

Diss. ETH No. 25792

Intuitive Control of Animated Scenes



A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

LOIC FLORIAN CICCONE

Diplôme d'ingénieur, Grenoble INP Ensimag, France

Born on 10.04.1990

Citizen of France

accepted on the recommendation of

Prof. Dr. Robert W. Sumner, examiner

Prof. Dr. Markus Gross, co-examiner

Prof. Dr. Daniel Sýkora, co-examiner

2019

Abstract

Animation is a fundamental means of human expression that can dissolve the boundaries of reality and bring imaginative characters to life in visual form. It remarkably combines Arts for the creation of compelling shapes and movements, and Technology for the incorporation of automation and interaction into design paradigms.

The transition from hand-drawn 2D animation to 3D computer animation has introduced new tools and creative possibilities. It has yielded different rendering styles, physical simulations, and faster accomplishment of some tasks. However, it completely changed the workflow and the modes of interaction for crafting an animation, sometimes at the expense of the freedom of creation.

Current tools used by digital artists to author 3D content are very powerful; they allow the creation of very compelling animations and provide a high level of control over the final result. However, some interaction metaphors provided by such software do not accommodate specific tasks that users need to accomplish. This leads to elements of the artist's workflow that are achievable in a tedious, repetitive and/or unnatural way. As a result, current 3D tools require a lot of practice to be mastered. To this day, a novice user has no way to create a compelling character animation without prior intensive training. This limits drastically the range of people who can express their creativity through animated features.

The overall goal of this thesis involves the study and understanding of intuitive, user-centric design processes for computer animation as well as the application of these ideas to create intuitive interaction techniques and expand the state of the art in a way that focuses on user-oriented systems. In particular, this thesis explores new interaction interfaces for three prominent aspects of computer animation: environment design, movement crafting, and direct high-level control of character motion. We demonstrate that the introduction of more intuitive and natural animation tools has the potential to both facilitate the work of professional artists and make the computer animation technologies accessible to novice users.

Résumé

L'animation est un moyen d'expression fondamental pour l'Homme, qui lui permet de dépasser les frontières de la réalité et de donner vie à des personnages imaginaires sous forme visuelle. Elle combine de manière élégante l'Art pour le design de formes et mouvements expressifs et la Technologie pour la capacité de concevoir et interagir avec les éléments virtuels.

La transition de l'animation 2D traditionnelle à l'animation 3D par ordinateur a introduit de nouveaux outils et de nouvelles possibilités créatives. Des styles de rendu inédits, des simulations physiques et l'accélération de certaines tâches ont ainsi pu voir le jour. Cependant, cette transition a complètement modifié le processus d'animation et les moyens d'interaction avec la scène virtuelle, parfois au prix de la flexibilité de création.

Les outils actuels utilisés par les artistes pour produire du contenu 3D sont très puissants. Ils permettent la création d'animations très convaincantes et fournissent un contrôle très précis sur le résultat final. Cependant, certains moyens d'interaction fournis par ces logiciels ne sont pas adaptés à des actions spécifiques que les utilisateurs doivent accomplir. Cela conduit à des tâches qui sont fastidieuses et répétitives. Par conséquent, les logiciels 3D actuels demandent beaucoup de pratique pour être maîtrisés. À ce jour, un utilisateur débutant n'a aucun moyen de créer une animation de personnage convaincante sans formation intensive préalable. Cela limite considérablement le nombre de personnes pouvant exprimer leur créativité au travers des films d'animation.

L'objectif de cette thèse est l'étude et la compréhension d'outils de conception naturels pour l'animation par ordinateur, ainsi que l'application de ces idées pour créer des techniques d'interaction intuitives et étendre l'état de l'art de manière à se concentrer sur l'expérience utilisateur. En particulier, cette thèse explore de nouvelles interfaces intuitives pour trois aspects importants de l'animation par ordinateur : la conception de l'environnement, la création de l'animation et le contrôle en temps réel de personnages. Dans cette thèse, nous démontrons que de proposer des outils d'animation plus intuitifs et naturels a le potentiel de faciliter le travail des artistes professionnels et de rendre les technologies d'animation par ordinateur plus accessibles aux apprentis.

Acknowledgments

In 2015, Prof. Robert W. Sumner offered me the privilege of being his PhD student. During the following three and a half years, as I was striving to prove myself worthy of this honor, he continuously provided pertinent guidance to make this thesis advance in the right direction, while always taking into consideration my opinions, aspirations and competences. Bob's altruism and his contagious joie de vivre made this PhD a pleasant experience. I furthermore had the chance to evolve in a remarkably inspiring environment, thanks to Prof. Markus Gross who welcomed me with open arms in Disney Research Zurich. I benefited from his constant efforts for making this lab, as well as the Computer Graphics Laboratory, enjoyable working places.

Additional professors played a key role in the success of this PhD. I particularly want to express gratitude to Prof. Marie-Paule Cani, my mentor during my Master studies, who provided decisive support and advice to propel me to that doctorate position. I am also greatly thankful to my third committee member, Daniel Sýkora; even though he never met me in the past, he kindly accepted to take the time to examine my thesis, attend my defense and evaluate my work.

Doing a PhD is not always a happy time. There are of course the gratifying paper acceptances, the enjoyable technical conferences and the diverse social events. But there are also the frustrating code bugs, the difficult deadlines and the working weeks way beyond the usual 40 hours. In those cases, family's and friends' encouragements are essential for keeping focus, and colleagues' presence is crucial for staying motivated. For that, I would like to thank my parents, my sister, Sabrina Lachal, Maxime Lucht, Chappuis family, Christian Schumacher, Derek Bradley, Pascal Bérard, Fabio Zünd, Julia Chatain, Alessia Marra, Ming Zheng and Romain Prévost, just to name a few, for their strong support. Special thanks as well to Antoine Milliez who, more than being a colleague and a friend, was also a mentor. With his few extra years of experience, he always generously gave sincere advice to guide me on the right path.

In the middle of my PhD, I had the invaluable chance of doing a three months internship in Walt Disney Animation Studios. Despite being so short, this professional experience was certainly the most inspiring one of my life. I am very grateful to everyone who made this possible, including my supervisors Dmitriy

Pinskiy and Ricky Arietta, and to all the artists and scientists there who took the time to discuss with me. Talking about influential artists, I also value very high Maurizio Nitti's constant dedication to assist me with the artistic aspects of the projects, both for the design of user experiences and the production of results.



Last but not least, this project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 642841. This is a formal way to say that none of this would have been possible without the financial support provided by DISTRO, a European network. It was a great pleasure to meet once or twice a year all people from that consortium, who formed together a lovely and uplifting community.

Contents

Abstract	iii
Résumé	v
Acknowledgements	vii
Contents	ix
Introduction	1
1.1 Animation Legacy	1
1.2 Artistic Principles	2
1.3 3D Character Representation	3
1.4 Authoring the Motion	4
1.5 Real-time Animation	6
1.6 Publications	7
Related Work	9
2.1 Objects Deformation	9
2.1.1 Deformation of Single Objects	9
2.1.2 Deformation of Multiple Objects	10
2.2 Animation Crafting	11
2.2.1 Space vs Time	11
2.2.2 Space-time Curves	12
2.2.3 Performance Animation	13
2.3 High-level Character Control	13
2.3.1 Space-time Constraints	13
2.3.2 Real-time Manipulation	14
Flow Curves: an Intuitive Interface for Coherent Scene Deformation	17
3.1 Background	19
3.2 Overview	20
3.3 Subjective Curve Elements	21
3.3.1 Principal Curves	22
3.3.2 Abstract Contours	23

Contents

3.4	Flow Curves	25
3.4.1	Flow Curve from Sketched Stroke	25
3.4.2	Flow Curves Network from Center-Point	26
3.5	Direct Deformation of Scene Objects	28
3.5.1	2D Grid Embedding	29
3.5.2	Automatic Correspondence	29
3.5.3	Deformation	30
3.6	Results and Discussion	31
3.7	Conclusion	34
Tangent-Space Optimization for Interactive Animation Control		37
4.1	Background	40
4.2	Approach	40
4.2.1	Problem Formulation	40
4.2.2	Tangent Space Optimization	43
4.2.3	Implementation Details	46
4.2.4	Timing Manipulations	48
4.2.5	Static Case	49
4.3	Evaluation	50
4.3.1	Examples of Authoring Difficult Animations	50
4.3.2	User study: Simpler Curves for Complex Motions	52
4.3.3	User Study: Faster Editing Process	52
4.3.4	Qualitative Assessment from Professional Animators	54
4.3.5	System Performance	55
4.4	Conclusion	55
Authoring Motion Cycles		59
5.1	Background	61
5.2	Overview and Workflow	62
5.3	Cycle Specification	63
5.4	MoCurves	65
5.4.1	Spatial Manipulations	66
5.4.2	Temporal Manipulations	67
5.4.3	Contacts	68
5.5	Results	70
5.6	Conclusion	73
PuppetPhone: Puppeteering Virtual Characters Using a Smartphone		75
6.1	Background	77
6.2	Approach	77
6.2.1	MotionStick	77
6.2.2	State Machine	78

6.2.3	Animation Blending	79
6.3	Application: Build a Snowman	82
6.4	Application: Multi-Reality Games	85
6.5	Conclusion	88
Conclusion		91
7.1	Summary of Contributions	92
7.1.1	Coherent Scene Deformation	92
7.1.2	Motion Visualization and Manipulation	92
7.1.3	Performance Animation for Motion Cycles	93
7.1.4	Interactive Character Control	93
7.2	Future Work	94
7.2.1	Animating with Curves	94
7.2.2	Accommodating New Technologies	95
Attribute limits during interpolations		97
References		99

C H A P T E R

1

Introduction

1.1 Animation Legacy

Traditional hand-drawn animation provides a practical means to tell stories. By solely requiring a pencil and papers, it enables artists to fully express their creativity. This media, born almost two centuries ago, has a singular way of breathing life into virtual characters, and creating stories where the only limit is the creator's imagination.

Creating animated features, such as the acclaimed Disney classics, is nonetheless a long and fastidious process that requires a lot of manual work. In order to improve the efficiency of their workflow, professional artists had to be inventive and come up with new techniques. For example, *layering* was invented to separate a fixed background from moving characters and therefore avoid re-drawings. They also introduced *keyframing* to carefully plan the trajectory and timing of an action. There, instead of drawing frames in the chronological order, animators first draw the main states of the motion, called *keyframes*, and then complete the gaps with interpolation.

Even with these enhancements, creating an animation is laborious as it requires drawing multiple frames of motion per second. Therefore, artists are constrained to stick to simple scenes and lighting conditions. Complex effects such as hair and liquid simulations are simplified as much as possible. This limits animators in their means of expression.

The introduction of 3D computer animation in the 1980s has opened up new possibilities and challenges for animated content. Since then, a tremendous

amount of research have been accomplished, allowing physical simulations of complex systems, very realistic or stylized renderings, diverse mechanisms for character control, etc. This growth was accompanied by many applications, becoming an essential component in the movie industry, video games and even medicine.

Not only the creative possibilities changed, but also the means: no more pencil and paper, but a mouse and a computer; no more sketches and lines, but triangles in a 3D space. It is a whole new system that was established. But despite its differences, 3D animation strongly inherits from traditional 2D animation. Movements are again composed of several frames per second, they obey to the same aesthetic principles, and animators continue to use keyframing as the main method to animate characters.

However, changing the means while conserving the principles can yield inappropriate workflows. As a result, some tasks that were easy and natural with hand-drawn animation become challenging and unintuitive with computers. Nowadays, creating a compelling animation in 3D is a task that requires an intensive training, and therefore has a drastically limited accessibility. In this thesis, we propose to make simple things easy again. We develop user-centric tools and algorithms to make some tasks of 3D animation more natural to professional artists and more accessible to novice users.

1.2 Artistic Principles

Enhancing the workflow of 3D computer animation starts by understanding the basis on which it rests. For that reason, we are particularly interested in the 12 principles of animation introduced by Ollie Johnston and Frank Thomas in their book *The Illusion of Life* [Thomas and Johnston, 1981]. There, they describe how to produce an illusion of characters adhering to the basic laws of physics, as well as conveying emotion and appeal. Additionally, the principles of visual arts [Lidwell et al., 2003] reflect the different aspects of the human visual system and serve as guidelines for shaping scenes into aesthetically pleasing images.

Though originally intended to apply to traditional hand-drawn animation, these principles still have great relevance for today's more prevalent computer animation. Reviewing those aesthetic principles and best-practices developed in the context of classical art, illustration and hand-drawn animation permits to better understand the core factors at play in the field of aesthetic design. It appears that the freedom offered by traditional 2D sketch-

ing for crafting shapes and applying these visual principles is not reached by current 3D interfaces.

A particular example of this is the principle of *movement*, consisting in directing the viewer's eye as it flows through the screen. To achieve this principle, artists use our tendency to pick up contrast edges across multiple objects and join them together into longer imaginary *subjective curves* that our eyes unconsciously follow. Hence, by carefully shaping and coordinating contrast edges across objects in a scene, visual artists are capable of bringing our attention to a specific area. While traditional 2D sketching is a natural fit for this task, current 3D tools are object-centric and do not accommodate coherent deformation of multiple shapes into smooth flows.

In Chapter 3, we address this shortcoming by providing an intuitive interface to craft such whole scene deformations in a fast, easy and natural way. As traditional 2D sketching offers the freedom to naturally design the flow of a scene, we choose to adopt a sketch-based approach, that we call *Flow Curves*.

1.3 3D Character Representation

In 3D animation packages, geometries are defined by a multitude of vertices connected together by quads or triangles. Moving each vertex one by one in order to define character movements would be extremely cumbersome, which is why artists use *rigging*. Rigging is what defines the set of possible transformations for a character and help its manipulation (Fig. 1.1). A rig is essentially a digital skeleton, composed of joints and bones, bound to the 3D mesh. Similar to real humans' anatomy, moving a bone of the digital character moves the attached vertices, like a skin.

The rig then enables posing the character, which can be done in two different ways. *Forward kinematics* (FK) consists in setting the right joints orientations to obtain the desired pose, while *inverse kinematics* (IK) consists in automatically computing the rotations of a chain of joints that satisfy the positional constraint of an end-effector. The latter is particularly useful for specifying contacts, such as placing a hand on an object. This problem has been widely studied in computer graphics and robotics research, as elaborated on in the survey by Aristidou et al. [2017]. However, since IK is heavy and makes the pose evaluation slower, characters typically contain a limited set of available IK chains: usually on the arms, legs and spine.

Furthermore, in production environments, character rigs can become much more complex. Their skeleton is equipped with different kinds of handles

Introduction



Figure 1.1: *Left: a 3D mesh represents the geometry of the character. Middle: a digital skeleton, reproducing real bones' behavior, coordinately drives the vertices. Right: Rig controllers allow manipulating the bones and other deformers.*

to manipulate it in various ways (mainly FK and IK). Also, additional kinds of deformers are provided to adjust the geometry in more detail, such as for facial expressions. In total, a human character can easily contain more than 100 rig controllers, representing over 300 degrees of freedom.

1.4 Authoring the Motion

Making a character move consists in defining a pose for each frame of the animation (usually 24 per second). As mentioned earlier, 3D artists usually use keyframing for this task. It is one of the 12 principles of animation, also called *pose to pose*. In this practice, animators use the rig controllers to pose the character at different points in time, and the in-between states are automatically generated by interpolation. In principle, this represents a huge gain compared to hand drawn animation: where a secondary animator would often be the one manually drawing all the in-between frames, under the instructions of the primary animator, the computer is now directly and automatically interpolating keyframes. In practice however, the generated interpolations barely ever meet the animator's expectations, which necessitate further laborious work.

Indeed, the interpolation algorithm does not take into consideration any instruction from the artist, and naively applies a Bezier interpolation on every parameter of the character’s rig — generating what we call *animation curves*. To modify in-betweens, the user has to manually edit the tangents of interpolation of every animation curve (as a reminder, there are hundreds of them). It is such a cumbersome and unintuitive process that animators will almost always prefer to add more keyframes, often ending up with keys at every couple of frames.

Furthermore, this process is frustrating in the sense that it offers no intuition of timing. The temporal separation of keyframes is defined by their placement on a timeline, which makes it difficult to evaluate if the animation is too slow or too fast. The artist thus plays the animation and applies modifications in a trial and error fashion. In addition, animating requires continuously switching between at least three different interfaces: the viewport for character posing, the timeline for keyframes placement, and the graph editor for animation curves manipulation (see Fig. 1.2).

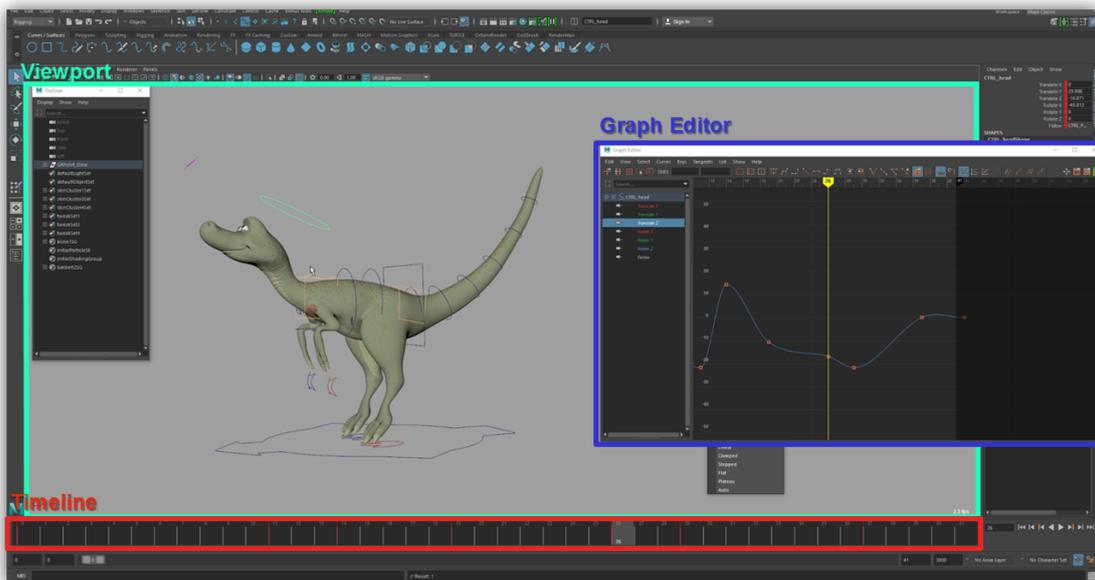


Figure 1.2: Example of a general purpose animation tool (Autodesk Maya). We highlight the minimum three interfaces that are essential to animate a character: the viewport, the timeline and the graph editor.

We propose to alleviate these limitations in Chapter 4 by introducing a tangent-space optimization that provides a natural and interactive control on interpolations. Our system presents a space-time curve representation to manipulate the movement of any part of the character, and instead of

adding keyframes it optimizes for the tangents of interpolation of the rig parameters that satisfy user constraints. This approach is non-intrusive to the animators' workflow and enables them to animate using fewer keyframes. Furthermore, our method does not require any predefined IK chain, which fastens the animation process, provides more freedom to the artist and abstracts technical considerations such as hierarchy and FK vs IK.

To animate a character, an alternative to keyframing is *performance animation*. It consists in acting out the motion using a capture device — e.g. a full-body motion capture suit — and the performance is directly transferred into the virtual character. However, this technique is rarely used for animated features because of the lack of control it provides. The action has to be acted out in real time, making difficult the performance of stylized effects, and the movement is registered as one keyframe per frame which yields complex animation curves that are very difficult to modify with traditional tools. Besides, it requires dedicated devices and is generally not appropriate for non-human characters because mimicking the movement of an animal is challenging.

In this thesis, we notice that performance animation can be especially appropriate for motion cycles since repeatedly acting out a motion allows to progressively improve it. Moreover, motion cycles play a preponderant role in computer animation as they are used for character development, for motion references, and almost everywhere in video games. Regardless, there exists no tool dedicated to their design. In Chapter 5 we present a system that takes advantage of performance animation to create motion cycles. We propose a layered method to make it applicable with any capture device and on any type of character. Additionally, to provide artists with a fine level of control, we use space-time curves to interactively edit the resulting animation. With this approach, a user can animate in the comfort of a single viewport, without the need for any additional window.

1.5 Real-time Animation

The animation techniques discussed in the previous Section permit to animate characters with a high level of control. However, they require precision and iterations, which do not allow to interactively give life to a virtual character. Even when performing the actions, further editing is required to correct some movements, adapt to different morphologies and add secondary motion. Yet, the capability for a character to interact with a human user

is essential for some applications like video games. Using a few inputs, the character is expected to react immediately and depict believable movements.

For real-time character control, the most common approach is the push-button metaphor, where predefined animations are triggered each time the user presses a key. Even when using more advanced devices such as smartphones, clickable buttons are overlaid on the screen to retrieve that usual interface. However, this interaction is limited in the freedom it provides to the user; for example, a "Move Forward" or a "Jump" button does not allow to specify the speed or the height of the action. It results a detached interaction that is far from the satisfactory feeling of grasping and moving a character around, which children cherish doing with physical toys.

We propose a new interaction metaphor in Chapter 6 that reduces the gap between physical toys and virtual characters. We take advantage of different smartphones' characteristics to interactively puppeteer virtual objects. As the user moves the phone around, a character that responds in real time to the manipulations is seen through the screen, as if it was attached to the phone via an imaginary rigid stick. This yields a natural interaction, similar to moving a physical toy, and the puppet now feels alive because its movements are augmented with predefined compelling animations — possibly created using our systems from Chapters 4 and 5. We show various applications of this approach as Mixed Reality games.

1.6 Publications

This thesis is backed by the following peer-reviewed papers:

- **L. Ciccone**, M. Guay, R. W. Sumner
Flow Curves: an Intuitive Interface for Coherent Scene Deformation
Computer Graphics Forum (Proceedings of Pacific Graphics), pp. 247–256, 2016
- **L. Ciccone**, M. Guay, M. Nitti, R. W. Sumner
Authoring Motion Cycles
Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA), pp. 8:1–8:9, 2017
- R. Anderegg, **L. Ciccone**, R. W. Sumner
PuppetPhone: Puppeteering Virtual Characters Using a Smartphone
Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games (MIG), pp. 5:1–5:6, 2018

Introduction

- **L. Ciccone***, L. Casas*, G. Çimen, P. Wiedemann, M. Fauconneau, R. W. Sumner, K. Mitchell
Multi-Reality Games: an Experience Across the Entire Reality-Virtuality Continuum
Proceedings of the ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI), pp. 18:1–18:4, 2018

C H A P T E R

2

Related Work

As established in the previous Chapter, the transition from hand drawn animation to 3D computer animation has changed the way artists work, sometimes at the expense of the ease of creation. Therefore, many works have introduced new algorithms and manipulation techniques in order to provide more freedom to artists and make it more easy to apply artistic principles. Some of them intend to get closer to the natural sketching mechanism characteristic of hand drawn animation, while others propose new interaction metaphors. In the following Sections we provide an overview of related research works for the intuitive control of animated scenes.

2.1 Objects Deformation

2.1.1 Deformation of Single Objects

We have exposed in Section 1.3 how characters are deformed using skeletal bones and rig controllers. However, most of other objects in the scene do not possess such handles and need to be deformed differently. For that, several algorithms exist, such as lattice grids (FFD) to deform free-form objects like point clouds or triangle soups [Sederberg and Parry, 1986; Singh and Fiume, 1998; Milliron et al., 2002].

These types of deformers feature a click-and-drag interface, where the user has to select and transform individual controls. Unfortunately, this interaction does not accommodate most of artistic principles. That is why many researchers investigated sketching as a more natural way of deforming static

Related Work

objects [Hua and Qin, 2003; Kho and Garland, 2005; Nealen et al., 2005; Zimmermann et al., 2007; Kraevoy et al., 2009] and characters [Davis et al., 2003; Sýkora et al., 2005; Lin et al., 2010; Guay et al., 2013; Öztireli et al., 2013; Hahn et al., 2015; Mahmudi et al., 2016].

Thanks to their intuitive aspect and ease-of-use, sketch-based interfaces are very appreciated. However, due to depth, recovering 3D information from a 2D sketch is an ambiguous and under-constrained problem; many solutions can exist for the same sketch. A common approach to solve this issue is to work in the viewing plane [Kho and Garland, 2005; Zimmermann et al., 2007; Guay et al., 2013; Öztireli et al., 2013; Mahmudi et al., 2016]. Another solution is to let the user choose from several possible solutions, as Davis et al. [2003] suggest. Others supplement the problem with prior information on the solutions. For instance, physiological insight on the human anatomy leading to specific joint limits, combined with constraints to maintain balance were used to sketch sitting poses [Lin et al., 2010].

Sketching a deformation requires to define curves by pair: one for the reference and one for the target. The target one is drawn by the user while the reference one is usually computed from the mesh geometry [Nealen et al., 2005; Zimmermann et al., 2007; Kraevoy et al., 2009] or the skeleton of the character [Davis et al., 2003; Guay et al., 2013; Öztireli et al., 2013]. However, these techniques do not allow deforming multiple objects simultaneously with a single stroke, as might be required when crafting the movement of the scene.

2.1.2 Deformation of Multiple Objects

One challenge for multi-object deformation is the fact that different objects are typically deformed with different algorithms (e.g. skeleton, FFD, etc). Hence, the ability to quickly edit an existing raw scene requires a unified approach to deformation. One solution is to formulate the deformation directly on the mesh vertices, typically by penalizing the distortion of triangle elements [Sorkine et al., 2004; Sorkine and Alexa, 2007] while matching some vertex position constraints. Unfortunately, this approach does not support disconnected meshes and point clouds, such as fluids. Additionally, the complexity of the deformation is proportional to the number of vertices, which can impede interactive refinement. An alternative is to automatically compute an embedding of the geometry [Sumner et al., 2007; Botsch et al., 2007; Sýkora et al., 2009] and use similar deformation formulations on the grid embedding elements instead.

In 2D, grid embeddings are mainly used in the context of image warping [Carroll et al., 2010; Orbay et al., 2012]. Unfortunately, a single 2D

grid does not reflect the structural components of the objects in the scene and can lead to undesirable distortions. For example, bending a branch will distort the space around it. Similar to spatial warping, rendering using a nonlinear projection [Coleman and Singh, 2004; Coleman et al., 2005; Brosz et al., 2007] allows to deform images. These techniques do not allow deforming individual parts of objects with ease and, as with spatial warping, do not allow deforming objects without warping the space around them (e.g. the background).

The solution we propose in Chapter 3 for the coordinate deformation of multiple objects benefits from the fast computation of 2D grid deformation while preserving the intrinsic shape of objects. We compute a 2D grid embedding of the 3D elements, in screen-space, that reflects the intrinsic shape of objects, and we automatically compute multi-objects abstractions which allow the user to directly deform multiple objects with a single stroke.

2.2 Animation Crafting

2.2.1 Space vs Time

We have discussed in Section 1.4 the established framework for character animation, called keyframing. It consists in specifying keyposes and placing them at specific points on a timeline to control the timing.

Many works specifically aim to ease the posing process by exploring different interfaces. Sketching is one of them, as presented in the previous Section, but other interaction methods were also explored such as the pin-and-drag metaphor [Yamane and Nakamura, 2003; Shi et al., 2007], the use of reference poses [Wei and Chai, 2011; Choi and Lee, 2016] or even physical devices [Yoshizaki et al., 2011; Glauser et al., 2016]. Separately, other papers focus specifically on improving the timing process through time warping [Witkin and Popovic, 1995; Hsu et al., 2007; Coleman et al., 2008] and gesture-based retiming [Terra and Metoyer, 2004; Terra and Metoyer, 2007; Walther-Franks et al., 2012].

These methods, however, are only focused on either spatial or temporal editing, and do not help the coordination process. Having a timeline disconnected from spatial realities makes the creation of an appealing and believable motion very unintuitive.

One way to improve coordination is through visualization of the motion. Recently, the de facto timeline visualization has been questioned and enhanced with editable pose-icons representing the motion over the timeline [Mukai

Related Work

and Kuriyama, 2009] and deformable spatial planes rendered directly onto the viewport [Yoo et al., 2015]. This helps controlling the timing with some visual feedback, but does not provide control on spatial transformations. Therefore, in this thesis we opt for space-time curve representations that allow the visualization of several aspects of the motion as well as their coordinate manipulation.

2.2.2 Space-time Curves

Space-time curves are geometric representations that permit to visualize and manipulate the trajectory of a certain object/joint/controller over time. They give a good intuition on the character’s movement as well as provide a direct control on the arcs of motion. If drawing them is very straightforward, manipulating them represents a much bigger challenge. Indeed, it not only necessitates an intuitive curve editing mechanism, but also requires to define how the whole character behaves to a specific trajectory’s modifications — e.g. if the motion of a hand is edited, how much of the character’s movement should be affected.

Some recent works have explored the concept of space-time curves for authoring or editing motions. Guay et al. [2015] enable the creation of a full character motion using a single stroke and refine it using additional types of stroke edits. Unfortunately, their strokes are designed around specific types of motions such as bouncing, rolling and waving, and their work only demonstrates a few simple characters (e.g. no bipeds nor quadrupeds). Choi et al. [2016] allow editing a wide range of motions using sketches, but their work is restricted to editing and cannot author new motions. Using screen-space strokes to define 3D deformations, these two papers end up with an underconstrained problem and thus need to make assumptions about the user’s intentions. More importantly, the methods proposed by these previous works require control on every single frame (i.e. one keyframe per frame), which is not compatible with contemporary animation workflows and not practical for editable animations.

Other works use space-time curves as an abstract way to define motion from preexisting animations. In those cases, the curves are used as constraints to retrieve and compose movements from a database [Lee et al., 2002; Min et al., 2009] or as abstract indications to trigger predefined animations [Thorne et al., 2004]. However, these methods do not provide a fine control over the resulting animation, and limit it to the set of preexisting motions.

Our work strives to give the artist freedom over the animation process with minimal restrictions and conflicts with the current animation workflow. We

moreover retain a high degree of genericity so that our control strategy can be applied to any type of content, such as IK handles, bone transformations, or even vertices.

2.2.3 Performance Animation

In order to have control on timing and motion simultaneously, others have explored performance animation (or *puppeteering*) techniques. They consist in using a capture device (mouse, Leap Motion, Vive, full-body motion capture suit, etc.) to mimic the desired motion, that it directly transferred to a virtual character. As humans have an instinctive sense of movement and timing, performance animation has been used as a natural way of specifying motions either for the full character [Chai and Hodgins, 2005; Igarashi et al., 2005; Tautges et al., 2011; Kim et al., 2013] or through a layered type of approach [Dontcheva et al., 2003; Neff et al., 2007; Choi et al., 2008; Martin and Neff, 2012; Jin et al., 2015].

These methods introduce a much more intuitive way to create movements, but elaborate animations are still very hard to achieve. Indeed, when gesturing motions, users are not accurate enough to perfectly perform some stylized effects or specific actions such as contacts. Furthermore, the resulting motion is typically driven by animation curves with a complex parameterization, which does not permit easy editing.

In this thesis, we make the important observation that performance animation is particularly appropriate for motion cycles, and is most natural when the motion is repeated several times. We then propose an algorithm to deliver a single closed loop from a sequence performed by the user, and a curve representation that enables a precise editing of the motion.

2.3 High-level Character Control

2.3.1 Space-time Constraints

Space-time constraints were introduced by Witkin and Kass [1988] to achieve motions that satisfy constraints. In this paradigm, a user can specify high-level spatial and temporal constraints and the motion is produced automatically via non-linear optimization. This inspired a lot of further research, including many works focusing on articulated characters' animation, which extended the approach to provide a user interface [Cohen, 1992], create transitions between motion segments [Rose et al., 1996], enable the easy edi-

Related Work

tion of existing animations [Gleicher, 1997], retrieve a motion in a large database [Min et al., 2009] and even synchronize the motion of multiple characters [Kim et al., 2009].

A particular category of space-time constraints techniques seek to automatically generate believable transitions by relying on simulations, databases or heuristics. Important work with probabilistic models of human motion was used for filling gaps of animations [Chai and Hodgins, 2007; Wang et al., 2008; Lehrmann et al., 2014], collision detection was used to correct default interpolations [Nebel, 1999], and more recent research used deep learning to generate animations that interpolate key poses [Zhang and van de Panne, 2018; Harvey and Pal, 2018].

To allow for more controllability, Koyama and Goto [2018] provide mathematical formulations for the control of an optimization based on physically inspired energy terms. Their work has the additional benefit of modifying tangents instead of adding keyframes, similar to our motivations in Chapter 4. However, their approach is fundamentally different from ours: while we aim at real-time control of interpolations with an intuitive interface directly in the viewport, Koyama and Goto [2018] provide an indirect and offline motion manipulation tool via sliders and graph editors.

If space-time constraints allow for a fast editing of motion via high level properties, producing animation in this way can be difficult as it requires some intuition in how exactly to balance the different hard and soft terms of the cost function such that the desired motion is produced. Fine-tuning the result is also laborious as most often the parametrization of the animation curves is altered based on the computational needs of the system. Furthermore, with space-time constraint techniques, the set of achievable motions is restrained by the physical formulations or the library of movements the optimization is based on.

2.3.2 Real-time Manipulation

Most of the mentioned approaches to ease the animation process do not enable a real-time crafting of the full character motion. Sketch interfaces [Thorne et al., 2004; Jeon et al., 2010; Guay et al., 2015] require to entirely specify sketch abstractions before the character can move; curve editing techniques [Lee et al., 2002; Min et al., 2009; Choi et al., 2016] involve iterations for the specification of different points' trajectories; and space-time constraints [Witkin and Kass, 1988] demand a careful definition of the different optimization weights. Performance animation, when not used in a layered approach, can enable the interactive manipulation

of an entire character, usually using a full body motion-capture suit [Song et al., 2017]. However, even if many works aimed to reduce the number of sensors required when performing the motion [Oore et al., 2002; Chai and Hodgins, 2005; Shiratori and Hodgins, 2008; Liu et al., 2011; Tautges et al., 2011; Kim et al., 2012], this approach requires specialized devices that casual users rarely possess, and it is limited to physically realistic motions of human-like characters.

Video games is the most prominent example of a system that involves real-time animation of virtual characters. Most often, the interaction consists in a push-button approach, where the user hits a button or a point on the screen to trigger a specific action (e.g. 'Move there', 'Jump', 'Kick', 'Shoot here', etc.). Since it is an underconstrained interface, the full character animations is generated using motion data-bases [Arikan and Forsyth, 2002; Min and Chai, 2012; Holden et al., 2016; Holden et al., 2017], physical simulations [Laszlo et al., 2000; Yin et al., 2007; Coros et al., 2010; Geijtenbeek et al., 2013] or even both [Liu et al., 2010; Geijtenbeek et al., 2012; Zordan et al., 2014]. To not limit the interface to a set of buttons, other works propose to use abstract gestures [Rhodin et al., 2015; Cui and Mousas, 2018]. In those techniques, user movements are linked to specific actions of the character, which makes it possible to animate it using different devices and body parts. However the interaction is very abstract and unnatural.

For a more natural control on the character's displacement, Willis et al. [2011] propose to use a handheld projection system and animate a character in the virtual world as one moves its projection on a wall. Unfortunately, their technique only allows displacements in 2D (on the wall) and simplistic animations. We alleviate this shortcoming in Chapter 6 by introducing a similar grasping metaphor in Augmented Reality using a smartphone. This makes our system accessible and easy to use. We furthermore require very few preexisting motions, and we compose blended ones on the fly using a technique based on the inverse distance weighting.

Related Work

Flow Curves: an Intuitive Interface for Coherent Scene Deformation

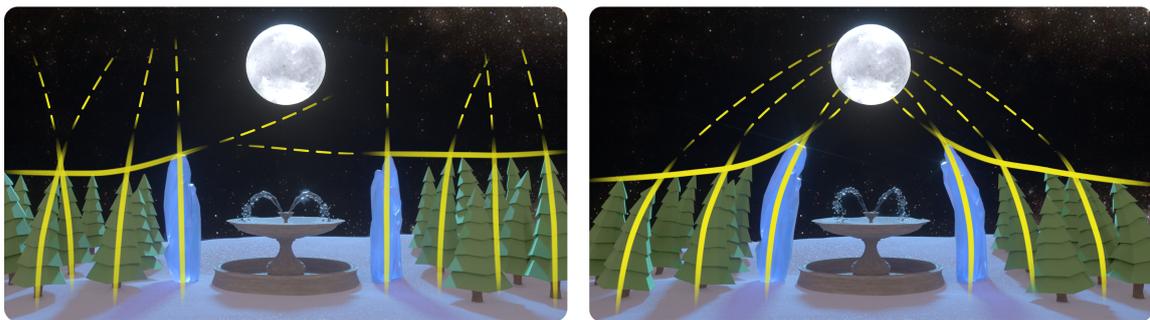


Figure 3.1: Our new Flow Curves interface is designed to help artists take a scene with an ambiguous flow (left), and quickly turn it into a compelling scene (right) by simply sketching strokes — inducing multi-object deformations.

We discussed in Section 1.2 how effective compositions rely on the principles of visual arts. They allow skilled artists to cleverly exploit mechanisms of our visual system in order to more effectively communicate the important elements of an image. In particular, they use our tendency to pick up contrast edges across multiple objects and join them together into longer imaginary *subjective curves* to direct the eye’s movement across the image. Hence, by carefully shaping and coordinating contrast edges across objects in a scene, visual artists are capable of controlling the very way we look at an image and bring our attention to a *center-point* (see Fig. 3.2). While this concept is referred to as the principle of *movement* in visual arts, it is also called *flow* in design, and *good continuation* in Gestalt perceptual theory.



Figure 3.2: *Movement in existing artwork (Left: book cover of Predator’s Gold, Middle: screenshot from Mickey’s Christmas Carol, Right: poster of Ratatouille). Yellow lines represent subjective curves pointing to a center-point.*

Traditional 2D sketching offers the freedom to naturally craft and design scene-wise subjective curves. In contrast, current 3D tools are designed in an object-centric fashion which does not easily accommodate building a coherent movement, for a given view-point. In fact, digital artists are obliged to shape each object separately, using several rig and deformation handles that typically differ from object to object. In other words, these tools are agnostic to the artistic principle of movement and provide no means of expressing coherence across multiple objects in a scene. Hence, despite the dramatic importance of movement in visual design, 3D artists are left with a cumbersome, indirect and time consuming workflow.

Our work in this Chapter addresses this shortcoming by introducing a new sketch-based interface called *Flow Curves* that provides a direct coordination control across multiple objects. From raw input geometry, we automatically compute a deformation embedding for objects in the scene that allows directly deforming shapes without rig setup time. To provide the user with the possibility to deform objects with a single stroke, we automatically compute *subjective curve elements* (or *SEcurves*) spanning multiple objects in the form of *principal curves* and *abstract contours*. By defining a *flow curve* as a shape constraint over close-by SEcurves, we can optimize for a scene deformation that conforms to the sketched flow curves. When compared to traditional workflows, our interface offers a significant increase in efficiency (see Table 3.1 for comparison).

“The central point of interest has nothing to do with the center of the frame, but everything to do with where you wish the audience’s eye to eventually rest.” [Button, 2002]

3.1 Background

Artistic principles. Some principles of visual arts have already been integrated into computational tools, such as *balance* for photo composition [Liu et al., 2010] and scene layout placement [Liu et al., 2015], or *emphasis* for directing gaze [Cole et al., 2006; Bailey et al., 2009]. In this Chapter, we focus on the principle of movement, that Glatstein defines as “*the way a viewer’s eye is directed to move through a composition, often to areas of emphasis.*” [2013]. To our knowledge, this principle has never been used for whole scene deformation, but only for individual characters with the *line of action* drawing concept [Öztireli et al., 2013; Guay et al., 2013], where the whole shape of a character forms a smooth (skeletal) curve. Our flow curves subsume lines of action and allow deforming whole scenes — including characters — using a single stroke.

Subjective curves Those are imaginary curves that we perceive when grouping together salient contrast edges and points. Gestalt theory refers to this perceptual phenomenon as the principle of “good continuation” [Wertheimer, 1938]. Due to its subjective nature, a precise mathematical characterization of good continuation remains elusive. Additionally, the bulk of academic research in computer vision is geared towards identifying *existing* subjective curves, either via fixed primitives such as circle arcs and Euler spirals [Ullman, 1976], minimal curvature splines [Horn, 1983; Mumford, 1994], or probabilistic models [Williams and Jacobs, 1997] where a scalar (probability) field is computed from all the pairwise interactions between edge segments (curve formed by tracing paths along probability peaks). These methods identify subjective curves in existing images, while our work provides a practical tool for shaping and forming new ones.

Automatic shape abstractions. To allow the user to deform raw scenes with a single stroke, we automatically compute abstractions in the form of principle curves and abstract contours. While it would be straightforward to use rigged objects with our method (skeletons representing a good abstraction for many shapes), we decided to work in the general case where most objects of the scene are not rigged (trees, fences, furniture, etc.). Automatic skeletonisation techniques compute a skeleton from a mesh or point cloud [Au et al., 2008; Tagliasacchi et al., 2009; Huang et al., 2013] but yield skeletons with noisy branches that can rarely be sketched directly. Another approach is to assume a parametric curve such as a spline and optimize its shape as to conform to the object using an iterative closest point frame-

work [Kass et al., 1988]. We build upon this approach as it is well suited for sketching the smooth splines. Methods on shape abstractions that seek to conserve fine details [Vishwanath et al., 2013] provide a contour that is too fine for the smooth contours we search. On the contrary, other works seek to remove details while preserving a coarse version of the shape [Mi et al., 2009; Mehra et al., 2009]. However, pruning parts of objects solely based on geometric features does not result in proper outer-abstractions such as contours in the case of pointy objects, e.g. branches of a tree or poles of a fence.

3.2 Overview

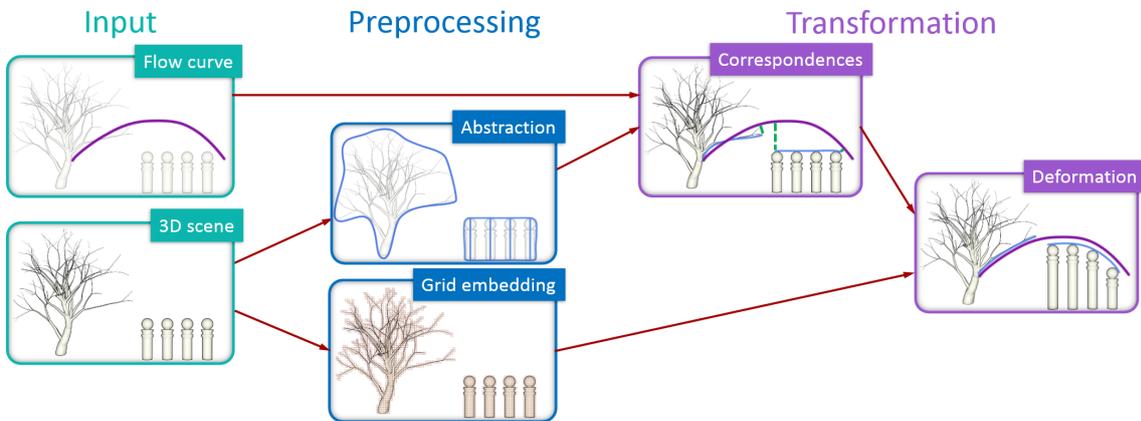


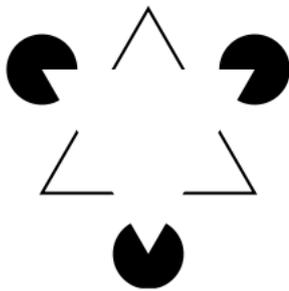
Figure 3.3: Overview of our approach applied on a simple scene. The user gives as input a 3D scene, on which our system automatically computes *SEcurves* (in the form of objects abstractions) and a 2D grid embedding of the objects (used for deformation). Then, when the user provides a flow curve, *SEcurve* pieces are selected and associated to corresponding flow curve pieces based on a geometric closest-point approach, and the grids are deformed as to have these *SEcurve* pieces match the shape of the flow curve, in screen-space.

First, we need to determine which contrasts in the scene could be deformed to form strong subjective curves. Since our goal is different from computing existing subjective curves in images, we compute partial subjective curve elements that we call *SEcurves*. While detecting these is highly ambiguous, we observed that in many cases, subjective curves are formed from a skeletal curve of objects, or from parts of abstract outer-contours. Based on these observations, we compute principal curves for thin objects and abstract contours for complex ones, as well as groups of objects (Section 3.3). While these *SEcurves* cover a wide range of cases, users may sketch additional ones if desired.

As a means of forming coherent subjective curves via scene deformations, we introduce a sketch-based interface called Flow Curves. The flow curves are defined as *shape* constraints for SEcurves in their vicinity (Section 3.4). We offer two intuitive ways of specifying flow curves: (1) by sketching strokes, which provides direct control onto the subjective curves of the scene and (2) by sketching a center-point which automatically generates a coherent network of flow curves converging towards it. This center-point provides coordinated control onto the scene’s movement as a whole.

To match the SEcurves to their corresponding flow curves, we need a deformation method that allows deforming different types of objects (including point clouds such as liquids), requires little manual setup time, and preserves the intrinsic shape of objects (i.e. does not distort space). Our solution to these requirements is to compute a 2D grid embedding in screen-space for each individual object in the scene and to formulate the constrained deformation on the grid embedding (Section 3.5). Constraints are the target position of SEcurves and the conservation of the objects placement layout.

3.3 Subjective Curve Elements



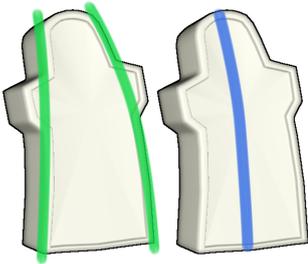
Our visual system perceives curves that are subjective and not entirely explicit: we fill-in the space between contrast edges and points, and even skip over parts of objects at high curvature points, in favor of longer smoother curves. For example, in the image on the left, we naturally see two solid triangles while none is entirely present. In this Section, we identify strong edges and shape features that could be elements of subjective curves — we call them SEcurves.

The goal of our tool is to facilitate coordinating SEcurves into forming long and coherent subjective curves — as opposed to finding existing subjective curves as is done in the computer vision literature. Computing edges is often noisy as well as impractical for sketching. Additionally, computing every possible edges that could be used for creating movement in a scene would make the selection process ambiguous (how to pick the right SEcurve from a soup of edges?). By observing users sketching their own subjective curves, we found that they are often formed from two categories of recurrent SEcurves. The first is principal curves which abstract thin objects, and the second is abstract contours which join the tips and edges of complex shapes such as trees, plants and fences, as well as join together groups of nearby objects.

Based on these observations, we first compute for each object in the scene both their principal curve (a generalization of the principal axis, detailed in Section 3.3.1) and abstract contour (Section 3.3.2). Then we decide which is best suited based on a measure of thinness. We estimate thinness as the variance of the point-wise distance d_i between each point of the principal curve and the contour. A small variance means that the principal curve is a better approximation of the object than the abstract contour. Hence, if $\sum d_i^2/N - (\sum d_i/N)^2$ is below a threshold, we keep the principal curve, otherwise we keep the contour.

Shapes may form a principal curve individually, but when located close to each other join and form a stronger contour, as shown by the four poles in Fig. 3.3. Hence, in a second step, we measure whether multiple principal curves are close to one another, in which case we compute an abstract contour around this group of objects. As a result, the group holds individual principal curves *and* a group-wise abstract contour.

3.3.1 Principal Curves



The tombs on the left illustrate how principal curves (the blue curve on the right) are a practical approximation of side contrast edges (the green curves on the left). We propose a method to automatically compute principal curves for arbitrary objects in the scene. Principal curves have been studied in statistics as a non-parametric model of curved manifolds and a generalization of the principal (vector) component [Silverman, 1985]. We follow a spline regression framework, where the correspondence between the data points and the spline is not known a priori but must be estimated — typically in an iterative closest point (ICP) fashion.

Applying this technique to 3D meshes leads to problems of its own. First, the distribution of points along the surface can influence the shape of the principal curve, hence we need to re-sample the surface uniformly. Second, minimizing the distance alone can lead to spurious curves when applied to shapes that are not equally spaced-out w.r.t. their center-line (such as the tree in Fig. 3.4). We address this issue with iterative regularization, i.e. we penalize the solution w.r.t. the last computed curve between consecutive ICP iterations.

Our first step is to compute a uniform sampling of the surface mesh. As we work in screen space, we use the depth map and sample pixels whose depth is lower than 1, yielding 2D surface points $X = \{x_1, x_2, \dots, x_N\}$. Note,

we believe our approach easily extends to the 3D case with 3D surface re-sampling. We initialize the spline $c_0(s)$ to the principal axis x_{axis} of the surface by computing the largest eigen vector of the sample covariance matrix formed from all the surface points $Cov(X, X) = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})(x_i - \bar{x})^T$, using principal component analysis. We use cubic Hermite splines in our implementation, and initialize the length of $c_0(s)$ to the size of the shape in its principal direction.

Then, we refine the curve by iteratively solving the following problem. At each iteration, we compute the parametric correspondence s_i^* of every surface point x_i to the current principal curve by computing the closest point ($s_i^* = \operatorname{argmin}_s \|x_i - c_k(s)\|$). And then we minimize the euclidian distance between both points, w.r.t. the spline degrees of freedom, while penalizing deviations in shape from the previous curve:

$$c_{k+1}(s) = \min_{c(s)} w_1 \sum_i \|x_i - c(s_i^*)\|^2 + w_2 \int_s \left\| \frac{\partial c(s)}{\partial s} - \frac{\partial c_k(s)}{\partial s} \right\|^2. \quad (3.1)$$

We then increase k and iterate until no more improvement is gained. In our implementation, we used a weight w_2 1000 times bigger than w_1 . To compute this integral, as well as all the following ones, we discretize the Hermite curve into equally-spaced points and sum over the desired values at these points. Examples of principal curves computed by our algorithm are shown in Fig. 3.4.

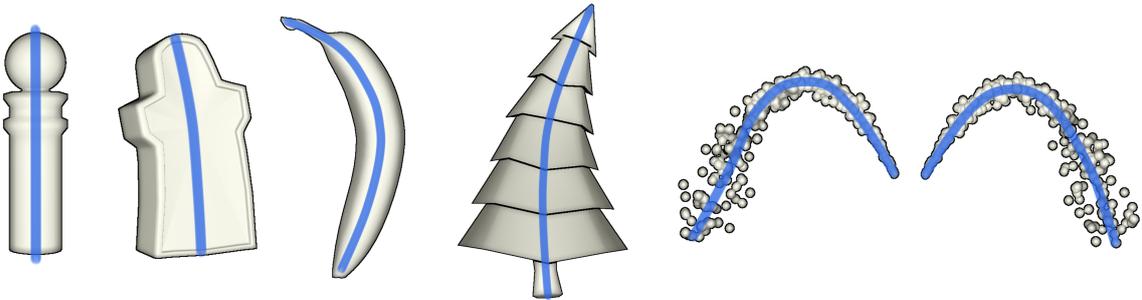


Figure 3.4: *Thin objects can be abstracted by a principal curve, which approximates their side edges.*

3.3.2 Abstract Contours

Subjective elements are often perceivable around the tips and edges of complex shapes, or joining groups of close objects. Outer-contour groupings are

often what we naturally sketch when drafting the shape of objects and as such are an important class of SEcurves (see Fig. 3.3 middle).

We also use spline fitting to compute the abstract (outer) contour of an object or group of objects, but here using a closed curve. We minimize the area $E_A(c)$ enclosed by the curve $c(s)$ while preventing points of the object from moving outside the curve, resulting in a penalizing energy term $E_O(c)$. The level of abstraction is controlled by weighting the curvature penalization term $E_C(c)$. For the closed curve $c(s)$, we use a cubic Hermite spline and initialize it to the bounding box around the object. We detail below the optimization problem we solve.

Area $E_A(c)$. We penalize the area enclosed by the curve, scaled by the object’s bounding box area a_0 . We discretize the curve into equal-length segments and compute the enclosed area by summing all the signed areas under these segments, using the determinant of the 2×2 matrix formed by setting the consecutive curve samples as columns: $[c(s_i) c(s_{i+1})]$, resulting in:

$$E_A(c) = \frac{1}{2 \cdot a_0} \left| \sum_i \text{Det} [c(s_i) c(s_{i+1})] \right|. \quad (3.2)$$

Outside points $E_O(c)$. We compute the number of edge pixels p_i outside the curve $c(s)$. To do so, we trace vertical lines (in the \vec{y} direction) starting at each edge pixel p_i , and count the number $I(p_i)$ of intersections with $c(s)$. The parity of $I(p_i)$ determines whether it is inside (odd) or outside (even). Note that this term acts as a hard constraint and thus has a large weight w_O .

$$E_O(c) = \sum_i 1 - (I(p_i) \% 2). \quad (3.3)$$

Curvature $E_C(c)$. We penalize the curvature of the contour $c(s)$, which corresponds to its second derivative (we approximate it numerically using equally-spaced samples). Controlling a soft curvature penalty weight controls the level of abstraction of the contour. Indeed, if $c(s)$ is allowed to be flexible, it can move inside cavities of the object in order to minimize area.

$$E_C(c) = \int_s \left\| \frac{\partial^2 c(s)}{\partial^2 s} \right\|^2. \quad (3.4)$$

The total energy is non-linear and the term E_O is discontinuous. Hence we minimize the sum of energies using stochastic optimization with Covariance Matrix Adaptation (CMA):

$$E(c) = w_A E_A(c) + w_O E_O(c) + w_C E_C(c), \quad (3.5)$$

setting the parameters to $w_A = 100$, $w_O = 10^5$, and $w_C = 5$ in our results. Note that there can be infinite reparameterizations of $c(s)$ that yield the same total energy values; this can cause global stochastic optimization to loop without significant gains. We alleviate this issue with a stopping criteria based on the relative improvement of the objective function. Examples of abstract contours computed by our algorithm are shown in Fig. 3.5.

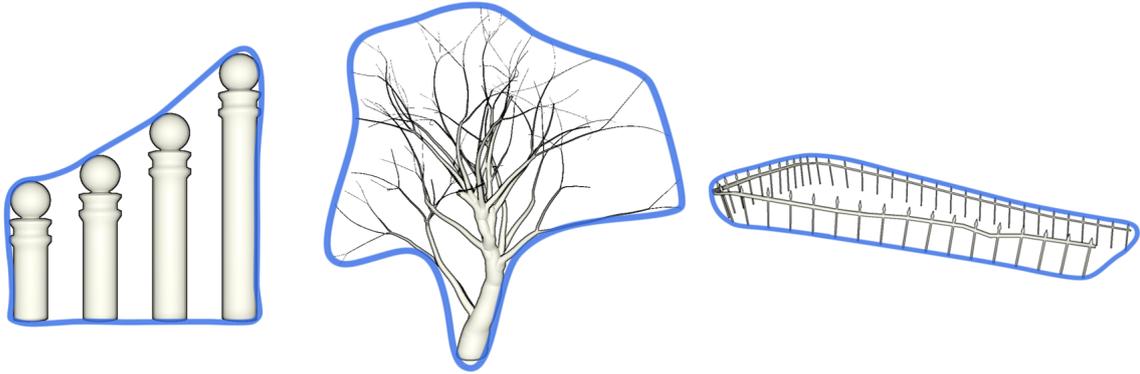


Figure 3.5: *Complex shapes, as well as groups of objects, form abstract contours. We compute them using spline fitting, i.e. by minimizing the area of a closed spline curve while ensuring that points of the object remain inside the closed curve. The level of abstraction is controlled by penalizing curvature.*

3.4 Flow Curves

We define flow curves as shape constraints over SEcurves. The goal of flow curves is to design smooth movement and we thus first describe how to control the smoothness of the sketched flow curve — if desired by the user. A second goal is coherence, not only for individual subjective curves, but also for the scene as a whole. Hence, the second control we provide to the user is a center-point, which generates a coherent network of smooth flow curves.

3.4.1 Flow Curve from Sketched Stroke

We can control the smoothness of a flow curve $\gamma(s)$ drawn by the user via spline fitting and curvature penalization. We fit a cubic Hermite curve to the stroke samples f_i while penalizing the second derivative of the curve:

$$\min_{\gamma(s)} \sum_i w_1 \|f_i - \gamma(s_i)\|^2 + w_2 \int_s \left\| \frac{\partial^2 \gamma(s)}{\partial^2 s} \right\|^2, \quad (3.6)$$

which we optimize w.r.t. all the spline degrees of freedom, i.e. the position and tangent of every control point. The number of control points n_γ is determined by the length l_γ of the sketched stroke. Observing that a flow curve of length $w_s/4$ (w_s being the width of the screen) is well represented by 4 control points, we used $n_\gamma = \left\lceil 3 \cdot \frac{4 \cdot l_\gamma}{w_s} \right\rceil + 1$.

3.4.2 Flow Curves Network from Center-Point

When the user sketches a circle shape, we interpret its center of mass as the center-point and generate a network of flow curves. To be consistent with the current scene, the generated network depends on the scene configuration and the present SEcurves. But its definition is unfortunately not unique; for example, on the top row of Fig. 3.6, each subjective element individually points towards the center-point, while on the second row, they move in accordance, eventually bending towards the center-point. Hence, we compute a parametric space of flow curves and provide the user with a flow radius parameter d_{Max} , indirectly controlling the number of flow curves generated.

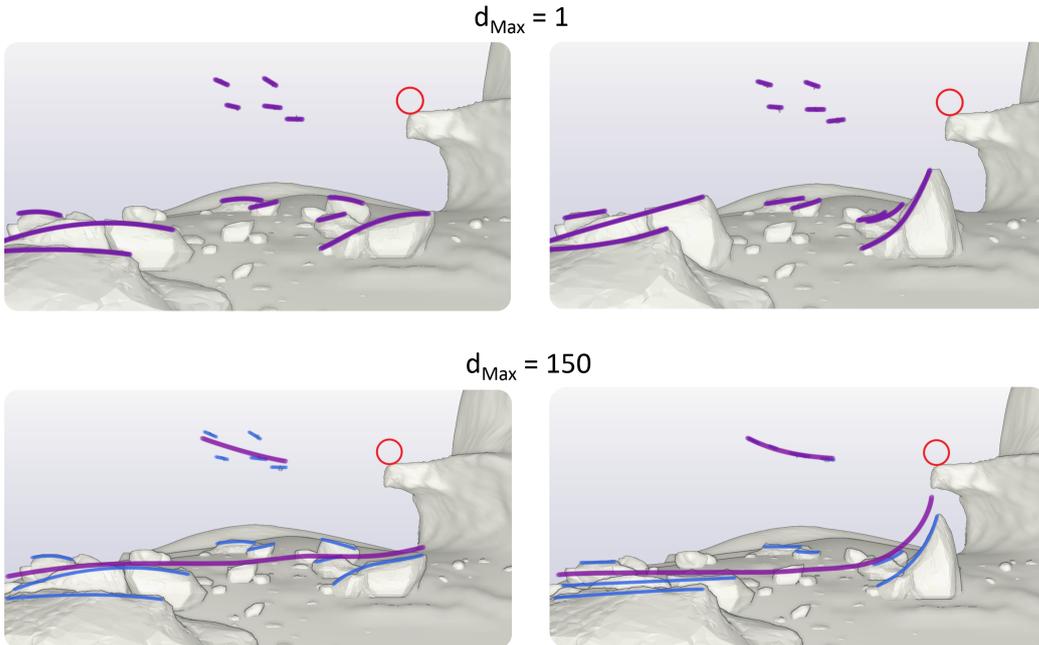


Figure 3.6: We allow the user to specify only a center-point, from which several flow curves are computed — each converging to the center-point. We first initialize the flow curves (in purple) from the SEcurves (in blue), as shown in the left column. The user can control the amount of grouping for the generated flow curves through a parameter d_{Max} . The right column shows the deformed curves and scenes.

Our idea is to first compute average curves from the SEcurves present in the scene, before bending them in order to make them coherently flow towards the center-point.

Average curves. To estimate the average flow curve in a region of the scene, we first extrapolate each SEcurve in both directions by interpolating the direction of the nearest SEcurves. To do so, we start with an extremity $c(1)$ (or $c(0)$) and look for the nearest points $c_j(s^*)$ on the other SEcurves at a distance below d_{Max} . Then we integrate the curve's position by averaging the directions with an inverse distance weighting scheme:

$$c(s + \Delta s) = c(s) + \Delta s \frac{\sum_{j=1}^N r(\|c_j(s^*) - c(s)\|) \frac{\partial c_j(s^*)}{\partial s}}{\sum_{j=1}^N r(\|c_j(s^*) - c(s)\|)}, \quad (3.7)$$

where r is a radial kernel function used to interpolate the directions of the neighboring SEcurves; we used $r(d) = \frac{1}{1+d^2}$. Note that we only include in the sum the points that verify $\|c_j(s^*) - c(s)\| < d_{Max}$ and $\angle \left(\frac{\partial c_j(s^*)}{\partial s}, \frac{\partial c(s)}{\partial s} \right) < \theta_{Max}$. This condition on the angle between tangents allows avoiding influence from inappropriate curves and supporting curves crossing each other (we used $\theta_{Max} = \frac{\pi}{3}$). We stop when there is no more point verifying these conditions.

Finally, knowing which SEcurves influenced the extrapolation of which other ones, we cluster the resulting curves. We then compute their average curve $\bar{c}(s)$ (by imposing a common parametrization on every clustered curve), that we smooth into a Hermite curve using Equation 3.6).

Bending curves. We now bend the computed average curves as to have them smoothly join the center-point (Fig. 3.6 right). For each curve $\bar{c}(s)$, we fix an extremity control point (its position and tangent) and deform the remaining part as to point towards the center-point, giving a flow curve $\gamma(s)$. To know in which direction to deform the curve (i.e. choosing between fixing $\bar{c}(0)$ or $\bar{c}(1)$), we deform both cases and measure the deformation magnitude (difference in shape), and we select the case yielding the smallest deformation. Note that we also use this measure to remove flow curves that would deform the scene too drastically.

Having a flow curve *join* a center-point means that if we extrapolate its path it should eventually touch the center-point. We approximate this measure with the angle between the curve's tangent at its tip and the vector between the tip position and the center-point, resulting in $\theta_p = \angle(\gamma(1) - x_p, \frac{\partial \gamma(1)}{\partial s})$.

Also, the length of the curve $l(\gamma)$ is constrained to stay close to the initial length $l(\bar{c})$. Hence by minimizing these values and controlling the smoothness via curvature, we obtain the following problem:

$$\begin{aligned} \min_{\gamma(s)} \quad & w_1 \theta_p^2 + w_2 |l(\gamma) - l(\bar{c})| + w_3 \int_s \left\| \frac{\partial^2 \gamma(s)}{\partial s^2} \right\|^2 \\ \text{subject to} \quad & \gamma(0) = \bar{c}(0), \quad \frac{\partial \gamma(0)}{\partial s} = \frac{\partial \bar{c}(0)}{\partial s}, \end{aligned} \tag{3.8}$$

where the direction term has a larger weight ($w_1 = 500$) than length conservation ($w_2 = 0.5$) and curve smoothing ($w_3 = 7$), to ensure pointing towards the center-point.

3.5 Direct Deformation of Scene Objects

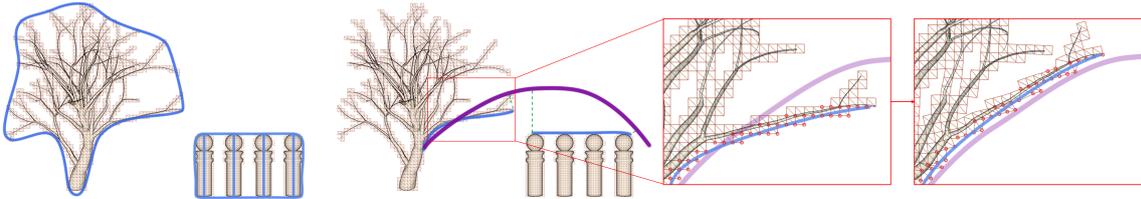


Figure 3.7: To deform arbitrary objects in the scene, we compute their 2D embedding, in screen-space, which is then used for deformation. The grid is a coarse approximation that preserves its intrinsic shape when deformed. The SEcurves are expressed as linear functions of the 2D grid triangles, shown with red points. We optimize for a deformation of the grid as to match the shape of SEcurves (in blue) to the shape of flow curves (in purple).

Deforming the scene objects as to have SEcurves match the shape of flow curves would require re-computing them at each step of the process. We circumvent this problem by using a common 2D grid embedding of both the object geometry *and* SEcurves (Section 3.5.1). The correspondence between SEcurves and flow curves is automatically computed using proximity and shape criteria (Section 3.5.2). From the correspondence, we derive position constraints on grid vertices for matching the curves, and then minimize an as-rigid-as-possible energy expressed on the grid embedding vertices (Section 3.5.3).

3.5.1 2D Grid Embedding

Since movement edits are screen-space refinements, we utilize a deformation representation designed to operate in screen-space. Given a 3D object with vertices $V = \{v_1, v_2, \dots, v_N\}$, we build a 2D triangle grid embedding defined by vertices x_g by uniformly tessellating the object's bounding box in screen space at a fixed resolution (15 pixels in our examples). We then remove all triangles not occupied by the mesh so that the grid closely conforms to the object's projected shape (see Fig. 3.7, left). We parameterize the 3D object vertices v_i to their projected position $\mathbf{P}v_i = \phi_i(x_g)$ in the grid using barycentric coordinates, together with depth values in view space $d_{z_i} = \|v_i - \mathbf{P}v_i\|$. Here, \mathbf{P} is the perspective projection matrix. After deforming grid elements x_g into x'_g (Section 3.5.3), we recover the deformed mesh position v'_i with:

$$v'_i = \phi_i(x'_g) + d_{z_i}x_{dir}, \quad (3.9)$$

where x_{dir} is the unit vector between the camera position and point $x'_g(i)$ on the screen.

3.5.2 Automatic Correspondence

Given a flow curve γ and SEcurves c_i , we automatically select parts of SEcurves that will be transformed and compute their correspondence to the flow curve. The result are new curve segments c_l and γ_l , sharing a common parameterization $s: c_l(s) \rightarrow \gamma_l(s)$. Our solution consists of two steps. We first select curve segments \tilde{c}_k and $\tilde{\gamma}_k$ based on a closest point approach combined with a curve similarity measure [Cohen and Guibas, 1997]. Because this initial selection may include SEcurve segments that are either too small or in conflict with one another (an example is the tree's contour in Fig. 3.7, which could have both parts of its base selected), we filter undesirable segments based on a score.

The initial segmentation process computes all the corresponding SEcurve segments \tilde{c}_k and flow curve segments $\tilde{\gamma}_k$, by going through all the points of the discretized curve c_i and storing the ones $c_i(s_j)$ whose closest point $\gamma(s_j^*)$ is under a threshold $\|\gamma(s_j^*) - c_i(s_j)\| < d_m$ (we use $d_m = 100$ pixels), and whose angle between tangents is under a threshold $\angle \left(\frac{\partial c_i(s_j)}{\partial s}, \frac{\partial \gamma(s_j^*)}{\partial s} \right) < \theta_m$ (we use $\theta_m = \frac{2}{3}\pi$). The result is a set of corresponding curve segments $\tilde{c}_k \rightarrow \tilde{\gamma}_k$.

Then, to filter undesirable segments, we compute a score $S(\tilde{c}_k)$ for each segment, based on its length and mean distance to the flow curve (i.e. mean of

$\|\tilde{\gamma}_k(s_j) - \tilde{c}_k(s_j)\|$):

$$S(\tilde{c}_k) = w_1 l(\tilde{c}_k) + w_2 \frac{1}{1 + d(\tilde{\gamma}_k, \tilde{c}_k)}. \quad (3.10)$$

We keep only segments whose score is above S_{min} , resulting in the set of segments $\{c_l, \gamma_l\}$. We use values $w_1 = 1$, $w_2 = 10^5$ and $S_{min} = 150$ in our implementation.

3.5.3 Deformation

Given a flow curve segment γ_l and its corresponding SEcurve segment c_l (computed in Section 3.5.2), our goal is to deform the grids as to match the shape of c_l to the shape of γ_l :

$$\frac{\partial c_l(s)}{\partial s} = \frac{\partial \gamma_l(s)}{\partial s}. \quad (3.11)$$

We achieve this by computing an as-rigid-as-possible grid deformation that satisfies this differential constraint. For increased speed, we use a fast implementation of ARAP which only solves for hard position constraints. We thus turn the differential constraints into hard position constraints of the grid vertices w_i . We also add positional constraints to preserve the initial placement of objects in the scene.

Layout positions constraints. We compute intersections between objects and create position constraints for all grid vertices to which these intersections project: $w_i^*, \forall i \in L$, where L is the set of all constrained positions' indices on a given object grid. When no position constraints are detected for a particular object, the natural behavior is to translate, allowing their SEcurves to match the flow curve, as is the case for the birds in Fig. 3.9.

Shape constraints into position constraints. We approximate the differential expression above (Equation 3.11) in terms of positional constraints by translating the flow curve γ_l to the position on the subjective curve that is closest to a constrained grid point: $\gamma_l(s) := \gamma_l(s) + (c_l(s^*) - \gamma_l(s^*))$, where $s^* = \operatorname{argmin}_s \|c_l(s) - w_i^*\| \forall i \in L$. When there are no constrained grid positions ($L = \emptyset$), we translate the SEcurve to the nearest flow curve point: $c_l(s) := c_l(s) + (\gamma_l(s^*) - c_l(s^*))$, where $s^* = \operatorname{argmin}_s \|c_l(s) - \gamma_l(s)\|$. We then use γ as a position matching constraint. We derive the appropriate constrained grid positions similarly to other sketch-based deformation methods [Zimmermann et al., 2007], by mapping the relative position of vertices close to c_l onto γ_l .

Given these constraints, we wish to compute a deformed grid that respects the constraints while minimizing the local distortion of grid elements. To do so, we compute a deformation energy $E_{shape}(\mathbf{T})$ that measures the non-rigidity of triangle transformations $\mathbf{T} = T_1, T_2, \dots$, where T_i transforms triangle i from its undeformed to its deformed state. As such deformations are well studied, we employ an as-rigid-as-possible deformation [Sorkine et al., 2004] to solve the following problem:

$$\begin{aligned} \min_{x_g} \quad & E_{shape}(\mathbf{T}) \\ \text{subject to} \quad & c_l(s) = \gamma_l(s) \quad \forall l \\ & w_i = w_i^* \quad \forall i \in L. \end{aligned} \quad (3.12)$$

Vertex position constraints are enforced by construction by removing them from the set of unknown variables. Due to the 2D formulation, the overall system is solved efficiently, leading to interactive performance. After solving for the grid deformation, we recover the mesh vertex positions using Equation (3.9).

3.6 Results and Discussion

Fig. 3.9 shows how four different scenes were edited using sketched flow curves. We refer to them as 1-Fountain, 2-Octopus, 3-Cliff and 4-Cemetery, following the top-to-bottom order. The last three scenes (2,3,4) are inspired by existing artworks shown in Fig. 3.2. The various scenes allowed us to evaluate our interface on different types of objects (connected and disconnected meshes such as the poles of a fence) and scene configurations.

The second functionality provided by flow curves is the ability to sketch a center-point circle. The center-point — taken as the center of the sketched curve — automatically generates coherent flow curves to deform the scene (as described in Section 3.4). In Fig. 3.8, we show the possibility for the user to interactively manipulate the center-point while our system generates coherent flow curves that directly deform the scene.

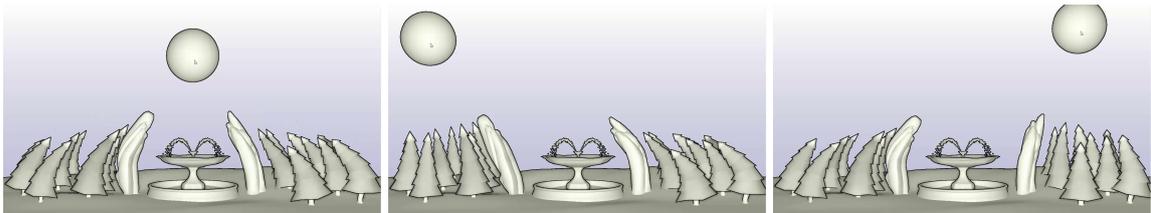


Figure 3.8: *The user can control the center-point, thereby interactively inducing flow-preserving deformations over the whole scene.*

Flow Curves: an Intuitive Interface for Coherent Scene Deformation

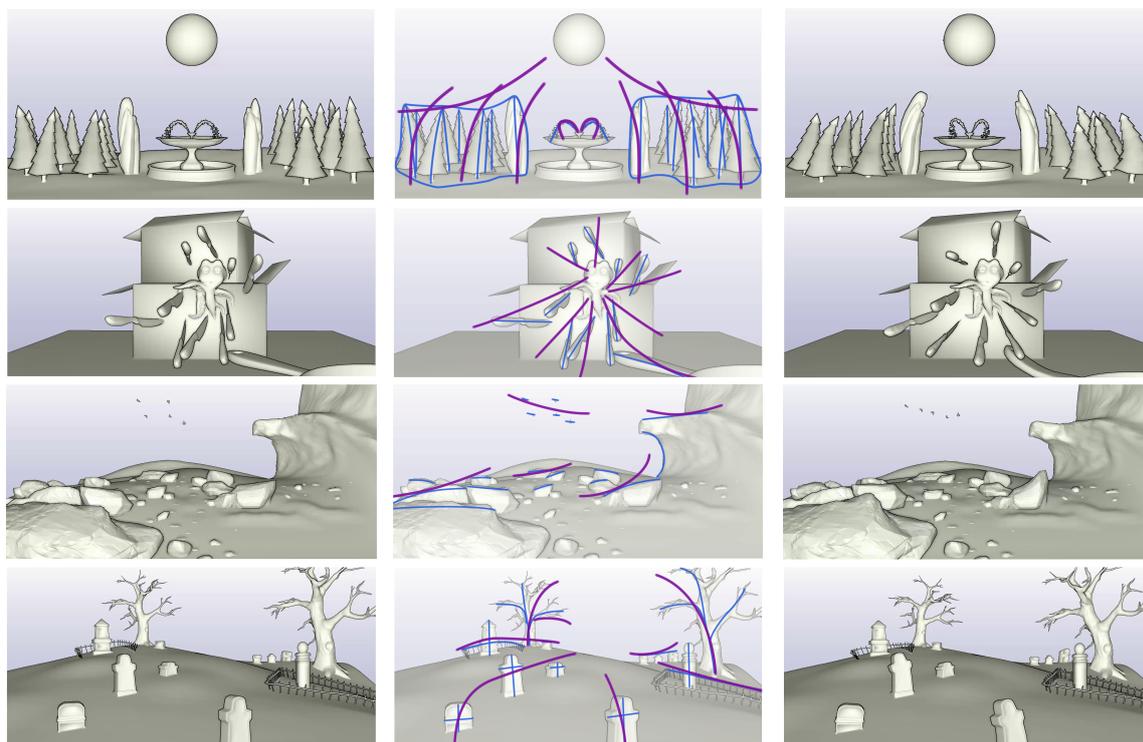


Figure 3.9: *Left: input scene. Middle: SEcurves (blue) and sketched flow curves (purple). The SEcurves were computed automatically in the first two rows, and manually specified in the last two rows. Right: final deformed scene. Note that we preserve the initial object placement layout: objects in contact remain so during deformation, as is the case with the trees and ground, or the knives and boxes. When there are no contacts, such as the small birds in the third row, the flow curves automatically become position constraints.*

To evaluate the efficiency of our tool, we invited two artists with more than 7 years of professional experience. We asked them to deform 5 scenes using their favorite software (Maya) in order to create a personalized movement. Then they used our flow curves to create similar deformations on the same initial scenes. A visualization of this evaluation is shown in Fig. 3.10.

Both artists were impressed by the ease of use of our tool and appreciated the direct use (i.e. no manual setup). Table 3.1 compares the times taken both in Maya and using our Flow curves, as well as the required number of mouse clicks: our interface is on average 6 times faster and require 8 times less clicks. Flow Curves was manifestly much more intuitive and faster than deformer available in Maya, however it did not offer the same level of control. In fact, both artists mentioned that they would like to use Flow Curves as a fast and natural way to deform their scenes, and to then use additional deformer if more detailed refinements are needed.

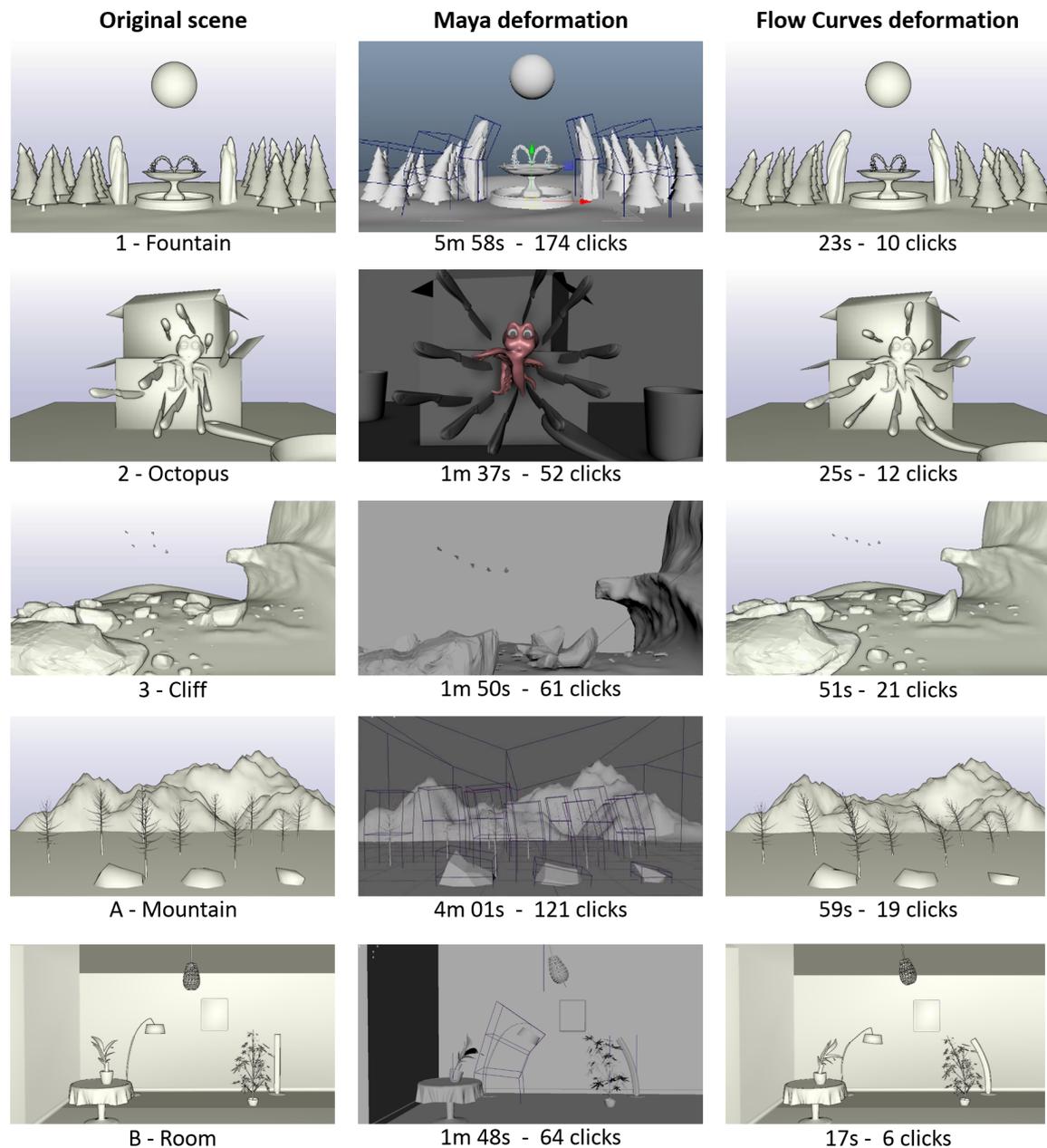


Figure 3.10: This figure shows results generated by Artist 1 in our user study. The five scenes in the left column were deformed by the artist using both Maya (middle column) and our Flow Curves interface (right column). Under each deformed image, we provide the corresponding time and number of mouse clicks required. Note that for scenes 3 and A, the artist has added his own SEcurves to the ones automatically computed.

Scene	Artist 1				Artist 2			
	Time		Mouse clicks		Time		Mouse clicks	
	Maya	Ours	Maya	Ours	Maya	Ours	Maya	Ours
1	5m 58s	23s	174	10	5m 02s	26s	138	12
2	1m 37s	25s	52	12	2m 22s	26s	103	13
3	1m 50s	51s	61	21	2m 07s	33s	75	19
A	4m 01s	59s	121	19	5m 06s	37s	144	13
B	1m 48s	17s	64	6	3m 10s	14s	92	6

Table 3.1: Comparison of the time and mouse clicks required by professional artists to deform the same scenes in Maya and with Flow Curves.

3.7 Conclusion

We introduced a new sketch-based interface called Flow Curves that provides intuitive ways of deforming whole scenes for visual coherence. With our interface, the user can more directly and efficiently design and edit the movement of a scene, simply by sketching flow curves, or sketching a circle center-point. Our direct deformation method requires little to no manual setup time and preserves the intrinsic shape of objects together with the initial objects placement layout. We demonstrated that our approach takes considerably less time to operate than current modern 3D digital tools, which require setting up diverse deformers and performing multiple edits from different view-points.

In our effort to offer a fully automatic interface that works on raw scene geometry with little manual setup, we made assumptions about the user’s intentions. In particular, we automatically compute the correspondence between SEcurves and flow curves (Section 3.5.2), but this may not reflect the expectation of the user (he or she may only want to modify closer SEcurves, or maybe sketch larger deformations). This issue is easily removed with manual intervention, such as by sketching over the desired SEcurve portions that one would like to modify. Also, when computing SEcurves in the scene, the performance of our algorithm depends on the segmentation of the scene. For example, the fence in Fig. 3.5 is one object and we obtain an outer-contour, while if it was segmented into several objects, a principal curve would be obtained for each individual pole.

We believe that our characterization of flow curves as constraints over fea-

ture subjective curves in the scene opens new opportunities beyond scene deformation. For instance, the same constraints could be used to guide the modeling or lighting of a scene. For example, a procedural modeling algorithm could use our constraints to generate objects whose SEcurves match user-specified flow curves through shading control.

In this Chapter, we have focused on static scenes. However, moving objects can form subjective curves over time due to visual persistence — our mind keeps track of previous positions and traces imaginary paths over time. That is why in the following Chapters we investigate dynamic scenes and we provide space-time curve representations that improve the visualization and control of animations.

Tangent-Space Optimization for Interactive Animation Control

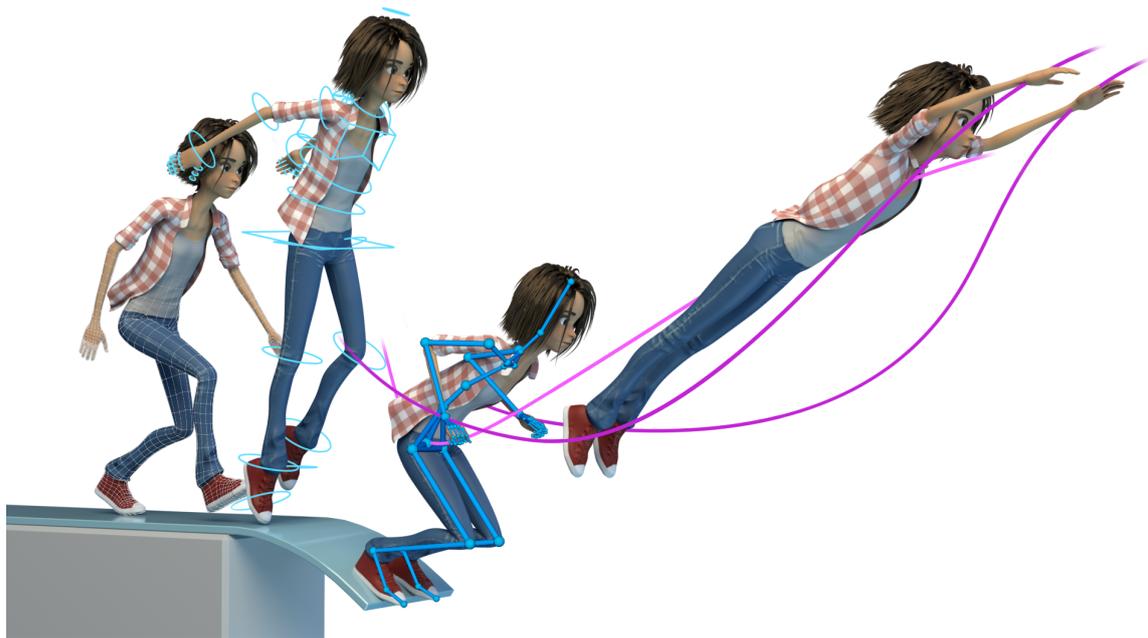


Figure 4.1: *Left: In traditional animation the character’s deformations are driven by rig controls (light blue), which provide fine control but require a granular interaction. Middle: Our system provides an armature (dark blue) that coordinately drives the rig elements for a more flexible character manipulation. Right: we introduce a curve representation (purple) for easily controlling the interpolation between key poses without adding any keyframe.*

Extending the principle of movement to dynamic elements would mainly refer to the animation principle of *arcs*. An arc is the visual path that an object or action draws over time. It is an important tool to make movements realistic and pleasing to the eye. Arcs also participate in the signature of an object or character; for example, a robot arm travels on an angular path while a dancer's arm moves in a circular fashion. However, the keyframing approach discussed in Section 1.4 provides a very indirect control on those arcs. While contemporary 3D software offer many controls and methods for posing, the tools available to adjust the result of the automatic in-betweening are comparatively rudimentary. Artists must manually manipulate the parameters of interpolation splines for individual animation variables. In this pose-centric view, the least burdensome way to adjust the interpolated movement and timing is to add additional keyframes, which displeasingly complicates the parametrization of arcs and animation curves. As a result, artists spend a tremendous amount of time posing and setting keyframes in order to achieve a high-quality animation.

For articulated movement, the situation is further complicated. The artist is given the choice of either manipulating *angles* with forward kinematics (FK) or *positions* with inverse kinematics (IK). However, the choice made when posing the character will also determine the nature of the interpolation between these poses: FK results in smooth arcs while IK produces linear movement, especially useful for contact points. This forces the artist to carefully plan when to use FK and IK. As mentioned earlier, FK arcs cannot be controlled other than with the rudimentary manipulation of individual spline parameters. It is also the case for IK, with the additional limitation that only identical kinematic configurations can be interpolated since existing systems assume that IK chains have the same bases and end-effectors. This forces riggers to define fixed IK chains on the character and prevents state-of-the-art techniques in flexible IK definition to be used in a production environment. In addition, IK-based controls slow down the rig evaluation, so a limited set is typically placed on the characters, which restricts the freedom of interaction. For example, artists rarely put IK on fingers due to the complexity of the hand structure.

We address these problems with a novel tangent-space optimization framework and a temporal interface for articulated movement that allows artists to intuitively adjust in-betweenings (Fig. 4.2). The key feature of our technique is that the optimization solves for the tangents of interpolation at existing keyframes. As a result, it does not add any complexity to the animation curves and is non-intrusive to the artist workflow. The optimization supports user-provided or character-implied constraints such as joint angle limits and stiffness for more natural deformations. Our system works

on top of typical character controls and does not add other complexities or structures to drive the animation. Furthermore, our method provides an abstraction on top of posing and animation that completely eliminates the need for fixed IK chains.



Figure 4.2: *Left: Traditional interface for editing interpolations — the artist must manually edit the tangents of multiple attributes’ animation curves. Right: Our interface — the artist directly manipulates the trajectory of an element, and our solver optimizes for the tangents that match the user inputs.*

We illustrate how our framework improves current animation pipelines with various examples and use-cases with both professional artists and novice users. In summary, we make the following contributions:

- A formulation of interpolation as optimization in the tangent space of rig controls with given positional, joint angle, and stiffness constraints. This leads to real-time interpolation control, which enables artists to work with fewer keyframes and cleaner animation curves.
- An interactive interface that utilizes the optimization framework and allows editing space-time curves for precise locations of controls during interpolations. This leads to a more fluid and natural interaction than with merely adjusting animation curves for transformations.
- A system that alleviates the need of fixed IK chains for interpolation, and thus unlocks the use of state-of-the art posing techniques in production environments.

4.1 Background

We have seen in Section 2.2.1 that many works proposed to ease the posing process. It started with the introduction of inverse kinematics where the user can specify positional constraints of end effectors at keyframes. They are then interpolated provided that the constraints configuration does not change (same bases and end effectors). Further works explored different interfaces for crafting poses, as already exposed. However, none of those techniques tackles the challenge of controlling the interpolation between the crafted poses. Their methods modify the FK values of the character controls, but since FK arcs are not editable — such as to define contacts — it makes those techniques impractical in a professional environment.

Our system provides a fine control on interpolated trajectories, including FK-driven ones, which finally permits the integration of such advanced posing techniques in a professional workflow. Previous animation interfaces based on space-time curves manipulation [Guay et al., 2015; Choi et al., 2016] usually exploited the IK chains present on the rig, which limits the set of trajectories that can be edited on the character. With our technique, the user has the freedom to manipulate the trajectory of any joint, or even between the joints, with the additional benefit that no keyframe will be added to the animation.

4.2 Approach

4.2.1 Problem Formulation

Our system starts with any rig structure. Some examples are graphs of controls for rotation and translation, more complex rigs with advanced controls, or a skeletal structure with joints (Fig. 4.3). From there, the user can directly start animating either with traditional tools, or with our system.

Once some keyframes are set, the *Motion Curves* corresponding to a point's path in time can be visualized (Fig. 4.4, left) by clicking on any point on the control structure. The Motion Curve also displays orientations (middle) and spacing of frames, i.e timing (right). The trajectory can be altered by dragging a point on the Motion Curve corresponding to the location of the chosen point at a particular frame. Its orientation and timing can also be controlled by rotation, and scaling to change the spacing of frames in time, respectively. In this paper, we define *state* as the positional and orientational configuration of a point at a specific time. Alterations specified by the user on a point's



Figure 4.3: *Left: our generic abstraction of character controls. It represents the structure that the artist is animating, which could be of any type: skeleton (middle), rig controls (right) or others.*

state are combined with further constraints, such as pinned points (motion curves that the user does not want to be altered), and contact points, in order to craft the resulting motion interactively. We show a step of an example editing process in Fig. 4.5. In this example, the user sets the right hand as contact, pins the trajectory of the hip, and manipulates the trajectory of the left arm as represented with a Motion Curve.

Although the manipulations and constraints described above give the user enough degrees of freedom and precise control, we observed that it is often difficult to get character specific deformations as all the above controls are agnostic to the character being animated. This is a common problem also in previous FK/IK based systems. We thus propose to further impose character specific properties. Angle limits are set in order to forbid undesired configurations (such as turning the elbow backward). Angle stiffness defines the resistance of joints to rotations; for example, on a human character, artists would usually set a lower stiffness on the shoulder than on the clavicle because the latter is less involved in arms' movements.

As in most animation systems, we assume that each control in the underlying rig structure is parameterized, and each parameter is controlled by a different animation curve (Fig. 4.6). Although our formulation supports arbitrary animation curves, we will assume cubic Bezier curves, since they are heavily used in practice. Once an artist defines keyposes, each parameter of

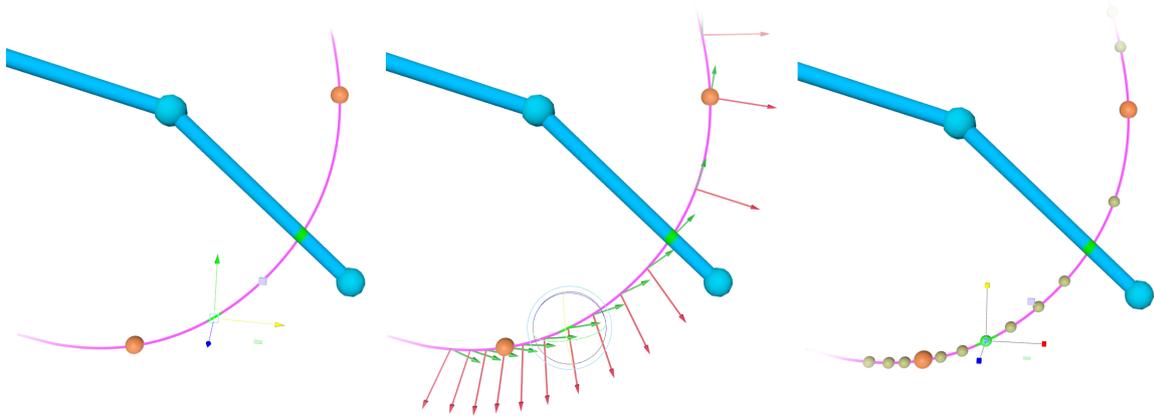


Figure 4.4: Motion Curve representation that allows to edit the trajectory (left), orientations (middle) and timing (right) during interpolations. The orange spheres represent keyframes, i.e. points that will not be altered by the modifications.

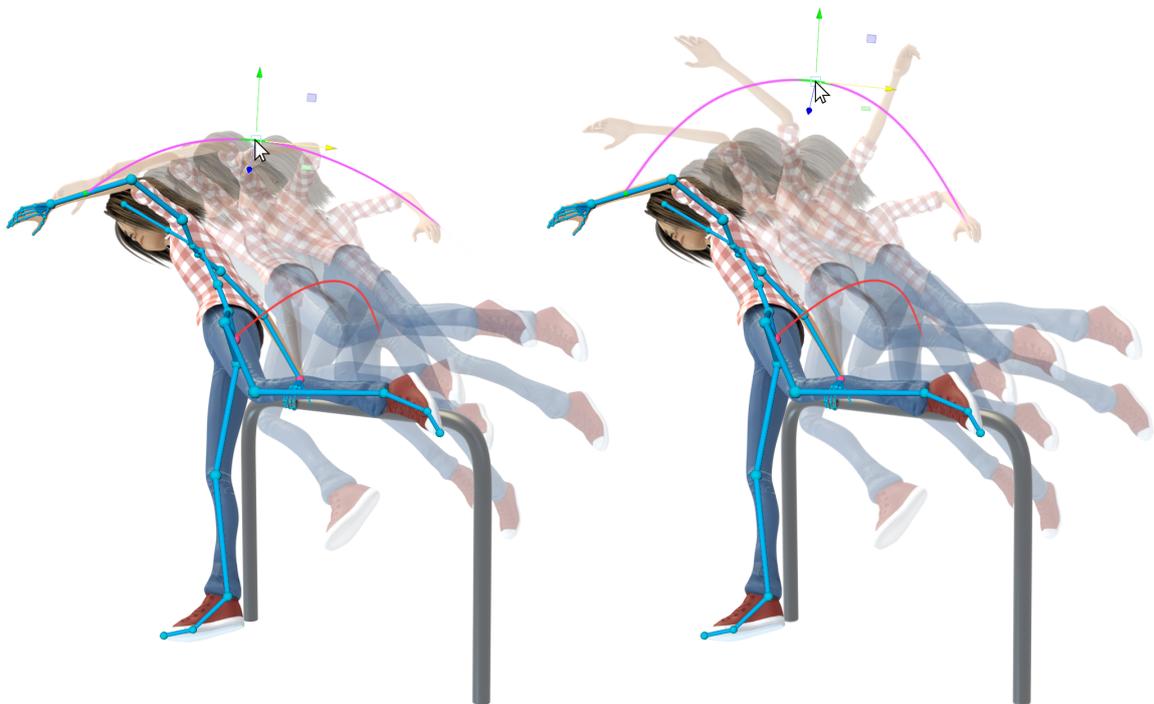


Figure 4.5: Manipulation of the interpolation. The user edits the Motion Curve of the left arm, while ensuring that the trajectory of the hip (red curve) and the right hand (which is a contact) will not be altered.

each controller is thus interpolated with a cubic Bezier curve that defines the motion. Using traditional systems, if the resulting animation is not satisfactory, editing the character motion requires to manually edit the tangents of every Bezier curve, a cumbersome process that requires a lot of effort and time to get the desired animation.

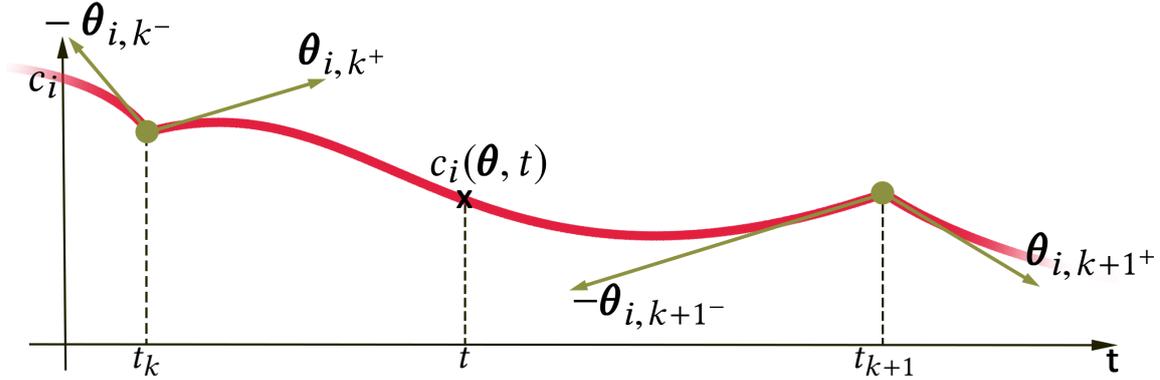


Figure 4.6: Illustration of the introduced variables.

We denote all tangents of all such animation curves with the vector θ , and all animation curves with $\mathbf{c}(\theta, t)$. At each editing step, the user can click on a point on the control structure, and change the parameters (position, orientation, and scale) at that point interactively. We denote the state of this point with $\mathbf{s}(\mathbf{c}(\theta, t))$, and the new one interactively specified by the user with $\mathbf{s}'(t)$. The goal is then to compute a new set of tangent values θ' in order to reach $\mathbf{s}'(t)$ (i.e. we want $\mathbf{s}(\mathbf{c}(\theta', t)) = \mathbf{s}'(t)$) while satisfying other constraints, as we will detail in the next Section.

4.2.2 Tangent Space Optimization

As we would like to formulate the problem in terms of the tangents θ , we first express all changes, and in particular $\Delta\mathbf{s}(\mathbf{c}(\theta, t)) := \mathbf{s}(\mathbf{c}(\theta', t)) - \mathbf{s}(\mathbf{c}(\theta, t))$ in terms of changes in the tangents. This can be easily carried out by a first order approximation:

$$\begin{aligned} \mathbf{s}(\mathbf{c}(\theta', t)) &\approx \mathbf{s}(\mathbf{c}(\theta, t)) + \mathbf{J}_s(\mathbf{c}(\theta, t))(\theta' - \theta) \\ \Delta\mathbf{s}(\mathbf{c}(\theta, t)) &\approx \mathbf{J}_s(\mathbf{c}(\theta, t))\Delta\theta, \end{aligned} \quad (4.1)$$

where \mathbf{J}_s is the Jacobian matrix that stacks the derivatives of \mathbf{s} with respect to the tangents θ . This is the starting point of most IK formulations, and in our case a well justified approximation as the user will incrementally alter the Motion Curves. The Jacobian can be further factorized into

$$\mathbf{J}_s(\mathbf{c}(\theta, t)) = \frac{\delta\mathbf{s}(\mathbf{c}(\theta, t))}{\delta\mathbf{c}(\theta, t)} \frac{\delta\mathbf{c}(\theta, t)}{\delta\theta}. \quad (4.2)$$

This form of \mathbf{J}_s is useful as the first term on the right hand side is typically simple. Denoting the i th component of \mathbf{c} with c_i , and the position and orientation of \mathbf{s} with \mathbf{s}_P and \mathbf{s}_O , respectively, if c_i defines a translation, $\frac{\delta \mathbf{s}_P}{\delta c_i} = \mathbf{v}$ and $\frac{\delta \mathbf{s}_O}{\delta c_i} = \mathbf{0}$, or if c_i defines a rotation, $\frac{\delta \mathbf{s}_P}{\delta c_i} = \mathbf{v} \times (\mathbf{s}_P - \mathbf{r})$ and $\frac{\delta \mathbf{s}_O}{\delta c_i} = \mathbf{v}$, where \mathbf{v} is the unit vector pointing along the translation or rotation axis, and \mathbf{r} is the rotation center. The second term in the Jacobian factorization is more involved, and often not possible to get analytically. We thus approximate it with finite differences.

For simplicity, we will drop \mathbf{c} and simply write $\mathbf{s}(\boldsymbol{\theta}, t)$ and $\mathbf{J}_s(\boldsymbol{\theta}, t)$ for the rest of this Section.

Energy terms

Given this linear approximation, we can now define three important quadratic energies. The first one aims at making the solution $\mathbf{s}(\boldsymbol{\theta}', t)$ stay as close as possible to the target $\mathbf{s}'(t)$ interactively provided by the user on the Motion Curve (as in Fig. 4.5):

$$E_m = \|\Delta \mathbf{s}'(t) - \mathbf{J}_s(\boldsymbol{\theta}, t) \Delta \boldsymbol{\theta}\|^2, \quad (4.3)$$

where $\Delta \mathbf{s}'(t) = \mathbf{s}'(t) - \mathbf{s}(\boldsymbol{\theta}, t)$. Next, to avoid abrupt deformations when manipulating a trajectory, we add an energy term to minimize the changes in tangents:

$$E_d = \|\mathbf{D} \Delta \boldsymbol{\theta}\|^2. \quad (4.4)$$

Here, \mathbf{D} is a diagonal matrix that stores stiffness parameters per tangent component, as defined in Section 4.2.1. Finally, the user can break some tangents, meaning that the animation curve for certain parameters can consist of multiple segments of Bezier curves that are not C^1 . For such cases, we add an energy term that forces the tangents at consecutive segments to stay close in order to have an as smooth as possible curve:

$$E_b = \|\mathbf{T}^+ \boldsymbol{\theta} - \mathbf{T}^- \boldsymbol{\theta}\|^2, \quad (4.5)$$

where the matrices \mathbf{T}^+ and \mathbf{T}^- select tangent pairs corresponding to the same keyframe of the same curve, but on different Bezier segments.

Pins and contact constraints

In addition to the interactively modified Motion Curves, the user can specify pins and contact constraints on the structure. This is important to achieve

and interpolate the poses as desired. The pinned points' trajectories and orientations — i.e. their state denoted by $\mathbf{s}_j(\boldsymbol{\theta}, t)$ — should not be altered by the modifications on $\boldsymbol{\theta}$. Therefore, we have the desired states $\mathbf{s}'_j(t) = \mathbf{s}_j(\boldsymbol{\theta}, t)$, at all times t . We sample the time into a set of frames, which gives us a constrained state per pin and per frame, that we stack into the vector $\boldsymbol{\rho}(\boldsymbol{\theta})$. The target states for these points are similarly stacked into $\boldsymbol{\rho}'$. We thus require that $\boldsymbol{\rho}' = \boldsymbol{\rho}(\boldsymbol{\theta})$. We elaborate on the sampling of time in Section 4.2.3.

Other points can be set as contact between two keyframes t_k and t_{k+1} , e.g. feet on the floor. This means that those points should not move between the two specified keyframes. Specifically, we have $\mathbf{s}_j(\boldsymbol{\theta}, t_k) = \mathbf{s}_j(\boldsymbol{\theta}, t_{k+1})$, and want $\mathbf{s}'_j(t) = \mathbf{s}_j(\boldsymbol{\theta}, t_k)$, $\forall t \in [t_k, t_{k+1}]$. Once again, we sample the time between t_k and t_{k+1} and append the set of constrained states $\mathbf{s}_j(\boldsymbol{\theta}, t)$ to the vector $\boldsymbol{\rho}(\boldsymbol{\theta})$, and the set of corresponding desired states $\mathbf{s}'_j(t)$ to $\boldsymbol{\rho}'$.

Finally, the state of the manipulated point can also be constrained at some specific time frames by selecting them directly on the Motion Curve. Similar to pins, at those times t_l we have $\mathbf{s}'(t_l) = \mathbf{s}(\boldsymbol{\theta}, t_l)$. We also append those constrained and desired states to $\boldsymbol{\rho}(\boldsymbol{\theta})$ and $\boldsymbol{\rho}'$.

All the aforementioned pins, contacts, and state constraints then define hard constraints in our optimization problem. Noting that $\boldsymbol{\rho}' = \boldsymbol{\rho}(\boldsymbol{\theta}')$ for some $\boldsymbol{\theta}'$, we can use the linear approximation given in Equation 4.1 to get

$$\Delta\boldsymbol{\rho}' - \mathbf{J}_\rho(\boldsymbol{\theta})\Delta\boldsymbol{\theta} = \mathbf{0}, \quad (4.6)$$

where $\Delta\boldsymbol{\rho}' = \boldsymbol{\rho}' - \boldsymbol{\rho}(\boldsymbol{\theta})$, and $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}' - \boldsymbol{\theta}$, as before.

Variables limits

Let us define $\boldsymbol{\theta}_{i,k}$ as the tangent for parameter c_i at keyframe t_k . If the tangent is broken, we respectively call $\boldsymbol{\theta}_{i,k^-}$ and $\boldsymbol{\theta}_{i,k^+}$ the tangents corresponding to the Bezier segments before and after t_k . Each of these tangents is composed of two components $\boldsymbol{\theta}_{i,k} = \begin{pmatrix} \theta_{i,k}^X \\ \theta_{i,k}^Y \end{pmatrix}$. The X component (horizontal expansion of tangent in Fig. 4.6) determines the timing of the interpolation. In order to ensure that the spline interpolation is injective, i.e. at each time t there is only one value of $c_i(t)$, we limit this component to:

$$\begin{aligned} 0 &\leq \theta_{i,k^+}^X \leq (t_{k+1} - t_k) \\ 0 &\leq \theta_{i,k^-}^X \leq (t_k - t_{k-1}). \end{aligned} \quad (4.7)$$

The Y component (vertical expansion) determines the range of values that $c_i(t)$ takes. In order to limit it within the range u_i to v_i , we have to restrict

the Y component of the tangents. This is important especially when $c_i(t)$ represents joint angles (see angle limits as defined in Section 4.2.1). We can then impose the following limits on the tangents:

$$\begin{aligned} \phi(u_i, k^+) &\leq \theta_{i,k^+}^Y \leq \psi(v_i, k^+) \\ -\psi(v_i, k^-) &\leq \theta_{i,k^-}^Y \leq -\phi(u_i, k^-). \end{aligned} \quad (4.8)$$

Depending on the type of interpolation, the functions ϕ and ψ can have complex expressions, or even have no closed form. It is the case for Bezier interpolations, for which we propose the following approximation:

$$\begin{aligned} \phi(u_i, k^\pm) &= \frac{4}{3}(u_i - \min(c_i(t_k), c_i(t_{k\pm 1}))) \\ \psi(v_i, k^\pm) &= \frac{4}{3}(v_i - \max(c_i(t_k), c_i(t_{k\pm 1}))). \end{aligned} \quad (4.9)$$

We demonstrate in Appendix A that this approximation satisfies the limits u_i and v_i for $c_i(t)$ at all times.

Quadratic problem formulation

Given the energies, constraints and limits defined above, we finally obtain the following minimization problem:

$$\begin{aligned} \min_{\Delta\theta} \quad & w_m \cdot E_m + w_d \cdot E_d + w_b \cdot E_b \\ \text{subject to} \quad & \Delta\rho' - \mathbf{J}_\rho(\theta)\Delta\theta = \mathbf{0} \\ & 0 \leq \theta_{i,k^+}^X \leq (t_{k+1} - t_k) \\ & 0 \leq \theta_{i,k^-}^X \leq (t_k - t_{k-1}) \\ & \phi(u_i, k^+) \leq \theta_{i,k^+}^Y \leq \psi(v_i, k^+) \\ & -\psi(v_i, k^-) \leq \theta_{i,k^-}^Y \leq -\phi(u_i, k^-) \end{aligned} \quad , \forall i, \forall k. \quad (4.10)$$

This is a quadratic programming problem that we solve using the Mosek solver [Mosek, 2010] with default parameters. In our implementation, we chose $w_m = 100.0$, $w_d = 1.0$ and $w_b = 1.0$ — here, the high value of w_m reflects the prevailing importance given to satisfying the user manipulations.

4.2.3 Implementation Details

Tangents reduction

We can drastically reduce the size of the optimization by realizing that it is not required to optimize for the entire set of tangents θ . Indeed, many tangent modifications will not affect the state of the manipulated point or of the

constrained ones (pins and contacts), so they do not need to be considered. We denote \mathcal{C}_s the set of all parameters that affect the state \mathbf{s} of the manipulated point, and \mathcal{C}_{s_j} the equivalent set for the state \mathbf{s}_j of each constrained point. With hierarchical structures such as skeletons, those sets correspond to the parameters of the point's parent chain. We further denote the set of parameters whose tangents are actually optimized in Equation 4.10 with \mathcal{C} .

We begin by setting $\mathcal{C} = \mathcal{C}_s$. Then, for each constrained point, we identify three cases: (1) if $\mathcal{C} \cap \mathcal{C}_{s_j} = \emptyset$, we ignore the constraint and remove it from the vector $\boldsymbol{\rho}$ because no modification on s will affect s_j ; (2) if $\mathcal{C}_{s_j} \subset \mathcal{C}_s$, we remove the constraint from $\boldsymbol{\rho}$ and reduce the set $\mathcal{C} := \mathcal{C} \setminus \mathcal{C}_{s_j}$; (3) otherwise, we keep the constraint and augment the set $\mathcal{C} := \mathcal{C} \cup \mathcal{C}_{s_j}$. Finally, if a point in the time range $t \in [t_k, t_{k+1}]$ is manipulated, only the tangents corresponding to that time range need to be modified. In conclusion, the tangents that we optimize for in Equation 4.10 are: $(\boldsymbol{\theta}_{i,k^+}, \boldsymbol{\theta}_{i,k+1^-})$, $\forall i \in \mathcal{C}$.

For the example in Fig. 4.5 where the left arm is manipulated while the hip and hand are pinned, our solver optimizes for 60 tangents, while the total number of tangents present in this scene, supposing that the entire animation contains only 3 keyframes, is 556. This optimization allows to achieve a real-time interaction, as presented in Section 4.3.5.

Time sampling

Pinning or setting contact points \mathbf{s}_j means imposing a constraint over a whole time period $[t_k, t_{k+1}]$. To do so, we sample the time. The set of samples is at most at every frame of the animation between t_k and t_{k+1} , but we can reduce its size in case $t_{k+1} - t_k$ is large or \mathcal{C}_{s_j} is small. Indeed, there is a limited number of degrees of freedom in the spline animation curves that define the state \mathbf{s}_j , therefore we only need to sample the time range that number of times, which is: $4 \times \text{size}(\mathcal{C}_{s_j})$.

Optimization and overconstrained cases

Most of the time, since the manipulations are incremental, one single iteration of the optimization is sufficient to satisfy the constraints. However sometimes it might not be enough, in which case we update the state and Jacobian values and run the optimization again — this happens if $\mathbf{s}'(t)$ is far from $\mathbf{s}(\boldsymbol{\theta}, t)$ and the linear approximation of Equation 4.1 is not accurate anymore. Our stopping criteria is that the manipulated point's state is close enough to the user given state, i.e. $\|\mathbf{s}(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}, t) - \mathbf{s}'(t)\| < \epsilon$.

If after several iterations (we used 5) the solution is not improved — i.e. the curve does not move closer to the user-specified point — we reject it and stay at the current configuration. This can happen when the problem is over constrained, i.e. the set of constraints (pins, contacts and manipulated point) are unreachable given the set of degrees of freedom (tangents). When the user comes back to a reachable solution, the state is updated, which provides direct feedback about the feasibility of the manipulations.

Solver stability

In order to improve the stability of the quadratic programming solver, we allowed a threshold on the hard constraint corresponding to pins and contacts in Equation 4.6. It becomes:

$$-\boldsymbol{\epsilon} \leq \Delta\boldsymbol{\rho}' - \mathbf{J}_\rho(\boldsymbol{\theta})\Delta\boldsymbol{\theta} \leq \boldsymbol{\epsilon}, \quad (4.11)$$

where $\boldsymbol{\epsilon}$ is a vector of ϵ values that affect the extent on which constrained points will be able to move around their desired state $\boldsymbol{\rho}'$. It can be chosen depending on the scale of the scene in order for the error to stay barely visible to the user. We used $\epsilon = 10^{-4}$.

The optimization problem being quadratic, we can rewrite Equation 4.10 in the form $\frac{1}{2}\Delta\boldsymbol{\theta}^T \cdot \mathbf{Q} \cdot \Delta\boldsymbol{\theta} + \mathbf{b}^T \cdot \Delta\boldsymbol{\theta} + c$. Jittery solutions can be obtained if \mathbf{Q} is not full rank. We avoid such cases by regularizing \mathbf{Q} with $\mathbf{Q} := \mathbf{Q} + \lambda\mathbf{I}$, with $\lambda = 10^{-6}$.

4.2.4 Timing Manipulations

Another important aspect to consider when manipulating interpolations is the timing. The extent of tangents in the X direction, $\theta_{i,k}^X$, influences the ease-in and ease-out of interpolations. We propose to let the user edit this easing directly on the Motion Curve, by scaling the timing up or down at any point (see Fig. 4.4, right). Scaling up means moving the frames apart around that point, i.e. making the motion faster, and scaling down means bringing the frames closer around that point, i.e. making the motion slower. If the user is scaling at time $t \in [t_k, t_{k+1}]$ by a factor σ , we modify the tangents as follows:

$$\begin{aligned} \theta_{i,k+}^X &:= \theta_{i,k+}^X + \sigma \frac{t - t_k}{t_{k+1} - t_k} \\ \theta_{i,k+1-}^X &:= \theta_{i,k+1-}^X + \sigma \frac{t_{k+1} - t}{t_{k+1} - t_k}, \end{aligned} \quad (4.12)$$

while still limiting the values to stay between 0 and $(t_{k+1} - t_k)$ in order to keep an injective function.

4.2.5 Static Case

It is interesting to notice that the system we introduce for the control of interpolations can be reduced to a posing system, similar to what we find in the literature [Yamane and Nakamura, 2003; Shi et al., 2007], if we remove time from the equations. Indeed, $\mathbf{c}(\boldsymbol{\theta}, t)$ simply becomes \mathbf{c} , meaning that we directly optimize for attributes' values instead of tangents. The minimization problem of Equation 4.10 then becomes:

$$\begin{aligned} \min_{\Delta \mathbf{c}} \quad & w_m \cdot \|\Delta \mathbf{s} - J_s(\mathbf{c})\Delta \mathbf{c}\|^2 + w_d \cdot \|\mathbf{D}\Delta \mathbf{c}\|^2 \\ \text{subject to} \quad & \|\mathbf{J}_\rho(\boldsymbol{\theta})\Delta \boldsymbol{\theta}\|^2 = 0 \\ & u_i \leq c_i \leq v_i, \quad \forall i, \end{aligned} \quad (4.13)$$

where $\boldsymbol{\rho}$ is the vector of pinned (i.e fixed) points and the Jacobian $J_s(\mathbf{c})$ is given by:

$$J_s(\mathbf{c}) = \frac{\delta \mathbf{s}(\mathbf{c})}{\delta \mathbf{c}}. \quad (4.14)$$



Figure 4.7: *The user is manipulating the character's left arm for a static pose. The feet and right hand are pinned, ensuring that they do not move during manipulations.*

The obtained system provides similar interaction abilities as our interpolation control, always with angle limits and stiffnesses (Fig. 4.7). This shows that our problem formulation can be generalized to cover a wide range of the animation pipeline, and that it completely removes the need for fixed IK chains in the rig. It is the system we use in Section 4.3 to design the keyposes of our results.

4.3 Evaluation

4.3.1 Examples of Authoring Difficult Animations

We implemented our system as an Autodesk Maya plugin. To demonstrate its potential, we let two artists design several animations for different types of characters. While our method is designed to work in harmony with traditional tools, these animations were entirely authored using our system, from the design of key poses to the editing of interpolations.

Complex interactions of hands with objects, such as playing the guitar, are some of the most difficult animations to create due to fingers' particular gestures and contacts. Animating this case typically involves IK chains on the fingers, which drastically slows down the rig evaluation, further hindering the creation process. In contrast, creation becomes very natural using our system, as one can pin some fingers, and freely move any other part of the character. Contacts can be specified for certain time ranges to ensure that the fingers won't move during interpolation while a note is being played. The animation presented in Fig. 4.8-left was created using our system without requiring any IK chain.

In a similar vein, the diving animation shown in Fig. 4.1 and Fig. 4.8-right necessitates contacts between the feet and the ground, which are easily handled with our system. Additionally, the interpolation for the jump requires a particular trajectory, with careful orientation and timing control, in order to achieve a realistic falling movement. Our system allows for a natural interaction with direct trajectory control for this case.

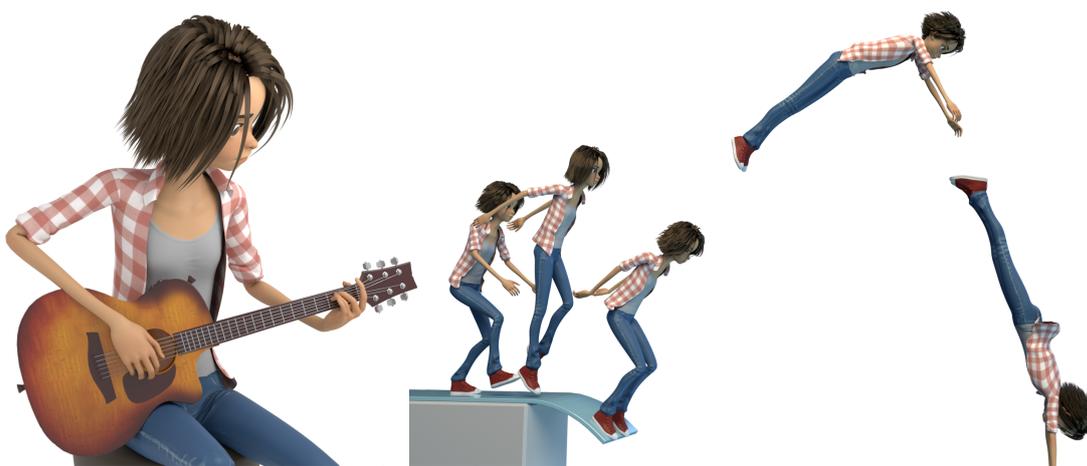


Figure 4.8: Results on a human character. Left: Playing guitar animation, which necessitates specific contacts on fingers. Right: Diving animation, requiring foot contacts and a believable falling trajectory.

Our system is not limited to human models. In Fig. 4.9, we show results with a spider robot and a dinosaur. The spider animation is a walk cycle that was created using only two key poses, demonstrating the effectiveness of the ability to control interpolations with tangents. The dinosaur punching animation is another example where the control of timing plays an important role. Furthermore, near the end of the animation, the dinosaur is standing on a part of his tail. Handling this contact with traditional tools would be a challenging task, especially since a classic rig would most likely not contain any IK chain with an end effector at the middle of the tail. These issues are effortlessly solved using our system.

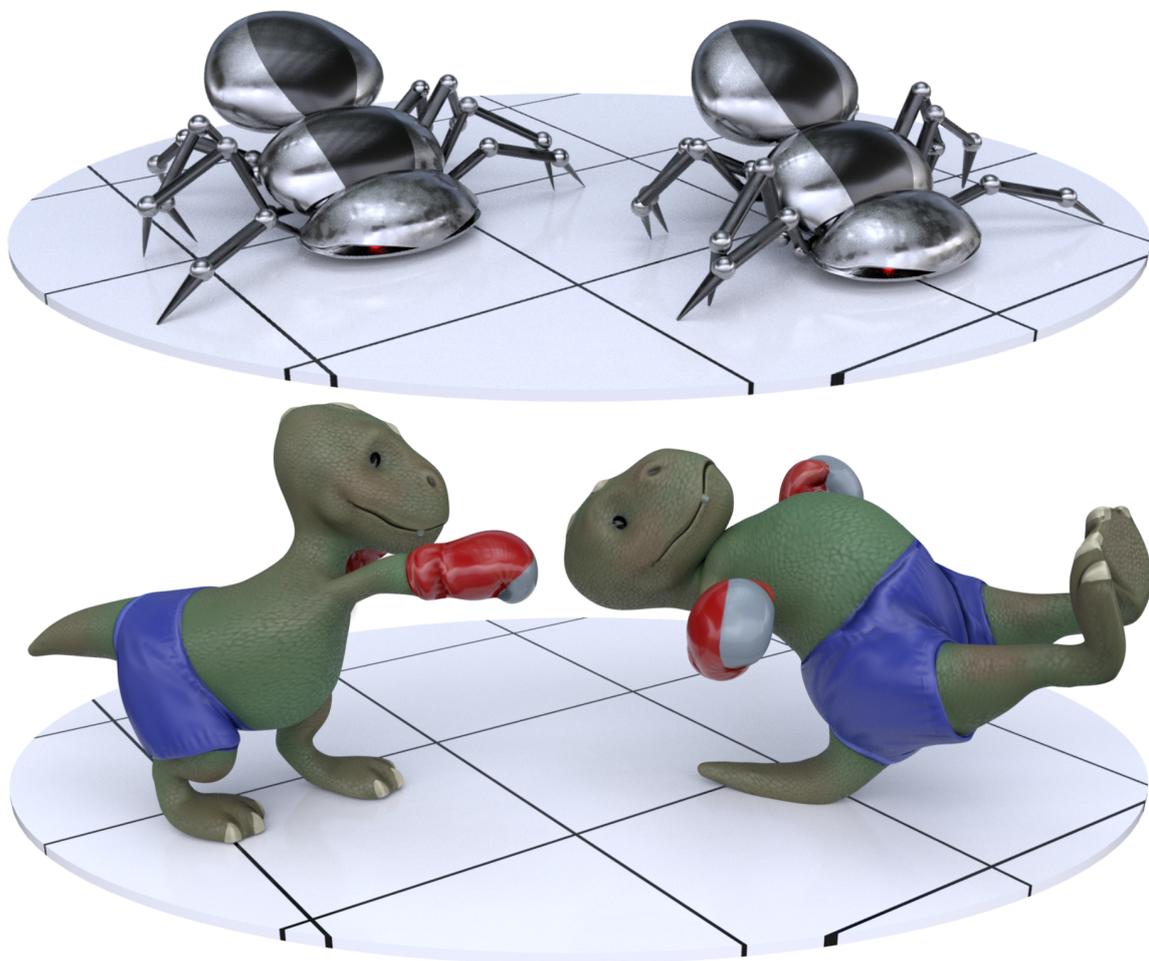


Figure 4.9: Results on non-human characters. Top: Walking animation of a spider robot, generated using only the shown two key poses. Bottom: Punching animation of a dinosaur, who switches his contact component with the ground from the feet to the tail.

4.3.2 User study: Simpler Curves for Complex Motions

One objective of our system is to help animators work with clean animation curves with the least amount of keyframes, making further editing as easy and fast as possible. As the animations are typically refined and altered many times in production, this is crucial for an effective workflow. To evaluate this aspect on complex animations, we asked two professional artists with a long experience using Autodesk Maya to animate scenes in Fig. 4.8 and Fig. 4.9 using traditional tools. For each animation, we counted the number of keyframes used, which are listed in Table 4.1. The number of keyframes naturally depends on the animation style, but for all cases we can clearly see that our system allows to animate using fewer keyframes (on average, 2.2 times less). Moreover, since with traditional tools artists require IK chains, the rig evaluation is considerably slowed down, further hindering interaction. In Table 4.1, we show that since our system does not require IK chains, the speed of the playback is on average 60% faster.

Animation	Num keyframes		Framerate	
	Traditional	Ours	Traditional	Ours
Human Guitar	37	13	20.0	32.8
Human Dive	24	10		
Spider Walk	4	2	44.9	82.1
Dino Punch	19	12	36.9	47.7

Table 4.1: For each animation, we compare the number of used keyframes when animating with a traditional system and with ours. Note that different controls might have different keyframes, so we choose the most keyframed one. We also compare the speed of rig evaluation in each case, in fps.

4.3.3 User Study: Faster Editing Process

To evaluate the effectiveness and accessibility of our system, we conducted a further user study. We invited 16 people — 6 artists and 10 novice users — who were presented with a simple robot flying animation composed of only 4 keyframes, where the character is hitting obstacles during interpolations. The objective of the exercise was to modify the animation in order to avoid those obstacles (see Fig. 4.10). Using Maya, they were free to use FK or IK, add keyframes, or edit the curves of interpolations in the graph editor. Using our tool, they only edited the motion curves in the viewport.

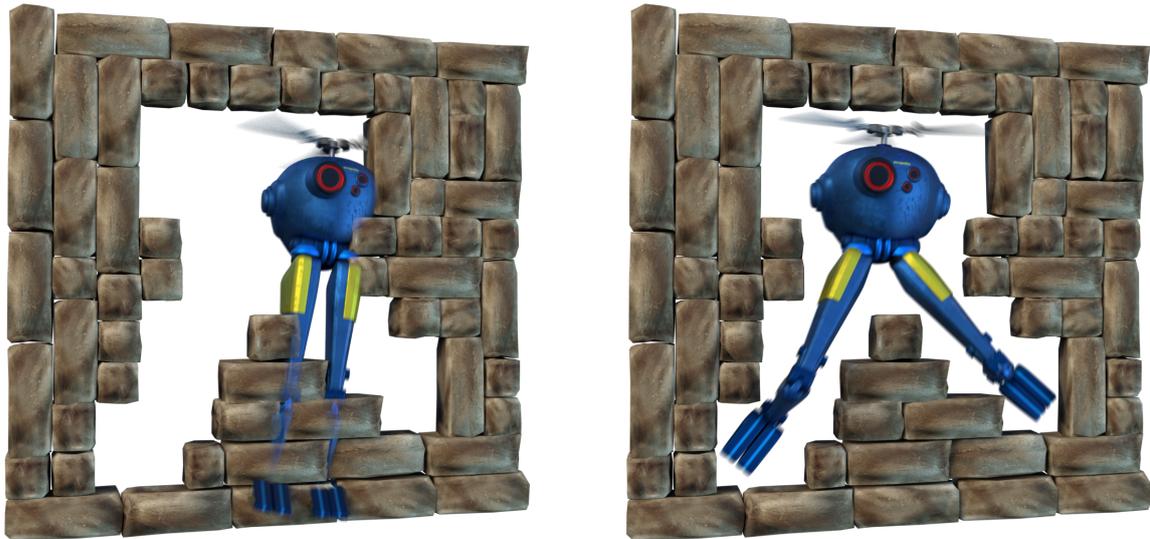


Figure 4.10: 16 users were asked to edit a robot flying animation (left) in order to avoid obstacles (right), first using traditional tools on Maya and then using our plugin.

Each user received a 10 minutes introduction to each system before starting. We measured the time, number of clicks, and keyframes to produce the final animations. Furthermore, each participant was asked to self-evaluate the quality of his/her results on a scale from 0 to 5. All numbers are presented in Table 4.2.

We observe that professional artists were able to edit the animation even more effectively with our system than with traditional tools they are trained on. Same goes for novice users, which demonstrates the accessibility of our system. On average, both the time and number of clicks were reduced by a third using our system. We also notice that this efficiency does not come at the cost of poorer animations. Indeed, our approach received significantly higher self-evaluation scores in all cases. This is partially due to the reduced number of keyframes, which yields a smoother motion. Note that only one user (novice #5) attempted to edit the curves of interpolation in Maya’s graph editor, a choice he eventually regretted. We believe that with a longer experience of animators on our tool, these gains will be further increased. We already observed this with Artist #1, who achieved the fastest edit with our tool since he was already trained by working on the animations presented in Section 4.3.1.

	Time		Clicks		Keyframes		Score	
	Maya	Ours	Maya	Ours	Maya	Ours	Maya	Ours
Artist #1	6m52s	2m28s	278	137	10	4	3.5	3.5
Artist #2	6m15s	4m32s	139	113	11	4	3.0	5.0
Artist #3	5m41s	4m27s	252	154	11	4	1.0	2.0
Artist #4	6m12s	5m11s	252	202	14	4	3.0	4.5
Artist #5	7m30s	5m27s	276	201	11	4	2.5	4.0
Artist #6	5m53s	4m33s	139	117	12	4	2.0	4.0
Novice #1	9m01s	6m48s	214	115	12	4	2.5	4.0
Novice #2	11m02s	8m52s	168	154	12	4	2.5	5.0
Novice #3	9m31s	5m29s	203	96	15	4	2.0	3.5
Novice #4	10m28s	5m03s	181	99	14	4	2.5	4.5
Novice #5	25m26s	7m19s	465	153	4	4	2.0	4.0
Novice #6	6m48s	4m52s	170	151	8	4	1.0	3.0
Novice #7	6m30s	6m54s	175	166	12	4	1.0	4.0
Novice #8	6m17s	5m52s	178	150	9	4	2.0	3.0
Novice #9	14m17s	12m39s	338	232	14	4	3.5	4.5
Novice #10	7m29s	6m46s	145	121	10	4	2.0	4.0

Table 4.2: For each participant who edited the flying robot animation using Maya and our system, we here compare the required time, the number of clicks, the number of keyframes and the self-evaluation score.

4.3.4 Qualitative Assessment from Professional Animators

We further conducted a survey and gathered observations on our system from professional animators, along the lines of Koyama and Goto’s evaluation [2018]. The artists were overall thrilled by the potential of the system, and stated that our approach was solving the current main shortcomings of the traditional animation systems, which they face everyday in production. They had a much more natural and efficient interaction with characters. Moreover, they really appreciated that our tool can be applied to characters without complex rig controls — by acting directly on the skeleton —, which eliminates one step of the animating process. The artists expressed a clear desire to see such a system available in their workflow.

As animators like to keep a relatively local control, we realized that they were reluctant to edit long chains or long interpolations in time. Also, their main reservation was about working with global keyposes — i.e. assuming that all elements are keyed at the same frames —, as they are used to working with different keyframes for different parts of a character, a strategy they use for e.g. follow-through effects. This could be easily solved computationally, as we develop in Section 4.4, but would be challenging to represent visually.

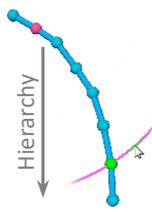
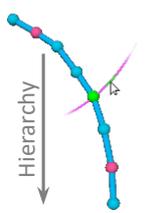
4.3.5 System Performance

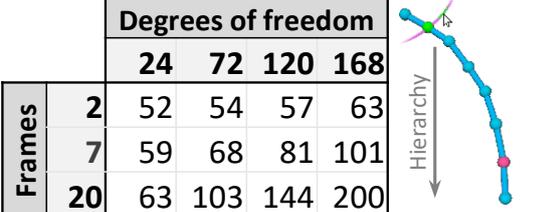
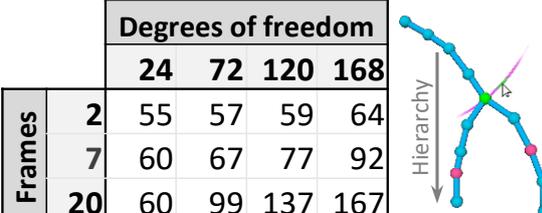
We tested the performance of our approach on a machine with an Intel Core i7-4930K CPU and 32GB of RAM (we do not specify the graphics card since we do not use any GPU optimization, neither does the optimization library we use in our system [Mosek, 2010]). Table 4.3 shows the computing times required to obtain the tangent values from user input, depending on the number of degrees of freedom (i.e. tangents), the length of the interpolation (i.e. spacing between two keyframes), and constraint configuration (number of pins and contacts, lower or upper in the hierarchy of character controls). Note that our system does not depend on the total length of the animation since we only consider a section of it during our optimization (as discussed in Section 4.2.3).

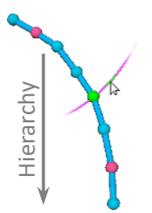
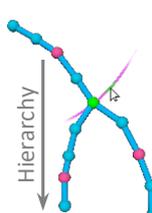
Given that artists rarely space their key poses by more than a few frames, and that most configurations do not exceed 100 degrees of freedom, we observe high enough frame rates for fluid interactions for typical animations in all cases. Some latency can be observed when a very large number of tangents are involved over a long interpolation. Even for such rare cases, we can maintain an interaction speed feasible for editing. Currently, creating the matrices of the QP problem — i.e. computing the Jacobians — represents on average 20% of the computing time in our tests, and this number goes up to 38% when a large number of DoFs and constraints are involved. Therefore, using a GPU optimization to parallelize the computation of all derivatives (from Equation 4.2) can significantly speed up the overall computation.

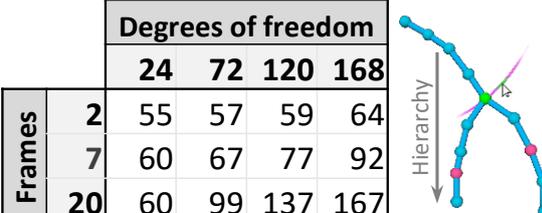
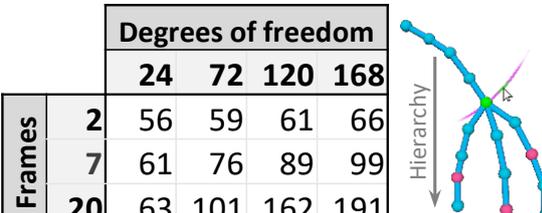
4.4 Conclusion

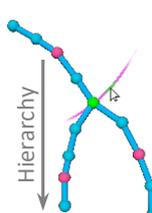
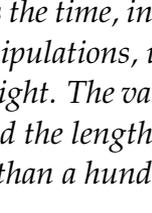
We proposed a new optimization-based keyframed animation system. Formulating common constraints and user interactions as an optimization problem in the tangent space of animation curves allowed us to handle the problem with fast quadratic programming based solvers. The result is an efficient real-time system that abstracts away the difficult choice of FK/IK from the

		Degrees of freedom				
		24	72	120	168	
Frames	2	10	12	16	21	
	7	10	13	16	21	
	20	10	12	16	20	

		Degrees of freedom				
		24	72	120	168	
Frames	2	52	54	57	63	
	7	59	68	81	101	
	20	63	103	144	200	

		Degrees of freedom				
		24	72	120	168	
Frames	2	52	57	69	86	
	7	58	80	94	130	
	20	67	106	158	227	

		Degrees of freedom				
		24	72	120	168	
Frames	2	55	57	59	64	
	7	60	67	77	92	
	20	60	99	137	167	

		Degrees of freedom				
		24	72	120	168	
Frames	2	53	59	63	68	
	7	63	80	91	113	
	20	65	103	144	187	

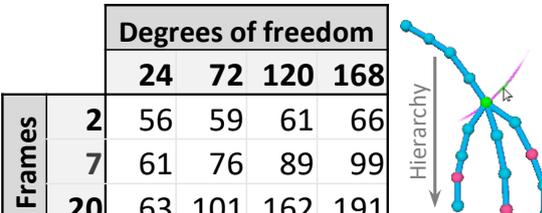
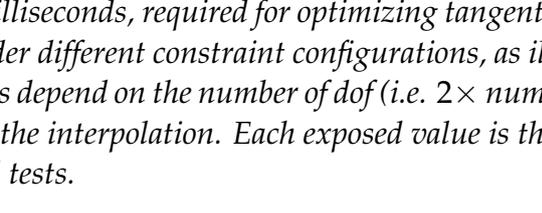
		Degrees of freedom				
		24	72	120	168	
Frames	2	56	59	61	66	
	7	61	76	89	99	
	20	63	101	162	191	

Table 4.3: Each table presents the time, in milliseconds, required for optimizing tangents to satisfy user manipulations, under different constraint configurations, as illustrated on their right. The values depend on the number of dof (i.e. $2 \times$ number of tangents) and the length of the interpolation. Each exposed value is the median over more than a hundred tests.

user, without adding keyframes or complicating animation curves. These properties make the proposed system practical and easy to incorporate into existing animation processes. We believe that with its stable and fast implementation, our system is an important addition to the current animation tools.

In our method, we assumed that animators use global keyposes. For certain animations, it might be more convenient to use different keyframes for different parts of a character. Computationally, this can be achieved by using different keyframes for different attributes when optimizing tangents in Equation 4.10 (i.e. attribute-dependent t_k values). However, the interaction might suffer from the inability to visualize clear segments of motion, and therefore yield unexpected behaviors when editing. We plan to develop alternative visualization strategies for these cases.

In this Chapter we have worked with connected structures such as skeletons

and body rig controls. But some structures, such as blendshape deformer on a facial rig, consist of a set of independent handles. Our system naturally extends to such cases, with a simplified optimization as only the tangents of that handle's attributes would need to be optimized. Visualization of motion curves would again be the main challenge for these cases.

Finally, it is possible that there is no solution for tangents in Equation 4.10 that would satisfy the user manipulations, especially if there are numerous contacts and pins specified. This means that there is no configuration of Bezier interpolations on joints' rotations that result in a certain trajectory for the end effector, while satisfying the constraints. For these cases, what we currently have is similar to that of IK systems: the optimization gives the closest possible solution. Alleviating this shortcoming, which comes from the limited degrees of freedom provided by the tangents, would require adding keyframes. However, this is not desired by animators in current animation workflows. Therefore, in the following Chapter, we propose an alternative to keyframing for the case of motion cycles: we base our approach on performance animation, and our system keeps control on every frame of the animation to let the user finely edit the motions with a similar space-time curve representation.

Tangent-Space Optimization for Interactive Animation Control

CHAPTER

5

Authoring Motion Cycles



Figure 5.1: *Left: To specify a motion cycle, the user acts out several loops of the motion using a variety of capture devices. Middle: A looping motion cycle is automatically extracted from the noisy performance. Right: A custom motion representation tool, called MoCurves, allows controlling and coordinating spatial and temporal transformations from a single viewport.*

The previous Chapter examined in depth the keyframing process that is used to author general animations and proposed to enhance it with a space-time curve interface. For the creation of motion cycles, artists rely on the same animation tools and principles, which makes our proposed system applicable. However, making the motion loop perfectly from a set of separate keyframes remains a challenging task. In this Chapter, we make the core observation that the cyclic nature of motion cycles makes them especially

appropriate for performance animation, and we therefore propose an alternative to keyframing for authoring them.

Motion cycles play an important role in animation production and game development. In animated films and visual effects, artists use walk cycles and other looping animations during character development to explore a character's particular movement style. During production, these cycles aide the animation workflow by providing a starting point for walking, running, and other cyclic movements. In games, motion cycles play an even more prominent role. Motion cycles for locomotion allow characters to move arbitrarily under the player's direction. Punching, kicking, flipping, and countless other cycles are created to enable game combat. Even idle game characters are animated with motion cycles for breathing and other subtle movements that give them life. In fact, motion cycles play such an important role that game engines have specialized animation components designed to play back and seamlessly blend between different motion cycles.

Although specialized components for *using* motion cycles are commonplace, little work has explored the specific challenges of *authoring* them. Instead, the authoring process relies on general-purpose animation packages such as Autodesk Maya, 3ds Max, or Blender. These packages, by design, accommodate a broad spectrum of animation tasks with support for features and workflows used in animated films, visual effects, and games. This generality comes at a tremendous cost. The learning curve for any commercial animation package is steep, and the animation process relies on complex mechanisms that require expert knowledge to master. As a consequence, although a motion cycle may only be a few frames of repeated animation, creating those frames with general-purpose animation software is difficult, time consuming, and restricted to expert users.

This Chapter explores the challenge of motion cycle authoring and provides a system simple enough for novice animators while maintaining the flexibility of control demanded by experts. Our system allows the user to act several loops of motion using a variety of capture devices ranging from the computer mouse to a full body motion-capture suit. Since these acted cycles will inevitably contain imprecisions, we propose an optimization algorithm to analyze and automatically extract a looping cycle from this potentially noisy input. By supporting multidimensional input, cycles can accommodate an arbitrary number of animation variables. We then propose a space-time curve interface, similar to the one used in the previous Chapter, to edit the resulting motion cycle. These curves, that we call MoCurves, serve as a representation and manipulation of the movement and allow the user to control and coordinate spatial and temporal transformations from a single

viewport. They encompass translation, rotation, scale and time. Motion cycles for different character components can be authored independently in a layered fashion and synchronized in time using an optimization-based time-warping function built into the MoCurve interface. Finally, since contact with the ground or other surfaces is pervasive in motion cycles yet difficult to precisely control with a performance-based interface, we support a sketch based contact specification in which a single sketched contact line induces a spatio-temporal transformation that respects planar contacts without sliding.

Our work introduces an effective system tailored to motion cycles authoring. Our core technical contributions include a generic motion cycle extraction algorithm, the *MoCurve* representation with support for coordinated spatial and temporal editing, and a contact specification method that uses a single sketched line to establish non-slipping planar contacts. We implemented our approach as an Autodesk Maya plugin that permits motion cycle authoring independent of Maya's more complex animation features. We evaluated the effectiveness of our work through tests with both novice and expert users and showed motion cycles created with performance input from the mouse, Wacom Cintiq tablet, Leap Motion hand tracker, HTC Vive, as well as full-body motion capture. All these elements show that the creation process can be dramatically simplified using our software, allowing novice animators to author quality animations in minutes.

5.1 Background

One core contribution of our work is the automatic cyclification of an acted motion. While some existing research [Rose et al., 1996; Ahmed et al., 2003; Mukai, 2011] explores cyclification, these methods operate under the single-cycle assumption and cannot accommodate cyclification of repeated motions. They focus on matching boundaries of a single cycle and do not allow identifying a period in a longer sequence of an imprecisely repeated motion. In Rose et al. [1996], the user is even asked to manually specify the start and end points of a cycle. In our case, taking multiple loops as input is a technical challenge as it requires identifying a recurrent pattern over imprecise loops in space and time.

Some previous methods [Tsai et al., 1994; Silva et al., 1999] can take as input multiple loops, but can only process 1- or 2-dimensional curves. In contrast, we solve for N dimensions, which allows us to find the best period for the N -dimensional signal directly. Moreover, similar to their approaches, we experimented with frequency domain decomposition and found that for

performance animation the input is considerably noisy in the temporal dimension (i.e. all acted cycles do not have the same duration), which prevents us from using frequency analysis such as Fast Fourier Transform, and led to our feature-based solution.

5.2 Overview and Workflow

Our system is designed to create, represent and manipulate cyclic animations in a natural way. To create a motion cycle, the user performs the motion using any capture device, such as a mouse, Leap Motion, HTC Vive or full body motion capture suit (as shown in our Results Section 5.5). Depending on the device being used, the user may choose to perform the whole character motion or to animate parts of the character in a layered fashion. Also, since synchronizing several motions is a crucial element of animation, we play the animation of all previously created motions while the user performs for other items (or other transformations of the same items). For example, one would be able to act out the rotation of a foot while watching its displacement.

Because users like to author cycles by performing them several times — each time refining the motion — we begin by extracting a single cyclic curve from the multiple performed cycles. Our solution is formulated as an optimization problem which takes a repetitive and nearly cyclic motion as input, and outputs a single clean cyclic motion (Section 5.3). This optimization is performed in the N -dimensional space, where N is the number of dimensions being captured — as shown in our examples, N can vary from 1 or 2 (mouse input, Fig. 5.8) to a few thousands (physical simulation, Fig. 5.10). The cycle extraction may be applied to any degree of freedom parameterizing the character’s pose, such as rig controllers, skeletons, vertices, and any transformation such as translations, rotations, scales.

To allow the user to directly edit and refine the cyclic motion from a single viewport, we introduce in Section 5.4 *MoCurves*, a curve editor that combines translations, rotations, scalings and timing into a single unified geometric interface. We then describe in Section 5.4.3 an additional edit to our curves that allows specifying planar contacts and solving sliding effects with a single stroke.

5.3 Cycle Specification

In this Section we describe how we automatically extract a single closed cycle from multiple *nearly cyclic* repetitions. The performed trajectory is a discrete function p which associates times t_i to a vector of values $\theta_i = (\theta_i^1, \theta_i^2, \dots, \theta_i^N)$. These values can represent any type of transformation (positions, orientations, scales, a mix of them, etc.). The performed trajectory p is supposed to be a periodic movement, but is imperfect in both space — noisy θ_i — and time — noisy t_i . Hence our goal is to identify points p_i in the nearly cyclic motion that are geometrically similar across periods. We start by defining a curve descriptor that allows us to estimate the most probable value for the period T . We then extract a set of nearly cyclic curves that we average in order to obtain the shape of the final signal. We finally stitch the extremities together in order to obtain a perfectly looping cycle.

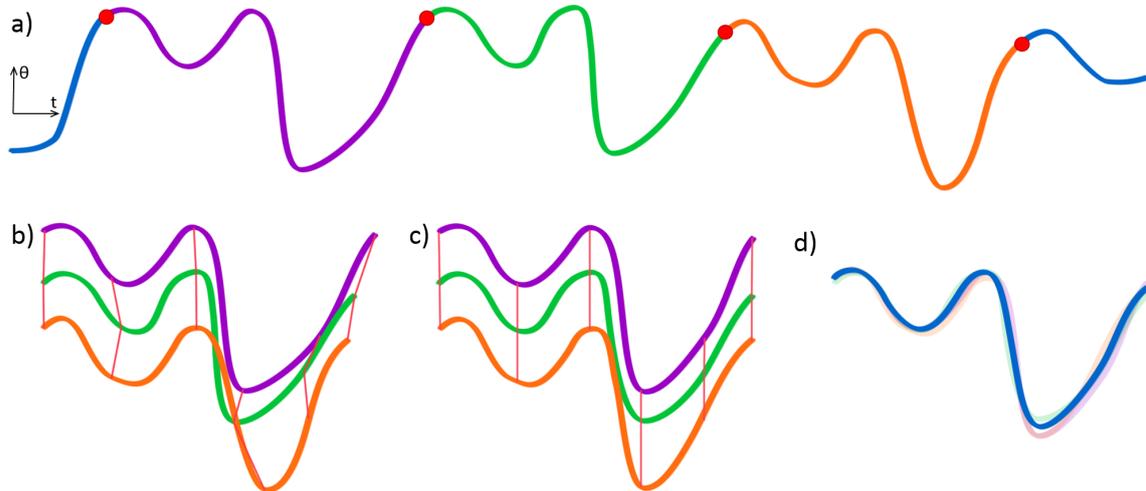


Figure 5.2: Steps of the cycle extraction algorithm illustrated on a simple example — only one attribute θ is animated, so p is of dimension 2. a) Points with a similar neighboring shape are identified (red circles) using a new curve descriptor. This allows to compute the average period of the motion, and to partition p into several cycles. b) Correspondences between cycles are computed. c) Each curve is non-uniformly scaled in order to align the corresponding points. d) These curves are averaged to form the final cycle.

Curve descriptor The function p represents an $N + 1$ -dimensional curve, in which we seek to identify repetitive patterns. To measure the similarity between points in a curve, we devised a descriptor that characterizes the neighboring shape of a point on a curve. Our descriptor is similar to the one used by Mori et al. [Mori et al., 2005], which we extend to the n -dimensional

case and make variant to rotations. Hence, we compute two descriptors \mathbf{h}_i and \mathbf{g}_i that respectively measure shape and velocity variations:

$$\mathbf{h}_i = \begin{bmatrix} \hat{\mathbf{h}}_i^0 \\ \dots \\ \hat{\mathbf{h}}_i^k \end{bmatrix} \quad \text{where } \hat{\mathbf{h}}_i^k = \sum_{|i-j| \leq k} \boldsymbol{\theta}'_j \quad \text{and } \boldsymbol{\theta}'_j = \frac{\boldsymbol{\theta}_{j+1} - \boldsymbol{\theta}_{j-1}}{t_{j+1} - t_{j-1}} \quad (5.1)$$

$$\mathbf{g}_i = \begin{bmatrix} \hat{\mathbf{g}}_i^0 \\ \dots \\ \hat{\mathbf{g}}_i^k \end{bmatrix} \quad \text{where } \hat{\mathbf{g}}_i^k = \sum_{|i-j| \leq k} \boldsymbol{\theta}''_j \quad \text{and } \boldsymbol{\theta}''_j = \frac{\boldsymbol{\theta}'_{j+1} - \boldsymbol{\theta}'_{j-1}}{t_{j+1} - t_{j-1}}$$

where we used $k = 5$, and we then define the similarity between two points as:

$$d_D(i, j) = \|\mathbf{h}_i - \mathbf{h}_j\|_2 \cdot \|\mathbf{g}_i - \mathbf{g}_j\|_2. \quad (5.2)$$

Period evaluation Considering the i -th point of the curve, p_i , we define J_i as the set of all indices j that are a local minima of $d_D(i, j)$, while remaining under a threshold d_D^m . By construction, J_i contains points that are similar in shape and speed to p_i . By conservatively choosing d_D^m high enough — thus favoring to select too many rather than too few points — we ensure that all the points corresponding to p_i in other cycles are present in the set J_i . Then, we eliminate the outliers in J_i and divide the period T by computing the minimal period that satisfies:

$$T = \min_{T \in \mathbb{N}} T \quad \text{s.t.} \quad \forall k \in \left\{ \left\lceil \frac{t_1 - t_i}{T} \right\rceil, \dots, \left\lfloor \frac{t_n - t_i}{T} \right\rfloor \right\}, \quad (5.3)$$

$$\exists j \in J_i \text{ s.t. } |t_j - (t_i + kT)| < m_T$$

Here, the threshold m_T depicts the variation in time of the cycles in p ; we used $m_T = \lceil T/8 \rceil$, which we found reasonable in practice for cyclic motions performed by humans. In some particular cases, often when two parts of the motion are very similar in shape and velocity, the point i can be badly chosen resulting in an incorrect period. To eliminate this undesirable case, we perform this computation for ten random points and select the median period.

Average cycle Given the set of indices J_i cleaned from outliers (i.e. J_i only contains points corresponding to p_i in other cycles), we cut the curve p at the corresponding points and extract a set of cycles c_1, c_2, \dots, c_p (Fig. 5.2a). Similarly to the construction of J_i , we measure the similarity d_D over

evenly spaced points to find correspondences between the cycles (Fig. 5.2b): $\mathbf{c}_1(t_1^k) \leftrightarrow \mathbf{c}_2(t_2^k) \leftrightarrow \dots \leftrightarrow \mathbf{c}_p(t_p^k)$. We then non-uniformly scale the cycles such that $t_i^k = t_j^k \forall i, j, k$ and the temporal length of each cycle is T (Fig. 5.2c). We finally compute the average cycle \mathbf{c} (Fig. 5.2d) yielding:

$$\mathbf{c}(t) = \frac{1}{p} \sum_{i=1}^p \mathbf{c}_i(t), \forall t \in [0...T] \quad (5.4)$$

The curve c we averaged may contain a discontinuity at its extremities, i.e. $\mathbf{d}_{ex} = \mathbf{c}(0) - \mathbf{c}(T)$ may not be $\mathbf{0}$. In order to make it perfectly cyclic, we stitch the curve c as follows:

$$\mathbf{c}(t) = \mathbf{c}(t) + \left(\frac{t}{T} - \frac{1}{2} \right) \mathbf{d}_{ex}, \forall t \in [0...T] \quad (5.5)$$

Spline fitting. In order to have a smoother representation of the motion, as well as a simpler editing, we fit a cubic Bezier curve to each component of the cycle c — i.e. one for the translations (γ_P), one for the rotations (γ_R) and one for the scales (γ_S) of each moving item. Many B-Spline fitting methods already exist; we chose to use the one implemented in the Autodesk Maya API.

5.4 MoCurves

To edit motion cycles in an intuitive manner, we combine spatial and temporal controls into a single geometric representation that allows the user to edit both aspects in a single viewport. This is particularly challenging for time as the user needs to precisely and intuitively edit temporal constraints using 3D spatial manipulations. For this purpose, we introduce MoCurves. Their representation is similar to the Motion Curves introduced in Chapter 4, but their functioning is different since here we do not solve for the tangents of interpolation but for every frame of the animation.

For each animated *item* (i.e. object, rig controller, bone, etc.), we define one MoCurve that represents its spatial (i.e. translation, rotation and scale) and temporal transformations. The following subsections describe how MoCurves allow the visualization and manipulation of diverse aspects of the motion, as well as permit the simple editing of planar contacts and automatic solving of sliding effects. Note that to amplify the intuition of movement and timing, we provide the ability to make all manipulations in real-time, while the animation is played; a benefit of working with motion cycles is that they seamlessly loop, so this does not create a visual discomfort.

5.4.1 Spatial Manipulations

As described in Section 5.3, an item's position, rotation and scale over time are described by cubic Bezier splines γ_P , γ_R and γ_S . There is a direct correspondence between the parametrization of these curves and the time of the animation (e.g. at time t^* , the item is at position $\gamma_P(s_P^*)$). We note φ_P , φ_R and φ_S the bijective functions giving t from the parametrization of each curve — i.e. $t^* = \varphi_P(s_P^*) = \varphi_R(s_R^*) = \varphi_S(s_S^*)$. This provides a unified correspondence between all aspects of the motion.

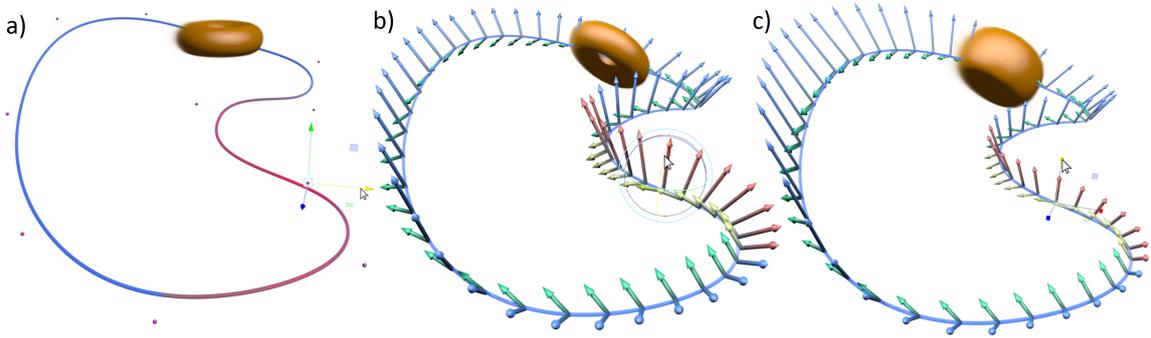


Figure 5.3: *MoCurves allow three types of spatial edition. A 3D spline with editable control points represents the trajectory over time (a). Arrows at every time frame represent both the orientation (b) and scale (c) over time; they are directly manipulable in the viewport. In red are the regions affected by the manipulations.*

The curve γ_P is displayed in the viewport in order to represent the displacement (i.e. translations) over time. The user can directly edit the trajectory by manipulating control points of γ_P (Fig. 5.3a). We represent the orientation (i.e. rotations) at each time frame by two orthogonal arrows centered at the corresponding position (Fig. 5.3b), and the scaling by the geometry of these same arrows (Fig. 5.3c). Thus, by looking at a MoCurve, a user has a clear overview of the item's movement.

The orientation and scale of arrows can directly be manipulated in order to modify the rotations and scales of the curve over time. The modification of one frame also modifies neighboring ones in order to smooth the motion, just like the editing of a curve's control point affects a certain fraction of the curve. The range τ of affected frames (red arrows in Fig. 5.3) is set by the user. When an attribute of arrow i is deformed by $d\theta_i$, we propagate the deformation on arrows j using a Gaussian radial basis function:

$$d\theta_j = d\theta_i \cdot e^{-\sigma(i-j)^2}, \quad \forall j \in \{i - \tau, i + \tau\} \quad (5.6)$$

5.4.2 Temporal Manipulations

Animators often draw time bars at a constant temporal interval to convey the timing of a hand-drawn motion. Inspired by this representation, we render on top of the Mocurve γ_P keypoints at each time frame of the animation. A keypoint can either be represented by a point or by arrows that also give information about orientation and scale (see Section 5.4.1). Hence, if keypoints are close to each other it means that the motion is slow, while if they are very distant it means that the motion is very fast.

Once again, the visual representation also serves as a manipulation tool: the user can directly edit the position of keypoints in order to edit the timing of the cycle, as shown in Fig. 5.4. We formulate the deformation as shape preserving deformation, similarly to [Kim et al., 2009], but here applied to the case of periodic curves.

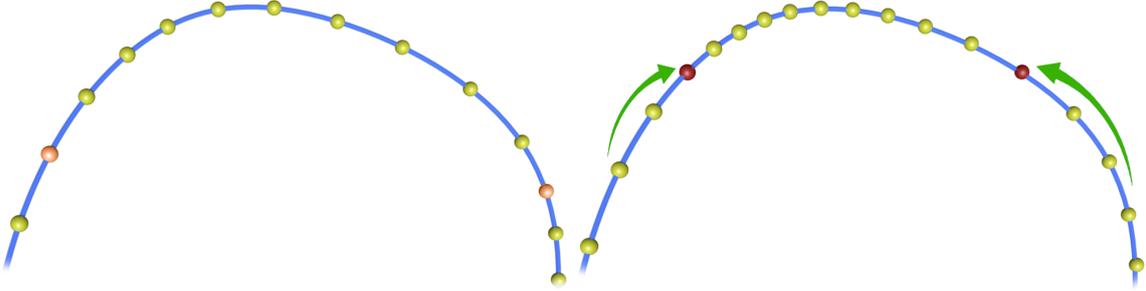


Figure 5.4: Two keypoints are moved upwards on the curve, setting new timing constraints (red points). A smooth time warping is applied, resulting in a motion that is slower at the top. Note that the motion is consequently faster on the rest of the curve in order to conserve the cycle period.

By moving keypoints, the user defines a set of spatial constraints: $\gamma_P(s_i^*) = \gamma_P(s_i^c), \forall i \in \mathcal{C}$. We thus seek a new distribution of keypoints along the curve $s^* = (s_0^*, \dots, s_m^*)$ satisfying $s_i^* = s_i^c, \forall i \in \mathcal{C}$. This is equivalent to computing a new temporal distribution $t^* = (t_0^*, \dots, t_m^*)$ satisfying $t_i^* = \varphi_P(s_i^c), \forall i \in \mathcal{C}$. To do so, we solve a quadratic optimization problem containing four energy terms.

Constraints $E_C(t^*)$ We penalize the distance between the constrained points and their desired position:

$$E_C(t^*) = \sum_{i \in \mathcal{C}} (t_i^* - \varphi_P(s_i^c))^2 \quad (5.7)$$

Period $E_T(t^*)$ A crucial constraint is that cycles must conserve the period:

$$E_T(t^*) = (t_m^* - t_0^* - T)^2 \quad (5.8)$$

Velocity $E_V(t^*)$ We regularize velocities by penalizing deviations from the ones on the original curve. In other words, the temporal spacing between consecutive t_i^* should be stable:

$$E_V(t^*) = \sum_{i=0}^{m-1} ((t_{i+1}^* - t_i^*) - (t_{i+1} - t_i))^2 \quad (5.9)$$

Speed variation $E_S(t^*)$ The time warping must not introduce points where the motion is accelerated or decelerated abruptly. That means that two consecutive segments have to stay close in size:

$$E_S(t^*) = \sum_{i=0}^{m-1} ((t_{i+1}^* - t_i^*) - (t_i^* - t_{i-1}^*))^2 \quad (5.10)$$

The new time distribution is thus obtained by minimizing the total energy, while ensuring that t^* increases:

$$\begin{aligned} \min \quad & w_1 E_C(t^*) + w_2 E_T(t^*) + w_3 E_V(t^*) + w_4 E_S(t^*) \\ \text{subject to} \quad & (t_{i+1}^* - t_i^*) \geq 0, \forall i \in \{0, \dots, m-1\} \end{aligned} \quad (5.11)$$

We solve this constrained quadratic programming problem using a qp solver, where we choose the weights w_1 and w_2 to be 10^4 times bigger than w_3 and w_4 because they act as strong constraints. Finally, we recover the positions, orientations and scales at each time step using $\gamma_P(\varphi_P^{-1}(t_i^*))$, $\gamma_R(\varphi_R^{-1}(t_i^*))$ and $\gamma_S(\varphi_S^{-1}(t_i^*))$.

5.4.3 Contacts

While contacts are present in most cyclic character motions such as locomotion, they can hardly be acted out accurately as they involve sharp corners in the trajectory. To edit a MoCurve as to exhibit sharp corners and straight lines, we introduce a stroke-based editor that cuts the curve with a straight line. This line, extruded along the viewing direction, defines a plane on which all points from a section of the curve are projected (Fig. 5.5).

This solves contacts in terms of space, but not in terms of time. Indeed, two points simultaneously touching the ground can introduce a sliding effect if

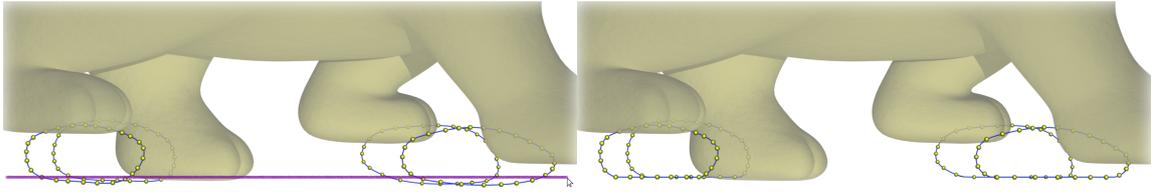


Figure 5.5: *The user can draw a line (here in purple) to specify ground contacts. Portions of the trajectories are then projected on the contact plane, as shown on the right.*

they are not coordinated. In other words, we need to ensure that at every frame of the animation, two items being in contact with the ground have the same velocity — note that velocities are expressed with respect to the character’s root. The earlier projection step gives us information about which items are in contact and at which time. This allows us to construct a graph of the velocities of items in contact over time, as shown in Fig. 5.6. We fit a spline λ minimizing the distance to these curves, using a method similar to Section 5.3. λ defines the desired velocity of all items in contact with the ground over time.

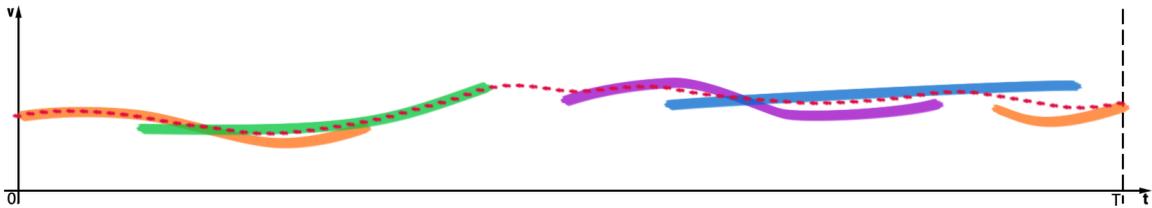


Figure 5.6: *This graph represents the velocity over time of four items touching the ground (one per color). To unify them, we fit a spline λ (red dashed curve) that averages the contacts velocity. Note that here we illustrate for only one dimension, but the velocity is usually 3D.*

We now spatially modify the curve γ_P of each colliding item in order to satisfy the contact velocity λ . First, we transform the part of γ_P being in contact, which we define as $\gamma_P^C \subset \gamma_P$, such that:

$$\gamma_P^C(\varphi^{-1}(t + dt)) = \gamma_P^C(\varphi^{-1}(t)) + \lambda(t) \cdot dt, \quad \forall t \quad (5.12)$$

This may change the length of γ_P^C by a scale factor s_l . In order to keep a smooth connection between γ_P^C and the rest of the curve, we also scale $\gamma_P \setminus \gamma_P^C$ by s_l in the direction of the contact. Finally, in order to ensure consistent contacts — i.e. contact points are fixed in world space — we move the root of the character with the velocity $-\lambda(t)$ in world space coordinates. This way, by simply specifying a contact surface, the user is able to make a character move inside the environment in a consistent way, without sliding effect.

5.5 Results

We apply our system to four different characters with diverse shapes and rig complexities. In this Section, we present a number of compelling motion cycles that both novice and expert users were able to author using a variety of input devices. Table 5.1 gives statistics about the number of elements and transformations that were animated in each case — note that diverse full-body motion capture suits were used for the Mocap motions of Fig. 5.9, which is why the numbers vary at the bottom of the table.

Motion cycle	Figure	Anim. elem.	Anim. transf.
Dinosaur walk	5.7 (a)	13	39
Human punch	5.7 (b)	5	8
Robot swim	5.7 (c)	8	23
Human dance	5.7 (d)	12	33
Dinosaur dance	5.7 (e)	18	49
Dragon eat	5.8 (left)	10	29
Dragon fly	5.8 (middle)	35	51
Human juggle	5.8 (right)	16	52
Dinosaur leap	5.9 (left)	11	21
Human kick	5.9 (middle)	11	30
Mocap punch	5.9 (right)	21	126
Mocap walk	5.9 (right)	60	253
Mocap samba	5.9 (right)	52	159

Table 5.1: For each motion cycle presented in the Results section, this table gives the number of elements (i.e. rig controllers or skeleton joints) and the number of transformations (i.e. translations, rotations, scales) that were animated.

To evaluate the accessibility of our system, we invited five novice users, who never animated any character before, and gave them a limit of one hour to author a motion cycle using our tool. The resulting animations, presented in Fig. 5.7, are particularly convincing considering the inexperience of the creators. This study was also a social success as users were enchanted to be able to animate a character, several of them concluding: “this was the most enjoyable user study of my life”. For comparison, we also asked two of these novice users to create a similar motion cycle using Autodesk Maya, without our plugin. After one hour of struggle (which was their time limit), no exploitable content was created. This result confirms our belief that general purpose animation packages such as Maya require significant training before even simple animations can be created.

Additionally, our tool was used by a professional artist in order to evaluate

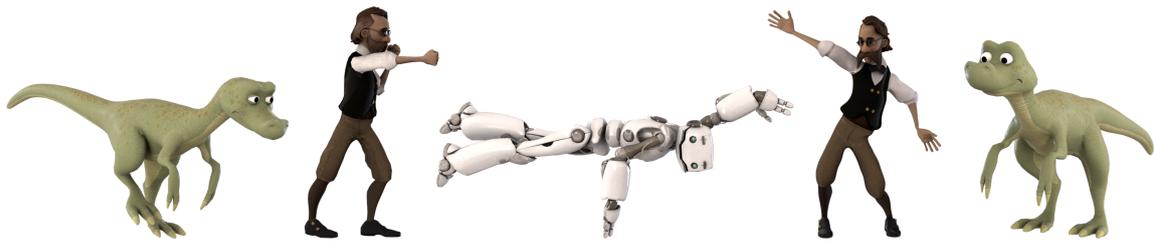


Figure 5.7: *Five novice users, who never created a character animation before, used our system. In less than one hour, they were respectively able to create (from left to right): a walk, a punch, a swim and two dance cycles.*

how well our system is integrable into an expert’s pipeline. The artist used a Wacom pen and tablet and authored the three motion cycles presented in Fig. 5.8: the eating dragon was animated in approximately 20 minutes, the flying dragon in approximately 15 minutes and the juggling human in approximately 35 minutes. The last example exhibits how much our system enables a fine control over the spatial and temporal aspects of the motion, allowing the composition of complex synchronizations. As a feedback, the artist shared how delighted he was to be able to directly perform the motion he had in mind without having to be super precise, and how important the MoCurves were in order to maintain a full control over the final result. He claimed that this is a tool he would like to use regularly for the motion cycles he needs to create.



Figure 5.8: *A professional artist used our system to author three motion cycles. From left to right: a dragon eating, a dragon flying and a human juggling. They were respectively created in about 20 minutes, 15 minutes and 35 minutes.*

A large variety of devices can be used to practice performance animation: these range from a computer mouse to full body motion-capture suits and include Leap Motion, Kinect [Wang et al., 2012], tactile surfaces [Lockwood and Singh, 2012; Chung et al., 2015] and other dedicated devices [Oore et al., 2002; Slyper and Hodgins, 2008; Shiratori et al., 2013; Glauser et al., 2016]. Our method is generic enough to work with any type and dimensionality of data as input. Most of our results were generated

Authoring Motion Cycles

using the most familiar devices — a mouse or a digital pen — but we also demonstrate in Fig. 5.9 proper functioning with Leap Motion, HTC Vive and full body motion-capture suits.

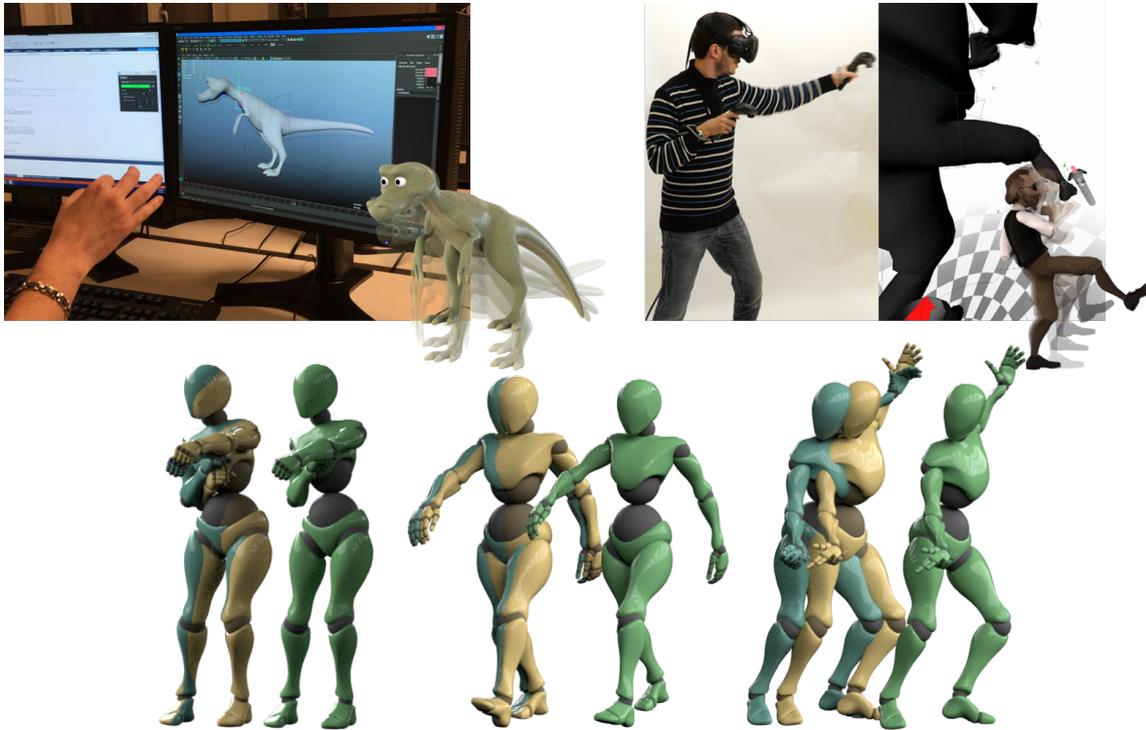


Figure 5.9: *Our system supports a variety of performance capture devices. Here we show a jump cycle created using the Leap Motion hand tracker (top left), a kick cycle created using the HTC Vive (top right), and a punch, a walk and a samba cycles created using a full body motion-capture suit (bottom). In these last examples, the overlapped blue and yellow mannequins show the spatial difference of cycles in the imperfect performed motion, while the green mannequin shows the looping cycle extracted by our algorithm. Note that in each case, we solve for a single multidimensional curve representing the whole motion (dimension 127 for the punch cycle, 254 for the walk cycle and 160 for the samba cycle).*

In terms of performances, the system is responsive enough to permit an interactive utilization. The cycle extraction algorithm takes less than one second to be executed, even in high-dimensional cases (such as motion-capture, Fig. 5.9) where the curve being analyzed can have several hundreds of dimensions. As for the MoCurves manipulations, as specified in Section 5.4, they are executed in real-time, while the animation is being played.

5.6 Conclusion

In this Chapter, we introduced an authoring tool tailored to cyclic motion design. Our tool enables a user to repeatedly act out a periodic movement and automatically extract a single closed cyclic motion. In order to conserve a fine level of control required by artists, we then introduced MoCurves, a curve editor that combines both space and time into a single geometric entity that allows coordinated editing in a single viewport. By removing a thick layer of expert knowledge required by general purpose animation tools, we allowed both professional artists and novice users to create compelling animations.

When extracting a cycle from a performed motion, our algorithm requires rough consistency in the input cycles, both in shape (same overall displacement) and timing (cycles of similar duration). If the recorded performance cycles vary widely, the algorithm will fail to find a period or the extracted loop will be of poor quality. However, in our experiments, even inexperienced users were able to perform loops consistent enough to deliver quality results.

MoCurves allow the visualization and editing of the most widely used attributes in animation. However, artists sometimes customize their rigs with additional attributes, such as roll and lean for a foot. In our results, we supplanted them with the translation or rotation of additional rig elements, but it would be interesting to explore mechanisms that would enhance MoCurves for the representation and manipulation of supplementary attributes. As well, we provide a way to easily edit planar contacts, but the intuitive authoring of more complex interactions, such as non-planar or dynamic contacts, remains an open and challenging problem.

Our algorithm can find cycles even in structured physical simulations such as cloth, as shown in Fig. 5.10. However, it does not conserve the physical correctness of the original simulation. Integrating mechanical constraints into our algorithm in order to create physically-accurate cyclic simulations would be an interesting direction for future work.

We showed that the animation tools presented in this Chapter, as well as in the previous one, permit to animate characters in a faster and more intuitive way than traditional systems. However, they focus on providing a fine motion control and do not permit an interactive animation of the entire character. In the next Chapter, we introduce a new interaction mechanism where a user can steer a virtual character with simple and natural gestures. Predefined motion cycles, potentially created using this Chapter's system,

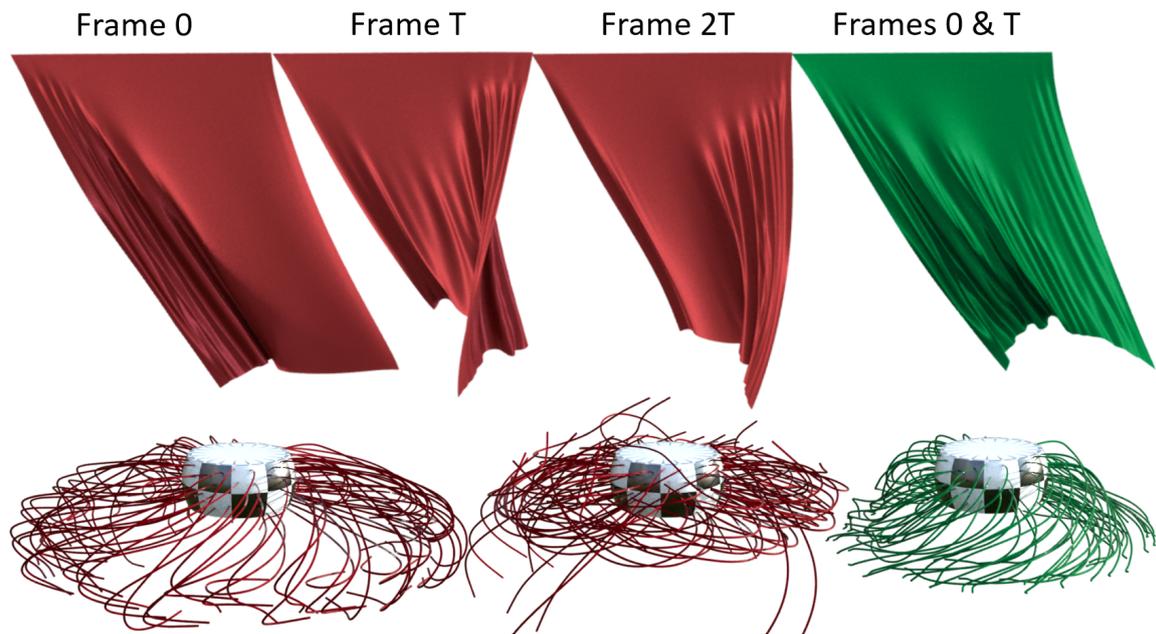


Figure 5.10: *Handles of these two physics simulations (i.e. translating top of cloth and rotating basis of hair) move periodically. Though, the simulated motions are not periodic, as shown in red with frames spaced by the period T . Our algorithm extracts looping motions, in green, from these simulations. The dimensionality of the cyclified curves is respectively 7804 and 2611.*

are used in order to augment the character's displacements with compelling animations.

C H A P T E R

6

PuppetPhone: Puppeteering Virtual Characters Using a Smartphone

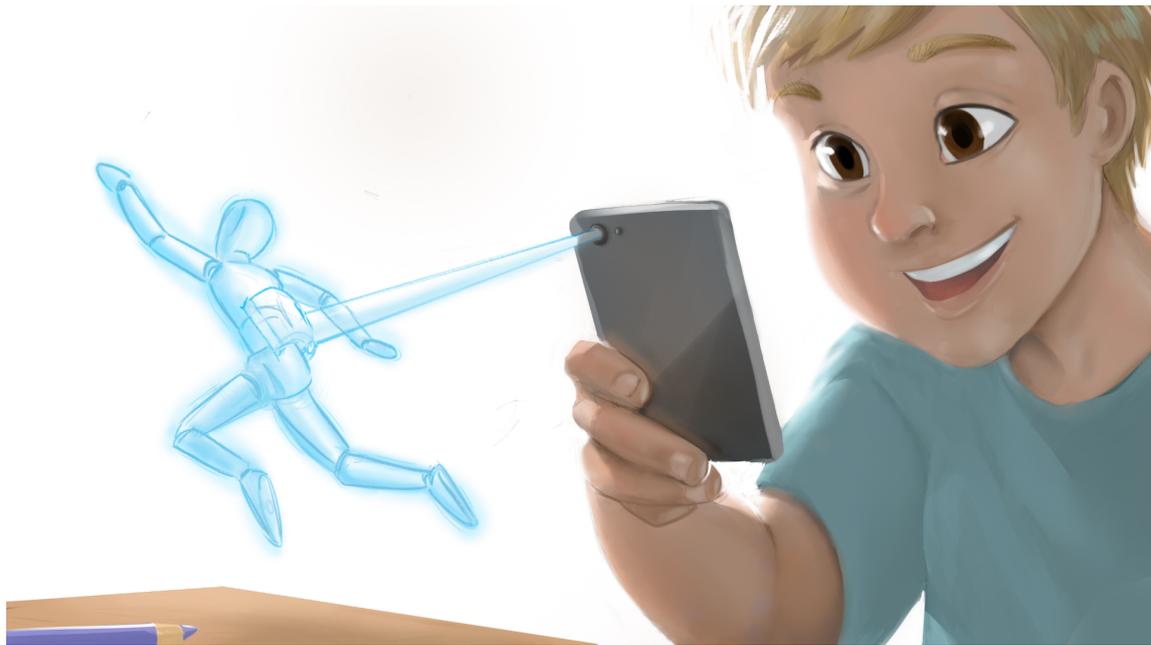


Figure 6.1: *Using our new system, a player is able to manipulate a virtual puppet using a smartphone. The character responds compellingly and in real-time to the user motions, so that it stays at all times at a fixed distance from the phone.*

When playing with toys, people cherish grasping them and manipulating them freely. They imagine characters, create stories and solve quests by moving the toys around and putting them in diverse situations. The major frustrating aspect is that the handled physical puppets are inanimate;

they follow the player's gestures like a lifeless and unconscious ragdoll. The player's imagination then has to fill the secondary motions with compelling animations like walk cycles, jumps and kicks.

Playing in virtual worlds tackles this shortcoming because full character animations can be achieved from little inputs. Usually in video games, the player simply presses a button to trigger a predefined action. However, this makes the interaction more abstract and it lacks the satisfactory feeling of grasping the character and physically moving it.

Recent years have seen the appearance of diverse control devices such as Kinect, Wii controllers, Vive controllers, Leap Motion and powerful smartphones. Each of them is able to track 3D movements to a certain extent, which opens the door to new interaction metaphors. Even if many applications have then been introduced, like mimicking movements and grabbing virtual objects, little work has been done to retrieve or enhance the particular feeling of manipulating a puppet.

In this Chapter, we introduce an enhanced puppet interaction system, where a virtual character accompanies the user's gestures in a compelling manner. The player uses a smartphone to observe, grasp and move the virtual puppet, whose movements are beautified with detailed animations. Using a phone, our method ties together the control of the character and camera into a single interaction mechanism. The virtual character moves in order to stay visible to the user, as if it was attached to the phone via a rigid stick. It thus reacts in real time to the user's motions, similar to a physical puppet but with the difference that it now looks alive. We achieve this by interpreting the user's manipulations, with respect to the current character's state and the neighboring environment, into a weighted combination of predefined animations (see Section 6.2). Our system requires a minimal amount of provided animations, that we adapt to different environments and character dimensions.

We illustrate our system in an Augmented Reality application in Section 6.3, where the player can grasp and move a character to make it, among other actions, walk, jump, pick up objects and even create controllable snowmen of any dimensions. We also show a second application as a *Multi-Reality* game in Section 6.4, where the user is progressively immersed into the virtual world.

6.1 Background

Smartphones are very powerful computing devices that comprise a large variety of sensors — multi-touch screen, cameras, accelerometer, gyroscope, etc. — which provide a lot of information about their manipulation, and in particular their displacement. The gesturing of smartphones has been explored in several domains of computer graphics, for example to model simple 3D shapes [Vinayak et al., 2016] and edit animations [Lockwood and Singh, 2016]. Despite that, a large majority of mobile applications only takes benefit of the tactile screen to drive a virtual character, using a push-button approach as described earlier. A few works have taken advantage of the displacement information to reconstruct a motion, using a single [Haegwang et al., 2014] or several [Pascu et al., 2013] smartphones, but none of them allows to move the character in a puppeteering manner as we do.

6.2 Approach

6.2.1 MotionStick

We propose a new interaction principle, the *MotionStick*, that works as an extension of the *MotionBeam* introduced by Willis et al. [2011]. A user manipulates a smartphone that has information about its orientation, position and movement in space. By looking at the virtual environment through the screen, they can point the phone towards an object and grab it by holding down the touchscreen. The object is then fixed to the end of an invisible *MotionStick*, as represented in Fig. 6.2, and will react appropriately to any movement of the smartphone caused by the user. While being held this way, the distance and relative rotation to the phone is maintained, giving this control scheme a very responsive and direct feel. In some cases these constraints can be relaxed, especially to handle collisions. For example, if the grabbed object is pushed into the floor, the length of the *MotionStick* is shortened appropriately in order to prevent it from phasing through the floor.

This interaction metaphor, similar to a physical reach extender, yields a natural interaction with virtual objects. The very light user interface, simply consisting of pointing with a smartphone, makes it very easy to learn and use. Moreover, regardless of this simplicity, it provides the user with a fine and expressive control on the virtual space. This will be showcased in Section 6.3 with our implementation prototype, where our application does not require any additional user input interface.

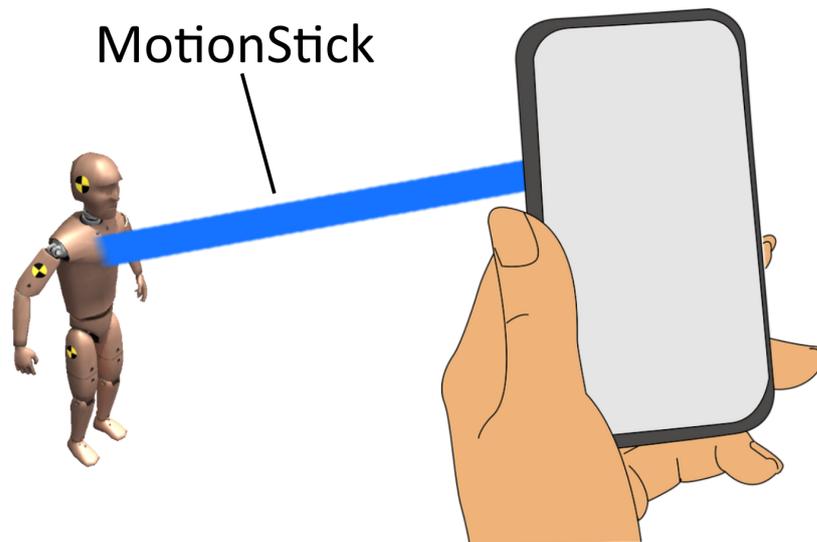


Figure 6.2: *Using the MotionStick metaphor, a virtual character is manipulated as if it was attached to the end of an imaginary stick fixed to the smartphone.*

When moving a virtual object around, it has the ability to react in more ways than just updating its position and rotation according to the state of the MotionStick. In the following subsections, we will describe a method that enables an object to come to life by animating it in accordance to the way it is moved. From here on out, we will use a terminology corresponding to a humanoid (e.g. walking, crouching); however, note that our system is adaptable to any type of animated object, such as quadrupeds and cars.

6.2.2 State Machine

We use a state machine to determine how the virtual character is animated to react in real time to the user movements. For example, in one state the character stands on the ground but as soon as the user flicks the character up it switches to the jump state. Each state is defined by its internal logic, root position, configuration of the character animation and the IK state. The context of the state machine consists of the environment – i.e. the ground and other objects surrounding the character – and the user inputs – i.e. movements of the MotionStick. One state is the *active state* and at every time-step its logic updates the state of the character animation based on the context. Events can be defined that trigger a state change and thus another state becomes the active one, changing the behaviour of the animated character.

The MotionStick metaphor does not restrict the manipulation of the object by the user. That is why the system controlling the character has to be pre-

pared for any input, even when no predefined animation is appropriate. Our system handles such unexpected inputs by having a default state that is activated when such a situation arises. A fitting default state would be to turn the character into a ragdoll. It is for example necessary when a character that has no jump momentum is held in the air; instead of having a character with an inappropriate jumping animation in the air, the user would instead be holding a physically simulated ragdoll.

Unpredictable user inputs also mean that animations with a high degree of interaction with the world have to be adapted (e.g. the character picks up an object from different directions and poses). In these cases, we use inverse kinematics to adapt animations to the given situation. For example, when the character picks up an object, its hands are moved close to the object independent of the underlying animation.

6.2.3 Animation Blending

One challenge when animating a character with MotionStick is that the input movement is very continuous. Therefore, contrary to interfaces with a push-button metaphor, we cannot simply play a limited set of preexisting animation clips. For example, if animations are defined for *Walking* and *Running* states, it is unclear which one to choose when the manipulation speed is just between the two. The solution we choose is to use animation blending. There exists different algorithms for blending animation clips – e.g. linear, cubic, etc. – and our method can be used with any of them; we will thus treat the blending algorithm as a black box.

Our system requires a small database of animations, that can be used to blend together new ones. Each provided animation i has N blend parameters $p_{i,j} \in [0, 1]$, that are used to place it in the N -dimensional blend space – we call that point $p_i = (p_{i,1}, \dots, p_{i,N})$. Given a new point p in that space, the blending algorithm returns a new animation that is a combination of the neighbouring ones. Fig. 6.3 gives a blend space example where five animation clips are predefined – idle, walking, running, crouching idle and crouching walking – and each of them has two blend parameters – corresponding to the root velocity and the crouching height. The challenge is then to map the values from the user input u_i (i.e. the smartphone’s position, orientation, speed, etc.) to the blending parameters $p(u_i)$.

Most mappings are linear, which makes the computation of $p(u_i)$ straightforward. For example, the height of the smartphone u_{height} is mapped linearly to the crouching height: $p(u_{height}) = (u_{height} - a) * 1 / (b - a)$ (clamped

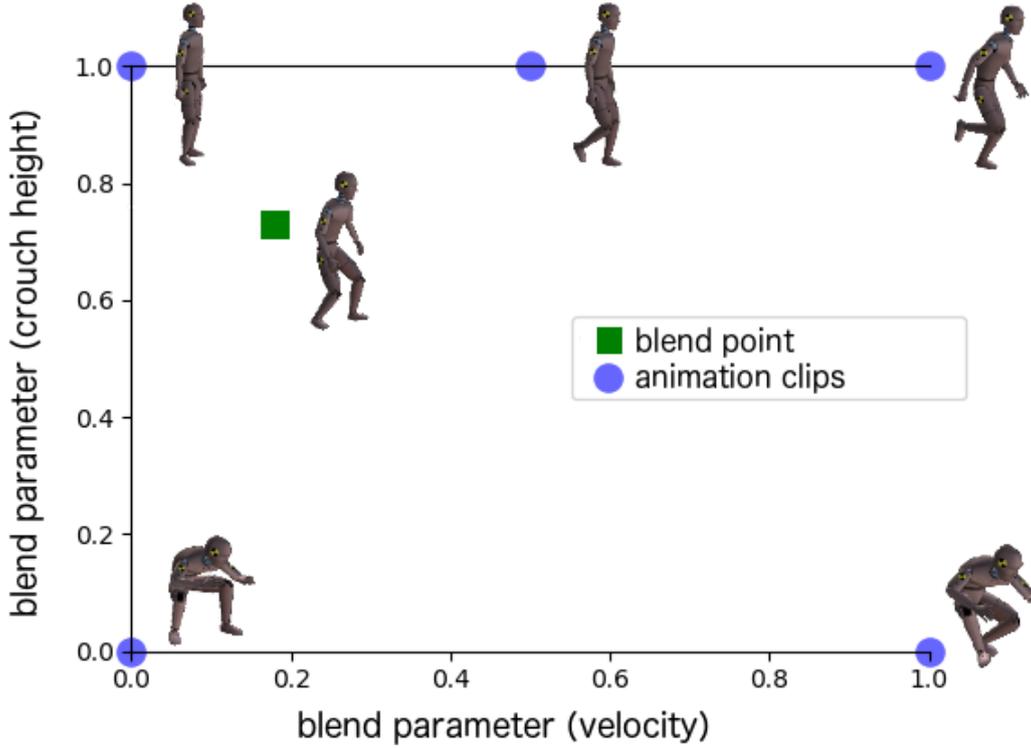


Figure 6.3: An example of a blend-space graph with the predefined animation clips *idle(0,1)*, *walking(0.5,1)*, *running(1,1)*, *idle crouching(0,0)* and *walking crouching(1,0)*. In green is the desired blended animation with parameters $p_{vel} = 0.2$ and $p_{height} = 0.73$.

to $[0, 1]$). However, other mappings are non-linear, in particular the smartphone velocity u_{vel} . This is especially important because an inexact mapping would produce the wrong animation and result in foot sliding artifacts. We solve this by creating a lookup table and make it continuous using inverse distance weighting.

We fill our lookup table by choosing a set of probe points p_k^* in the blend space with a high enough density (e.g in a grid). The user properties u_k^* of these probe points can automatically be measured by blending the corresponding animation and then measuring its properties, for example the movement velocity. With that, when a user provides new input values u , the corresponding blend parameters are computed using inverse distance weighting:

$$p(u) = \begin{cases} \frac{\sum_k w_k(u) p_k^*}{\sum_k w_k(u)}, & \text{if } d(u, u_k^*) \neq 0 \text{ for all } i \\ p_k^*, & \text{if } d(u, u_k^*) = 0 \text{ for some } i \end{cases} \quad (6.1)$$

with

$$w_k(u) = \frac{1}{d(u, u_k^*)^q} \quad (6.2)$$

where d is the euclidean distance between two points and $q \in R_+$ is the power parameter – in our implementation we used $q = 7$. A high q leads to a "sharper" resolution because only the points very close to u , but also needs a higher density of probe points to prevent jerky transitions. In Fig. 6.4, we show the lookup table obtained from the example in Fig. 6.3. Notice that the border, defined by all animation clips with $p_{vel} = 1$, is not a straight line. This demonstrates that the mapping of the velocity parameter is non-linear.

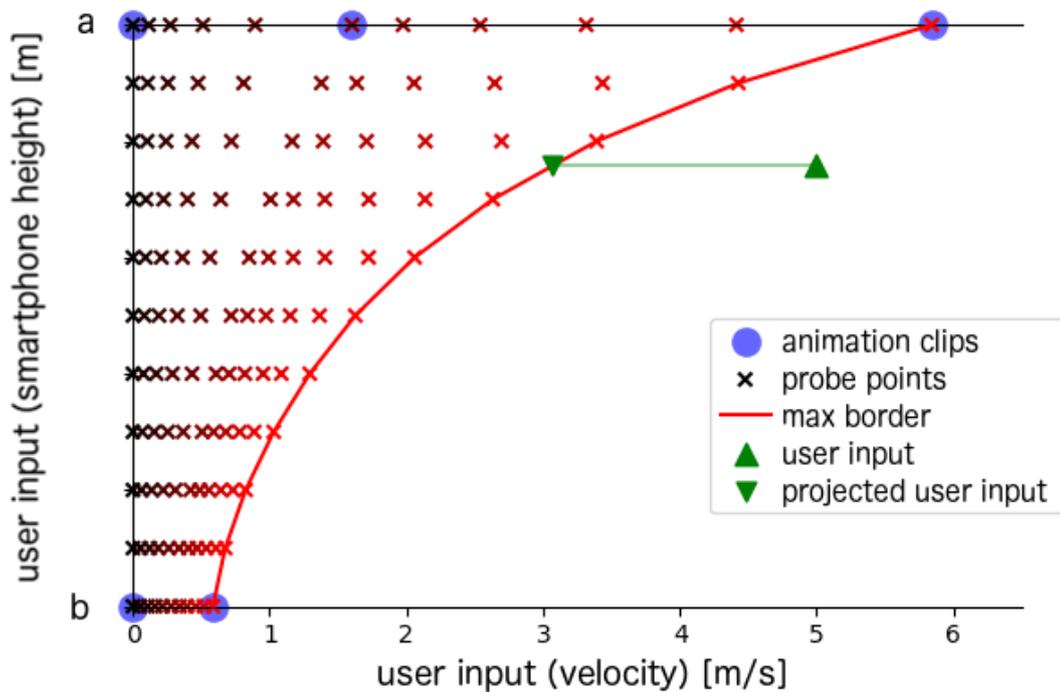


Figure 6.4: The generated lookup graph used in our implementation, with a user input that has to be projected to be inside the area of well defined blend parameters. The blend parameter for velocity p_{vel} is encoded in the red color channel of the probe points, ranging from 0 to 1.

There is the corner case where the user input u is outside of the defined area of possible animations – e.g. the user moves the character so fast that no animation can be blended to match that velocity. To tackle that, we first project u onto the border of the set of feasible configurations, obtaining u' . In higher dimensions, the border would be a multi-dimensional mesh. In the case of movement velocity, we then speed up the animation by a factor $\frac{u_{vel}}{u'_{vel}}$ in order to achieve a motion with the desired velocity.

6.3 Application: Build a Snowman

We demonstrate our system by developing an Augmented Reality application running on iPhone X. It was implemented using Unity with the libraries Vuforia (for the AR), FinalIK (for inverse kinematics) and PuppetMaster (for interpolating between ragdoll simulation and static animation). The ARKit framework allows Vuforia to use the computational power of the iPhone X to enable robust markerless AR tracking. For blending between animations, we used the native animation framework of Unity. We hereafter describe the characteristics of our prototype and how they relate to the challenges described in Section 6.2 (Fig. 6.5).

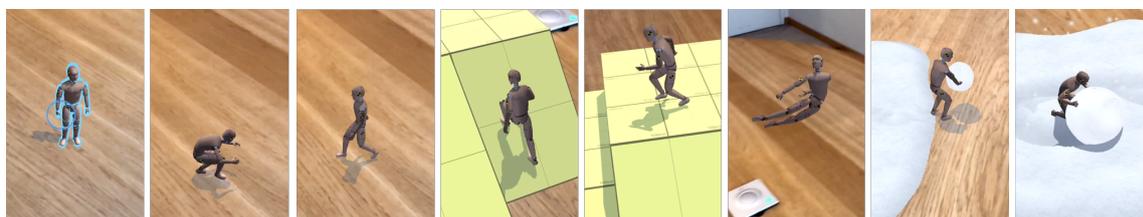


Figure 6.5: *An outline is shown when an object is in focus to be grabbed (left). When pressing down on the touchscreen the character is controlled with the MotionStick metaphor using smartphone movements. Animations are generated to fit to any given situation and user input.*

Our implementation uses only 7 animation clips: idle, walking, running, idle crouched, walking crouched, get-up and mid-jump. All other animations are a combination of animation blending, ragdoll simulation and IK. For example, rolling a snowball is made possible by taking the crouched animation and then placing the hands on the surface of the snowball. The user inputs include the position, velocity and orientation of the smartphone in space. Additionally, we use the touchscreen as a single button to grab the puppet with the MotionStick metaphor. No other buttons or inputs are used which makes the interface extremely easy to learn. The environment of our state machine contains the distance of the puppet to the ground and the set of manipulable objects nearby.

Please refer to Fig. 6.6 for our state machine. The ragdoll state is used in any situation where the user would force an undesired situation. E.g. when the jumping arc would be too unrealistic. The walking and crouched animations are generated with our animation blending method discussed in Section 6.2.3. This lets the character react to any user movement with a suitable blended animation. More details about the jump state are given in the following paragraph.

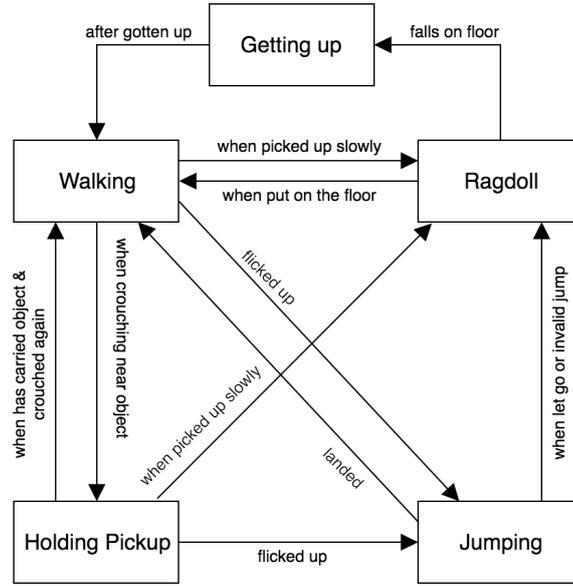


Figure 6.6: The state machine used in our implementation. Arrows indicate our events that switch to a new active state.

While the user input for making the character jump is simply to lift up the phone with a high enough velocity, a compelling jump animation requires to squat before taking off (i.e. *Anticipation* principle of animation). Therefore, we force a short delay between the user movement and the jump in order to build that anticipation. After that, the character smoothly returns back to the position given by the MotionStick, which has meanwhile moved along the jump path. We can estimate the jump path with a 2-dimensional parabola when looking from the side view. At each frame the parabola is calculated given the starting position of the jump (x_0, y_0) , the current position (x_t, y_t) and the current slope of the jump y'_t :

$$y(x) = ax^2 + bx + c$$

$$\text{where } a = \frac{y'_t(x_t - x_0) - y_t + y_0}{(x_t - x_0)^2}$$

$$b = \frac{y'_t(x_0^2 - x_t^2) - 2x_t(y_t - y_0)}{(x_t - x_0)^2} \quad (6.3)$$

$$c = \frac{x_t^2(y'_t x_0 + y_0) - x_t x_0(y'_t x_0 + 2y'_t) + y'_t x_0^2}{(x_t - x_0)^2}$$

We use the position of the apex $\left(-\frac{b}{2a}, y\left(-\frac{b}{2a}\right)\right)$ to animate the character accordingly. Furthermore, we detect if the jump is not valid, in which case we switch to the ragdoll state. A jump is considered invalid if: (1) It rises again after starting the descent (i.e. multiple apexes), (2) It turns while in air, or

(3) It stops in the air. If none of these cases happen, the character lands back on the ground and absorbs the shock of the landing by doing a very short crouch.



Figure 6.7: Screenshots of our application show the character building a snowman, which then itself builds a second snowman.

We propose to showcase our control scheme with a building experience. Please refer to Fig. 6.7 for a selection of screenshots. Our application lets the user build a snowman out of snowballs, that can be rolled to variable diameters. This snowman then comes to life and can be controlled just like the original puppet character. Consequently, the snowman can then also crouch, jump and even build more snowmen. Because the snowballs making up the snowman have variable sizes, the proportions of the snowman must also be variable. We achieve this by extending specific bones of the rig, e.g. by adapting the length of the neck to accommodate for the head size. An example of this setup is shown in Fig. 6.8.

However, having an overly elongated bone would result in very stiff movements. In our case, we resolve this problem by interpolating the snowball positions between chest and pelvis with a quadratic bezier curve. The control points are given by the pelvis position, the chest position, and the point p_1 defined as:

$$p_1 = \frac{p_{pelvis} + (1 - b) \cdot p_{chest} + b \cdot p_{up}}{2} \quad (6.4)$$

where p_{up} is the position of the chest when the snowman is be standing upright. In our implementation we used a bend factor $b = 0.3$.



Figure 6.8: Our animation rig (blue) with variable bone lengths and our smoothed spine (red dotted) on a snowman with a long spine bone.

6.4 Application: Multi-Reality Games

The mechanism we propose is designed to be very close to the way we interact with physical toys. However, the remaining difference is that the character is virtual, which makes it intangible and constitutes a gap between the player and his/her toy. In order to reduce this gap, we seek to make the user feel more connected to the virtual elements. To that end, we built an application where the user is progressively immersed into the virtual world, a concept we called *Multi-Reality Games*.



Figure 6.9: (i.) Our Multi-Reality game starts with objects and characters from the real world. (ii.) Physical assets get animated using photo-realistic AR. (iii.) Moving a step forward in the RVC, the user interacts with a scene where physical and virtual assets coexist. (iv.) Finally, our game ends with a fully virtual scene with only CG-assets.

The term *Mixed Reality* (MR) refers to a set of technologies that aim to present the real and the virtual worlds unified in the same space and time. This con-

cept was first defined by Milgram et al. [Milgram et al., 1994] as a connection between the real and virtual environments, forming the *Reality-Virtuality Continuum* (RVC). This linear scale starts from the real environment itself and covers technologies such as Augmented Reality (AR) and Augmented Virtuality (AV) until reaching a fully virtual environment (see Fig. 6.10). If many of these technologies find a relevant place in the RVC, seamlessly transitioning between them — and therefore spanning the entire continuum within a single application — remains an open challenge. Here, we propose Multi-Reality experiences as a natural, fluid and seamless approach to travel throughout the RVC. In this Section, we detail an application to go from reality to virtuality, nonetheless, this could also be experienced in the opposite direction, from virtuality to reality.

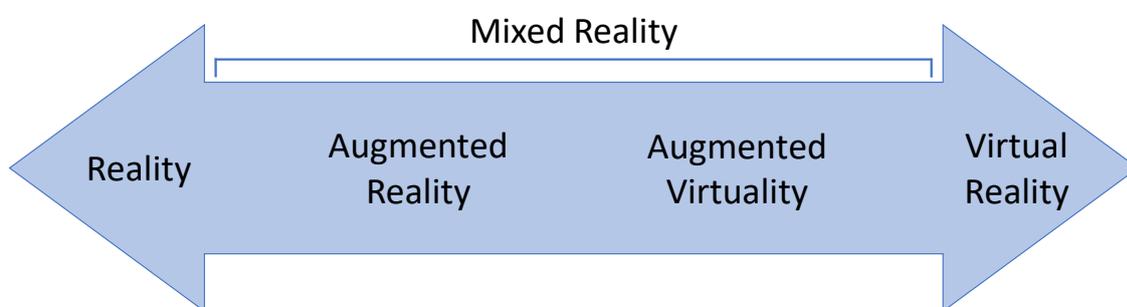


Figure 6.10: *Illustration of the Reality-Virtuality Continuum and the position of AR, AV, VR and MR in it.*

We propose an adventure game where the goal is to solve a quest to restore freedom in the hidden world of *Tasbada*. The game starts in the real world: the user decide where the Multi-Reality experience will take place and select the physical objects that will compose the real-world scene.

As soon as the user has defined a location and the real-world assets for the gaming experience, we start travelling across the RVC. We use a pre-defined 3D printed object to incorporate a narrator into the game (as shown in Fig. 6.11). We use *Props Alive* [Casas et al., 2017] to create the illusion of movement of static objects from the real world with photo-realistic renderings. Additionally, when these animated real-world objects cast shadows, we use *Shadow Retargeting* [Casas et al., 2018] to account for their deformation. Hence, our application enhances the user’s imagination by presenting extended capabilities, such as speech and movement, to inanimate objects in the real world.

Once the narrator has explained the mission to the player, we ask him to take a photo of himself (or a friend) to be transported into the game. Ex-

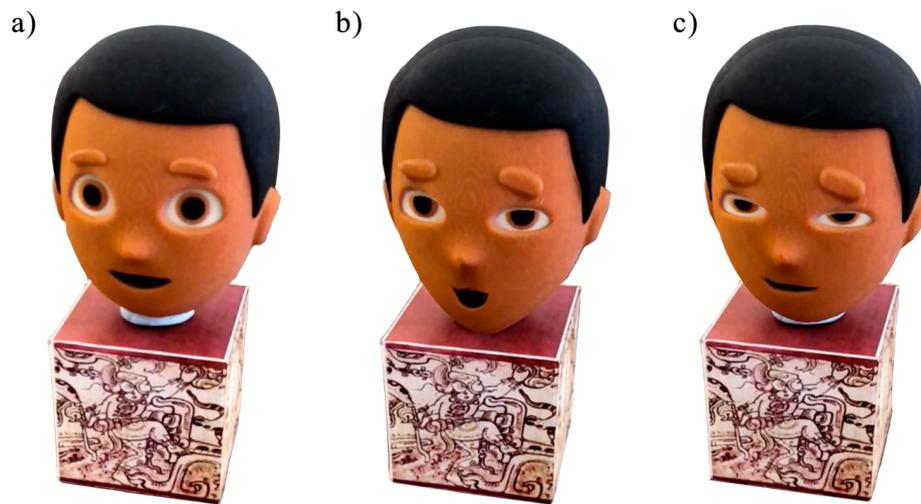


Figure 6.11: a) Static and inanimate 3D printed narrator. b-c) Narrator speaking to the user through photorealistic Augmented Reality using a mobile phone.

tending [Nijholt, 2005], we use virtual avatars for this Multi-Reality game experience; this brings closer the physical and augmented worlds as the user can feel connected to the virtual character. We use *AR Poser* [Cimen et al., 2018] to capture the user's initial pose, and we additionally extract the representative color of the clothes to apply on the avatar (Fig. 6.12).



Figure 6.12: The user's initial pose and the representative color of their clothes are applied to the virtual character of the game.

Now that the player is embodied into the virtual character, he is asked to solve a quest by unlocking several milestones in the augmented world. We use the MotionStick metaphor to let the user control his avatar with the movement of the phone. Once the milestones are accomplished and the final goal of the game is reached, we embark the user in a fully virtual world, as a reward for their achievement.

In this final stage, the player can explore the liberated world of Tasbada through the mobile phone and interact with it. Virtual objects present in the augmented reality scene remain, but the real-world scenario becomes overlaid with a fully virtual setting. This last transition allowed the user to smoothly progress one step further in the RVC, from AR to VR. The user started the game at the real end of the continuum and ended it on the totally virtual side of it.

6.5 Conclusion

In this Chapter, we have introduced a new interaction metaphor to control virtual characters. It combines advantages of both real and virtual worlds by providing a great freedom of motion, similar to the manipulation of physical toys, while augmenting the character's responses with engaging animations. We proposed solutions to the new challenges emerging from such a flexible interaction system, like the interpretation of users' gestures, the real-time formation of accurately timed animations and their adjustment to characters with variable proportions. We validated our approach with an AR application that allows to create living snowmen of any dimensions.

We additionally proposed to make the user even more engaged with the virtual characters by seamlessly traversing the Reality-Virtuality Continuum. To illustrate that theory, we successfully implemented and assembled recent techniques in Mixed Reality to provide a game where the player is progressively immersed into a virtual world. We believe that future advancements in Mixed Reality technologies would contribute for even more immersive applications.

The control interface we propose being light, if many different animations and character behaviors are added to the experience, it might become unclear what the intent of the user is. For example, gestures corresponding to a punch, a kick, a throw and a headbutt might be too similar. This can be tackled by introducing more input possibilities (such as using the tactile screen) or defining specific and more abstract gestures for certain actions, similar to what Thorne et al. [2004] propose.

As a control device, we chose the smartphone for its versatility, its prevalence and the unification of controller and camera it provides. That being said, our method is adaptable to any other device that tracks position and orientation over time. For example, one could use a VR system such as HTC Vive or Oculus Rift to grab and move a character in virtual reality; there, the user could even more closely interact with the puppet since the MotionStick

could be of length 0. VR systems also support multiple controllers, opening up possibilities like the simultaneous control of several puppets or more control on a single one.

PuppetPhone: Puppeteering Virtual Characters Using a Smartphone

C H A P T E R

7

Conclusion

Traditional hand drawn animation is a fascinating means of expression that is accessible to everyone. However, manually drawing every frame of a movie requires a tremendous amount of work and patience. The introduction of computer animation has enabled the automation of certain tasks, and therefore a huge gain in time. However, early in this thesis we observed that this new tool also yielded a less natural interaction with the scene and a more cumbersome achievement of some simple tasks. Mastering current 3D animation software requires an intense training, which makes this media much less accessible.

We started by reviewing aesthetic principles and best-practices developed in the context of classical art, illustration, and hand-drawn animation to understand the core factors at play in the field of aesthetic design. From these observations, we identified elements of the animation workflow that were particularly tedious to achieve with current techniques, and we expanded the state of the art in a way that focuses on artist-oriented animation tools. The new interfaces and algorithms we proposed were validated with the achievement of production quality results and the conduction of several user studies.

By its nature, computer animation lays at the intersection between arts and technology. On the one hand, it requires strong artistic skills to shape appealing characters and infer believable movements. On the other hand, it involves very complex algorithms for the computation and rendering of every frame. Therefore, it is very delicate to find the right balance between automation and manual inputs, and the animation workflow can easily suf-

Conclusion

fer from that. While requiring too much manual work can make the process tedious and repetitive, providing too much automation can hinder the artistic control. In this thesis, we strove to design tools that are accessible and intuitive, while still providing a fine level of control to the user.

7.1 Summary of Contributions

7.1.1 Coherent Scene Deformation

We started by working on static scenes and observed that the artistic principle of movement is particularly tedious to apply with current techniques due to the object-centric nature of deformers. Thus, in Chapter 3 we introduced an interface called Flow Curves that enables artists to quickly turn a scene with an ambiguous movement into a compelling scene by simply sketching strokes or specifying the center of interest.

A core component of our method is a deformation representation tailored to the view-dependent nature of the principle of movement. It enables the coordinate deformation of multiple objects from a single stroke, and works on any type of objects. This representation allows to work in a single viewport, without the need for any additional window. Furthermore, to save setup time and efforts, we provided algorithms to automatically identify lines of interest — possibly spanning multiple disconnected objects — that may contribute to the scene movement.

7.1.2 Motion Visualization and Manipulation

When dealing with animated scenes, a similar artistic principle comes into action, called arcs. It states that moving elements should also display visually pleasing curves over time. However, applying this principle is not accommodated by the most common animation process used by professional artists: keyframing. There, key poses are defined at specific points in time and are then interpolated. This pose-centric system provides a very indirect control on the arcs of motion. In Chapter 4, we enhanced keyframing with a space-time curve representation that allows to visualize and modify those arcs in an intuitive way.

Our system is designed to be non-intrusive to the artist's workflow. It does not require a different character representation, nor introduce any additional keyframe. Rather, when the user manipulates a Motion Curve, our system optimizes for the tangents of interpolation of existing character parameters

that match the user-constraints. Our pin-and-drag interface enables a fine level of control and abstracts technical considerations such as hierarchy and FK vs IK.

7.1.3 Performance Animation for Motion Cycles

After demonstrating that the keyframing process can be greatly enhanced, we also explored an alternative that is more movement-centric. To do so, we used performance animation which consists in mimicking the motion of an animated element. In Chapter 5, we observed that this could particularly benefit motion cycles, since repeatedly acting a movement helps improving it. We therefore proposed a system where the user first acts out several cycles of an action to create it, and then edits the curves of motion to refine it.

Since human movements are imprecise, we proposed an algorithm to extract a clean loop from the user input. We also adapted our space-time curve representation from Chapter 4 to cyclic motions. Since now the movement is not driven by a set of interpolated keyframes, much more freedom can be provided to the user for the manipulation of MoCurves. We even enabled a very precise control of the timing directly on those curves. Our method is generic enough to work with any capture device as input, using a layered approach when the dimensionality of the device is smaller than the animated content.

7.1.4 Interactive Character Control

After proposing different ways for animating virtual characters with a fine level of detail, we explored the challenge of interactively maneuvering them. Most video games already use such real-time motion control, but the detached interaction they propose is abstract and limiting. In Chapter 6, we took advantage of the power and versatility of smartphones to propose a new interaction metaphor that is close to the feeling of grasping a real puppet.

Using our tool, a user can see a virtual character through the screen and grab it. It then responds compellingly and in real-time to the user motions so that it stays at all times visible through the phone and at a fixed distance. It is similar to holding a puppet with a reach extender, with the benefit that the puppet now looks alive. We demonstrated the usability of this new mechanism with two Mixed-Reality applications.

7.2 Future Work

This thesis introduced various techniques to make the interaction with animated scenes more intuitive and accessible. Follow-up research efforts were suggested for each individual contribution, and in this Section we will discuss what we identify as two major overall directions for future work in animation interfaces.

7.2.1 Animating with Curves

Artists extensively use visual curves in order to make scenes more appealing. That was observed earlier in this thesis when investigating the principles of animation and visual arts. This is why curves are also prominent in our work: they provide a familiar tool to the artist and constitute a natural fit for many design tasks. Therefore, exploring additional curve-based interfaces is a pertinent direction for future work, especially with the emergence of more sophisticated devices (as discussed in the following Section).

In particular, in Chapters 4 and 5 we proposed two different space-time curve representations for the control of motion. Having them tested by professional animators from the Walt Disney Company was a great chance, and it allowed us to evaluate the pros and cons of each method — as elaborated in the respective Sections. Our conversations with those artists convinced us of the pertinence of using curves for the visualization and control of movement. In the future, we would like to investigate a new representation that would combine the advantages of both methods, enabling the unconstrained control of any part of the character without being intrusive to the usual workflow. This could be done through animation layers, where the first layer would conserve the imposed keyframes and the others would apply an additional transformation per frame (like a delta) for finer edits.

Earlier in this thesis we emphasized the importance of adapting the tools to the existing workflow, in order for the artists to embrace them more easily. However, when looking further into the future, we believe that the animation workflow will itself require transformations in order to better integrate the new methods. With systems like the ones we propose, where IK chains and graph editors are not necessary anymore, many optimizations can be made on the usual pipeline. Keyframing could eventually become obsolete, and animators would work with continuous movements (no frames) edited through space-time curves.

7.2.2 Accommodating New Technologies

With the continuous progress of technologies, the art of animation is in constant evolution. And as the algorithms and devices continue to innovate, the interaction systems will need to keep adapting. In particular, recent years have seen the emergence of Augmented Reality and Virtual Reality, which have the potential to strongly impact computer animation. In Chapter 6 already, we showed how exploiting those new instruments can transform the way we interact with virtual characters.

One key revolution induced by Mixed Reality technologies, and in particular VR, is the increase of dimensionality: the user can now visualize in stereoscopic 3D, has a camera disconnected from the manipulation devices, and can maneuver two 6-dof controllers simultaneously. This introduces new possibilities to visualize and manipulate virtual content, which interaction systems can definitely benefit from. For example, to continue our discussion on curves, defining them in VR is much more straightforward, as demonstrated with recent applications such as Tilt Brush [Google, 2016]. Many systems designed for computers, such as those proposed in this thesis, can thus be extended to VR and become even more intuitive due to the suppression of depth ambiguity.

Of course, new interaction challenges also arise when working with those devices. One of them is the reduction of the number of buttons; artists are used to work extensively with shortcuts on the keyboard, so switching to VR/AR will require to define alternatives. We believe that adding buttons in the virtual environment would be distractive, so we suggest studying user gestures that can be interpreted as commands as a more appropriate solution. Also, visual and physical tiredness that occurs when wearing a headset and performing large movements is an important constraint. Future user experiences will need to account for that and propose systems that accommodate an easy switch between computer screen and headset and that can be executed in a sitting position with elbows resting on the desk.

New technologies can not only innovate the way we create animations but also the way we watch them. We discussed early in this thesis how the transition from traditional hand drawn animation to 3D computer animation changed the crafting mechanism, but since animated content was still displayed on flat screens, the artistic principles that guided their design were conserved. However, watching an animation in Mixed Reality is substantially different. Bringing the viewer's attention to the relevant area of the scene and designing movements that look compelling from several points of views are particularly challenging. As a result, new artistic principles that

Conclusion

are specific to this new media will emerge, and it will be essential to provide the right tools and algorithms to apply them.

A P P E N D I X

A

Attribute limits during interpolations

In this appendix, we demonstrate that if a parameter c is constrained to stay between u and v , the limits on the tangents we defined in Equations 4.8 and 4.9 satisfy the parameter's constraints at all times.

Between two keyframes t_k and t_{k+1} , we consider that the value of $c(\boldsymbol{\theta}, t)$ is interpolated by a Bezier cubic spline:

$$\begin{bmatrix} t \\ c(\boldsymbol{\theta}, t) \end{bmatrix} = \begin{bmatrix} B_c^X(\lambda) \\ B_c^Y(\lambda) \end{bmatrix} = \begin{bmatrix} t_k \\ c(t_k) \end{bmatrix} (2\lambda^3 - 3\lambda^2 + 1) + 3\lambda(1 - \lambda)^2 \boldsymbol{\theta}_{k+} + \begin{bmatrix} t_{k+1} \\ c(t_{k+1}) \end{bmatrix} (3\lambda^2 - 2\lambda^3) - 3\lambda^2(1 - \lambda) \boldsymbol{\theta}_{k+1-},$$

where $\lambda \in [0, 1]$. Therefore, imposing $c(\boldsymbol{\theta}, t) \leq v \forall t \in [t_k, t_{k+1}]$ is equivalent to imposing $B_c^Y(\lambda) \leq v \forall \lambda \in [0, 1]$. By injecting the upper limits of Equation 4.8 into $B_c^Y(\lambda)$, with the definitions of ϕ and ψ proposed in Equation 4.9, we obtain the following inequality, where we note $mx = \max(c(t_k), c(t_{k+1}))$ for easier reading:

$$B_c^Y(\lambda) \leq \frac{c(t_k)(2\lambda^3 - 3\lambda^2 + 1) + 4(v - mx)\lambda(1 - \lambda)^2}{+ c(t_{k+1})(3\lambda^2 - 2\lambda^3) + 4(v - mx)\lambda^2(1 - \lambda)}.$$

Knowing that both $(2\lambda^3 - 3\lambda^2 + 1)$ and $(3\lambda^2 - 2\lambda^3)$ are positive for $\lambda \in [0, 1]$, and by definition both $c(t_k)$ and $c(t_{k+1})$ are lower than mx , we obtain:

$$B_{c,i}^Y(\lambda) \leq 4(v - mx)\lambda(1 - \lambda) + mx.$$

Finally, since $\lambda(1 - \lambda) \leq 0.25$ on $[0, 1]$ and $v - mx$ is positive (we suppose that the limit is respected at keyframes), we end up with the desired result:

Attribute limits during interpolations

$$B_{c,i}^Y(\lambda) \leq v.$$

The same demonstration is valid for the lower limit. Indeed, if u is the lower limit of c , that means that $-u$ is the upper limit of $-c$, which brings us back to the above computation.

References

- [Ahmed et al., 2003] Amr Ahmed, Farzin Mokhtarian, and Adrian Hilton. Cyclification of human motion for animation synthesis. In *Eurographics 2003 - Short Presentations*, 2003.
- [Arikan and Forsyth, 2002] Okan Arikan and David A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, 2002.
- [Aristidou et al., 2017] Andreas Aristidou, Joan Lasenby, Yiorgos Chrysanthou, and Ariel Shamir. Inverse kinematics techniques in computer graphics: A survey. *Computer Graphics Forum*, 37(6):35–58, 2017.
- [Au et al., 2008] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):44:1–44:10, 2008.
- [Bailey et al., 2009] Reynold Bailey, Ann McNamara, Nisha Sudarsanam, and Cindy Grimm. Subtle gaze direction. *ACM Trans. Graph.*, 28(4):100:1–100:14, 2009.
- [Botsch et al., 2007] Mario Botsch, Mark Pauly, Martin Wicke, and Markus Gross. Adaptive space deformations based on rigid cells. *Computer Graphics Forum*, 26(3):339–347, 2007.
- [Brosz et al., 2007] John Brosz, Faramarz F. Samavati, M. Sheelagh T. Carpendale, and Mario Costa Sousa. Single camera flexible projection. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, pages 33–42, 2007.

References

- [Button, 2002] Bryce Button. *Nonlinear Editing: Storytelling, Aesthetics, and Craft*. CMP Books, 2002.
- [Carroll et al., 2010] Robert Carroll, Aseem Agarwala, and Maneesh Agrawala. Image warps for artistic perspective manipulation. *ACM Trans. Graph.*, 29(4):127:1–127:9, 2010.
- [Casas et al., 2017] Llogari Casas, Maggie Kosek, and Kenny Mitchell. Props alive: A framework for augmented reality stop motion animation. In *2017 IEEE 10th Workshop on Software Engineering and Architectures for Realtime Interactive Systems*, 2017.
- [Casas et al., 2018] Llogari Casas, Matthias Fauconneau, Maggie Kosek, , Kieran Mclister, and Kenny Mitchell. Image based proximate shadow retargeting. In *Proceedings of the Computer Graphics and Visual Computing (CGVC) Conference 2018*, 2018.
- [Chai and Hodgins, 2005] Jinxiang Chai and Jessica K. Hodgins. Performance animation from low-dimensional control signals. *ACM Trans. Graph.*, 24(3):686–696, 2005.
- [Chai and Hodgins, 2007] Jinxiang Chai and Jessica K. Hodgins. Constraint-based motion optimization using a statistical dynamic model. In *ACM SIGGRAPH 2007 Papers*, 2007.
- [Choi and Lee, 2016] Myung G. Choi and Kang H. Lee. Points-based user interface for character posing. *Computer Animation and Virtual Worlds*, 27(3-4):213–220, 2016.
- [Choi et al., 2008] Byungkuk Choi, Mi You, and Junyong Noh. Extended spatial keyframing for complex character animation. *Comput. Animat. Virtual Worlds*, 19(3-4):175–188, 2008.
- [Choi et al., 2016] Byungkuk Choi, Roger Blanco i Ribera, J. P. Lewis, Yeongho Seol, Seokpyo Hong, Haegwang Eom, Sunjin Jung, and Junyong Noh. Sketchimo: Sketch-based motion editing for articulated characters. *ACM Trans. Graph.*, 35(4):146:1–146:12, 2016.
- [Chung et al., 2015] Se-Joon Chung, Junggon Kim, Shangchen Han, and Nancy S. Pollard. Quadratic encoding for hand pose reconstruction from multi-touch input. In *EG 2015 - Short Papers*, 2015.
- [Cimen et al., 2018] Gokcen Cimen, Christoph Maurhofer, Bob Sumner, and Martin Guay. Ar poser: Automatically augmenting mobile pictures with digital avatars imitating poses. In *12th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing 2018*, 2018.

- [Cohen and Guibas, 1997] Scott D. Cohen and Leonidas J. Guibas. Partial matching of planar polylines under similarity transformations. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 777–786, 1997.
- [Cohen, 1992] Michael F. Cohen. Interactive spacetime control for animation. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pages 293–302, 1992.
- [Cole et al., 2006] Forrester Cole, Doug DeCarlo, Adam Finkelstein, Kenrick Kin, Keith Morley, and Anthony Santella. Directing gaze in 3d models with stylized focus. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, pages 377–387, 2006.
- [Coleman and Singh, 2004] Patrick Coleman and Karan Singh. Ryan: Rendering your animation nonlinearly projected. In *Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering*, pages 129–156, 2004.
- [Coleman et al., 2005] Patrick Coleman, Karan Singh, Leon Barrett, Nisha Sudarsanam, and Cindy Grimm. 3d screen-space widgets for non-linear projection. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pages 221–228, 2005.
- [Coleman et al., 2008] Patrick Coleman, Jacobo Bibliowicz, Karan Singh, and Michael Gleicher. Staggered poses: A character motion representation for detail-preserving editing of pose and coordinated timing. In *Proc. of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 137–146, 2008.
- [Coros et al., 2010] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Trans. Graph.*, 29(4):Article 130, 2010.
- [Cui and Mousas, 2018] Yaoyuan Cui and Christos Mousas. Master of puppets: An animation-by-demonstration computer puppetry authoring framework. *3D Research*, 9(1):158:1–158:14, 2018.
- [D. D. Willis et al., 2011] Karl D. D. Willis, Ivan Poupyrev, and Takaaki Shiratori. Motionbeam: A metaphor for character interaction with handheld projectors. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 1031–1040, 2011.
- [Davis et al., 2003] James Davis, Maneesh Igarashi, Erika Chuang, Zoran Popovic, and David Salesin. A sketching interface for articulated figure animation. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 320–328, 2003.

References

- [Dontcheva et al., 2003] Mira Dontcheva, Gary Yngve, and Zoran Popović. Layered acting for character animation. *ACM Trans. Graph.*, 22(3):409–416, 2003.
- [Geijtenbeek et al., 2012] Thomas Geijtenbeek, Nicolas Pronost, and A. Frank van der Stappen. Simple data-driven control for simulated bipeds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 211–219, 2012.
- [Geijtenbeek et al., 2013] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 32(6):206:1–206:11, 2013.
- [Glatstein, 2013] Jeremy Glatstein. *Formal Visual Analysis: The Elements & Principles of Composition*. 2013.
- [Glauser et al., 2016] Oliver Glauser, Wan-Chun Ma, Daniele Panozzo, Alec Jacobson, Otmar Hilliges, and Olga Sorkine-Hornung. Rig animation with a tangible and modular input device. *ACM Trans. Graph.*, 35(4):144:1–144:11, 2016.
- [Gleicher, 1997] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, pages 139–ff., 1997.
- [Google, 2016] Google. Tilt brush, 2016.
- [Guay et al., 2013] Martin Guay, Marie-Paule Cani, and Remi Ronfard. The line of action: an intuitive interface for expressive character posing. *ACM Transactions on Graphics*, 32(6):205:1–205:8, 2013.
- [Guay et al., 2015] Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. Space-time sketching of character animation. *ACM Trans. Graph.*, 34(4):118:1–118:10, 2015.
- [Haegwang et al., 2014] Eom Haegwang, Choi Byungkuk, and Noh Junyong. Data-driven reconstruction of human locomotion using a single smartphone. *Computer Graphics Forum*, 33(7):11–19, 2014.
- [Hahn et al., 2015] Fabian Hahn, Frederik Mutzel, Stelian Coros, Bernhard Thomaszewski, Maurizio Nitti, Markus Gross, and Robert W. Sumner. Sketch abstractions for character posing. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 185–191, 2015.
- [Harvey and Pal, 2018] Félix G. Harvey and Christopher Pal. Recurrent transition networks for character locomotion. In *SIGGRAPH Asia 2018 Technical Briefs*, pages 4:1–4:4, 2018.
- [Holden et al., 2016] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.*, 35(4):138:1–138:11, 2016.

- [Holden et al., 2017] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Trans. Graph.*, 36(4):42:1–42:13, 2017.
- [Horn, 1983] Berthold K. P. Horn. The curve of least energy. *ACM Trans. Math. Softw.*, 9(4):441–460, 1983.
- [Hsu et al., 2007] Eugene Hsu, Marco da Silva, and Jovan Popović. Guided time warping for motion editing. In *Proc. of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 45–52, 2007.
- [Hua and Qin, 2003] Jing Hua and Hong Qin. Free-form deformations via sketching and manipulating scalar fields. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications*, pages 328–333, 2003.
- [Huang et al., 2013] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65:1–65:8, 2013.
- [Igarashi et al., 2005] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. Spatial keyframing for performance-driven animation. In *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 107–115, 2005.
- [Jeon et al., 2010] Jaewoong Jeon, Hyunho Jang, Soon-Bum Lim, and Yoon-Chul Choy. A sketch interface to empower novices to create 3d animations. *Computer Animation and Virtual Worlds*, 21(3-4):423–432, 2010.
- [Jin et al., 2015] Ming Jin, Dan Gopstein, Yotam Gingold, and Andrew Nealen. Animesh: Interleaved animation, modeling, and editing. *ACM Trans. Graph.*, 34(6):207:1–207:8, 2015.
- [Kass et al., 1988] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.
- [Kho and Garland, 2005] Youngihn Kho and Michael Garland. Sketching mesh deformations. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 147–154, 2005.
- [Kim et al., 2009] Manmyung Kim, Kyunglyul Hyun, Jongmin Kim, and Jehee Lee. Synchronized multi-character motion editing. In *ACM SIGGRAPH 2009 Papers*, pages 79:1–79:9, 2009.
- [Kim et al., 2012] Jongmin Kim, Yeongho Seol, and Jehee Lee. Realtime performance animation using sparse 3d motion sensors. In *Motion in Games*, pages 31–42, 2012.

References

- [Kim et al., 2013] Jongmin Kim, Yeongho Seol, and Jehee Lee. Human motion reconstruction from sparse 3d motion sensors using kernel cca-based regression. *Computer Animation and Virtual Worlds*, 24(6):565–576, 2013.
- [Koyama and Goto, 2018] Yuki Koyama and Masataka Goto. Optimo: Optimization-guided motion editing for keyframe character animation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 161:1–161:12, 2018.
- [Kraevoy et al., 2009] Vladislav Kraevoy, Alla Sheffer, and Michiel van de Panne. Modeling from contour drawings. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 37–44, 2009.
- [Laszlo et al., 2000] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive control for physically-based animation. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 201–208, 2000.
- [Lee et al., 2002] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. In *Proc. of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 491–500, 2002.
- [Lehrmann et al., 2014] Andreas M. Lehrmann, Peter V. Gehler, and Sebastian Nowozin. Efficient nonlinear markov models for human motion. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1314–1321, 2014.
- [Lidwell et al., 2003] William Lidwell, Kritina Holden, and Jill Butler. *Universal principles of design*. Rockport publ. cop., 2003.
- [Lin et al., 2010] Juncong Lin, Takeo Igarashi, Jun Mitani, and Greg Saul. A sketching interface for sitting-pose design. *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, 18(11):111–118, 2010.
- [Liu et al., 2010] Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. *ACM Trans. Graph.*, 29(4):128:1–128:10, 2010.
- [Liu et al., 2011] 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*, pages 133–140, 2011.
- [Liu et al., 2015] Tianqiang Liu, Jim McCann, Wilmot Li, and Thomas Funkhouser. Composition-aware scene optimization for product images. *Comput. Graph. Forum*, 34(2):13–24, 2015.
- [Lockwood and Singh, 2012] Noah Lockwood and Karan Singh. Finger walking:

- Motion editing with contact-based hand performance. In *Proc. of the ACM SIG-GRAPH/Eurographics Symposium on Computer Animation*, pages 43–52, 2012.
- [Lockwood and Singh, 2016] Noah Lockwood and Karan Singh. Gestural motion editing using mobile devices. In *Proceedings of the 9th International Conference on Motion in Games*, pages 25–30, 2016.
- [Mahmudi et al., 2016] Mentar Mahmudi, Pawan Harish, Benoît Le Callennec, and Ronan Boulic. Artist-oriented 3d character posing from 2d strokes. *Comput. Graph.*, 57(C):81–91, 2016.
- [Martin and Neff, 2012] Tyler Martin and Michael Neff. Interactive quadruped animation. In *MIG*, pages 208–219, 2012.
- [Mehra et al., 2009] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. *ACM Trans. Graph.*, 28(5):137:1–137:10, 2009.
- [Mi et al., 2009] Xiaofeng Mi, Doug DeCarlo, and Matthew Stone. Abstraction of 2d shapes in terms of parts. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, pages 15–24, 2009.
- [Milgram et al., 1994] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Mixed reality (mr) reality-virtuality (rv) continuum. *Systems Research*, 2351:282–292, 1994.
- [Milliron et al., 2002] Tim Milliron, Robert J. Jensen, Ronen Barzel, and Adam Finkelstein. A framework for geometric warps and deformations. *ACM Trans. Graph.*, 21(1):20–51, 2002.
- [Min and Chai, 2012] Jianyuan Min and Jinxiang Chai. Motion graphs++: A compact generative model for semantic motion analysis and synthesis. *ACM Trans. Graph.*, 31(6):153:1–153:12, 2012.
- [Min et al., 2009] 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, 2009.
- [Mori et al., 2005] Greg Mori, Serge Belongie, and Jitendra Malik. Efficient shape matching using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(11):1832–1837, 2005.
- [Mosek, 2010] APS Mosek. The mosek optimization software. *Online at <http://www.mosek.com>*, 54(2-1), 2010.
- [Mukai and Kuriyama, 2009] Tomohiko Mukai and Shigeru Kuriyama. Pose-timeline for propagating motion edits. In *Proc. of the 2009 ACM SIG-GRAPH/Eurographics Symposium on Computer Animation*, pages 113–122, 2009.

References

- [Mukai, 2011] Tomohiko Mukai. Motion rings for interactive gait synthesis. In *Symposium on Interactive 3D Graphics and Games*, pages 125–132, 2011.
- [Mumford, 1994] David Mumford. Elastica and computer vision. In *Algebraic Geometry and its Applications*, pages 491–506, 1994.
- [Nealen et al., 2005] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.
- [Nebel, 1999] Jean-Christophe Nebel. Keyframe interpolation with self-collision avoidance. In *Computer Animation and Simulation '99*, pages 77–86, 1999.
- [Neff et al., 2007] Michael Neff, Irene Albrecht, and Hans-Peter Seidel. Layered performance animation with correlation maps. *Comput. Graph. Forum*, 26:675–684, 2007.
- [Nijholt, 2005] A. Nijholt. Meetings in the virtuality continuum: send your avatar. In *2005 International Conference on Cyberworlds (CW'05)*, pages 8–82, 2005.
- [Oore et al., 2002] Sageev Oore, Demetri Terzopoulos, and Geoffrey Hinton. A desktop input device and interface for interactive 3D character animation. In *Proceedings of the Graphics Interface 2002 Conference, May 27-29, 2002, Calgary, Alberta, Canada*, pages 133–140, 2002.
- [Orbay et al., 2012] Günay Orbay, Mehmet Ersin Yümer, and Levent B. Kara. Sketch-based aesthetic product form exploration from existing images using piecewise clothoid curves. *J. Vis. Lang. Comput.*, 23(6):327–339, 2012.
- [Öztireli et al., 2013] A. Cengiz Öztireli, Ilya Baran, Tiberiu Popa, Boris Dalstein, Robert W. Sumner, and Markus Gross. Differential blending for expressive sketch-based posing. In *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–164, 2013.
- [Pascu et al., 2013] Tudor Pascu, Martin White, and Zeeshan Patoli. Motion capture and activity tracking using smartphone-driven body sensor networks. In *Third International Conference on Innovative Computing Technology (INTECH 2013)*, pages 456–462, 2013.
- [Rhodin et al., 2015] Helge Rhodin, James Tompkin, Kwang In Kim, Edilson de Aguiar, Hanspeter Pfister, Hans-Peter Seidel, and Christian Theobalt. Generalizing wave gestures from sparse examples for real-time character control. *ACM Trans. Graph.*, 34(6):181:1–181:12, 2015.
- [Rose et al., 1996] 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*, pages 147–154, 1996.
- [Sederberg and Parry, 1986] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.
- [Shi et al., 2007] Xiaohan Shi, Kun Zhou, Yiying Tong, Mathieu Desbrun, Hujun Bao, and Baining Guo. Mesh puppetry: Cascading optimization of mesh deformation with inverse kinematics. *ACM Trans. Graph.*, 26(3):81–89, 2007.
- [Shiratori and Hodgins, 2008] Takaaki Shiratori and Jessica K. Hodgins. Accelerometer-based user interfaces for the control of a physically simulated character. In *ACM SIGGRAPH Asia 2008 Papers*, pages 123:1–123:9, 2008.
- [Shiratori et al., 2013] Takaaki Shiratori, Moshe Mahler, Warren Trezevant, and Jessica K. Hodgins. Expressing animated performances through puppeteering. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 59–66, 2013.
- [Silva et al., 1999] Fernando Wagner da Silva, Luiz Velho, Jonas Gomes, and Siome Goldenstein. Motion cyclification by time x frequency warping. In *Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing*, pages 49–58, 1999.
- [Silverman, 1985] Bernard W. Silverman. Some aspects of the spline smoothing approach to nonparametric regression curve fitting (with discussion). *Journal of the Royal Statistical Society, Ser.B*, 47(1):1–52, 1985.
- [Singh and Fiume, 1998] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 405–414, 1998.
- [Slyper and Hodgins, 2008] Ronit Slyper and Jessica K. Hodgins. Action capture with accelerometers. In *Proc. of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 193–199, 2008.
- [Song et al., 2017] Jaewon Song, Roger Blanco i Ribera, Kyungmin Cho, Mi You, J. P. Lewis, Byungkuk Choi, and Junyong Noh. Sparse rig parameter optimization for character animation. *Comput. Graph. Forum*, 36(2):85–94, 2017.
- [Sorkine and Alexa, 2007] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, pages 109–116, 2007.
- [Sorkine et al., 2004] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa,

References

- Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Symposium on Geometry Processing*, volume 71, pages 175–184, 2004.
- [Sumner et al., 2007] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Trans. Graph.*, 26(3), 2007.
- [Sýkora et al., 2005] Daniel Sýkora, Jan Buriánek, and Jiří Žára. Sketching cartoons by example. In *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 27–34, 2005.
- [Sýkora et al., 2009] Daniel Sýkora, John Dingliana, and Steven Collins. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, pages 25–33, 2009.
- [Tagliasacchi et al., 2009] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or. Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 Papers*, pages 71:1–71:9, 2009.
- [Tautges et al., 2011] Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. Motion reconstruction using sparse accelerometer data. *ACM Trans. Graph.*, 30(3):18:1–18:12, 2011.
- [Terra and Metoyer, 2004] Sílvio C. L. Terra and Ronald A. Metoyer. Performance timing for keyframe animation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 253–258, 2004.
- [Terra and Metoyer, 2007] Sílvio C. L. Terra and Ronald A. Metoyer. A performance-based technique for timing keyframe animations. *Graphical Models*, 69(2):89–105, 2007.
- [Thomas and Johnston, 1981] Frank Thomas and Ollie Johnston. *The illusion of life : Disney animation*. Disney Editions, 1981.
- [Thorne et al., 2004] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: An interface for sketching character motion. In *ACM SIGGRAPH 2004 Papers*, pages 424–431, 2004.
- [Tsai et al., 1994] Ping-Sing Tsai, Mubarak Shah, Katharine Keiter, and Takis Kasparis. Cyclic motion detection for motion based recognition. *Pattern Recognition*, 27(12):1591 – 1603, 1994.
- [Ullman, 1976] Shimon Ullman. Filling the gaps: The shape of subjective contours and a model for their generation. *Biological Cybernetics*, pages 1–6, 1976.
- [Vinayak et al., 2016] Vinayak, Devarajan Ramanujan, Cecil Piya, and Karthik Ramani. Mobisweep: Exploring spatial design ideation using a smartphone as a

- hand-held reference plane. In *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 12–20, 2016.
- [Vishwanath et al., 2013] A. V. Vishwanath, R. Arun Srivatsan, and M. Ramanathan. Minimum area enclosure and alpha hull of a set of freeform planar closed curves. *Comput. Aided Des.*, 45(3):751–763, 2013.
- [Walther-Franks et al., 2012] Benjamin Walther-Franks, Marc Herrlich, Thorsten Karrer, Moritz Wittenhagen, Roland Schröder-Kroll, Rainer Malaka, and Jan Borchers. Dragimation: Direct manipulation keyframe timing for performance-based animation. In *Proceedings of Graphics Interface 2012*, pages 101–108, 2012.
- [Wang et al., 2008] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298, 2008.
- [Wang et al., 2012] Xin Wang, Qing Ma, and Wanliang Wang. Kinect driven 3d character animation using semantical skeleton. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, pages 159–163, 2012.
- [Wei and Chai, 2011] Xiaolin Wei and Jinxiang Chai. Intuitive interactive human-character posing with millions of example poses. *IEEE Computer Graphics and Applications*, 31(4):78–88, 2011.
- [Wertheimer, 1938] M. Wertheimer. *Laws of organization in perceptual forms*. 1938.
- [Williams and Jacobs, 1997] Lance R. Williams and David W. Jacobs. Stochastic completion fields: A neural model of illusory contour shape and salience. *Neural Comput.*, 9(4):837–858, 1997.
- [Witkin and Kass, 1988] Andrew Witkin and Michael Kass. Spacetime constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 159–168, 1988.
- [Witkin and Popovic, 1995] Andrew Witkin and Zoran Popovic. Motion warping. In *Proc. of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 105–108, 1995.
- [Yamane and Nakamura, 2003] Katsu Yamane and Yoshihiko Nakamura. Natural motion animation through constraining and deconstraining at will. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):352–360, 2003.
- [Yin et al., 2007] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. In *ACM SIGGRAPH 2007 Papers*, 2007.
- [Yoo et al., 2015] Innfarn Yoo, Michel Abdul Massih, Illia Ziamtsov, Raymond Hassan, and Bedrich Benes. Motion retiming by using bilateral time control surfaces. *Comput. Graph.*, 47(C):59–67, 2015.

References

- [Yoshizaki et al., 2011] Wataru Yoshizaki, Yuta Sugiura, Albert C. Chiou, Sunao Hashimoto, Masahiko Inami, Takeo Igarashi, Yoshiaki Akazawa, Katsuaki Kawachi, Satoshi Kagami, and Masaaki Mochimaru. An actuated physical puppet as an input device for controlling a digital manikin. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 637–646, 2011.
- [Zhang and van de Panne, 2018] Xinyi Zhang and Michiel van de Panne. Data-driven autocompletion for keyframe animation. In *MIG '18: Motion, Interaction and Games (MIG '18)*, 2018.
- [Zimmermann et al., 2007] Johannes Zimmermann, Andrew Nealen, and Marc Alexa. Silsketch: Automated sketch-based editing of surface meshes. In *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, pages 23–30, 2007.
- [Zordan et al., 2014] Victor Zordan, David Brown, Adriano Macchietto, and KangKang Yin. Control of rotational dynamics for ground and aerial behavior. *IEEE Transactions on Visualization and Computer Graphics*, 20(10):1356–1366, 2014.