Real-time Full Body Motion Control

John Collomosse and Adrian Hilton

Abstract This chapter surveys techniques for interactive character animation, exploring data-driven and physical simulation based methods. Interactive character animation is increasingly data-driven, with animation produced through the sampling, concatenation and blending of pre-captured motion fragments to create movement. The chapter therefore begins by surveying commercial technologies and academic research into performance capture. Physically based simulations for interactive character animation are briefly surveyed, with a focus upon technique proven to run in real-time. The chapter the focuses upon concatenative synthesis approaches to animation, particularly upon motion graphs and their parametric extensions for planning skeletal and surface motion for interactive character animation.

1 Introduction

Compelling visuals and high quality character animation are cornerstones of modern video games and immersive experiences. Yet character animation remains an expensive process. It can take a digital artist weeks to skin (design the 3D surface representation) of a character model, and then rig it with a skeleton to facilitate full body control and animation. Animation is often expedited by re-targeting human performance capture data to drive the character's movement. Yet creativity and artistic input remains in the loop, blending hand-crafted animation with motion capture data which itself may be an amalgam of multiple takes (e. g. it is common for separate passes to be used for face, head, and hands). Performance capture itself is

John Collomosse
Centre for Vision Speech and Signal Processing (CVSSP), University of Surrey, UK e-mail: j.collomosse@surrey.ac.uk

Adrian Hilton
Centre for Vision Speech and Signal Processing (CVSSP), University of Surrey, UK e-mail: a.hilton@surrey.ac.uk

expensive; equipment hire, operation and studio/actor time can approach millions of US dollars on a high-end production. The recent resurgence of virtual and augmented reality (VR/AR) experiences, in which character interaction takes place at very close quarters, is further driving up expectations of visual realism.

Creating believable interactive digital characters is therefore a trade-off between project budget and quality. Better tool support inevitably leads to efficiency and so a re-balancing toward higher quality. In this chapter we survey state of the art technologies and algorithms (as of 2015) for efficient interactive character animation. Whilst a common goal is a drive toward increased automation, which in some cases can produce interactive characters with near complete automation, one should not lose sight that these are tools only, and the need for the creative artist in the loop remains essential to reach the high quality bar demanded by modern production. As such this chapter takes a practical view on animation, first surveying the commercial technologies and academic research into performance capture and then surveying the two complementary approaches to real-time animation — physically based approaches (examined further in Chapter C-2) and data-driven approaches.

Although character animation is frequently used within other domains with the Creative Industries (movies, broadcast) its use within games requires new animation sequences to be generated on-the-fly, responding in real-time to user interaction and game events. This places some design restrictions on the underpinning algorithms (efficient data structures, no temporal look-ahead for kinematics). This chapter therefore focuses upon algorithms for interactive, rather than more general offline, character animation covered elsewhere in this book.

2 State of the Art

Historically character animation has been underpinned by meticulous observations of movement in nature, for example the gait cycles of people or animals. This link has been made explicit by contemporary character animation, which is trending toward a data driven process in which sampled physical performance is the basis for synthesising realistic movement in real-time. This chapter therefore begins by surveying commercial technologies, and state of the art Computer Vision algorithms, for capturing human motion data.

2.1 Commercial Technologies for Performance Capture

Motion Capture (mocap) technology was initially developed within the Life Sciences for human movement analysis. The adoption of mocap for digital entertainment, commonly referred to as Performance Capture (PC) is now widespread. PC accounts for 46% of the total 3D motion capture system market which is growing annually at a rate of around 10% and expected to reach 142.5 million US dollars

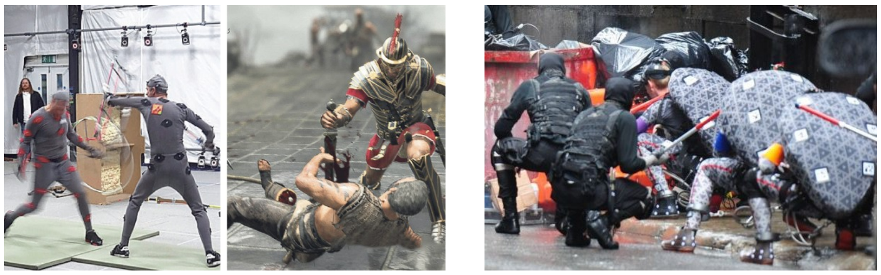


Fig. 1 Performance capture technologies. Left: Vicon IR based system being used to pre-visualize character performance in real-time within the UNREAL Games engine (EPIC). Right: Industrial Light and Magic's fractal suit enabling visual light based tracking outdoors.

by 2020 [29]. Indeed many of the innovations in mocap (e. g. marker-less capture) are now being developed within the Creative Industries, and transferred back into domains such as bio-mechanics and healthcare.

PC systems enable sequences of skeletal joint angles to be recorded from one or several actors. The key distinction between PC systems are the kind of physical marker or wearable device (if any) required to be attached to the actors.

The predominant form of PC in the Creative Industries is marker based, using passive markers that are tracked visually using synchronised multiple viewpoint video (MVV). Popular systems for passive marker PC are manufactured by *Vicon* (UK) and *Optitrack* (US), which require the actor to wear retro-reflective spheres (approximately 20-30 are typically used for full body capture). A region of the studio (capture volume) is surrounded by several infra-red (IR) cameras in known locations, and illuminated by several diffuse IR light sources. Prior to capture of performance data, a calibration process is performed to learn the relative locations (extrinsic parameters) of the cameras. This enables the world location of the markers attached to the actor to be triangulated, resulting in a 3D point cloud from which a skeletal pose is inferred using physical and kinematic constraints. Modern software (e. g.*Blade*, or *MotionBuilder*) can perform this inference in real-time providing immediate availability of a pose estimate for each actor in the scene. PC service providers (e. g.*The Imaginarium Studios*, London UK) have begun to harness this technology to pre-visualize the appearance of digital characters for movie or Game production during live actor performance. Such facilities provide immediate visual feedback to both the actor, and Director, on-set removing the trial-and-error and so improving efficiency in the capture process (Figure 1, left). Other forms of passive PC in regular use include the fractal suits patented by *Industrial Light and Magic* for full body motion capture (Figure 1, right). The suits are tracked using visible light, and so are more amenable to deployment in outdoor sets where strong natural light makes IR impractical.

Active marker based systems include offerings from *CodaMotion* (UK) and *PhaseSpace* (US). Markers are bright IR light emitting diodes (LEDs) that pulse with unique signatures that identify the marker to one or several observing cameras. Since markers are uniquely labelled at source, automated tracking of markers is trivial making marker confusion highly unlikely. By contrast, the labelling of triangulated markers in a passive system is performed during pose inference and may be incorrect in the presence of clutter (e. g. multiple occluding actors). Marker mislabelling causes errors in pose estimation, that can only be removed through addition of more witness cameras so reducing the chance of occlusion, or manually correcting the data post-capture. An advantage of active marker based systems is therefore the need for fewer cameras, and reduced data correction. Active systems tend to perform better outdoors, again due to obviating the need for large area IR illumination. The disadvantage is the additional expense and time required for actor set-up (wires and batteries) due to the complexity of the markers. The workflow to produce a skeletal pose estimate from active marker data is identical to passive systems, since the capture again results in a sequence of 3D point movements.

Inertial motion capture system use inertial motion sensing units (IMUs) to detect changes in joint orientation and movement, providing an alternative to visual tracking and so removing the problem of marker occlusion. IMUs are worn on each limb (around 12-14 for full body capture), and connected wirelessly to a hub which forwards the data for software processing. Common IMU captures solutions include *AnimeZoo* (UK), *XSens* (Netherlands) and most recently the crowd-funded *PerceptionNeuron* (US) system. All of these solutions again rely upon a robust back-end software product to infer a skeletal pose estimate using physical and kinematic constraints. The disadvantage of inertial capture is drift, since the IMUs output only a stream of relative joint angles. For this reason, IMU mocap is sometimes combined with a secondary modality e. g. laser ranging or passive video to capture the world-position of the actor.

An emerging form of PC is marker-less mocap, using Computer Vision to track the actor without the need for wearables. Although the accuracy of commercial marker-less systems has yet to reach parity with marker based solutions, the greatly reduced setup time and flexibility to use only regular video cameras for capture makes such systems a cost effective option. For the purposes of teaching data driven animation production, marker-less technologies are therefore attractive. Solutions include the *OrganicMotion* stage (US); a cube arrangement of around 20 machine vision cameras that calculates human pose using the silhouette of the performer against a uniform background from the multiple camera angles. More recently *The Captury* (Germany) launched a software-only product for skeletal PC that estimates pose against an arbitrary background using a possibly heterogeneous array of cameras. Yet although commercial solutions to marker-less PC remain in their infancy, academic research is making good progress as we next discuss.

2.2 Marker-less Human Motion Estimation

Passive estimation of human pose from video is a long-standing Computer Vision challenge, particularly when visual fiducials (markers) are not present. Methods can be partitioned into those considering monocular (single-view) video, or multiple view-point video.

2.2.1 Monocular Human Pose Estimation

Human pose estimation (HPE) often requires the regions of interest (ROIs) representing people to be identified within the video. This person localization problem is can be solved using background [39] or motion [1] subtraction, in the cases of simple background. In more cluttered scenarios, supervised machine can be applied to detect the presence of a person within a sliding window swept over the video frame. Within each position of the window, pre-trained classifiers based on Histogram of Gradient (HoG) descriptors can robustly identify the torso [12], face [35] or entire body [11].

Once the subject is localised within the frame, the majority of monocular HPE algorithms attempt to infer only a 2D i. e. apparent pose of the performer. These adopt either: a) top-down fitting of a person model, optimizing limb parameters and projecting to image space to evaluate correlation with image data; or b) individually segmenting parts and integrating their positions in a bottom-up manner to produce a maximal likelihood pose.

Bottom-up approaches dominated early research into HPE, over one decade ago. Srinivasan and Shi [31] used an image segmentation algorithm (graph-cut) to parse a subset of salient shapes from an image and group these into a shape resembling a person using a set of learned rules. However the approach was limited to a single person, and background clutter was reported to interfere with the initial segmentation and so the eventual accuracy of the approach. Ren *et al.* proposed an alternative algorithm in which Canny edge contours were recursively split into segments, each of which was classified as a putative body part using shape cues such as parallelism [28]. Ning *et al.* [16] similarly attempted to label body parts individually, applying a Bag of Visual Words (BoVW) framework to learn codewords for body zone labelling — segmenting 2D body parts to infer pose. Mori and Malik described the first bottom-up algorithm capable of estimating a 3D pose in world space, identifying the position of individual joints in a 2D image using scale and symmetry constraints – and then matching those 2D joint positions to a set of many ‘training images’ each of which had been manually annotated *a priori* with 2D joint positions [25], and was associated also with a 3D ground-truth. Once the closest training image had been identified by matching query and training joint positions in 2D, the relevant 3D pose was returned as the result.

Top-down approaches, in which the entire 2D image is used as evidence to fit a model are more contemporary. The most common form of model fitted to the image is a ‘Pictorial Structure’; essentially a collection of 2D limbs (regions) articulated by springs, that can be iteratively deformed to fit to evidence in the image under an optimization process [12, 2]. However such approaches do not yield recover a 3D pose estimate, or if so are unstable due to ambiguity in reasoning from a single image.

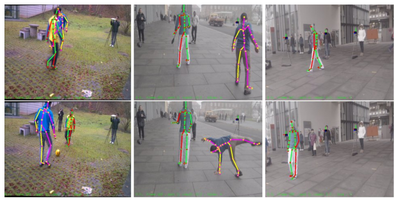
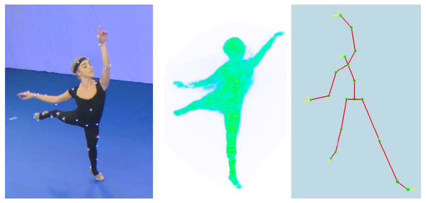
(a)  (b) 

Fig. 2 Convolutional neural networks (CNNs) used for pose estimation in multi-view point video. (a) using 2D detections of body parts fused in a 3D probabilistic model (from [13]), (b) recognition of pose from 3D volumetric data recovered from multiple views (from [34]).

2.2.2 Multi-view Human Pose Estimation

A 3D estimate of human pose may be inferred with less ambiguity using footage captured from multiple viewpoints. In such a setup, a configuration of cameras (typically surround a subject in a 180 or 360 degree arc) observes a *capture volume* within which a performance is enacted. The cameras are typically calibrated, i. e. for a subject observed by C camera views $c = [1,C]$ the extrinsic parameters $\{R_c, COP_c\}$ (camera orientation and focal point) and intrinsic parameters $\{f_c, o_c^x, o_c^y\}$ (focal length, and 2D optical centre) are known.

Two categories of approach exist: (a) those estimating 2D pose from each view independently, and fusing these to deduce a 3D pose; (b) those inferring a 3D pose from 3D geometric proxy of the performer recovered through volumetric reconstruction.

Computer vision has undergone a revolution in recent years, with deep convolutional neural networks (CNNs) previously popular in text recognition being extended and applied to solve many open problems including human pose estimation. CNNs have shown particularly strengths in general object detection, with some state of the art networks e. g. GoogLeNet (Google Inc.) surpassing human performance in certain scenarios. Most recently CNNs have also been used to detect human body parts in single and multiple viewpoint video and infer from these human pose. Elhayek et al. [13] estimate human body parts from individual video viewpoints using CNN detectors and the fuse these under a probabilistic model fusing colour, and motion constraints from a body part tracker to create a 3D pose estimate. The CNN detection step is robust to clutter, making the system suitable for estimation of 3D pose in complex scenes including outdoors (Figure 2a).

In volumetric approaches, a geometric proxy of the performer is built using a visual hull [15] computed from foreground mattes extracted each camera image I_c using a chroma key or more sophisticated image segmentation algorithm. To compute the visual hull the capture volume is coarsely decimated into a set of voxels at locations $\mathscr{V} = \{V_1, \ldots, V_m\}$; a resolution of 1cm^3 is commonly used for a capture volume of approximately $6 \times 2 \times 6$ metres. The probability of the voxel being part of the performer in a given view c is:

$$p(V|c) = B(I_c(x[V_i], y[V_i])), \qquad (1)$$

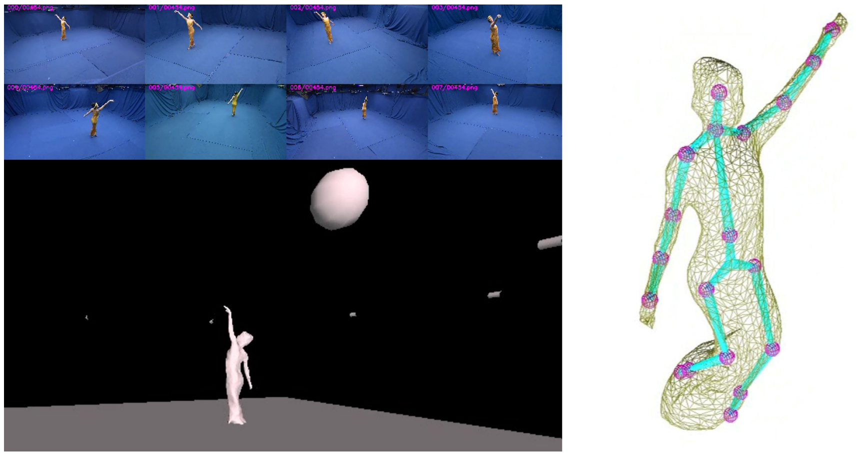


Fig. 3 4D Performance Capture: Multiple video views (top) are fused to create a volumetric representation of the performance which is meshed (bottom). The per-frame meshes are conformed to a single deforming mesh over time, into which a skeleton may be embedded and tracked (right).

where $B(.)$ is a simple blue dominance term derived from the RGB components of $I_c(x,y)$, i. e. $1 - \frac{B}{R+G+B}$, and (x,y) is the point within I_c that V_i projects to:

$$x[V_i] = \frac{f_c v_x}{v_z} + o_c^x \quad \text{and} \quad y[V_i] = \frac{f_c v_y}{v_z} + o_c^y, \quad \text{where,} \tag{2}$$

$$\begin{bmatrix} v_x \; v_y \; v_z \end{bmatrix} = COP_c - R_c^{-1} V_i. \tag{3}$$

The overall probability of occupancy for a given voxel $p(V)$ is:

$$p(V_i) = \prod_{i=1}^{C} 1/(1 + e^{p(V|c)}). \tag{4}$$

We compute $p(V_i)$ for all $V_i \in \mathscr{V}$ to create a volumetric representation of the performer for subsequent processing. An iso-contour extraction algorithm such as *marching cubes* [24] is used to extract a triangular mesh model from the voxel based visual hull (Figure 3). The result is a topologically independent 3D mesh for each frame of video. This can be converted into a so called '4D' representation using a mesh tracking process to conform these individual meshes to a single mesh that deforms over time [7]. Once obtained, it is trivial to mark up a single frame of the performance to embed a skeleton (e. g. marking each joint limb as an average of subsets of mesh vertices) and have the skeleton track with the performance as the mesh deforms. As we explain in subsec. 3.2, either the skeletal or surface representations from such a 4D performance capture may be used to drive character animation interactively.

CNNs have also been applied to volumetric approaches, with a spherical histogram (c.f. subsec. 3.2.2) derived from the visual hull being fed into a CNN to directly identify human pose [34]. The system contrasts with Elhayek et al. [13] where the CNN operates in 2D rather than 3D space, and similarly adds robustness to visual clutter in the scene.

3 Interactive Character Animation

Interactive character animation often takes place within complex digital environments, such as Games, in which multiple entities (characters, moveable objects, and static scene elements) interact continuously. Since these interactions are a function of user input they cannot be predicted or scripted *a priori*, and enumerating all possible eventualities is intractable. It is therefore necessary to plan animation in real-time using fast, online algorithms (i. e. algorithms using data from the current and previous timesteps only). Two distinct categories of algorithm exist.

First, algorithms drawing upon pre-supplied database of motion for the character, usually obtained via PC and/or manual scripting. Several fragments of motion data ('motion fragments') are stitched and blended together to create a seamless piece animation. A trivial example is a single cycle of a walk, which can be repeatedly concatenated to create a character walking forward in perpetuity. However more complex behaviour (e. g. walks along an arbitrary path) can be created by carefully selecting and interpolating between a set of motion fragments (for example three walk cycles, one veering left, one veering right, and one straight ahead) such that no jarring movement occurs. This form of motion synthesis, formed by concatenating (and in some cases interpolating between) several motion fragments is referred to as 'concatenative synthesis'. The challenge is therefore in selecting and sequencing appropriate motion fragments to react to planning requirements (move from A to B) under environmental (e. g. occlusion) and physical (e. g. kinematic) constraints. This is usually performed via a graph optimization process, with the motion fragments and valid transitions between these encoded in the nodes and edges of a directed graph referred to as a 'move tree' or 'motion graph' [21]. The key advantages of a motion graph are predictability of movement and artistic control over the motion fragments that are challenging to embody within a physical simulation. The disadvantage is that motion cannot generalise far beyond the motion fragments i. e. character movement obtained via PC in the studio. We discuss concatenative synthesis in detail within subsec. 3.2.

Second, algorithms that do not require pre-scripted or directly captured animation but instead simulate the movement under physical laws. Physics simulation is now commonly included within Games engines (e. g. Havoc, PhysX) but used primarily to determine motion of objects or particles, or animation of secondary characteristics such as cloth attached to characters [4]. Yet more recently, physics based character animation has been explored integrating such engines into the animation loop of principal characters [14]. Physics based simulation offers the significant advantage of generalisation; characters modelled in this manner can react to any situation with the virtual world and are not bound to a database on pre-ordained movements. Nevertheless, the high computational cost of simulation forces accuracy-performance trade-offs for real-time use. Simplifying assumptions such as articulated rigid bodies for skeletal structure are very common. It is therefore inaccurate to consider physically simulated animation as being more ‘natural’, indeed the tendency of simulation to produce ‘robotic’ movements lacking expressivity has limited practical uptake of these methods for interactive character animation until comparatively recently. We briefly discuss physics based character control in the next section, restricting discussion to the context of real-time animation for interactive applications. A detailed discussion of physics based character animation in a broader context can be found in Chapter C-2.

3.1 Real-time Physics based Simulation

Physically simulated characters are usually modelled as a single articulated structure of rigid limb components, inter-connected by two basic forms of joint mimicking anatomy in nature. Characters modelled under physical simulation are typically humanoid [18, 27], or animal [36] consisting predominantly of hinge joints, with hips and shoulders joints implemented as ball-socket joints. Depending on the purpose of the simulation, limbs may be amalgamated for computational efficiency (e. g. a single component for head, neck and torso) [33]. More complex simulations can include sliding joints in place of some hinge joints, that serve to model shock absorption within the ligaments of the leg [22].

3.1.1 Character Model Actuation

The essence of the physical simulation is to solve for the forces and torques that should be applied to each limb, in order to bring about a desired motion. This solve is performed by a ‘motion controller’ algorithm (subsec. 3.1.2). The locations at each limb where forces are to be applied is a further design consideration of the modeller. The most common strategy is to consider torque about each joint (degree of freedom), a method known as *servo actuation*. Whilst intuitive, servo actuation is not natural — effectively assuming each joint to contain a motor capable of rotating its counter-part. Careful motion planning is necessary to guard against unnatural motion arising under this simplified model.

Biologically inspired models include simulated muscles that actuate through tendons attached to limbs, effecting a torque upon the connected joints. Muscle actuated models are more challenging to design motion controllers for, since the maximum torque that can be applied by a muscle is limited by the turning moment of the limb which is dependent on the current pose of the model. Furthermore the number of degrees of freedom in such models tends to be higher than servo-actuated models, since muscles tend to operate in an antagonist manner with a pair of muscles per joint enabling ‘push’ and ‘pull’ about the joint. Moreover such models cannot be considered natural unless the tendons themselves are modelled as non-rigid structures, capable of stretch and compressing to store and release energy in the movement. The high computational complexity of motion controllers to solve for muscle actuated models therefore remains a barrier to their use in real-time character animation, whose applications to digital entertainment (rather than say, bio-mechanics) rarely require biologically accurate simulation. We therefore do not consider them further in this chapter.

3.1.2 Character Motion Control

Use cases for character animation rarely demand direct, fine-grain control of each degree of freedom in the model. Rather, character control is directed at a higher level e. g. 'move from A to B at a particular speed, in a particular style'. Such directives are issued by game AI, narrative or other higher level controllers. Motion controllers are therefore a mid-layer component in the control stack bridging the semantic gap between high-level control and low-level actuation parameters. In interactive scenarios, simple servo-based actuation (i. e. independent, direct control over joint torques) is adopted to ensure computation of the mapping is tractable in real-time.

Solving for the movement is performed iteratively, over many small time steps, each incorporating feedback supplied by the physics engine from each actuation of the model at the previous time-step under *closed-loop control*. This obviates the need to model complex outcomes of movements within the controller itself. Feedback comprises not only global torso position and orientation, but also individual joint orientation and velocity post-simulation of the movement. It is common for controllers to reason about the stability (balance) of the character when planning movement. The center of mass (COM) of the character should correspond to the zero-moment point (ZMP) i. e. the point at which the reaction force from the world surface results in a zero net moment. When the COM and ZMP coincide the model is stable.

We outline two common strategies to motion control that are applicable to physically based real-time interactive character animation.

Control in Joint-Space via Pose Graphs. Some of the earliest animation engines comprised carefully engineered software routines, to procedurally generate motion according to mechanics models embedded within kinematics solvers and key-framed poses. This approach is derived from real-time motion planning in the robotics literature.

Such approaches model the desired end-position of a limb (or limbs) as a 'key-pose'. Using a kinematics engine, the animation rig (i. e. joint angles) are gradually adjusted to bring the character's pose closer to that desired key-pose. The adjustment is an iterative process of actuation, and feedback from the environment to determine the actual position of the character and subsequent motions. For example, the COM and ZMP as well as the physical difference between the current and intended joint

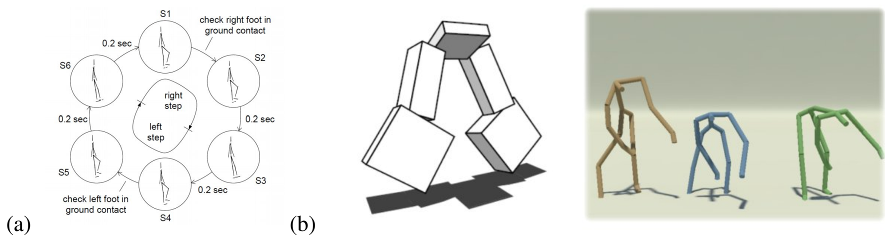

S1
check right foot in ground contact
0.2 sec
S6
S2
right step
left step
0.2 sec
0.2 sec
S5
S3
check left foot in ground contact
0.2 sec
S4
(a)
(b)

Fig. 4 Physically based interactive character animation. (a) Pose Space Graph used to drive high level goals for kinematics solvers which direct joint movements (from [23]); (b) Ambulatory motion of a creature and person learned by optimization processes mimicking nature (from [30] and [19]) respectively.

positions are monitored to ensure the intended motion does not unbalance the character unduly and that progress is not impeded by itself or other scene elements.

A sequence of such key-poses are defined within a 'Pose Space Graph' (PSG) , where the nodes in the graph are procedurally defined poses i. e. designed by the animator, but the movements between poses are solved using an inverse kinematics engine (IK). A motion such as a walk, is performed by transitioning through states in the PSG (Figure 4 illustrates a walk cycle in a PSG). Due to physical or timing constraints, a character often will not reach a desired pose within the PSG before being directed toward the next state. Indeed it is often unhelpful for the character to decelerate and pause (i. e. obtain ZMP of zero) and become stable at a given state before moving on the next; a degree of perpetual instability for example exists within the human walk cycle. Therefore key-poses in the PSG are often exaggerated on the expectation that the system will approximate rather than interpolate the key poses within it.

The operation of PSGs is somewhat analogous to motion graphs (c.f. subsec. 3.2.1), except that IK is used to plan motion under physical models, rather than pre-captured performance fragments concatenated and played back.

Control via Machine Learned Models. Although expensive to train, machine learning approaches offer a run-time efficiency unrivaled by other real-time motion controller strategies. Most commonly, neural networks (NN) are used to learn a mapping between high-level control parameters and low-level joint torques, rather than manually identified full body poses (subsec. 3.1.2). Notwithstanding design of the fitness function of the network and its overall architecture, the training process itself is fully automatic using a process trial-and-error via feedback from the physics simulation. A further appeal of such approaches is that such training is akin to the biological process of learned actuation in nature.

Networks usually adopt a shallow feed-forward network such as a multi-layer perceptron (MLP) [26], although the growing popularity of deeply learned networks has prompted some recent research into their use as motion controllers [19]. Training the MLP proceeds from an initially randomised (via Gaussian, or 'white' noise) set of network weights, using a function function derived from some success metric

– typically the duration that the controlled model can execute movement (e. g. walk) for without destabilising and falling over. Many thousands of networks (weight configurations) are evaluated to drive character locomotion, and the most successful networks and modified and explored further in an iterative optimization process to train the network [30].

Optimization of the NN weights is commonly performed by an evolutionary algorithm (EA) in which populations of network configurations (i. e. sets of weights) are evaluated in batches. The more successful configurations are propagated the subsequent batch, and spliced with other promising configurations, to produce batches of increasingly fitter networks [38]. In complex networks with many weights and complex movements, it can be challenging for EAs to converge during training. In such cases, weight configurations for the NN can be boot-strapped by training the same network over simpler problems. This improves up white noise initialisation for more complex tasks. In practice, training a NN can take tens of thousands of iterations to learn an acceptable controller [30] for even very simple movements. Yet, once learned the controller is trivial to evaluate quickly and can be readily deployed into a real-time system. Even with boot-strapped training however, NN cannot learn complex movement and it was not until the recent advent of more sophisticated (deeper) NNs that locomotion of a fully articulated body was demonstrated using an entirely machine-learned motion controller [19].

3.2 Concatenative Synthesis

Motion concatenation is a common method for synthesising interactive animation without the complexity and computational expense of physical simulation. In a *concatenative synthesis* pipeline, short fragments of motion capture are joined (and often blended) together to create a single seamless movement. In the simplest example, a single walk cycle may be repeated with appropriately chosen in-out points to create a perpetual gait. A more complex example may concatenate walk cycles turning slightly left, slightly right, or advancing straight-ahead to create locomotion along an arbitrary path.

3.2.1 Skeletal Motion Graphs

Concatenative synthesis is dependent on the ability to seamlessly join together pairs of pre-captured *motion fragments* – sub-sequences of performance capture – to build complex animations. An initial step when synthesising animation is therefore to identify the *transition points* within performance captured footage, at which motion fragments may be spliced together. Typically the entire capture (which may in practice consist of several movements e. g. walking, turning) is considered as a single long sequence of $t = [1, N]$ frames, and pairs of frames $\{1..N, 1..N\}$ identified that could be transitioned between to without the view perceiving a discontinuity.

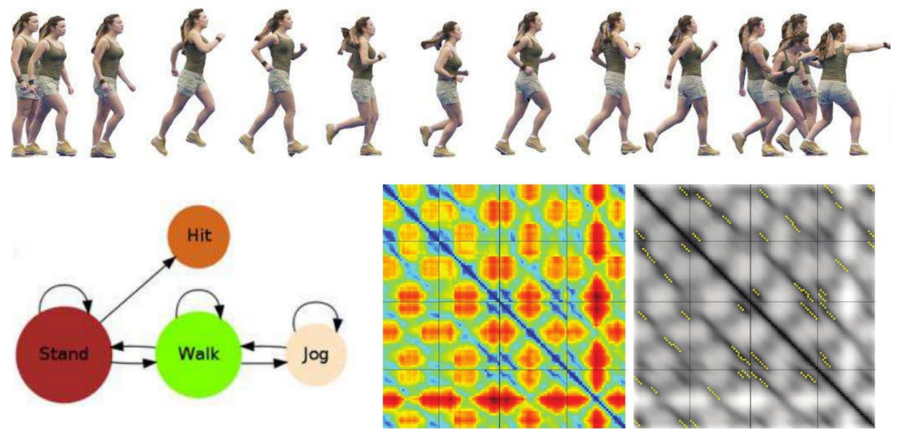


Fig. 5 An example of an animation (top) generated by a motion graph (left) comprising four actions. An visualisation of the inter-frame distance comparison used to compute a motion graph (right).

A measure of similarity is defined, computable from and to any time instant in the sequence, and that measure thresholded to identify all potential transition points. Figure 5 visualises both the concept, and an example of such a comparison computed exhaustively over all frames of a motion capture sequence — brighter cells indicating closer matching frame pairs.

Measures of pose similarity. Pose similarity measures (which, in practice, often compute the dissimilarity between frames) should exhibit three important properties:

1. Be invariant to global rotation and translation — similar poses should be identified as similar regardless of the subject's position in world-space at both time instants. Otherwise, few transition points will be detected.
2. Exhibit spatio-temporal consistency — poses should not only appear similar at the pair of time instants considered, but also move similarly. Otherwise, motion will appear discontinuous.
3. Reflect the importance of certain joints over others. Otherwise, a difference in position of e. g. a finger might outweigh a difference in position of a leg.

Common similarity measures include direct comparison of joint angles (in quaternion form), or more commonly, direct comparisons of limb spatial position in 3D. A set of 3D points $p_{1..m}$ is computed either from limb end-points, or from the vertices of a coarse mesh approximating the form of the model and a sum of squared differences used to evaluate the dissimilarity $D(p,p')$ between point sets from a pair of frames p and p' at times t and t' respectively:

$$D(p,p') = \min \theta, x_0, z_0 \sum_{i=1}^{m} \omega_i \left| p_i - M_{\theta,x_0,z_0} p'_i \right|^2 . \qquad (5)$$

Where $|.|$ is the Euclidean distance in world-space, p_i is the i^{th} point in the set, and M is a Euclidean transformation best aligning the two sets of point clouds, via a translation on the ground-plane (z_0, z_0) and a rotation θ about the vertical (y) axis — so satisfying property (1). In order to embed spatio-temporal coherency (2), the score is computed over point sets not just from a given pair of times $\{t, t'\}$ but for a k frame window $[t - \frac{k}{2}, t + \frac{k}{2}]$. This is effectively a low-pass filter over time, and explains the blurred appearance of Figure. 5 (right). For efficiency, pair-wise scores are computed and the resulting matrix low-pass filtered. The relative importance of each point (associated with the limb from which the point was derived) is set manually via ω_i satisfying property (3).

Motion Graph Construction. Local thresholding is applied to the resulting similarity matrix, identifying non-adjacent frames (t, t') that could be concatenated together to produce smooth transitions according to properties (1-3). For example, if the mocap sequence contains several cycles of a walk it is likely that corresponding points in the walk cycles (e. g. left foot down at the start of each cycle) would be identified as transitions. Playing one walk cycle to this time t, and then 'seeking' forward or backward by several hundred frames to the corresponding time t' in another walk cycle will not produce a visual discontinuity despite the non-linear temporal playback.

The 'in' (t) and 'out' (t') frames of these transition points are identified and represented as nodes in a graph structure (the *motion graph*). Edges in the graph correspond to clips of motion i. e. motion fragments between these frames in linear time. Additional edges are introduced to connect the 'in' and 'out' frames of each transition. Usually the pose of the performer at the 'in' and 'out' points differs slightly, and so this additional edge itself comprises a short sequence of frames constructed by interpolating the poses at 'in' and 'out' respectively e. g. using quarternion based joint angle interpolation.

Motion Path Optimization. Random walks over the motion graph representation can provide a useful visualisation to confirm that sensible transitions have been identified. However interactive character control requires careful planning of routes through the motion graph, to produce movement satisfying constraints the most fundamental of which are the desired end-pose (and position in the world, p_v), the distance that the character should walk (d_v), and the time it should take the character to get there (t_v). Under the motion graph representation, this corresponds to computing the optimal path routing us from the current frame of animation (i. e. the current motion capture frame being rendered) to a frame corresponding to the target key-pose elsewhere in the graph. Since motion graphs are often cyclic there are potentially unbounded number of possible paths. The optimal path is the one minimising a cost function, expressed in terms of these four animation constraints (C_{trans}, C_{time}, C_{dist} and $C_{spatial}$):

$$C(P) = C_{trans}(P) + \omega_{time} \cdot C_{time}(P) + \omega_{dist} \cdot C_{dist}(P) + \omega_{space} \cdot C_{space}(P). \quad (6)$$

Studying each of these terms in turn, the cost of a path P is influenced by C_{trans}; reflecting the cost of all performing animation transitions along the path P. Writing the sequence of N_f edges (motion fragments) along this path as $\{f_j\}$ where $j = \left[1, N_f\right]$ this cost is a summation of the cost of transitioning at each motion graph node along that path:

$$C_{trans}(P) = \sum_{j=1} N_f - 1 \mathscr{D}\left(f_j \mapsto f_{j+1}\right), \tag{7}$$

where $\mathscr{D}\left(f_j \mapsto f_{j+1}\right)$ expresses the cost of transitioning from the last frame of f_j to the first frame of f_{j+1}, computing by measuring the alignment of their respective point clouds p and p' via $D(p,p')$ (eq. 5)

The timing cost $C_{time}(P)$ is computed as the absolute difference between the target time t_v for the animation sequence, and the absolute time $time(P)$ taken to transition along the path P:

$$\begin{aligned} C_{time}(P) &= \left|time(P) - t_v\right|. \\ time(P) &= N_f \cdot \Delta t, \end{aligned} \tag{8}$$

where Δt is the time take to display a single frame of animation e. g. $\Delta t = \frac{1}{25}$ for 25 frames per second.

Similarly, the cost $C_{dist}(p)$ is computed as the absolute difference between the target distance d_v for the character to travel, and the absolute distance travelled $dist(P)$ computed by summing the distance travelled for each frame comprising P.

$$\begin{aligned} C_{dist}(P) &= \left|dist(P) - t_v\right|. \\ dist(P) &= \sum_{j=1} N_f - 1 \left|\mathscr{P}(f_j) - \mathscr{P}(f_{j+1})\right|, \end{aligned} \tag{9}$$

where $\mathscr{P}$ is a 2D projection operation, projecting the 3D points clouds p and p' corresponding to the end frame of f_j and start frame of f_{j+1} respectively to the 2D ground $(x - z)$ plane and computing the centroid.

The final cost $C_{spatial}$ is computed similarly via centroid projection of the animation end-point, penalising a large distance between the target end-point of the character and end-point arising from the animation described by P.

$$C_{space}(P) = \left|\mathscr{P}(f_{N_f-1}) - p_v\right|. \tag{10}$$

The three parameters ω_{time}, ω_{dist}, ω_{space} are normalising weights typical values of which are $\omega_{time} = 1/10, \omega_{dist} = 1/3, \omega_{space} = 1$ [3].

Finding the optimal path P^{opt} for a given set of constraints $C_{\ldots}$ is found by minimising the combined cost function $C(P)$ (eq. 6):

$$P^{opt} = \underset{P}{\operatorname{argmin}} C(P). \tag{11}$$

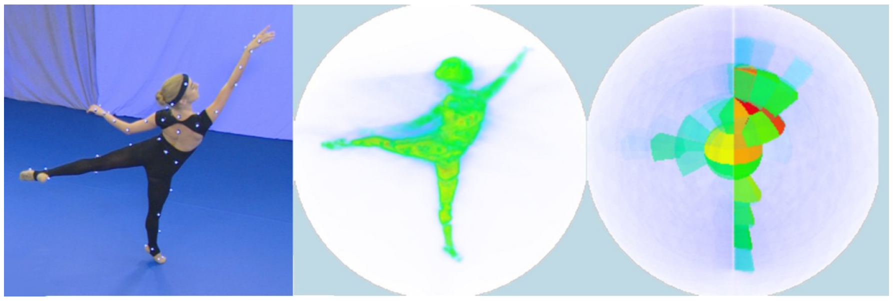


Fig. 6 Visualisation of a spherical histogram computed from a character volume. Multiple video views (left) are combined to produce a volumetric estimate of the character (middle) which is quantized into a spherical (long-lat) representation at multiple radii from the volume centroid.

An efficient approach using integer programming to search for the optimal path that best satisfies the animation constraints can be found in Huang et al.[20] and is capable of running in real-time for motion fragment datasets of several minutes. Note that C_{trans} is be pre-computed for all possible motion fragment pairs, enabling run-time efficiencies — the total transition cost for a candidate path P is simply summed during search.

3.2.2 Surface Motion Graphs

Surface motion graphs (SMGs) extend the skeletal motion graph concept beyond joint angles, to additionally consider the 3D volume of the performer. This is important since the movement of 3D surfaces attached to the skeleton (e. g. hair or flowing clothing) is often complex, and simple concatenation of pre-animated or captured motion fragments without considering the movement of this surface geometry can lead to visual discontinuities between motion fragments.

Consideration of surfaces, rather than skeletons, requires the motion graph pipeline to change only in two main areas. First, the definition of frame similarity i. e. eq. 5 must be modified to consider volume rather than joint positions. Second, the algorithm for interpolating similar frames to create smooth transitions must be substituted for a surface interpolation algorithm.

3D Shape Similarity. To construct a SMG, an alternative measure of frame similarity using 3D surface information is adopted, reflecting the same three desirable properties of similarity measures outlined in subsec. 3.2.1. A spherical histogram representation is calculated from the 3D character volume within the frame. The space local to the character's centroid is decimated into sub-volumes, divided by equispaced lines of longitude and latitude – so yielding a 2D array encoding the histogram that encodes the volume occupied by the character. Spherical histograms are computed over a variety of radii, as depicted in Figure 6 (right) yielding a three dimensional stack of 2D spherical histograms.

The SMG is computed as with skeletal motion graphs, through an optimization process that attempts to align each video to every other — resulting in a matrix of similarity measurements between frames. The similarity between the spherical histograms $H_r(.)$ at radius r of the 3D character meshes Q_a and Q_b is computed by:

$$D(Q_a, Q_b) = \min_{\phi} \frac{1}{R} \sum_{r=1}^{R} \omega_r \left| H_r(Q_a, 0) - H_r(Q_b, \phi) \right|, \tag{12}$$

where $H(x, \phi)$ indicates a spherical histogram computed over a given mesh x, rotated about the y axis (i. e. axis of longitude) by ϕ degrees. In practice this rotation can be performed by cycling the columns of the 2D histogram obviating any expensive geometric transformations; an exhaustive search across $\phi = [0, 359]$ degrees is recommended in [20]. The use of the model centroid, followed by optimization for ϕ fulfills property (1) i. e. rotational and translational invariance in the comparison. The resulting 2D matrix of inter-frame comparisons is low-pass filtered as before to introduce temporal coherence, satisfying property (2). Weightings set for each radial layer of the spherical histogram ω_r weight the importance of detail as distance increases, satisfying (3).

Transition Generation. Due to comparatively high number of degrees of freedom on a 3D surface, it is much more likely that the start and end frames of a pair of motion fragments f_j and f_{j+1} selected on an optimal path P^{opt} will not exactly match. To mitigate any visual discontinuities on playback, a short transition sequence is introduced to morph the former surface (S_j) into the latter (S_{j+1}). This transition sequence is substituted in for small number of frames (L) before and after the transition point. Writing this time interval $k = [-L, L]$, a blending weight $\alpha(k)$ is computed:

$$\alpha(k) = \frac{k + L}{2L}, \tag{13}$$

and a non-linear mesh blending algorithm (such as the Laplacian deformation scheme of [32]) is applied to blend $S_j \mapsto S_{j+1}$ weighted by $alpha(k)$.

3.2.3 Parametric Motion Graphs

Parametric motion graphs (PMG) extend classical motion graphs (subsec. 3.2.1) by considering not only the concatenation, but also the blending, of motion fragments to synthesise animation. A simple example is a captured sequence of a walk and a run cycle. By blending these two motion fragments together one can create a cycle of a walk, a jog, a run or anything in-between. Combined with the concatenation of cycles, this leads to a flexibility and storage efficiency not available via classic methods — a PMG requires only a single example of each kind of motion fragment, whereas a classical approach would require pre-captured fragments of walks and runs at several discrete speeds.

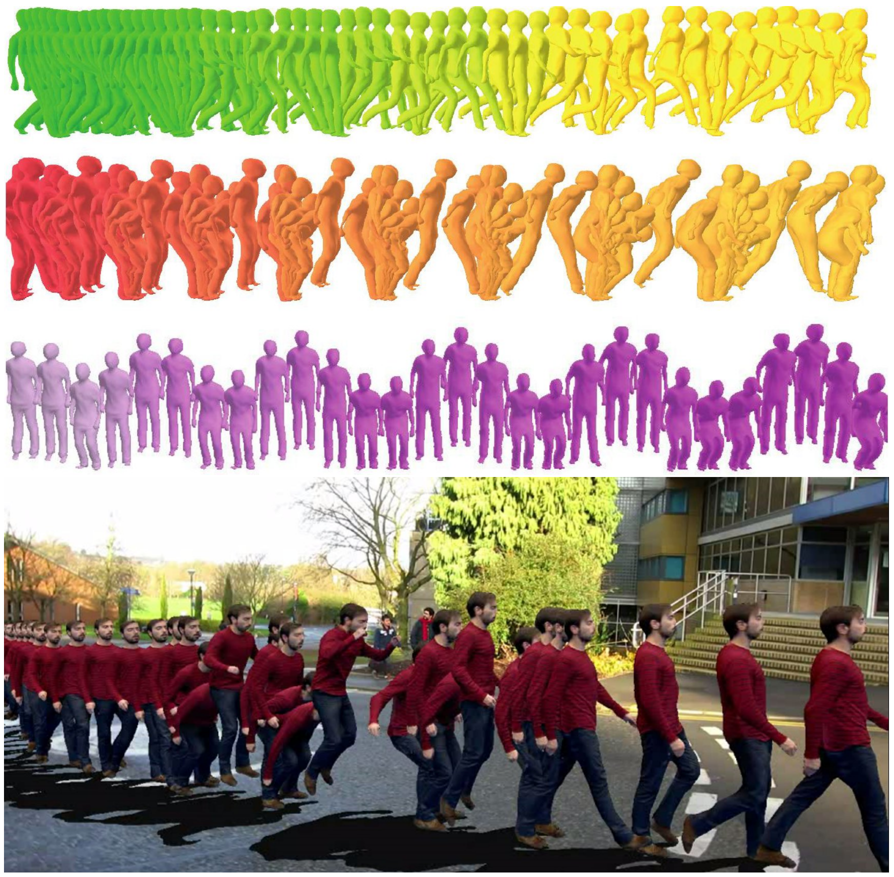


Fig. 7 Three animations each generated from a parametric motion graph [9]. First Row: Walk to Run (speed control). Second Row: Short to Long Horizontal Leap (distance i. e. leap length control). Third Row: Short to High Jump (height control). Final Row: Animation from a Parametric Motion Graph embedded within an outdoor scene under interactive user control [10] (time lapse view).

Parametric extensions have been applied to both skeletal [17] and surface motion graphs [9]. Provided a mechanism exists for interpolating a character model (joint angles, or 3D surface) between two frames, the method can be applied. Without loss of generality we consider surface motion graphs (SMGs) here.

SMGs assume the availability of 4D performance capture data; i. e. a single 3D mesh of constant topology deforming over time to create character motion (subsec. 2.2.2). We consider a set of N temporally aligned 4D mesh sequences $\mathscr{Q} = \{Q_i(t)\}$ for $i = [1,N]$ motion fragments. Since vertices are in correspondence, it is possible to interpolate frames from such sequences directly by interpolating vertex positions in linear or piecewise linear form. We define such an interpolating function $b(Q,w)$ yielding an interpolated mesh $Q_B(t,w)$ at time t given a vector of weights w expressing how much influence each of the meshes from the N motion fragments at time t should contribute to that interpolated mesh.

$$Q_B(t,w) = b(\mathscr{Q},w), \tag{14}$$

where $w = \{w_i\}$ for normalised weights $w \in [0,1]$ that drives a mesh blending function $b(.)$ capable of combining meshes at above 25 frames per second for interactive character animation.

Several steps are necessary to deliver parametric control of the motion fragments: time-warping to align pairs of mesh sequences (which may different in length) so that they can be meaningfully interpolated; the blending function $b(.)$ to perform the interpolation; and mapping between high level 'user' parameters from the motion controller to low level blend weights w. Considerations such as path planning through the graph remain, as with classical motion graphs, but must be extended since the solution space now includes arbitrary blendings of motion fragments, as well as the concatenated of those blended fragments. Exhaustively searching this solution space is expensive, motivating real-time methods to make PMG feasible for interactive character animation.

Mesh Sequence Alignment. Mesh sequences are aligned using a continuous time warping function $t = f(t_u)$ where the captured timebase t_u is mapped in non-linear fashion to a normalised range $t = [0,1]$ so as to align poses. The technique is described in Witkin and Popovic [37]. Although coarse results are obtainable without mesh alignment, failure to properly align sequences can lead to artifacts such as foot skate.

Real-time Mesh Blending. Several interpolation schemes can be employed to blend a pair of captured poses. Assuming a single mesh has been deformed to track throughout the 4D performance capture source data (i. e. all frames have constant topology) a simple linear interpolation between 3D positions of corresponding vertices is a good first approximation to a mesh blend. Particularly in the presence of rotation, such approximations yield unrealistic results. A high quality solution is to use differential coordinates i. e. a Laplacian mesh blend [5] however solution of the linear system (comprising a $3v \times 3v$ matrix of vertex positions, where v is of the order 10^5 is currently impractical for interactive animation. Therefore a good compromise can be obtained using a piece-wise linear interpolation [9], which precomputes offline a set of non-linear interpolated meshes (e. g. via [5]) and any requested parametric mesh is produced by weighted linear blending of the closest two precomputed meshes. The solution produces more realistic output, in general, that linear interpolation at the same computational cost.

High-level Control. High-level parametric control is achieved by learning a mapping function $f(w)$ between the blend weights w and the high-level motion parameters p e. g. from the motion controller. A mapping function $w = f^{-1}(p)$ is learned from the high-level parameter to the blend weights required to generate the desired motion. This is necessary as the blend weights w do not provide an intuitive parame-

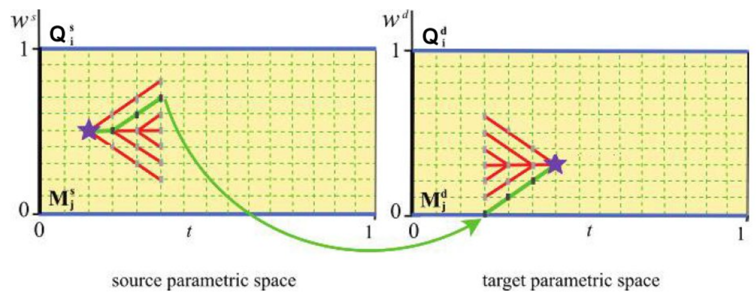


Fig. 8 Real-time motion planning under a parametric motion graph. Routes are identified between trellises fanned out from the source pose Q^s and end pose Q^d. The possible (red) and optimal (green) paths are indicated (illustration only).

terisation of the motion. Motion parameters p are high-level user specified controls for a particular class of motions such as speed and direction for walk or run, and height and distance for a jump. The inverse mapping function f^{-1} from parameters to weights can be constructed by a discrete sampling of the weight space w and evaluation of the corresponding motion parameters p.

Parametric Motion Planning. PMGs dispense of the notion of pre-computed transition points, since offline computation of all possible transition and blend possibilities between e. g. a pair of mesh sequences would yield an impractical number of permutations to permit real-time path finding.

We consider instead a continuous 'weight-time' space with the weight modelling the blend between one mesh sequence (e. g. a walk) and another (e. g. a run). We consider motion planning as the problem of finding a route through this space, taking us from a source time and pose (i. e. weight combination) to a target time and pose. Fig. 8) illustrates such a route finding process. The requirement for smooth motion dictates we may only modify the weight or time in small steps, yielding to a 'fanning out' or *trellis* of possible paths from the source point in weight-time space. The optimal path P^{opt} between two parametric points in that space is that minimising a cost function balancing mesh similarity $E_S(P)$ and time taken i. e. latency $E_L(P)$ to reach that pose:

$$P^{opt} = \underset{P \in \Omega}{\operatorname{argmin}} E_S(P) + \lambda E_L(P), \qquad (15)$$

where λ defines the trade-off between transition similarity and latency. The transition path P is optimized over a trellis of frames as in Fig. 7 starting at frame $Q^s(t^s, w^s)$ ending at $Q^d(t^d, w^d)$ where Q^s and Q^d are interpolated meshes (eq. 14).

The trellis is sampled forward and backward in time at discrete intervals in time Δt and parameters Δw up to a threshold depth in the weight-time space. This defines a set of candidate paths $\mathscr{P}$ comprising the transitions between each possible pair of frames in the source and target trellis.

For a candidate path P, the latency cost $E_L(P)$ is measured as the number of frames in the path P between the source and target frames. The transition similarity cost $E_S(P)$ is measured as the similarity in mesh shape and motion at the transition point between the source and target motion space for the path P, computable via eq. 12 for mesh data (or if using purely skeletal mocap data, via eq. 5). Casas et al. [8] proposed a method based on precomputing a set of similarities between the input data, and interpolating these at runtime to solve routing between the two trellis at interactive speeds. Figure 7 provides examples of animation generated under this parametric framework.

4 Summary and Future Directions

This chapter has surveyed techniques for interactive character animation, broadly categorising these as either data-driven or physical simulation based. Arguably the major use cases for interactive character animation are video games and immersive virtual experiences (VR/AR). In these domains, computational power is at a premium — developers must seek highly efficient real-time algorithms, maintaining high frame-rates (especially for VR/AR) without compromising on animation quality. This has led interactive animation to trend toward data-driven techniques that sample, blend and concatenate fragments of performance capture rather than spend cycles performing expensive online physical simulations.

This chapter has therefore focused upon synthesis techniques that sample, concatenate and blend motion fragments to create animation. The chapter began by surveying commercial technologies and academic research into performance capture. Although commercial systems predominantly focus upon skeletal motion capture, research in 4D performance capture is maturing toward practical solutions for simultaneous capture of skeletal and surface detail. The discussion of *motion graphs* focused upon the their original use for skeletal data, and their more recent extensions to support not only 4D surface capture but also their parametric variants that enable blending of sequences in addition to their concatenation. Physical simulation based approaches for character animation were examined within the context of interactive animation, deferring broader discussion of this topic to chapter C-2.

Open challenges remain for interactive character animation, particularly around expressivity and artistic control. Artistic directors will often request editting of animation to move in a particular style (jaunty, sad); adjustments that can be performed manually by professional tools such as Maya (Autodesk) or MotionBuilder (Vicon), but that cannot be applied automatically in an interactive character animation engine. Whilst work such as Brand et al.'s Style Machines [6] enables stylisation of standalone skeletal mocap sequences, algorithms have yet to deliver the ability to modulate animation interactively e. g. to react to emotional context in a game. An interesting direction for future research would be to integrate stylisation and other high level behavioural attributes into the motion graph optimization process.

References

1. Agarwal, A., Triggs, B.: Recovering 3d human pose from monocular images. IEEE Trans. PAMI **28**(1), 44–58 (2006)
2. Andriluka, M., Roth, S., Schiele, B.: Pictoral structures revisited: People detection and articulated pose estimation. In: Proc. Computer Vision and Pattern Recognition (2009)
3. Arikan, O., Forsyth, D.: Synthesizing constrained motions from examples. ACM Trans. on Graphics (2002)
4. Armstrong, W., Green, M.: The dynamics of articulated rigid bodies for purposes of animation. The Visual Computer **4**(1), 231–240 (1985)
5. Botsch, M., Sorkine, O.: On linear variational surface deformation methods. IEEE Transactions on Visualization and Computer Graphics **14**(1), 213–230 (2008)
6. Brand, M., Hertzmann, A.: Style machines. In: Proc. ACM SIGGRAPH, pp. 183–192 (2000)
7. Budd, C., Huang, P., Klaudiny, M., Hilton, A.: Global non-rigid alignment of surface sequences. International Journal of Computer Vision **102**(1), 256–270 (2013)
8. Casas, D., Tejera, M., Guillemaut, J.Y., Hilton, A.: 4d parametric motion graphs for interactive animation. In: Proc. Symp. on Interactive 3D Graphics and Games (I3D) (2012)
9. Casas, D., Tejera, M., Guillemaut, J.Y., Hilton, A.: Interactive animation of 4d performance capture. IEEE Trans. Visualization and Comp. Graphics (TVCG) **19**(5), 762773 (2013)
10. Casas, D., Volino, M., Collomosse, J., Hilton, A.: 4d video textures for interactive character appearance. Computer Graphics Forum (Proc. Eurographics 2014) (2014)
11. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detetion. In: Proc. Computer Vision and Pattern Recognition, vol. 3, pp. 886–893 (2005)
12. Eichner, M., Ferrari, V.: Better appearance models for pictorial structures. In: Proc. British Machine Vision Conf. (BMVC) (2009)
13. Elhayek, A., Aguiar, E., Tompson, J., Jain, A., Pishchulin, L., Andriluka, M., Bregler, C., Schiele, B., Theobalt, C.: Efficient convnet-based marker-less motion capture in general scenes with a low number of cameras. In: Proc. Computer Vision and Pattern Recognition (2015)
14. Geijtenbeek, T., Bogert, A.J.V.D., Basten, B., Egges, A.: Evaluating the physical realism of character animations using musculoskeletal models. In: Proc. Conf. on Motion in Games, vol. 5, pp. 11–22. Springer-Verlag (2010)
15. Grauman, K., Shakhnarovich, G., Darrell, T.: A bayesian approach to image-based visual hull reconstruction. In: Proc. CVPR (2003)
16. H. Ning W. Xu, Y.G., Huang, T.: Discriminative learning of visual words for 3D human pose estimation. In: Proc. CVPR (2008)
17. Heck, R., Gleicher, M.: Parametric motion graphs. In: Proc. Symp. on Interactive 3D Graphics and Games (I3D), pp. 129–136 (2007)
18. Hodgins, J.: Biped gait transition. In: Proc. Conf. on Robotics and Automation, pp. 2092–2097 (1991)
19. Holden, D., Saito, J., Komura, T.: A deep learning framework for character motion synthesis and editing. In: Proceedings of ACM SIGGRAPH (2016)
20. Huang, P., Hilton, A., Starck, J.: Human motion synthesis from 3d video. In: Proc. CVPR (2009)
21. Kovar, L., Gleicher, M., Pighin, F.: Motion graphs. ACM Trans. Graph. **21**(3), 473–482 (2002)
22. Kwon, T., Hodgins, J.: Control systems for human running using an inverted pendulum model and a reference motion capture sequence. In: Proc. Eurographics Symp. on Computer Animation (SCA) (2010)
23. Laszlo, J., Fiume, E.V.D.: Limit cycle control and its application to the animation of balancing and walking. ACM Trans. on Graphics (1996)
24. Lorensen, W., Cline, H.: Marching cubes: A high resolution 3d surface construction algorithm. ACM Computer Graphics **21**(4), 163–169 (1987)
25. Mori, G., Ren, X., Efros, A., Malik, J.: Recovering human body configurations: Combining segmentation and recognition. In: Proc. Computer Vision and Pattern Recognition, pp. 326–333 (2004)

26. Pollack, J., Lipson, H., Ficici, S., Funes, P.: Evolutionary techniques in physical robotics. In: Proc. Intl. Conf. Evolvable Systems (ICES). Springer-Verlag (2000)
27. Raibert, M., Hodjins, K.: Animation of dynamic legged locomotion. ACM Computer Graphics **25**(4), 349–358 (1991)
28. Ren, X., Berg, E., Malik, J.: Recovering human body configurations using pairwise constraints between parts. In: Proc. Intl. Conf. on Computer Vision, vol. 1, pp. 824–831 (2005)
29. Rohan, A.: 3D motion capture system market – global forecast to 2020. Tech. rep., Markets and Markets Inc., Vancouver (2015)
30. Sims, K.: Evolving virtual creatures. ACM Trans. on Graphics (1994)
31. Srinivasan, P., Shi, J.: Bottom-up recognition and parsing of the human body. In: Proc. Computer Vision and Pattern Recognition, pp. 1–8 (2007)
32. Tejera, M., Casas, D., Hilton, A.: Animation control of surface motion capture. ACM Trans. on Graphics pp. 1532–1545 (2013)
33. Tin, K., Coros, S., Beaudoin, P.: Continuation methods for adapting simulated skills. ACM Trans. on Graphics **27**(3) (2008)
34. Trumble, M., Gilbert, A., Hilton, A., Collomosse, J.: Learning markerless human pose estimation from multiple viewpoint video. In: Proc. ECCV Workshops (2016)
35. Viola, P., Jones, M.: Robust real-time object detection. Intl. Journal of Computer Vision **2**(57), 137–154 (2004)
36. Wampler, K., Popovic, Z.: Optimal gait and form for animal locomotion. ACM Trans. on Graphics **28**(3) (2009)
37. Witkin, A., Popovic, Z.: Motion warping. ACM Trans. on Graphics pp. 105–108 (1995)
38. Yao, X.: Evolving artificial neural networks. In: Proc. of the IEEE, vol. 87 (1999)
39. Zhao, T., Nevatia, R.: Bayesian human segmentation in crowded situations. In: Proc. Computer Vision and Pattern Recognition, vol. 2, pp. 459–466 (2003)