Tangible and Modular Input Device for Character Articulation

Alec Jacobson[1] Daniele Panozzo[1] Oliver Glauser[1] Cédric Pradalier[2] Otmar Hilliges[1] Olga Sorkine-Hornung[1]

[1]ETH Zurich [2]GeorgiaTech Lorraine — CNRS UMI 2958

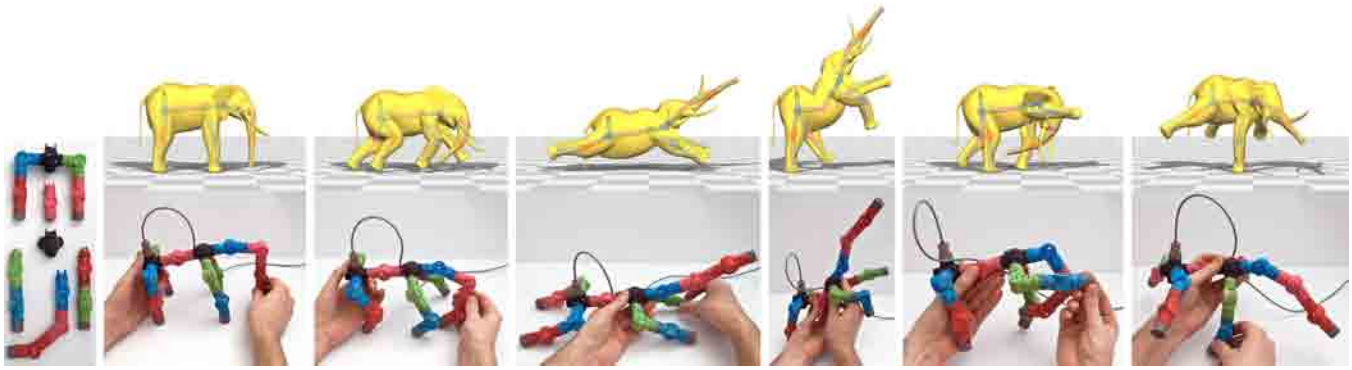


Figure 1: *Assembled from modular, interchangeable, and hot-pluggable parts (left), our novel device forms a skeletal tree matching the* Elephant*. As the user manipulates each joint of the device, measured bone rotations animate a skeletal rig, and the* Elephant *comes to life.*

Abstract

Articulation of 3D characters requires control over many degrees of freedom: a difficult task with standard 2D interfaces. We present a tangible input device composed of interchangeable, hot-pluggable parts. Embedded sensors measure the device's pose at rates suitable for real-time editing and animation. Splitter parts allow branching to accommodate any skeletal tree. During assembly, the device recognizes topological changes as individual parts or pre-assembled subtrees are plugged and unplugged. A novel semi-automatic registration approach helps the user quickly map the device's degrees of freedom to a virtual skeleton inside the character. User studies report favorable comparisons to mouse and keyboard interfaces for the tasks of target acquisition and pose replication. Our device provides input for character rigging and automatic weight computation, direct skeletal deformation, interaction with physical simulations, and handle-based variational geometric modeling.

CR Categories: I.3.1 [Computer Graphics]: Input devices

Keywords: tangible input, skeletal deformation, animation system

Links: DL PDF WEB VIDEO

1 Introduction

Interactively articulating virtual 3D characters lies at the heart of computer animation and geometric modeling. Sophisticated techniques express shape deformation in terms of a small set of degrees of freedom, most often the joint angles of an internal skeleton. Yet, even the ubiquitous skeletal deformation systems have too many parameters to be managed directly with traditional mouse and keyboard input. Animators must rely on indirect methods such as inverse kinematics or decompose complex and integrated motions into sequential manipulations of a small subset of the parameters—for example, iteratively positioning each bone of a skeleton hierarchy.

While direct manipulation mouse and touch interfaces are successful in 2D [Shneiderman 1997], 3D interactions with 2D input are ill-posed and thus more challenging. Virtual 3D widgets, e.g. Arcballs [Shoemake 1992], map 2D mouse input to 3D rotations, but interaction with the underlying 3D character is indirect and limited to a subset of the overall articulation parameters. Successful commercial products with 2D interfaces, e.g. Autodesk's MAYA, are notorious for steep learning curves and require interface-specific training.

Mouse and keyboard interfaces fall short because their control spaces do not match the perceptual space of the 3D interaction task [Jacob et al. 1994]. So instead, we propose direct physical manipulation via a *tangible interface* [Ishii and Ullmer 1997] with degrees of freedom in direct correspondence with the 3D rotations at skeletal joints of the virtual character. The device is composed of modular, hot-pluggable mechanical parts. The user may quickly assemble a custom combination of measurement joints and branching splitters to establish a one-to-one mapping between the physical device and virtual skeleton (see Figure 1). We leverage modern advances in 3D printing to produce compact, ready-to-assemble parts, comfortably held with one or two hands.

Exploiting the benefits of proprioception and physical affordances, an assembled device allows interaction with a physical manifestation of the virtual object, without the need for a literal, fixed replication. Each joint measures three rotation angles with accuracy of $\sim 1^\circ$ at a frequency up to 250 Hz. This makes the device not only suitable for rapid prototyping, but also precise control tasks such as meticulous keyframe posing and real-time animation capture. Complementary to the physical device, we introduce algorithms to facilitate the device's employment in the standard character rigging and animation pipelines. A novel semi-automatic registration algorithm accounts for the disparity between the device's physical proportions and those of the given virtual character. The user may then quickly match the rest pose of a character and immediately begin animating.

In traditional animation and posing, *absolute* positions and orientations are often specified separately or even by an external procedure (e.g. game physics engine). Thus, our device only measures *relative* bone orientations, allowing the user to move the device around on her desk to suit her comfort (analogous to the relative positioning of a typical desktop mouse).

Demonstrating our device's capabilities, we discuss results from a wide range of examples. Our device successfully provides input for character rigging and automatic weight computation, direct skeletal deformation, interaction with physical simulations, and handle-based variational geometric modeling. User studies—testing target acquisition and pose replication—report favorable findings comparing to traditional mouse and keyboard interfaces. Our device is no less accurate, but significantly more efficient.

2 Related work

Current commercial modeling and animation packages with mouse and keyboard interfaces favor control over ease of use and demand a high level of expertise. To alleviate this, many previous systems devise alternative user interfaces for 3D character articulation ranging from sketching (e.g. [Lin et al. 2012]) to full-blown motion capture (e.g. [Ishigaki et al. 2009]). We briefly review several areas of the related work that our system builds upon and extends.

Vision-based motion capture systems are among the most prevalent methods to create life-like animations of anthropomorphic, humanoid characters, and the field is widely studied (e.g. [Ishigaki et al. 2009]). While producing high-quality results, most systems require expensive equipment, large performance spaces, and often require body suits with markers. Requiring a human actor, they are not well-suited for the animation of non-humanoid characters. Nonetheless, some systems map human body motions to non-humanoid shapes, creating casual, but life-like animations [Seol et al. 2013; Chen et al. 2012]. This mapping introduces ambiguities, and performance capture still requires a large space and an actor.

Recent developments in hand-pose estimation assuage space requirements and bring motion capture to desktop sized spaces [Romero et al. 2010; Oikonomidis et al. 2012]. Wang & Popović track human hand poses to animate arbitrary shapes [2009]. Again, mapping ambiguities prevent more precise interactions and now an actor's hand is needed. Tracking a doll avoids the issue of needing an actor's body or hand [Feng et al. 2008; Shim 2010], but like all vision systems there are lingering issues concerning calibration, camera placement and lighting. Most relevantly, a hand-held input device is especially prone to occlusion problems (see Figure 2). As a consequence of these issues, state-of-the-art computer vision techniques rely heavily on a fixed skeletal topology as a prior [Shotton et al. 2013]. In contrast, our mechanical device is modular and automatically detects topology changes. Its internal sensors function accurately regardless of the environment, making the device not only suitable as a desktop tool at artist workstations but also as a performance instrument.

Figure 2: *Vision systems suffer from occlusion problems. Our mechanical device does not and is held comfortably with both hands.*

Low degree of freedom tangible UIs. Ishii & Ullmer introduce tangible user interfaces for manipulating virtual objects [1997]. A vast range of tangible UIs exist using a variety of methods to track physical objects, including computer vision, electrical tags and visual barcodes. Character animation prominently appears among many application areas. In [Johnson et al. 1999], a plush toy with embedded sensing recognizes discrete physical input events and triggers playback of pre-recorded animations. More recently, a vision-based 2D puppeteering system lets users perform 2D animations in real time using hand-drawn paper cut-out characters [Barnes et al. 2008]. Analogously in 3D, Held et al. animate 3D characters by tracking rigid physical objects' positions and orientations [2012]. These systems already demonstrate the expressiveness of tangible controllers for animation and storytelling. But unlike our approach, which allows fine-grained control over a character's many degrees of freedom, they are restricted to independent rigid transformations.

Nonetheless, tracking rigid objects independently in time may be used to animate deformable characters by *layering* animations [Oore et al. 2002; Dontcheva et al. 2003; Shiratori et al. 2013]. A user evaluation with expert animators shows benefits of layered animations over direct, full-body motion capture (in particular for non-humanoid characters) but also reveals the need for integrated motion control mechanisms (i.e. chains of joints) [Shiratori et al. 2013]. This is difficult to achieve with their physical controllers. Our system builds upon these ideas and due to its modularity allows for seamless transitions between layered animations of single joints, skeletal subtrees and the integrated animation of the entire skeleton.

Mechanical systems. A number of systems directly map a mechanical skeleton to an on-screen character. The mechanical control of non-humanoid, but custom fixed-topology, characters is perhaps originally introduced by the "Dinosaur Input Device", used to produce *Jurassic Park* [Knep et al. 1995]. The Monkey system of [Esposito et al. 1995] and later the Qumarion of [Celsys, Inc. 2013] optimize their designs for fixed humanoid topology at the cost of generality. The fixed humanoid-topology robot of [Yoshizaki et al. 2011] complements rotation sensors with actuating motors. In the context of keyframe posing, this convenient feature allows the device to reconfigure itself to previous poses. While certainly useful, actuation comes with serious drawbacks such as limited range of motion, slower response time and bulkier designs with limited or no reconfigurability. The fixed topology of the robot becomes particularly problematic when animating characters, as highlighted in [Yoshizaki et al. 2011]. The reconfigurable input device of [Weller et al. 2008] employs large, ball-and-socket joints. Though ball-and-sockets are intuitive in many scenarios, the achieved angular precision of 20° is insufficient for direct character animation. In contrast, our joints are compact and measure sub-degree accurate angles with high precision. Solving in some ways the inverse problem to ours, the Topobo device of [Raffle et al. 2004] is an assembly system that enables the construction of robots which record and replay motion sequences. Each piece is a separate unit that measures angles using servo motors. However, a piece does not know its relationship with others or its place in the global topology, thus it may not be readily used as an input device.

Skeleton fitting and deformation transfer. The way we help *register* the device within an input shape bears similarity to existing methods for skeleton extraction, registration or fitting. In geometry processing, techniques exist to robustly extract the medial axis or "curve skeleton" of a shape (e.g. [Tagliasacchi et al. 2012]). Though ostensibly similar to animation skeletons, curve skeletons are meant as a compact, abstract representation of the shape. Skeleton *fitting* methods for animation do exist, notably [Baran and Popović 2007]. The method fits a skeleton with given topology to an upright-oriented character, performing best on humanoids. Their expensive combina-

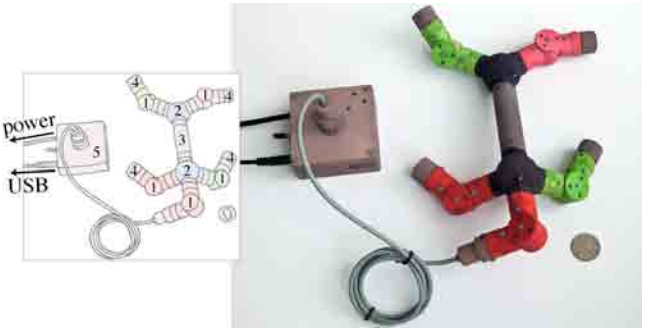


Figure 3: *A device instance may consist of joints (1), splitters (2), extension segments(3), endcaps (4), and a controller (5).*

torial optimization would not easily take advantage of the real-time angle measurements of our device. Our interactive approach exactly matches measured orientations. It may register to the entire shape or to parts of it, and it updates as the device topology changes. In computer vision, skeleton fitting has become part of the standard motion capture and pose recognition pipelines. Robust fits are found by querying databases of skeleton poses with feature keys based on depth images or tracked extremities [Shotton et al. 2013; Sridhar et al. 2013]. However, databases are generally populated with example poses of specific topologies tailored for common scenarios, e.g. upright humans or pointing hand gestures.

Our skeleton *matching* procedure maps measured angles to a preexisting animation rig's skeleton, whose proportions and orientations may differ. The more general topic problem of transferring deformations between arbitrarily different skeletons or shapes is well studied, e.g. [Sumner and Popović 2004; Baran et al. 2009; Bharaj et al. 2012]. We avoid such heavy-handed methods by utilizing our device's modularity: instead of mapping to an ill-fitting device, the user may simply reassemble a more appropriate one.

3 Hardware

Our goal is to design an easy to use input device for general-purpose 3D skeletal articulation. This goal breaks down into sub-criteria. To control a variety of characters with different skeletal topologies (alligators, centaurs, ostriches, etc.), the device must be modular and reconfigurable. To operate comfortably at a desk, the device should be compact in size and structurally sound to prevent accidental deformation. Finally, the device must measure 3D rotations with high precision and accuracy (see Figure 4).

A user constructs an instance of our device on the fly from modular parts or *nodes*. Figure 3 shows a typical configuration with a legend of node types. Mechanically moveable *joints* have embedded sensors measuring three intrinsic Euler angles. Static *splitters* allow branching in the skeletal tree. Static *extension segments* increase separation between joints and splitters. Passive *endcaps* cover exposed electronics and provide a comfortable manipulation handle. The entire skeletal tree connects to the host computer via the *controller* which transmits data, and powers the other nodes.

While the entire device is a custom design, it is nonetheless easy to reproduce. We provide a complete OpenHardware specification in supplemental material. We include CAD files for 3D printing and circuit schematics, both ready for outsourced fabrication.

Angular measurements at joints. Skeletal deformation is conducted with a kinematic tree of internal *rigid* bones. The skeleton's deformation is then parameterized by relative rotations orienting

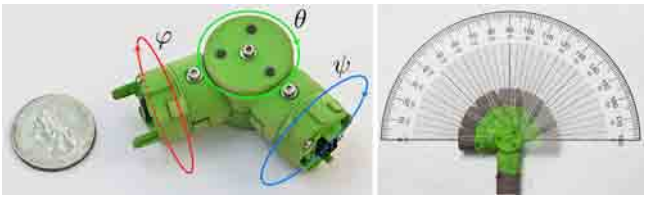


Figure 4: *Left: A joint has three rotational degrees of freedom parameterized by Euler angles: φ (twist), θ (bend), ψ (twist). Right: Bending range is just over 180° and comparing visually measured physical angles with the sensor's reveals an accuracy of $\sim 1^\circ$.*

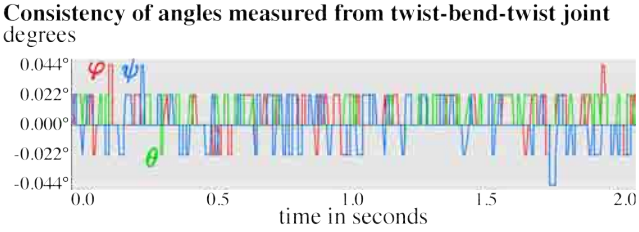


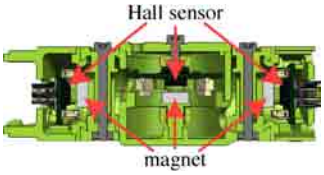
Figure 5: *An undisturbed joint left on a desk for two seconds measures variations in angle $< 0.1^\circ$ for all three sensors. The theoretical minimum jitter is 0.022° [Melexis Sys. 2013]. Observing these angles for 10 minutes shows maximum variation $\pm 0.066^\circ$ with a mean of 0.005° and standard deviation of 0.013°.*

each bone with respect to its parent. Though typically fixed during deformation, bone lengths vary across skeletons and models. To maximize modularity and facilitate design, our joints are a single fixed size, in general not matching the lengths of virtual skeletons' bones. We alleviate this disparity in software (see Section 4). To steer such a skeleton, our device reports relative rotations for each corresponding bone in the form of three intrinsic Euler angles. Our joints are composed of two twisting parts connected via a bending hinge (see Figure 4, left). The resulting relative rotation at the ith joint in the device is the composition of rotations by the twist angle φ, the bend angle θ and the final twist angle ψ:

$$\mathbf{R}_i = \mathbf{R}_z(\varphi)\mathbf{R}_x(\theta)\mathbf{R}_z(\psi), \tag{1}$$

where $\mathbf{R}_w(\alpha)$ rotates about the w-axis by angle α. We align the relative z-axis to lie along the joint when all angles read 0°.

The signed rotation angle of each part is measured by a corresponding *Hall sensor* and permanent magnet pair. Hall sensors are readily available in very small form-factors [Melexis Sys. 2013] and measure the orientation of our magnets' magnetic fields with an accuracy of $\sim 1^\circ$ (see Figure 4, right) and a precision of $< 0.1^\circ$ (see Figure 5). We place a small, flat magnet less than a millimeter away from each sensor. This ensures a very stable magnetic field not affected by external magnetic perturbations common in office environments. Each joint consumes 50mA at 5V. Powered solely by USB, our device could support up to 10 joints. With an external power-source (e.g. battery or wall adapter) more joints can be supported by the controller.



Before arriving at Hall sensors, we also evaluated two alternative sensor types. *Potentiometers* do not provide sufficient precision and suffer from a "dead-zone". A combination of a *3D accelerometer* and a *3D magnetometer* seems promising, since it frees mechanical design, but would require a locally stable magnetic field within

Figure 6: *Splitters need one-time calibration of outlet orientations (left). This is easy using a joint rotated so that outgoing frames match incoming frames (right), aided by embossed arrows (yellow).*

a range of 10cm. Our experiments show that the magnetic field diverges wildly inside an office environment, leading to angular errors above 40°. Our final Hall sensor and magnet pairing avoids this as the magnetic field is entirely dominated by the nearby magnet, not even detectably affected by the other magnets on the same joint.

Instead of our twist-bend-twist joints, one could imagine a ball-and-socket style joints akin to the human shoulder joint. The socket would need to simultaneously hold the joint securely in place and not limit the range of motion. The articulated 3D-printed characters of [Bächer et al. 2012; Calì et al. 2012] use ball-and-socket joints and cite friction and range of motion as lingering challenges, even without worrying about embedding sensor electronics. Accurate angle measurements also appear elusive. The LED and photosensors used for the large ball-and-socket joints in [Weller et al. 2008] measure rotations with orders of magnitude worse accuracy than our twist-bend-twist joints with Hall sensors. Our rotation parameterization suffers inherently from gimbal lock, but retains high accuracy, range of motion and easy friction control. Twisting angles span a range slightly less than 360°, the bending angle slightly more than 180°. Three accessible screws control the frictional stiffness of each joint.

Data marshalling. Each joint also contains a dedicated microcontroller. An assembled instance of our device can be understood as a reconfigurable sensor network. Each joint acquires angular data locally and communicates via a shared bus with the controller. Each component has a small amount of persistent memory and stores information such as its unique *ID*, node type, and color. Joints store three calibration offset angles. Splitters store additionally the number of connected children and relative orientation of its outlets, having been calibrated once (see Figure 6). Via the unique node *ID*, additional information may be associated to nodes and stored in software on the host computer (e.g. positional hints in Section 4). This memory facilitates topology detection and on screen visualization, but also allows a user to *resume* previously assembled devices. When a previous configuration is recognized, parameters may be restored so the user can immediately resume working with zero overhead.

Communication through a wired bus supports angle measurements at a frequency that is inversely proportional to the number of joints: from 250Hz with a single joint to 20Hz with 24 joints. All parts connect electronically using off-the-shelf male-female connectors with six pins: two for power, two for data and one for topology detection (and one unused). Inside each joint, components are connected with flexible wires, shielded in silicon to prevent breakage when bending or twisting. Mechanical linkage uses an asymmetric hook-and-lock to prevent faulty or incorrect connection.

Figure 7: *Our splitter design is general, supporting many different branching valences and outward orientations.*

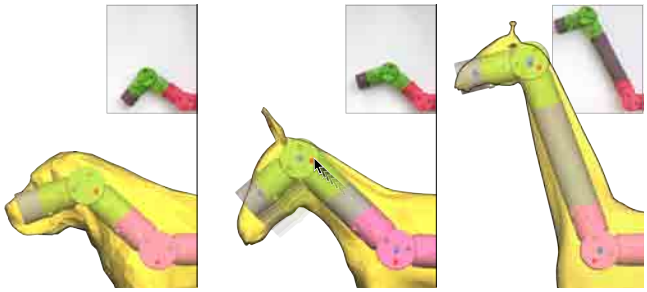


Figure 8: *Two joints roughly match the proportions of the* Mastiff*'s neck (left). We overcome differences in physical and virtual proportions in two ways: 1) stretching the space between the virtual joints as in the* Donkey*'s neck (middle) and for more extreme cases like the* Giraffe*, 2) inserting physical extension segments.*

We equip each node with controllable multi-color LEDs. By default, they indicate status: powered, initialized, detected within topology. In software, we replicate LED states in our visualization and activate LEDs to assist the user: for example, selecting a node on screen blinks a blue LED on the corresponding physical node.

Topology detection is achieved by passing messages between connected nodes. We use a simple distributed algorithm to visit nodes in depth-first order. Every joint begins initialization by listening to broadcasts on the shared bus. The controller emits a "topology pulse" to the first connected joint. Recursively, when a joint receives the pulse, it reserves a unique *ID* and broadcasts it to all other nodes. After receiving confirmation from the controller, the joint sends the topology pulse to its children. During the entire process, the controller forwards messages, containing node types and unique *IDs*, to the host computer, which then reconstructs the device on screen. The topology discovery is triggered every time a node is added or removed and takes approximately 100ms.

Static parts. The remaining parts in our device have no moveable pieces aside from small sliding connection locks. The branching of the virtual character's skeleton is in general arbitrary both in terms of valence and geometry. We propose the general concept of a radial splitter which supports as many outlets as will fit without overlap at any orientation. Ideally, we would manufacture a splitter with the same outlets at the same orientation as the given character. But this is not practically feasible if we wish to support arbitrary input shapes. Instead we settle on a small but sufficient set of common arrangements (see Figure 7). Extension segments intensify proprioception by helping overcome the physical-virtual size disparity. To keep electronics simple, refresh rates high and power consumption minimal, extension segments are unseen during topology detection. Finally, plastic endcaps are purely ergonomic.

4 Method

We now discuss algorithmic contributions that unleash our device's full potential within the character animation and geometry process-

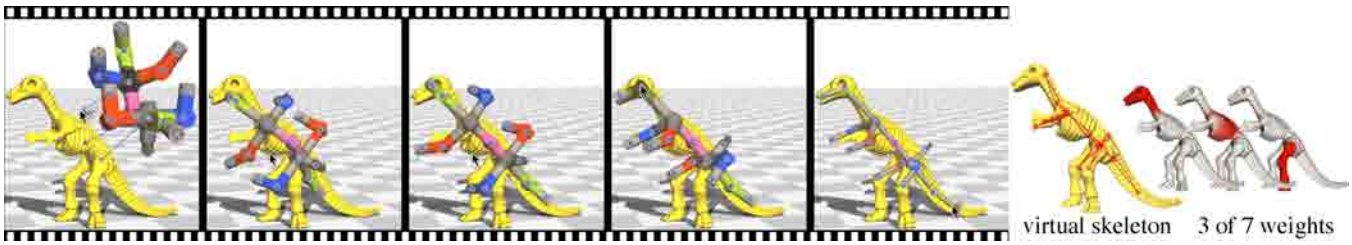


Figure 9: *Left to right: As the user adds joint position hints, the* virtual device *snaps into place within the un-rigged* Dino *model. This defines a virtual skeleton form which we may automatically compute skinning weights.*

ing pipelines. We support two common use cases. First, we consider the animation of characters without existing *rigs*, that is, just a triangle mesh without an associated control skeleton or skinning weights. With our assistance, the user may use our device to define such a skeleton and compute weights automatically. Second, we help the user *attach* the input device to characters with existing rigs, potentially manually created by a professional animator. In this case, we match the device's degrees of freedom to the rig's and appropriately adjust measured rotations while deforming to ensure intuitive control.

4.1 Rest pose registration

Standard skeletal deformation defines bone transformations relative to some *rest pose*. Thus, to control un-rigged characters we must first infer such a rest-pose skeleton from the current device configuration.

Our device's joint angles and topology fully determine a skeletal tree in space up to a global rigid transformation. However, the distances between joints of the physical skeleton will—in general—not match those of the character on screen (see Figure 8). Extension segments help alleviate this, but their proportions are also fixed.

Hence, we must *register* the device's current configuration to the character by finding appropriate lengths between each pair of neighboring nodes and a global rigid transformation. Thus creating a *virtual device* embedded in the on-screen character (Figure 9, left).

This could be achieved manually by dragging handles on the virtual device and restricting changes to agree with the current measured angles. As the tree hierarchy propagates changes downstream in the kinematic chain, this would require many iterations of adjustments. On the other hand, fully automatic fitting is ill-posed. A good fit requires heavy assumptions [Baran and Popović 2007] or semantic knowledge of the model and desired output animation.

We opt for a semi-automatic, variational approach. Given a sparse set of joint position hints, our optimization continually determines appropriate values for all bone lengths and the global rigid transformation, incorporating device input and new positional hints interactively. After registration, a skeleton is inferred and skinning weights computed automatically (e.g. with [Jacobson et al. 2011], see Figure 9, right), and the user can begin animating directly.

Assuming a topology is detected, let the current device configuration be represented by a list of non-root node positions $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ for $\mathbf{x}_i \in \mathbb{R}^3$ and a global rotation $\mathbf{Q}_0 \in SO(3)$ and translation $\mathbf{x}_0 \in \mathbb{R}^3$ associated with the root.

The position of each non-root node i may be written recursively using forward kinematics as

$$\mathbf{x}_i = \mathbf{x}_j + \mathbf{Q}_j \begin{pmatrix} 0 \\ 0 \\ s_i \end{pmatrix}, \tag{2}$$

with $\mathbf{Q}_j = \mathbf{R}_i \mathbf{Q}_k$ if k is the parent of j, and $s_i \in \mathbb{R}$ is the length of the edge between node i and its parent j. If we let $\mathbf{p}_i$ be the origin of the parent of a non-root node i, so that if node j is the parent of node i then $\mathbf{p}_i = \mathbf{x}_j$, then we may rewrite explicitly that

$$\mathbf{x}_i = s_i \mathbf{Q}_0 \hat{\mathbf{v}}_i + \mathbf{p}_i, \tag{3}$$

where $\hat{\mathbf{v}}_i$ is the directional unit-vector, determined — up to rotation by $\mathbf{Q}_0$ — via Equation (1) by the angles read from the device.

Energy formulation. The lengths $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$, global rotation $\mathbf{Q}_0$ and global translation $\mathbf{x}_0$ span node positions $\mathbf{X}$ exactly maintaining the relative orientations of the device. To find the *best* values, we solve the following non-linear optimization problem:

$$\underset{\mathbf{S},\mathbf{Q}_0,\mathbf{x}_0,\mathbf{X}}{\operatorname{argmin}} \quad w_{\text{user}} E_{\text{user}}(\mathbf{X}) + w_{\text{reg}} E_{\text{reg}}(\mathbf{S}) + w_{\text{drag}} E_{\text{drag}}(\mathbf{S}, \mathbf{x}_0, \mathbf{X}), \tag{4}$$

$$\text{subject to} \quad \mathbf{x}_i = s_i \mathbf{Q}_0 \hat{\mathbf{v}}_i + \mathbf{p}_i, \quad \forall i \in \{1, \ldots, n\}, \tag{5}$$

$$s_i > 0, \quad \forall i \in \{1, \ldots, n\}, \tag{6}$$

where the scalar weights w_{user}, w_{reg}, and w_{drag} balance energy terms. Though $\mathbf{X}$ is fully determined by the other variables via linear equality constraints (5), we simplify the description (and implementation) by treating all $\mathbf{S}$, $\mathbf{x}_0$, $\mathbf{Q}_0$ and $\mathbf{X}$ as variables. We delegate our solver to eliminate degrees of freedom and enforce hard constraints.

The user may specify a desired location $\mathbf{u}_i$ for any node i. Generally speaking, the current measured angles will prohibit exactly satisfying an arbitrary constellation of such constraints. Therefore, we attempt to achieve each location in a least-squares sense:

$$E_{\text{user}}(\mathbf{X}) = \sum_{i \text{ if } \exists \mathbf{u}_i} \|\mathbf{x}_i - \mathbf{u}_i\|^2. \tag{7}$$

With only a few user-specified hints, our problem is underdetermined, e.g. a single constraint only, or a straight chain with user constraints at either end. To ensure a reasonable skeleton, we punish unnaturally large differences between adjacent lengths:

$$E_{\text{reg}}(\mathbf{S}) = \sum_{\{e,f\} \in \mathcal{E}} \|s_e - s_f\|^2, \tag{8}$$

where $\mathcal{E}$ is the set of pairs $\{e, f\}$ where the edges incident on nodes e and f share a common node: that is, edges e and f are neighbors in the edge-dual graph of the skeletal tree. This energy term is also recognizable as a graph Laplacian regularization defined over the edge-dual graph.

The last energy term discourages numerical drift and instability in under-determined situations. This is important for realizing tempo-

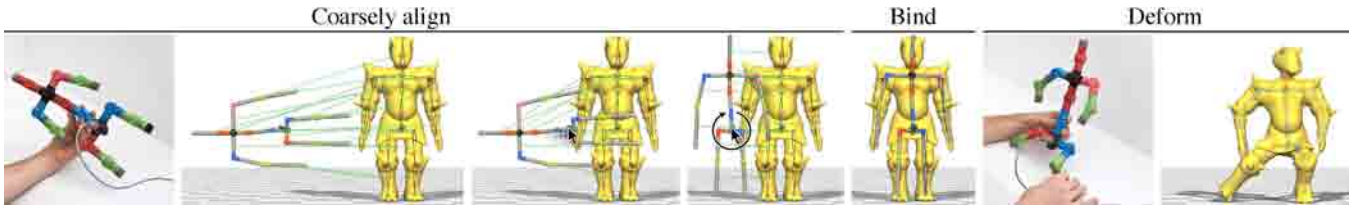


Figure 10: *We help the user match an assembled device to an existing rig of the* Knight. *Matching updates in real-time (dashed green) while the user positions the virtual device near the rig skeleton. Once close, we bind the device to the rig and the user starts animating immediately.*

rally smooth response. Therefore, we punish changes over time:

$$E_{\text{drag}}(\mathbf{x}_0, \mathbf{X}, \mathbf{S}) = E^{\mathbf{x}}_{\text{drag}}(\mathbf{x}_0, \mathbf{X}) + E^{s}_{\text{drag}}(\mathbf{S}) \quad (9)$$

$$= \sum_{i=0}^{n} \|\mathbf{x}_i - \bar{\mathbf{x}}_i\|^2 + \sum_{i=1}^{n} \|s_i - \bar{s}_i\|^2, \quad (10)$$

where $\bar{\mathbf{x}}_i$ and $\bar{s}_i$ are the position and incident edge length of node i from the previous frame respectively. Intuitively, this term amounts to introducing drag during the optimization. Our optimization frame rate is fast enough that no lag is noticeable, yet the term is significant enough to eliminate instabilities due to the problem being otherwise invariant to global rotation and scale in under-determined scenarios.

Finally, the constant lower bound constraints (6) ensure that lengths stay positive, otherwise edges between nodes could *flip* backwards.

Implementation. We solve our constrained optimization problem with a block coordinate descent (a.k.a. alternating optimization) in the style of [Sorkine and Alexa 2007]. If $\mathbf{Q}_0$ and $\mathbf{x}_0$ are fixed, then the problem reduces to a quadratic program (QP). We optimize for $\mathbf{S}$ and $\mathbf{X}$ using an open source active set QP solver [Jacobson et al. 2013]. If $\mathbf{S}$ and $\mathbf{X}$ are fixed, then the only non-constant energy terms are E_{user} and $E^{\mathbf{x}}_{\text{drag}}$. Minimizing these with respect to the rigid transformation represented by $\mathbf{Q}_0$ and $\mathbf{x}_0$ is a variant of the classic shape matching problem solved by a 3×3 singular value decomposition [Kabsch 1976].

As both steps do not increase our energy we may alternate until convergence. In practice, convergence takes a few iterations with a warm start and a single iteration (small dense QP solve and 3×3 SVD) takes less than one microsecond, so we simply apply a fixed number of iterations per frame (conservatively 100).

Our interface allows adding new positional constraints on the fly by dragging out from a virtual node's center with the mouse. To further facilitate user-interaction, the target position $\mathbf{u}_i$ is the *unprojection* of the mouse's coordinates at a depth between the first two hits on the model along the viewing ray. This simple feature greatly reduces the number of necessary viewpoint changes, allowing the user to focus on tuning the angles of the device.

Finally, we note that the weighting parameters w_* are unimportant in so far as w_{user} is comparatively large: our examples use $w_{\text{user}} = 1$, $w_{\text{reg}} = 0.001$, and $w_{\text{drag}} = 0.0001$.

4.2 Attaching to an existing rig

If we have only a 3D model as input, the previous algorithm helps embed the device's skeleton inside the shape. In other situations, a complete skeletal *rig*, consisting of a model, internal skeleton and weights may already be available.

The modularity of our device allows the user to construct a device that closely matches all or part of an existing rig's skeleton. To control such a rig, we need a mapping from the device's degrees

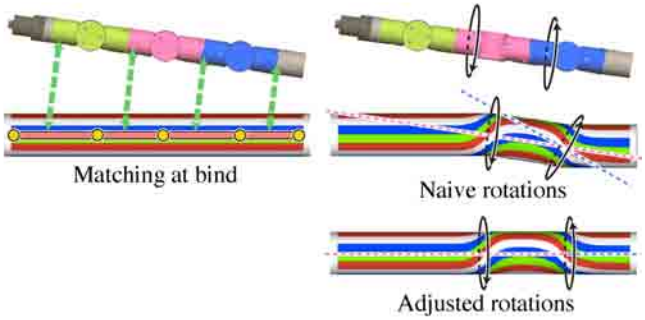


Figure 11: *We match to a skeletal rig without a precise spatial alignment (left). Propagating the device's relative rotations down the rig's kinematic chain produces unintuitive results (top right). Twisting the pink joint does not twist its matched bone; rather rotates about an axis parallel to the joint's initial orientation. Matters are worse further down the chain. Our adjustment fixes this (bottom).*

of freedom to the rig's. We take advantage of the fast rest-pose registration to help determine this mapping. First, the user guides the virtual device near the rig's skeleton. Meanwhile, we automatically determine a mapping between each rig bone and a device bone (see Figure 10). We cast this as a minimal matching problem over the complete bipartite graph between the sets of bones in the rig skeleton and those of the device. Each graph edge is given a cost c_{ij} set to the Hausdorff distance between the line segment of bone i of the device and the line segment of bone j in the rig. The optimal matching is found via the Hungarian method [Kuhn 1955]. Once combinatorially matched, the positional alignment may be iteratively refined with the rest-pose registration procedure, this time replacing user constraints with the locations of matched joint locations in the rig.

If rig bones and the virtual device perfectly overlap in space, then it would be sufficient to deform the rig directly using bone transformations determined by the device's kinematic chain. To support both exact and casual interactions, we assume that the bones are closely but not perfectly aligned. If bone lengths and joint positions differ, then directly using transformations from the device's kinematic chain will effectively tear rig bone's apart at joints and rotate about the device's virtual joint locations rather than the rig joints. Slightly better would be to propagate bone rotations measured on the device down the rig's kinematic chain. This maintains centers of rotation at rig joints, but the three Euler angles read from the device's joints are relative to the reference frames of the device's bones rather than the respective bones in the rig. For example, twisting a joint would spin the rig's bone about an axis parallel to the device's virtual bone axis rather than rig bone's axis (see Figure 11).

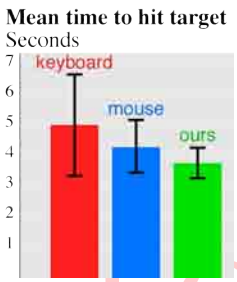
Instead, we precompute the minimal rotation $\mathbf{W}_{ij}$ that transforms the rest vector of the device's ith bone to that of its matched bone j in the rig. Using this rotation as a change of basis, we apply the measured rotation $\mathbf{R}_i$ from bone i on the device to matched bone

j on the rig as $\mathbf{W}_{ij}\mathbf{R}_i\mathbf{W}_{ij}^{\mathsf{T}}$. This aligns the device's Euler angle parameterization of joint rotations to the rig bones' reference frames before applying forward kinematics. The same procedure can be used to *rebind* to a previous keyframe pose. This allows the user to start posing from any previous configuration, not just the rest pose.

5 User studies

It is difficult to measure performance in terms of 3D character articulation directly. In the end what makes for a good animation or pose is not easily captured in a single metric such as task completion time or joint angle error. However, we do know that mouse- and keyboard-based UIs remain the most prevalent in character articulation systems. Therefore, we design two experiments that allow us to compare the effect of our input device on user performance directly to that of mouse and keyboard based UIs.

Targeting with two degrees of freedom. As a baseline experiment we briefly compared keyboard, mouse and our device in a 2.5D target acquisition task. This is a useful, controlled comparison as it removes the complexity and idiosyncrasies of a full 3D task.



We detail our experimental design and analysis in Appedix B. In terms of task completion time, the keyboard is the slowest (mean = 4.99s, standard deviation = 0.3s), followed by the mouse (4.31s, 0.4s) and our device (3.77s, 0.235s). Inset shows means with 95% confidence intervals. These results show a clear advantage of our device over the keyboard and conservatively equal performance to the mouse. This is particularly interesting as the task has only two degrees of freedom and thus is well-suited for the mouse. This provides further evidence that there is a benefit to direct, physical control of 3D objects, as often theorized in the tangible UI literature.

3D articulation. Following our baseline results, we excluded the keyboard from further experiments and now directly compare the mouse and our device. We compared the effect of an assembled device against that of a mouse-based UI in a complex character posing task, requiring precise control of many degrees of freedom. While still reasonably controlled, this experiment simulates the reality of character articulation and therefore allows us to draw conclusions about the actual usability in similar applications.

We asked subjects to replicate a series of predefined poses. Subjects will attempt to register the yellow *Dino* (see Figure 12) on top of the target pose rendered in red. We compare our device against a mouse-based UI for skeletal deformation identical to MAYA's. We compare to a forward kinematics mouse interface rather than inverse kinematics in order to measure fine-grained full character posing, not just end-effector placement. Requiring additional parameters, inverse kinematics would also make the experiment less controlled. In both conditions, the *Dino* was rigged with the same skeleton containing seven articulated bones. The sequence of poses was identical per condition. The presentation of interface order was counterbalanced. Excluding the creative element of posing, this experiment is not a perfect simulation of the animation process, but it does measure some aspects of posing: spatial thinking, 3D manipulation and impact of input device on accuracy and task completion time.

For this experiment we recruited from our university 11 participants (6 male, 5 female), ages 23 to 38. All participants reported intermediate to no experience with modeling tools like MAYA. None had previous experience with our device. Learning from [Yoshizaki et al. 2011], we refrained from imposing a hard cutoff time, instead instructing subjects to decide when they are "close enough or no longer making efficient progress".

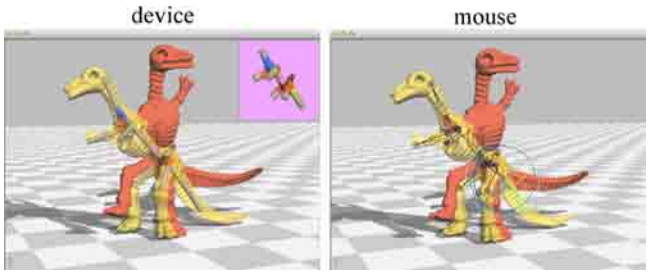


Figure 12: *On screen stimuli for 3D pose reproduction user study.*

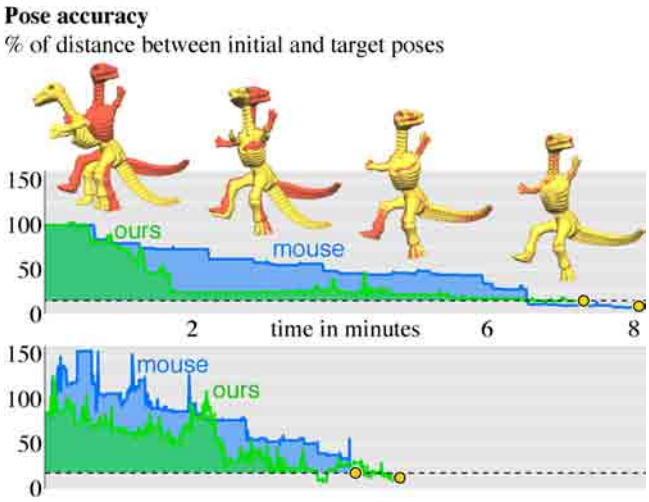


Figure 13: *Two typical posing sessions show the mouse (blue) and our device (green) decreasing pose distance from 100% to minimal values at completion (yellow dots). We take the maximum of the two minima as a baseline (dashed black) and compute integrals under both curves up to the time they reach this distance. This value is then normalized to define the amount of* work *necessary.*

To quantify performance we use three metrics: time to completion, accuracy and amount of work necessary to reach a "close enough" pose. The last, being an atypical metric, deserves some elaboration. When posing 3D characters, it is often important to quickly get a rough configuration, for example to communicate an idea to team members. To capture this crucial period in the creative process, we analyze our data in terms of *work*. In order to define a meaningful and fair measure we look at the integral distance for each interface up to the same small distance: the maximum of the minimal distances across the two interfaces. Then we normalize with respect to the time to reach this point in order to compare across users (see Figure 13).

Distance is measured as the sum of absolute angles of the smallest rotation aligning each bone's frame orientation in the target to the subject's pose. This is easily computed when representing orientations as unit quaternions. With the target and subject's frame orientation $\mathbf{q}_t$ and $\mathbf{q}_s$ respectively, our metric is

$$d(\mathbf{q}_t, \mathbf{q}_s) = 2\cos^{-1}(|\operatorname{Re}(\mathbf{q}_t\mathbf{q}_s^*)|) \tag{11}$$

where $d(\mathbf{q}_t, \mathbf{q}_s) \in [0, \pi]$. Distances are then normalized across poses and we report measures in terms of *percentage of distance*

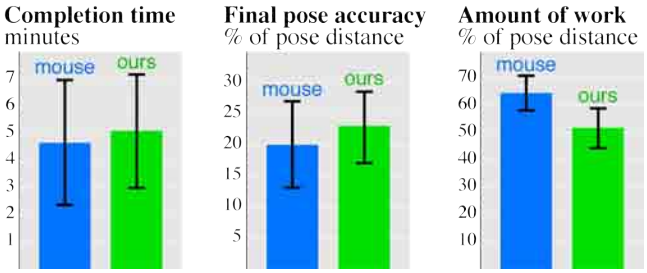


Figure 14: *Our 3D pose replication user study shows no significant difference between* MAYA*-clone mouse interface and our device in terms of speed and long-run accuracy. But our device is significantly better in terms of work needed to reach an acceptable distance.*

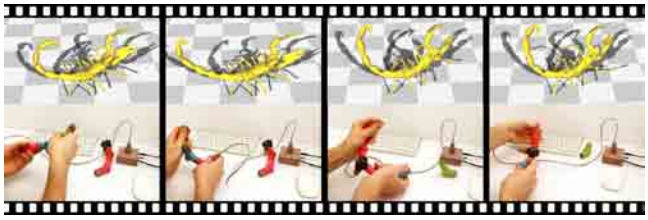


Figure 15: *A user may bind a small device to a subset of a rig and then layer animations for multiple subsets. The same parts—in different arrangements—first animate the body of the* Scorpion*, the tail, claws, and each pair of legs.*

between initial pose and target pose. For each pose, distance starts at 100% and decrease as the subject makes progress toward (but typically not perfectly reaching) 0%.

Of our results (summarized in Figure 14), the most immediate is the dramatic variability in total posing time across subjects (from 15 to 90 minutes). However, within subjects total posing times are similar. The mouse has (mean=4.66m, SD=3.4m) and for our device (mean=5.08m, SD=3.2m). A Student's t-test reveals that no significant difference between these means ($p = 0.768$).

Next we consider absolute distance to the target pose at task completion. Again we see no significant difference ($p = 0.465$) between mouse and our device. The the mouse has (mean=20%, SD=10.3) and our device (mean=23.0%, SD=8.2%).

Finally, regarding the amount of *work*, we see a significant difference ($p = 0.008$) between the two conditions. The *work* for the mouse has (mean=64.8%, SD=9.5%) and our device (mean=51.9%, SD=11.0%). Visualizing distances for each interface as a function of time illustrates this pattern (see Figure 13). Our device typically makes very fast initial progress, achieving a close enough pose and then slows down. The mouse by comparison makes steady but slow progress.

6 Applications

We manufactured a kit of 20 joints, nine splitters, four extension segments, 14 endcaps, and a controller (inset). We demonstrate the effectiveness of assembled device instances in different contexts and with different characters.

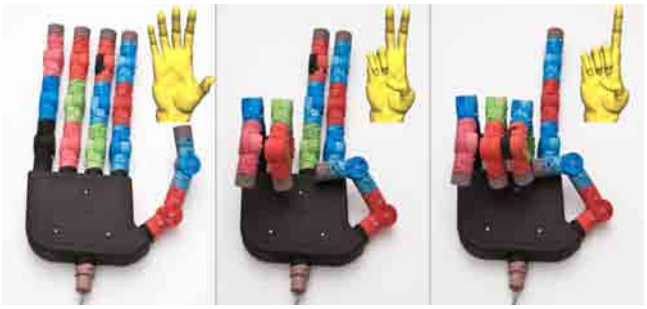


Figure 16: *Our design generalizes to customized splitters like this hand. The friction control of the 14 attached joints allows the device to remain in pose as the user poses the* Chimpanzee Hand*.*

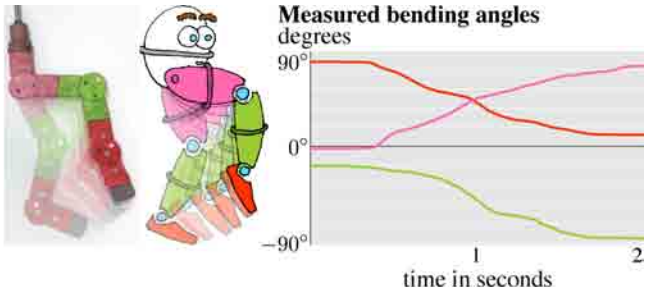


Figure 17: *A device with three bending joints (left, user's hands omitted) raises and points the leg of a 2D character (middle), animating all joints simultaneously (right).*

Figure 18: *Our device is also useful for non-skeletal deformation paradigms like handle-based variational modeling.*

The main application of our device is the interactive posing and animation of 3D characters. Figure 22 highlights some poses created with our device. These include the humanoid characters such as the *Knight* or the *Dino* and non-humanoid characters such as the *Crocodile*, *Ostrich*, *Frog* and the *Swedish Lamp*. It is noteworthy that some of these characters (*Scorpion* and *Swedish Lamp*) are rigged using external software making use of our skeleton *matching* procedure and the others use our rest-pose *registration*.

For very complex topologies it is often easier to layer animations recorded for different parts separately, rather than control all degrees of freedom simultaneously. Figure 15 shows the creation of a layered animation of the *Scorpion*, (re-)using only a small subset of parts. Mouse interfaces easily allow layering animations *up* the forward kinematics tree as changes will propagate down the tree. Exploiting human proprioception, our device takes advantage of the full hierarchy allowing layering both orders on the kinematic tree.

We purposefully design a general radial splitter that supports many branching cardinalities and orientations (see Figure 7). However, our firmware and algorithms support custom non-radial splitters for special purposes (see Appedix A). For example, we 3D printed

Figure 19: *A user interacts with an elastic simulation of the* Basset Hound *in real time using a single joint to control the dog's neck.*

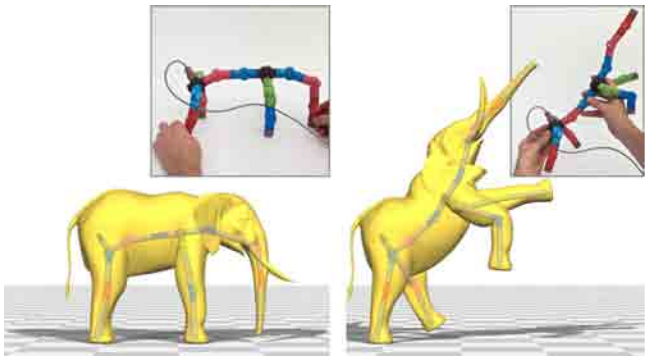


Figure 20: *If the user is unsatisfied with the limited control in the back of the* Elephant *in Figure 1, she may quickly insert a joint along the spine to increase flexibility.*

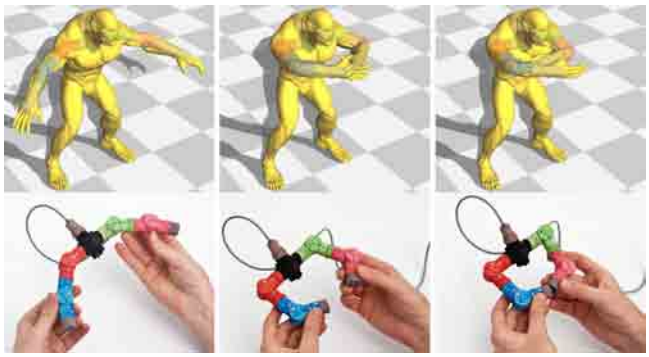


Figure 21: *Rigged with two joints controlling each arm (left), the* Beast*'s hands make contact* virtually *despite lack of contact in the device (middle). Similarly, making contact between end effectors of the device does not correspond to the same contact virtually.*

a hand shaped splitter used to create a fine-grained pose of the *Chimpanzee Hand* (see Figure 16).

Our device is not only suitable for 3D tasks, but by ignoring twist measurements also functions for 2D cartoon animation control. Figure 17 shows a user animating a one-legged 2D character. Our device allows for the simultaneous manipulation of multiple joints. Such simultaneous control is impossible with a mouse interface.

Figure 19 illustrates a particularly playful example. Here our device directs the *Dog*, who is governed by a dynamic elastic simulation, Our device may also provide input for non-skeletal deformation systems. A chain of joints manipulates the *Shark* in Figure 18, driving a handle-based deformation method [Sorkine and Alexa 2007]. Only the first and last joints are mapped to handles. The middle joint, while its measurements are ignored, acts implicitly via the kinematics chain, helping to manage distances.

The modularity of our device allows users to perfect and refine their working device to match the complexity of the desired articulation. In Figure 20, the user adds an additional joint to the device used in Figure 1. Our registration and binding algorithms reduce the effort required to make changes in the skeletal hierarchy and resume articulation.

Limitations and future work. In future work, we would like to optimize the design and performance of our device, arriving at smaller faster, and cheaper parts. Professional character rigs may contain hundreds of bones, spanning many levels in the skeletal hierarchy. Our device's size and power consumption become practical limitations when binding to such large rigs, though partial binding is possible at any level of a rig's hierarchy (see Figure 15). We plan to construct isolated twisting or bending joints, which would reduce the overall size in situations where certain degrees of freedom are semantically unnecessary (e.g. in Figures 16 and 17). Despite the aforementioned issues, we are working on a possible ball-and-socket joint which would overcome gimbal locking and allow tracing rotational geodesics.

Our device accurately reports *relative* orientations but has no sense of its absolute location or orientation. It would be interesting to combine our system with a rigid tracking system [Held et al. 2012]. Though our registration algorithm helps overcome physical-virtual disparities, certain semantic relationships do not carry over such as contact and collisions (see Figure 21). Perhaps haptic feedback could be useful to overcome this, though motors would encumber design and increase costs. Finally, our precise measurements would complement an augmented reality environment such as [Ando et al. 2002].

7 Conclusion

Tangible manipulation of arbitrary topology skeletons proves to be a powerful interface for posing, animation and modeling. Our complementary rest-pose registration and rig attachment algorithms lower the barrier of entry for 3D articulation tasks. The accuracy achieved by first-time users optimistically opens the door to building an expert skill set around tangible manipulation devices such as ours for character animation.

Our device is one step toward greater immersion and tangibility in the context of posing, designing and animating deformable 3D shapes. As displays make advances toward convincing autostereoscopy and 3D printing becomes more commonplace, we see potentially large impact from tangible input devices for virtual 3D content. To this end, we attach the complete hardware blueprints (OpenHardware) and accompanying source code in the hopes of fostering future research in this direction.

Acknowledgements

We are indebted to Ladislav Kavan for illuminating discussions and to Gilles Caprari (gctronic.com) for providing electronic and engineering support. We are grateful for the hours spent by our user study participants. We thank Marco Attene for making MESHFIX open source and Olga Diamanti, Romain Prévost, Christian Schüler, Kenshi Takayama and Emily Whiting for proofreading. This work was supported in part by the ERC grant iModel (StG-2012-306877), by an SNF award 200021_137879 and the Intel Doctoral Fellowship. The *Knight* was initially created using Cosmic Blobs® software developed by Dassault Systemes SolidWorks Corp.

References

ANDO, Y., TAKAHASHI, S., AND SHIBAYAMA, E. 2002. A 3D animation system with superimposing cg on a physical armature. *Proc. APCHI*.

BÄCHER, M., BICKEL, B., JAMES, D. L., AND PFISTER, H. 2012. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31*, 4.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3, 72:1–72:8.

BARAN, I., VLASIC, D., GRINSPUN, E., AND POPOVIĆ, J. 2009. Semantic deformation transfer. *ACM Trans. Graph. 28*, 3.

BARNES, C., JACOBS, D. E., SANDERS, J., GOLDMAN, D. B., RUSINKIEWICZ, S., FINKELSTEIN, A., AND AGRAWALA, M. 2008. Video puppetry: a performative interface for cutout animation. *ACM Trans. Graph. 27*, 5, 124.

BHARAJ, G., THORMÄHLEN, T., SEIDEL, H.-P., AND THEOBALT, C. 2012. Automatically rigging multi-component characters. *Comput. Graph. Forum 30*, 2.

CALÌ, J., CALIAN, D. A., AMATI, C., KLEINBERGER, R., STEED, A., KAUTZ, J., AND WEYRICH, T. 2012. 3d-printing of non-assembly, articulated models. *ACM Trans. Graph. 31*, 6.

CELSYS, INC., 2013. QUMARION. http://www.clip-studio.com.

CHEN, J., IZADI, S., AND FITZGIBBON, A. 2012. Kinetre: Animating the world with the human body. In *Proc. UIST*, ACM Press, New York, New York, USA, 435.

DONTCHEVA, M., YNGVE, G., AND POPOVIĆ, Z. 2003. Layered acting for character animation. *ACM Trans. Graph. 22* (July).

ESPOSITO, C., PALEY, W. B., AND ONG, J. 1995. Of mice and monkeys: a specialized input device for virtual body animation. In *Proc. I3D*.

FENG, T.-C., GUNAWARDANE, P., DAVIS, J., AND JIANG, B. 2008. Motion capture data retrieval using an artist's doll. In *Proc. ICPR*, 1–4.

HELD, R., GUPTA, A., CURLESS, B., AND AGRAWALA, M. 2012. 3d puppetry: A kinect-based interface for 3d animation. In *Proc. UIST*, ACM, New York, NY, USA, 423–434.

ISHIGAKI, S., WHITE, T., ZORDAN, V. B., AND LIU, C. K. 2009. Performance-based control interface for character animation. *ACM Trans. Graph. 28*, 3.

ISHII, H., AND ULLMER, B. 1997. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proc. CHI*.

JACOB, R. J. K., SIBERT, L. E., MCFARLANE, D. C., AND MULLEN, JR., M. P. 1994. Integrality and separability of input devices. *ACM Trans. Comput.-Hum. Interact. 1*, 1 (Mar.), 3–26.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4, 78:1–78:8.

JACOBSON, A., PANOZZO, D., ET AL., 2013. libigl: A simple C++ geometry processing library. http://igl.ethz.ch/projects/libigl/.

JOHNSON, M. P., WILSON, A., BLUMBERG, B., KLINE, C., AND BOBICK, A. 1999. Sympathetic interfaces. In *Proc. CHI*.

KABSCH, W. 1976. A solution of the best rotation to relate two sets of vectors. *Acta Crystallographica*, 32, 922.

KNEP, B., HAYES, C., SAYRE, R., AND WILLIAMS, T. 1995. Dinosaur input device. In *Proc. CHI*, 304–309.

KUHN, H. W. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2, 83–97.

LIN, J., IGARASHI, T., MITANI, J., LIAO, M., AND HE, Y. 2012. A sketching interface for sitting pose design in the virtual environment. *IEEE TVCG 18*, 11, 1979–1991.

MELEXIS SYS., 2013. MLX90316 DataSheet.

OIKONOMIDIS, I., KYRIAZIS, N., AND ARGYROS, A. A. 2012. Tracking the Articulated Motion of Two Strongly Interacting Hands. In *IEEE CVPR*.

OORE, S., TERZOPOULOS, D., AND HINTON, G. 2002. A desktop input device and interface for interactive 3D character animation. In *Proc. Graphics Interface*, 133–140.

RAFFLE, H. S., PARKES, A. J., AND ISHII, H. 2004. Topobo: a constructive assembly system with kinetic memory. In *Proc. CHI*.

ROMERO, J., KJELLSTROM, H., AND KRAGIC, D. 2010. Hands in action: real-time 3D reconstruction of hands in interaction with objects. In *IEEE ICRA*, 458–463.

SEOL, Y., O'SULLIVAN, C., AND LEE, J. 2013. Creature features: online motion puppetry for non-human characters. In *Proc. SCA*.

SHIM, B. 2010. Best student project prize talk: The wonder hospital. *SIGGRAPH Computer Animation Festival*.

SHIRATORI, T., MAHLER, M., TREZEVANT, W., AND HODGINS, J. K. 2013. Expressing animated performances through puppeteering. In *3DUI*, IEEE, 59–66.

SHNEIDERMAN, B. 1997. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *Proc. IUI*, 33–39.

SHOEMAKE, K. 1992. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proc. CGI*.

SHOTTON, J., SHARP, T., KIPMAN, A., FITZGIBBON, A., FINOCCHIO, M., BLAKE, A., COOK, M., AND MOORE, R. 2013. Real-time human pose recognition in parts from single depth images. *Commun. ACM 56*, 1.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proc. SGP*, 109–116.

SRIDHAR, S., OULASVIRTA, A., AND THEOBALT, C. 2013. Interactive markerless articulated hand motion tracking using rgb and depth data. In *Proc. ICCV*.

SUMNER, R., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3, 399–405.

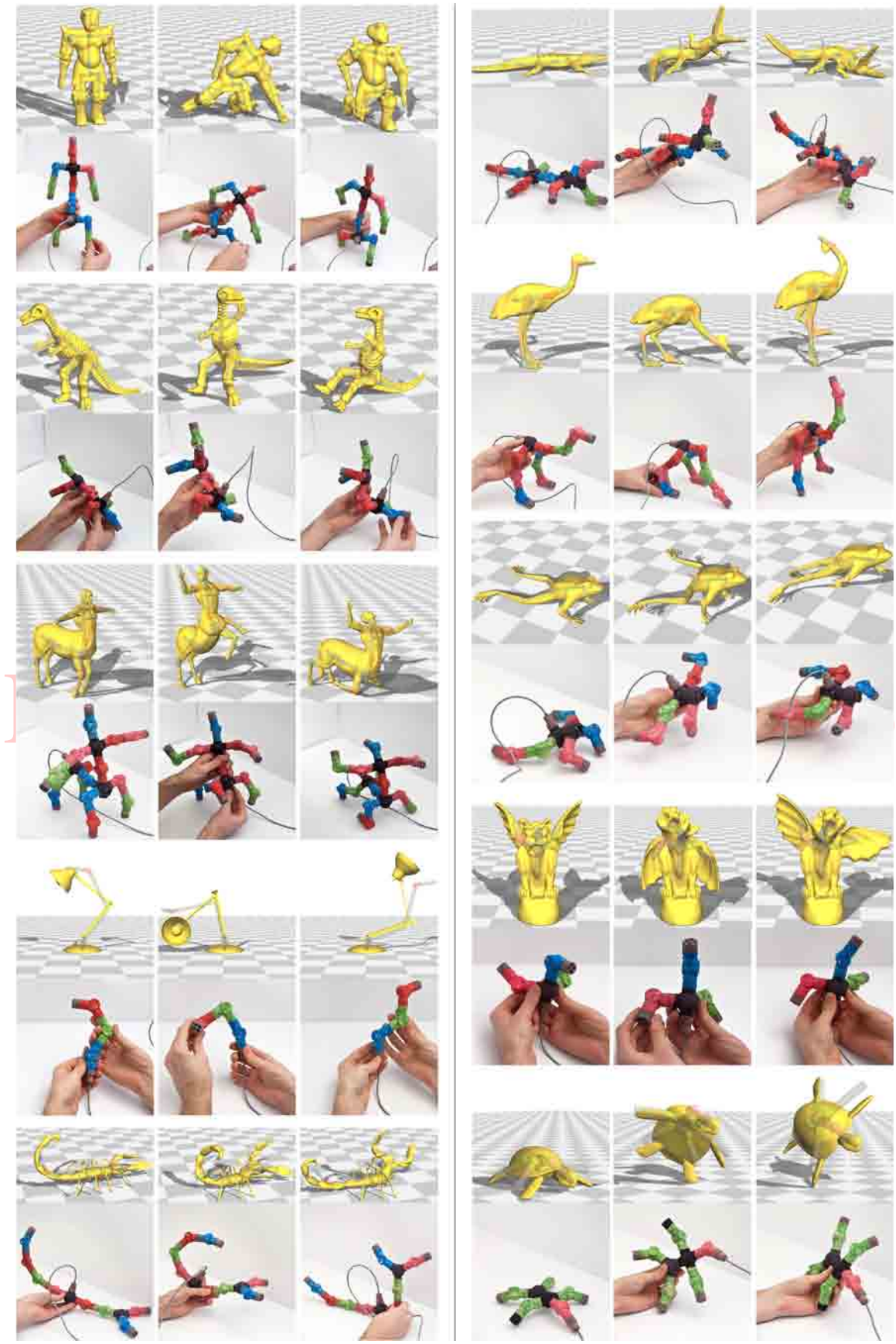


Figure 22: *We test our device on a wide range characters with skeletons of varying topological and geometric complexity. Each pair of rows is a virtual character and paired device. The leftmost column for each pair shows the device and character at their rest or bind states. The other two columns show poses.*

TAGLIASACCHI, A., ALHASHIM, I., OLSON, M., AND ZHANG, H. 2012. Mean curvature skeletons. *Comput. Graph. Forum 31*, 5.

WANG, R. Y., AND POPOVIĆ, J. 2009. Real-time hand-tracking with a color glove. *ACM Trans. Graph. 28*, 3, 63:1–63:8.

WELLER, M. P., DO, E. Y.-L., AND GROSS, M. D. 2008. Posey: instrumenting a poseable hub and strut construction toy. In *Proc. TEI*, 39–46.

YOSHIZAKI, W., SUGIURA, Y., CHIOU, A. C., HASHIMOTO, S., INAMI, M., IGARASHI, T., AKAZAWA, Y., KAWACHI, K., KAGAMI, S., AND MOCHIMARU, M. 2011. An actuated physical puppet as an input device for controlling a digital manikin. In *Proc. CHI*, 637–646.

A Non-radial splitters

Equation (5) does not apply to non-radial splitters (e.g. the hand splitter in Figure 16). For a non-radial splitter node i we introduce auxiliary origins $\mathbf{a}_{ij}$ at the outlets corresponding to each child j. By placing these origins accordingly, we may redefine $\mathbf{p}_j = \mathbf{a}_{ij}$ so that $\mathbf{x}_j = s_j\hat{\mathbf{v}}_i + \mathbf{a}_{ij}$. We can express each $\mathbf{a}_{ij}$ in terms of a scaling term and the original splitter origin $\mathbf{x}_i$:

$$\mathbf{a}_{ij} = t_{ij}\hat{\mathbf{u}}_{ij} + \mathbf{x}_i, \tag{12}$$

where t_{ij} and $\hat{\mathbf{u}}_{ij}$ are analogous to s_i and $\hat{\mathbf{v}}_i$. That is, $\hat{\mathbf{u}}_{ij}$ is fixed and indicates the direction of the offset origin and t_{ij} accounts for the stretch. Differently from before, we now must change all the scaling factors associated with the splitter uniformly. This can be modeled by additional linear equality constraints:

$$t_{ij}/\bar{t}_{ij} = t_{ik}/\bar{t}_{ik}, \quad \forall j \neq k, \tag{13}$$

where $\bar{t}_{ij}$ are the normalized (default) scales of each origin offset. Since the ratio t_{ij}/t_{ik} is fixed, each constraint is linear in in t_{ij} and t_{ik}. Similarly to the other constraints, they are not affected by $\mathbf{Q}_0$ and only show up during the QP solve.

B Design of targeting user study

We asked 18 subjects (14 male, 4 female) to aim the cannon of a virtual tank using mouse, keyboard and a single joint of our device (see inset). The subjects, again from our university.

For the keyboard, subjects control the cannon's pitch and yaw angles in small increments using the up, down, left and right keys. For the mouse, the absolute (x, y) coordinates of the hidden cursor map linearly to these angles, respectively. For our device, the first twist angle turns left and right and the bending part controls pitch of the cannon. Bullets fire using a foot pedal for all conditions. Repeated attempts are allowed until a hit, and a new target appears at a random location. Presentation order of the conditions is counterbalanced in a Latin-Square design. Each user is asked to hit a total of 50 targets per input device in blocks of 10, allowing for rest periods between blocks.

A two-way repeated measures ANOVA with the *interface* and the *block id* as independent variables reveals a main effect for blocks and a slight decrease in mean task completion time, indicating a mild learning effect. However, post-hoc analysis yielded no significant differences in speed between blocks (all $p > 0.05$). In terms of the *interface* used, the mean task completion time for keyboard was the slowest (mean= 4.99s, standard deviation = 0.3), followed by the mouse (4.31s, 0.4s) and our device (3.77s, 0.235s). A main effect for the interface on task completion time exists ($F_{(2,34)} = 22.8, p < 0.001$) and pairwise comparisons show significance between keyboard and mouse ($p = 0.002$) and between keyboard and device ($p < 0.001$). While our device scores slightly better than the mouse, the difference is not strictly significant ($p = 0.1$).