, there are others in which it generates uncoordinated-looking motion. The non-trivial nature of angle blending is illustrated by the simple example of a 10° angle and one of 350°, in which

a proper blend is clearly not obtained with the arithmetic mean, unlike for angles of 10° and 20° where the mean appears valid. Even greater ambiguity is found when performing a weighted blend of three input motions.

The solution is presented by Mardia and Jupp [MJ00] in the context of directional statistics, and can alternatively be derived with a phasor diagram [WP85] as shown in Appendix B. It involves blending $m_k \sin p_k$ from each motion, while doing the same with $m_k \cos p_k$, following which the blended phase angle is then given by

$$p_{k,\ blended} = \text{arctan2}\big((m_k \sin p_k)_{blended}, (m_k \cos p_k)_{blended}\big) \tag{5.2}$$

where for three-input blending as used in triangle networks

$$(m_k \sin p_k)_{blended} = w_1(m_{k_1} \sin p_{k_1}) + w_2(m_{k_2} \sin p_{k_2}) + w_3(m_{k_3} \sin p_{k_3}) \tag{5.3}$$

$$(m_k \cos p_k)_{blended} = w_1(m_{k_1} \cos p_{k_1}) + w_2(m_{k_2} \cos p_{k_2}) + w_3(m_{k_3} \cos p_{k_3}) \tag{5.4}$$

and where w_1 is the weighting for motion 1 and, as defined above, m_{k_1} and p_{k_1} are the magnitude and phase angle of its k^{th} harmonic, with similar definitions applying for input motions 2 and 3. arctan2 is the two-argument version of the arctangent function, the cost of which can be reduced by replacing it with an algorithm based on the following approximation of the standard $\arctan(x)$ function, valid for positive x.

$$\arctan(x) \simeq p - \frac{q}{x + r} \tag{5.5}$$

where p, q and r are 1.597, 1.992 and 1.237 respectively for $0 \leq x \leq 0.5$, and 1.583, 1.259 and 0.609 for $x > 0.5$. Any angle calculated with this arctan2 algorithm (that based on the arctan approximation) has an error below 1°. While not required, higher accuracy is easily achievable by subdividing the domain into further bands.

The above-described method provides an unambiguous way of calculating blended phase angles. It must be emphasised that the magnitude and phase angle coefficients, m_{k_1}, m_{k_2}, m_{k_3} and p_{k_1}, p_{k_2}, p_{k_3}, for the motions of each blending triangle are obtained in preprocessing, as are the six products $m_{k_1} \sin p_{k_1}$, $m_{k_1} \cos p_{k_1}$, $m_{k_2} \sin p_{k_2}$, $m_{k_2} \cos p_{k_2}$, $m_{k_3} \sin p_{k_3}$ and $m_{k_3} \cos p_{k_3}$ whose evaluation imposes no runtime cost whatsoever.

Moreover both the arctan2 function, or alternatively the cheaper arctan2 approximation, involve minimal cost, because they remain valid for the *entire* output sequence, as further detailed in Figure C.2 of Appendix C, while making redundant half the trigonometric functions used in Equation 5.1, which need to be calculated on a *per-frame* basis. The contexts in which a performance gain is seen in practice are elucidated in Sections 5.7 and 5.8. The advantage remains if lookup tables are used instead of trigonometric functions. Section 5.7 (Results) quantifies the associated savings.

5.4 Preprocessing Input Motions

The steps discussed in this section prepare the input data for blending and only need be performed once for any number of subsequent animation sessions.

5.4.1 Selection

As blending is performed *between* input motion cycles, the number required can be low – a minimum of four or six being sufficient for small networks as in Figures 5.11 and 5.13 (Section 5.6.1), each of which show a frame of the demonstration video which accompanies this chapter (URL given with Results in Section 5.7 and in Appendix E). The small number of input motions makes it a trivial task to manually select them by browsing motion capture files. The main requirement is that they contain a complete cycle of data with start and end frames depicting similar poses.

Automating the process with a distance function as in [KGP02, KG04, HG07] to select cycles with well-matching endpoints is possible, and was implemented as an optional feature in the work in Chapter 6. However, such automation is not seen as a necessity as the aim is not to build a large database of motions but instead to select a relatively small number of the most visually appealing ones – a process for which many, in practice, will prefer to use human judgement. Indeed, the highest levels of automation may not be the choice of character animation practitioners [HG07], while automated extraction of input motions from source libraries is neither fully reliable [KG04], nor, apparently, does it necessarily justify the extensive implementation time.

A principal difference here, with the manual selection process of [PL06], is their requirement that all sequences be in phase, for example that all start with a left foot-strike. In the streamlined approach motion synchronisation (Section 5.4.6) makes this unnecessary, and instead allows a post-selection adjustment to synchronise motions, thereby allowing the input motion cycles to be selected solely on the basis of quality, without the restriction of enforcing a particular start frame, which is especially important when motion capture data is sparse. Furthermore, detailed examination of the input data is not required at this or any later stage, completely obviating the difficulty mentioned by Unuma, [UAT95], in estimating the period from measured joint angle data.

5.4.2 Root Rotation Angle Re-sequencing

The position of each node in the skeleton relative to its parent is given by the concatenation of a translation (fixed bone length) and three rotations (of the parent node), $R_a R_b R_c T$.

Similarly the relative position of the end node of a three-bone branch can be written

$$\begin{aligned} P_{end\ node} = & \; R_{root_a}\, R_{root_b}\, R_{root_c}\, T_{root} \\ & + \; R_{mid_a}\, R_{mid_b}\, R_{mid_c}\, T_{mid} \\ & + \; R_{end_a}\, R_{end_b}\, R_{end_c}\, T_{end} \end{aligned} \tag{5.6}$$

with the chain of transformations being applied in sequence from right to left and ending with the three root rotations.

At runtime pre-calculated root y-rotation angle increments discussed below in Section 5.4.4 are blended as defined by user-input, and accumulated frame by frame to create a continuous stream of *new* y-axis rotations for the root node, thereby allowing the rates of turn of input motions to be merged as desired and turns to be endlessly built up. To ensure that accumulated y-rotations merely rotate the skeleton about the vertical axis, the final (leftmost) matrix rotation in Equation 5.6 must be the root's y-axis rotation. The root angles in the motion captured input data are thus pre-processed to use YXZ ordering, acquiring fresh rotation values which when applied in this new sequence yield the same overall 3D rotation as did the initial angles in the original order.

The lightweight accumulation process described above, contingent to angle resequencing, fully determines the character's world-coordinate bearing at runtime, consistent with the path so far followed, and requires only a simple addition of interpolated pre-processed values. It occurs *instead* of the usual blending and application of y-axis values taken straight from the motion capture file. Runtime cost reduction is significant compared to established methods as in [KG03], where transformations must be computed and coordinate frames oriented (and positioned) prior to blending.

Furthermore, the problem said in [KG03] to beleaguer conventional blending when the relative bearing of two motions exceeds 180° (Figure 5.4) simply never occurs in the

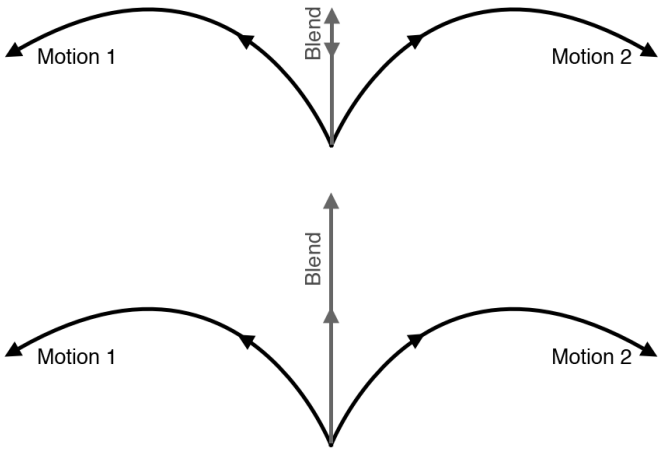

Figure 5.4: Interpolation of two walking motions with increasingly divergent bearings. In Kovar and Gleicher [KG03] the blended path is said to collapse when the bearing of one motion relative to the other exceeds 180° (top) unless using coordinate frame alignment, a feature of their presented registration curves, giving (bottom) a correct path which avoids folding back on itself. In comparison, the streamlined approach is immune to this failing, tracing the same path as the latter (bottom) but without its runtime cost, and with prerequisite angle re-sequencing being in *preprocessing* only. (Image based on [KG03].)

streamlined approach which, by blending small angle increments to create any turn, is entirely immune from this deficiency.

5.4.3 Cyclification

The term 'cyclification' as used in [AMH03, PB00, GBT04b], refers to techniques for processing individual motion cycles selected from input data, notably to amendments yielding compatible endpoints, with the aim of enabling smooth concatenation of the motion with itself, analogous to the tiling of graphic images.

In the streamlined approach, cyclification of an input cycle offsets the angular motion data of each DOF for the first and last frames by equal and opposite amounts so they acquire the same values. Intermediate frames are adjusted accordingly by motion displacement mapping [BW95, WP95, KG03, LCR+02]. The duplicate final frame (describing a pose identical to the first) is then discarded. This simple procedure creates

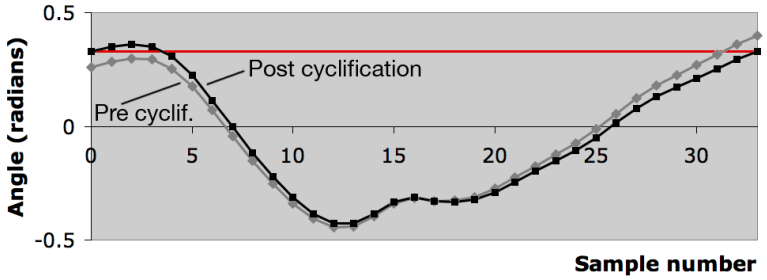

Figure 5.5: Cyclification offsets the angle values of a DOF waveform (grey) so values for the first and final frames become identical with intermediate frames being adjusted accordingly (black). The end frame (still shown in figure) is then discarded to avoid duplication.

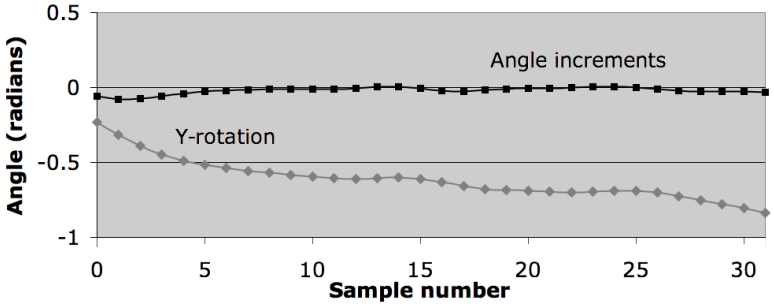

Figure 5.6: Example of the elimination of the sharp discontinuity in the root node y-rotations at the ends of the motion sequence by converting them to angle increments.

an input motion cycle able to be repeated with no perceived discontinuity. Compatible endpoint *velocities* are assured in practice by the steady pace of the selected input cycles. Cyclification of the waveform for a single DOF is shown in Figure 5.5.

Cyclification ignores root y-rotations, as making them match at cycle endpoints would prevent characters from accumulating rotations during successive cycles and walking in a circle.

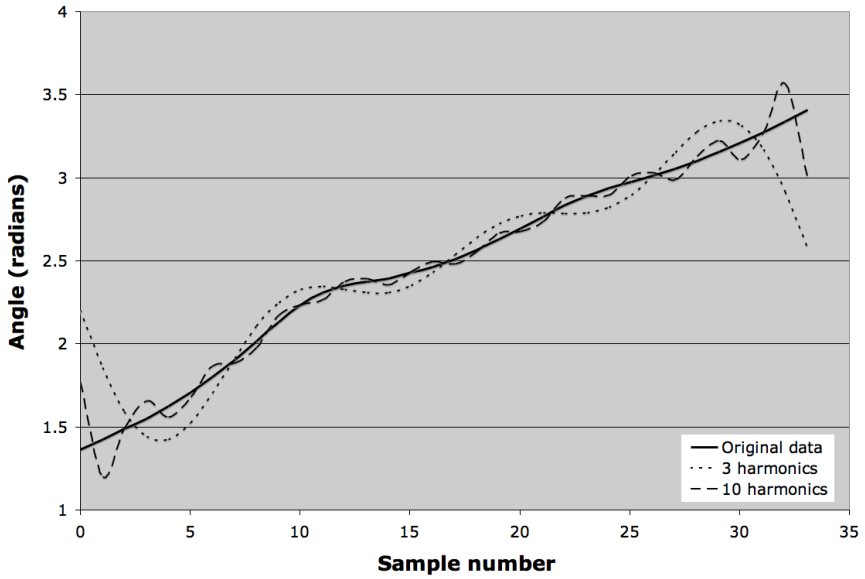

Figure 5.7: Reconstruction of a waveform (solid line) from its frequency domain components is less accurate when fewer harmonics are used, especially in the region of any discontinuity, as may apply, near the sequence endpoints, with root y-rotation data if left unaddressed (due to the large jump in angle value when passing from the final frame to the first).

5.4.4 Root *Y*-axis Angle Increments

As root y-rotation data must be disregarded by the cyclification process, discontinuities may be present, (as shown in the lower plot in Figure 5.6). Such root data, taken from three input motions, is to be combined in the frequency domain and rebuilt into a single hybrid angle sequence using Fourier synthesis. However, in order to reduce cost the number of harmonics used during synthesis is restricted, which, if left unaddressed, would result in distortions appearing in the regions of any discontinuity in the output motion (such discontinuity resulting from blending discontinuous inputs). This is illustrated in Figure 5.7 which, in the interest of clarity, considers not a hybrid blended motion but merely a single discontinuous input motion, being reconstructed from its constituent parts obtained via the DFT. Furthermore, the process of blending

and concatenating motion is far simpler with continuous data. The discontinuity is therefore removed, which is done by replacing the y-axis data for the root with angle increments as shown in Figure 5.6.

The increments are given by

$$i_n = \begin{cases} a_n - a_{n-1} & n \geq 1 \\ (a_{N-1} - a_{N-2} + a_1 - a_0)/2 & n = 0 \end{cases} \tag{5.7}$$

where a is the angle value and N the number of samples in the motion cycle.

During animation it is the angle increments, instead of the angles themselves, which are blended as guided by user-input, and the new increments thus obtained are accumulated frame by frame, creating the y-axis orientation used in the synthesised motion.

5.4.5 Limp Correction

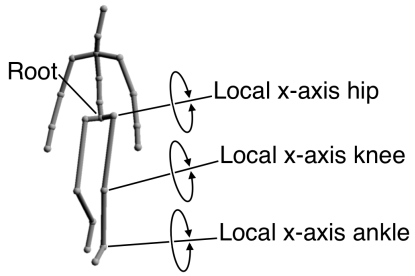

Figure 5.8: Limp-removal by returning the root node to its original elevation, if affected by cyclification.

While cyclification always generates motions which can be repeated without discontinuity, it can, on occasion, generate a limp if input data with sufficiently similar start and end frames could not be sourced [1]. This is removed by storing the vertical position of the root node (pelvis) for each frame prior to cyclification, and later adjusting

[1] This issue disappeared entirely in the implementations presented in Chapters 6 and 7, for which improved source data was obtained.

the x-axis angles in the ankles, knees, and hips (Figure 5.8) to re-establish the same elevations of the pelvis as were present before.

5.4.6 Motion Synchronisation

In order for motions to blend, they must first be aligned with each other. Dynamic time warping [Sen08] as used in [KG04, BW95] and discussed in Chapters 1 and 4, is an established and useful method whose complexity can be dispensed with, however, as blending is performed within network triangles, between sets of three pre-processed cycles of similar content and short duration, which instead allows a more straightforward approach to motion alignment. Pettré and Laumond [PL06] also dispensed with timewarping, but ensured synchronisation by requiring that input sequences start with the same posture, such as a left foot strike, which can limit the selection of input cycles, and this, too, is avoided in the streamlined approach.

Motion synchronisation, instead, aligns one sequence to another by shifting the angle values for each DOF of one motion relative to those of the other, with frames pushed out of a sequence being re-inserted at the opposite end. The angle increments calculated for root y-rotations (Section 5.4.4) are similarly offset. Figure 5.9 highlights the duality of input motion data, which can be considered both as a collection of DOF values and a collection of frames, and furthermore illustrates the frame-offsetting process underlying motion synchronisation. Synchronisation involves blending a 50% contribution of one input sequence with the same contribution from one of the other input motions in a blending triangle, and monitoring the resulting animation to determine the best offset value. Offsetting is performed on all DOFs simultaneously, and synchronisation of one motion relative to the other need only be performed once for each pair of motions in a blend. To blend between three motions A, B and C, aligning A to B and B to C is sufficient – the process will also have synchronised motion A with motion C.

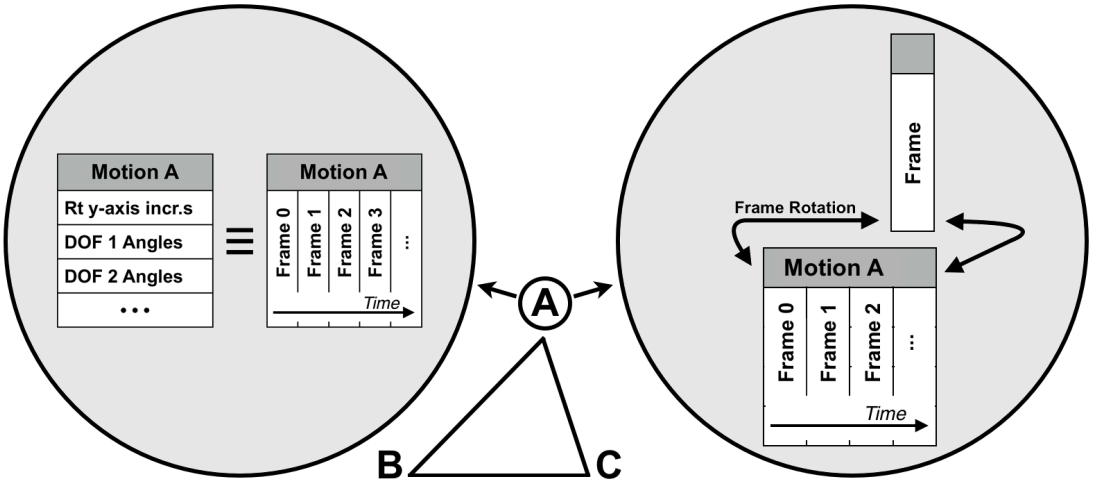

Figure 5.9: Input motions assigned to blending triangle vertices can be viewed in two ways (left), either as collections of angle values (or angle increments for root y-rotation data as explained in Section 5.4.4), or as a sequence of frames over time. Motion synchronisation (right) rotates the frames of one sequence relative to those of another chosen as reference.

Automatic synchronisation is implemented and detailed in Chapter 6, however, the earlier work described here uses real time visual feedback of the animation for various values of synchronisation offset to determine the most appropriate. This manual once-only procedure is in fact quite sufficient, taking only seconds to perform, with saved results remaining available for any number of animation sessions to be run in future.

5.4.7 Discrete Fourier Transform

The DFT is then performed on the smoothed (cyclified) motion data for every angular DOF, and in the case of the root y-rotations, on the angle increments. This calculates, for each DOF, two sequences in the frequency-domain, one real and one imaginary, of length N equal to that of the original sequence. From these, two further sequences are calculated, also of length N, which give the amplitudes and phase angles of the input waveform's component sinusoids. These particular amplitude and phase

angle sequences are chosen for reasons of efficiency, and are not the same coefficients calculated by [PL06] (Section 5.3).

DFT formulae vary slightly and any could have been used. The selected one, given by [Med00] and [CDH00], is shown below

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi nk}{N}} \tag{5.8}$$

where X_k is a complex value comprising the k^{th} element of the real and imaginary sequences mentioned above and x_n is the n^{th} value of the time-domain sequence on which the DFT is being performed.

The DFT is performed in preprocessing on relatively short data sequences. A fast Fourier transform (Appendix A) could have been used but is not required.

5.4.8 Phase Angle Blending Pre-calculation

As explained in Section 5.3, in order to enhance Fourier synthesis efficiency while maintaining unambiguous blending of phase angles, the sines and cosines of each phase angle, and the products $m_k \sin p_k$ and $m_k \cos p_k$ are calculated in preprocessing for the harmonics of each DOF, which is done at this stage.

5.5 Runtime Processing of Output Motion

The following blending and Fourier synthesis steps are required each time the weightings of the input motions need to change. Thus for a mouse-based user-driven system, execution of these steps and the associated cost cease when the mouse stops moving, although animation of the skeleton proceeds unabated.

5.5.1 Blending

As blending is done between three motions at a time, it occurs in a triangular blending space. Phase angle blending was covered in Section 5.3. Magnitude blending calculates a weighted mean of the amplitude sequences which the DFT produced for each DOF of each of the three motions in preprocessing. Thus for each DOF a new hybrid set of Fourier coefficients is created. The weightings used are given by areal barycentric coordinates, thus

$$w_a + w_b + w_c = 1, \tag{5.9}$$

where w_a, w_b and w_c are the weightings for the three input motions. Extrapolation is possible by placing the blending point outside the triangular area, in which case one or two of the weightings will be negative.

5.5.2 Fourier synthesis

Fourier synthesis builds the output waveform from the blended Fourier coefficients. It exists in a number of variants. The version shown by Equation 5.10 is matched to the DFT formula given above.

$$R_n = \frac{m_0}{N}\cos p_0 + \frac{2}{N}\sum_{k=1}^{H} m_k \cos\left(\frac{2\pi k n}{\mathcal{N}} + p_k\right) \tag{5.10}$$

where N is the length of the blended Fourier coefficient sequences, $\mathcal{N}$ is the desired output sequence length and H is the highest harmonic used (indexing such that 1 is the fundamental). m, p and R hold the k^{th} blended Fourier magnitude and phase angle coefficients, and the n^{th} time domain output value respectively, where n varies from 0 to $\mathcal{N} - 1$.

The DC value, shown as $\frac{m_0}{N}\cos p_0$ for the sake of completeness, can, in practice, be replaced by a number of cheaper variants. To this end, the implementation used the weighted average of the DC values of the input motions, these values resulting from the preprocessing of the input sequence means.

Unlike Equation 5.1 used by [PL06], Equation 5.10 in the streamlined approach includes no time parameter. Instead n specifies the frame in the output motion cycle for which time domain values are being calculated.

The use of $\mathcal{N}$ for the output length in addition to N for the input allows the synthesised motion to have a dynamically varying sequence length. It is chosen to be the weighted average of the lengths of the input motion data. Thus, the more the output sequence is based on a given input motion, the more its length will resemble that of this particular input motion. As an auxiliary benefit the adjustable output length also provides a convenient way to resample input motions, by applying the DFT followed by synthesis to create a sequence comprising any desired number of frames, although the streamlined approach itself, unlike that of [PL06], does not require resampling at any processing stage.

Input sequence lengths require consideration too, as input motions may comprise different numbers of frames, and consequently sequences of Fourier coefficients of varying lengths. If these lengths are K, N and M, with M the shortest, the generation of a weighted average can only take place for the first M coefficients in each sequence. There are thus at most M hybrid amplitudes and phase angles available for Fourier synthesis, and the highest number of harmonics which can be used to create the time-domain output sequence is $M - 1$. In practice this has never been a problem, however, as the number of harmonics required even for highest quality output is always well below the number available.

Using all the DFT-generated frequency domain data for a given input motion, and replacing $\frac{2}{N}$ by $\frac{1}{N}$ in Equation 5.10, would allow an exact copy of that motion to be re-built, akin to using the inverse-DFT[1]. However, this would be costly, and Equation 5.10 is preferred which, unlike the aforementioned alternative, generates output waveforms of the desired amplitude when using small numbers of harmonics, as expounded in Appendix A.

5.5.3 Frame Counter Adjustment

At any point in time during character animation, the skeleton adopts a single pose from the entire collection in the output sequence[2]. The frame counter holds the frame index for that pose, and increments it at each time step, rotating it back to zero upon exceeding the final frame.

The streamlined approach, however, allows blending to take place between input cycles of *different* length, with a consequently *varying* output sequence duration $\mathcal{N}$. To prevent the character from jumping forward or backwards in time within the animation during user input, the frame counter requires adjusting, following changes in output length, such that normalised progress within the cycle (a value from 0 to 1) remains invariant under these changes.

5.5.4 Root Translation

The streamlined method makes no use of any fixed root offset or time-varying root translation in the motion capture data. Foot constraints are used instead to drive the skeleton, akin to actual human motion. As explained below, this completely avoids

[1]The IDFT is explained in Appendix A, Section A.5, and shown to be essentially the same as Fourier synthesis, the difference between the two lying in the format of their input data.

[2]Whether the complete set of frames is computed, or only a single one, depends on the use to which the implementation is put.

footskate, the frequently surfacing issue of sliding feet [CH07, LBJK09, Stu98b]. When a foot which was previously higher than the other becomes the lower one, the position of that foot, projected in the ground plane, is stored and known as the 'anchor point'. In subsequent frames the skeleton is translated such that the appropriate foot is placed at the anchor point which propels the skeleton forward. The root position at each frame is thus explicitly calculated to zero all motion of the stance foot instead of relying on motion captured root position values, thereby making footskate impossible.

This elementary approach to propelling the skeleton is merely intended, and sufficient, to demonstrate the synthesis of walking motions by the proposed Fourier blending method. It is extensible, however, and enhanced in following chapters, to cater for running motions, and for higher quality output.

5.6 Blending Triangles

5.6.1 Manual Vertex Placement

Pettré and Laumond, [PL06], employed automatic positioning of triangle vertices applicable to the motion planning context of their work. Vertex placement was made to reflect the linear and angular velocities of the skeletal root node, as shown in Figure 5.10. Such constraints, can, however, impose significant limitations. One is that being based on root velocities, such triangle networks cannot be used for motions where these velocities are not significant, such as standing near-motionless, gently dancing on the spot or standing around while fidgeting. In the streamlined approach the vertices within networks are placed manually, however, avoiding this limitation, as shown in Figure 5.11 which is taken from the aforementioned video (URL in Section 5.7 or Appendix E).

Considerable freedom is possible during vertex placement. Triangles can, if desired, be stretched or squashed to modify the sensitivity of the blending point, as shown in the

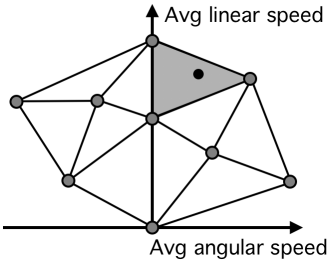

Figure 5.10: Calibrated blending triangle network of the type used by [PL06].

lower part of Figure 5.12. Assuming the same input motions are used as in the network shown bottom left, that on the right hand side would facilitate delicate tracking of straight lines and slight curves, while still allowing turns to be as sharp as before.

Perhaps the most appealing aspect of manual positioning is that it allows the triangle network to be displayed as a user-friendly interface element, proportioned to best suit the application in question, which is both more intuitive and easier to control than would be mouse or joystick manoeuvring in the awkwardly shaped networks of [PL06], or the high-level editing tools suggested in [BW95] (such as sliders to control the gain of individual frequency bands). A simple interface-style network is shown in Figure 5.13, and again, in the accompanying video.

While showing the network and blending point on-screen provides an intuitive way for the *user* to control a character, manually designed networks can also act as interfaces between program modules to allow *automated* path-following, despite their not being calibrated as were those of [PL06]. Figure 5.14 provides an illustration which compares favourably with path following in the seminal work of [KGP02], highlighting the superiority of blending-based methods for fine control over motion synthesis by concatenation.

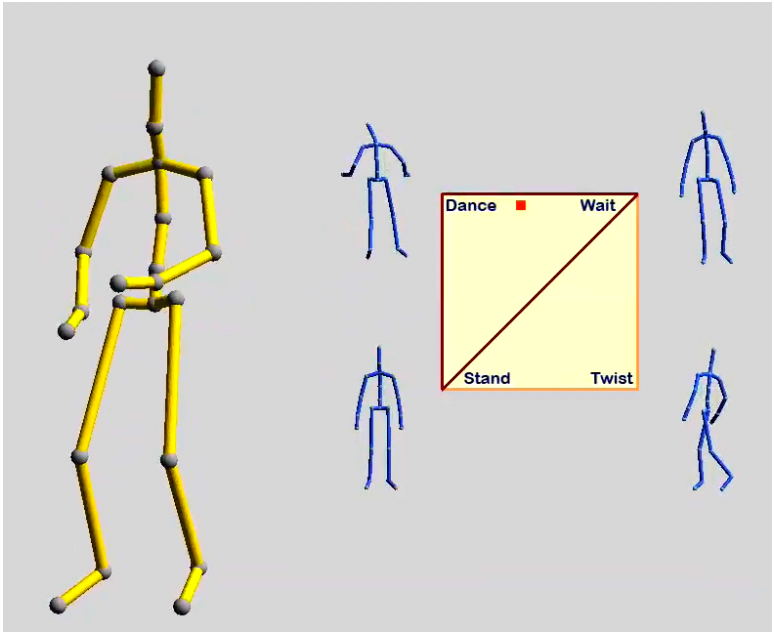

Figure 5.11: Manual vertex placement instead of automatic as in [PL06] allows blending of motions with negligible root velocities, as highlighted by the small animated skeletons enacting input motions, these being dancing on the spot, twisting round, and two different styles of standing-around type motions. Blended output is displayed on the large skeleton.

5.6.2 Choice of Diagonal

Figure 5.12 shows two similar versions of a four-triangle area. The top left-hand network has different diagonals from the top right-hand one, which means (almost) any point within the blending area will generate a blend from different motions depending on which network is used. In practice, however, little difference is noticed, if any, and moving the blending point, say, vertically from the midpoint of edge DC to that of edge BA creates a slow-paced moderate left turn, accelerating to a faster-paced one, whichever the diagonal direction.

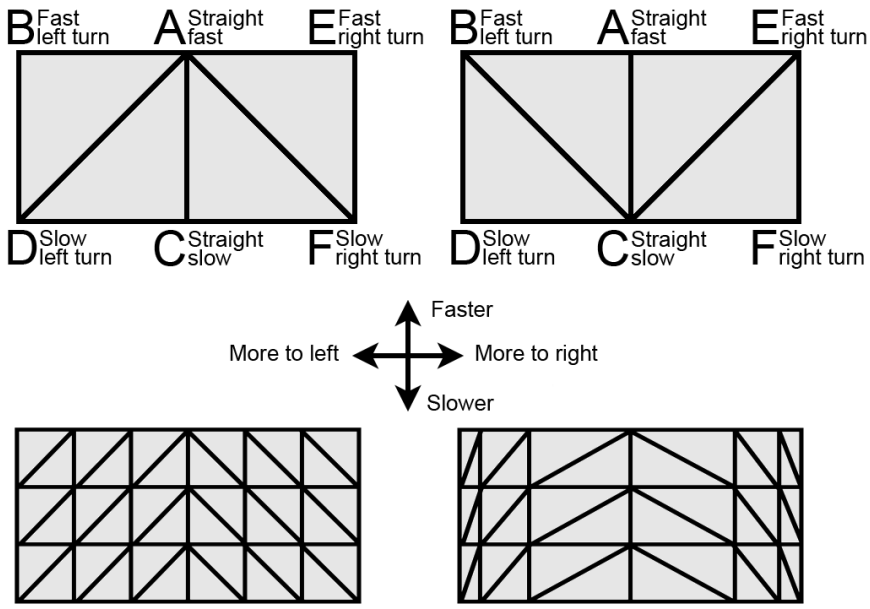

Figure 5.12: Diagonal directions, top left and right, have little effect. The denser triangle networks, bottom, give higher quality output and have different blending point sensitivities.

5.6.3 Vertex Density

The blending space in Figure 5.12, top, spans motions from slow to fast, and from sharp right to sharp left turns. If suitable motion capture clips are available, the same area could be covered by a denser network resulting in blends between motions of greater similarity, and hence a higher quality synthesised output (bottom left). Nevertheless, even a sparse network, as shown top, left and right in the figure, can produce natural-looking motion if the input sequences are of sufficient quality.

5.6.4 Network Cascading

Networks are not confined to these simple two-dimensional examples. The input motions at triangle vertices can themselves be controlled by blending triangles which have their own blending points. Thus the nature of input motions could – occasionally and hence efficiently – be automatically and gradually adjusted, for example, between tired,

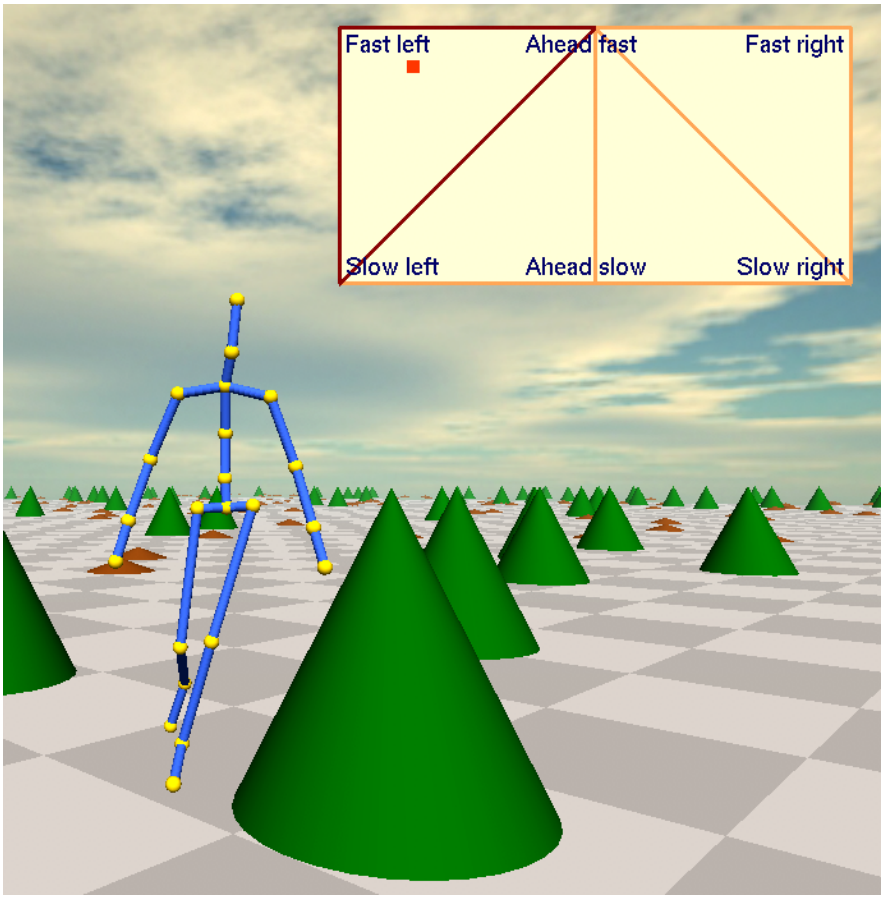

Figure 5.13: Manually placed triangle vertices create an intuitive interface for user-friendly animation control. The blending point is in the top left portion of the network, with the skeleton, accordingly, performing a fast left-hand turn.

energetic and relaxed gaits, in response to the character's changing virtual physical state. In comparison, the superposition of qualities such as tiredness, as introduced by Unuma et al [UAT95] (Chapter 4, Section 4.7) would need to be performed continuously to maintain a tired walk, thus involving greater cost.

5.6.5 Substituting for Missing Input Motion

The small triangle network shown top left in Figure 5.12 is included in the implementation video, where it demonstrates the variety of motion that can be synthesised from

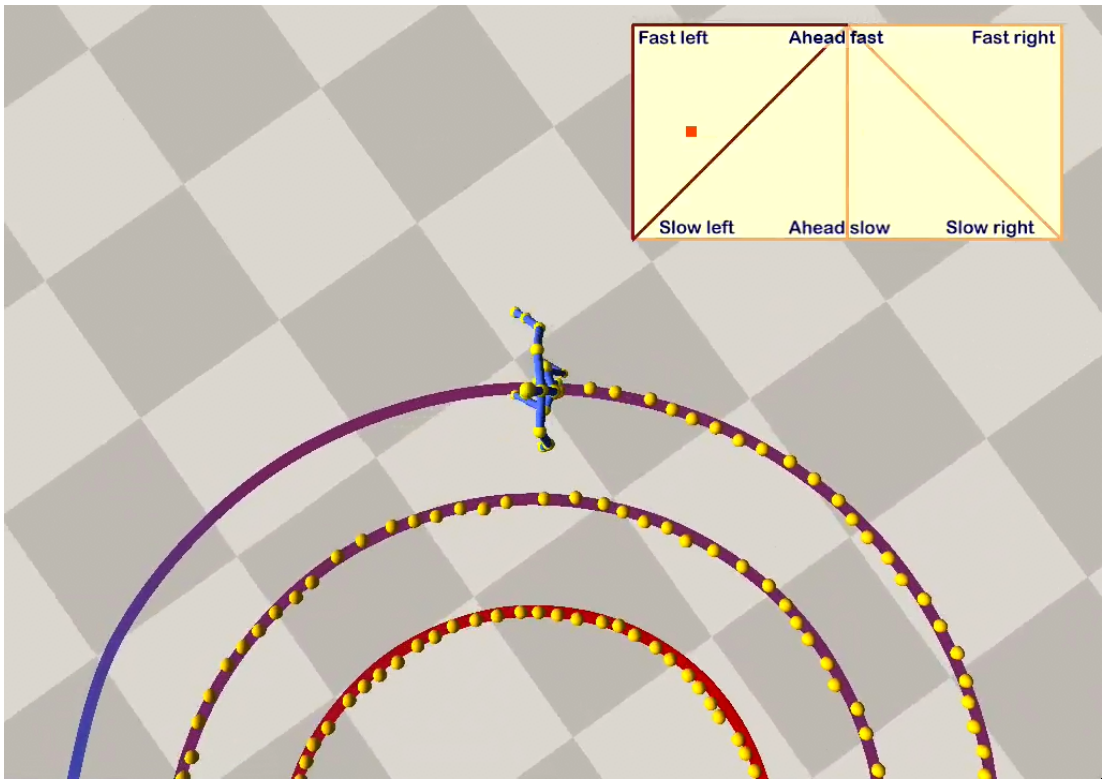

Figure 5.14: Simple path-following example, with the blending point position within the network being automatically adjusted to track the indicated route, whose colour specifies the intended speed for that section of the track, ranging from red for slow to blue for fast. The density of the trail of yellow dots, showing root position at each time step, thus varies accordingly. Some small side-to-side tracking errors are unavoidable due to the natural lateral swinging motion of the pelvis while walking as well as the relatively low real time frame rate (30fps).

a very small number of input motions, having used only three short captured sequences (A with 30 frames, B with 34 and C having 42) to create all the motion available in the rectangular blending area. While E and F were created by reflection, D was created using an extension of the above-mentioned superposition technique.

Instead of adding a quality such as briskness to an existing motion, a very different geometry was created. The straight fast walk (A) was subtracted from the fast left turn (B) resulting in a motion which is not a walk cycle but could be described as

"turning-left-ness" and this was then added to the straight slow walk (C) to generate a slow left turn (D) where previously none existed. Thus not only does Fourier blending function well with little input data, but it is sometimes able to create input sequences where these are not available.

5.7 Results

5.7.1 Preprocessing

Preprocessing of each input motion was performed in seconds with the result being saved to file. The precise times for this once-only process are deemed inconsequential and no attempts were made to optimise them. (In the implementations of the following chapters the time for performing these same steps drops to fractions of a second per motion – given, for comparison, similarly-lengthed input sequences – though again this is seen as incidental with focus, instead, being on runtime cost). Furthermore, preprocessing scales linearly with the number of input motions, it is performed once only, and the motions thereby stored are immediately available upon following program launches. The preprocessing duration is thus insignificant.

5.7.2 Demonstration Video

A video can be downloaded from `http://www.urbanmodellinggroup.co.uk/fouBlend.mp4.zip`. It demonstrates a whole spectrum of continuously variable motions, displaying favourable quality given the small number of input sequences, the constraints of real time synthesis, and the absence of any post-processing which would embellish the results proper. (Post-processing is discussed in Section 5.8). Additionally, two small interface-style triangle networks are shown in succession, each, in its operation, depicting palpably intuitive user-interaction with the skeleton. Furthermore, a section is

included showing the interpolation of motions with negligible root velocity, which could not have been blended with the automatically-created networks of [PL06].

5.7.3 Fourier Synthesis

Table 5.1: Fourier synthesis and blending times per DOF for a 33-frame output sequence using various numbers of harmonics. Times refer to *sequence* creation, not merely frames.

Harmonics	1	2	3	4	5	10
Blend. (µs)	0.14	0.24	0.37	0.49	0.59	1.18
Synth. (µs)	2.68	5.01	7.44	9.84	12.30	24.23

Measured times required for blending and Fourier synthesis using various numbers of harmonics are given in Table 5.1. They were obtained on an Apple MacBook Pro *laptop* with Intel Core 2 Duo CPU running at 2.53 GHz with 4GB RAM. The values refer to the creation of an entire 33-sample output sequence for a single DOF, created by blending sequences of length 30, 34 and 42. Blending times shown *include* the associated triangle selection and weighting calculations.

A direct comparison with [PL06] is difficult due to the older hardware they used (2006 publication, 2010 for Streamlined approach). They reported 0.92 ms for their equivalent steps to compute a single posture on a 62 DOF skeleton using Fourier coefficients up to the eighth rank. It can be deduced from Table 5.1 that a single posture for a 62 DOF skeleton using the same number of Fourier coefficients as did [PL06] would be computed in 38 µs. While admittedly a simplistic comparison, the result does appear favourable.

Having computed a walk cycle, blending and synthesis need only be performed when a *change* in the currently playing motion sequence is desired with the animation otherwise proceeding at zero cost. In comparison, Fourier synthesis is stated to be ongoing in the method of [PL06], as further discussed in Section 5.8.

5.7.4 Fourier Synthesis – Relative Performance

Table 5.2: Synthesis times relative to the method of [PL06].

Harmonics	1	2	3	4	5	10
t_{rel} Pettré and Laumond	1.0	1.0	1.0	1.0	1.0	1.0
t_{rel} streamlined approach	0.807	0.726	0.718	0.691	0.702	0.689

An objective comparison with the method of [PL06] *is* possible, however, in respect of Fourier synthesis. Table 5.2 gives computation times for the synthesis formula of the streamlined method *relative* to that of Pettré and Laumond [PL06], obtained by running an efficiently coded version of each on the same hardware. While the compact formula requires the arctan2 approximation for phase angle blending, this overhead is trivial, as described in Section 5.3 and shown in Table 5.1 (whose blending times include more than just phase angle blending). Table 5.2 shows the streamlined approach to be significantly cheaper, and slightly more so for higher numbers of harmonics. Relative performance is highly dependent on selected circumstances. Those chosen here assume an entire motion cycle (two steps) is animated without user interaction, whereby in practice user (or autonomous) input could be both more or less frequent. In no circumstances, however, is the streamlined approach more expensive than that of [PL06], though it can be significantly cheaper. Section 5.8 discusses this issue in more detail.

5.7.5 Triangle Selection

Pettré and Laumond [PL06] state the triangle selection process to be logarithmically dependent on the number of motion captures. This can be made more efficient, however, due to the coherence of position of the blending point from frame to frame, which in any time step never moves further than to an adjoining triangle, and usually remains in that

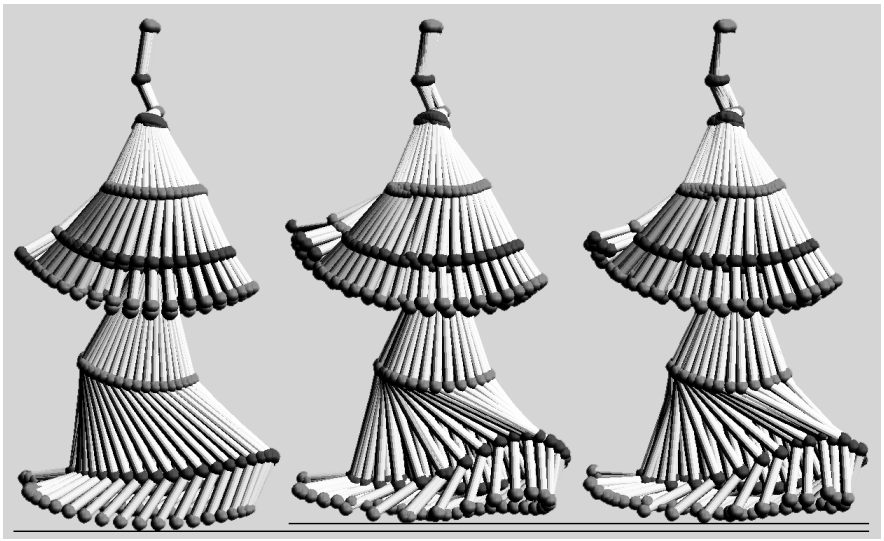

Figure 5.15: Walk cycles using (left to right) 1, 3 and 25 harmonics. Horizontal lines show the vertical reach of the step.

previously occupied. Accordingly, the implemented streamlined approach [MLD10] uses an appropriate data structure, whereby blending triangles store information about their immediate neighbours, and only relevant triangles are considered when determining occupancy, enhancing triangle selection to an $\mathcal{O}(1)$ process.

5.7.6 Harmonics

Unuma et al [UAT95] found 3 to 7 harmonics to be needed for realistic motion, which testing in the context of this thesis confirmed, while, similarly, [PL06] neglected Fourier coefficients above the 8^{th} rank. Figure 5.15 illustrates walk cycles using 1, 3 and 25 harmonics. Using a single harmonic looks overly smooth and exhibits a slight bounce due to the exaggerated vertical reach of the step. With 3 harmonics the walk cycle exhibits similarity with the near-perfect motion constructed using 25, though closer observation does reveal remaining discrepancies, the significance of which depends on context, ranging from inconsequential for less exacting conditions such as fast-moving action or

middle-ground characters, to excessive for foreground views in exacting applications using high-quality motion data.

The quality of motion achievable with differing harmonic counts is more objectively evaluated by comparing the node coordinates of two skeletons. Table 5.3 gives the average error between the node positions of a reference skeleton playing back unaltered motion capture data, and another playing back a synthesised version of the same walking motion, using various numbers of harmonics.

Table 5.3: Average node position error during synthesis.

Harmonics	1	2	3	4	5	10
Er_{avg} (cm)	2.22	0.83	0.46	0.38	0.19	0.08

With the employed skeleton of 1.673 m height, the average error was, as shown in the table above, circa 5 mm given 3 harmonics, falling off to well below 1 mm given 10. As discussed in the following section, these measurements do appear consistent with the subjective assessments given above, which, stemming also from [UAT95, PL06], deemed between 3 and 7, or possibly up to 8 harmonics, to be ample for all quality requirements possibly encountered in practice.

5.8 Discussion

5.8.1 Motion Quality and Post-processing

Post-processing, for "fixing small scale offensive artifacts" is seen by [AF02] as the third of "three natural stages of motion synthesis", so frequently is it required. In the frequency domain work of [BW95] and [PL06] the need for this step is explicitly mentioned, though it is not included in presented work. The same requirement will surely apply to [UAT95], as concurred by [GBT06], who similarly attributes this ne-

cessity to the time domain methods of [GBT04a, KG03, PSS02, RCB98]. In [LBJK09] and [AF02], post-processing is implemented as part of their presented work, although dedicated methods do exist also, as with [LMT07], and, more popularly, [KSG02], for footskate removal.

The streamlined approach, as with [UAT95, BW95, PL06], is presented without post-processing, though little would be required, and the method is entirely free of footskate. Although relatively minor, one motion-quality issue does surface, as now detailed below.

While devoid of footskate-proper for the reason given in Section 5.5.4, some unwanted sliding-*like* motion of the foot may be visible *just prior* to ground contact. This effect is all but removed in the following chapters, which use higher quality data, a greater density of input motions and enhanced foot constraints.

5.8.2 Triangle Networks

A principal difference between the streamlined approach and that of Pettré and Laumond [PL06] lies in the triangle networks used, with each type befitting its intended application. A comparison now follows, along with a further exploration of the selected type.

Calibration. The networks of [PL06] were intended for use with a motion planning platform. To this end, they were calibrated with triangle vertices accurately positioned within a two-dimensional velocity space. By joining the vertices to form a triangulation, the encompassed area specified all available combinations of linear and angular root velocities able to be synthesised by interpolation. Furthermore, the calibrated space meant that for required velocities of the skeletal root, the corresponding location could be pinpointed in the network, blending weights computed, and character motion synthesised possessing those specified root velocities.

Such networks were especially well-suited to path following, trajectory tracking and goal reaching.

Of particular interest in this thesis, however, are user-driven characters, as required, for example, in interactive applications such as video games. While vertices in the streamlined walking network are treated with a degree of similarity to [PL06], being placed higher up on screen for faster motion and further sideways for greater rates of turn, precise calibration is uncalled for, and indeed disadvantageous. The user employs visual feedback to navigate the character with a mouse, or other input device, and has no need to specify precise numerical rates of turn or forward velocity. Nor is such precision required for basic path following, as was demonstrated on video and shown in Figure 5.14.

Network Usage Connection with Footskate. Emphasis in the networks of Pettré and Laumond was on control of velocities, and not on the quality of motion, which [PL06] confirmed was afflicted by "unbelievable motion details" such as footskate. This is to be expected, however, as while blending root node data is straightforward (including by the method of [PL06]), applying the same blending weights to joint angles will not yield poses whose motion perfectly matches the synthesised root velocities, due to the nonlinear relationship between joint angles and the position of skeletal nodes. Footskate is thus unavoidable with such usage of triangle networks, closely related as it is, to the network's calibration. It is for this reason that the streamlined approach, instead, used foot constraints to propel the character, resulting in zero footskate proper, made possible, ironically, by the *lack* of any need for precise treatment of root velocities.

Motion repertoire. In the context of interactive user-driven characters, the triangle networks of Pettré and Laumond have significant limitations. One is that users

may wish to control a variety of motions which are not differentiated by their root velocities. In a calibrated network, these could not even form triangles. This highlights an advantage of the manually placed vertices in the streamlined approach, which, as demonstrated on video, can blend such motions, as in dancing on the spot. The range of motions escaping the remit of calibrated networks should not be underestimated, and also includes motions which *do* have significant root velocities, but where these are mutually similar, and therefore would correspond to nearby locations in a calibrated space. For this reason, motions of different style, yet similar speed and rate of turn, cannot be blended by the method of [PL06] (which ascribed motion style blending to future work), though the streamlined approach is quite able to do so. Furthermore, such motions as cannot be blended by [PL06], need not be confined to their own detached networks and used in isolation, but can, more usefully, reside within greater networks which include other actions like walking. Enhanced networks catering for this can be created in a number of ways, of which three are found listed further below.

User Interface. The above considered the motions themselves that the two styles of network can work with. Another important consideration for applications like games, however, is the extent to which each network type can accommodate favourable user-interface design. When a user drives the character at runtime, the blending point is continuously moved around within the blending space, which, for calibrated networks with their awkward shape, would mean continually paying attention to the network itself. The streamlined approach allows the building of rectangular networks, as demonstrated, whose simple shape makes them almost transparent in use, as the full extent of the available blending space, as well as the current location of the blending point, can be taken in at a glance. A further

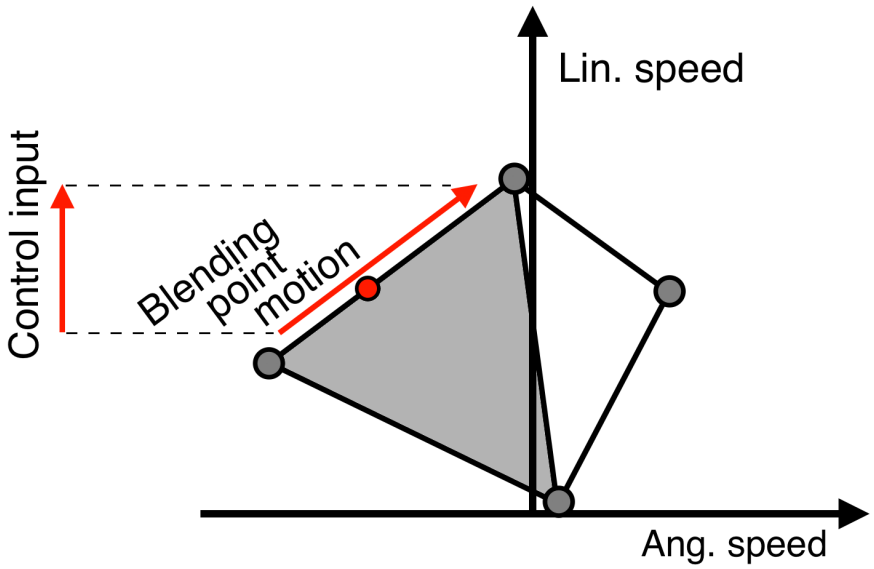

Figure 5.16: Algorithmically restricting the blending point within the network boundary eases user-operation. In calibrated networks, however, control inputs in the elementary up/down/left/right directions then no longer necessarily result in a corresponding motion of the blending point, nor in the expected response from the character. (Network simplified for illustrative purposes).

reason for interface transparency is the proficiency of rectangular networks for use with zero visual feedback, as detailed in the following paragraph.

It is useful to programmatically restrict the blending point so it always lies within the network, thus relieving the user from having to ensure that it does so. With rectangular networks, this allows the user to keep his attention on the character, as, within the bounds of the network, control inputs in the fundamental up, down, left and right directions always generate corresponding character motions (faster, slower, leftwards and rightwards), and any attempt to exceed these bounds simply caps the motion to the available range. This is not the case for calibrated networks, however, where the movement of the blending point may take a very different direction from that of the control input, as shown in Fig-

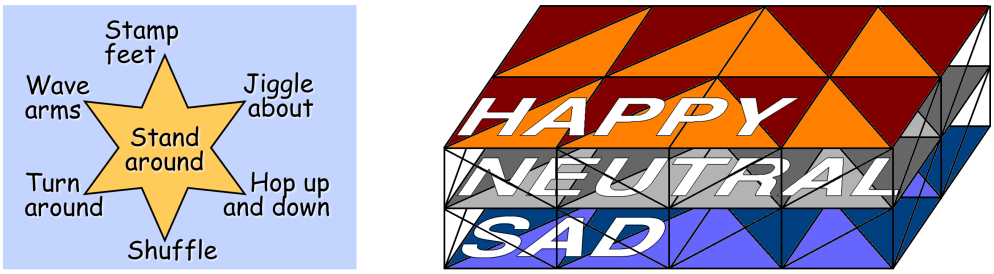

Figure 5.17: (Left) Network use in a children's game, exemplifying how the streamlined approach is not limited to rectangular networks, and how the internal triangulation need not be shown. (Right) Three-dimensional network, combining full navigation with additional control of motion style.

ure 5.16. Cropping the calibrated velocity space to a rectangular subspace within it might appear to eliminate this difference between networks, but this too would have disadvantages, of which the wasting of unused blending space is but one.

While rectangular networks are ideal for navigational input, the streamlined approach is not restricted to these, and, given suitable input data, allows networks to be shaped however befits the application. Figure 5.17, left, illustrates a star-shaped network as might be used in a children's game, where the points of the star represent different activities (and where, since locomotion is not the aim, the non-rectangular shape is no impediment). Another use of manual placement for the adaptation of interface design was shown in Figure 5.12, bottom, where appropriate positioning of vertices enabled rudimentary control of blending point sensitivity.

Manually constructed triangle networks need not be limited to simple 2D structures, however. Possible extensions include the following.

- Three-dimensional networks, as in Figure 5.17, right, allow a continuous change between motion styles, such as happy and dejected gaits, while main-

taining the previous control over velocity and rate of turn.

- Detached networks can be seamlessly joined with natural-looking transitions, using a novel method detailed in Chapter 6. This enhances the previously isolated blending spaces as their motions can form part of greater animation sequences comprising a variety of motion types.
- Network cascading allows the *result* of a hierarchically-lower motion-blending process to form the *input* to a new triangle vertex, as was further described in Section 5.6.4. By this method, making a gait increasingly tired while climbing a hill, for example, could be performed at negligible cost.

Freedom in vertex positioning (given suitable input motions) makes network construction in the streamlined approach a *general*, universal method, encompassing many network types of which those of Pettré and Laumond [PL06] are a special-case subset. The potential of manually constructed networks extends even beyond the above-discussed points, however. For example, network construction does not have to lie with the application developer, but can be left to the user, adding to the utility of the streamlined approach. The implemented program catered for this, with a drag-n-drop interface for vertex placement as well as for input motion allocation. The subsequently required preprocessing steps can, in fact, be performed automatically, as was implemented (as an optional feature) in the methods of the following two chapters. It is thus quite feasible for both network design and its subsequent use to be entirely user-controlled.

5.8.3 Interpolation Methods

The streamlined approach uses barycentric coordinates as blending weights, as they are cheap to calculate, and provide a linear – and hence predictable – response to blending

point movement within the occupied triangle. Only that triangle need be considered, as wherever the blending point is placed within it, the three input sequences at the triangle vertices contain all the necessary information needed to create the corresponding output motion.

However, alternative interpolation methods could be considered. For example, many networks, like those of Figure 5.12, have triangles arranged in a rectangular grid, for which bilinear interpolation appears fitting. Additionally, the choice of diagonal direction, as illustrated in the figure, would then become redundant. Similarly, for 3D networks (Figure 5.17) trilinear interpolation might be used.

The streamlined approach experiences no discontinuity in the skeletal pose when crossing triangle borders. However, motion style *gradients* can vary from triangle to triangle, so fast sweeping movements of the blending point could result in a discontinuity, at triangle borders, of the *rate* at which the animation style is changing.

This unwanted effect (reproducible, though not to be seen in thesis videos) should be alleviated by blending between a greater number of motions at a time. However, as disparate movements cannot be successfully blended, interpolation should only merge motions which are – by some selected heuristic – deemed sufficiently close within blending space. Natural neighbour is one such method, though the cost of repeated Voronoi tessellation would need to be evaluated. A blend of the k nearest neighbours appears cheaper, and, in the context of streamlined triangle networks, partly computable in preprocessing, though an appropriate runtime weighting scheme would be required. Radial basis functions, as used, for example, in [RCB98, SRC01] could also be considered.

5.8.4 Synthesis Cost

The implementation written for this chapter (in C/C++ and OpenGL) assigns a thread to blending and Fourier synthesis. Its impact on frame rate is effectively nil, as the

animation frequency is sharply capped to the rate specified in the motion capture data, and even if it were not and instead was allowed to run unimpeded, the program would vastly exceed this rate, with, it was found, negligible slowdown from the parallel-running thread. This threading, as so lightweight, was enabled continuously (with one exception below), computing *entire walk cycles* in succession, and did so fast enough (Table 5.1, above) for responsive motion cycle updates upon changing user input. Assuming use of this set-up, thus computing complete cycles instead of single frames, the streamlined approach gives rise to the results of Table 5.1 and also to the relative savings in Table 5.2, the reasons for the latter having been explained at the end of Section 5.3. Furthermore, assuming a user-driven skeleton, the thread can immediately be killed whenever mouse or joystick input ceases, whereupon the skeleton pursues its animated course at zero cost, representing a betterment over tabulated results. Calculating entire walk cycles also provides information useful to various program modules, and has potential for the simulation of crowd scenes, as skeletons can be loaded with complete motion-sequences, and left to enact them endlessly until requiring navigational changes, for example, for collision avoidance.

A quite different approach is to perform Fourier synthesis for a *single* frame only, synthesising only that required at each time step, as did [PL06], and which the streamlined approach can equally do following trivial adjustment. In this case, the above-demonstrated results remain valid, but they do require a different interpretation, as unlike the scenarios of the above paragraph, it is then *all* cost savings which depend on the breaks observed in typical user input. Thus, for example, when mouse movement stops, frame-by-frame *blending* can cease, as the already-blended Fourier coefficients describe the *entire* motion cycle which is gradually being executed, and accordingly, the arctan2 function (associated with phase angle blending, Section 5.3) is temporarily dispensed with, reducing the number of trigonometric functions employed in contrast

with the method of [PL06]. If user-input remains in abeyance, the relative cost savings accumulate as the motion is computed time step by time step at the animation frame rate, until a complete cycle has been built, at which point the same results and relative savings are reached as found in Tables 5.1 and 5.2. If user (or control module) input continue to remain absent, further synthesis would merely duplicate previously computed DOF-values, so the streamlined approach would then suspend synthesis with its cost falling to zero, while the skeleton walks on unabated. In contrast, in the words of Pettré and Laumond [PL06], "while the user's directives remain unchanged, given the animation timing and the analytical expressions of the synthesised locomotion, a posture can be immediately deduced", suggesting, by their need to make use of "analytical expressions" while mentioning no exceptions to this, the ongoing computational expense of their method in the absence of user input.

5.8.5 Harmonics

As quantified in Table 5.3, the average node position error during synthesis with three harmonics is 5 mm. Static objects around 5 mm in size may, however, often be discerned in foreground views, and, given the extreme sensitivity of the human visual system in its effortless judgement of motion patterns [CHP89, MCC09, GMPO00], it seems reasonable to assume that aberrations of this size in skeletal trajectories might well sometimes be noticeable too. The error value under 1 mm for 10 harmonics, however, is clearly very small and surely of little significance, if any. The measured values thus appear consistent with the subjective assessments in Section 5.7.6 of the number of harmonics required for sufficiently high-quality motion.

While evaluations of the perception of movement can be found in [MLD+08, Tro02, KC77a, KC77b], these papers, while explicitly focusing on issues of perception, do not address the issue of concern here.

5.8.6 Level of Detail

Level of detail (LOD) adjustments for this, as well as for previous work, include limiting the number of DOFs in distant characters, and reducing the harmonic count for DOFs which are retained. In addition, the streamlined method, with its inherent resampling ability, makes it easy for distant characters to be animated at a lower frame rate, while still traversing the terrain at the same speed (analogous to increasing the time step in the method of [PL06]). Combining such LOD measures with the comparatively lightweight streamlined approach makes frequency domain methods more amenable to the simulation of crowds. An avenue for further research would be to map such character animation to the GPU, since Fourier synthesis can be applied in parallel to the many data elements obtained via the DFT, and, with a high ratio of arithmetic operations to memory usage, is well suited to parallel processing [NVI12], enabling this method to be further exploited in crowd simulations. The inclusion of random noise [PG96, Per85] could also be used to add subtle variety [MLD+08] to the monotonicity of cyclified motion.

5.8.7 System Overview

The streamlined approach is enhanced in Chapter 6, and further still in Chapter 7, with a brief overview of the final resulting algorithm being given in Appendix C. In this view, the position of the preprocessing and runtime steps presented in this chapter, as well as the associated equations, can be seen in the context of the overall system.

5.9 Conclusion

The proposed Fourier blending method has been compared to previous work, especially to that of [PL06], which is the most similar. The streamlined approach is more efficient,

as confirmed by results focussing on Fourier synthesis, with the prerequisite for this efficiency gain being the application – not previously seen in frequency domain character animation blending – of the proper technique for phase angle blending. Further benefits, including runtime cost savings, were clarified throughout the implementation-level description of the presented system. The flexibility in the streamlined approach to triangle network construction makes possible the blending of motions for which the method of [PL06] could not create networks at all, with further advantages like adjustable blending point sensitivity and the use of networks as intuitive user-interfaces. The streamlined approach is well suited to real time interactive simulations and games, and while not intended for the motion planning context [PL06] focussed on, it can be used to automatically follow a pre-defined path.

6

Hybrid Networks

6.1 Introduction

The Fourier blending approach of Chapter 5 and presented in [MLD10], belongs to the category of interpolation-based synthesis, as was used for skeletal animation in [KG03, GBT04a, WH97] and as part of hybrid approaches in [HG07, SO06]. Interpolation has the advantage of creating an infinitely fine gradation of potential output motions, by applying corresponding weightings to the inputs. While successful when blending similar input sequences, results increasingly tend to look unnatural, generate unwanted artefacts, and eventually fail, as input motion similarity is reduced. Furthermore, even when blending is feasible, transitioning between motions of different style using time-varying weightings often looks less convincing than a motion captured sequence depicting that very change.

Methods which rearrange motion frames, on the other hand, such as [TLP07, LCR+02, LCL06], provide an alternative which can readily integrate varied motions, as long as the database comprises sufficient data to enable continuous joins between concatenated sequences. Motion variety is thus superior to that of blending-based methods, but the range of potential output motions is granular in the sense that it lacks the endlessly smooth control provided by interpolation.

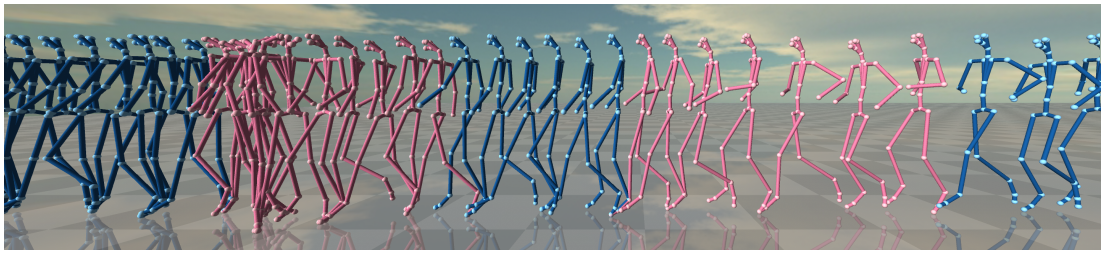

Figure 6.1: Hybrid network in action generating, with smooth continuity, running motion (blending-based) followed by a run-to-walk transition (playback-based), then walking (blending-based), a walking-twirl (playback-based) and finally walking (blending based).

This chapter addresses these issues with an elegant extension to Fourier triangle networks, giving them the advantages of both types of motion synthesis. Previous blending-only networks acquire the ability to play back motion clips, and to switch between these and blending-based synthesis in a smooth and intuitive manner. The former blending networks are augmented by the inclusion of arcs between input motions, representing sequence playback bridging one motion to the other. The played-back sequences, termed *transitions*, are seamlessly integrated into the existing network by the introduction of *buffer triangles* (Section 6.4.4). These enhancements create a novel structure, the *hybrid triangle network*, which adds considerably to the utility of previous blending networks. Potential motion variety is greatly increased. Additionally, a natural-looking transformation, progressing from one network to another is now possible, even where these synthesise motions of significantly different style. A hybrid network was implemented to accompany this chapter and can be seen in the corresponding demonstration video (URL given in Results section and Appendix E). Figure 6.1 shows the network generating smooth continuous motion as it repeatedly switches between blending-based synthesis and playback.

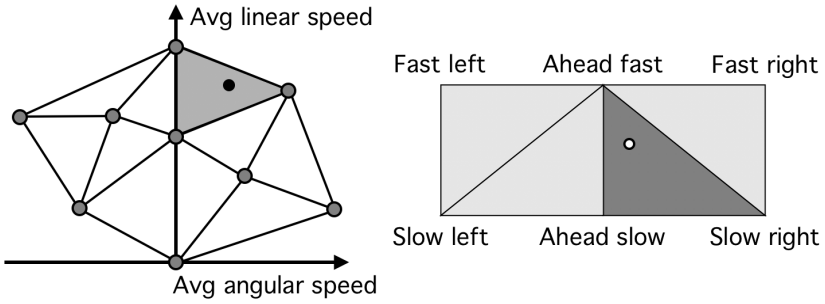

Figure 6.2: Triangle networks used in previous work. Left, a calibrated network as used by [PL06] for motion planning. Right an interface-style network, employed by [MLD10] for user-controlled animation of interactive applications.

6.2 Context and Scope

The functioning and benefits of previous triangle networks [PL06, MLD10] were detailed in Chapters 4 (Section 4.7) and 5. Both network types created a blending space in which the location of the blending point specified an interpolation to be performed on input data in the frequency domain. Pettré and Laumond [PL06] had employed automatic positioning of triangle vertices creating a calibrated network appropriate to the motion-planning context of their work, while in the presented streamlined approach, Chapter 5 and [MLD10], the triangle network was fashioned into an interface device by the manual placement of triangle vertices. A reminder of each network type is shown in Figure 6.2. The enhancements presented in this chapter could be integrated within either network type, but are illustrated in the context of the interface-style networks of Chapter 5 and [MLD10]. To avoid duplication, the operation of these networks is not elaborated anew, and the scope of this chapter is strictly confined to the novel incorporation within them of playback-based transitions. Specifically, the contribution here will be to show how a captured sequence can act both as input motion for blending followed by Fourier synthesis, and also serve as source data for motion playback, while ensuring continuity within the transition sequence and its seamless integration within the blending network.

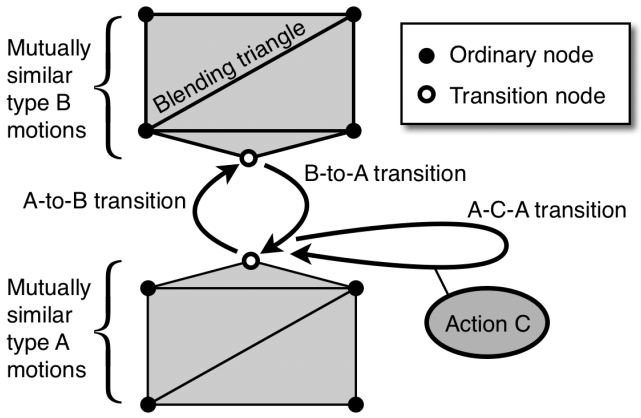

Figure 6.3: Dual-purpose nodes in hybrid networks define blending triangles, while also supporting playback-based transitions represented by arcs.

6.3 Transition Types

6.3.1 Transition Definition

To avoid ambiguity the term 'transition' requires definition. It refers here to an arc representing motion playback of a specific action, and thus bears some resemblance to graph theory, while making no reference to the alternative definition, that of a time varying blend from one motion to another.

6.3.2 Components of a Hybrid Network

In hybrid networks, pre-processed cyclified input motions are represented by dual-purpose nodes. Firstly, they define the vertices of a blending triangle network, as recapitulated in Section 6.2, above, whereby in the streamlined version which the hybrid networks presented here are built upon, nodes have been placed as desired to create a user-interface suitable for the intended application. Secondly, nodes *can* (but need not) be joined by unidirectional arcs. An arc represents a transition between the motions associated with its source and destination nodes -known as transition nodes- during

which synthesis results from motion playback instead of blending. This is illustrated in Figure 6.3. Source and destination may be either distinct or one and the same node, leading to the two types of transition described below.

6.3.3 Inter-network Transitions

Inter-network transitions move the blending point from one triangle network to another. Such transitions are required when interconnecting networks that synthesise disparate motion types, since bridging using blending triangles would fail, as would any interpolation-based method blending between markedly dissimilar motions. Inter-network transitions can also increase motion quality, as even when network similarity makes blending feasible, a change from one motion type to another using time-varying blending weights often looks inferior to a motion captured sequence enacting that very change. These issues are discussed further in Section 6.8. Two transitions of this type, walk-to-run and run-to-walk, as demonstrated in the implementation for this chapter, serve as examples of the great number and variety which networks can accommodate.

6.3.4 Intra-network Transitions

Intra-network transitions are cyclic with the arc returning to the node it originated from. The process thus starts with one motion, and having played an intermediate sequence, ends with the same motion it started with, although as will be seen, this final motion is strictly speaking merely of identical *style* to the first.

The purpose of intra-network transitions is to add variety to triangle networks. Pirouette-style motions were chosen to provide examples of this, which, as seen in Figure 6.4, include a walk-twirl-walk sequence in the walking network and a run-twirl-run transition in the running block. Given the prerequisite motion capture data – one clip per transition – any sequence can be integrated in this manner, no matter how much

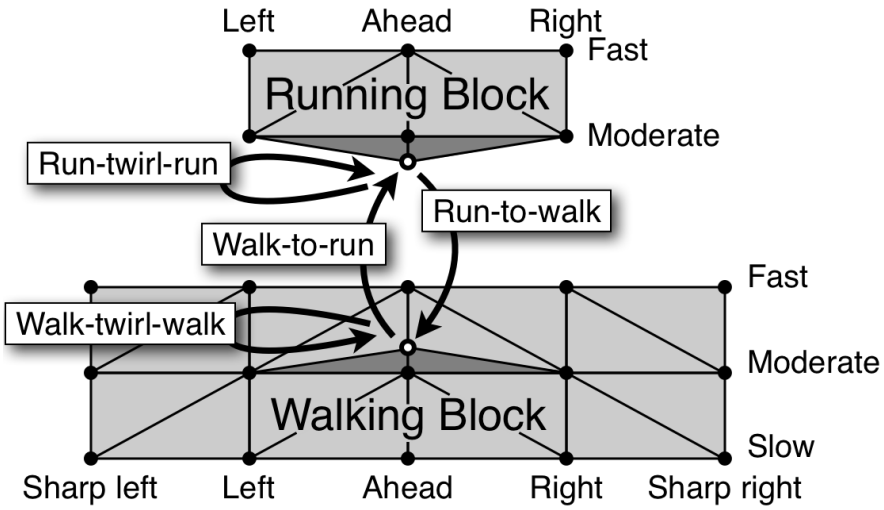

Figure 6.4: The hybrid network shown in the accompanying video. Buffer triangles – covered later – are shown highlighted.

it may differ from the motion being blended, thus cartwheeling, a headstand, or a swimming motion would be equally feasible. Two intra-network transitions are demonstrated, while emphasising that the number which networks can support is effectively unlimited.

6.4 Transition Structure and Operation

Seamless integration into existing blending networks is a key requirement of transitions, dictating their structure and operation as found described in this section.

6.4.1 Transition Phase Evolution

A transition originates from a single raw motion capture clip comprising three phases (Figure 6.5, top). Once pre-processed (in centre of figure), this yields three distinct motion sequences: phase 1 and 3 pre-processed sequences which are cyclified and assigned to network nodes, and phase 2 which is not. Similarly, the DFT is performed only on phase 1 and 3 thus representing them in the frequency domain like any other

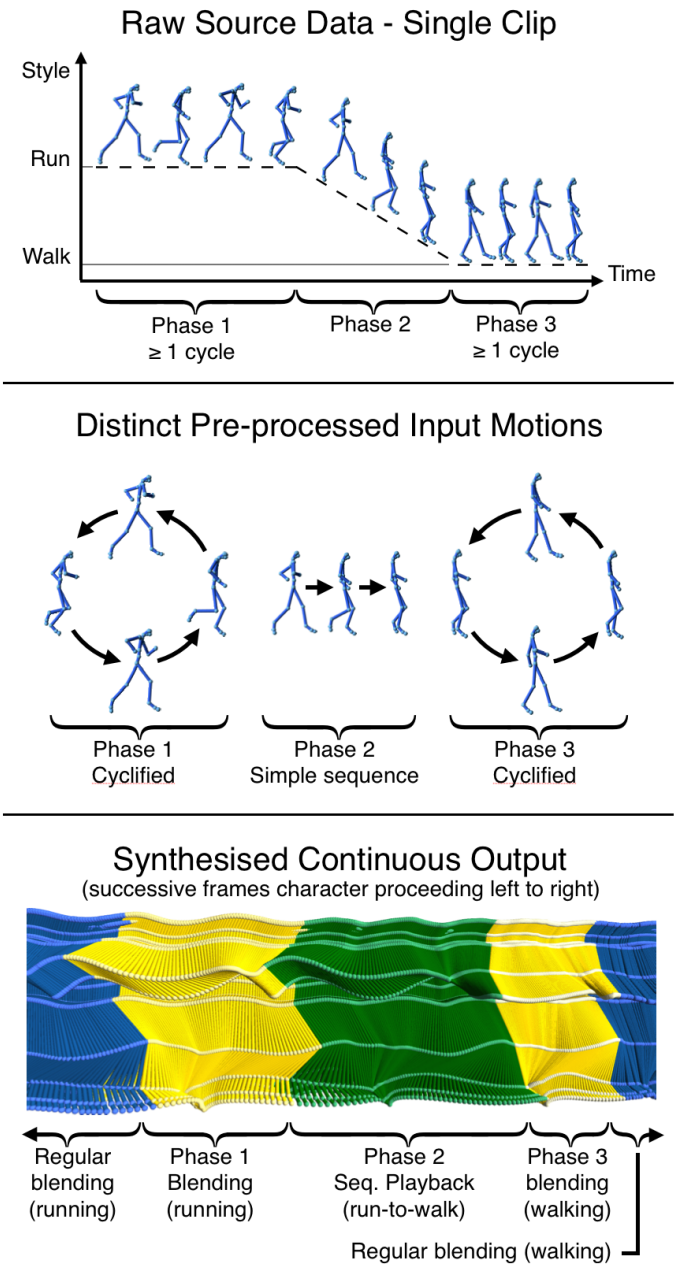

Figure 6.5: Walk-to-run example illustrating the evolution of the three phases of a transition, existing first in a single raw input motion clip, later as three distinct pre-processed input motions, and finally during transition execution, generated in sequence as synthesised output.

input motion, prior to blending and Fourier synthesis. During transition execution at runtime (bottom of figure), blending-based synthesis first generates phase 1 output. Phase 2 output is then obtained by sequence playback, followed by phase 3, synthesised in a similar manner to phase 1. Three distinct phases thus exist both within the source data, and, correspondingly, in the stages of motion production.

6.4.2 Transition Source Data Requirements

A raw capture sequence must honour three prerequisites to yield a high quality transition. These are now enumerated with reference to the run-to-walk example at the top of Figure 6.5 (requirements for intra-network transitions are similar). The raw data should begin with at least one full cycle of running motion and should similarly end with at least one complete cycle of walking movement, leaving a central part with frames depicting, and only depicting, a continuous progression from run to walk. This ensures phase 1 and 3 in their raw state can be cyclified during preprocessing and anchored into the blending network whose operation requires cyclified input motion. Phase 2 will not be directly embedded in the network, its requirements merely serving to ensure high motion quality and prompt execution at runtime.

6.4.3 Phase-Phase Continuity

Continuity of motion within a transition is maintained at all stages of motion creation.

- Phase 1, 2 and 3 in their raw state (Figure 6.5, top) occupy the same motion clip and thus possess perfect continuity.

- Input motions for *regular* blending networks as in [MLD10] and Chapter 5 were cyclified by offsetting the angular motion data for the first and last frames of each degree of freedom (DOF) by equal and opposite amounts, thereby equating their

values, and adjusting the intermediate frames using displacement mapping. To maintain phase-phase continuity in *hybrid* networks after preprocessing, however, the cyclification-method for phase 1 data is altered so as to offset DOF values at the initial frame only, leaving the final frame untouched, before applying displacement mapping to the intermediate phase 1 frames. Similarly, cyclification of the phase 3 pre-processed motion offsets its end frame only. The inherent continuity between the transition phases of Figure 6.5, centre, is thus untouched by preprocessing.

- During transition execution (Figure 6.5, bottom), phases 1 to 3 are generated in succession. As about to be shown in Section 6.4.5, phase 1 synthesised output is very similar to pre-processed phase 1 motion, while phase 3 runtime output replicates its pre-processed counterpart with yet greater fidelity. Phase 2 output remains strictly identical to that resulting from preprocessing. The continuity found in the original capture sequence thus remains all but intact in the synthesised output.

6.4.4 Buffer Triangles

Transitions are anchored to blending networks by creating a node to represent the transition's pre-processed phase 1 or phase 3 motion. Moreover, for each such node, two pre-existing network nodes are selected whose assigned motions resemble that of the transition node. Thus, phase 1 of the run-to-walk example requires the creation of a new node, to be associated with two of the network's existing running motions. A new triangle is then formed, allowing runtime blending of these three motions in any desired proportion, thus allowing a smooth change from any existing blended output to one based entirely on phase 1 preprocessed motion. For this reason no clear demarcation

exists between regular and phase 1 blending, and perfect continuity is assured, as applies equally to phase 3. Such triangles act as a buffer, connecting transitions to the pre-existing blending network, and shall thus be known as 'buffer triangles'. Four such triangles are shown in Figure 6.4.

6.4.5 Runtime Execution

As the user drives the blending point towards a buffer triangle's transition node, a *trigger zone* is entered once the weighting for that node reaches or exceeds the empirically established value of 0.95. The synthesised output then closely resembles pre-processed phase 1 motion and can thus form a smooth-looking join with the phase 2 playback section. In essence, playback can then start as soon as phase 1 synthesis reaches its sequence end frame, though in practice motion synchronisation must be accounted for to ensure continuity, as explained in Section 6.5.

When phase 2 playback starts, blending ceases. The user-driven blending point then becomes meaningless and is removed from the user interface until the final playback frame is reached, whereupon it reappears in the destination buffer triangle positioned over the transition node, corresponding to a weighting of 1.0. Resulting phase 3 synthesis then, at the next time step, all but replicates the cyclified motion of phase 3 after preprocessing (as stated in Chapter 5 Section 5.7.6 three to maximum eight harmonics during Fourier synthesis replicate motion with no perceived inaccuracy). For a smooth connection, synthesised phase 3 motion could start with its first frame, if it were not, as before, for the effect of motion synchronisation on the operation of transitions.

6.4.6 Input Motion Reassignment

The arcs of intra-network transitions originate and end at the same network node. The associated animation, however, starts with synthesised phase 1 motion and ends

with phase 3, motions which while of the same type (eg walking) are based on distinct cyclified input sequences, and are therefore not identical. Input motion reassignment is thus required during phase 2 playback. It allocates phase 3 to the common node before completing the transition.

By the time phase 3 synthesis commences the blending point has reappeared and the user can take over, guiding it to modulate the animation. As it exits the buffer triangle, phase 1 motion is automatically reallocated, thus priming the triangle for future use. Being no longer occupied, however, the sudden change of input motion cannot affect synthesis, which remains free of discontinuity despite the reassignment.

6.5 Motion Synchronisation

Motion synchronisation impacts on the functioning of transitions and must thus be accounted for in their creation, as will now be elaborated after an explanation of the process itself.

6.5.1 Synchronisation Procedure

Motions such as walking or running must be aligned prior to blending. The approach used in Chapter 5, Section 5.4.6, avoided both the complexity of dynamic time warping [KG04, BW95, SO06] and the limitation in [PL06] of having to select input motions which move in step. It operates, as before, by shifting the values of all DOFs (mostly joint angles now in Euler or quaternion representation) of one cyclified motion relative to those of another, the reference motion. The process is one of rotation, with values from one frame being shifted to the neighbouring lower-indexed frame, and those pushed out of the sequence start being re-inserted at the end, or the reverse for rotation in the opposite direction. The most favourable degree of rotation, or synchronisation offset, is established via the pre-existing synthesis mechanism, by adjusting the offset

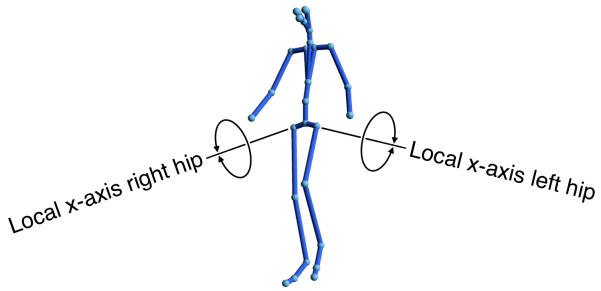

Figure 6.6: The body-coordinate x-axes around which hip joint rotations are measured during automatic motion synchronisation.

while monitoring the result of a 50/50 blend between the sequence being aligned and the reference motion. While [MLD10] and Chapter 5 employed visual monitoring, an automated evaluation process is now proposed, which was used to synchronise the motions of all nodes in the implemented hybrid network.

6.5.2 Automatic Synchronisation

While synchronisation seeks the offset giving the best-looking blend, the metric used for automatic synchronisation seeks – as justified below – that giving the worst, from which the offset for best quality motion can then be deduced.

The local-coordinate x angles (Figure 6.6) of the left and right hip joints at the n^{th} frame of synthesised output are denoted X_{lnO} and X_{rnO} respectively, where O is the synchronisation offset under consideration. The total difference between hip angles is accumulated over each frame of the synthesised sequence, repeating for each possible offset value.

$$X_{accum\ diff\ O} = \sum_{n=0}^{N-1} |X_{lnO} - X_{rnO}| \qquad \forall O \in [0, M) \tag{6.1}$$

where N is the length of the synthesised output and M that of the sequence being synchronised.

Tables of accumulated differences – of which Table 6.1 in Section 6.7 (Results) is an example from the hybrid network implementation – exhibit a sharp and algorithmically detectable drop in the value of $X_{accum\ diff\ O}$ for a limited number of contiguous offset values, with the middle of this range, O_w, corresponding to the *worst* or most *out-of-synch* blend. The underlying heuristic reflects behaviour observed when blending pairs of walking or running motions, which if made increasingly out of synch, suddenly generate a blend which tends to move both feet forward and backwards simultaneously, instead of placing one foot in front of the other.

The offset for *best* synchronisation is then obtained by shifting the motion being aligned by half a cycle, as given by the following formula.

$$O_b \quad = \quad \left\lfloor O_{w\ avg} + \frac{M}{2} + 0.5 \right\rfloor \bmod M \tag{6.2}$$

where the average, $O_{w\ avg}$, is taken of two worst-offset values, O_{w1} and O_{w2}, prior to half-cycle adjustment and rounding, as adding a node to a network involves the creation of a triangle with *two* pre-existing (mutually synchronised) nodes. $O_{w\ avg}$ must take account of the circular nature of offset values (an offset of M frames equals zero offset) and is given by

$$O_{w\ avg} = \begin{cases} \frac{O_{w1}+O_{w2}}{2} & |O_{w1} - O_{w2}| < \frac{M}{2} \\ \frac{O_{w1}+O_{w2}-M}{2} & |O_{w1} - O_{w2}| > \frac{M}{2} \end{cases} \tag{6.3}$$

whereby $|O_{w1} - O_{w2}| = \frac{M}{2}$ does not occur in practice.

6.5.3 Synchronisation in Transitions

Transitions require the flexibility to smoothly join with blending networks, no matter what the synchronisation offset of existing network nodes. To this end the cyclified phase 1 and 3 pre-processed motions are synchronised with the two motions they share

a buffer triangle with, but the consequent reallocation of DOF values to different frames breaks the transition's inherent phase-phase continuity. More precisely, values which were initially at the end of the phase 1 input sequence will have been offset to another frame. It is thus this frame which is able to smoothly adjoin phase 2, and at runtime, the animation of this frame in the *analogous synthesised phase 1 sequence* (Section 6.4.5) which should be reached prior to phase 2 playback. Similarly, come the end of phase 2 playback, phase 3 animation should start with that frame which corresponds to the first frame of phase 3 *prior to* synchronisation.

When taking account of motion synchronisation, the steps in a transition become those in the following pseudocode.

TASK: perform transition

ASSERTION: if trigger zone already entered, blending and synthesis thread will have been killed as temporarily redundant. Transition status will have been set to Phase 1.

// Performed at *each* time step after incrementing frame counter (Ch. 5, Sect. 5.5.3).

SELECT transition status

CASE Transition off:

// do nothing

CASE Phase 1:

IF $frame\ counter = frame_{last} - offset_{Phase\ 1}$

Hide blending point.

Set status to Phase 2A.

END IF

CASE Phase 2A:

Copy Phase 2 data into skeleton.

Set frame counter to 0.

Set status to Phase 2B.

CASE Phase 2B:

IF *frame counter* = *last frame*

Move blending point to destination node.

IF intra-network transition

Assign Phase 3 input to source/destination node.

END IF

Set status to Phase 3.

END IF

CASE Phase 3:

Copy Phase 3 data into skeleton.

Set frame counter to

$(duration_{Phase\ 3} - offset_{Phase\ 3})$ mod $duration_{Phase\ 3}$.

Enable blending and Fourier synthesis thread.

// (Copied data will be used until thread takes over).

Set status to transition-off.

END SELECT

// Following code builds and displays skeleton.

The location of the above pseudocode within the overall system algorithm is shown in Figure C.1 of Appendix C, which uses colour coding to differentiate between the streamlined approach and the components which are added by hybrid networks. (The diagram also includes the enhancements discussed in the following chapter).

6.6 Interface

6.6.1 Unifying Nodes

While a single buffer triangle is sufficient to anchor the start or end of a transition, the interface is sometimes improved by using pairs of such triangles at each transition node, as evidenced in Figure 6.4 which has two such pairs, one pair at the transition

node in the running block and the other pair in the walking block. Despite doubling the number of buffer triangles compared to 6.3, the number of transition nodes remains unchanged, and as before, each outgoing or incoming inter-network transition, and each intra-network transition, is represented by one arc.

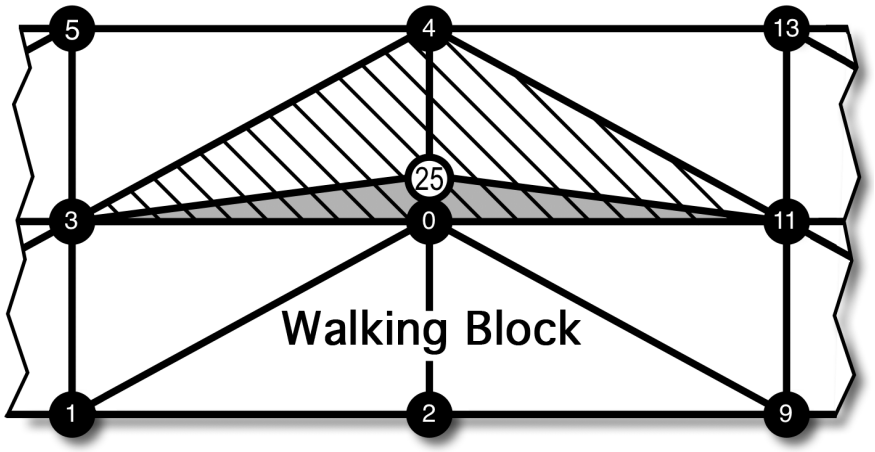

Figure 6.7: The non-arbitrary nature of buffer triangle placement can lead to triangle overlap, as applied to those assigned motions {0,4,3}, {0,25,3}, {0,11,4} and {0,11,25} in the implemented network. Disambiguation is needed at runtime to determine triangle occupancy.

6.6.2 Overlapping Triangles

When forming a buffer triangle, the conglomeration of transition phase 1 or 3 motion with two pre-existing network nodes requires all three to be of sufficient similarity to allow satisfactory blending. Buffer triangle placement is thus not arbitrary. This can lead to overlapping triangles in the network (Figure 6.7), giving rise to ambiguities at runtime when determining which triangle is occupied by the blending point. Furthermore, buffer triangles can be gateways to a choice of transitions. In order to disambiguate user intentions, the interface is given several operating modes which change both its

appearance and the nature of the network. The implemented hybrid network employed three modes, as enumerated in the next section.

6.6.3 Operating Modes

For brevity, the following explanation is limited to the walking block, although the three-mode approach is used throughout the hybrid network. With reference to Figure 6.8, in the first mode, buffer triangles are ignored and the entire walking block is available for blending. In the second, the blending point is restricted to the middle group of triangles, including the buffer triangles. The latter are then active, while those they overlap are disabled. Restriction to this central group of triangles ensures the blending point can only enter a buffer triangle from below, thus via a normal triangle-triangle shared edge, which avoids the abrupt change in motion quality which would result if it were entered via a triangle it overlaps. The third mode is similar to the second, but primes buffer triangles for the walk-twirl-walk transitions instead of walk-to-run and run-to-walk. While a description is inevitably unwieldy, the interface changes mode with a single key-press, and, as shown in the demonstration video, remains intuitive and fluid in practice.

6.7 Results

6.7.1 Motion Quality

The functioning of both types of transition is shown in the demonstration video which can be downloaded from `http://www.urbanmodellinggroup.co.uk/hybridnetworks.mp4.zip`. The run-to-walk transition and both the pirouette motions where based on source data which largely fulfilled the specified prerequisites (Section 6.4.2) and transitions are seen to look natural and fluid. Run-to-walk, for example, exhibits an idiosyncratic tendency to relax the arms and lean backwards during the change, and

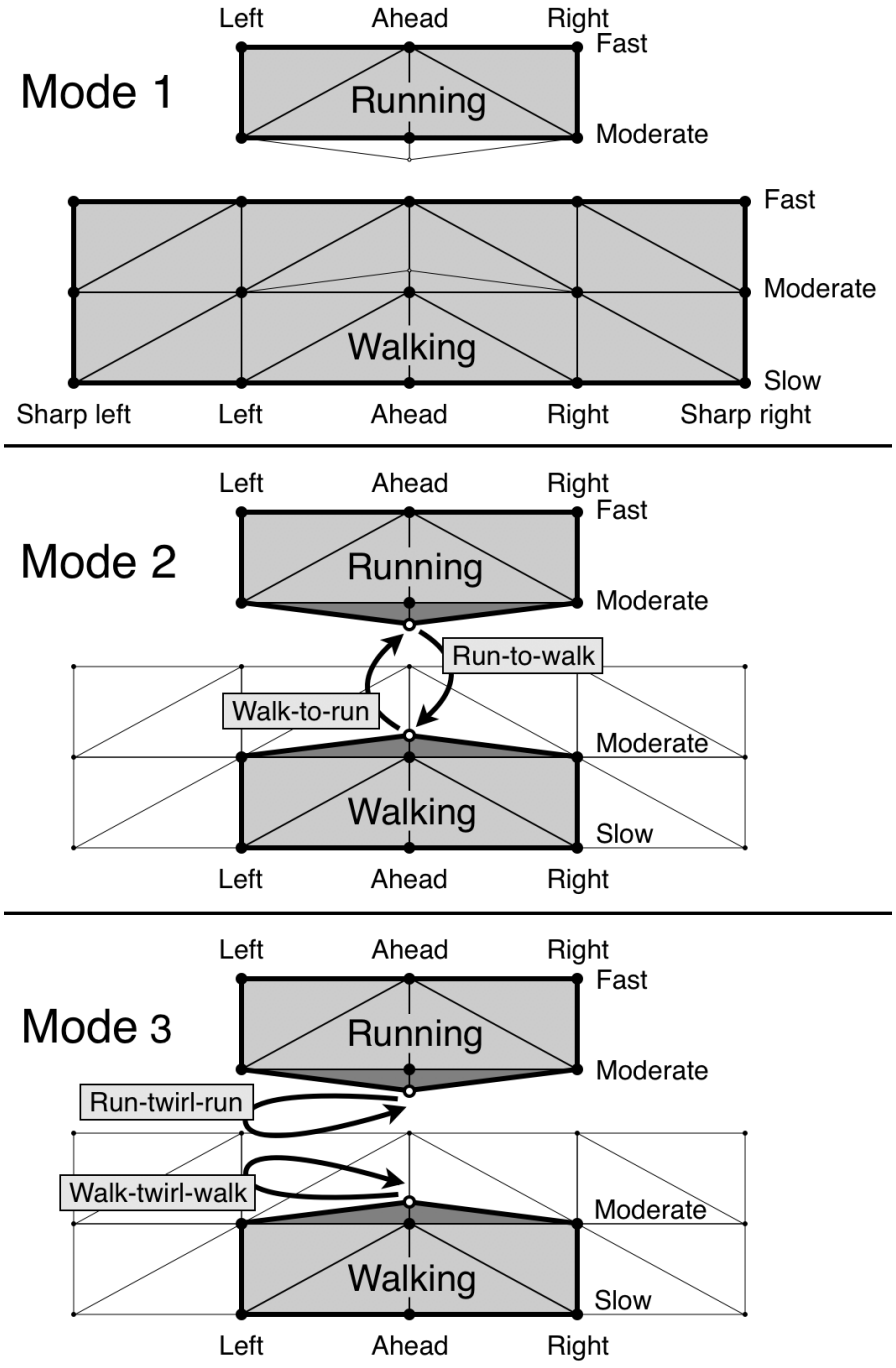

Figure 6.8: Operating modes used to disambiguate selection of overlapping triangles and nodes supporting multiple transitions. Triangles shown in white are disabled.

a realistic timing of the unfolding sequence, conveying a realism surpassing that of a blending-based change.

6.7.2 Interface

The interface – a key part of hybrid networks – is seen to provide the same intuitive control of the animation as applied in [MLD10]. New to this work is the passage through transition arcs, shown to occur promptly and smoothly after reaching the trigger zone. It can be seen how the mouse-driven blending point is algorithmically constrained to never cross the bold border which demarcates where the operating modes allow it to be. Compliance thus imposes no burden on the user.

6.7.3 Building Transitions – Synchronisation

Motion synchronisation is required to build both transitions and the blending networks they enhance. The data of Table 6.1, taken from the implemented network, is typical of all its motions, and indicates the feasibility of the presented automatic synchronisation process (Section 6.5.2).

6.7.4 Latency

Moving the blending point to a trigger zone to launch a transition cannot always be done abruptly, as a sudden change in motion style may look unnatural. A delay is thus unavoidable and depends entirely on the original position of the blending point and the location of the transition node. In the running network with only a single transition node, moving the blending point from, for example, a medium-speed moderately-sharp turn to the trigger zone requires 0.2 seconds to avoid excessive abruptness, though the user is free to take longer. Moving it from a medium-speed straight-ahead run – a motion which closely resembles that of phase 1 – could be done as fast as the user is able to move the mouse, 0.02 seconds being a typical limit.

Table 6.1: Automatic motion synchronisation data for node 13 of the implemented network (Figures 6.4 & 6.7) when offset by amount O_{13} with respect to nodes 4 and 11. The sharply reduced values shown in bold indicate an average offset for *worst* synchronisation of 78. As explained in (Section 6.5.2), motion 13, of length 102, consequently exhibits *best* synchronisation with an offset of 27.

O_{13}	$X_{accum\ diff\ O}$ (O_{13} *wrt* 4)	$X_{accum\ diff\ O}$ (O_{13} *wrt* 11)
0	56.81	55.66
1	56.77	55.68
2	58.01	55.69
...	...	...
76	56.77	55.33
77	56.77	**8.46**
78	**7.28**	**9.44**
79	**7.20**	55.62
80	57.29	55.60
...	...	...
99	56.78	55.58
100	56.79	55.43
101	56.83	55.64

Once the blending point is positioned in the trigger zone, the mean delay before phase 2 synthesis, $MD_{phase\ 1\ to\ 2}$, is

$$MD_{phase\ 1\ to\ 2} = \frac{Length_{synth\ phase\ 1}}{2} time\ step \tag{6.4}$$

which is 0.38s for run-to-walk and 0.39s before a running twirl. When transitioning from the walking block, with inevitably slower walk cycles, the mean delay is 0.64s for walk-to-run and 0.67s before twirling.

The delay before transitions has some similarity with real-life motion. It is the only one which exists, phase 3 synthesis follows phase 2 without delay. Reference is again made to the demonstration video, which gives no impression of undue latency when transitions are shown.

Table 6.2: Blending and Fourier synthesis times for the hybrid network running in Euler angle mode, using increasing numbers of harmonics. Values are per DOF and per frame.

Harmonics	1	2	3	4	5	10	20
Blend. (ns)	1.97	3.01	3.95	5.08	6.10	11.5	21.7
Synth. (ns)	86.1	168	259	335	427	849	1760

Table 6.3: Costs for blending triangle operation in the hybrid network, running in quaternion mode. Times shown are for each dof of each frame.

Harmonics	1	2	3	4	5	10	20
Blend. (ns)	2.59	3.92	5.28	6.74	8.17	15.0	28.9
Synth. (ns)	115	208	311	402	500	977	1967

6.7.5 Runtime Cost

Runtime expense in the blending triangle network is the cost of blending and Fourier Synthesis. Although not the focus of this chapter, the times for the hybrid network are given for the sake of completeness. Both are reduced by using Euler angles where possible, these being quite suitable for walking motions. However, running motions, whose joint angles may pass through 90°, require more expensive quaternions to avoid discontinuities which are not compatible with the DFT/Fourier synthesis procedure. Tables 6.2 and 6.3 provide costs for each mode, the most suitable of which is automatically switched to during phase 2 of inter-network transitions. Blending times shown include triangle selection and weighting calculation. Measurements refer to the creation of a 140 frame output sequence depicting a 69 DOF skeleton, with cost expressed per DOF and per frame. As was explained in Chapter 5 cost applies only when blending weights change, remaining zero at all other times.

Blending with triangle networks is an $\mathcal{O}(1)$ process. Establishing which triangle to use for blending mostly requires merely testing that occupied on the previous time step,

failing which only neighbouring triangles, whose indices were stored in preprocessing, need be tested.

The demonstrated hybrid network uses foot constraints to move the skeleton in 3D space while avoiding footskate. In its most basic form the process is too short to measure. However, as an *optional* feature, the indices of frames at which feet first contact or leave the ground can be blended too, and the result used to precisely set foot elevation at runtime. This feature, useful for close ups of foreground characters, adds 6µs to the total sequence blending time, or $<$ 1ns to each blending time shown in the Tables 6.2 and 6.3.

Phase 2 playback is cheaper than blending, so transitions add no cost to triangle networks. The time for operating mode selection is one during which the user interacts with the application, and is hence not perceived as any delay. In the implemented network, this duration was that for, at most, two successive key-presses. As these occurred during navigation, the demonstration video shows no interruption while the user switches operating mode.

All results were obtained on an Apple MacBook Pro *laptop* with Intel Core 2 Duo CPU running at 2.53 GHz, with 4GB RAM, and code compiled with gcc.

6.8 Discussion

The presented method is seen to work well. Blending and transition playback are integrated seamlessly, merely requiring navigation of an intuitive interface to drive the skeleton.

6.8.1 Transition Quality

The run-to-walk transition and both twirling motions were based on data which *mostly* fulfilled the prerequisites of Section 6.4.2, and correspondingly looks especially realistic.

In contrast, the source data for walk-to-run was very poor, with only a fraction of the stipulated whole cycle being present in the raw data for phases 1 and 3, yet despite this, the resulting transition still looks reasonable. In comparison, in *blending*-based transitions in previous work, "the transition from running to walking by the method is unnatural" [UAT95]. Even [KG03], thus leading research, shows the difficulty involved, by transitioning from walk to run with the arms held overly straight and distant from the body[1] (possibly to avoid self-collision), and also lingering in what in reality should be a momentary phase between walk and run, both of which failings prove unnatural when carefully compared with real-life motion. Such *blending*-based transitions between walking and running are generally borderline cases, while others, like a blend between a run and a head-stand would clearly fail. Yet with the presented mechanism, given source data prerequisites being met, *any* transition becomes possible, while entirely avoiding the issues of quality degradation at times associated with motion interpolation.

6.8.2 Source Data Requirements

Hybrid networks require suitable source motions, as do all synthesis methods reliant on motion capture, but the number is low at merely one per transition and one for each vertex in the blending network. Unlike the hybrid method of Heck and Gleicher [HG07], for which "we cannot represent transitions between two nodes if there is any motion in the source node that cannot transition to the target node", blending spaces can *always* be connected by transition in the hybrid method of this chapter. However, despite the greater potential connectivity, given N blending spaces (instead of the demonstrated two), only a subset of the $N(N-1)/2$ possible inter-network transitions would actually be wanted in practice, and no risk exists of any combinatorial explosion in source data

[1] Video accompanying [KG03]. Section "Application: Continuous Control", frame times 4:54 to 5:03. (Download details in Appendix E).

requirements. Furthermore, the stipulated transition source data prerequisites will present no problem for institutions and companies with access to a motion capture studio, able, if necessary, to repeat a capture in order to meet them.

6.8.3 Latency

The latency of transitions is hard to evaluate. This is partly because it tends to drop the more transitions the network contains, since the distance between blending point and trigger zone will then tend to be lower. Moreover, a real-life person will not, for example, start a run immediately when instructed to do so. They may first change their walking style or wait for a suitable part of the walk cycle, before accelerating to a run. Furthermore, a relaxed switch to a run may incur greater delay than one due to imminent danger. A comprehensive assessment of the latency of transitions would thus require extensive analysis of human motion to enable comparison. No sense of undue delay is experienced, however, when viewing the demonstrated method.

The other facet of hybrid networks is motion blending, which provides *immediate* response to user input. This is in stark contrast to the method of Heck and Gleicher [HG07] for which "we do not adjust the parameter vector while generating a motion" and "it may take time for the character to react to user requests".

6.8.4 User-friendliness

The blending part of hybrid networks is (with some added refinements) the streamlined approach of Chapter 5. Its greater user-friendliness than the *blending-only* methods of [BW95, PL06] has already been mentioned in Section 5.6.

A further comparison with the *hybrid* method of Heck and Gleicher [HG07], is that when performing node-node transitions with their method (from blending space to blending space), the user needs to select a point within a small target subspace in

the destination node, but this target box is repeatedly changing in size and in position within the destination node. Heck and Gleicher work around this problem by overriding user input and adjusting it to lie within the target, which means, however, that avatar behaviour does not necessarily follow user input. This is indeed confirmed by the authors' stating that "by limiting the transitions to good ones, our characters occasionally miss targets". In comparison, when blending in the hybrid networks proposed in this chapter, the user is always effortlessly aware of the entire extent of the blending space as well as the location of the static and hence easily selectable transition nodes, and the character will always respond faithfully to user instructions via the interface.

The method of Shin and Oh [SO06] was also a hybrid one, but as previously explained in (Chapter 4, Section 4.6.1), this required the user to select an instant on a timeline, to then select a joint on the character, and finally to reposition that joint as desired which would be quite unwieldy for use in games and for continuous real time character navigation in a virtual environment, in contrast to the simple mouse-pointer (or joystick) positioning within the hybrid networks described in this chapter[1].

6.8.5 Variety and Usage

Given the necessary source data, hybrid triangle networks can support *large numbers of transitions* while maintaining the benefits of blending. A case in point would be characters walking within a city. The precise control of direction and speed imparted by the blending network is ideal for following any desired path along a pavement, road or alley, but, additionally, transitions could be used in numerous ways. They could impart variety by allowing characters to, for example, come to a stop in a realistic

[1]The blending point could also be made to travel autonomously to the transition node upon user request via key-press, this being similar to the implemented automatic repositioning of the blending point during phase 2 playback (Section6.4.5).

fashion and look in shop windows, or move heavy bags from one hand to another. With transitions to sidestep obstacles, or walk onto or down from pavements, characters could navigate more proficiently. To animate the city, triangle network-based path-following, as demonstrated in the previous chapter and in [MLD10], merely requires automatic triggering of transitions.

The *existing* mechanism allows more variety than may be apparent. For example, the walk-to-run transition could be complemented with further transitions, connecting the *same* two nodes, but playing-back different-styled motions, such as walk-to-skip-to-run or walk-to-stumble-to-run.

The method of Heck and Gleicher [HG07] uses, without exception, blending spaces built from *many* similar input sequences to create and enact the displayed animation. Adding variety in the *presented* hybrid networks, however, conveniently requires only *one* additional sequence, which can, if desired, be quite different from any other motion in the network. It can also be of any length, unlike the method of [HG07] for which "a parametric motion space produces a short motion".

6.8.6 Modularity

Blending triangles and transitions are modular components from which hybrid networks are built. Not only can varied networks (as for cartwheeling, crawling or sitting motions) be bridged with transitions of choice, and variety be added by reflexive intra-network transitions (as, for example, when interrupting a walk to tie a shoelace), but the modular system itself provides a *transition functionality* which extends beyond the monolithic nature of the simple played-back transitions. Thus if a cartwheel were required for example, in the middle of a walking motion, a single rigid played-back sequence could indeed be used, but instead of this simple transition, the presented method in its current form could equally well transition to a small blending network

of cartwheeling motions, in which the user can control the speed and direction of any number of cartwheel cycles for as long as desired, before transitioning out this network and returning to the walking motion. The modularity of hybrid networks thus effectively allows them, without further modification, to provide *adjustable* transition sequences guided by user input.

6.8.7 Future Work

A simple extension for future work, feasible where phase 2 is periodic, would be the cyclification of this phase, allowing its repeated playback at runtime. Walk-twirl-walk could then become walk-twirl-twirl...twirl-walk, with the number of twirls user-controlled at runtime, and great potential for interactive games. Another enhancement would be the reflection of phase 2 capture data, giving two out-of-step sequences with one automatically selected to reduce latency before playback. An important improvement could make the triggering of transitions more flexible, by broadening the zone in blending space from which a transition can be launched, and possibly extending it thereby to the entire triangle network. Looking further ahead, transitions bestowed with motion-graph functionality could play back successive clips to obey higher-level commands, but even simpler solutions show considerable potential.

6.9 Comparison – Heck and Gleicher 2007

This section conglomerates the similarities and contrasts between the hybrid networks of this chapter, and the parametric motion graphs of Heck and Gleicher [HG07]. Despite being quite different methods, they do address similar issues and bear a degree of similarity, which warrants proper comparison. A detailed description of [HG07] was given in Chapter 4 Section 4.7, and so will not be repeated here, though a *brief* reminder of their notable work now follows.

Parametric motion graphs comprised interconnected blending spaces, providing – within certain limitations specified below – continuous control within these spaces, as well as transitions between them. Graph nodes represented collections of similar motions, from which animation resulted by interpolation, while edges stored pre-processed data which enabled the computation at runtime, of a subspace in the target node, to any part of which a transition from the source node could be made, via linear blend transition. The contribution of [HG07] was to extend the concept of motion graphs, by using transitions to interconnect *continuous* motion spaces instead of discrete motion clips, thereby adding some of the benefits of interpolation-based synthesis to sequence rearrangement methods.

A key difference between hybrid networks and the graphs of [HG07], is that the latter allow transitions to originate from *any* part of a blending space, while the hybrid network prototype does not, instead launching transitions only from trigger zones. This feature of parametric motion graphs is an important one, which provides hybrid networks with a clear (and feasible) area for future improvement. Nevertheless, it must be realised that parametric motion graphs do not allow the linking of blending spaces in *any* desired fashion, as might be assumed from their name, or from limited reading of [HG07]. Firstly, node-node transitions are *not* always possible, and secondly, transitions can only connect to a *limited subspace* of the destination node, and not, at will, to any point within it. Although hybrid networks do not suffer from either of these limitations, it is, nevertheless, parametric motion graphs which come closest to the concept of a universal interconnecting of blending spaces.

Another difference lies in the motion spaces themselves, which in the case of [HG07] are parameterised on one or more continuously valued parameters, allowing, for example, the target location of a reaching movement to be quite accurately specified. In hybrid networks such reaching action would instead proceed in an interactive fash-

ion, with visual feedback and user-control providing the finer adjustments necessary, in conjunction with algorithmic assistance to any extent found necessary. For certain movements, however, like an accurate boxing punch with its faster-moving action, target location specification may need to precede the motion, for which parameterisation (as in [WH97, SO06, KG04]) would be one possible solution, and a potential candidate for future work.

In contrast to the above two features of parametric motion graphs, the intention behind hybrid networks was to provide a straightforward and practical solution, a construction kit, for the building of interconnected blending networks encompassing varied motions. The higher modularity of hybrid networks becomes evident when considering both the relative ease of implementation and the small number of new motions required to add variety to a network. With both transitions and triangles in the hybrid network toolbox, the added motions need *not* be inflexible sequences, but can be controllable motions too. Furthermore, motion compatibility is required to join blending spaces in the method of [HG07], yet not with hybrid networks. The latter do, however, impose prerequisites on the input data, but this only in respect of transitions (thus only four such sequences out of the 25 used for the entire demonstrated implementation). In a greater sense, though, all input data is subject to specified requirements anyway, including, for example, the 275 motions used for a three-node boxing graph, which [HG07] acknowledge were supplied by a colleague, thus highlighting the general difficulty in obtaining suitable source motion, and putting the hybrid network transition data requirements very much in context.

Additional comparisons made in previous sections are summarised in Table 6.4. By highlighting the strengths and weaknesses of each, this section shows the methods to complement each other. Additionally, their underlying mechanisms suggest merging would be possible, and if achieved while retaining a clean interface, the resulting

Table 6.4: Reiteration of points brought up in previous sections, comparing hybrid networks to the time domain method of Heck and Gleicher [HG07]. (Reference to a section of the video of Chapter 7, instead of 6, is to demonstrate motion from the most advanced hybrid network constructed for this thesis).

Network Feature	Hybr. Networks	Param. Mo. Graphs
Transitions always possible between blending spaces?	Yes (given prerequisite input data)	No (even given best-quality data)
Transitions can suffer from blending artefacts, jerkiness etc?	No (as played-back motion)	Yes (as blending-based)
Demonstrated navigation smooth?	Yes (see Ch. 7 video, "Variable Blending Weights" section)	Can look robotic (see [HG07] video, "Interactively Controlled Walking" section)
Immediate response to user input while blending?	Yes	No
Easy selection of static unchanging target to trigger transition?	Yes (selection of fixed transition node, but see Future Work Sect. 6.8.7)	No (selection within target box of changing size & position)
Skeleton always honours user input?	Yes	No (user input may be overridden)
Min. number of motions req. to add new motion type & increase variety.	1	Entire collection of motions, typically dozens are used.
Additional blending spaces must resemble those already existing?	No	Partly (sufficiently similar frames req. for transitions)
Maximum length of transitions between blending spaces.	No limit	Very short
Transitions can be new actions, and do not merely interconnect?	Yes	No

application could be powerful indeed.

6.10 Conclusion

The contribution of this chapter was to present a novel structure, the hybrid triangle network, which seamlessly combines motion sequence playback with an existing method

for blending character motion in the frequency domain. It demonstrated how blending triangle functionality can serve to provide a smooth join with played back sequences by means of buffer triangles, while also maintaining, within transitions, the inherent continuity of captured sequences.

The presented approach unites the finesse of motion blending with the variety and quality of sequence playback, while providing the user with a simple interface. It greatly enhances the utility of triangle networks, can increase their size and offers an alternative to blending-based transitions which more directly reflects reality, and thus often looks superior. Hybrid networks are modular systems of already great functionality, but also have considerable potential for future work.

7

Single-source Harmonic Switching

7.1 Introduction

Good, or well-corrected input motion, would suffice to ensure satisfactory output, if the animation were merely to involve the playback of captured data. Motion *creation*, however, is a generative process which can simultaneously bring about motion defects, both during preprocessing and synthesis at runtime. Where significant, as is often the case, such defects will need correcting, and this over and above any measures applied to correct inadequacies in the input data[1].

Motion synthesis aberrations are commonplace and, as mentioned in Chapters 1 and 4 (Section 1.4.2 and 4.12), include ground penetration and hovering feet during the stance phase, inaccurate handholds, and very frequently, footskate, as found (or referred to) in [GBT04b, CH07, MCC09]. Jerky or robotic motions are also frequent [CH07, WMC11], and their appearance in distinguished research work clearly indicates the challenge involved, as with, for example, the method of Heck and Gleicher [HG07],

[1]The source data for this chapter as well as Chapter 6 required *extensive* corrections to reach the basic standard required for implementation work. These measures, however, are not the focus of this chapter.

which the hybrid approach of the previous chapter was repeatedly compared with[1].

The streamlined approach (Chapter 5), and the blending portion of hybrid networks (Chapter 6) employ blending-based synthesis. While artefacts are common with blending per se, side-effects born of interpolation-based synthesis are not exclusive to the blending stage or even to runtime, but can occur in preprocessing too. Some of the aberrations encountered during the practical work for this thesis are listed below.

- In walking or running characters, unwanted swaying from side to side, or forward and backwards, in step with the locomotion cycle.
- Generation of unnatural-looking asymmetry.
- Undesirable overshoot taking the foot outside such a trajectory as is deemed visually acceptable. Examples include foot-ground penetration, hovering feet, a footskate-like sliding effect *prior* to ground contact as well as footskate itself.
- Slight and rare solitary twitch, especially at the skeletal end-nodes of arms, legs and head.
- Occasional intervals of jerkiness in walking motions, experienced when – to achieve best possible output quality – a higher number of harmonics is used.

It has to be emphasised that the enumerated faults are no reflection on the quality of the undertaken practical work, as those thought a significant annoyance were, where

[1]The Parametric Motion Graphs video showing the method of [HG07], exhibits markedly robotic walking and running motion upon changing direction, despite the forgiving camera viewpoint moving in unison with the skeleton (section "Interactively Controlled Walking", frame times 1:36 to 1:56, and section "Interactively Controlled Running", frame times 1:57 to 2:05). The demonstrated walk-to-run transition (section "Interactively Controlled Walk-Run", frame times 2:23 to 2:33) is overly abrupt, and footskate is evident (for example in section "Random Everyday Actions", at around frame time 0:57, visible despite partial occlusion). (Download information in Appendix E).

feasible, reduced or entirely avoided by implemented corrective measures and implementation design choices, especially in the current and the previous chapter. In contrast, the need to *post-process* unwanted artefacts is often explicitly mentioned in the literature [AF02, BW95, PL06] and even then is not necessarily undertaken, and may be left uncorrected, with or without reference to a third party method like [KSG02], mentioned as a suitable cure. It would be hard to find such artefacts as might be allocated to post-processing in the motion resulting from the method of this chapter. The aberrations enumerated above thus clearly indicate no failing.

Of the above-listed artefacts, it is correction of the last which is the focus of this chapter. While often imperceptible, the aberration, when apparent, involved a side-to-side jerkiness in the character's coronal plane. Any irritation thereby experienced appears fully justified, as natural-looking motion is inherently smooth resulting from a minimisation of energy use and associated joint torques [WK88, RGBC96].

The remainder of this chapter expands on the nature of this artefact and presents a solution which greatly reduces it, while furthermore lowering the cost of frequency domain motion synthesis. Section 7.3 illustrates the symptoms in detail by concentrating on their manifestation in the motion of the foot, the effect of which takes one of two implementation-dependent forms. One such form, the above-mentioned coronal plane jerkiness, has, to a degree, at times been present in the demonstration videos which accompanied earlier chapters. The exemplifications of Section 7.3, followed by analysis of the underlying cause in Section 7.4, show the problem to be a general one inherent in the field of blending itself, including, importantly, time domain blending, thus showing it not to be limited to frequency domain work, (nor, indeed, to the implementation work of previous chapters). In-depth consideration is then given to a corrective measure which works, in brief, by forfeiting the conventional use of blending weights in multitarget blending, and instead of interpolating between input motions, switches

from one to the other. Switching follows a pattern whose design and rationale are expanded on in Section 7.5. It is quite possible to apply the method only locally to parts of the network, which then smoothly co-exist with areas using conventional blending. This mechanism is explained in Section 7.6. Input motion switching is applied only in respect of certain harmonics, to which the name *single-source harmonics* (SSH) is correspondingly given, while calling the process itself *SSH switching*. To quantify the improvement in motion quality resulting from SSH switching, a means is developed to measure the distortion it aims to counteract. A metric is elaborated in Section 7.7 which also provides a network-wide plot of pre-correction distortion intensity. The following three Sections (7.8 to 7.10) then present and discuss results, and provide final remarks.

SSH switching, a part of SSH blending which includes the other, normally blended, harmonics, is dependent on interpolation being performed in the frequency domain. It thus expands the field of frequency-domain-based blending while providing functionality not available in the time domain.

7.2 Structure of Preliminaries

Before proceeding to detail the mechanism of SSH switching, the distortion it is designed to counteract requires both description and analysis. As this preliminary stage is quite extensive, it is useful to describe the coherence of its structure. It divides into the following steps.

Illustration. A complete description of the unwanted distortion is first given, *before* proceeding to analyse its underlying cause, as the object to be analysed, ie the distortion in its various facets, should first be firmly established. This is provided by Section 7.3 which is further subdivided as follows.

- Illustrate distortion in skeleton nodes, in the first of two manners it can manifest during frequency domain blending (Section 7.3.1).
- Illustrate *same* distortion, as before at node level, but now manifest in second possible manner, again while frequency domain blending (Section 7.3.1).
- Show spread of distortion amongst various frequency domain harmonics (Section 7.3.2).
- Show similar distortion to exist in *degrees of freedom* (as opposed to node level) when blending in the frequency domain (Section 7.3.3).
- Show same distortion to be present in the *time domain*, at both the node *and* the DOF level(Section 7.3.4).
- Provide reasonable conjecture on further extent of distortion, extrapolating beyond available data (Section 7.3.4).

Analysis. Having fully described the problem to be remedied, the reasons for its manifestation are investigated (Section 7.4). The motion data employed in the blend is considered as well as the process of merging it by blending. Since played back input data alone exhibits no distortion, but blended motion does, the origin of the distortion should then become apparent, which indeed it does.

Thus, while Section 7.3 is purely illustrative and devoid of explanations, these follow in Section 7.4, in which the source of distortion is fully addressed.

7.3 Upper-harmonic Distortion – Illustrations

This section depicts the nature of the impinging artefact, aiming to *illustrate* the problem in its various facets (and leaving analysis of its cause to Section 7.4).

7.3.1 Foot and Root Trajectory Corruption

Jerkiness in skeletal output motion can be equally described as a distortion in the node trajectories of the generated walk cycle, which in turn is due to unwanted fluctuations in the angular degrees of freedom affecting the chains leading to these nodes. The end-nodes of the legs are chosen to illustrate trajectory distortion, and for brevity, the term 'foot' is loosely allowed for such nodes, and 'foot-trajectory' for the path they follow from frame to frame. Foot-trajectories are selected as they greatly affect animation quality, though it is stressed that the distortion under investigation is not limited to the motion of these nodes.

The significance of foot-trajectory distortion depends on implementation design, and its impact is now clarified before proceeding.

1. In cases where the position of the root node is determined by blending those of the input motions, such foot trajectory defects, if left uncorrected, would lead to unsightly sliding of the foot in an undulating fashion, or poor vertical positioning instead of precise ground contact. The error would take the form of intermittent fluctuations and would be in addition to any footskate or foot misplacement due to other causes.

2. The implemented system avoids both footskate and foot-elevation inaccuracies, using methods which nail the contacting foot to the ground. This, in turn, dictates the position of the root node, but thereby also transfers the aforementioned foot trajectory distortions to the root, and hence to the entire skeleton. The result is then jerky motion reflecting the oscillatory frequency and sporadic nature of the contaminated foot trajectories.

A distorted foot trajectory is shown in Figure 7.1 which, while relatively rare in such extreme form, provides a convenient case of known-bad blending for consideration

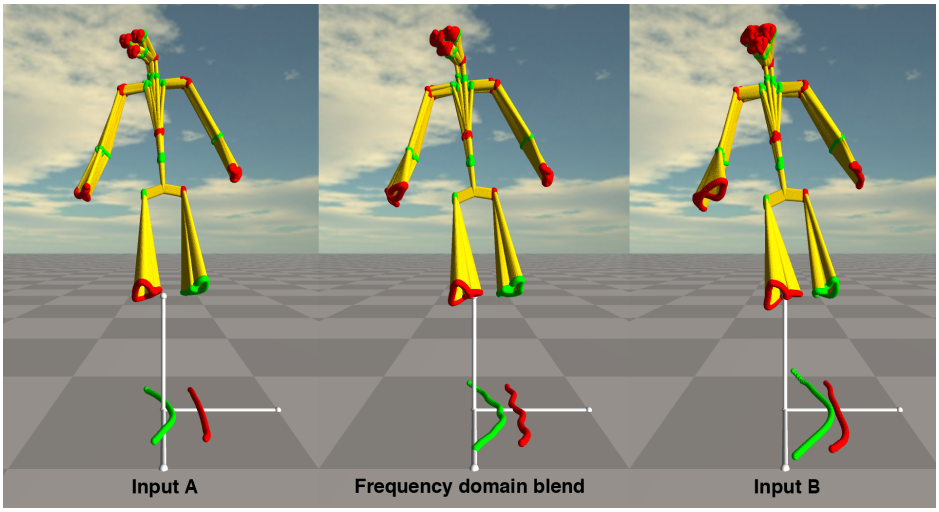

Figure 7.1: "Foot trajectories" traced by showing successive frames of the ground contact stage of each foot, with the skeleton walking on the spot, y-axis root rotations locked, thereby illustrating the leg end-node position relative to the root. Skeletal segments between knee joint and end-node are hidden to prevent occlusion. A simple 50/50 blend (centre) exhibits distortion not seen in the two input motions (left and right).

throughout this chapter. The blend is between two inputs given equal weightings, which fortuitously helps keep explanations simple. However, such blending applies only at the midpoint of triangle edges, and it should be remembered that interpolation in the triangle network in general is between three inputs, using any legal combination of blending weights.

For the purpose of analysis, the skeleton's x/z ground-plane root translation and y-axis root rotations are disabled, causing it to walk on the spot, above the reference frame origin, facing the positive z-axis. The contact stage of each foot thus becomes one in which the foot is moved along the ground beneath the stationary skeleton. This approach allows a trace to be made of each foot showing the subsequent frames of its ground-contact stage, thereby highlighting any unwanted trajectory distortions.

Disabling the skeleton's rotation about the y-axis has no effect on the extent to which it leans forward or sideways, which, if it did, would greatly affect foot placement.

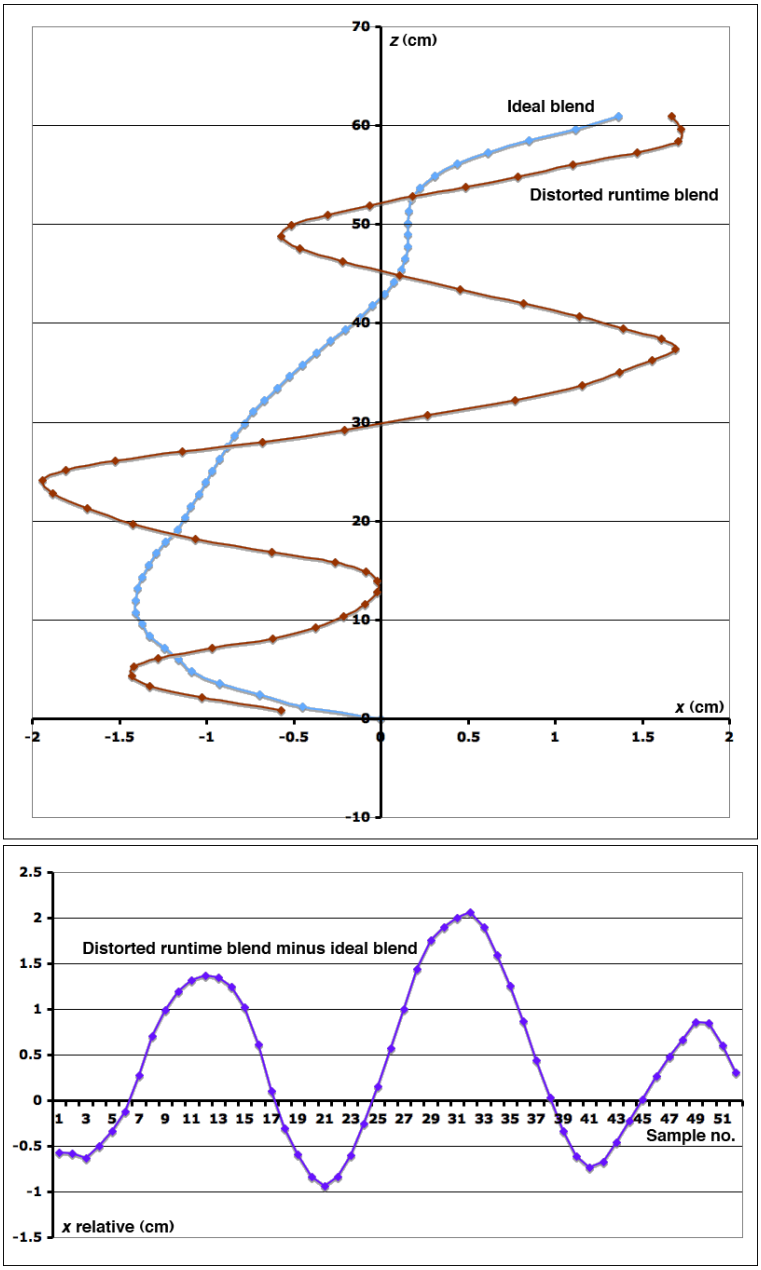

Figure 7.2: (Top) Root trajectory during left-foot ground-contact phase as skeleton walks towards positive z-axis (positive x-axis shown on right of diagram as per convention is on skeleton's left). Plots show distorted motion which this chapter will present a method of correcting, and an undistorted path for comparison. (Bottom) Distortion extracted from root trajectory by subtracting the ideal blend from the distorted path.

This is avoided as, in the implemented system, x and z root rotations are performed first during synthesis as detailed in Chapter 5 Section 5.4.2. For this reason, other than changing the bearing towards which the skeleton points, its actual pose, and thus the nature of its foot placement, remain unaffected by y-axis rotation locking. Preventing the skeleton from turning as it walks on the spot facilitates the visual comparison of different motions, and simplifies the measuring of any differences between them.

When enabling foot constraints, as applies to the implementation, the now-*travelling* skeleton exhibits the distorted *root* trajectory seen in Figure 7.2, top, which focuses on the worst afflicted left-foot contact stage. An error-free ideal root trajectory is also shown for reference. The latter was created by interpolating root-node location sequences taken from the input motions, after resampling them to give them frame counts which became both mutually identical and equal to that of the distorted synthesised blend. (Node position interpolation – while used to create the reference plot – is rejected as an option for full-skeleton runtime synthesis as explained later in this section. The demonstrated distortion could thus not simply be eliminated from the animation in this way).

The bottom of Figure 7.2, shows the distorted path relative to the error-free blend, thereby displaying the unwanted oscillations themselves. These are seen to reach peak-to-peak values of around 3 cm in the centre of the plot, with the sideways shift occurring in 0.11 seconds in what was a 100 frame-per-second animation, thus a noticeable jerk given the character's 1.775 m height [1].

Having highlighted foot- and root-trajectory distortion equivalence, focus now returns to the former as a means of examining both. In Figure 7.1, the wobbles in the blend are clearly anomalous as not present in the input motions. This suggests one

[1] Estimate, based on skeleton's measured 1.64 m height, from ground to eye-node, in T-pose

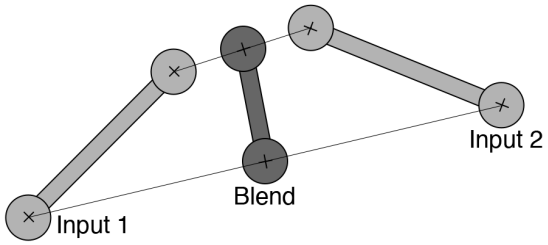

Figure 7.3: Node position interpolation generating a blended segment of different length to that of the inputs – an unacceptable violation.

simply blend between node locations, instead of joint angles, which would easily provide a neat foot trace unspoiled by unwanted distortions. However, naively blending node positions would lead to changes in skeletal segment lengths, as indicated by Figure 7.3. While variable segment lengths *have* been used in previous work, for example in [BC89], this clearly does not reflect reality, and even where variations are small and seemingly able to be overlooked, problems would result in the following stage of dressing the character with simulated clothing [Sim12]. Admittedly, end-node position blending might be a feasible strategy when combined with inverse kinematics and biomechanical data to obtain the joint angles of the legs, assuming runtime costs proved low enough. However, this approach is rejected in favour of a frequency-domain-blending-based solution which is consistent with previous implementation work and thus neatly extends it, while remaining within the Fourier blending focus of this thesis.

7.3.2 Harmonic-dependent Distortion

Figure 7.4 shows the foot trace for the same 50/50 blend as did Figures 7.1 and 7.2, while employing different numbers of harmonics. The varying shapes of the initial few traces are to be expected, as they comprise insufficient harmonics to fully create the form of the foot trajectory. The general trajectory profile is seen to be achieved given 4 harmonics, as only a small change results when increasing the number to 5. Adding

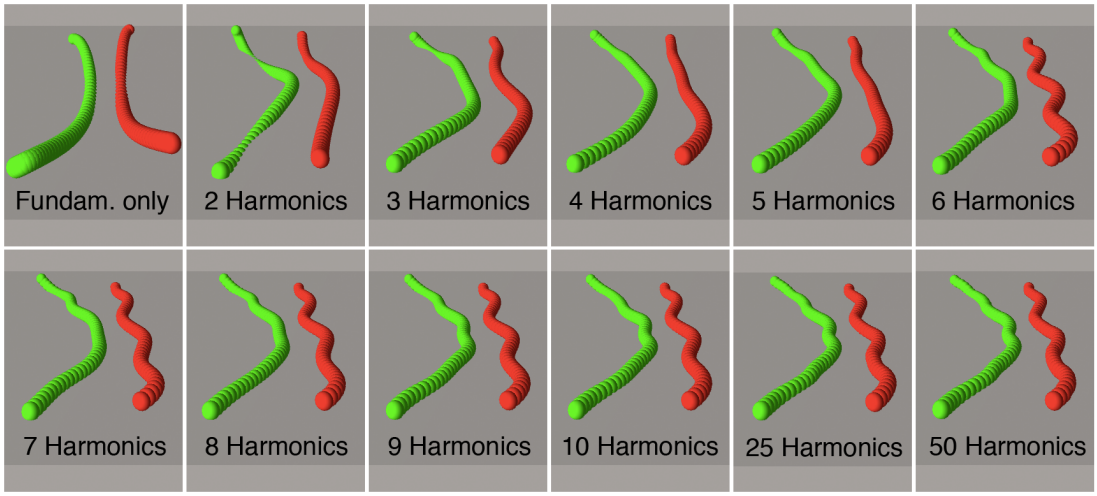

Figure 7.4: Foot trajectory trace variations for differing numbers of harmonics applied to the same blend as in Figure 7.1, with identical orientation and reference frame. As before, red indicates left foot and green the right.

the sixth harmonic, however, introduces distortion to a striking degree, which remains present when further harmonics are included.

To assess the magnitude of the effects of harmonics H6 and higher (where Hn denotes the n^{th} harmonic) Figure 7.5 plots the *contribution* to left-foot trajectory distortion created by individual harmonics. The plot for Hn results by subtracting the trajectory x-values for the motion with n harmonics from that using $n-1$, whereby the positive and negative x-axes extend on the skeleton's left and right hand side respectively. A declining contribution is seen with each increase in harmonic index, with the tenth harmonic adding a circa 1 mm peak-to-peak waveform whose effect would only be noticed in the most exacting of conditions. Accordingly, the tenth harmonic was, somewhat arbitrarily, chosen as the final significant contributor to this distortion, with the undulatory foot-positioning defect considered to be introduced by joint rotation harmonics within the band spanning – in the context of the implementated system –

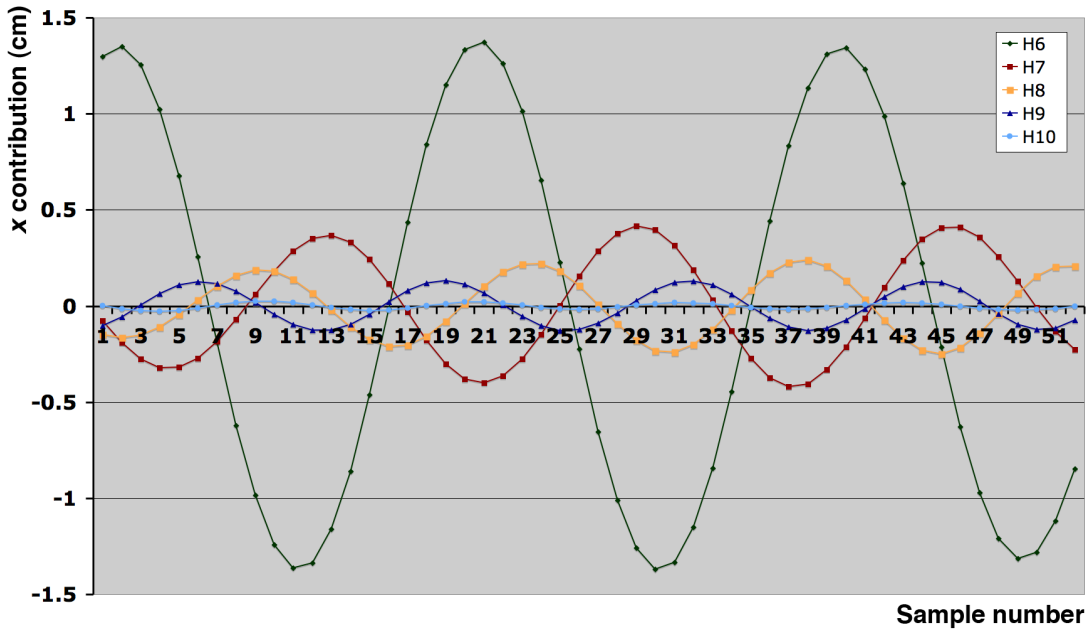

Figure 7.5: x-direction contribution of harmonics H6 to H10 to the final waveforms shown in Figure 7.4 (positive x-axis extends to skeleton's left). Contribution falls rapidly with increasing harmonic index.

H6 to H10.

Similar band limits were found in other parts of the network suffering from this defect. Furthermore, when, experimentally, more disparate input motions were placed at triangle vertices, thus significantly transforming the network, foot trace distortions were still experienced. As expected in the light of these network modifications, the extent to which individual harmonics contributed was somewhat changed, but their existence in a band lying above those required for basic motion creation applied exactly as it had before. Chapter 5 Section 5.7.6 stated 3 harmonics to be sometimes sufficient for rudimentary synthesis, while the especially discerning user, exacting viewing conditions and highest quality input motion may demand 7 or possibly 8. The [H6–H10] range, whose upper limit was based on the magnitude of the H10 contribution to foot trace distortion, thus covers – with leeway to spare – the most demanding requirements

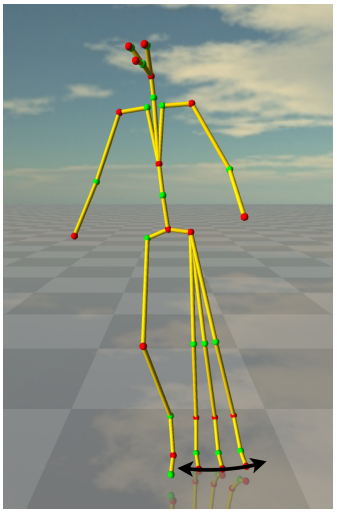

Figure 7.6: Rotations about the local z-axis of the left hip being examined instead of the foot-location sequences of Section 7.3, thus moving the focus of investigations to the angular degrees of freedom quantified in motion capture data.

encountered in practice for higher-frequency harmonic content. The artefact can thus be said to corrupt the higher of those harmonics which would be used to generate the best possible quality motion. As a convenient reference throughout this chapter, the terms *upper-harmonic distortion* (UHD) and *UH distortion* are thus selected for it.

7.3.3 DOF-level Distortion

The distorted left-foot trace of the previous section evidently results from angles in the chain from root to foot, of which one is now selected for further study. The left hip z-axis rotation (Figure 7.6) will now be considered, as any unwanted undulations of this angle would be echoed in the left foot trace in a direct manner, and likewise, the observed foot trace distortions can be assumed to be to a great extent generated by this particular DOF. To test this, Figure 7.7 compares the left hip z-axis rotations applicable while creating the low-distortion 5-harmonic trajectory with the angles for the 6- and 25-harmonic traces beset with UH distortion. The full 113-frame walk cycle

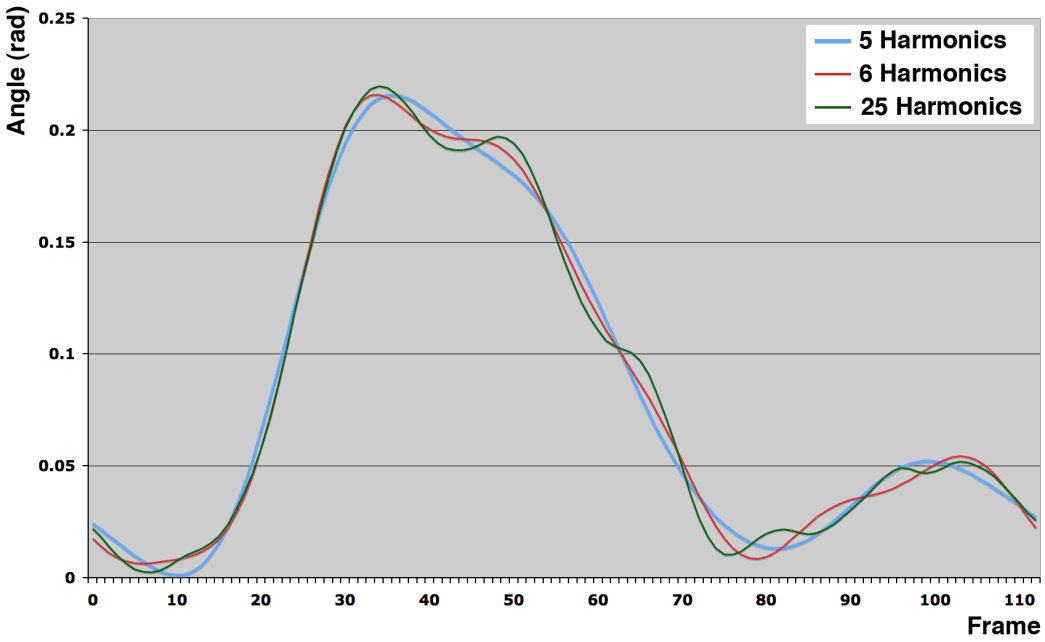

Figure 7.7: Left hip z-axis rotation angles during entire walk cycle, showing UH distortion – previously observed only in foot-*location* traces – now visible at the *DOF* level. The 5-harmonic trace is smooth, those with 6 and 25 are blatantly distorted.

is shown which better illustrates the presence of undulations (the left-foot contact stage spans from frame 86 to 23 inclusive). The 5-harmonic angle sequence displays a smooth appearance. In contrast, conspicuous undulations appear in those comprising 6 and 25 harmonics, thus confirming that UH distortion is recognisable at the DOF level, which the investigation now proceeds at.

7.3.4 Distortion in Time Domain

Having considered UH distortion in frequency domain blending, the z-axis hip angles for the *input* motions and their blending in the *time domain* are now examined. For consistency, the same 50/50 blend is considered as before.

Figure 7.8, top, shows hip angle sequences for the two input motions in their original form as used in the implementation of this chapter. The middle plot shows them

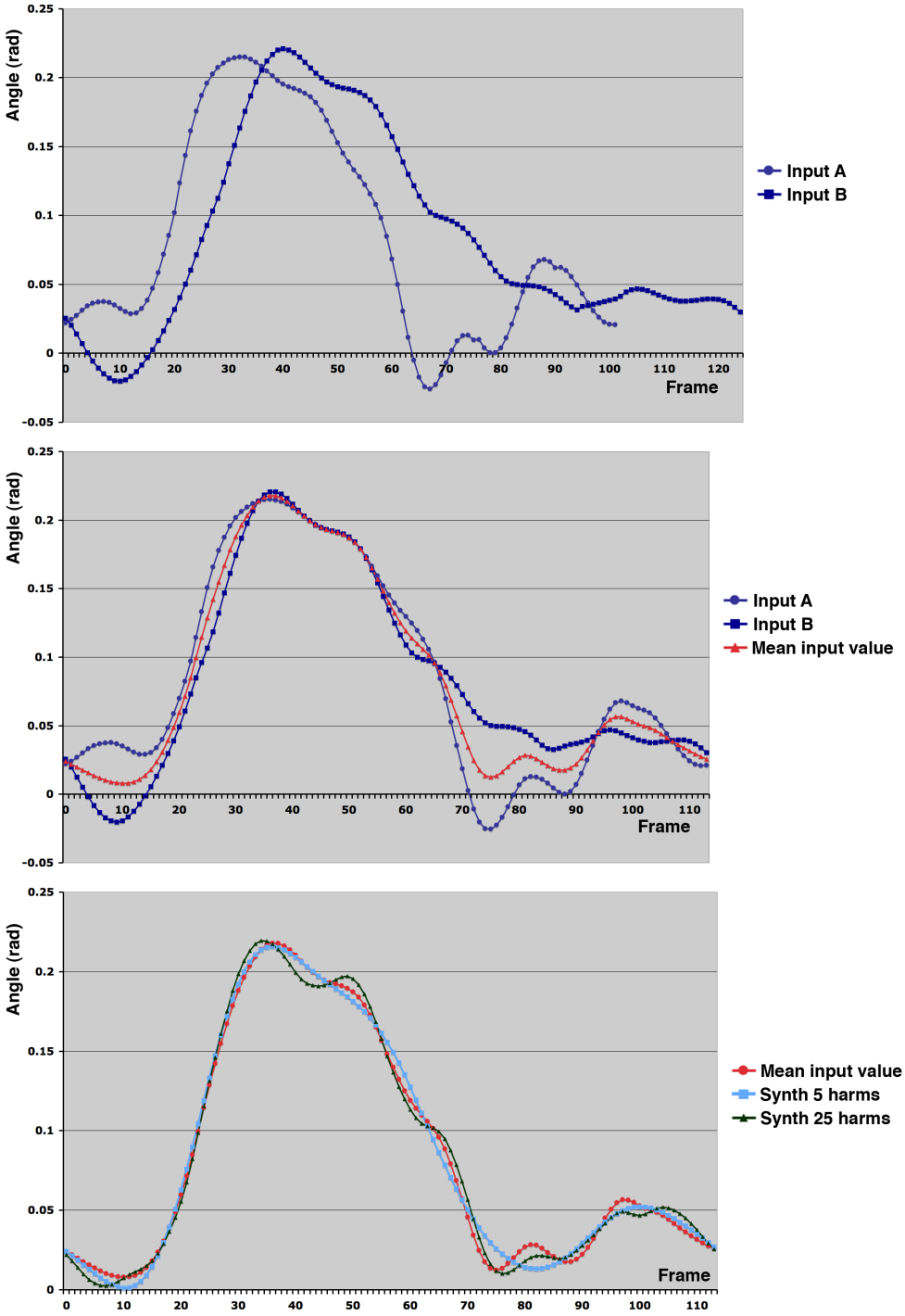

Figure 7.8: (Top) The two input motions to the blends of Section 7.3 in their original form. (Centre) Inputs resampled to acquire equal lengths and arithmetic mean calculated, the latter being equivalent to 50/50 time domain blending. (Bottom) Time domain blend (identical to centre) exhibiting UH distortion particularly apparent when compared to smooth frequency domain blended 5-harmonic output.

resampled to be of equal length, with their arithmetic mean superimposed upon them. The mean is equivalent to a simple graphically-derived average, as well as to 50/50 blending in the time domain, and follows the expected path, midway between the two input motions. Such blending makes use of all harmonics in the input curves, and dispenses with frequency-domain-based procedures altogether. Significantly, the bottom of Figure 7.8, which duplicates the above-mentioned time-domain blend , shows it to exhibit distortions rather similar to the frequency-domain-synthesised 25-harmonic angle plot. Some variation between the two UH-distorted waveforms (frequency and time domain) is to be expected, one being the sample-by-sample mean of the two inputs, and the other resulting from the blending of harmonic magnitudes and phase angles followed by Fourier synthesis, as described in Chapter 5 and Appendices A and B – mathematically non-identical processes. Such variations were, after all, found even when comparing frequency-domain-based-only curves exhibiting UH distortion (Figure 7.7). What is significant here, is that both time and frequency domain curves exhibit a similar style of undulations not present in the smooth 5-harmonics-only motion, which is included for reference. The similarity is further highlighted by the fact that Figure 7.8, bottom, and Figure 7.7 look very much alike. UH distortion is thus seen to exist quite independently of interpolation being performed in the frequency domain, something further confirmed by Figure 7.9 demonstrating foot *position* trace distortion while blending, on this occasion, in the *time domain*.

Thus, in the context of the captured walking motions under consideration, UH distortion is seen to be inherent in blending itself, and quite independent of whether performed in the time or the frequency domain. Moreover, as all walking motions are similar, it must be concluded that UH distortion is a general phenomenon, potentially exhibited during any walk cycle blending. This claim is upheld by past experience, as well as by the pre-correction plots of Figure 7.19 (Sections 7.7) demonstrating that

while its magnitude varies greatly, UH distortion is widespread within the blending mechanism of the implemented triangle network.

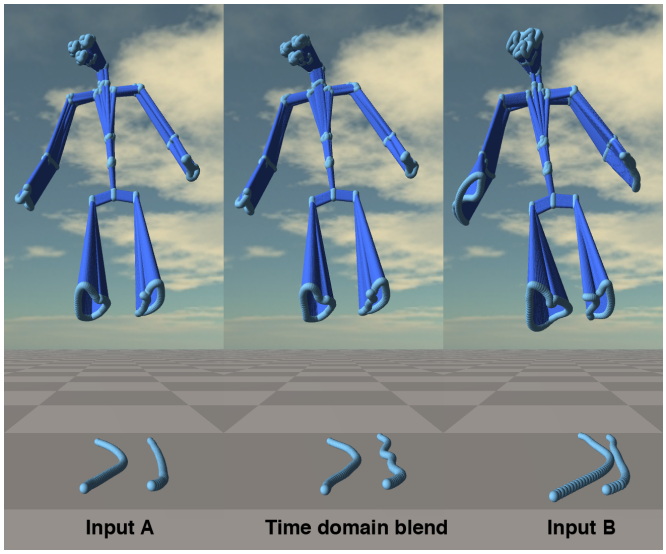

Figure 7.9: *Time domain* blending leading to UH foot-trajectory distortion akin to that found in the frequency domain in Figures 7.1 and 7.4.

7.4 Upper-harmonic Distortion – Analysis

The phenomenon of UH distortion has been thoroughly demonstrated above. In this section, focus moves on to the underlying cause of UHD, and reveals not only its origin, but also explains the reason for its fluctuating nature.

Appendix D provides background knowledge to this section, which the reader may wish to examine before proceeding.

7.4.1 UH Distortion Origin and Variability

An explanation as to the source of UH distortion and a reason for its inconsistent severity – from negligible to pronounced – is suggested by means of Figure 7.10, bottom,

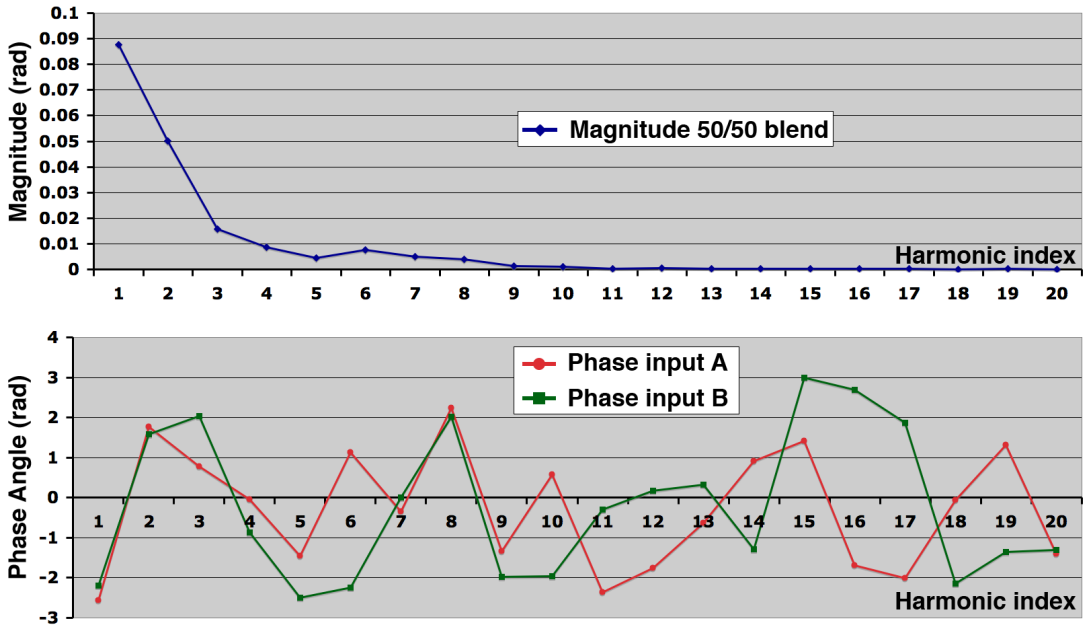

Figure 7.10: (Top) Magnitude of the first twenty harmonics available in the left hip z-axis rotations of the same 50/50 frequency domain blend as shown in Figures 7.1 and 7.4. Magnitudes of H11 and higher are seen to be effectively nil. (Units are radians as harmonics are constituents of the z-axis rotation *angle* trajectory). (Bottom) Phase angles of the two input motions in the blend, showing expected similarity for the fundamental but quickly becoming unpredictable with increasing harmonic index.

which shows the phase angles of the various harmonics inherent in the z-rotation angle sequence of the left hip for the two input motions in the 50/50 blend referred to throughout both this and the previous section. The phase angles of a given degree of freedom of an input motion are properties of those harmonics, which if accumulated, would reconstruct that DOF data sequence. The plot (Figure 7.10, bottom) thus relates to two distinct types of angle, hip angles – the data itself – and the phase angles of its constituent frequency components.

The fundamentals (first harmonics) of each input motion are seen to have similar phase angles. As further explained in Appendix D (Sections D.2 and D.3), this is to be expected for two synchronised walk cycles since, for each DOF from root to foot, the

first harmonic provides the basic sinusoid controlling leg movement, so in-step similar gaits must have aligned first-harmonic phase angles.

Higher-indexed harmonics add the finer peculiarities of a particular motion, and while those neighbouring the fundamental have a tendency to be mutually in phase, this is quickly lost when moving further away, whereby phase angles become unpredictable, as seen in Figure 7.10, bottom. Synchronisation thus concentrates on the lowest of frequencies, which, however, is precisely how it *should* operate, as explained in Appendix D.

The range [H6–H10] thus has the *potential* for the phase angles of one motion to vary greatly from those of the other (in a rotational sense) and consequently for blends *in respect of these harmonics* to experience great disparity between input motions, with an associated increase in the *risk* of blending artefacts. Harmonics above H10 are simply too small in magnitude to create significant levels of distortion, as seen in Figure 7.10, top.

Referring to the specific frequency domain blends of Figure 7.4, the sudden appearance of distortion when moving from 5 harmonics to 6, is consistent with the phase angles in Figure 7.10, bottom, suddenly diverging to a state of near-antiphase upon reaching H6. What results at the H6 level is akin to the blending of two out-of-synch walking characters, one striding ahead with his left leg forward, the other with his right leg leading, resulting in a blend which resembles neither. Combining this effect with the non-negligible H6 magnitude (Figure 7.10, top) results in the visible distortion seen in the foot-trajectory. The similarity of the [H7–H8] phase angles, and the low magnitudes of [H9–H10] would explain why little change is witnessed when successively increasing the number of harmonics from six to ten.

The unpredictable nature of phase angles in the UH band, and their non-irrelevant magnitudes, underlies not only the presence of UH distortion specifically in the [H6–H10]

range, but also its occasional nature. While the band limits must be seen as implementation-dependent and subject to some variation, the reasons for the presence and the variations in UH distortion remain the same.

7.5 SSH Band Structure and Control

An approach to lessen the problem of UH distortion is now presented, but only after excluding the simplest method of all: avoiding the UH band altogether. The obvious drawback is that such motion would clearly be of a quality below the highest attainable. More precisely, while pleasingly smooth, it would, when viewed at length or assessed by the discerning viewer, be sensed to lack a quality detailed below. Motion perception is a personal and somewhat subjective experience, so the liberty is now taken to describe this missing quality in words of a corresponding nature: UH-band-devoid motion is, in some contexts, seen to lack "freshness" and appears somewhat "impeded". A need thus exists to remove UH distortion without clipping the harmonic content.

7.5.1 Single-source Harmonics

In order to retain upper harmonics yet escape potential distortion, a method is proposed which employs the UH band but avoids the need to blend within it. It is based on the observation that afflicted blends like that in Figure 7.1 become distortion-free, if, for the harmonics in the UH band, a 100% contribution from just one of the input motions is used instead of a 50% contribution from each. Thus, conventional blending is still employed below the UH band (for the "DC" value and H1 to H5), but single-input sourcing is used within it (H6 to H10), thereby circumventing any blending artefacts associated with upper harmonics.

It is important to note that the basic nature of the output motion, and all but the finest of its idiosyncrasies, are provided by harmonics *below* the UH band, and thus

remain unaffected by the change just described. This can be illustrated by reference to the motion underlying Figure 7.1, which is, in fact, a sharp right turn (once y-axis root rotations are re-enabled). Unlike vehicles, humans sway from side to side during locomotion, and cannot walk in a perfect circle with an ever-constant radius. The *maximum* turn radius was thus measured for the blended turning motion, and was found to be 78.9 cm when using normal-style UH-distortion-ridden blending, and 79.2 cm while sourcing all UH band harmonics from one motion (in this case motion B). Thus, enabling single-source harmonics merely altered the maximum turn radius by 0.33% – a quite trivial difference easily compensated for by the human or software controller steering the skeleton at runtime.

Despite the single-source layer and the blended-harmonic foundation beneath it being somewhat alien in nature, the combination creates motion of a quality perceived in most cases to be very close to that of the input motions, and far superior to the distortion-plagued ordinary blending it aims to replace. The mere presence of upper harmonics, despite their one-sided constitution, successfully avoids the sometimes stifled appearance of harmonically clipped motion.

Harmonics of this type, which draw their data from only one input motion, will be designated *single-source harmonics* (SSH) giving the UH distortion band in which they reside the alternative name *SSH band*, better describing its newly proposed function. The difference between harmonics in the SSH band and those below lies solely in the treatment of blending weights. When applied to two-input blends like that of previous sections, each harmonic in the SSH band has one weighting set to 1.0, while the other is 0.0.

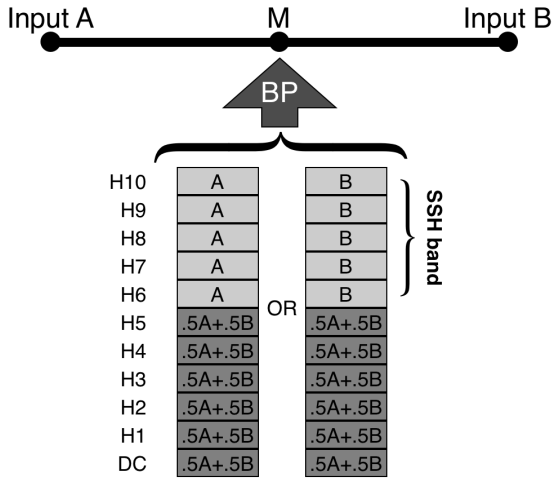

Figure 7.11: Alternative SSH band compositions, for when avoiding UH distortion by feeding all upper harmonics entirely from one of the input motions – an option available for cases where the blending point, BP, is at the midpoint of a 1D blending space.

7.5.2 One-dimensional Case – SSH Band Structure at Key Points

As all SSH band harmonics in the above blend are copied from one and the same input motion, the question arises as to which motion to choose. Furthermore, blending point locations other than mid-way between motion A and motion B may require alternative SSH band compositions. These issues are discussed below, with respect to the blending-point-specific band configuration – designated *SSH stack* – at key points in the blending space.

Midpoint M. The two possible SSH stacks for the 50/50 blend, thus with the blending point at the midpoint M of the blending space, are shown in Figure 7.11. The choice of input motion from which to populate the SSH band need only be made once and remains unchanged for all subsequent program runs. An algorithmically derived choice providing automatic triangle network set-up would be feasible as discussed in Section 7.9, although with only three decisions needed per triangle, one for each input motion pair, the prototype implementation for this chapter was

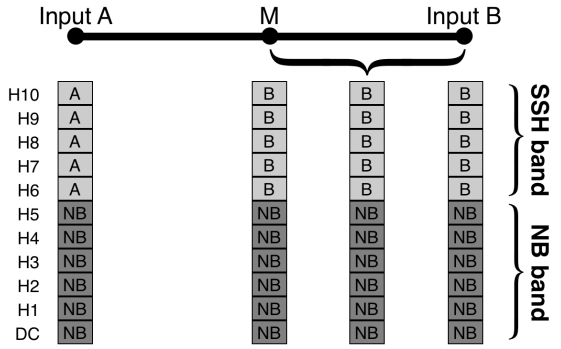

Figure 7.12: For maximum affinity with the NB band below them, SSH stacks at endpoints A and B must be $\langle 5a \rangle$ and $\langle 5b \rangle$ respectively. Assuming $\langle 5b \rangle$ was chosen for midpoint M, then compatibility with the NB band remains for the entire range [M-B].

based on visual examination to determine which input generated the best looking SSH blend. The difference between the two choices is often small, however, with either motion giving good and near identical results.

Endpoint A. The normally blended (NB) band, spanning [DC–H5], always obtains its harmonics by taking a weighted average of the corresponding harmonics in input motions A and B. With the blending point at endpoint A, this will comprise a 100% contribution from motion A, though the equivalence with the single-source approach is purely coincidental. Clearly, combining this A-based NB band with an A-populated SSH band, as expressed by the notation $\langle SSH \rangle = \langle a,\ a,\ a,\ a,\ a \rangle = \langle 5a \rangle$, would generate good motion, since the result ($\langle NB,\ SSH \rangle = \langle 6a, 5a \rangle = \langle 11a \rangle$) would be equivalent to input motion A itself. (The higher harmonics of motion A, ie H11, H12, etc, are ignored as insignificant). It is furthermore apparent, that changing any harmonic in the SSH band to a B-type harmonic, would degrade the synthesised output, as input motion, whatever its quality, undergoes a loss of naturalness when mixed with another. SSH stack composition at point A is thus as shown in Figure 7.12

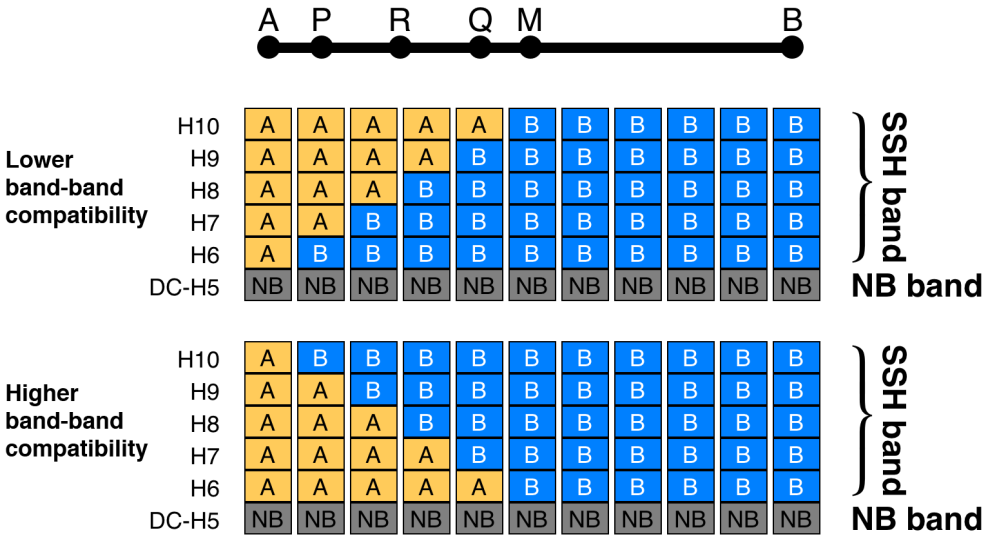

Figure 7.13: SSH band structures for one-dimensional blending space. Each transmutes, without blending, from B-based at point M, to A-based at endpoint A. The upper structure, however, can generate distortion in the region of P (see text), while the lower structure avoids this.

Endpoint B. For similar reasons to those above, at point B, the opposite end of the blending space, the SSH band should comprise B-type harmonics only, giving $\langle NB,\ SSH\rangle = \langle 6b, 5b\rangle = \langle 11b\rangle$.

Range [M-B]. Assuming, for example, that of the two choices (A or B) motion B harmonics were selected to fill the SSH band at point M, as per right-hand option of Figure 7.11, then the entire space from M to B will be able to use $\langle SSH\rangle = \langle 5b\rangle$, as indicated, again, in Figure 7.12. The reason for this is that as B-type SSH band harmonics were observed to be compatible with the blended harmonics of the NB band at point M (where $\langle \mathrm{NB}\rangle = \langle 6 \times \frac{a+b}{2}\rangle$), one can, all the more, expect compatibility when blending closer to motion B, ie when the NB band becomes more heavily weighted with that motion.

P in Range [A-M]. In order to move the blending point from point M to A, the

SSH band needs to change from $\langle 5b \rangle$ to $\langle 5a \rangle$ whereby interpolating from one to the other is excluded, since blending within the band causes the very UH distortions SSH processing is intended to avoid. The proposed solution is to successively *switch* harmonics, from 100%-B-sourced to 100%-A-sourced, as the blending point travels from M to A. This, however, again leads to two possible approaches. That of Figure 7.13, top, would potentially lead to severe distortions when blending at point P, near A, as H6, being the lowest harmonic in the stack, has, in general, more influence on motion quality than any other, and B-based H6 is not compatible with the mainly A-based NB band below it, since A and B lie at opposite ends of the blending space. Adding a B-based H10 harmonic at P, however, as in Figure 7.13, bottom, is quite acceptable, as H10 plays a far less significant role than H6, making the A/B incompatibility of little consequence.

Q in Range [A-M]. With reference, once more, to Figure 7.13, bottom, at point Q near the opposite end of [A-M], the situation is at first sight similar to the problem at P. A-based H6 is disparate from the others in the SSH stack as well as, to some extent, from the approximately 50/50-blended NB band below it. However since $\langle NB,\ SSH \rangle = \langle NB,\ 5a \rangle$ was an alternative option to the actually-employed $\langle NB,\ SSH \rangle = \langle NB,\ 5b \rangle$ for the motion's composition at point M, $\langle NB,\ a + 4b \rangle$ which due to the importance of H6 is similar to $\langle NB,\ 5a \rangle$, works well near point M. Thus, for point Q like P before it, the stack shown at the bottom of Figure 7.13 is suitable for motion synthesis.

R in Range [A-M]. Finally the midpoint R between A and M is more heavily A-weighted in the NB band, than applied at Q or M, allowing co-existence with the now heavily A-weighted SSH band above it. Strong A-bias exists in this

band about point R, whether $\langle SSH \rangle = \langle 3a, 2b \rangle$ or $\langle 2a, 3b \rangle$, due to the greater importance of lower harmonics.

It is thus the blending-space-wide *SSH pattern* or *SSH structure*, shown in Figure 7.13, bottom, which ensures greater compatibility between the two bands, and forms the basis of SSH switching.

7.5.3 One-dimensional Case – SSH Pattern Computation

Switching in triangle networks requires computation of that part of the SSH pattern corresponding to the current position of the blending point. The one-dimensional case is detailed first, and later extended to two dimensions in Section 7.5.4.

In the 1D case, harmonics feed off one of two possible input motions, thus becoming harmonics of one type or the other, and the cardinality of each needs to be determined. B-type harmonic cardinality throughout the blending space as a function of B-weighting, w_B, is given by

$$K_B = \left\lfloor \min\left(\max\left((w_B - w_{B_{start}}) \frac{wi}{w_{B_{stop}} - w_{B_{start}}}, 0 \right), wi \right) + 0.5 \right\rfloor \tag{7.1}$$

where wi is the width of the SSH band (5 in this particular case), and $w_{B_{start}}$ and $w_{B_{stop}}$ are the w_B weightings (0.0 and 0.5 respectively) defining the endpoints of the ramp section shown by the red line in Figure 7.14.

The number of single-source harmonics copied from motion A is, then, self-evidently

$$K_A = wi - K_B \tag{7.2}$$

The cardinalities determined by Equations 7.1 and 7.2 are illustrated for 0.1-sized weighting increments in Table 7.1, and yield the SSH band *structure*, as shown in

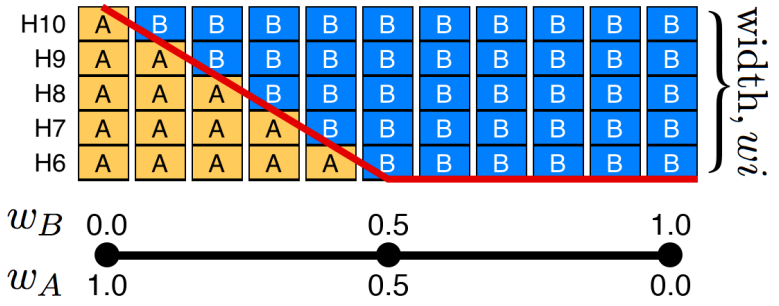

Figure 7.14: Ramp endpoints, as required by Equation 7.1, are defined as shown by the red line. The ramp *starts* at $w_B = 0.0$, not 0.5, as it is at the former that B-type cardinality – the quantity under consideration – starts to increase.

Table 7.2 once combined with the following rule which dictates higher positioning in the SSH band for B-type harmonics than for those of type A.

$$Index_{\textit{B-based harmonic}} > Index_{\textit{A-based harmonic}} \tag{7.3}$$

Had the left-hand option of Figure 7.11 been selected as generating the best 50/50 blend (thus using $\langle SSH \rangle = \langle 5a \rangle$ at point M instead of $\langle 5b \rangle$), an equally valid SSH band of different structure would result. Equation 7.1 yields the cardinalities for this alternate composition as well, but requires $w_{B_{start}}$ and $w_{B_{stop}}$ to be incremented by 0.5 to reflect a necessary shift in ramp location. The ramp direction needs to switch too, achieved by inverting the priority rule to become

$$Index_{\textit{A-based harmonic}} > Index_{\textit{B-based harmonic}} \tag{7.4}$$

which yields the A-centric structure of Table 7.3.

Table 7.1: SSH band cardinalities obtained from Equations 7.1 and 7.2 with blending point moved between two input motions in 0.1-size steps.

Weight B	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Type A Cardinality	5	4	3	2	1	0	0	0	0	0	0
Type B Cardinality	0	1	2	3	4	5	5	5	5	5	5

Table 7.2: SSH structure over one-dimensional blending space obtained by combining the priority rule of Equation 7.3 with the cardinalities of Table 7.1.

Weight B	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
H10	a	b	b	b	b	b	b	b	b	b	b
H9	a	a	b	b	b	b	b	b	b	b	b
H8	a	a	a	b	b	b	b	b	b	b	b
H7	a	a	a	a	b	b	b	b	b	b	b
H6	a	a	a	a	a	b	b	b	b	b	b

Table 7.3: Alternative SSH pattern befitting the left-hand option in Figure 7.11. It remains of the higher-compatibility type, as in the lower structure of Figure 7.13.

Weight B	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
H10	a	a	a	a	a	a	a	a	a	a	b
H9	a	a	a	a	a	a	a	a	a	b	b
H8	a	a	a	a	a	a	a	a	b	b	b
H7	a	a	a	a	a	a	a	b	b	b	b
H6	a	a	a	a	a	a	b	b	b	b	b

7.5.4 Extension to Two-dimensional Blending Space

The above one-dimensional two-input blending is now extended to the two-dimensional three-input case for use throughout the triangle network, in accordance with the following criteria:

- The 1D 2-input behaviour expanded on above should exist at triangle edges.
- For points within the triangle, the distance to each edge should determine the

extent of a contribution of its 1D edge behaviour, to be used and merged with appropriate contributions from the other two edges.

- While edge behaviour itself is by design asymmetrical, as laid out in Sections 7.5.2 and 7.5.3, three-way symmetry should be ensured to prevent bias towards any one edge.

- Within the limits imposed by the discrete nature of SSH harmonics (these being of one type or another and never merging), SSH band structure should be a continuous function of the blending weights thus varying smoothly as the blending point is moved.

To determine the contribution from a given side of the triangle, the blending point P is initially projected onto it, by moving P along line VP to its intersection P' with the edge, where V is the opposing vertex (Figure 7.15).

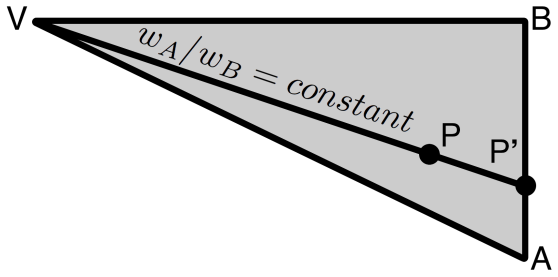

Figure 7.15: Projection of blending point P onto a triangle edge as the initial step in determining its contribution to the motion-type cardinalities underlying the final SSH stack applicable at point P. Weightings w_A and w_B are barycentric coordinates.

Assuming the input motions associated with the edge endpoints are A and B, the cardinality K_B at point P' would be given, as before, by Equation 7.1. However, the same result is obtained by forfeiting the projection process, considering only point P and using the enhanced formula

$$K_B = \min\left(\max\left(\left(\frac{w_B}{w_A + w_B} - w_{B_{start}}\right)\frac{wi}{w_{B_{stop}} - w_{B_{start}}}, 0\right), wi\right) \tag{7.5}$$

where previously used variables have the same meaning as in Equation 7.1 and w_A is the weighting for motion A. The floor function seen in Equation 7.1 is absent here as rounding in the 2D case is performed later.

Having obtained K_B, K_A succeeds, as before, via Equation 7.2. Furthermore, similar formulae to Equation 7.5 effectively project onto the other two sides, from which a further two pairs of single-source harmonic type cardinalities ensue ($\{K_B, K_C\}$ and $\{K_A, K_C\}$), one for each of the two remaining triangle edges.

The above cardinalities apply only for the blending point projections onto the triangle periphery, and suitable weightings, specific to the current blending point location, are required to specify the extent to which each edge's input motion cardinality pair is to contribute to the final SSH band structure. This is achieved by dividing the triangle into three parts, as shown in Figure 7.16.

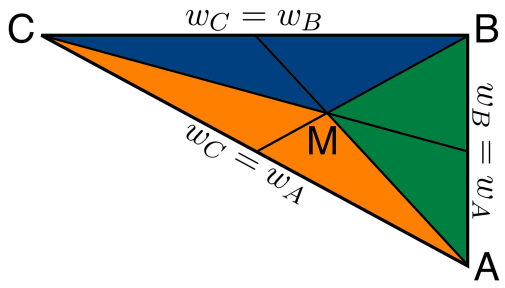

Figure 7.16: Triangle medians defining three sub-triangles AMB, BMC and CMA. Computation of the SSH stack takes account of which sub-triangle the blending point is currently located in. w_A, w_B and w_C are barycentric coordinates.

Triangle medians define the coloured sub-triangles, whereby for each point along a median, two of the barycentric coordinates are always equal as indicated in the figure.

Inequalities arise either side of a median, providing a convenient test for sub-triangle occupancy.

In whichever sub-triangle holds the blending point, such as AMB (Figure 7.16), three separate weightings are used to dose the extent to which each of the three previously obtained projection-based SSH cardinality pairs are to be applied:

$$w_{AB} = -2w_C + 1 \tag{7.6}$$

$$w_{CB} = (1 - w_{AB})/(w_B + w_A)w_B \tag{7.7}$$

$$w_{CA} = (1 - w_{AB})/(w_B + w_A)w_A \tag{7.8}$$

where w_A, w_B and w_C are the conventional barycentric coordinates as used for blending the NB band, while w_{CA}, w_{AB} and w_{BC} are *edge* weightings, to be applied in respect of edges CA, AB and BC. It will be noted that w_{AB} ranges from 1 at the triangle periphery to $\frac{1}{3}$ at its centroid, and that w_{BC} and w_{CA}, also of value $\frac{1}{3}$ at the centroid – thus complying with the requirement for three-way symmetry – represent a weighted share of the contribution lost by w_{AB} as the blending point moves away from edge AB. Similar equation triples apply to each of the other two sub-triangles.

Figure 7.17 plots the edge weightings over two triangles in the implemented SSH-switching-equipped network, making clear the discontinuous nature of such weightings over much of the sub-triangle boundaries within either of these two triangles, especially when further away from their centroids. Such discontinuities do not impinge on SSH switching smoothness, however, as shown later in Section 7.8 (Results).

Combining edge weightings as in Equations 7.6 to 7.8 with edge-based SSH stack values as from Equations 7.5 and 7.2 gives the conglomerated SSH stack cardinalities for the blending point location within the 2D blending space:

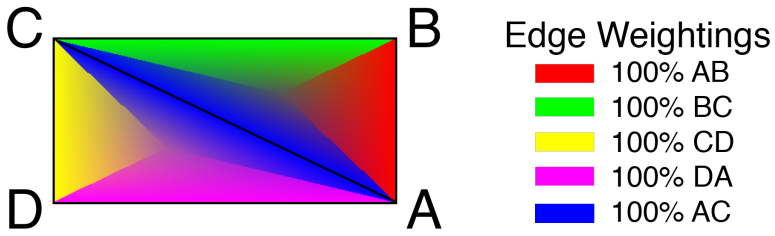

Figure 7.17: Edge weightings used to control the extent to which each triangle edge contributes to that SSH stack which would apply, if the blending point were projected onto the edge in question. 100% contributions are found only at the periphery of the two triangles shown, while influence at their centroids (where sub-triangles meet) is 33% from each edge.

$$KTot_A = w_{CA} \cdot K_{CA_A} + w_{AB} \cdot K_{AB_A} \tag{7.9}$$

$$KTot_B = w_{AB} \cdot K_{AB_B} + w_{BC} \cdot K_{BC_B} \tag{7.10}$$

$$KTot_C = w_{CA} \cdot K_{CA_C} + w_{BC} \cdot K_{BC_C} \tag{7.11}$$

where w_{CA}, w_{AB} and w_{BC} are the *edge* weightings as before, and K_{XY_Z} is the cardinality for Z-type input motion obtained by considering edge XY.

A rounding function provides the final three integer cardinalities for the current SSH stack. The previous, conglomerated values were floating-point values, total value wi, whose rounding to integer however, may result in a sum of $wi + 1$ or $wi - 1$, which no longer matches the stack size. In the former case, a correction is made by decrementing the smallest cardinality by 1, while the latter requires incrementing the largest by 1. The choice of which cardinality to amend may actually be of insignificant consequence, yet a strategy is required as the increment or decrement operation itself *is* necessary. The approach here is to make any change in such a direction as increases the homogeneity of the final SSH stack, on the grounds that combining different motions tends to increase distortion. Admittedly, a more sophisticated technique would also

take account of the current composition of the NB band, whose interaction with stack cardinality after rounding-based changes is at least hypothetically relevant.

The actual *order* of the harmonic types in the stack, thus the vertical priority for the final integer cardinalities, is controlled by a rule of the type seen in Equation 7.4 but now arranging all three input motions, instead of two. Its derivation becomes visible in Figure 7.14, where the SSH stacks forming the ramp section show B to lie higher than A, an ordering denoted here as B/A, and which in fact is that applicable to side AB in the upper triangle of Figure 7.17. The SSH structures (not shown) for the other two sides of the triangle are found, in a similar way, to dictate B/C and A/C ordering. The overall triangle-wide rule is thus:

$$Index_{C\text{-}based\ harmonic} < Index_{A\text{-}based\ harmonic} < Index_{B\text{-}based\ harmonic} \tag{7.12}$$

It will be clear that priority rule creation is subject to certain constraints. To ensure continuity when crossing from triangle to triangle, edge AC in the lower triangle of Figure 7.17 *must* adopt the same of the two possible SSH structures as used by AC in the upper triangle. (The two types of structures were set out in Tables 7.2 and 7.3). Identical SSH structures demand identical motion-type ordering, hence A/C is required as before.

Another restriction is that not all pairwise demands are necessarily compatible. For triangle inputs X, Y and Z, the priorities X/Y, Y/Z and Z/X cannot simultaneously be met. In such a case, one of the SSH structures will need to be changed to the alternate form, consequently reversing, for example, Z/X to X/Z, which in turn enables the overall rule X/Y/Z honouring each of the edge-requirements.

As shown in Figure C.1 of Appendix C, the above-described mechanism for SSH stack control is integrated within the overall system just prior to the blending stage, with

blending enhanced so that in addition to being able to operate conventionally, as with the streamlined approach or with hybrid networks, it can also, instead, as required by SSH switching, simply copy harmonic content from a specified input motion in respect of any given harmonic.

7.6 Merging SSH Switching with Ordinary Blending

Although SSH switching can be left enabled throughout the network, thus even in areas with negligible UH distortion, the option does exist to activate it only where strictly needed.

This merely necessitates tagging vertices in SSH-free parts of the network as abstaining from their role in SSH switching, and having normal blending replace the one-motion-only contribution which the affected harmonics thereby lose. Thus, when the switching mechanism seeks to label a harmonic in the stack as receiving input from the motion at an abstaining vertex, it instead receives a label commanding normal weighted-average blending. The SSH stack is, of course, by definition normally devoid of normal blending, but an exception is made when merging areas of pure SSH switching with those of purely ordinary blending. The mode change goes unnoticed, as the existing switching process introduces normal blending in a gradual step-wise fashion, and furthermore, UHD is not experienced, as the change from SSH to conventional blending would only be made to occur in areas deemed free of distortion. Further details are provided in Section 7.9 (Discussion).

7.7 Error Measurement

This section describes the procedure used to measure UH distortion, and thus to quantify the effectiveness of the presented method used to counteract it. While – as mentioned above – UH distortion is not limited to the feet, the foot-node trace of Section 7.3

indicates a useful basis for its measurement. The symptoms in the foot trace are side-to-side undulations, and it is these which the employed metric quantifies.

7.7.1 Frame of Reference

The foot traces in Figures 7.1, 7.4 and 7.9 were created with skeletons walking on the spot, and it is the specific restrictions and freedoms applicable to their motion which specify the reference frame for distortion measurement.

- As mentioned in Section 7.3, the root x and z rotations are not locked and hence do modulate the extent to which the character leans forward, backwards or sideways during the walk cycle, which in turn affects the foot-node (world) coordinates. Foot positions are to be measured, including the effect of these root rotations, which precludes the use of body coordinates as these are impervious to rotations at root level. In this respect the reference frame is similar to a world reference frame.

- However, in world coordinates, a foot in the ground contact phase remains at a fixed location, and strictly so when using foot constraints as applies to the implemented system. Measuring a foot *trace*, however, requires by definition that the foot change position relative to the frame of reference throughout this phase. This was achieved by having the skeleton walk on the spot, which is equivalent to locking the coordinate axes relative to the projection of the root in the ground plane. The resulting imperviousness of measurements to the skeleton's translation in the ground plane is a characteristic shared by body coordinates.

- Similarly, to assist numerical foot trace comparison, y-root rotations remain locked, as before, (Section 7.3). This has the same effect as locking the reference frame to the skeleton's y-axis root rotations, and thus making measurements

impervious to these particular root rotations – again a feature found in body coordinates.

Measurements can thus be seen as either performed in world coordinates on a movement-constrained skeleton, or in a hybrid world-body coordinate frame, on an unrestricted character.

7.7.2 Reference Motion

The unwanted waviness in a UHD-corrupted foot trace modifies its curvature, and it is the quantification thereof which forms the basis of the chosen metric. However, an ideal blend would also exhibit curvature, being founded on input motions which possess this too. Distortion-free motion would thus yield non-zero curvature, so a reference is required against which to compare the synthesised trace under investigation.

Unlike the illustrative examples of Section 7.3, blending in the triangle network is between three input motions, not two. A weighted average of their node positions is used to yield the reference blend – a conjecture of course, but one in the absence of any formal definition of an ideal blend. Thus for each location at which distortion is to be measured, both a synthesised sequence and a reference motion are created, based upon the same blending point location. It will be remembered that interpolation of node positions, which clearly avoids UH distortion, was rejected in Section 7.3 for violating the constancy of skeletal segment lengths. Its use here is feasible, however, as only for the purpose of constructing a reference foot trace, not skeletal chains with their constituent segments. The resulting distortion measurements are thus relative values, made with respect to a distortion free standard.

The input motions in the implementations of this chapter (like those of Chapters 5 and 6) can have varying sequence lengths, but even when coincidentally equal, differences are still likely in the durations of the foot ground-contact phases themselves.

The contact stages for each input motion are thus resampled prior to blending, giving all an identical length. The number of samples converted to, be it 100 or 1000, is inconsequential, though should not be shorter than any of the contact stages themselves as downsampling induces a loss of informational content. Resampling was via Catmull-Rom spline. These befit the task as they pass through the control points that define them. An alternative would have been the twofold use – once for each dimension in the data – of the discrete Fourier transform followed by Fourier synthesis (or alternatively by the inverse discrete Fourier transform [1]). (The resampling ability of Fourier synthesis was specified in Chapter 5, Section 5.5.2). Foot trace interpolation then follows to create the reference blend.

Resampling as above is also performed on the synthesised output. The two foot-trajectory sequences – reference and output motion – are then ready for alignment and comparison.

7.7.3 Synthesis-reference Comparison

It is the *shapes* of the foot trace waveforms which are to be compared. Their actual location within the reference frame is not considered, as a positional offset of the entire sequence clearly has nothing to do with the jerkiness of UH distortion. The two sequences – synthesised and reference – can thus be translated at will for comparison, and are given overlapping starting points by placing each at the reference frame origin – a choice which assists the following trace rotation step.

For the treatment presented here, the orientation of a foot trajectory is defined as that of the line passing through its first and last points. Again, differences are found between output curve and reference, though divergence is small, the network-wide

[1] As shown in Appendix A, Fourier synthesis and the IDFT differ only in terms of input data format, this being harmonic magnitudes and phase angles for the former, and complex values for the latter.

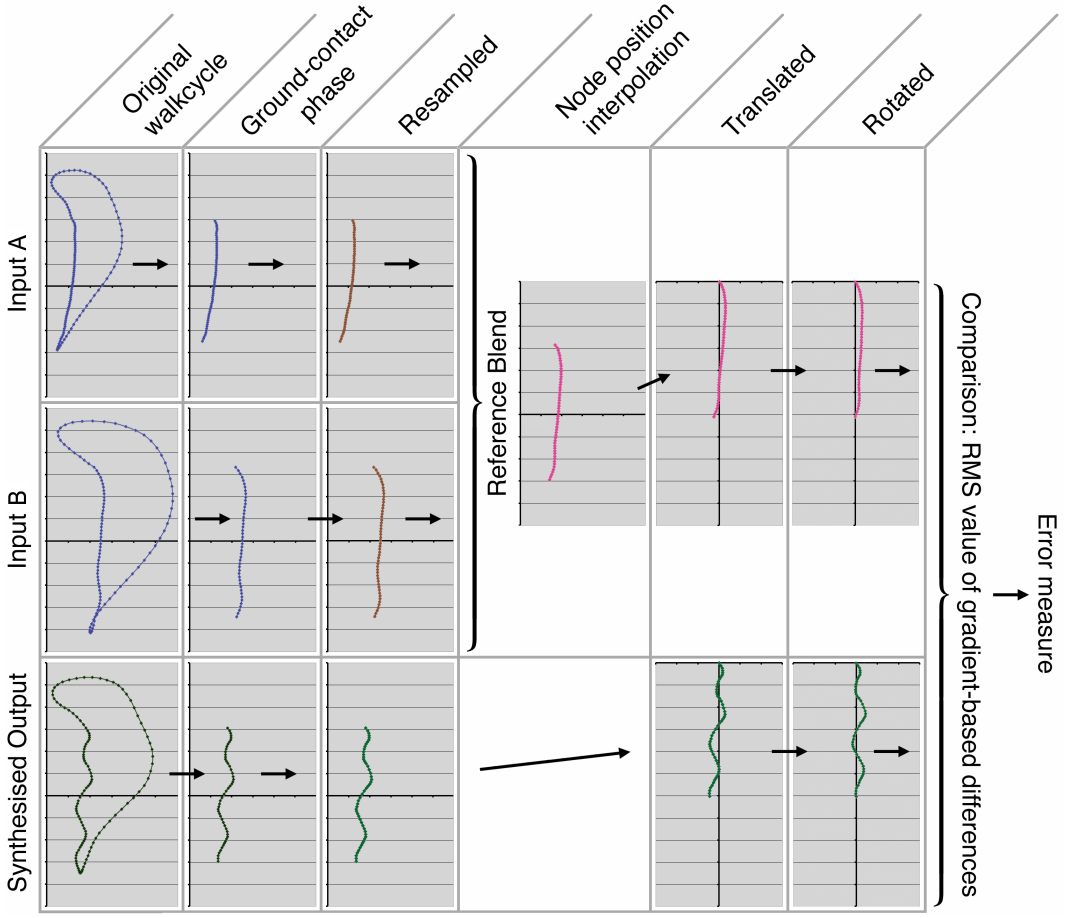

Figure 7.18: UH distortion measurement involving the creation of a reference blend against which to compare the synthesised output. Diagram is limited to left foot trace only. Actual measurement considers both feet and three input motions, not two.

maximum being only 0.079rad (4.5°) with an arithmetic mean of 0.026rad (1.5°). Furthermore, for a given sample in the distorted trajectory, the magnitude of the unwanted undulation is defined as being the extent of the deviation in a direction perpendicular to the trace orientation. Measurements which comply with this are thus orientation-invariant, making it quite acceptable to rotate the sequences about the origin making them parallel before their mutual comparison. The curves thereby maintain their start-points at the coordinate frame origin, and acquire endpoints on the negative z-axis (the

latter in compliance with the choice of having skeletons looking up the z-axis). The x-coordinate of each sample in these curve then conveniently gives the signed distance from sample to z-axis, facilitating the ensuing calculations.

The two curves, now aligned, are compared by evaluating, segment by segment, the difference between the arctangent of the first derivative of the synthesised output and that of the corresponding section in the reference sequence. The RMS value of the collected set of angle differences, then gives the error for one foot, to which, by repeating the entire process, is added that for the other foot, thereby giving the final UH distortion measurement for the foot trace. Arctangents are employed to avoid data spikes arising from near-vertical gradients occasionally exhibited at the sample-by-sample level. These would vastly increase the final error reading despite no corresponding distortion being visible in the animation. It is thus by comparing foot trajectory bearings over their entire length that the correspondence in curvature is reflected by an error value, with a lower correspondence yielding greater error. The complete measurement process is shown diagrammatically in Figure 7.18.

7.7.4 Pre-correction Distortion Plot

Repeating the above measurements throughout the network reveals the spread of UH distortion seen in Figure 7.19. All three plots show *pre*-correction UHD. The top two are normalised to span from shades of white ($error \simeq 0$) to blacks ($error \simeq max\ error$), using a linear colour scale with $\frac{max\ error}{2}$ represented by mid-grey. For added detail, error in the lowest 1% is shown blue, and network points in the top 1% – only one in this case – are shown red. At triangle vertices the blending weights, used by the synthesised output as well as in reference blend creation, are 1, 0 and 0, making both curves identical to the corresponding input motion, and thus mutually identical, explaining the blue highlighting at vertices. The third plot of Figure 7.19 is enhanced to reveal UHD

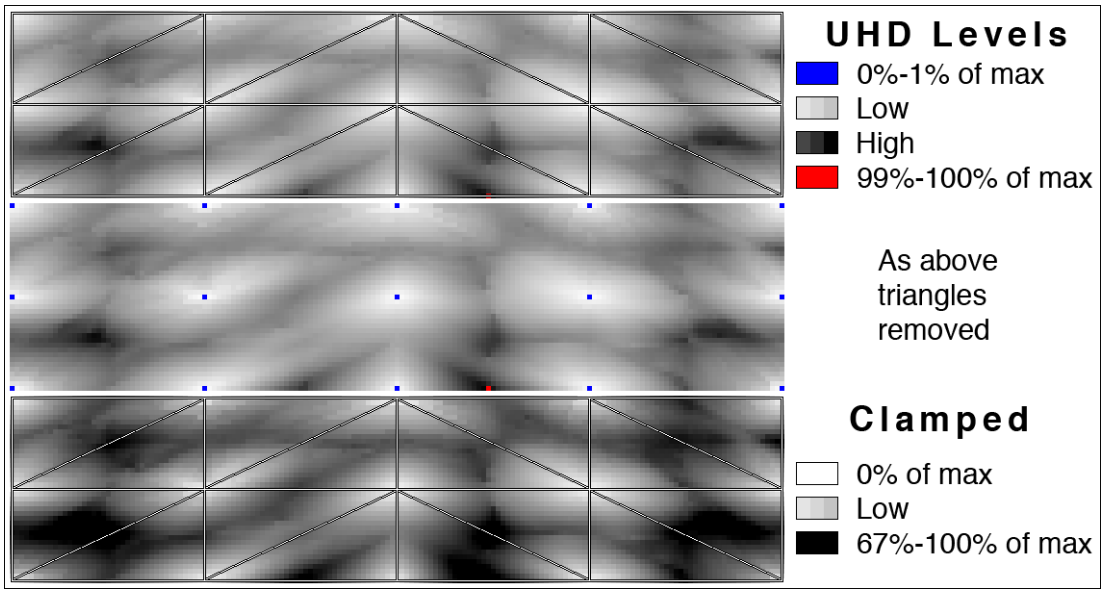

Figure 7.19: UH distortion throughout the walking section of the triangle network, blending with 10 harmonics, *prior* to SSH switching. The top two plots employ a linear colour scale, unlike the lower which is enhanced by clamping to emphasise UHD hotspots.

hotspots by means of a nonlinear colour scale which highlights distortion levels lying in the top third of the spectrum. The geographical spread of pre-correction UHD is discussed in Section 7.9 (Discussion).

7.8 Results

7.8.1 One-dimensional Example

The one-dimensional example of Sections 7.3 and 7.4 is considered first. Figure 7.20 shows the same blend as in Figure 7.1 but with SSH switching enabled (third screenshot) and a clear reduction in UH distortion compared to the UHD-afflicted use of conventional blending (duplicated in second image). Superimposing the distorted trajectories on the corrected ones, shows overlap throughout their lengths (fourth image), confirming SSH blending does no more than remove the unwanted undulations. Close

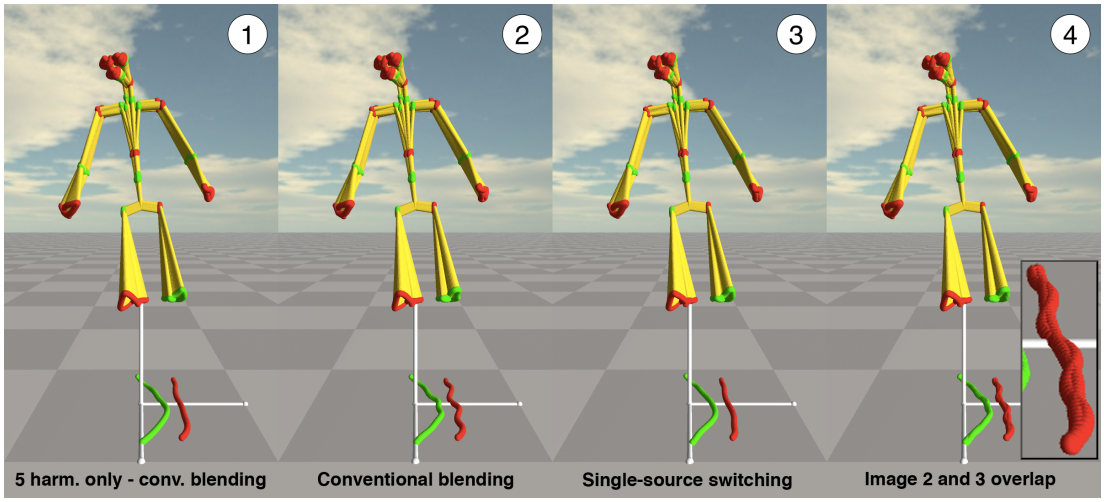

Figure 7.20: (1) Conventional blending made free of UH distortion by clipping to five harmonics, followed by unrestricted yet distorted motion (2), both provided for comparison with SSH-switched harmonically-rich undistorted motion (3). Final image (4), with inset, gives overlap of second and third, confirming the impact of SSH switching lies only in UH distortion reduction with no unwanted changes to the underlying motion.

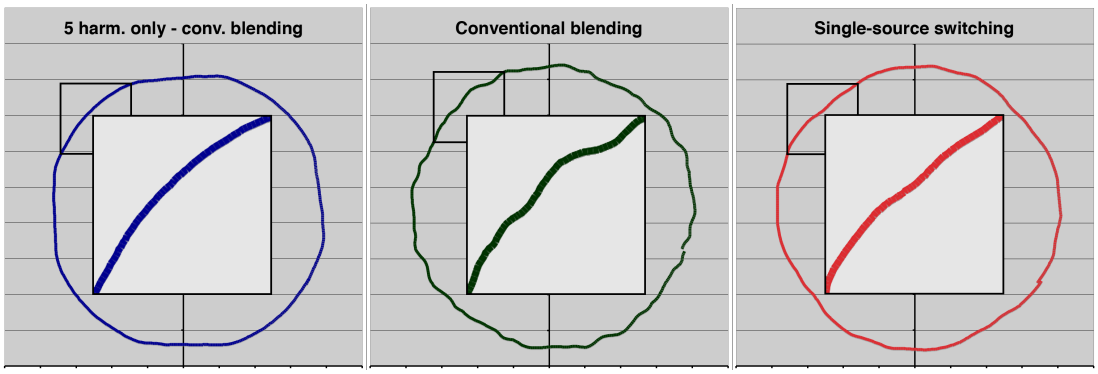

Figure 7.21: As Figure 7.20, but with the skeleton free to walk instead of being confined to the spot, whereby distortion then transfers to the root trajectory. Motion for this particular blend happens to be a tight turn, giving roughly circular root paths, with a portion of each additionally shown magnified. The effectiveness of SSH switching is seen as before.

Table 7.4: Proportion of plot samples having improved or worsened under SSH switching. Threshold is a percentage of the plot-wide maximum pre-correction UH distortion level. Only samples exceeding this degree of distortion increase or decrease count as having changed.

Threshold (% UHD max)	0%	5%	10%	15%	20%	25%
Sample points improved	75.0%	41.0%	19.5%	7.1%	2.7%	1.2%
Sample points worsened	24.3%	7.3%	2.1%	0.3%	0.0%	0.0%
Sample points unchanged	0.6%	51.7%	78.4%	92.6%	97.3%	98.8%

scrutiny reveals small differences between the SSH-blended trace (image three) and the trajectory for conventional blending using 5-harmonics-only (first screenshot). This is to be expected, as the latter is only free of distortion through being harmonically clipped, making it overly smooth, while the former avoids distortion despite its full (albeit one-sided) harmonic content, which inevitably adds subtle structure to the foot trace.

Section 7.3 emphasised how, depending on the implementation, foot-trajectory distortion can impact on root trajectory, thus making the entire skeleton jerky in motion. Figure 7.21 gives this alternative view of UHD, in which the skeleton was no longer confined to the spot but free to walk in a circle. UH distortion and its correction by SSH switching are seen as before.

Distortion measurement as described in Section 7.7 evaluates the reduction in UH distortion illustrated by Figures 7.20 and 7.21 to be a 61.3% drop for the left foot, and 44.3% for the right.

7.8.2 Post-correction Distortion Plot

A plot of network distortion during SSH blending is given in Figure 7.22. To enable visual comparison with the pre-correction distortion plot of Figure 7.19, the same colour scale is used in each. A visual assessment of the top two images in each Figure, indi-

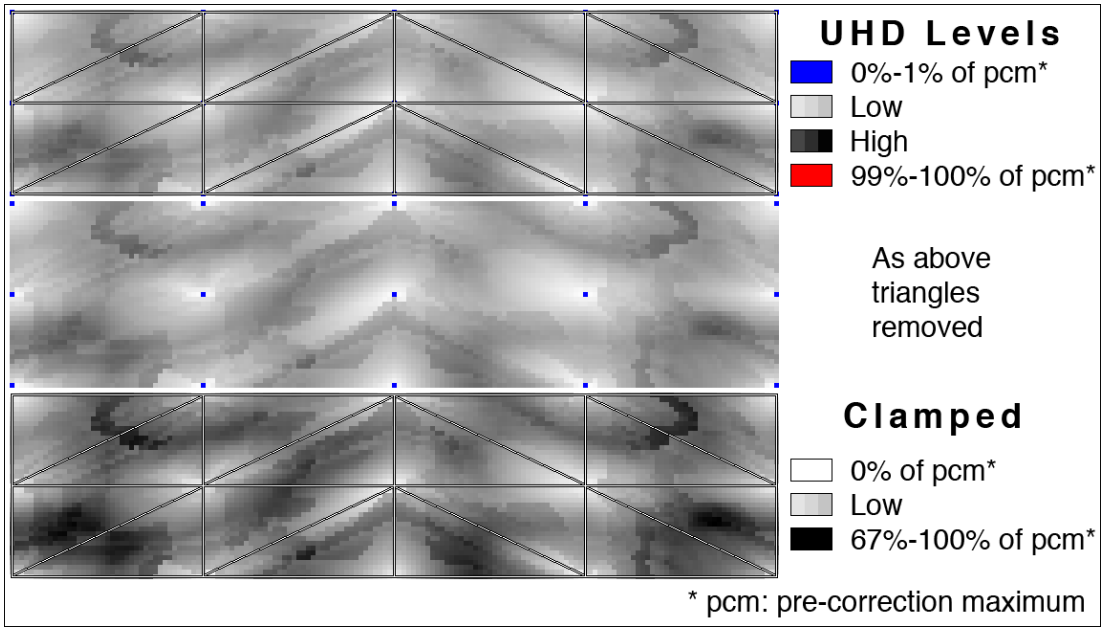

Figure 7.22: UH distortion *after* correction by SSH switching. For all three plots the colour scale is the same as in Figure 7.19 allowing direct comparison. The top two images show the worst (darkest) areas of UH distortion previously seen in Figure 7.19 to have greatly improved. The bottom image, highlighting distortion at the top end of the spectrum, exhibits far fewer pure black patches than existed before correction in Figure 7.19. Detailed examination reveals only one black sample point (257 initially), and hence the removal of 99.6% of hotspot-resident samples by SSH switching.

cates an overall benefit of SSH switching, with many sample points showing reduced distortion, while a lesser number seem to have worsened. Table 7.4 quantifies each type, for various improvement and deterioration thresholds. Thus while 75% of samples show improvement with 24.3% showing a loss of quality, this includes all levels of quality change, even those of little practical significance. Considering only *noticeable* changes, however, such as those exceeding 10% of the maximum pre-correction distortion level, shows considerable benefit to SSH switching with 19.5% of plot samples having improved, and only 2.1% having worsened. Furthermore, plotted values for post-correction data never reach the top end of the scale (red and shades of black),

which far from being an oversight simply indicates the efficacy of SSH switching.

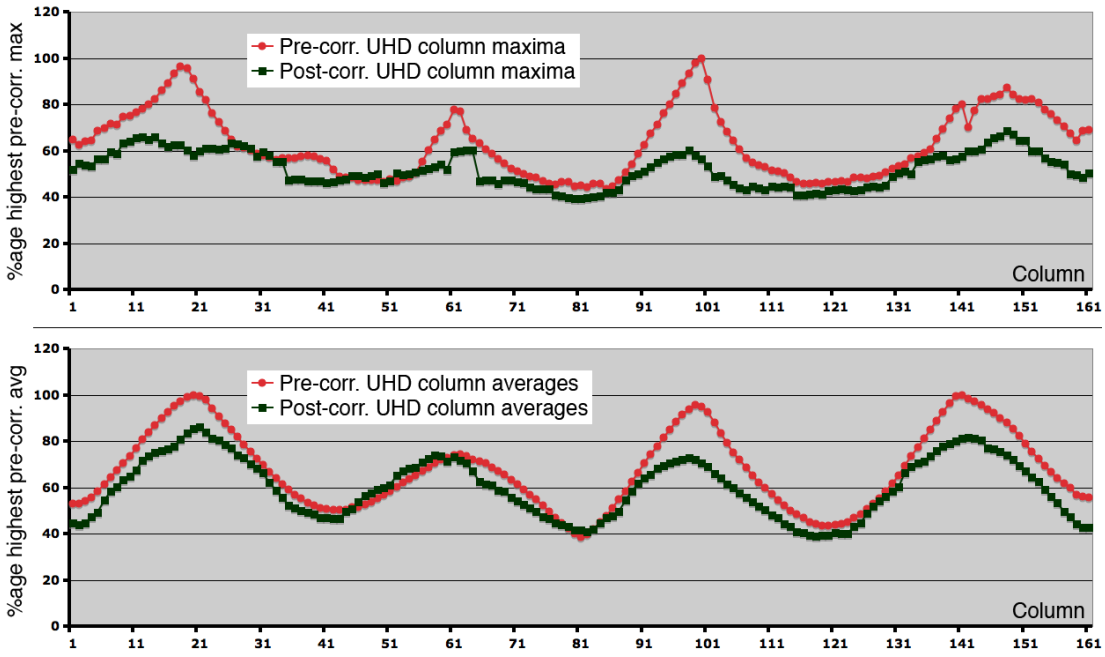

Figure 7.23: Maximum and mean distortion levels for each of 161 column in the plots of Figure 7.19 and 7.22. For most columns SSH switching significantly reduces UHD, especially regarding distortion hotspots shown by peaks in the pre-correction maximum-value graph.

It is distortion at the higher end of the scale which is the most irritating to the user, and most in need of correction. To further quantify the effectiveness of SSH-switching at this level, UHD hotspots are defined as clusters of samples whose distortion magnitudes lie in the top-third of the maximum experienced network-wide. In the bottom image of Figures 7.19 (pre-correction UHD levels) and 7.22 (post-correction levels), distortion values at or above 67% of the pre-correction maximum were clamped before normalisation, thereby depicting all hotspot samples in pure black. This view reveals a strong improvement after SSH switching, with pure black patches all but eliminated. More precisely, only one sample (0.015% of the blending space) remains within the top-third-distortion threshold compared to 257 (3.9%) before correction,

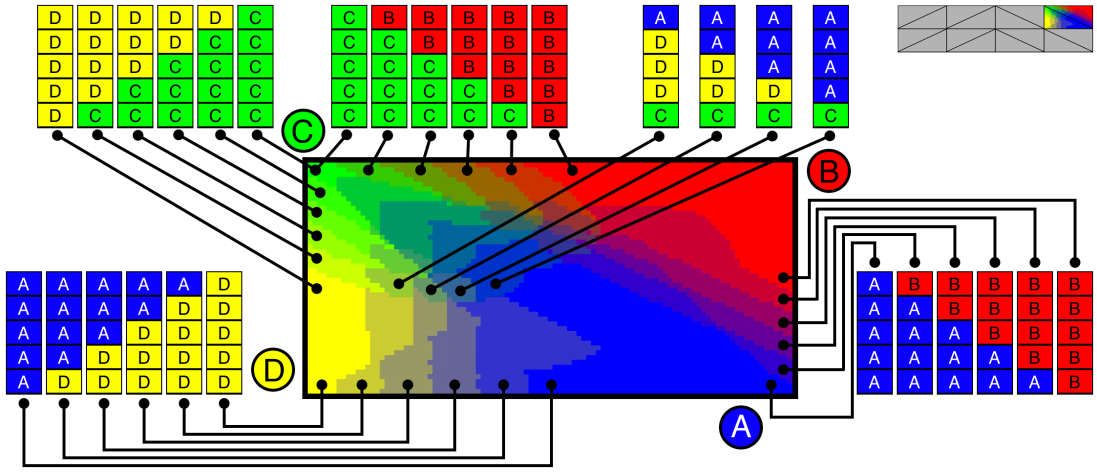

Figure 7.24: SSH structure over two adjacent triangles in the walking network, with selected stacks enumerated. Changes – although inevitably discrete in a switching structure – proceed in gradual manner as the blending point travels within the blending space.

thus 99.6% of hotspot-level samples dropped out of the hotspot zone.

To give the colour scale added meaning, it is here specified that the blended trajectories shown in Figures 7.20 and 7.21 refer to a point on the very right-hand edge of the network, one quarter of the way down from the top. This point, as seen in the distortion plots, becomes much lighter with SSH enabled.

Each column in the network plots has a maximum and mean distortion level, shown in the graphs of Figure 7.23. They depict distortion levels post-correction to be, in general, well below the uncorrected values, especially in respect of UHD hotspots indicated by peaks in the pre-correction UHD-column-maxima plot.

7.8.3 Switching Pattern

The SSH stack structure for two contiguous triangles is given in Figure 7.24. Within the limitations imposed by switching being a discrete process, the stack is seen to change composition gradually as the blending point moves within the network. The smoothness is further highlighted in Figure 7.25 which presents the extended SSH-structure over

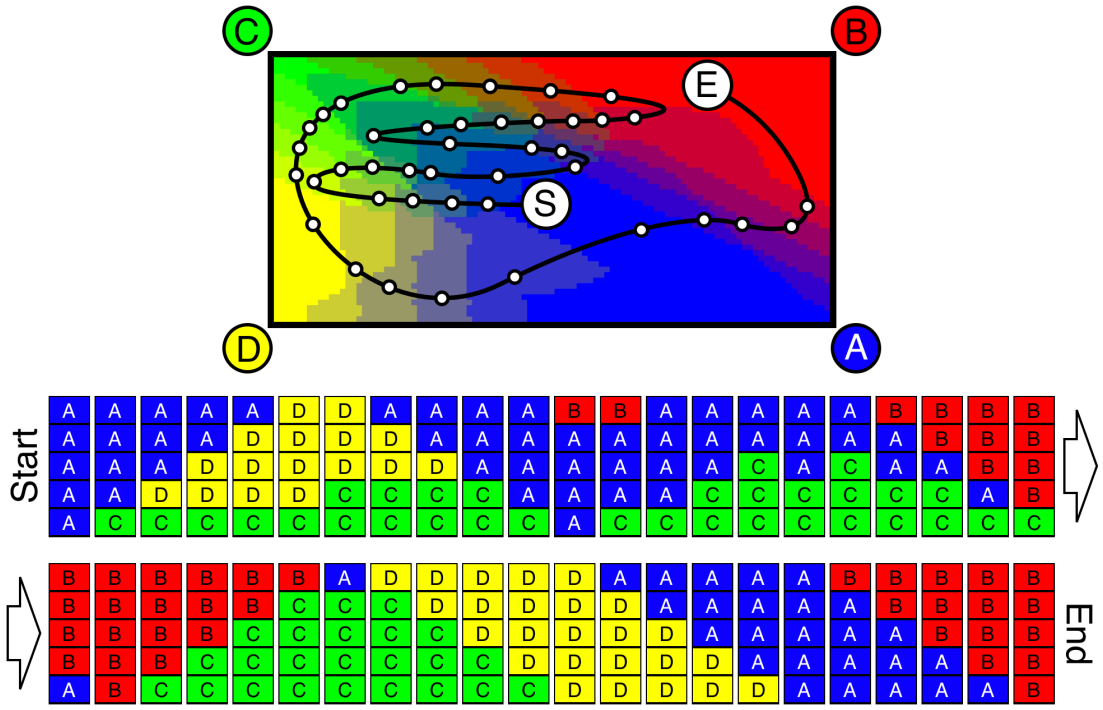

Figure 7.25: Extended journey throughout the space of Figure 7.24. Stack hierarchy rules (B/A/C and A/D/C) are honoured with cardinality changes never exceeding unity.

a path cut throughout the blending space. Changes in SSH stack ordering are indeed seen to fulfil the continuity specification at the start of Section 7.5.4, despite the sharp edge weighting discontinuities underlying stack creation (Figure 7.17).

7.8.4 Demonstration Video

A video file demonstrating SSH switching can be downloaded from `http://www.urbanmodellinggroup.co.uk/SSH_Switching.mp4.zip`. This link is also found in Appendix E.

Animation is shown with and without SSH switching, for comparison. It concentrates on parts of the network which experience some of the worse cases of UH distortion, and illustrates the successful reduction in UHD described in results above. Moving the

blending point throughout the network is shown to create no popping in the animation, despite the all-or-nothing nature of single-source harmonics.

7.8.5 Runtime Cost

SSH switching incurs no overall cost. Fourier synthesis was left unchanged, though a considerable efficiency gain is possible as discussed in the next section. It is the *blending* stage prior to synthesis which is under consideration here. As was seen in Chapter 5, Table 5.1, for the streamlined approach, and Chapter 6, Tables 6.2 and 6.3, regarding hybrid networks, the blending phase in frequency-domain-based motion creation is minor in comparison with the ensuing process of Fourier synthesis. Nonetheless, the overall cost of this blending stage proper is further reduced with SSH blending as explained in the following paragraph.

Each time blending is required, thus each time the user-input changes, a small 2.9µs overhead is required to determine the SSH-stack structure required for the current position of the blending point. In the interest of clarity, and to differentiate this stage from the actual blending/switching stage, the term 'stack design' will be used to denote the process. The once-per-blend stack design cost is greatly outweighed, however, by the time saved in avoiding having to blend the SSH band harmonics for each skeletal DOF, and instead being able to use a simple copy process to generate them. Table 7.5 compares the blending costs with SSH switching both on and off, giving timings for both Euler angles and quaternions, and furthermore allows comparison with the methods of previous chapters[1]. Blending times are expressed per DOF, and, for consistency with the rest of this chapter, assume the use of 10 harmonics (DC

[1]Hybrid network blending times were given per-frame, per-DOF, in Chapter 6, as explained in that chapter. To assist comparison the same measurements are expressed simply per DOF in Table 7.5, hence the apparent difference.

Table 7.5: Blending times with SSH switching enabled (the normal operating mode) and disabled (optional), as well as a comparison with the methods of Chapter 5 and 6. Despite including all the features of hybrid networks (Chapter 6), the blending times in the SSH switching implementation, when enabled, are seen to be markedly lower.

Blending times per DOF using 10 harmonics (μs)	Streamlined approach	Hybrid networks	SSH switching disabled	SSH switching enabled
Euler angles	1.18	1.61	1.81	1.27
Quaternions	—	2.10	2.24	1.64

plus 5 NB band harmonics topped by 5 SSH band harmonics when SSH enabled, and DC plus 10 NB band harmonics when disabled), whereby times would correspondingly drop using lower harmonic counts. The stack-design overhead, expressed per DOF, is 0.042μs. Measurements were obtained from the 69-DOF skeleton shown in Figure 7.6, which was animated at 100 frames per second with a 113-frame walk cycle.

The third column in Table 7.5, refers to the implemented SSH-equipped network, with SSH switching disabled, thus with a stack configuration comprising the DC value plus 10 *ordinarily blended* harmonics. The fourth column refers to the same implemented software with SSH switching enabled, and hence a stack configuration comprising the DC value, 5 blended harmonics and 5 SSH-switched harmonics above them. Ordinary Euler angle blending times per DOF inevitably increases as capabilities are added to the system (1.18μs then 1.61μs then 1.81μs). The same applies to ordinary blending of quaternions (2.10μs then 2.24μs with none having been used in the streamlined approach). Enabling SSH switching reduced the SSH blending cost (ie that of blending, or alternatively switching, harmonics in the entire DC-to-H10 stack) for Euler angles by 29.8% and that of quaternion blending by 26.8%. In order to show the costs of SSH blending itself, the stack design overhead is not included in Table 7.5. This would simply require the small increase of 0.042μs to all values in the right hand half of the table. Even then, however, the blending phase of the SSH implementation, would,

when enabled – and there is no need to ever disable it – remain significantly cheaper than hybrid networks despite including all their functionality (1.31µs down from 1.61µs in hybrid networks, and 1.68µs down from 2.10µs).

As in previous chapters, results were obtained on an Apple MacBook Pro *laptop* with Intel Core 2 Duo CPU running at 2.53 GHz, with 4GB RAM, and code compiled with gcc.

7.9 Discussion

7.9.1 UH Distortion Geographical Spread

As seen in Figure 7.19, during conventional blending, UH distortion on triangle edges is worse near their midpoints. It falls on approaching the vertices because the 1.0, 0.0, 0.0 weightings applicable there, applied to all harmonics, mean blending within the UH band – the locus of UH distortion – has ceased. Similarly, there is a tendency for the sub-triangle connecting edge midpoints to exhibit UH distortion along its *own* edges, and often within them as well. The weightings outside this zone are always > 0.5 for the closest vertex, with greater proximity leading to greater magnitude. The one-sidedness of the blending weights is thereby increased, reducing the extent to which motions are merged in the UH band, so lower distortion is experienced outside the sub-triangle, reaching zero, as before, at the vertex.

7.9.2 UHD and SSH Switching Anatomical Scope

UH distortion was analysed and measured by consideration of its presence in foot trajectories, which, if feet are constrained to the ground during their contact phase, leads to root path distortions and jerks affecting the entire skeleton. But even with the root held immobile, distortion is found throughout the skeleton, as, for example, in the trajectory of the hand. *All* occurrences lie within the scope of the presented method,

however, as SSH blending, being applied to all degrees of freedom, counteracts UHD whichever the afflicted node.

This chapter focussed on walking motions only, but the potential for UHD in other types should not be dismissed. Its impact, though, will surely vary. Running, for example, with its shorter stance-phase and more ballistically defined motion, will not be affected in the same manner and could never exhibit a distorted root path as severe as seen in the middle of Figure 7.21.

7.9.3 Error Measurement

The perceived benefit of SSH blending, as visible in the foot trace screenshots of Figure 7.20, might, arguably, be judged to exceed measurement-based evaluations of gain. For example, the figure shows – to subjective human judgement – that SSH switching has all but removed, if not entirely eliminated, UH distortion for the left foot, whereas the measured reduction was "only" 61.3%. Such discrepancies are not unexpected, however, as UHD measurements are relative to a conjectured hypothetical ideal blend, which even motion entirely free of UH distortion might fall short of by a significant margin. Thus, all but fully corrected motion, which of course would greatly resemble motion entirely free of UHD, would, in the absence of a true standard to describe the latter, be measured against the *surmised* reference standard, which, while exhibiting some variance with the true yet unknown reference, would lead to elevated error readings for the SSH-corrected motion, with a consequent reduction in the calculated percentage drop to post-correction level.

7.9.4 SSH Structure Asymmetry

Network triangles and their allocated motions indicate no bias towards any particular vertex, so the asymmetry of SSH patterns, as seen in Figure 7.14, may be unexpected,

perhaps found almost unwieldy. This suggests the ramp, instead of lying to one side of the centre, might span the entire width of the structure. Further examination, however, shows this to lower the compatibility between the SSH band and the underlying NB band, with potential for severe distortions at one end of the blending space, thereby reducing SSH switching effectiveness. The problem would be the same as was found, in Section 7.5.2, to apply to the upper structure of Figure 7.13.

Centre-structure asymmetry is a necessity too, exhibiting absolute bias towards one endpoint or the other, as in Figure 7.11, where only two options exist: either $\langle SSH \rangle = \langle a, a, a, a, a \rangle$ or $\langle SSH \rangle = \langle b, b, b, b, b \rangle$. Interleaving harmonic types, yielding $\langle SSH \rangle = \langle a, b, a, b, a \rangle$ or $\langle SSH \rangle = \langle b, a, b, a, b \rangle$ might appear preferable, as less one-sided and certainly more compatible with the 50/50 blended NB band below it. However, while interlacing in the UH band is fundamentally different from blending within it, it nevertheless exhibits UH distortion – as induced by the latter – though to a lesser degree. This is no surprise, as an interleaved stack represents more of a mixture than one of single type, and also more than one comprising two (or three) clusters of homogenous motion types, as often found in SSH blending.

The ramp section, which thus resides within a half-width of the SSH structure, is made to span this half-width in full to keep it as shallow as possible. This reduces the rate at which harmonics need to switch from one motion to the other when traversing the structure, maintaining it below that level at which popping might appear in the animation.

7.9.5 Merging with Conventional Blending

Section 7.6 described how SSH-blended zones can co-exist with conventionally blended ones, the former comprising vertices which contribute to SSH switching, and the latter made of ones which abstain. The boundaries of such zones will run through triangles

with vertices of both types, for which the SSH stacks, while similar in principle to those in the patterns of Figures 7.24 and 7.25, will, instead of comprising up to three single-source harmonic types, comprise only one or two, with the remaining harmonics being normally blended, and corresponding diagrams thus requiring an update to show type 'NB'. SSH patterns in such zone-interface triangles therefore retain the gradual stepwise changes seen within zones of unmitigated single-sourcing, and provide smooth popping-free transitions between areas which enable SSH switching and those which do not. The presence of NB harmonics within the stack, normally disallowed as the very cause of UH distortion SSH switching is designed to avoid, is not an issue, simply because transition to SSH-free zones would only be desired in areas where distortion is not a problem. Furthermore, incompatibilities between the stack and the NB foundation below it do not arise, since the replacement of single-source harmonics with NB-type harmonics, makes them closer in nature to the NB band they rest upon.

This approach could be modified to a miniature scale for localised use, avoiding the rare sample clusters for which distortion noticeably worsened in the plot of Figure 7.22. In brief, a map based on the network-wide evaluation of SSH switching effectiveness could highlight points where the SSH method is to be overridden, either by conventional blending (thus leaving UHD unchanged) or by harmonically clipped motion (thus avoiding UHD but reducing the animation to lower quality). The latter seems the better choice, as a momentary switch to lower quality while the blending point crosses the overridden points would go unnoticed. Either way, post-correction motion would be improved overall, as before, but, additionally, this enhanced approach would never manifestly worsen UH distortion at any point.

7.9.6 Further Work

The Fourier synthesis process for the implementation of this chapter is identical to that used in the streamlined approach (Chapter 5) as well as to that in the blending portion of hybrid networks (Chapter 6). SSH switching thus imposes no additional cost on the synthesis process. It could, however, significantly reduce it, because instead of using Equation 5.10 of Chapter 5 to calculate the time-domain contribution of SSH-switched harmonics, these values, which are both of feasible cardinality [1] and determinable in advance, could be correspondingly computed in preprocessing.

SSH switching is distinctly beneficial overall, but limited network sections were seen to suffer non-trivial detriment. Further research into potential incompatibilities between the SSH and NB bands might alleviate this.

While the approach to UH measurement possibly dampens the reported benefit of SSH switching, it does highlight areas of perceived distortion well, and properly indicates which of two distortions is the greater or smaller. This makes possible the development of a fully automatic set-up process, where all network parameters are chosen algorithmically, selecting those which generate the best output motion quality.

7.10 Conclusion

This chapter focussed on a puzzling artefact at times experienced during motion interpolation work associated with this thesis. Investigation revealed it to exist at the higher end of the spectrum of harmonics found necessary [UAT95, PL06] for best-quality animation, hence the name 'upper-harmonic distortion'. The phenomenon was shown to

[1] Every SSH band harmonic creates one real value (time domain contribution) for each frame of output motion. Preprocessing these should account for every possible output sequence length, and would be for each DOF of each input motion.

be inherent in – though not necessarily limited to – the blending of walking animations, be it in the time or the frequency domain.

A novel solution, single-source harmonic switching, was introduced and explained in depth, with a video of a full working implementation provided to demonstrate the effectiveness of the method. Additional confirmation was provided by the design of a UH distortion measurement technique, which revealed notable improvement in network-wide plots created with SSH blending compared to those without.

Being applied only to certain harmonics, SSH switching is conditional on blending being performed in the frequency domain. The field of Fourier blending is thereby enhanced, gaining a feature not available in the time domain.

8

Conclusions

8.1 Introduction

This chapter serves a twofold purpose. Firstly, it consolidates this thesis, by providing a reminder of its context and a summary of the work undertaken, while reiterating key contributions, thus presenting an overview of this long-term project. A secondary goal, however, is to clarify the *motivations* underlying implementation Chapters 5, 6 and 7, and thereby to reinforce the approach undertaken.

8.2 Thesis Context

While their value, both good and less so, is worthy of discussion, one aspect of computers is beyond question, their ever-increasing role in life today, with many a human activity finding new expression in a digital medium. It is ironic, and telling, perhaps, that this electronic facet of modern society includes virtual environments, reflections of a real life temporarily left behind. Extensive online server-based worlds like Second Life, [Incb], OpenSimulator, [Ope], and ActiveWorlds, [Inca], invite interplay between users who otherwise might never have met, in an engrossing realm of make-believe, allowing an escape from reality much valued by its users. Interactivity is provided by video game consoles too, with local storage and dedicated hardware enabling visually striking

worlds which, despite their artificial nature, satisfy a desire for experience which life itself often fails to provide. Serious uses are plentiful as well [Fre08, BEL02, Sie], with education, health, commerce and tourism just some of the areas with a virtual presence. It is such worlds, expansive or modest, in their multifarious kinds, which require animation by characters and provide – as does the active research area of character animation – the context for the work presented in this thesis.

8.3 Research Motivation

Previous work was covered in Chapters 2, 3 and 4 and included keyframed characters in the hand-drawn process of traditional animation [Tho58, TJ95] which was later emulated in computerised form [Las87, Stu98a], and itself followed by the evolving field of capture-based synthesis still at the forefront today, whose three main categories have comprised motion interpolation [PSS02, RCB98, GBT04b], synthesis by concatenation [KGP02, LCR$^+$02, AFO03] and statistical methods [WMC11, LWS02, CH07]. Interpolation, more usually performed in the time domain [PSS02, WH97] can also be applied to individual frequencies or bands of frequencies in the motions being blended [BW95, UAT95, MLD10]. It was the intriguing character of frequency domain methods with their potential for targeted motion control which prompted the *general* direction for this thesis, although the sparsity of past papers did suggest avenues for further work may perhaps be lacking. The *specific* motivations underlying Chapters 5, 6 and 7 are reiterated in Sections 8.5, 8.6 and 8.7, which, for each implementation project, summarise the research problems addressed, the contribution made and its degree of success.

8.4 Literature-based Practical Work

Before the *own* contributions set forth in Chapters 5, 6 and 7, programming work was conducted to explicitly emulate selected methods of *other* researchers, the purpose of which was to acquire and demonstrate a proper understanding of this previous work. Thus a multibody physics-intense simulation was built (Figure 3.2, Chapter 3), emulating the dynamic constraints modelling paper of Barzell and Barr [BB88]. An inverse kinematics-driven robot manipulator (Figure 2.4, Chapter 2) was also created, this time inspired by [Wel93] and [ESHD05]. Demonstration videos were provided of each, with download locations in literature review Chapters 2 and 3 and additionally in Appendix E. (Precursory implementations on motion capture usage as well as interpolation, and editing, in the frequency domain, based in part on Unuma et al. [UAT95] were also created, but videos are omitted as superseded by those of Chapters 5, 6 and 7).

The literature itself was described in Chapters 2, 3 and 4, with selected works from the extensive field of character animation treated in depth, to convey the intricacy which is fundamental to the individual contributions made. While seminal works were included, so were papers of lesser renown, which properly reflects the heterogeneous nature of the published body of literature.

8.5 The Streamlined Approach to DFT-based Blending

Chapter 5 presented the streamlined approach to frequency domain blending, the main contributions of which are listed below, after first, however, shedding light on the incentive underlying its multi-facetted structure, which markedly contrasts with the single focus of the following two chapters (6 and 7).

8.5.1 Motivation

In concurrent reading and practical work, the one-dimensional Fourier-domain blending of Unuma et al. [UAT95] was, as a natural progression, experimentally extended to 2D, with a fully-working triangle network prototype built, before the strikingly similar work of Pettré and Laumond [PL06] came to light. The concept of triangle-based Fourier-domain blending having previously been published, the method presented in Chapter 5 and in [MLD10] instead took the form of a *juxtaposition* with previous work, especially that of Pettré and Laumond [PL06]. The aim was thus to *compare* and to highlight the various advantages of the streamlined approach, especially in the chosen context of user-driven applications such as games.

8.5.2 Contribution

The key benefits of the streamlined approach are repeated below.

- Increased efficiency, notably during Fourier synthesis – the most expensive part of frequency domain blending – as was detailed in Chapter 5. The contribution, however, lay not in the cheaper formula thereby used, but in the proper treatment of the ambiguous issue of phase angle blending which alone allowed the formula to consistently generate correctly blended character poses. Previous work had either omitted this altogether [UAT95] or avoided blending phase angles by use of a more costly approach to Fourier synthesis [PL06].

- Triangle networks are used to guide the interpolation in the frequency domain. Those of Pettré and Laumond [PL06] had, in accordance with their motion-planning context, vertex locations strictly dictated by the linear and angular root velocities of the associated input motions. With its fundamentally different view of triangle networks, the streamlined approach was devoid of such constraints

and allowed, as demonstrated on video (Chapter 5, Appendix E), the blending of motions with negligible root velocities which the method of [PL06] could not. Network flexibility was thus increased providing the ability to blend between a greater range of input motions.

- Manual vertex placement additionally allows triangle networks to be seen as intuitive interface devices, well-suited to games or other virtual environments requiring user-friendly character control. Network layout can be modified and shaped to best fit the application in question, or to adjust input device sensitivity in various parts of the network.

Miscellaneous further benefits are included in Chapter 5, which furthermore details the key steps, and their established necessary sequence, for the implementation of the streamlined approach to frequency domain blending.

8.6 Hybrid Networks

Hybrid networks extend their blending-only predecessors. It is positive reviewer feedback apropos video demonstrations of the streamlined approach with its blending triangle networks which suggested the latter be further developed. Additionally, the limitations of interpolation-based synthesis *per se* hinted at an avenue for improvement. This limitation, and the intended improvement, are described in more detail below.

8.6.1 Motivation

The strength of motion blending, as in [PSS02, WH97, UAT95], as well as in the streamlined approach, is delicate motion control which is essential for *precise* navigation in a virtual environment. Merging two or more poses, or entire frame sequences in this way, can clearly fail, however, given dissimilar inputs. Thus with increasing input

motion disparity, blending quality tends to suffer with the generation of unwanted artefacts, and consequently, the demand for motion comparability tends to reduce diversity within blending networks.

To overcome this, *discrete* networks can be built enacting varying motion types, but these then require interconnecting to provide continuous synthesis upon changing motion styles. Such bridging can sometimes be performed by blending, thus transitioning from one motion type to another using time-varying blending weights, but even when possible it may inevitably look inferior to the simple playback of unimpaired motion capture acting out the required change – a feature not available in conventional blending.

In contrast to interpolation-based synthesis are methods which concatenate selected motion clips and play them back in sequence, as in the Motion Graphs paper of Kovar et al. [KGP02] and in others which followed [AF02, LCL06, LL06]. These exhibit, at least potentially, a greater variety of output, and can display any motion within an *existing* network, given the requisite source data. Furthermore, two disparate motions clips might indeed be successfully bridged by another. This is of course no panacea, as, for one thing, the output stems from motion clip selection whose discrete nature lacks the fine gradation and control found in motion interpolation. Nevertheless, while conditional on suitable input, required both to enact the desired movements *and* to provide the necessary frame compatibility for smooth concatenation, the play-back based motion underlying concatenation synthesis is less prone to the limitations of variety and motion-bridging difficulties found in interpolation-based methods.

It was to merge the benefits of motion play-back with the continuous control of blending, that the streamlined approach was extended with the creation of hybrid networks.

8.6.2 Contribution

Hybrid networks are novel structures introduced in Chapter 6, which enhanced the previous blending-only networks by the inclusion of 'transitions' – specific actions achieved by sequence playback. The main research contribution lay in the development of the structure, and seamless operation, of these transitions, in a manner which neatly exploited the mechanism underlying the existing Fourier blending networks.

Two forms of transition were presented, diverging in purpose and also somewhat in their functioning. The inter-network kind served to link networks of disparate motion types, while intra-network transitions enabled the integration, within a given network, of alien motion styles. Fundamental to either case was the idea of a single source data clip comprising a central played-back motion sequence encapsulated by cyclification-viable sections at each end. By masquerading as regular input motions these endpoints could be seamlessly embedded within the network, at dual-purpose transition nodes (replacing earlier vertices).

Continuity in transitions – the prime requirement and challenge – resulted from manifold factors. A cyclified endpoint could be smoothly given increased weighting in the blending network, as already possible for any input motion. Having reached sufficient weighting, thus effectively executing *single*-input blending, the motion was indistinguishable from motion playback, and with appropriate synchronisation could smoothly join the central sequence of played-back motion proper, before, eventually, returning to the blending network by a process of opposite sequence. It is this *absence of any clear demarcation between blending and playback* which lends elegance to the solution of network integration. Continuity was imperative between transition phases too, and this was guaranteed by their having originated from a single raw input sequence, and by the use of processing stages, including steps for motion synchronisa-

tion, specifically designed to maintain it.

Underlying complexities notwithstanding, such as multiple-motion nodes and overlapping triangles, the user-interface, now extended, remained intuitive as before, retaining all the fine control previously available. Its simplicity is illustrated by the metaphor of transitions being *escalators in the blending space*, requiring only to be stepped on. At most, user mode switching added trivial complexity, but without impeding the action as the demonstration video shows (Chapter 6 or Appendix E).

Hybrid networks fully met the objectives of greatly increasing potential network variety and utility, and of providing natural-looking transformations to bridge disparate networks. A detailed description was provided of the mechanism underlying the updated interface, with its seamless merging of two very different methods of motion synthesis.

8.7 Single-source Harmonic Switching

Frequency domain blending is an established field. Viewed dispassionately, however, the literature leaves one uneasy question not entirely answered: why blend in this manner at all? Why endure the runtime penalty of Fourier synthesis, when much can be done in the time domain quite simply avoiding this cost?

Some benefits are mentioned of course, like the filtering of noisy motion by the attenuation of harmonics [PL06], but filtering is possible before, or after, time-domain blending too [KG03]. Blending weights manually set for each frequency band, as well as motion editing or even its creation by the individual control of harmonics [BW95], may be impressive in concept, but are also *highly* unintuitive. Motion, unlike music, is not easy to refine by adjusting the levels of its frequency bands, and the rare user-friendly examples, such as a gait made to shiver by an injection of higher frequencies [UAT95], seem of limited use.

Admittedly, the benefits of processing character motion in the frequency domain, as revealed in the literature, were not overwhelming. A significant benefit, however, arose with the introduction of *single-source harmonic switching*, a process presented in Chapter 7 and motivated by the reasons below.

8.7.1 Motivation

The previous character animation implementations for this thesis showed the quest to improve the *quality* of motion to be the greatest and most time-consuming challenge by far. Compared to vehicle simulations (Figure 3.1, Chapter 3), aberrations in human locomotion synthesis tend strongly towards being more obvious, more frequent, and often singularly hard to redress. Despite encountering, and for the most-part counteracting, many problems of this kind, a perplexing distortion, exhibiting occasional waves of jerkiness, remained unaddressed. It is this which motivated the focus of Chapter 7, resulting in the attenuation, to a considerable degree, of this previously elusive distortion.

8.7.2 Contribution

Starting out with no more than a distortion of unknown cause, whose most specific description was 'jerky motion', made worse still by its intermittent nature, first required – as a precursory contribution – the clarification of its origin, before any attempts could be made at correction. Experiments found it only to manifest when blending in the upper frequency range of that spectrum necessary, as concurred by [UAT95, PL06], to yield highest quality motion. From this arose the term *upper-harmonic distortion* (UHD) and the name for its locus, the *UH band*. Consequent investigation showed the inception, and also the unpredictability of UHD, to be reflected in the spectral content of the captured walking sequences being used as sources for blending. UHD

was shown to be inherent in the very process of blending such data, be it in the time or the frequency domain, and hence to be ubiquitous to walk cycle interpolation itself, which, it is reasonable to assume, extends to other motion types too.

The problem specification imposed the use of all harmonics in the higher-quality range, which precluded the brute-force elimination of UHD by a jettison of the UH band. The presented solution indeed retained this band but dispensed with blending inside it. Harmonics in the synthesised output had *previously* always resulted from the blending of contributions from the corresponding harmonics of each input motion. The proposed method however, avoided the corruption of affected frequencies by instead *duplicating* the sole harmonic from just one of the inputs, creating thereby, in the frequency domain, a stack of *single-source harmonics* within the UH band (now equivalently known as the *SSH band*). SSH stack composition, designed to minimise incompatibilities with the conventionally blended harmonics below it, varied dynamically, and gradually, as the blending point moved within the network. *SSH switching*, the name for the process controlling the stack, was expounded in detail in Chapter 7.

Screenshots and the demonstration video showed SSH switching to be highly effective, despite the untuned prototype employed. To quantify results a measurement process was additionally developed. It compared the actual synthesised output to a deemed-ideal reference motion. Statistics and plots from network-wide measurements were found to give good results too, especially at UHD hotspots. Moreover SSH-switching can be left permanently enabled while blending in the frequency domain, with the permanent benefit of *negative* cost thus *reducing* the impact at runtime.

The primary contribution of Chapter 7 lay in the development of the SSH switching mechanism, an effective and novel solution to a blending artefact which, while afflicting blending in both time and the frequency domain, can be corrected only in the latter, thereby offering a useful enhancement to the field itself.

8.8 Final Thoughts

The novel methods presented in this thesis, both supplement, and indeed augment the selected subfield of frequency domain interpolation methods for character animation. Especially SSH-switched hybrid networks seem worthy of expansion, with the interplay between blending and motion sequence playback extended in new ways, bringing animation closer to the ideal of simultaneously available subtle control and rich motion variety, while retaining a truly intuitive interface.

Regarding the expansive field of character animation itself, it is striking how after decades of research, the greatest contribution towards natural-looking motion lies in motion capture, supporting, as it does, animation methods in which motion *creation*, in the strictest sense, is not achieved at all, but only a refashioning of recorded data. So one thing, perhaps, should not be forgotten. The realism achieved today stems from *copying* nature. This simple fact suggests that while complex ways of manipulating existing data can certainly bear fruit, there might, in ways so far undiscovered, be reward to be found in new ways of understanding, and also of recording, the processes inherent in human motion itself.

Appendix A

The Discrete Fourier Transform and Fourier Synthesis

A.1 Introduction

The implementations of Chapters 5, 6 and 7 used the discrete Fourier transform (DFT) in preprocessing, and, after blending at runtime in the frequency domain, returned to the time domain by means of Fourier synthesis. This appendix provides a brief glimpse of the extensive theory associated with these processes, with the aim of providing some background to the formulae used in this thesis. Discussion is restricted to the processing of one-dimensional data series and functions, as applied to the undertaken practical work. It is intended as an introduction for the computer scientist, for whom matters Fourier may lie outside his field of expertise, these being more typically the purview of digital signal processing engineers, mathematicians and other specialists. The treatment is based mostly on [CDH00, Med00, Coo03, Sha95].

A.2 Fourier Analysis and the Fourier Series

Fourier analysis is the process of separating a waveform into its constituent sinusoids of various frequencies plus a single steady-state 'DC' value, the summation of which

yields the original function by a process known as Fourier synthesis. This allows, in fields such as signal processing and physics, for the response to waveforms applied to linear systems, to be established by the simpler consideration of the effect of the system on their component sinusoids.

It is evident that the summation of sinusoids of the form $A\sin(2\pi ft+\phi)$ will, whatever their amplitude (A) or phase angle (ϕ), build a *periodic* waveform if the frequency f of one sinusoid, known as the fundamental, is f_1, while those of the remaining sinusoids are integer multiples thereof. The constructed waveform, a function of time, will be of frequency f_1, and this even if the fundamental is missing from the set being coalesced. It thus comes as no surprise that (most) periodic waveforms can be broken down into constituent sinusoids of frequency $nf_1, n \in \mathbb{Z}^+$, known as 'harmonics', as well as a DC component providing any necessary vertical offset needed to recreate the original waveform.

The decomposition of the original function, $f(t)$, known as its Fourier series, is given by

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \big(a_n \cos(\omega nt) + b_n \sin(\omega nt)\big) \tag{A.1}$$

where $\omega = 2\pi f$ is the angular frequency. Alternatively expression in terms of the period, T, gives

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos\frac{2\pi nt}{T} + b_n \sin\frac{2\pi nt}{T}\right) \tag{A.2}$$

The required constant values, a_0, a_n and b_n, known as Fourier coefficients, are specific to the waveform being decomposed and thus define it. These are given by

$$a_0 = \frac{2}{T}\int_0^T f(t)\,dt \tag{A.3}$$

$$a_n = \frac{2}{T}\int_0^T f(t)\cos\frac{2\pi nt}{T}\,dt \qquad n \in \mathbb{Z}^+ \tag{A.4}$$

$$b_n = \frac{2}{T}\int_0^T f(t)\sin\frac{2\pi nt}{T}\,dt \qquad n \in \mathbb{Z}^+ \tag{A.5}$$

whereby the limits of integration can be any convenient complete period. As indicated above, not all periodic function can be expanded into a Fourier series. The requirements, known as the Dirichlet conditions, are specified differently from source to source, being given as follows in [CDH00].

i) $\int |f(t)|\,dt$ to be finite over a complete period

ii) $f(t)$ to have a finite number of discontinuities, at most, in any finite interval

The general equations A.1 or A.2 may, at first sight, appear to indicate infinite series, but this is not necessarily so as Fourier coefficients can be found to have zero-magnitude. Thus, for example, while a square wave comprises infinite odd-numbered harmonics, the trivial case of $f(t) = \sin(t)$ is clearly finite, which, furthermore, if processed using the above equations, does indeed, as of course expected, yield only one non-zero Fourier coefficient – that of the fundamental.

Discussion of harmonics in this thesis has made repeated mention of phase angles. Fourier series harmonics expressed with potentially both sine and cosine terms specify the phase angle indirectly by the coefficients a_n and b_n. Conversion, if desired, to a single sinusoid of choice – sine or cosine – and an *explicitly* stated phase angle is straightforward, with the formula for either case shown below.

$$R_n \sin(\omega nt + \phi_n) = \sqrt{{a_n}^2 + {b_n}^2} \sin\left(\omega nt + \arctan\frac{a_n}{b_n}\right) \quad (A.6)$$

$$R_n \cos(\omega nt + \phi_n) = \sqrt{{a_n}^2 + {b_n}^2} \cos\left(\omega nt + \arctan\frac{-b_n}{a_n}\right) \quad (A.7)$$

where R_n and ϕ_n are the magnitude and phase angle of the n^{th} harmonic, and a_n and b_n the corresponding Fourier coefficients.

A.3 Fourier Series Complex Notation

The Fourier series is also found written in complex notation, a form from which can be derived the continuous Fourier transform, and in turn the discrete Fourier transform used for implementation work in this thesis. The complex series is given by

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi nt/T} \quad (A.8)$$

where the Fourier coefficients, including the DC term $c_0 = a_0/2$, are given by

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t) e^{-j2\pi nt/T}\, dt \quad (A.9)$$

As before, any convenient complete period of $f(t)$ can be chosen over which to evaluate the integral.

Complex notation arises from the substitution into formulae A.1 and A.2 of the right-hand side of $\cos\theta = \frac{e^{j\theta}+e^{-j\theta}}{2}$ and $\sin\theta = \frac{e^{j\theta}-e^{-j\theta}}{2j}$, equations themselves resulting from Euler's formula $e^{\pm j\theta} = \cos\theta \pm j\sin\theta$.

A.4 The Fourier Transform

As mentioned in earlier chapters, the Fourier domain practical work carried out for this thesis always made use of the DFT, which might have suggested that some guise of Fourier *transform* is always needed to break a waveform down into component frequencies. The previous two sections refute this assumption, and furthermore, obtaining Fourier series coefficients algorithmically employing numerical integration is a realistic alternative to using the DFT.

The Fourier transform, also called the *continuous* Fourier transform to distinguish it from the DFT, is a generalisation of the complex Fourier series which, unlike the series, is not limited to periodic functions. While defined alternatively as a real-numbered or complex expression, the latter is most commonly used, and shown below.

$$\mathcal{F}\{f(t)\} \quad = \quad F(\omega) \quad = \quad \int_{-\infty}^{\infty} f(t)e^{-j\omega t}\,dt \tag{A.10}$$

The Fourier transform is thus seen to be an integral, and no longer a sum of discrete components as pertained with the Fourier series. Furthermore, applying the transform converts a function of time to one of frequency (since $\omega = 2\pi f$). Equation A.10 is one of several existing variations in the definition, and the inverse Fourier transform, below, is one selected to be compatible such that Equations A.10 and A.11 form a *Fourier transform pair*.

$$\mathcal{F}^{-1}\{F(\omega)\} \quad = \quad f(t) \quad = \quad \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}\,d\omega \tag{A.11}$$

As before, the above transform and its inverse are dependent on prerequisite conditions being met, of which those stated by [CDH00] now follow.

i) $\int_{-\infty}^{\infty} |f(t)|\, dt$ exists

ii) $f(t)$ and $f'(t)$ must be piecewise continuous in every finite interval

As seen above, $f(t)$ has both a Fourier transform $F(\omega)$ and, distinct from this, the integral representation $\frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}\, d\omega$ which includes it. It is the transform itself, a complex function of angular frequency, which reveals the frequency domain content of $f(t)$, its modulus $|F(\omega)|$ providing the amplitude, and the argument $\arg(F(\omega))$ giving the phase angle.

The corresponding plots against frequency, known as magnitude and phase spectra, comprise a range of continuous frequencies for non-periodic functions. In comparison, periodic functions exhibit discrete spectra, with non-zero values only at specific frequencies, thus reflecting findings from the Fourier series (Equations A.1 and A.2).

A.5 The Discrete Fourier Transform

The finite or discrete Fourier transform (DFT), derived from the continuous Fourier transform, $F(\omega)$, converts *samples* of $f(t)$, taken at regular intervals, to a complex *sequence* describing an approximation of $F(\omega)$. Again, sources vary, and the factor multiplying the DFT and its inverse (seen below to be 1 and $1/N$) as well as the signs of the exponents are merely one of several conventions. The DFT formula used for thesis implementations, as shown in Chapter 5, Section 5.4.7, is based on [CDH00] and [Med00] and takes the form

$$\mathcal{D}\{f[n]\} \;=\; F[k] \;=\; \sum_{n=0}^{N-1} f[n]e^{-j2\pi nk/N} \qquad \text{for } k = 0,\ 1,\ 2,\ \ldots\ N-1 \qquad \text{(A.12)}$$

The process thus generates a complex sequence $F[k]$ of N values, from an input sequence $f[n]$ of the same length. The inverse discrete Fourier transform (IDFT) recreates the original sequence from $F[k]$, and a version matched to the DFT above, is

$$\mathcal{D}^{-1}\{F[k]\} \;=\; f[n] \;=\; \frac{1}{N}\sum_{k=0}^{N-1} F[k]e^{j2\pi nk/N} \quad \text{for } n = 0,\ 1,\ 2,\ \ldots N-1 \quad \text{(A.13)}$$

$F[k]$ represents frequency domain data, whose amplitude and phase is given, as before, by the complex modulus and argument. The frequency associated with each discrete value of k is $\frac{kf_s}{N}$ where f_s is the sampling frequency. The DFT thus ranges from 0 (DC) for the first element of $F[k]$ to $\frac{(N-1)f_s}{N}$ for the last.

To approximate the Fourier transform using the DFT it is necessary to multiply the obtained complex values of $F[k]$ by the sampling period T_s, though definitions of the DFT, as, for example, that of Equation A.12, usually do not include this. Furthermore, the DFT is periodic with period $\frac{2\pi}{T_s}$ and for real-valued input sequence, as applies when using motion capture data, its spectra exhibit symmetry about the Nyquist frequency $\frac{f_s}{2}$ (the highest frequency that can be represented without corruption – known as aliasing – when sampling at frequency f_s). The right-hand half of the DFT output is thus redundant with corresponding magnitude and phase spectra deviating strongly from those of $F(\omega)$ in that part of the spectrum. The input sequence can thus be rebuilt from the left hand half of the DFT output, as illustrated in the following section.

A.6 Post-DFT Fourier Synthesis

Fourier synthesis provides an alternative to the IDFT[1] (Equation A.13) for the building of a sequence $f[n]$ from its discrete Fourier transform $F[k]$, and was chosen for the implemented projects as deemed the more intuitive of the two. This section discusses the relationship between these alternative methods, and also considers the order in

[1]This thesis uses the terms 'Fourier synthesis' and 'the IDFT' to differentiate between the two. They are, however, essentially the same, as shown in Section A.6.1, and confusingly, the IDFT could itself be described as Fourier synthesis.

which harmonics should be accumulated when progressively constructing a waveform, and thereby further clarifies, and justifies, Equation 5.10, the synthesis formula given in Section 5.5.2 of Chapter 5. Although the demonstrated practical work used Fourier synthesis to create *new* motion from a dynamically changing blend of Fourier coefficients obtained from three input motions, the discussion below is limited to the synthesis process itself, for which it is sufficient to consider the rebuilding of a single data sequence after its decomposition via the DFT.

A.6.1 Fourier Synthesis Versus the IDFT

While Fourier synthesis was favoured over the IDFT for the implemented work, the two are closely related, simply working with data in different formats. The complex values $F[k]$ resulting from the decomposition of a waveform via the DFT can be represented either as $real_k$ and $j\,imag_k$ the real and imaginary components respectively[1], or they can be expressed as a magnitude and a phase angle, m_k and p_k, the equivalence being illustrated in Figure A.1. The IDFT recreates the original waveform directly from these complex values, while Fourier synthesis uses magnitude and phase data. In either case an approximation results if using only a subset of the available harmonics. The relationship between both methods is now demonstrated below.

The IDFT of Equation A.13 compares to Fourier synthesis in its basic form, without the modifications seen in Equation 5.10, the synthesis formula of Chapter 5. These modifications are discussed further below, but a straightforward formula is initially considered, which fully rebuilds a waveform using all its constituent harmonics, and is given by

[1]Standard notations for the real and imaginary parts include $z = x + iy$ as well as $x = \mathrm{Re}\{z\}$ with $y = \mathrm{Im}\{z\}$. Magnitude and phase may be written $m = |z|$ and $\phi = \arg(z)$.

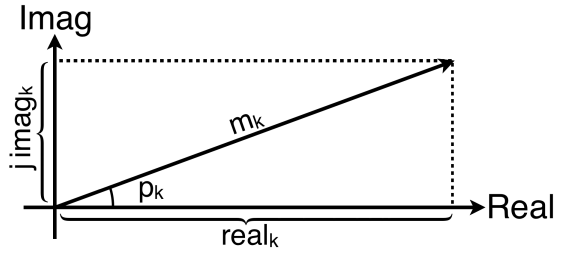

Figure A.1: Argand diagram showing how the k^{th} complex value from the DFT can be expressed either in terms of real and imaginary components, $real_k$ and $imag_k$ respectively, or as a magnitude and a phase angle, m_k and p_k.

$$R_n = \frac{m_0}{N}\cos p_0 + \frac{1}{N}\sum_{k=1}^{N-1} m_k \cos\left(\frac{2\pi kn}{N} + p_k\right) \tag{A.14}$$

or, more compactly,

$$R_n = \frac{1}{N}\sum_{k=0}^{N-1} m_k \cos\left(\frac{2\pi kn}{N} + p_k\right) \tag{A.15}$$

where N is the length of the data sequence, whether in the time or the frequency domain, making $N-1$ the highest harmonic used. m, p and R_n hold the k^{th} Fourier magnitude and phase angle coefficients, and the n^{th} time domain output value respectively, where n varies from 0 to $N-1$.

The k^{th} harmonic of the n^{th} time domain output sample is given by

$$R_{nk} = \frac{1}{N} m_k \cos(\theta + p_k) \qquad \text{where } \theta = \frac{2\pi kn}{N} \tag{A.16}$$

Thus

$$R_{nk} = \frac{1}{N} m_k (\cos\theta \cos p_k - \sin\theta \sin p_k) \tag{A.17}$$

and, with reference to Figure A.1,

$$R_{nk} = \frac{1}{N}(real_k \cos\theta - imag_k \sin\theta) \tag{A.18}$$

where $imag_k$ is the magnitude of the imaginary part of the k^{th} DFT-generated value.

In comparison, turning now to the IDFT of Equation A.13, and again assuming θ as defined above, the k^{th} harmonic of the n^{th} output sample is given by

$$\frac{1}{N}F[k]e^{j\theta} = \frac{1}{N}(real_k + j\ imag_k)(\cos\theta + j\ \sin\theta) \tag{A.19}$$

$$= \frac{1}{N}\big((real_k \cos\theta - imag_k \sin\theta) + j\ (real_k \sin\theta + imag_k \cos\theta)\big) \tag{A.20}$$

which shows the real part of the IDFT output of Equation A.20 to be identical to the Fourier synthesis output of Equation A.18. The basic Fourier synthesis formulae of Equations A.14 and A.15 thus recreate the *real* part of the output of the IDFT, which as further explained in the following section, is all that is needed to fully rebuild a real-valued waveform as found in character animation, or simply to approximate it with a limited number of harmonics.

It can similarly be shown, by substituting sin for cos in Equations A.14 and A.15 that the following synthesis formulae

$$I_n = \frac{m_0}{N}\sin p_0 + \frac{1}{N}\sum_{k=1}^{N-1} m_k \sin\left(\frac{2\pi kn}{N} + p_k\right) \tag{A.21}$$

$$I_n = \frac{1}{N}\sum_{k=0}^{N-1} m_k \sin\left(\frac{2\pi kn}{N} + p_k\right) \tag{A.22}$$

yield the imaginary component of the IDFT output. Thus, unless intentionally restricted, Fourier synthesis generates the same complex output as the IDFT.

The synthesis formula employed[1] for all implemented work, was Equation 5.10 of Chapter 5, which is repeated below for convenience

$$R_n = \frac{m_0}{N}\cos p_0 + \frac{2}{N}\sum_{k=1}^{H} m_k \cos\left(\frac{2\pi k n}{\mathcal{N}} + p_k\right) \tag{A.23}$$

where N, m, p and R_n are defined as above for Equation A.15. $n \in [0, \mathcal{N}-1]$ now replaces $n \in [0, N-1]$, however, which allows the output sequence to be resampled to a different length $\mathcal{N} \neq N$. Furthermore, as explained in Section 5.7.6 of Chapter 5, the number of harmonics actually used is limited to $H \in [3, 8]$, instead of $N-1$ seen in Equation A.15. Another modification, the factor $\frac{2}{N}$, is discussed in the following section. All these changes, however, could equally be applied to the IDFT, and the enhanced synthesis formula of Equation A.23 (or 5.10) would then yield precisely the real component of the IDFT output.

A.6.2 Harmonic Summation Sequence

In the case of *continuous* data, and the synthesis of a sawtooth, square or triangular waveform, the accumulation of harmonics *in order of increasing frequency* yields a waveform whose quality improves in ever-decreasing steps. It might thus be assumed that the same applies for *sampled* data, whether summing harmonics by Fourier synthesis or by using the IDFT, with steady improvements of diminishing importance up to that point where all harmonics are included, and the original signal is rebuilt. That this is not so, however, can be taken from Figure A.2 top and centre, which illustrate a sawtooth waveform and its magnitude spectrum. The latter exhibits symmetry

[1] As explained in Chapter 5, to lower runtime cost in the implementation, the term $\frac{m_0}{N}\cos p_0$ was replaced by the mathematically equivalent input signal mean (or more precisely, by the blend of input sequence means – due to the previous three-input blending step).

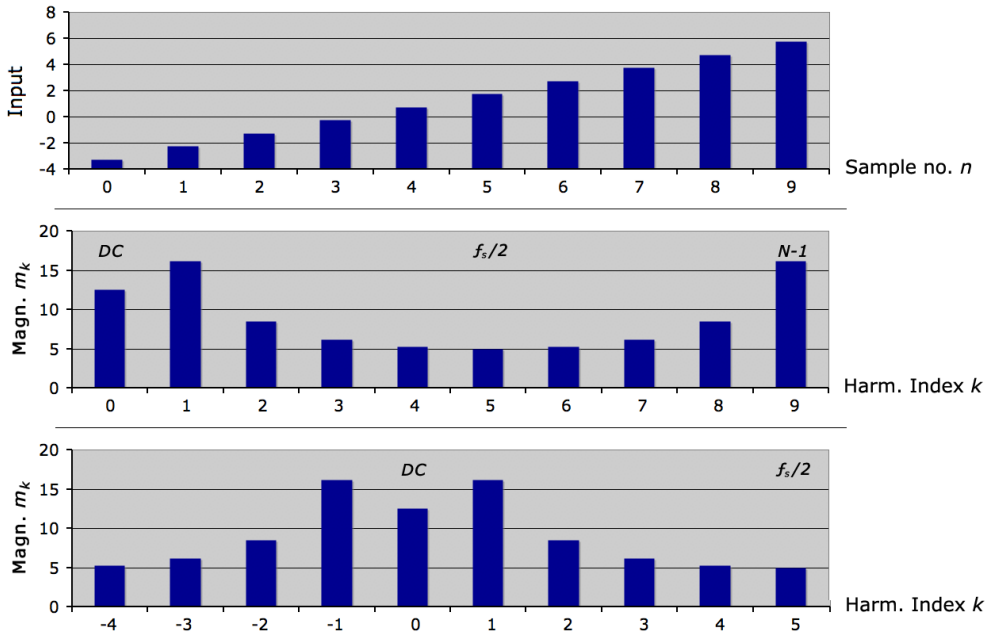

Figure A.2: Sawtooth waveform (top) and associated magnitude spectra, symmetrical about the Nyquist frequency, $f_s/2$, (centre), and about the frequency zero, (bottom).

about the Nyquist frequency $\frac{f_s}{2}$, revealing how a gradual summation of harmonics of increasing frequency, unlike the continuous case, would *not* yield an approximation whose refinement proceeded in ever-smaller increments. For example, in Figure A.2, centre, the ultimate harmonic has a relatively great, not a minor impact, suggesting the requirement for a different ordering in the summation of harmonics.

The periodicity of the DFT, however, makes the representation of Figure A.2, bottom, equally valid, in which what previously was considered the ultimate harmonic now exhibits negative frequency and index -1, and, with other harmonics similarly offset, yields spectrum symmetry about $f = 0$. (The N-sized index difference, as when switching allocation from $N - 1$ to -1 has no effect on IDFT output, nor does it on Fourier synthesis). The bottom spectrum of Figure A.2 indicates that in order to imitate the continuous case, the summation of harmonics should proceed with the zeroth

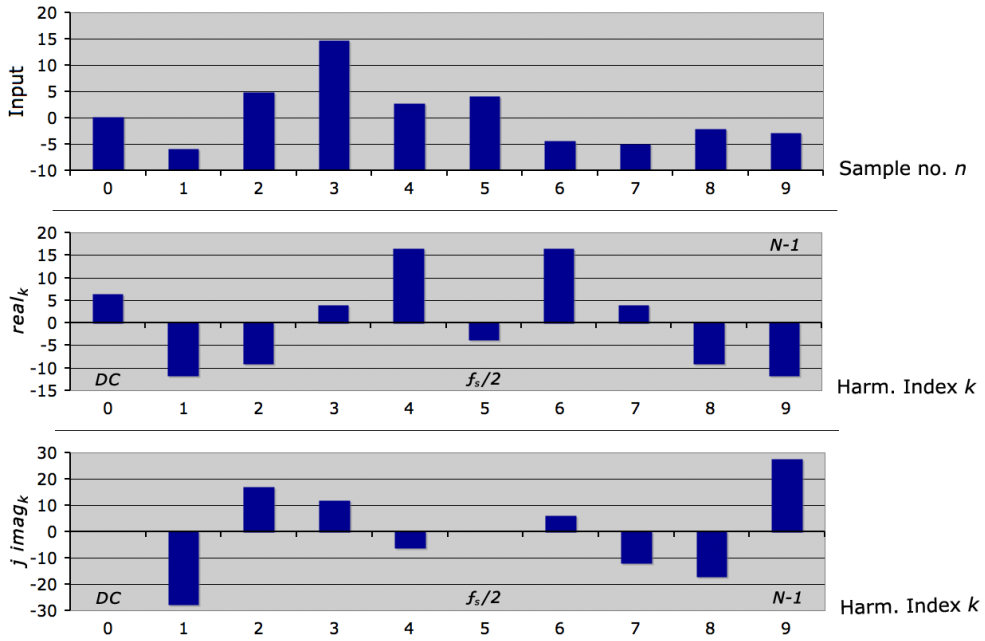

Figure A.3: Real-valued input sequence (top) and real and imaginary spectra (centre and bottom respectively) obtained via the DFT, showing corresponding harmonics, those of index k and $(N-k)$, to be complex conjugates.

first, and then pairwise indices {1, -1}, {2, -2}, etc, or for the spectrum of Figure A.2, centre, with indices in the order 0, $\{1, N-1\}$, $\{2, N-2\}$, and so forth – the accepted ordering for waveform construction. For any real-valued function, such as those occurring in character animation, harmonics $F[k]$ paired in this way are complex conjugates, as seen in the spectra of $real_k$ and $j\,imag_k$ illustrated in Figure A.3. This leads to the cancellation of the imaginary component of the IDFT output when using such pairs of harmonics. It is for this reason that Fourier synthesis in character animation need only generate the real part of the IDFT output.

The same output is obtained at half the cost, however, using Equation A.23 above which duplicates that given in Chapter 5, and uses just one from each pair of harmonics, while compensating for this by doubling the generated real-valued output. This underlies the factor $\frac{2}{N}$ instead of the $\frac{1}{N}$ seen in standard formulae like Equation A.13. Thus,

despite summing harmonics in plain index order, from 0 to H (where $H \in [3, 8]$), the proper *pairwise* addition is *simulated*, yielding output of correct magnitude, notwithstanding the discarded harmonics.

A.7 The Fast Fourier Transform

The DFT was quite sufficient for preprocessing relatively short sequences in the implementations of Chapters 5, 6 and 7, so the fast Fourier transform was simply not required. It is, nevertheless, touched upon here for the sake of completeness.

The DFT, being an $\mathcal{O}(n^2)$ process, is prohibitively expensive for longer sequences, especially for runtime use. This lead to the development of the fast Fourier transform (FFT) and its inverse (the IFFT or FIFT), a class of much faster $\mathcal{O}(n \log(n))$ algorithms, generating the same data sequences as do the DFT and the IDFT.

Seminal work in this area is the 1965 paper of Cooley and Turkey [CT65] which helped drive a rapid expansion in the field of digital signal processing, though the algorithm can be traced back to Gauss in 1805 [DV90, Bur08]. Numerous versions followed offering improvements over predecessors in terms of computational expense, ease of implementation, or practical limitations of the algorithm (such as the requirement for input data to be of length 2^n, achieved, if necessary, by padding with zeroes, or the necessity for input data to be real-valued) [DV90].

Description of these algorithms is a lengthy topic beyond the scope of these few pages, but FFTs are, in general, based on exploiting the symmetries, periodicity and orthogonality of the basis functions of the DFT [Bur08]. The relationship between the DFT and convolution has also been exploited. All FFT algorithms use a divide and conquer approach [DV90], which in the case of the Cooley-Turkey algorithm (and that of Gauss) depends on recursively breaking down a DFT of length $N = N_1 N_2$ into

smaller DFTs of size N_1 and N_2. A concise description of the Cooley-Turkey FFT algorithm is found in [CLW69, PTVF92, Coo03].

A.8 Conclusion

This appendix has provided, for the computer scientist with no specialist knowledge of Fourier mathematics, some background and key formulae relevant to the time-domain frequency-domain duality made use of in the practical work of this thesis. It extends beyond what is needed for implementation purposes by providing a basic understanding of related material, but should not be seen as anything more than a mere snapshot of a very substantial subject.

Appendix B

Phase Angle Blending – Derivation of Presented Approach

B.1 Introduction

In Chapter 5, Section 5.3, it was demonstrated how for input motions represented by Fourier series expansions of a form which uses an explicitly specified phase angle for each harmonic, the phase angle coefficients of corresponding harmonics from each motion cannot be blended in a naive manner, due to ambiguities inherent in this approach. For reasons of efficiency it is this form of expansion which was selected during implementation work, requiring an answer to the problem of phase angle blending. In the presented solution, the correctly blended phase angle $p_{k,\,blended}$ for the k^{th} harmonic was said to be

$$p_{k,blended} \quad = \quad \arctan2\big((m_k \sin p_k)_{blended}, (m_k \cos p_k)_{blended}\big) \tag{B.1}$$

where for three-input blending as used in triangle networks

$$(m_k \sin p_k)_{blended} = w_1(m_{k_1} \sin p_{k_1}) + w_2(m_{k_2} \sin p_{k_2}) + w_3(m_{k_3} \sin p_{k_3}) \quad \text{(B.2)}$$

$$(m_k \cos p_k)_{blended} = w_1(m_{k_1} \cos p_{k_1}) + w_2(m_{k_2} \cos p_{k_2}) + w_3(m_{k_3} \cos p_{k_3}) \quad \text{(B.3)}$$

and where w_1, m_{k_1} and p_{k_1} are the weighting for motion 1 and the magnitude and phase angle of its k^{th} harmonic, with similar definitions applying for input motions 2 and 3. A derivation of the above can be found in Mardia and Jupp [MJ00], albeit in the context of directional statistics. This appendix provides alternative derivations, one analytical and one geometrical.

B.2 Analytical Derivation

The Fourier synthesis formula used in Chapter 5 and given in Section 5.5.2 is repeated below[1]

$$R_n = \frac{m_0}{N} \cos p_0 + \frac{2}{N} \sum_{k=1}^{H} m_k \cos\left(\frac{2\pi k n}{\mathcal{N}} + p_k\right) \quad \text{(B.4)}$$

whereby N is the length of the blended Fourier coefficient sequences, $\mathcal{N}$ is the desired output sequence length and H is the highest harmonic used (indexing such that 1 is the fundamental). m, p and R_n hold the k^{th} blended Fourier magnitude and phase angle coefficients, and the n^{th} time domain output value respectively, where n varies from 0 to $\mathcal{N} - 1$.

Magnitudes m_k can be blended in a straightforward manner, but phase angles p_k cannot. Fourier coefficients of this type (m_k, p_k) were used in Chapters 5, 6 and 7 to

[1] As stated in Chapter 5, to lower cost in the implementation, the term $\frac{m_0}{N} \cos p_0$, shown here for completeness, was equivalently replaced by the blend of the input sequence means.

represent not only blended motion, but the inputs it depends upon too. As seen in Equation B.4, in this representation, the k^{th} harmonic h_k takes the form

$$h_k = \frac{2m_k}{N}\cos\left(\frac{2\pi kn}{\mathcal{N}} + p_k\right) \tag{B.5}$$

Now since

$$R\cos(\theta+\phi) = R(\cos\theta\cos\phi - \sin\theta\sin\phi) \tag{B.6}$$

$$= (R\cos\phi)\cos\theta + (-R\sin\phi)\sin\theta \tag{B.7}$$

the k^{th} harmonic of input motions 1, 2 and 3 can be written as

$$h_{k_1} = \left(\frac{2m_{k_1}}{N}\cos p_{k_1}\right)\cos\frac{2\pi kn}{\mathcal{N}} + \left(\frac{-2m_{k_1}}{N}\sin p_{k_1}\right)\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.8}$$

$$h_{k_2} = \left(\frac{2m_{k_2}}{N}\cos p_{k_2}\right)\cos\frac{2\pi kn}{\mathcal{N}} + \left(\frac{-2m_{k_2}}{N}\sin p_{k_2}\right)\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.9}$$

$$h_{k_3} = \left(\frac{2m_{k_3}}{N}\cos p_{k_3}\right)\cos\frac{2\pi kn}{\mathcal{N}} + \left(\frac{-2m_{k_3}}{N}\sin p_{k_3}\right)\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.10}$$

where m_{k_1} and p_{k_1} are the magnitude and phase of the k^{th} harmonics of input motion 1, with similar notation applying to the other input motions. Equations B.8 to B.10 can be written more succinctly as

$$h_{k_1} = A_{k_1}\cos\frac{2\pi kn}{\mathcal{N}} + B_{k_1}\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.11}$$

$$h_{k_2} = A_{k_2}\cos\frac{2\pi kn}{\mathcal{N}} + B_{k_2}\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.12}$$

$$h_{k_3} = A_{k_3}\cos\frac{2\pi kn}{\mathcal{N}} + B_{k_3}\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.13}$$

where for harmonic k input motion 1,

$$A_{k_1} = \frac{2m_{k_1}}{N}\cos p_{k_1} \tag{B.14}$$

$$B_{k_1} = \frac{-2m_{k_1}}{N}\sin p_{k_1} \tag{B.15}$$

and A_{k_2}, B_{k_2}, A_{k_3} and B_{k_3} are similarly defined for the other motions.

The right hand side of Equations B.11 to B.13 are now of the same general form as the expression for harmonics in the more expensive Fourier synthesis formula used by Pettré and Laumond [PL06] shown in Chapter 5, Section 5.3, which is duplicated below for convenience.

$$m(t) = \frac{\alpha_0}{2} + \sum_{k=1}^{N} \alpha_k \cos\left(\frac{k\pi t}{T}\right) + \beta_k \sin\left(\frac{k\pi t}{T}\right) \tag{B.16}$$

The formula itself is not considered in detail here as only the general manner in which harmonics are represented is relevant, for which it suffices to specify that α_0, α_k and β_k are Fourier magnitude coefficients.

As none of the coefficients in Equations B.11 to B.13 (or B.16) are phase angles, the harmonics they describe can be blended in a straightforward manner without ill-effect. The result $h_{k_{bl}}$ of blending the k^{th} harmonic of input motions 1, 2 and 3 is thus given by the following representation

$$h_{k_{bl}} = \frac{2m_{k_{bl}}}{N}\cos\left(\frac{2\pi kn}{\mathcal{N}} + p_{k_{bl}}\right) = A_{k_{bl}}\cos\frac{2\pi kn}{\mathcal{N}} + B_{k_{bl}}\sin\frac{2\pi kn}{\mathcal{N}} \tag{B.17}$$

$$A_{k_{bl}} = w_1 A_{k_1} + w_2 A_{k_2} + w_3 A_{k_3} \tag{B.18}$$

$$B_{k_{bl}} = w_1 B_{k_1} + w_2 B_{k_2} + w_3 B_{k_3} \tag{B.19}$$

where w_1, w_2 and w_3 are the weightings used for input motions 1, 2 and 3 respectively. But with reference to Equations B.14 and B.15 $A_{k_{bl}}$ and $B_{k_{bl}}$ can be expanded to

$$A_{k_{bl}} = \frac{2}{N}\big(w_1(m_{k_1}\cos p_{k_1}) + w_2(m_{k_2}\cos p_{k_2}) + w_3(m_{k_3}\cos p_{k_3})\big) \quad \text{(B.20)}$$

$$B_{k_{bl}} = \frac{-2}{N}\big(w_1(m_{k_1}\sin p_{k_1}) + w_2(m_{k_2}\sin p_{k_2}) + w_3(m_{k_3}\sin p_{k_3})\big) \quad \text{(B.21)}$$

Furthermore, as Equations B.7 can be written

$$R\cos(\theta+\phi) = a\cos\theta + b\sin\theta \quad \text{(B.22)}$$

$$a = R\cos\phi \quad \text{(B.23)}$$

$$b = -R\sin\phi \quad \text{(B.24)}$$

the phase angle ϕ can be expressed in terms of the factors preceding the cosine and sine functions

$$\phi = \arctan2(-b, a) \quad \text{(B.25)}$$

where the two-argument quadrant-aware arctan2 function is required to generate phase angles in the range - $\pi < \phi \leq \pi$.

Applying the same principle to Equation B.17 gives

$$p_{k,\,blended} = \arctan2(-B_{k_{bl}}, A_{k_{bl}}) \quad \text{(B.26)}$$

and in turn the expression for correctly blended phase angles which this appendix set out to prove

$$p_{k,\,blended} = \arctan2\big((m_k\,\sin p_k)_{blended}, (m_k\,\cos p_k)_{blended}\big) \quad \text{(B.27)}$$

where, as mentioned above, for three-input blending

$$(m_k \sin p_k)_{blended} = w_1(m_{k_1} \sin p_{k_1}) + w_2(m_{k_2} \sin p_{k_2}) + w_3(m_{k_3} \sin p_{k_3}) \quad \text{(B.28)}$$

$$(m_k \cos p_k)_{blended} = w_1(m_{k_1} \cos p_{k_1}) + w_2(m_{k_2} \cos p_{k_2}) + w_3(m_{k_3} \cos p_{k_3}) \quad \text{(B.29)}$$

with variables as previously defined.

B.3 Geometrical Derivation

The same result can be obtained with a geometrical analysis, by means of a phasor diagram as used, for example, by electrical and electronics engineers [WP85].

Since frequency domain blending interpolates between the *corresponding* harmonics of each input motion, blending is performed between sinusoids of identical frequency[1] which vary only in terms of amplitude and phase angle. This allows the k^{th} harmonic from each input motion, h_{k_1}, h_{k_2} and h_{k_3} to be represented as a phasor (phase vector) $\overrightarrow{h_{k_1}}$, $\overrightarrow{h_{k_2}}$ and $\overrightarrow{h_{k_3}}$ as shown by the grey-coloured vectors in Figure B.1. Phasor length in this diagram indicates harmonic amplitude corresponding to $\frac{2m_k}{N}$ in Equation B.5 whereby we refrain from using RMS values – typically used in such diagrams – as not required in this context. The angle ϕ_k between a phasor and the positive horizontal axis is the phase angle p_k of Equation B.5 for the harmonic in question – a statement valid when time t, or animation frame index n, are zero, which is all that needs to be considered here[2].

[1] In practical implementations frequencies of corresponding harmonics may differ somewhat due to belonging to input motion cycles of potentially different duration, with an associated theoretical reduction in the quality of blending otherwise possible with the formulae derived in this appendix.

[2] An alternative view, valid at any time t, is to consider phasors to be stationary on the diagram, with ϕ_k denoting the phase angle relative to an actually depicted, or merely implied, reference phasor which has zero phase angle and lies on the positive horizontal axis.

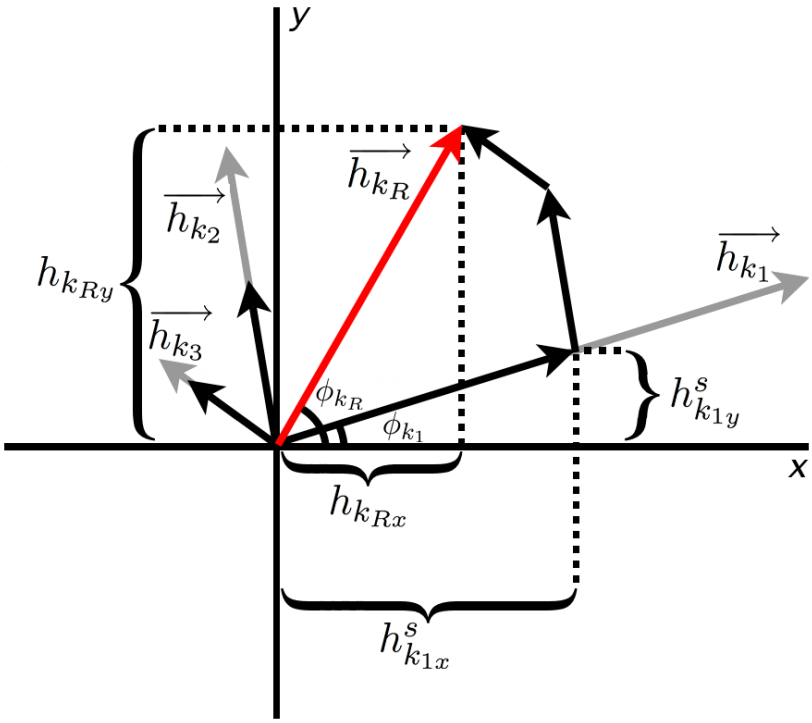

Figure B.1: Phasor diagram for the geometrical derivation of the formula for the phase angle of a blend $\overrightarrow{h_{k_R}}$ of three input motion harmonics $\overrightarrow{h_{k_1}}$, $\overrightarrow{h_{k_2}}$ and $\overrightarrow{h_{k_3}}$ all assumed to be of identical frequency. The corresponding harmonic amplitudes (vector lengths) are $m_{k_{blended}}$ for the resultant and m_{k_1}, m_{k_2} and m_{k_3} for the inputs.

Blending involves a sum of weighted sinusoids. Weighted harmonics are shown in black as phasors with scaled magnitudes, which are then added by vector addition to give the resulting blend $\overrightarrow{h_{k_R}}$ (red).

$h_{k_{Rx}}$ and $h_{k_{Ry}}$, the x and y components (scalars) of $\overrightarrow{h_{k_R}}$, are the weighted sum of the x and y components respectively of the input motion harmonics, thus

$$h_{k_{Rx}} = h^s_{k_{1x}} + h^s_{k_{2x}} + h^s_{k_{3x}} \tag{B.30}$$

$$h_{k_{Ry}} = h^s_{k_{1y}} + h^s_{k_{2y}} + h^s_{k_{3y}} \tag{B.31}$$

where $h^s_{k_{1x}}$ and $h^s_{k_{1y}}$ are the x and y components of the weighted (scaled) phasor for

harmonic k of input motion 1, with similar expressions applying to input motions 2 and 3.

Expressing in terms of weightings, amplitudes and phase angles gives

$$h_{k_{Rx}} = w_1 m_{k_1} \cos p_{k_1} + w_2 m_{k_2} \cos p_{k_2} + w_3 m_{k_3} \cos p_{k_3} \tag{B.32}$$

$$h_{k_{Ry}} = w_1 m_{k_1} \sin p_{k_1} + w_2 m_{k_2} \sin p_{k_2} + w_3 m_{k_3} \sin p_{k_3} \tag{B.33}$$

where, as previously, w_1, m_{k_1} and p_{k_1} are the weighting for motion 1 and the magnitude and phase angle of its k^{th} harmonic, with similar definitions for the remaining input motions.

From Figure B.1 angle ϕ_{k_R} and thus the correctly blended phase angle $p_{k_{blended}}$ is given by

$$p_{k_{blended}} = \arctan2\left(h_{k_{Ry}}, h_{k_{Rx}}\right) \tag{B.34}$$

for which $h_{k_{Rx}}$ and $h_{k_{Ry}}$ are already defined. This is the same result as was given in Equation B.27 when using the analytical approach.

Although this completes the intended derivation, it is worth noting that Figure B.1 indicates strictly correct amplitude blending for sinusoids of equal frequency to require the following formula

$$m_{k_{blended}} = \sqrt{\left(h_{k_{Rx}}^2 + h_{k_{Ry}}^2\right)} \tag{B.35}$$

with, again, $h_{k_{Rx}}$ and $h_{k_{Ry}}$ as already defined.

However, naive blending of amplitudes m_{k_1}, m_{k_2} and m_{k_3} does not impact animation in as detrimental a manner as does the simplistic blending of phase angles, with its inherent ambiguity explained in Chapter 5 Section 5.3

B.4 Conclusion

The formula required for the correct blending of harmonics defined by coefficients which include a phase angle has been derived, thereby also showing naive blending to be inappropriate. As magnitude and phase angle coefficients are obtained in preprocessing, evaluation of the six products $m_{k_1} \sin p_{k_1}$, $m_{k_1} \cos p_{k_1}$, $m_{k_2} \sin p_{k_2}$, $m_{k_2} \cos p_{k_2}$, $m_{k_3} \sin p_{k_3}$ and $m_{k_3} \cos p_{k_3}$ imposes no runtime cost.

Appendix C

Algorithmic Context

C.1 Introduction

This appendix provides an idea of the structure of the overall character animation program, allowing the contributions of Chapters 5, 6 and 7 to be seen positioned within the algorithm as a whole, as well as relative to each other. Furthermore, the DFT, frequency domain blending and Fourier synthesis formulae of Chapter 5, (Equations 5.8, 5.2 and 5.10 respectively), are shown in the context of immediately surrounding pseudocode, thereby further clarifying their function in the overall system.

C.2 Integrated Representation

Figure C.1 depicts a limited overview of the SSH switched system of Chapter 7 (subject to caveat, see Section C.4). Components specific to SSH switching are shown in red, those adopted from hybrid networks (Chapter 6) in blue, and those originally found in the streamlined approach (Chapter 5) are presented in black.

C.3 Program Structure

The program was written in C/C++ in conjunction with GLUT [Kil96] and OpenGL [SWND05].

In Figure C.1, the `init` function runs once at start-up, performing some of the system initialisation. This includes the preprocessing steps of the streamlined approach (Chapter 5, Section 5.4), of which one, the DFT, is further detailed in Figure C.2. The `init` function also performs various aspects of hybrid network initialisation (Chapter 6), as well as the specification of SSH switching parameter values (Chapter 7, Sections 7.5.2 to 7.5.4).

The `passiveMotion` function calculates the triangle weightings, and is called by GLUT when the pointer moves within the animation window[1], as occurs part of the time during user-guided navigation.

The most recent weighting values are used by the `blendingAndFS` function to perform blending and Fourier synthesis. Both of these operations are further detailed as pseudocode in Figure C.2 and share a dedicated thread, which (in the depicted operating mode) endlessly computes the output motion cycle in accordance with the latest user input. As shown in Figure C.1, (and mentioned in Section 6.5.3 of Chapter 6) this thread is killed when a transition is triggered, and recreated when it terminates.

The `move` function is the idle function, which is continuously called in the absence of window system events [Kil96], though as mentioned in Figures C.1 and C.2, it is *effectively* called repeatedly without interruption (calling does not simply stop during ongoing pointer movement as might perhaps be assumed). The `move` function updates the frame counter (Chapter 5, Section 5.5.3) which keeps track of the current output sequence frame during animation. It also uses either the cycle computed in the blending and synthesis thread, or, during transitions, motion data resulting from the hybrid network mechanism, to build the skeletal pose before calling for the current frame to be rendered.

[1]This applies as long as no mouse button is pressed [Kil96].

C.4 Caveat

It must be stressed that the system functionality described in this appendix, including that shown in Figures C.1 and C.2, corresponds to a very limited subset of the overall structure. For a start, it depicts just one of many available operating modes and is limited to the animation of the output skeleton (unlike the video of Chapter 5, which, in part, also shows the input motions animated). Similarly, many preprocessing steps, such as measures to correct input motion deficiencies, or further aspects of hybrid network initialisation (implied by Chapter 6) could not be detailed in the limited space. Furthermore, emphasis is on providing straightforward support for the content of Chapters 5, 6 and 7, best achieved by disregarding efficiency considerations in certain areas, which in reality were not overlooked. For example, it is clear that the preprocessing steps of Chapter 5, Section 5.4, need not be performed at each program launch, since, as mentioned in that chapter (Section 5.7.1) previously stored results were quickly loaded instead. However, it is useful to allude to these steps in Figure C.1, as this specifies where they belong in the overall algorithm.

The set-up conveyed diagrammatically in the figures is thus *not* presented as any kind of research contribution, nor as any reference, but serves merely to elucidate the work presented *elsewhere* in this thesis, ie that of Chapters 5, 6 and 7.

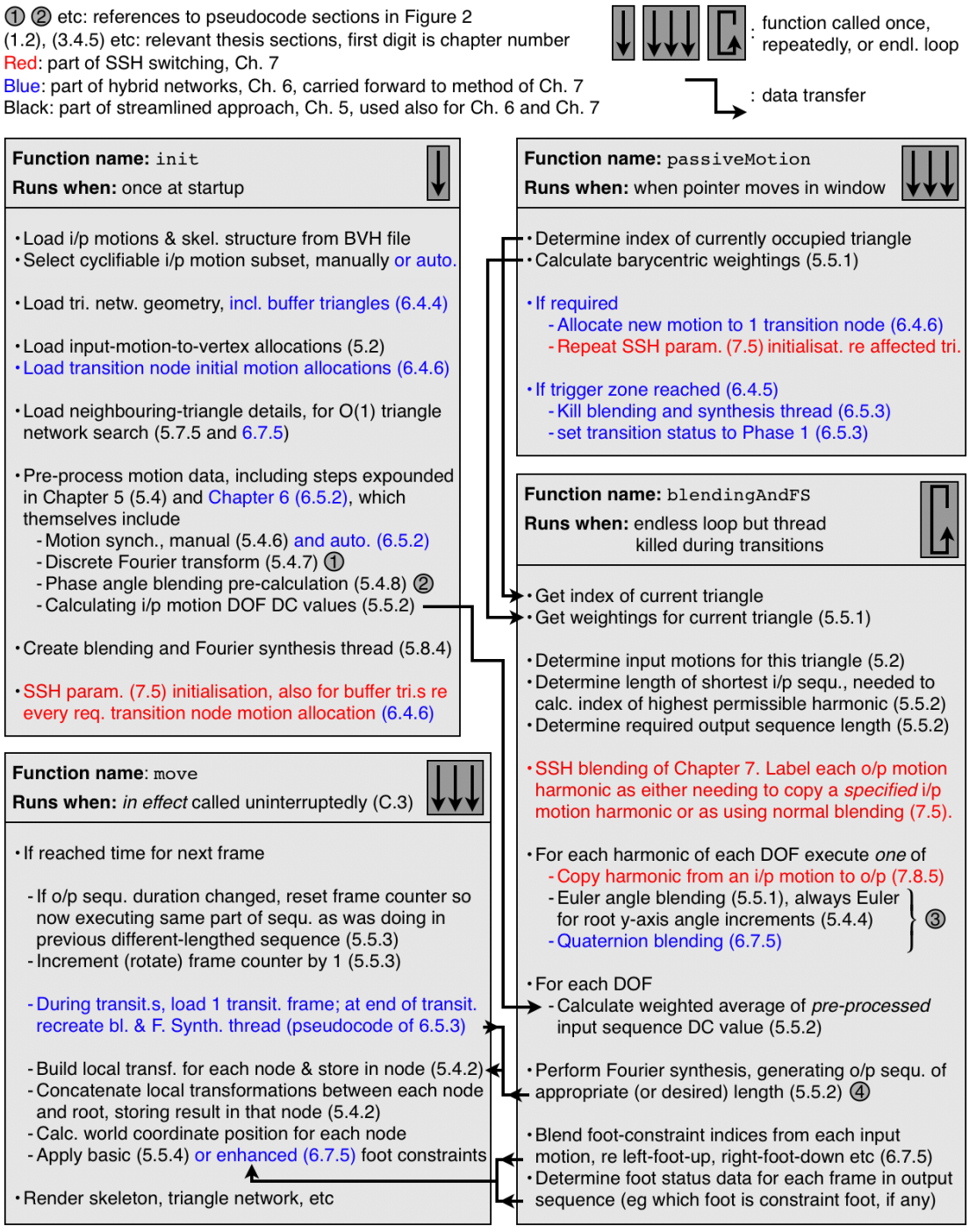

Figure C.1: Partial overview of the SSH switched system of Chapter 7, which encompasses hybrid networks and the streamlined approach, showing the relative positioning of individual components within the overall system. Selected program modules highlighted by numbered markers (1 to 4) are expanded in Figure C.2. Employed abbreviations include: *i/p* (input), *o/p* (output), *tri.* (triangle), *skel.* (skeleton), *netw.* (network), *transf.* (transformation), *sequ.* (sequence), *initialisat.* (initialisation), *param.* (parameter) and *transit.* (transition). (Caveat of Section C.4 applies).

① ② etc: references to sections in system algorithm of Figure 1
(1.2), (3.4.5) etc: relevant thesis sections, first digit is chapter number
(Eq. 1.2), (Eq. A.1) etc: corresponding thesis equations, first character defines chapter or appendix

Function name: `init`
Runs when: once at startup
Preprocessing/runtime: preprocessing

① **Discrete Fourier Transform** (5.4.7)

```
for each input motion
  for each DOF (Euler or quaternion)
    for each harmonic k
```

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi nk}{N}} \quad \text{(Eq. 5.8)}$$

```
      // whereby Xk = realk + j imagk
    }
    mk = √realk^2 + imagk^2   // magn.
    pk = atan2(imagk, realk) // ph. ang.
  }
}
```

② **Phase Angle Blending Pre-calc**. (5.3)

```
for each input motion i
  for each DOF in skeleton
    for each harmonic k
      calc. mki*cos(pki) and mki*sin(pki)
    }
  }
}
```

Function name: `move`
Runs when: *in effect* called uninterruptedly (C.3)
Preprocessing/runtime: runtime

Function name: `passiveMotion`
Runs when: when pointer moves in window
Preprocessing/runtime: runtime

Function name: `blendingAndFS`
Runs when: endless loop but thread killed during transitions
Preprocessing/runtime: runtime

③ **Freq. Domain Blending re Current Tri.**

```
for each DOF (Euler or quaternion)
  for each harmonic k in use
    // Magnitude blending
    mk0 = w1*mk1 + w2*mk2 + w3*mk3
    // where mk0, mk1, etc are kth harm.
    // magn. of o/p motion, i/p 1 etc,
    // and w1, w2, w3 are weightings.

    // Alternatively, for best results
```

$$m_{k_{blended}} = \sqrt{(h_{k_{Rx}}^2 + h_{k_{Ry}}^2)} \quad \text{(Eq. B.35)}$$

```
    // Ph. ang. blend. w/ pre-calc. values
    Blend mk*sin(pk) re i/p 1,2,3   (Eq. 5.3)
    Blend mk*cos(pk) re i/p 1,2,3   (Eq. 5.4)
```

$$p_{k,\,blended} = \arctan2\big((m_k \sin p_k)_{blended}, (m_k \cos p_k)_{blended}\big) \quad \text{(Eq. 5.2)}$$

```
  }
}
```

④ **Fourier Synthesis** (5.5.2)

```
for each DOF
  DCmean = weighted avg. of i/p sequ. means
  for each o/p sequence frame n
```

$$R_n = \text{DC}_{\text{mean}} + \frac{2}{N}\sum_{k=1}^{H} m_k \cos\left(\frac{2\pi kn}{\mathcal{N}} + p_k\right)$$

```
  }
}
```

Eq. 5.10, modified as per Section 5.5.2

Figure C.2: Pseudocode expanding selected sections of the system algorithm of Figure C.1. The identical outlines of Figures C.1 and C.2 highlight the interrelationship between them. Key thesis equations are shown in the context of neighbouring pseudocode, and in that of the overall system. Explanations of equation details can be found in the relevant chapters via the provided equation reference numbers. Abbreviations new to this figure include *avg.* (average), *magn.* (magnitude) and *ph. ang.* (phase angle). (Caveat of Section C.4 applies).

Appendix D

Motion Synchronisation and Phase Spectra

D.1 Introduction

This appendix explains and demonstrates the effect of motion synchronisation (Chapter 5 Section 5.4.6 and Chapter 6 Section 6.5) on the phase angles of input motion harmonics used in character motion blending. Furthermore, the uneven impact of synchronisation – weighted towards the lowest of frequencies – is shown to be a valid and necessary approach.

Harmonic magnitudes are not discussed, as not affected by synchronisation.

D.2 Pendulums

D.2.1 Unsynchronised

Pendulums are initially considered[1], providing a rough approximation of leg motion while walking or running. Figure D.1, top, shows two out-of-synch pendulums of iden-

[1]Proper treatment of pendulum physics would distract from the motion synchronisation focus of this appendix. It is sufficient to state that a swinging pendulum approximates the sinusoidal pattern of simple harmonic motion under given conditions whose provision, however, need not be considered here.

tical length and hence identical frequency[1], whose pure sinusoidal motions yield the illustrated phase spectra via the DFT (the spectra are shown as a single combined chart). For each pendulum, the simple motion contains a single frequency component, the fundamental, with an associated phase angle. As seen in Figure D.1, top, the difference between phase angles corresponds to the offset between the pendulums.

D.2.2 Synchronised

In the character animation work of this thesis, motion synchronisation applies a time shift to one of two cyclified input motions, to ensure that their gaits move in step. This is identical to setting the two pendulums so they swing in synchrony, thus removing their relative offset, which, as shown in Figure D.1, bottom, equalises their phase angles. As the blue pendulum could have been set to match the red instead of vice versa, it follows that although synchronisation compels the phase angles to be equal, it imposes no constraint on their actual mutual value[2].

D.3 Character Motion

D.3.1 Synchronisation Mechanics – Aligning Fundamentals

Leg motion is complex, unlike pendulum swings. However, the waveform of any of its DOFs is periodic, repeating at the same rate as the two-step cycle of the overall leg.

[1] This appendix is intended to support Chapter 7, Section 7.4 which explains the origin of UH distortion. UH distortion affects the blending of input motions, and can occur whether these are of equal or varying lengths. Similarly, the solution presented in Chapter 7 is effective whatever the relative input lengths. For simplicity, it can thus be assumed that all input motions have the *same* sequence length, and thus the same period to their cyclic motion, as do equal-lengthed pendulums.

[2] Were the pendulum motions to be discretised, their sinusoidal peaks might fall either on a sample or anywhere in between samples, and consequently, *precise* alignment of two such motions may no longer be possible. For this reason character animation data, being sampled, cannot be perfectly synchronised, and best-synchronisated motions yield very similar, but not strictly identical phase angles.

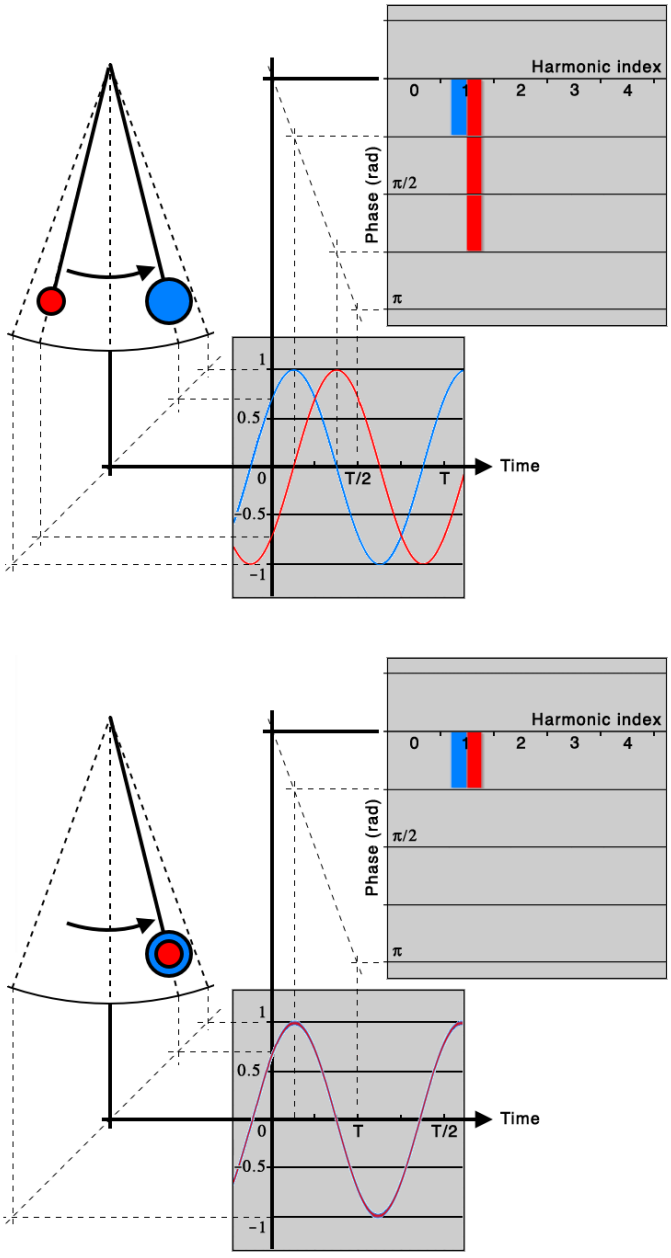

Figure D.1: (Top) Out-of-synch pendulums exhibit different phase angles. (Bottom) In-synch pendulums share a common phase angle, corresponding to the time-offset of their mutual motion. (Indicated phase angles were obtained via the DFT of Chapter 5, Equation 5.8, and are *directly* compatible with the corresponding *cosine-based* Fourier synthesis formula, Equation 5.10. Diagrams employ approximations and are for indication only.)

The DOF signal frequency is that of its most rudimentary component, the fundamental, since harmonics, being integer multiples thereof, do not change the repetition rate of the overall waveform. It follows that while the superposition of higher frequency harmonics adds intricacy to the trajectory of each DOF, and hence to the motion of the leg as a whole, these harmonics do *not* affect the rate of the leg's two-step cycle. The skeleton's step-frequency is thus governed by the DOF fundamentals, so corresponding DOFs of two, similar, in-step gaits, must have in-synch fundamentals, and vice versa for out-of-step gaits. This connection between synchronisation and fundamentals is confirmed by Figure D.2, which shows DOF phase angles for two character motions, and will be examined in the following sections. (For consistency with Chapter 7, Section 7.4, which this appendix aims to support, Figure D.2 illustrates z-axis rotations of the left hip, though any DOF between root and foot could be used).

D.3.2 Empirical Confirmation - Unsynchronised Motion

Repeated measurements of phase angle spectra for sundry unsynchronised cyclified motions, such as motions A and B of Figure D.2, top, reveal no correspondence whatsoever between the n^{th} harmonic of one, and the n^{th} of the other. This is no surprise, as the extent to which unsynchronised motions happen to be in- or out-of-step is entirely random.

Furthermore, a mere 13-frame offset (rotation) in the starting point of motion B (10% of its 125-frame cycle length) is seen in Figure D.2, centre, in conjunction with repeated observations during similar experiments, to *completely* reshuffle the phase angles of all but the fundamental harmonic[1], with some changing greatly, and others inevitably less so, as consistent with random change. (Due to the circular nature of

[1]There is no sharp cut-off point. The fundamental is least affected and neighbouring harmonics more so, though the impact is great even on the second and third harmonic.

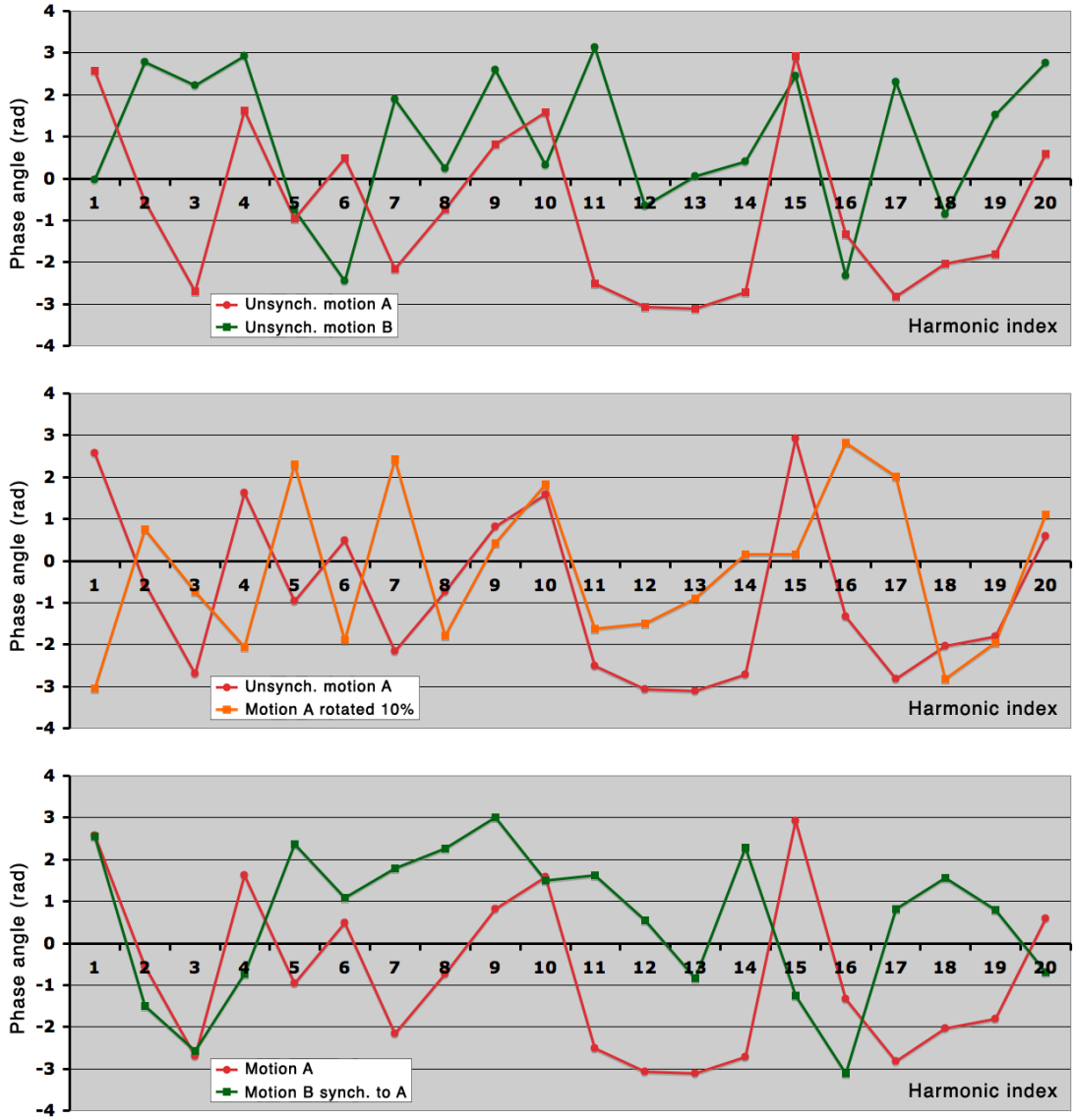

Figure D.2: (Top) Unsynchronised motions A and B exhibit *effectively* random phase angle correspondences (confirmed by the repeated observation of many such motions). (Middle) Motion A shown twice, firstly as previously in the top of the figure, and secondly, rotated (time-shifted) by 13 frames – 10% of the sequence length. Phase angles become reshuffled, though little change (considering the circular nature of angles) is seen at the fundamental. (Bottom) Motion B synchronised to meet motion A, thus time-shifted to ensure the gaits move in step. Lower harmonics then show phase angle alignment, which in general, based on several examples, tends to decrease with increasing frequency.

phase angles, those of the fundamental of the pre- and post-offset motions in Figure D.2, centre, ie 2.6 and -3.1 rad, are in fact fairly close despite their separation in the plot). This wild reshuffling of all but the lowest frequencies upon modest changes of input cycle starting point is significant, as *any* starting frame might have been chosen for unsynchronised sequences, since no constraint is imposed on this.

Thus while all phases were calculated, and exhibit predictable behaviour, this behaviour is nevertheless *effectively* random, with a randomness inherent in the motion data itself.

D.3.3 Empirical Confirmation - Synchronised Motion

Figure D.2, bottom, shows the measured effects of synchronising motion B so it moves in step with motion A.

Fundamentals. The phase angles of the two fundamentals are seen to have become near-identical, akin to those of the pendulums in Figure D.1, bottom. The particular value of phase angle is not relevant, as explained in Section D.2.2.

Higher frequencies. The effect on higher frequencies can be said to be nil, since, as described in Sections D.3.2, above, their phase angles were effectively random *prior* to synchronisation, and their observed, reshuffled, *post*-synchronisation state, seen in Figure D.2, bottom, is also effectively random. Such higher-frequency imperviousness to synchronisation was implied by Section D.3.2.

Intermediate frequencies. Frequencies, lying close to the fundamental, are seen – especially in repeated observations of which Figure 7.10, bottom, of Chapter 7 is a further example – to exhibit a *degree* of correspondence after synchronisation. This is to be expected, as in-step walking motions share a resemblance, at a

basic level (that coarser than the delicate subtleties), so they must contain somewhat similar contributions from lower-end harmonics. Phase angle synchrony should clearly tend to fall with increasing harmonic index, as indeed witnessed at the bottom of Figure D.2 (and the bottom of Figure 7.10).

D.4 Discussion - Synchronisation Method Validity

This appendix has explained, and empirically confirmed, the effect of motion synchronisation on character animation DOF-sequence phase angles.

It was seen that phase angle alignment is greatest at the lower end of the spectrum and non-existent at higher frequencies. This is precisely the required behaviour, as low-end correspondence ensures motion similarity (in-step gaits) indispensable for blending, while aligning all harmonic phase angles throughout the spectrum would be nonsensical, as it would make all input motions in the blend so similar, that there would be little left to blend between.

Whether UH distortion (Chapter 7) could be checked by a selective phase angle synchronisation in the UH band (Chapter 7) is an interesting question, but one which suggests potential deficiencies, like the danger that imposing such alignment might indeed *create* distortion. In comparison, SSH switching is shown in Chapter 7 to successfully counteract it.

Appendix E

Demonstration Video Download Locations

E.1 Introduction

This appendix conglomerates download locations given in previous chapters, as well as links to demonstration videos from other researchers, with which the work presented in this thesis has been compared.

E.2 Single Download

Downloads all videos by thesis author as a single zip file. Identical to those available separately in Sections E.3 and E.4.

Single Download (169.7 MB)
`http://www.urbanmodellinggroup.co.uk/MRLM_PhD_Impls.zip`

E.3 Literature Review Implementations

Links to videos showing implementations by thesis author, of work published by others.

IK Robot Manipulator (Chapter 2, Section 2.4. 7.8 MB, 1:24.)
`http://www.urbanmodellinggroup.co.uk/IK_Lit_Impl.mp4.zip`

Dynamic Constraints (Chapter 3, Section 3.6. 5.3 MB, 1:06.)
`http://www.urbanmodellinggroup.co.uk/BB88_Lit_Impl.mp4.zip`

E.4 Own Contributions of Chapters 5, 6 and 7

Links to demonstrations of own contributions, as presented in Chapters 5, 6 and 7.

Streamlined Approach (Chapter 5. 39 MB, 2:18.)
`http://www.urbanmodellinggroup.co.uk/fouBlend.mp4.zip`

Hybrid Networks (Chapter 6. 105.2 MB, 1:38.)
`http://www.urbanmodellinggroup.co.uk/hybridnetworks.mp4.zip`

SSH Switching (Chapter 7. 12.4 MB, 3:12.)
`http://www.urbanmodellinggroup.co.uk/SSH_Switching.mp4.zip`

E.5 Other Researchers

Links to work of other researchers, whose videos are referred to in Chapters 6 and 7.

Registration Curves (Chapter 6, Section 6.8.1)
`http://research.cs.wisc.edu/graphics/Gallery/kovar.vol/RegistrationCurves/regCurves.avi`

Parametric Motion Graphs (Chapter 7, Section 7.1)
`http://pages.cs.wisc.edu/~heckr/Research/pmg.html`

References

[Aba01] A. Sezgin Abalı. Animation of human motion with inverse kinematics using nonlinear programming. Master's thesis, Institute of Engineering and Science of Bilkent University, 2001.

[ABC96] Kenji Amaya, Armin Bruderlin, and Tom Calvert. Emotion from motion. In *Proceedings of the conference on Graphics interface '96*, GI '96, pages 222–229, Toronto, Ont., Canada, Canada, 1996. Canadian Information Processing Society.

[Add01] Alonzo C. Addison. Virtual Heritage - Technology in the Service of Culture. In *Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage*, VAST '01, pages 343–354, New York, NY, USA, 2001. ACM.

[AF02] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, 2002.

[AFO03] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3):402–408, 2003.

[AG90] Susan Amkraut and Michael Girard. Eurhythmy: Concept and process. 1(1):15–17, August 1990.

[AMH03] Amr Ahmed, Farzin Mokhtarian, and Adrian Hilton. Cyclification of human motion for animation synthesis. In *Short Paper Proceedings of Eurographics 2003*, 2003.

[Aut] Autodesk. Autodesk MotionBuilder. `http://usa.autodesk.com/adsk/servlet/pc/index?id=13581855&siteID=123112`. Last accessed Wednesday 10 October, 2012.

[AW01] G. Ashraf and K. Wong. Constrained framespace interpolation. In *Comp. Anim. 2001*, pages 61–72, South Korea, 2001.

[Baz72] André Bazin. *What Is Cinema? (Vol. 2)*. University of California Press, October 1972.

[BB88] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 179–188, New York, NY, USA, 1988. ACM.

[BC89] A. Bruderlin and T. W. Calvert. Goal-directed, dynamic animation of human walking. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 233–242, New York, NY, USA, 1989. ACM.

[BD09] Gregory Bennett and Andrew Denton. Developing practical models for teaching motion capture. In *ACM SIGGRAPH ASIA 2009 Educators Program*, SIGGRAPH ASIA '09, pages 3:1–3:5, New York, NY, USA, 2009. ACM.

[Bec03] Howard Beckerman. *Animation: the whole story.* Allworth Press, NY, revised edition (1 oct 2003) edition, 2003.

[BEL02] Norman I. Badler, Charles A. Erignac, and Ying Liu. Virtual humans for validating maintenance procedures. *Commun. ACM*, 45(7):56–63, July 2002.

[BES02] J. L. Brulé, G. Escarguel, and D. Sacchi. *L'art paléolithique à l'air libre: le paysage modifié par l'image*. GAEP, 2002.

[BG04] Christos Bouras and Eri Giannaka. Performance Monitoring on Networked Virtual Environments. In *International Conference on Internet Computing'04*, pages 302–308, 2004.

[Blo04] Jonathan Blow. Understanding Slerp, Then Not Using It. *Game Developer*, April 2004.

[BM00] Christoph Bregler and Jitendra Malik. Tracking people with twists and exponential maps. Technical report, Berkeley, CA, USA, 2000.

[Boo03] Betsy Book. Traveling through Cyberspace: Tourism and Photography in Virtual Worlds. In *Tourism & Photography: Still Visions - Changing Lives. 2003 Conference.*, June 2003.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3D characters. *ACM Trans. Graph.*, 26, July 2007.

[Bra10] Marta Braun. *Eadweard Muybridge (Critical Lives)*. Reaktion Books, 2010.

[Bur08] C. Burrus. Fast Fourier Transforms. `http://cnx.org/content/col10550/1.21/`, 2008. Online book, last accessed 4 May 2012.

[BW75] N. Burtnyk and M. Wein. Computer animation of free form images. *SIGGRAPH Comput. Graph.*, 9(1):78–80, April 1975.

[BW95] Armin Bruderlin and Lance Williams. Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104, New York, NY, USA, 1995. ACM.

[Cat78] Edwin Catmull. The problems of computer-assisted animation. *SIGGRAPH Comput. Graph.*, 12(3):348–353, August 1978.

[Cat98] Edwin Catmull. *Computer Animation: A Whole New World.* Nippan, 1998.

[CB97] E. Sahin Conkur and Rob Buckingham. Clarifying the definition of redundancy as used in robotics. *Robotica*, 15(5):583–586, September 1997.

[CBC+97] Gordon Cameron, Andre Bustanoby, Ken Cope, Steph Greenberg, Craig Hayes, and Olivier Ozoux. Motion capture and CG character animation (panel). In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 442–445, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[CC10] N. Courty and A. Cuzol. Conditional stochastic simulation for character animation. *Comput. Animat. Virtual Worlds*, 21(3‐4):443–452, May 2010.

[CDH00] Anthony Croft, Robert Davison, and Martin Hargreaves. *Engineering Mathematics.* Pearson Education Limited, Prentice Hall, third edition, 2000.

[Cen07] Michael Century. Exact imagination and distributed creativity: a lesson from the history of animation. In *Proceedings of the 6th ACM SIGCHI*

conference on Creativity & cognition, C&C '07, pages 83–90, New York, NY, USA, 2007. ACM.

[CH01] Gordon Collins and Adrian Hilton. Models for character animation. *Software Focus*, 2:44–51, 2001.

[CH07] Jinxiang Chai and Jessica K. Hodgins. Constraint-based motion optimization using a statistical dynamic model. *ACM Trans. Graph.*, 26(3), July 2007.

[CHP89] J. E. Chadwick, D. R. Haumann, and R. E. Parent. Layered construction for deformable animated characters. *SIGGRAPH Comput. Graph.*, 23:243–252, July 1989.

[CLW69] J. Cooley, P. Lewis, and P. Welch. The finite Fourier transform. *Audio and Electroacoustics, IEEE Transactions on*, 17(2):77 – 85, jun 1969.

[CMU] Graphics Lab CMU. CMU Graphics Lab Motion Capture Database. `http://mocap.cs.cmu.edu`. Last accessed Wednesday 10 October, 2012.

[Coo03] James W. Cooley. Fast Fourier transform (FFT). In *Encyclopedia of Computer Science*, pages 695–698. John Wiley and Sons Ltd., Chichester, UK, 2003.

[Cou06] M. Cousins. *The Story of Film: A Worldwide History.* Da Capo Press, 2006.

[CT65] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[DF02] Daniela Pucci De Farias. *The linear programming approach to approximate dynamic programming: theory and application.* PhD thesis, Stanford, CA, USA, 2002. AAI3048515.

[DFD08] Eva David, Lucienne Filippi, and Cléliat Dufayet. OS DE L'AUTOPODE (MÉTAPODES, PHALANGES PROXIMALES ET GRANDS SÉSAMOÏDES). January 2008.

[DRRT07] George Drettakis, Maria Roussou, Alex Reche, and Nicolas Tsingos. Design and Evaluation of a Real-World Virtual Environment for Architecture and Urban Planning. *Presence: Teleoper. Virtual Environ.*, 16(3):318–332, June 2007.

[DV90] P. Duhamel and M. Vetterli. Fast Fourier transforms: a tutorial review and a state of the art. *Signal Process.*, 19(4):259–299, April 1990.

[EPO95] Chris Esposito, W. Bradford Paley, and JueyChong Ong. Of mice and monkeys: a specialized input device for virtual body animation. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pages 109–ff., New York, NY, USA, 1995. ACM.

[ESHD05] Kenny Erleben, Jon Sporring, Knud Henriksen, and Kenrik Dohlman. *Physics-based Animation (Graphics Series).* Charles River Media, Inc., Rockland, MA, USA, 2005.

[FHP07] Christos Faloutsos, Jessica Hodgins, and Nancy Pollard. Database techniques with motion capture. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Fre08] Sara Freitas. Serious Virtual Worlds: a Scoping Study. Joint Information Systems Committee, 2008.

[Fur99] Maureen Furniss. Motion capture. In *Media in Transition Conference at MIT*, October 1999.

[GBT04a] P Glardon, R Boulic, and D Thalmann. A coherent locomotion engine extrapolating beyond experimental data. In *CASA*, pages 73–84, 2004.

[GBT04b] P. Glardon, R. Boulic, and D. Thalmann. PCA-based walking engine using motion capture data. In *CGI '04*, pages 292–298, USA, 2004. IEEE Comp. Soc.

[GBT06] Pascal Glardon, Ronan Boulic, and Daniel Thalmann. Robust on-line adaptive footplant detection and enforcement for locomotion. *Vis. Comput.*, 22(3):194–209, March 2006.

[GCFD07] Prabath Gunawardane, Eddy Chandra, Tien-Chieng Jack Feng, and James Davis. Keyframe animation using an artist's doll. In *ACM SIGGRAPH 2007 posters*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[GE07] Mohammad Ghavamzadeh and Yaakov Engel. Bayesian Actor Critic: A Bayesian Model for Value Function Approximation and Policy Learning, 2007. From 2-page summary by authors of Bayesian actor-critic algorithms. ICML24 (pp. 297304).

[Ger04] Margaret S. Geroch. Motion capture for the rest of us. *Journal of Computing Sciences in Colleges*, 19:157–164, January 2004.

[GHS+02] Margaret S. Geroch, Evan Hirsch, Joan Staveley, Tom Tolles, Barb Helfer, and Suba Varadarajan. How does motion capture affect animation? In *ACM SIGGRAPH 2002 conference abstracts and applications*, SIGGRAPH '02, pages 103–104, New York, NY, USA, 2002. ACM.

[GLD08] R. Galvao, R. G. Laycock, and A. M. Day. GPU techniques for creating visually diverse crowds in real-time. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, VRST '08, pages 79–86, New York, NY, USA, 2008. ACM.

[Gle98] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[GMPO00] Thanh Giang, Robert Mooney, Christopher Peters, and Carol O'Sullivan. Real-time character animation techniques. Technical report, Trinity College Dublin, 2000.

[GMWC06] Yan Gao, Lizhuang Ma, Xiaomao Wu, and Zhihua Chen. From keyframing to motion capture: The evolution of human motion synthesis. In *Human Interaction with Machines*, pages 35–42. Springer Netherlands, 2006.

[GW91] Michael Gleicher and Andrew Witkin. Differential manipulation. In *Graphics Interface*, pages 61–67, 1991.

[Har61] Lee Harrison. Notes for an early animation device. (Title used for 1992 reprint). *Eigenwelt der Apparate-Welt, 1992 Reprint of 1961 article*, pages 209–223, 1961.

[HG07] Rachel Heck and Michael Gleicher. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 129–136, New York, NY, USA, 2007. ACM.

[HOB98] Jessica K. Hodgins, James F. Obrien, and Robert E. Bodenheimer. Computer animation. In *Encyclopedia of Computer Science*, pages 301–304, 1998.

[Hod09] Gray Hodgkinson. The seduction of realism. In *ACM SIGGRAPH ASIA 2009 Educators Program*, SIGGRAPH ASIA '09, pages 4:1–4:4, New York, NY, USA, 2009. ACM.

[HWBO95] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1995. ACM.

[Inca] ActiveWorlds Inc. ActiveWorlds: The Web's most powerful 3D virtual worlds platform. `http://www.activeworlds.com/`. Last accessed Wednesday 10 October, 2012.

[Incb] Linden Research Inc. Second Life: Your World. Your Imagination. `http://secondlife.com/`. Last accessed Wednesday 10 October, 2012.

[Joh02] Scott F. Johnston. Lumo: illumination for cel animation. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, pages 45–ff, New York, NY, USA, 2002. ACM.

[Jon88] Crary Jonathan. Techniques of the observer. *October*, 45 (Summer 1988):3–35, 1988.

[Jul81] B. Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802):91–97, March 1981.

[KC77a] Lynn T. Kozlowski and James E. Cutting. Recognizing friends by their walk: Gait perception without familiarity cues. *Bulletin of the Psychonomic Society*, 9(5):353–356, 1977.

[KC77b] Lynn T. Kozlowski and James E. Cutting. Recognizing the sex of a walker from a dynamic point-light display. *Perception & Psychophysics*, 21(6):575–580, 1977.

[KG03] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 214–224, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[KG04] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, August 2004.

[KGP02] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482, New York, NY, USA, 2002. ACM.

[Kil96] Mark J. Kilgard. The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3. `http://www.opengl.org/resources/libraries/glut/glut-3.spec.pdf`, 1996.

[Kor02] Alexander Kort. Computer aided inbetweening. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, NPAR '02, pages 125–132, New York, NY, USA, 2002. ACM.

[KR07] T. S. Kelly and Anthony Rhind. *MC Insight: Marketing in Second Life and Other Virtual Worlds*. Media Contacts, Havas Digital, October 2007.

[Kra04] Jon S Krasner. *Motion Graphic Design & Fine Art Animation: Principles and Practice*. Elsevier Science & Technology, 2004.

[KSG02] Lucas Kovar, John Schreiner, and Michael Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, pages 97–104, New York, NY, USA, 2002. ACM.

[KYT+06] L. Kharevych, Weiwei Yang, Y. Tong, E. Kanso, J. E. Marsden, P. Schröder, and M. Desbrun. Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, pages 43–51, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Lan98] Jeff Lander. Working with motion capture file formats. *Game Developer*, January 1998.

[Las87] John Lasseter. Principles of traditional animation applied to 3D computer animation. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 35–44, New York, NY, USA, 1987. ACM.

[Las01] John Lasseter. Tricks to animating characters with a computer. *SIGGRAPH Comput. Graph.*, 35:45–47, May 2001.

[LBJK09] M. Lau, Z. Bar-Joseph, and J. Kuffner. Modeling spatial and temporal variation in motion data. *ACM Trans. Graph. (SIGGRAPH ASIA 2009)*, 28(5), 2009.

[LCF00] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[LCL06] Kang Hoon Lee, Myung Geol Choi, and Jehee Lee. Motion patches: building blocks for virtual environments annotated with motion data. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 898–906, New York, NY, USA, 2006. ACM.

[LCR+02] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, 2002.

[Les93] Richard J Leskosky. Phenakistoscope: 19th century science turned to animation. *Film History*, 5(2):176–189, 1993.

[LL06] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. *Graph. Models*, 68(2):158–174, 2006.

[LMT07] Etienne Lyard and Nadia Magnenat-Thalmann. A simple footskate removal method for virtual reality applications. *Vis. Comput.*, 23(9):689–695, 2007.

[LoCIoI10] Deloitte Touche Tohmatsu India Private Limited and Associated Chambers of Commerce & Industry of India. *Animation, Broadcasting, Gaming: On the Cusp of Growth*. ASSOCHAM, 2010.

[Los06] Elizabeth Losh. The Palace of Memory: Virtual Tourism and Tours of Duty in Tactical Iraqi and Virtual Iraq. In *Proceedings of the 2006 international conference on Game research and development*, CyberGames '06, pages 77–86, Murdoch University, Australia, Australia, 2006. Murdoch University.

[LP02] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on computer graphics and interactive techniques*, pages 408–416, New York, NY, USA, 2002. ACM.

[LWC+11] Huajun Liu, Xiaolin Wei, Jinxiang Chai, Inwoo Ha, and Taehyun Rhee. Realtime human motion control with a small number of inertial sensors. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 133–140, New York, NY, USA, 2011. ACM.

[LWS02] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 465–472, New York, NY, USA, 2002. ACM.

[MA95] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Mach. Learn.*, 21(3):199–233, December 1995.

[Mad01] M. Meredith S. Maddock. Motion capture file formats explained, 2001.

[Mar08] Azéma Marc. Representation of movement in the upper palaeolithic: An ethological approach to the interpretation of parietal art. *Anthropozoologica*, 43 (1):117–154, 2008.

[May] Autodesk Maya. `http://usa.autodesk.com/maya/`. Originally developed by AliasWavefront. Last accessed Tuesday 24 July, 2012.

[MCC09] Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. Interactive generation of human animation with deformable motion models. *ACM Trans. Graph.*, 29(1):9:1–9:12, December 2009.

[Med00] Robert Meddins. *Introduction to Digital Signal Processing*. Newnes, 2000.

[MFCD99] Franck Multon, Laure France, Marie-Paule Cani, and Gilles Debunne. Computer animation of human walking: a survey. *Journal of Visualization and Computer Animation (JVCA)*, 10:39–54, 1999. Published under the name Marie-Paule Cani-Gascuel.

[MG03] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 562–568, New York, NY, USA, 2003. ACM.

[MJ00] Kantilal Varichand Mardia and Peter E. Jupp. *Directional statistics*. Wiley series in probability and statistics. Wiley, Chichester, 2000. Previous ed. published as: Statistics of directional data. London : Academic Press, 1972.

[MLD+08] Rachel McDonnell, Michéal Larkin, Simon Dobbyn, Steven Collins, and Carol O'Sullivan. Clone attack! Perception of crowd variety. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–8, New York, NY, USA, 2008. ACM.

[MLD10] Michael Molnos, Stephen Laycock, and Andy Day. Using the Discrete Fourier Transform for Character Motion Blending and Manipulation - a

Streamlined Approach . In *EG UK Theory and Practice of Computer Graphics 2010*, pages 207–214, Sheffield University, United Kingdom, 2010. Eurographics Association.

[Moo65] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):114–117, April 1965.

[Mota] MotionAnalysis. MotionAnalysis Raptor-4 Digital RealTime System. `http://www.motionanalysis.com/html/animation/raptor4.html`. Last accessed Saturday 11 August, 2012.

[Motb] MotionAnalysis. MotionAnalysis: The Industry Leader for 3D Passive Optical Motion Capture. `http://www.motionanalysis.com/`. Last accessed Wednesday 10 October, 2012.

[Motc] MotionCapture3D. MotionCapture3D Motion Capture Studio. `http://motioncapture3d.com`. Last accessed Tuesday 19 July, 2011.

[MTT96] N. Magnenat-Thalmann and D. Thalmann. Computer animation. *ACM Computing Surveys '96*, 28(1):161–163, 1996.

[ND10] John Nickolls and William J. Dally. The GPU Computing Era. *IEEE Micro*, 30(2):56–69, March 2010.

[Neb99] Jean-Christophe Nebel. Keyframe interpolation with self-collision avoidance. In *Eurographics, Springer Computer Science*, pages 77–86. Springer, 1999.

[NHAH03] Christopher Niederauer, Mike Houston, Maneesh Agrawala, and Greg Humphreys. Non-invasive Interactive Visualization of Dynamic Architec-

tural Environments. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, I3D '03, pages 55–58, New York, NY, USA, 2003. ACM.

[Nik94] Daniel Nikovski. Dynamic simulation methods for animation of legged locomotion. Technical report, Southern Illinois University at Carbondale, 1994.

[NVI12] NVIDIA. *CUDA C Programming Guide Version 4.2*. NVIDIA, 2012. NVIDIA Developer Zone Documentation.

[Ope] OpenSimulator. `http://opensimulator.org/`. Last accessed Wednesday 10 October, 2012.

[Ott03] E. Otten. Inverse and forward dynamics: models of multi-body systems. *Philos. Trans. R. Soc. Lond. B. Biol. Sci.*, 358(1437):1493–500, 2003.

[PB00] Katherine Pullen and Christoph Bregler. Animating by multi-level sampling. In *Proceedings of the Computer Animation*, CA '00, pages 36–, Washington, DC, USA, 2000. IEEE Computer Society.

[PCLS05] Michael Pratscher, Patrick Coleman, Joe Laszlo, and Karan Singh. *Outside-In* anatomy based character rigging. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, pages 329–338, New York, NY, USA, 2005. ACM.

[Pej10] Jana Pejoska. Design Principles of Educational Virtual Worlds for Preschool Children: A Case Study of JumpStart World Kindergarten's Pedagogical Methods. Master's thesis, University of Jyväskylä, May 2010.

[Per85] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985.

[PG96] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 205–216, New York, NY, USA, 1996. ACM.

[Pix86] Pixar. Luxo Jr. Short film, 1986.

[PL06] Julien Pettré and Jean-Paul Laumond. A motion capture-based control-space approach for walking mannequins: Research articles. *Comput. Animat. Virtual Worlds*, 17(2):109–126, 2006.

[Pop00] Zoran Popović. Controlling physics in realistic character animation. *Commun. ACM*, 43(7):50–58, July 2000.

[PSS02] S. Park, Hyun J. Shin, and S. Shin. On-line locomotion generation based on motion blending. In *SCA '02*, pages 105–111, NY, USA, 2002. ACM.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing.* Cambridge University Press, New York, NY, USA, 1992.

[RCB98] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5):32–40, 1998.

[Ree81] William T. Reeves. Inbetweening for computer animation utilizing moving point constraints. *SIGGRAPH Comput. Graph.*, 15(3):263–269, August 1981.

[RGBC96] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In

Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, SIGGRAPH '96, pages 147–154, New York, NY, USA, 1996. ACM.

[RP12] Tiago Ribeiro and Ana Paiva. The illusion of robotic life: principles and practices of animation for robots. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, HRI '12, pages 383–390, New York, NY, USA, 2012. ACM.

[SB94] Bruno Serra and Marc Berthod. Subpixel contour matching using continuous dynamic programming. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn*, pages 202–207. IEEE Computer Society press, 1994.

[Sch10a] Adriana Schulz. Motion Capture. Technical report, Instituto Nacional de Matemtica Pura e Aplicada, Rio de Janeiro, May 2010.

[Sch10b] Adriana Schulz. State of the Art of Character Animation. Technical report, Instituto Nacional de Matemtica Pura e Aplicada, Rio de Janeiro, May 2010.

[Sco03] Remington Scott. Sparking Life: Notes on the Performance Capture Sessions for The Lord of The Rings: The Two Towers. *SIGGRAPH Comput. Graph.*, 37(4):17–21, November 2003.

[Sen08] Pavel Senin. Dynamic time warping algorithm review. Technical Report CSDL-08-04, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, December 2008.

[Sha95] Hagit Shatkay. The Fourier Transform - A Primer. Technical report, Brown University, Providence, Rhode Island, 1995.

[Sha01] Ahmed A Shabana. *Computational Dynamics*. John Wiley & Sons, Inc, second edition, 2001.

[Sha10] Geoff Shaw. Personal communication, 2010. Motion Analysis Corporation.

[Sie] Siemens. Jack and Process Simulate Human. `http://www.plm.automation.siemens.com/en_us/products/tecnomatix/assembly_planning/jack/index.shtml`. Last accessed Friday 12 October, 2012.

[Sie98] Dave Sieg. Scanimation in the analog days. *SIGGRAPH Comput. Graph.*, 32:58–59, August 1998.

[Sim12] Timothy Simnet. Personal communications, 2008–2012. PhD student, cloth simulation, University of East Anglia, UK.

[SLBR04] Antonio Carlos Sementille, Luís Escaramuzi Lourenço, José Remo Ferreira Brega, and Ildeberto Rodello. A motion capture system using passive markers. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, VRCAI '04, pages 440–447, New York, NY, USA, 2004. ACM.

[SLSG01] Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. Computer puppetry: An importance-based approach. *ACM Trans. Graph.*, 20(2):67–94, April 2001.

[Smi02] Lindsay I. Smith. A tutorial on principal components analysis. `http://www.sccg.sk/~haladova/principal_components.pdf`, February 2002. Last accessed Tuesday 3 July, 2012.

[SN07] Jun'ichiro Seyama and Ruth S. Nagayama. The uncanny valley: Effect of realism on the impression of artificial human faces. *Presence: Teleoper. Virtual Environ.*, 16(4):337–351, August 2007.

[SO06] Hyun Joon Shin and Hyun Seok Oh. Fat graphs: constructing an interactive character with continuous controls. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 291–298, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[SPCM97] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 163–172, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[SRC01] Peter-Pike J. Sloan, Charles F. Rose, III, and Michael F. Cohen. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 135–143, New York, NY, USA, 2001. ACM.

[Ste79] Garland Stern. Softcel - an application of raster scan graphics to conventional cel animation. *SIGGRAPH Comput. Graph.*, 13(2):284–288, August 1979.

[Stu94] David Sturman. A brief history of motion capture for computer character animation. In *Character Motion Systems, SIGGRAPH 94: Course 9*, 1994.

[Stu98a] David Sturman. The state of computer animation. *SIGGRAPH Comput. Graph.*, 32(1):57–61, 1998.

[Stu98b] David J. Sturman. Computer puppetry. *IEEE Comput. Graph. Appl.*, 18(1):38–45, January 1998.

[SWND05] Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition) (OpenGL)*. Addison-Wesley Professional, 2005.

[Tal12] F.A.A. Talbot. *Moving Pictures*. Literature of Cinema. Arno Press, 1912.

[Tho58] Bob. Thomas. *Walt Disney, the art of animation: the story of the Disney Studio contribution to a new art*. Simon and Schuster, 1958.

[TJ95] Frank Thomas and Ollie Johnston. *The Illusion of Life: Disney Animation*. Disney Editions, 1995.

[TLP07] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 7, New York, NY, USA, 2007. ACM.

[TM04] S. C. L. Terra and R. A. Metoyer. Performance timing for keyframe animation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, pages 253–258, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[TOSU] ACCAD The Ohio State University. The Ohio State University Advanced Computing Centre for the Arts and Design Motion Capture Lab. `http://accad.osu.edu/research/mocap/mocap_data.htm`. Last accessed Wednesday 10 October, 2012.

[Tra94] Wes Trager. A practical approach to motion capture: Acclaim's optical motion capture system. In *Character Motion Systems, Siggraph '94 Course Notes*, 1994.

[Tro02] N. Troje. Decomposing biological motion: A framework for analysis and synthesis of human gait patterns. *Journal of Vision*, 2:371–387, 2002.

[UAT95] Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 91–96, New York, NY, USA, 1995. ACM.

[Val03] Hélène Valladas. Direct radiocarbon dating of prehistoric cave paintings by accelerator mass spectrometry. *Measurement Science and Technology*, 14(9):1487, 2003.

[Vas02] M. Alex O. Vasilescu. Human motion signatures: Analysis, synthesis, recognition. *Pattern Recognition, International Conference on*, 3:30456, 2002.

[VAV+07] Daniel Vlasic, Rolf Adelsberger, Giovanni Vannucci, John Barnwell, Markus Gross, Wojciech Matusik, and Jovan Popović. Practical motion capture in everyday surroundings. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[Vic] Vicon. VICON MOTIONBUILDER STREAM 2012. `http://www.vicon.com/products/ViconPlug-Ins.html`. Last accessed Saturday 11 August, 2012.

[WB97] Andrew Witkin and David Baraff. Physically based modeling: Principles and practice. In *Siggraph '97 Course Notes*, 1997.

[Wel93] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993.

[WF02] Greg Welch and Eric Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Comput. Graph. Appl.*, 22:24–38, November 2002.

[WH97] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Comput. Graph. Appl.*, 17(6):39–45, 1997.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1988. ACM.

[WMC11] Xiaolin Wei, Jianyuan Min, and Jinxiang Chai. Physically valid statistical models for human motion generation. *ACM Trans. Graph.*, 30(3):19:1–19:10, May 2011.

[WOR] KZERO WORLDWIDE. Growth Forecast for the Virtual Worlds Sector. `http://www.kzero.co.uk/blog/growth-forecasts-for-the-virtual-worlds-sector/`. Last accessed Thursday 16 August, 2012.

[WP85] G. Waterworth and R. P. Phillips. *Electrical Principles for Technicians Volume 2*. Edward Arnold (publishers) Ltd, London, UK, 1985.

[WP95] Andrew Witkin and Zoran Popović. Motion warping. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 105–108, New York, NY, USA, 1995. ACM.

[ZB94] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph.*, 13(4):313–336, 1994.

[ZLUR+12] Alexander Zook, Stephen Lee-Urban, Mark O. Riedl, Heather K. Holden, Robert A. Sottilare, and Keith W. Brawner. Automated Scenario Generation: Toward Tailored an Optimized Military Training in Virtual Environments. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, pages 164–171, New York, NY, USA, 2012. ACM.

[ZVDH03] Victor Brian Zordan and Nicholas C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 245–250, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.