PuppetPhone: Puppeteering Virtual Characters Using a Smartphone

RAPHAEL ANDEREGG, ETH Zürich
LOÏC CICCONE, ETH Zürich
ROBERT W. SUMNER, Disney Research Zürich and ETH Zürich

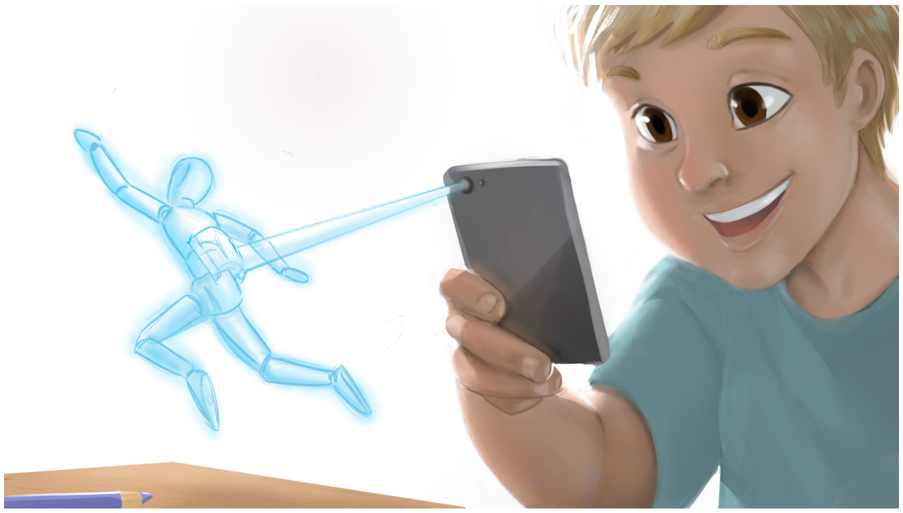

Fig. 1. Using our new system, a player is able to manipulate a virtual puppet using a smartphone. The character responds compellingly and in real-time to the user motions, so that is stays at all times at a fixed distance from the phone.

Video games enable the representation and control of characters that can agilely evolve in virtual environments. However, the detached character interaction they propose – often using a push-button metaphor – is far from the satisfactory feeling of grasping and moving physical toys. In this paper, we propose a new interaction metaphor that reduces the gap between physical toys and virtual characters. The user moves a smartphone around, and a puppet that responds in real time to the manipulations is seen through the screen. The virtual character moves in order to follow the user gestures, as if it was attached to the phone via a 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 compelling animations. Using the smartphone, our method ties together the control of the character and camera into a single interaction mechanism. We validate our system by presenting an application in Augmented Reality.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; • **Computing methodologies** → *Animation*;

Additional Key Words and Phrases: Character control, Interaction, Puppet, Smartphone

Authors' addresses: Raphael Anderegg, ETH Zürich; Loïc Ciccone, ETH Zürich; Robert W. Sumner, Disney Research Zürich, ETH Zürich.

ACM Reference Format:
Raphael Anderegg, Loïc Ciccone, and Robert W. Sumner. 2018. PuppetPhone: Puppeteering Virtual Characters Using a Smartphone. 1, 1 (September 2018), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

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 our work, 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. The character reacts in real time to the user's motions, similar to a physical puppet, 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 neighbouring environment, into a weighted combination of predefined animations (see Section 3). 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 4, 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.

2 RELATED WORK

Interaction metaphors. Interacting with virtual characters requires a correspondence between user commands and characters' actions. Most often in video games, 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.). Research works have explored more sophisticated interaction methods to specify character movements and displacements using sketch abstractions [Guay et al. 2015; Thorne et al. 2004], point trajectories [Jeon et al. 2010; Lee et al. 2002; Min et al. 2009], or finger performances on a touch-sensitive surface [Lockwood and Singh 2012]. However, these methods do not permit an interactive control since the curves or contact points need to be entirely specified before the character can move.
Another approach to interactively manipulate a virtual character is to mimic the motion, usually using a full body motion-capture suit. However, even if many works aimed to reduce the number of sensors required when performing the motion [Chai and Hodgins 2005; Kim et al. 2012; Liu et al. 2011; Oore et al. 2002; Shiratori and Hodgins 2008; Tautges et al. 2011], this approach requires specialized devices that casual users rarely possess. In contrast, our method solely needs a smartphone, which many individuals already own. It makes our tool accessible by a very large audience, and only involves a single hand to be operated. Other works aim to interactively animate virtual characters using abstract gestures [Cui and Mousas 2018; Rhodin et al. 2015]. In those techniques, user movements are linked to specific actions of the character, which make it possible to animate any type of character using different devices and body parts. However the interaction is very abstract and unnatural, in contrast to our grasping metaphor.

Smartphone as a control device. 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 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 take benefit of the tactile screen to drive a virtual character, using a push-button approach as described above. Very 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. Closer to our work, Willis et al. [2011] propose to attach 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.

Interactive motion generation. Most video games use underconstrained interfaces. In that case, full character animations can be generated using motion data-bases [Arikan and Forsyth 2002; Holden et al. 2017, 2016; Min and Chai 2012], physical simulations [Coros et al. 2010; Geijtenbeek et al. 2013; Laszlo et al. 2000; Yin et al. 2007] or even both [Geijtenbeek et al. 2012; Liu et al. 2010; Zordan et al. 2014]. Cases where the interface is not underconstrained are when the character has a very simple configuration, when the manipulation is not interactive – e.g. layered approach [Ciccone et al. 2017; Dontcheva et al. 2003; Neff et al. 2007] – or when the user is using an input device with a large number of degrees of freedom – e.g. motion capture suit [Song et al. 2017] –, none of which is our case. In our system, we opt for a database approach. We require very few preexisting motions, and we compose blended ones on the fly using a technique based on the inverse distance weighting.

3 APPROACH

3.1 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 it's 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. 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.

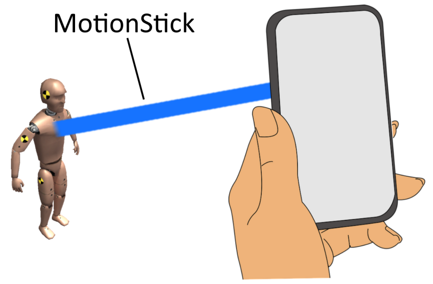

Fig. 2. Using the MotionStick metaphor, a virtual character is manipulated as if it was attached to the end of an imaginary stick fixed to the smartphone.

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 4 with our implementation prototype, where our application does not require any additional user input interface.

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.

3.2 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 prepared 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.

3.3 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.

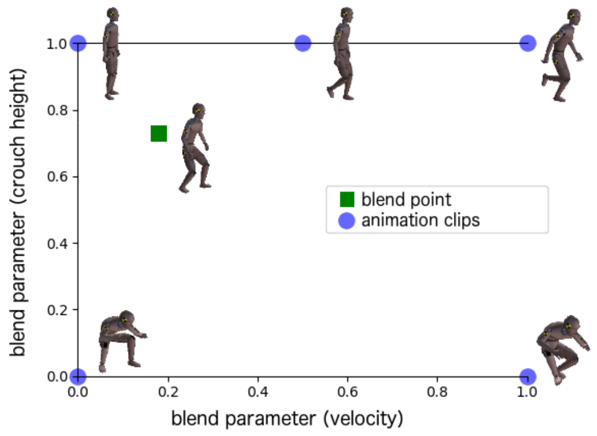

Fig. 3. An example of a blend-space graph with the predefined animation clips *idle*(0,1),*walking*(0.5,1), *running*(1,1), *idle crouching*(0,0) and *walking crouching*(1,0). In green is the desired blended animation with parameters $p_{vel} = 0.2$ and $p_{height} = 0.73$.

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 $p_i = (p_{i,1}, ..., p_{i,N})$. Given a new point p in that space, the blending algorithm returns a new animation that is a combination of the neighbouring ones. Fig. 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 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:

$$p(u) = \begin{cases} \frac{\sum_k w_k(u) p_k^*}{\sum_k w_k(u)}, & \text{if } d(u, u_k^*) \neq 0 \text{ for all } i \\ p_k^*, & \text{if } d(u, u_k^*) = 0 \text{ for some } i \end{cases} \quad (1)$$

with

$$w_k(u) = \frac{1}{d(u, u_k^*)^q} \quad (2)$$

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. 4, we show the lookup table obtained from the example in Fig. 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.

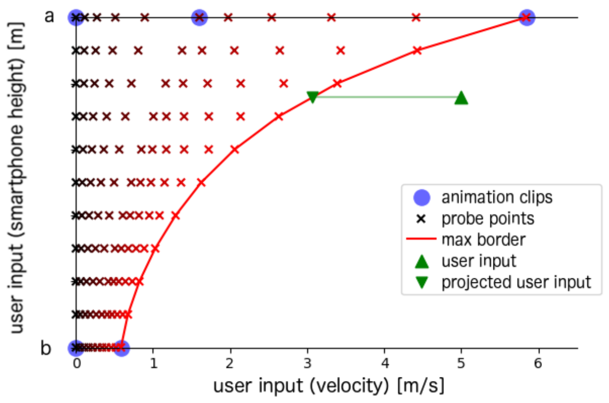

Fig. 4. The generated lookup graph used in our implementation, with a user input that has to be projected to be inside the area of well defined blend parameters. The blend parameter for velocity p_{vel} is encoded in the red color channel of the probe points, ranging from 0 to 1.

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.

4 APPLICATION

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 3. Please refer to Fig. 6 or the accompanying video to observe the principle features.

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.

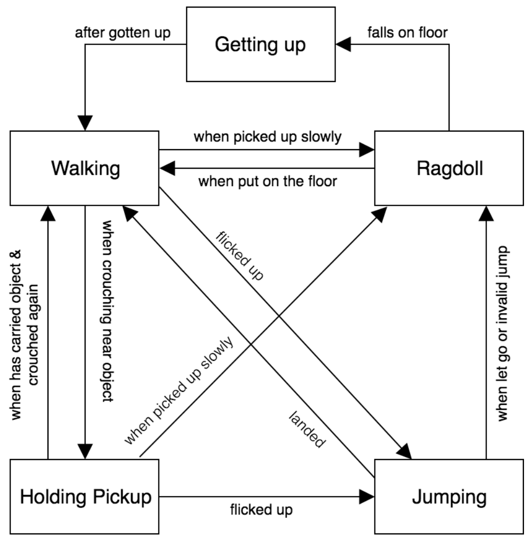

Fig. 5. The state machine used in our implementation. Arrows indicate our events that switch to a new active state.

Please refer to Fig. 5 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 3.3. 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. Please refer to the accompanying video from the timestamp 00:41.

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

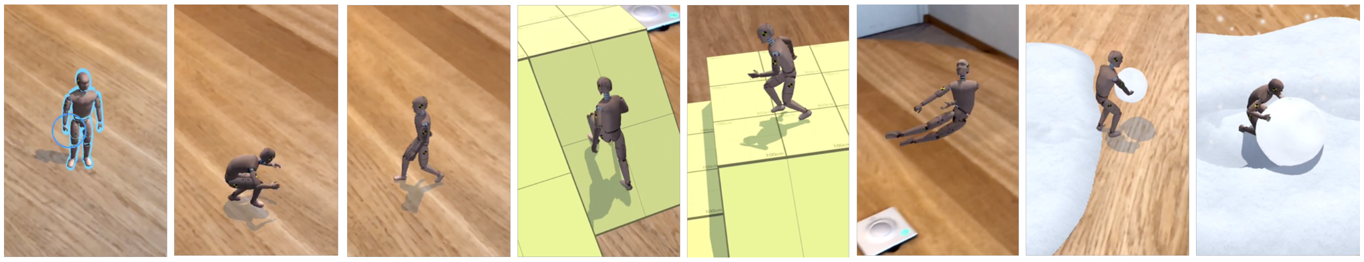
Fig. 6. An outline is shown when an object is in focus to be grabbed (left). When pressing down on the touchscreen the character is controlled with the MotionStick metaphor using smartphone movements. Animations are generated to fit to any given situation and user input.

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:

$$\begin{aligned} y(x) &= ax^2 + bx + c \\ \text{where} \quad a &= \frac{y'_t(x_t - x_0) - y_t + y_0}{(x_t - x_0)^2} \\ b &= \frac{y'_t(x_0^2 - x_t^2) - 2x_t(y_t - y_0)}{(x_t - x_0)^2} \\ c &= \frac{x_t^2(y'_t x_0 + y_0) - x_t x_0(y'_t x_0 + 2y'_t) + y'_t x_0^2}{(x_t - x_0)^2} \end{aligned} \tag{3}$$

We use the position of the apex $\left(-\frac{b}{2a}, y(-\frac{b}{2a})\right)$ 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.

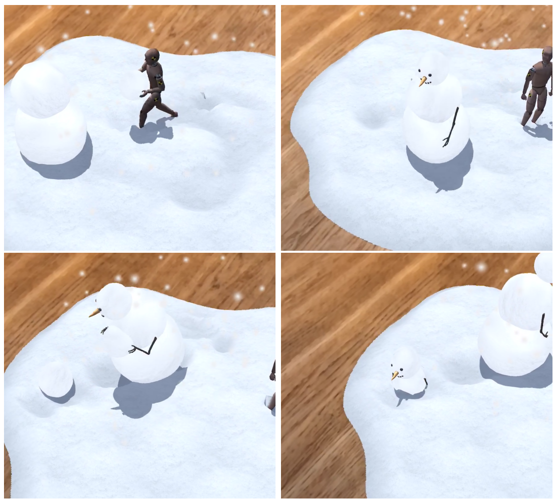
Fig. 7. Screenshots of our application show the character building a snowman, which then itself builds a second snowman.

We propose to showcase our control scheme with a building experience. Please refer to Fig. 7 for a selection of screenshots or to the accompanying video from timestamp 01:06. 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. 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:

$$p_1 = \frac{p_{pelvis} + (1 - b) \cdot p_{chest} + b \cdot p_{up}}{2} \tag{4}$$

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

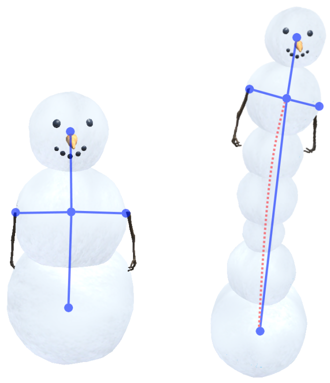
Fig. 8. Our animation rig (blue) with variable bone lengths and our smoothed spine (red dotted) on a snowman with a long spine bone.

5 LIMITATIONS AND FUTURE WORK

The control interface 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 or defining specific user 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 controlling several puppets at the same time or having more control on a single puppet. Another direction for future work would be to implement a multiplayer mode, so that several smartphones can control different puppets in the same environment and even interact with each other (e.g. shake hands or fight).

6 CONCLUSION

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 Augmented Reality application that allows to create living snowmen of any dimensions.

ACKNOWLEDGEMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642841.

REFERENCES

Okan Arikan and D. A. Forsyth. 2002. Interactive Motion Generation from Examples. *ACM Trans. Graph.* 21, 3 (2002), 483–490.

Jinxiang Chai and Jessica K. Hodgins. 2005. Performance Animation from Low-dimensional Control Signals. In *ACM SIGGRAPH 2005 Papers.* 686–696.

Loïc Ciccone, Martin Guay, Maurizio Nitti, and Robert W. Sumner. 2017. Authoring Motion Cycles. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation.* 8:1–8:9.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Trans. Graph.* 29, 4 (2010), Article 130.

Yaoyuan Cui and Christos Mousas. 2018. Master of Puppets: An Animation-by-Demonstration Computer Puppetry Authoring Framework. *3D Research* 9, 1 (2018), 5.

Karl D. D. Willis, Ivan Poupyrev, and Takaaki Shiratori. 2011. MotionBeam: A metaphor for character interaction with handheld projectors. In *Conference on Human Factors in Computing Systems - Proceedings.* 1031–1040.

Mira Dontcheva, Gary Yngve, and Zoran Popović. 2003. Layered Acting for Character Animation. In *ACM SIGGRAPH 2003 Papers.* 409–416.

T. Geijtenbeek, N. Pronost, and A. F. van der Stappen. 2012. Simple Data-driven Control for Simulated Bipeds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 211–219.

Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. 2013. Flexible Muscle-Based Locomotion for Bipedal Creatures. *ACM Transactions on Graphics* 32, 6 (2013).

Martin Guay, Rémi Ronfard, Michael Gleicher, and Marie-Paule Cani. 2015. Space-time Sketching of Character Animation. *ACM Trans. Graph.* 34, 4 (2015), 118:1–118:10.

Eom Haegwang, Choi Byungkuk, and Noh Junyong. 2014. Data-Driven Reconstruction of Human Locomotion Using a Single Smartphone. *Computer Graphics Forum* 33, 7 (2014), 11–19.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned Neural Networks for Character Control. *ACM Trans. Graph.* 36, 4 (2017), 42:1–42:13.

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4 (2016), 138:1–138:11.

Jaewoong Jeon, Hyunho Jang, Soon-Bum Lim, and Yoon-Chul Choy. 2010. A sketch interface to empower novices to create 3D animations. *Computer Animation and Virtual Worlds* 21, 3-4 (2010), 423–432.

Jongmin Kim, Yeongho Seol, and Jehee Lee. 2012. Realtime Performance Animation Using Sparse 3D Motion Sensors. In *Motion in Games.* 31–42.

Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. 2000. Interactive Control for Physically-based Animation. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques.* 201–208.

Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. 2002. Interactive Control of Avatars Animated with Human Motion Data. In *Proc. of the 29th Annual Conference on Computer Graphics and Interactive Techniques.* 491–500.

Huajun Liu, Xiaolin Wei, Jinxiang Chai, Inwoo Ha, and Taehyun Rhee. 2011. Realtime Human Motion Control with a Small Number of Inertial Sensors. In *Symposium on Interactive 3D Graphics and Games.* 133–140.

Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. 2010. Sampling-based Contact-rich Motion Control. *ACM Trans. Graph.* 29, 4 (2010), 128:1–128:10.

Noah Lockwood and Karan Singh. 2012. Finger Walking: Motion Editing with Contact-based Hand Performance. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 43–52.

Noah Lockwood and Karan Singh. 2016. Gestural Motion Editing Using Mobile Devices. In *Proceedings of the 9th International Conference on Motion in Games.* 25–30.

Jianyuan Min and Jinxiang Chai. 2012. Motion Graphs++: A Compact Generative Model for Semantic Motion Analysis and Synthesis. *ACM Trans. Graph.* 31, 6 (2012), 153:1–153:12.

Jianyuan Min, Yen-Lin Chen, and Jinxiang Chai. 2009. Interactive Generation of Human Animation with Deformable Motion Models. *ACM Trans. Graph.* 29, 1 (2009), 9:1–9:12.

Michael Neff, Irene Albrecht, and Hans-Peter Seidel. 2007. Layered Performance Animation with Correlation Maps. *Comput. Graph. Forum* 26 (2007), 675–684.

Sageev Oore, Demetri Terzopoulos, and Geoffrey Hinton. 2002. 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.* 133–140.

T. Pascu, M. White, and Z. Patoli. 2013. Motion capture and activity tracking using smartphone-driven body sensor networks. In *Third International Conference on Innovative Computing Technology (INTECH 2013).* 456–462.

Helge Rhodin, James Tompkin, Kwang In Kim, Edilson de Aguiar, Hanspeter Pfister, Hans-Peter Seidel, and Christian Theobalt. 2015. Generalizing Wave Gestures from Sparse Examples for Real-time Character Control. *ACM Trans. Graph.* 34, 6 (2015), 181:1–181:12.

Takaaki Shiratori and Jessica K. Hodgins. 2008. Accelerometer-based User Interfaces for the Control of a Physically Simulated Character. In *ACM SIGGRAPH Asia 2008 Papers.* 123:1–123:9.

Jaewon Song, Roger Blanco i Ribera, Kyungmin Cho, Mi You, J. P. Lewis, Byungkuk Choi, and Junyong Noh. 2017. Sparse Rig Parameter Optimization for Character Animation. *Comput. Graph. Forum* 36, 2 (2017), 85–94.

Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. 2011. Motion Reconstruction Using Sparse Accelerometer Data. *ACM Trans. Graph.* 30, 3 (2011), 18:1–18:12.

Matthew Thorne, David Burke, and Michiel van de Panne. 2004. Motion Doodles: An Interface for Sketching Character Motion. In *ACM SIGGRAPH 2004 Papers.* 424–431.

Vinayak, Devarajan Ramanujan, Cecil Piya, and Karthik Ramani. 2016. 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.* 12–20.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. In *ACM SIGGRAPH 2007 Papers.*

V. Zordan, D. Brown, A. Macchietto, and K. Yin. 2014. Control of Rotational Dynamics for Ground and Aerial Behavior. *IEEE Transactions on Visualization and Computer Graphics* 20, 10 (2014), 1356–1366.