Velocity Skinning for Real-time Stylized Skeletal Animation

Damien Rohmer, Marco Tarini, Niranjan Kalyanasundaram, Faezeh Moshfeghifar, Marie-Paule Cani, Victor Zordan

Velocity Skinning for Real-time Stylized Skeletal Animation

Damien Rohmer[1], Marco Tarini[2], Niranjan Kalyanasundaram[3], Faezeh Moshfeghifar[4], Marie-Paule Cani[1], Victor Zordan[3]

[1] LIX, Ecole Polytechnique/CNRS, IP Paris, [2] University of Milan, [3] Clemson University, [4] University of Copenhagen

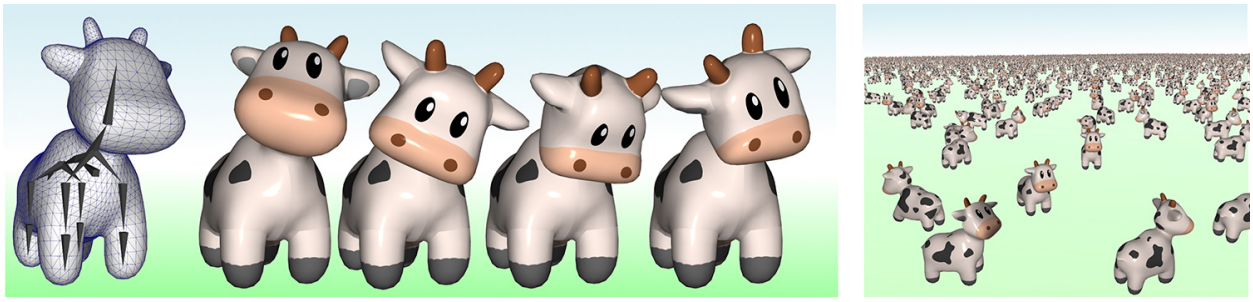


Figure 1: *Left: Skeletal rig, with a single bone in the head: When animated using velocity skinning, secondary animation effects are automatically added to the ear, and face, while the horn can be set as rigid. Right: The native efficiency and simplicity of the method is compatible with GPU implementation used to compute thousands of animated cows in real-time.*

Abstract

Secondary animation effects are essential for liveliness. We propose a simple, real-time solution for adding them on top of standard skinning, enabling artist-driven stylization of skeletal motion. Our method takes a standard skeleton animation as input, along with a skin mesh and rig weights. It then derives per-vertex deformations from the different linear and angular velocities along the skeletal hierarchy. We highlight two specific applications of this general framework, namely the cartoon-like "squashy" and "floppy" effects, achieved from specific combinations of velocity terms. As our results show, combining these effects enables to mimic, enhance and stylize physical-looking behaviours within a standard animation pipeline, for arbitrary skinned characters. Interactive on CPU, our method allows for GPU implementation, yielding real-time performances even on large meshes. Animator control is supported through a simple interface toolkit, enabling to refine the desired type and magnitude of deformation at relevant vertices by simply painting weights. The resulting rigged character automatically responds to new skeletal animation, without further input.

1. Introduction

Trained animators bring 2D characters to life through the inspired application of a few well-known principles, such as *squash, stretch* and *follow through* [TJ81, Las87, Wil01]. The latter are used to enhance the physicality of motion and make cartoon figures more expressive, with emotion and appeal. Unfortunately, the application of these rules to 3D animation is quite challenging. Indeed, the stylization and subtlety of the desired motion is at odds with the assumptions of existing pipelines [OZ10], such as linear blend or dual quaternion skinning techniques (LBS and DQS, respectively), where skin deformation is directly derived from the current skeletal pose. Further, the rules themselves require interpretation to be extended to 3D, which makes them difficult to formalize into generic algorithms [KL12, RM13, DBB*17] (among others, see Section 2).

We introduce a novel method to enhance 3D character animation pipelines with secondary motion control. Our method is easy to customize, handles the main animation principles, and supports fine-grain animator control.

While LBS and DQS derive skin deformation from the current, static configuration of skeletal joints, we claim that considering the additional influences of translational and rotational velocities along the skeletal structure is essential to bring more diverse and lively skin responses. *Velocity skinning* consists in adding this set of additional influences to a standard skinning method (LBS, in our implementation). This insight enables us to synthesize dynamic-looking, controllable skin deformations directly from the current state of skeletal motion. In particular, it enables us to achieve *squash & stretch* and *drag*-like exaggeration for *follow through*, among oth-

Author version accepted for publication at Eurographics 2021
Computer Graphics Forum 40(2), 2021.

ers (note that for sake of simplicity, we reuse these well-known terms from animation principles [TJ81], to denote the similar effects generated by our system).

Figure 1 illustrates this framework with a single animated bone, showing that it can bring various dynamic effects, such as the floppy ears and squashy face of the cow. While skilled animators could use key-frames to manually add such secondary motion, velocity skinning directly encodes them within the skinning pipeline. Thanks to GPU implementation, computed as a single-pass vertex shader, velocity skinning can be applied in real-time to any skeleton-based animated character, independently from its animation and in a reusable manner.

Our method uses, as input, a standard skinned mesh with a skeleton animation. It generates a deformation of the mesh, expressed as per-vertex displacements, that emphasizes (and stylizes) motion induced by the movement of the skeleton. We take advantage of the hierarchical nature of the skeleton to approximate dynamic deformations that otherwise, would not be accessible without either some manual input from a skilled animator or an expensive, physically-based simulation. The key idea is that the skeleton hierarchy embedded in the skinning computation provides enough information to decompose the motion of the mesh into meaningful, easily directable sub-motions that can be automatically associated with deformations. We contrast this approach to existing modeling tools, such as the deformers in a standard animation software package, which require manual setup and keyframing by a skilled artist for every animation, which is both time consuming and cumbersome.

Although the general velocity skinning framework supports a wide set of deformations, in this paper we showcase the utility of two specific characteristic deformations that we term "squashiness" and "floppiness". The magnitude of these two deformations effects can be finely tuned with scalar weighting parameters to reflect the different properties of the physical materials , and can be defined as per-vertex attributes "painted" on the mesh by an artist. Along with a small toolkit of interface handles, this allows simple control over an intuitive space of possible deformation behaviors, for example, to produce effects that appear to be made from heterogeneous materials.

The key contributions of this work are the following: we add a new term to the standard skinning formulation that creates customizeable spatial deformations based on velocity; we present a weighting scheme that derives consistent skeletal weights for the proposed framework from traditional (LBS) skinning weights; we split the translational and rotational velocity influences and show how they can be combined to create different effects, e.g. *squash & stretch* and (floppy) *follow through / drag*; and finally, we develop a set of deformation handles in support of making the effects controllable, based both on the desired output and the distinct characteristics of a given rigged skeleton.

We tested velocity skinning on both CPU and GPU implementations, to show its scalability to large meshes. A reference real-time web application is openly available at **https://velocityskinning.com**, and remains anonymous for reviewing purpose. The supplementary material associated with this submission contains the source code of the web application as well as the entire C++ and GPU interactive interface.

2. Related Work

Skin deformation of animated characters can be seen as mostly rigid, being driven by bones. Linear blend skinning (LBS) offers a simple solution for capturing this behaviour while enabling to smoothly blend deformations near joints [MTLT88]. Being fast and highly customizable, LBS is routinely used in the animation pipeline. Dual quaternion skinning [KCŽO08] is another popular choice, which solves some of the artifacts arising from LBS in case of high rotation angles, thanks to a non-linear blending mechanism that can be computed efficiently on the GPU. Encoding surface details with respect to a simpler deformed surface was also exploited to ease rig generation while improving visual results in general. For instance Delta Mush [MDR*14] uses Laplacian smoothing to generate such simple surface and encodes the details in a local reference frame to compute visually pleasing skinning very efficiently [LL19]. In contrast, implicit skinning [VBG*13] approximates skin as the iso-surface of an implicit field (the blend of fields associated with different bones) in which the mesh is embedded, each vertex storing its own iso-value to preserve details. The use of advanced blending operators enables not only to preserve volume at joints, but also to avoid inter-penetrations and achieve bulging skin in contact regions. This framework was extended to account for skin sliding effects [VGB*14]. Energy-based formulations associated with a set of positional, and possibly rotational constraints, were used to infer [JBK*12] and increase the range of possible deformations [WJBK15]. All these approaches do improve skin deformation in their own ways, but are not able to imitate dynamic behaviour, as our new method does. They could be combined with our work by serving as better input for the velocity-based deformer. In this paper, we rather derive velocity-based deformations from the standard LBS formulation, in order to remain fully compatible with standard production pipelines.

Skinning has also been subject to other improvements, related to an increase of the number of degrees of freedom, in order to fulfill specific criteria [JDKL14]. To mention only a few, this family of methods includes the interpolation in pose space [LCF00], the automatic insertion of extra bones [MG03], the addition of extra skinning weights [WP02, MMG06], the integration of limbs scaling [JS11], or local swing and twist deformers extracted from the blended bone transformations [KS12]. Allowing to deform the rest poses can also improve skinning. Such deformation can be automatically computed from a principal component analysis applied to a target model, such as a detailed finite element simulation for instance [KJP02]. Curved skeletons [YSZ06] were used to provide smoother skinning results and solve artifacts, and has been combined with extra deformers [FOKGM07].
These method however depart from the standard production pipeline. The latter [OZ10] is subject to strict constraints including computational efficiency as well as not easily accommodating changes affecting weights or the skeletal structure. Skeletons for which complex rigs may be scripted [NFB16], and possibly shared through multiple characters, should typically not be modified, as they are the core of animation assets. In addition, constant scalar

skinning weights painted by skilled artists are the de-facto standard of production-compatible skinning methods. Unfortunately, this pipeline maintains the animated skin as a purely passive geometry element, merely following the skeleton with no dynamic deformation effects.

In contrast, the principles of animation [TJ81], based on exaggeration and stylization of physical phenomena, have helped artists bring their characters to life for many decades. Since such "cartoon physics" is not supported by standard motion pipelines, professional animators often rely on additional deformers [May18] – for "squash", "jiggle", "bend", and so on. These act as extra, customized layers, designed to produce a wide variety of effects, but need to be tuned by adjusting their influence over mesh vertices from manual setting or procedural functions to be defined, as well as setting and attaching their possibly varying magnitude over the keyframes along the animation. Our method can be seen as an automatic parameterization of such deformers for cartoon like effects. It is readily available on rigged-animated model as it seamlessly makes use of existing skinning weights to set a model-aware influence along the mesh geometry, and rely on skeleton velocity to automatically adapt the magnitude of the deformation over time or keyframes. In addition, our approach still remain compatible with fine-grain artistic control from a per-bone control to magnitude weights painted at the per-vertex level if needed.

Computer graphic researchers have offered a variety of approaches related to "cartoon physics". Largely, the approaches allow the cartoon-like effects to be applied to the shape using additional inputs, for example, natural extensions of 2D effects can be achieved through the use of sketch-based interfaces guiding computer-generated deformation such as exaggeration [LGXS03], geometric constraints [NSACO05, RHC09], up to guiding an entire suggestive animation [KCGF14, KGUF16]. Example-based techniques have also been explored to model arbitrary predefined deformations [RM13, DBB*17, RPM] or rendering styles [BCK*13] that can be triggered during animation and transferred to a target shape [BLCD02, LYKL12]. These approaches provide a fine level of control and artistic expressiveness on the visual result, but they must be set up manually for each specific shape and animation. Regression based approaches were also used to generate secondary effects from simulations examples [dASTH10], or captured humans motions [PMRMB15], but were not used for exaggerated deformation.

Physically-based deformations, on the other hand, naturally handle automatic dynamic behaviours. Time integration as well as elastic energy formulation is widely used to improve skin deformation [DB13] as well as for avoiding self collision [CBC*05, MZSE11], and can be triggered by skeletal animation [CGC*02]. Efficient computation can be obtained from the use of subspaces modeling adapted deformation modes [JP02] and computed for instance in the rig space [HTCS13, WWB*19], using helper bone controller [MK16], or directly in pose space [XB16]. Position and projective based dynamics [MHTG05, BML*14, MMC16] are also popular physically-inspired methods, allowing real time deformation for moderately detailed shapes while being able to handle arbitrary non-linear constraints. They can successfully be used to animate dynamic characters [RF14, KB18] and even incorporate efficient voxels-based layers of bones, muscle and soft tissues [Nao15]. Interestingly, custom physically-based models were also developed to exaggerate specific cartoon-like effects [GDO07, CMT*12, BKLP16] on arbitrarily animated models. Zhang et al. [ZBLJ20] propose to enrich skinning animation with dynamic secondary effects. Their technique generates oscillation, follow through, and even collisions, that complement primary animations by computing these in the orthogonal subspace of the rig. In contrast, our system focuses on simpler effects aimed at an artist-driven workflow. While their technique requires a few seconds per frame for a mesh of a thousand elements, ours is interactive with much bigger models to support an artist's needs. We further note that velocity skinning does not aim solely at secondary effects as expressed by physical laws, rather the approach extends and exaggerates the motion in a flexible, general manner, especially integrating artist-directable parameters compatible with cartoon-like animation principles.

Simpler dynamic deformers have also been explored, from early work coupling particles and implicit surfaces to achieve cartoon-physics effects [OM94], through extra bones attached to the skeleton to model flesh oscillations [LCa05], or the use of sub-bones connected by springs to achieve curvy, dynamic shapes [KL08]. Muscles approximations have also been developed both in explicit [RL13] and implicit [RRC*18] formulations. Similar to our approach, kinodynamic skinning [AS07] proposes an integrated velocity-based formulation for skinning that can be tuned to exaggerate dynamic visual effects. While their deformation is expressed as a vector field to ensure fold-over-free trajectories for vertices, this requires costly numerical integration along streamlines which makes it computationally prohibitive.

In contrast, our method belongs to kinematic approaches, i.e. those that use position trajectory, or directly velocity and acceleration information, without requiring any time integration. In the specific case of predefined motion, time-based filter applied to the vertex position trajectory [WDAC06], or time-wrapper applied to the bone motion [KCJL06], were proposed to express exaggerated motion in space, as well as the notion of *follow through* and *anticipation* over time. The use of oscillating splines [KA08] aim at deforming the existing animation curves to mimic the trajectory of a damped mass-spring model. Associated to a phase shift along the surface geometry, the method can model drag effect followed by wiggling motion. The magnitude and phase field should however be parameterized by the user on every given model and do not automatically take advantage of the existing rig. Note that vertex trajectory filtering-based approaches can be seen as complementary of our method in caputing effects such as follow through effect that could be combined with ours.
Geometrically defined *squash & stretch* effects triggered by kinematics and collisions were also tackled by deforming the bones themselves [KL12], as well as slowing down trailing joints. While their approach allows local elongations along the bones, they only support global scaling with respect to the root-bone velocity. Therefore the space of deformations is more restricted than in our work which allows scaling and bending limbs along arbitrary directions computed from bone velocities. Closer to our work, Nobel et al. [NT06] developed a specific bending deformer able to curve limbs based on the direction of motion. Similar to our method, bone

velocity and geometric criterion are used as deformer parameters to bend limbs. However, the deformation for a given vertex is based solely on the attached bone, without considering the global hierarchy. Their velocity and deformation parameters are only computed with respect to a given bone and its parent, which lead to artifacts between joints. In contrast, our approach produces seamless deformations that remain coherent for arbitrary skeletal hierarchies.

Our method relies on the use of bone velocities to define closed-form, generic geometric deformers. As such, and similarly to static deformers, neither subsequent motion nor past configurations are required, which makes the method easier to add to a standard pipeline, at low cost, and greatly eases tuning. In addition, our deformations do not require any change to the input animation skeletons, while still being able to synthesize curvy skin shapes as well as mimicking time-delayed deformations. These smooth and coherent deformations are achieved thanks to a new formulation built on standard skeletons and skinning weights, as presented next.

3. Velocity skinning

We introduce our velocity skinning formulation by drawing from linear blend skinning (LBS). It could alternatively be directly plugged into another form of skinning such as DQS, as discussed in Section 6.

In LBS, at each frame, the position $\mathbf{p}^u$ of vertex u is computed as a weighted sum of the transforms associated to each bone applied to the rest pose $\mathbf{r}^u$ vertex as

$$\mathbf{p}^u = \left(\sum_i a_i^u \mathrm{T}_i \right) \mathbf{r}^u , \tag{1}$$

where T_i is the current frame's transform for bone i, obtained by accumulating rotations and translations along the hierarchical skeleton structure. Equivalently, $\mathbf{p}^u$ can also be understood as a linear combination of positions $\mathbf{p}_i^u = \mathrm{T}_i \mathbf{r}^u$ of the vertex u moved according to bone i as

$$\mathbf{p}^u = \sum_i a_i^u \mathbf{p}_i^u . \tag{2}$$

As the skeleton is animated by changing rotations (and sometimes translations) of bone joints at each frame, LBS moves vertex u to trace a trajectory over time.

For velocity skinning, we base the foundation of our approach on the premise that, like vertex position, vertex velocity $\mathbf{v}^u = \dot{\mathbf{p}}^u$ can also be understood as a linear combination of a set of *component velocities* $\mathbf{v}_i^u$. We exploit this by proposing to make component velocities induce separate, individually weighted displacements to the vertex, while allowing each velocity's influence to be customizable (to add distinct animation effects) through a function, ψ. The velocity-based position displacements are then added through linear combination to the final procedural mesh deformation, as displacement $\mathbf{d}^u$. That is,

$$\mathbf{d}^u = \sum_i b_i^u \psi_i(\mathbf{v}_i^u) , \tag{3}$$

where b_i^u are the bone weights defined per vertex. Deformer functions ψ_i take in account the geometry of bone i.

The LBS position $\mathbf{p}^u$ from Equation (1), which is determined by the *static* pose, is displaced by $\mathbf{d}^u$ from Equation (3), obtaining an additional mesh deformation which is automatically induced by the skeletal *animation*.

$$\mathbf{p}^{u\prime} = \mathbf{p}^u + \mathbf{d}^u . \tag{4}$$

To take advantage of the existing skinning weights from LBS, we show how we derive b_i^u from the existing skinning weights a_i^u in Section 3.1. To support the development of the desired velocity effects, we show in Section 3.2 a breakdown of velocity that separates the influences of translational and rotational elements. Finally, in Section 3.3, we illustrate a general framework for the function ψ, which can be employed to achieve different animation effects as well as combine the inputs of multiple effect *deformers*.

3.1. Velocity component weighting

While a general formulation for velocity skinning could introduce an arbitrary weighting scheme for b_i^u in combining the velocity influences in Equation (3), instead, we opt to take advantage of LBS by reusing the skinning weights already attached to the original mesh. To this end, we derive the velocity weights directly from the LBS weights and employ them in computing displacements.

Let us first consider the deformed position $\mathbf{p}_i^u$ defined by Equation (2), and decompose the velocity $\dot{\mathbf{p}}_i^u$ of vertices along the skeleton hierarchy. We call vector $\mathbf{v}_i^u$ the *component velocity* induced by the rigid motion of bone j relatively to its immediate parent. A straightforward decomposition along this kinematic chain shows that $\dot{\mathbf{p}}_i^u$ can be expressed as the sum over all these relative velocities such that

$$\dot{\mathbf{p}}_i^u = \sum_{j \in \mathcal{A}(i)} \mathbf{v}_j^u , \tag{5}$$

where $\mathcal{A}(i)$ denote the set of all ancestors of bone i (including i itself). Note that all velocity vectors are expressed here in the same global reference frame.

Differentiating Equation (2) with respect to time, and plugging Equation (5) into it leads to

$$\mathbf{v}^u = \sum_i a_i^u \left(\sum_{j \in \mathcal{A}(i)} \mathbf{v}_j^u \right) . \tag{6}$$

This can be rewritten as a single summation (see Appendix A for the full derivation):

$$\mathbf{v}^u = \sum_i \overleftarrow{a}_i^u \mathbf{v}_i^u \tag{7}$$

where the *upward-propagated* weight vector $\overleftarrow{a}_i^u$ (see Figure 2, middle) is defined at each vertex u as

$$\overleftarrow{a}_i^u = \sum_{j \in \mathcal{D}(i)} a_j^u \tag{8}$$

with $\mathcal{D}(i)$ denoting the set of descendants of bone i, i.e. the set of bones in the subtree rooted in bone i (including i). Equation (7) is the velocity counterpart of Equation (2). To create consistency in the skinning weights of LBS and velocity skinning we assign b_i^u to $\overleftarrow{a}_i^u$ in Equation (3). Note that in LBS, $\sum_j a_i^u = 1$ for every vertex u,

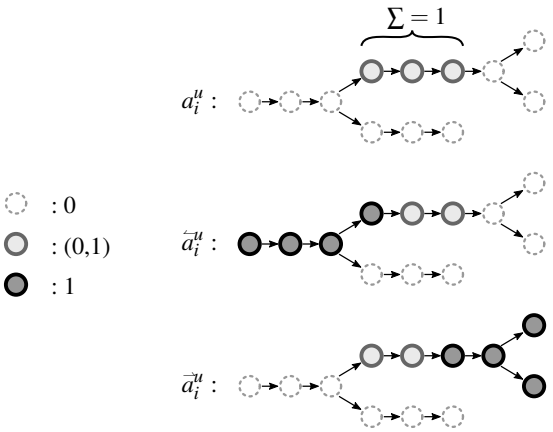


Figure 2: *An example of the upward-propagated weights (middle) and downward-propagated weights (bottom) derived for the original bone-to-vertex weights (top), for a given vertex u, as for Equations (8) and (18), respectively. For smooth skinning, each vertex is normally linked to a short sequence of interconnected bones (top). In velocity skinning, we use upward-propagated weights (middle) that can be derived from the original weights. Note, most ancestors have an influence of 1 on the given vertex. Bones i are represented as circles, and arrows represent the hierarchical structure of the skeleton.*

while this is not the case for the *upward-propagated* weights. Still each individual weight $\overleftarrow{a}_i^u$ belongs to the interval $[0,1]$.

3.2. Velocity component estimation

To support customization in the procedural use of the velocity deformers, we decompose velocity into its translational and rotational components. That is, for every vertex u, the velocity component associated to each bone i is given by

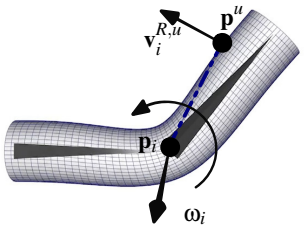
$$\mathbf{v}_i^u = \mathbf{v}_i^{R,u} + \mathbf{v}_i^{T,u}. \tag{9}$$

where $\mathbf{v}_i^R$ is the vertex velocity component induced by the *angular velocity* of the joint of bone i, and $\mathbf{v}_i^T$ is the vertex velocity component induced by the bone's *linear velocity* (when present).

To evaluate the terms of Equation (9), we must extract the angular ω_i and linear velocity $\mathbf{v}_i$ of each bone i. First, we extract them in the local space of the parent bone, then we propagate them upward in the skeleton hierarchy (using forward kinematics) to express them in the global reference frame. The actual computation of these velocities can either be carried through analytic derivation when the animation is provided as a set of parametric curves, or via finite differences between previous and current frames to easily support interactive deformation along *mouse motions*.

Vector $\mathbf{v}_i^{R,u}$ is then the linear velocity for vertex u in position $\mathbf{p}^u$ induced by the angular rotation ω_i around bone origin $\mathbf{p}_i$:

$$\mathbf{v}_i^{R,u} = \omega_i \times (\mathbf{p}^u - \mathbf{p}_i). \tag{10}$$



Vector $\mathbf{v}_i^{T,u}$ (the linear velocity of vertex u induced by the *translation* of bone i) is equal to the bone's translation $\mathbf{v}_i$, for all vertices. While in most character animations, translation is only allowed for the root, we do not make such assumption in our method, allowing translation, eg. when limbs lengths are animated.

3.3. Procedural velocity-driven deformations

In our method, we deform the mesh by computing a displacement for each vertex expressed as the weighed sum of displacements ψ_i (Equation (3)), that are function of the vertex velocity component associated with bone i. Now that we have selected our weights (Section 3.1) and isolated the contribution of each rotational and transitional bone animation (Section 3.2), let us detail the way we compute ψ_i. While the latter takes into account the geometry of bone i (e.g., the location of its origin $\mathbf{p}_i$), we will omit the index i in ψ below, for clarity in the exposition.

To increase the expressiveness of the method we opt for a version of ψ that affects the components of Equation (9) differently (rather than being applied to their sum), yielding

$$\mathbf{d}^u = \sum_i \overleftarrow{a}_i^u \, \psi(\mathbf{v}_i^{T,u}, \mathbf{v}_i^{R,u}). \tag{11}$$

Finally, as we want function ψ to combine a variety of effects, we set

$$\psi(\mathbf{v}_i^{T,u}, \mathbf{v}_i^{R,u}) = \sum_{\text{deform}} \psi_{\text{deform}}(\mathbf{v}_i^{T,u}, \mathbf{v}_i^{R,u}), \tag{12}$$

for a number of deformer functions ψ_{deform}. Thus, to compute $\mathbf{d}^u$, all that is left is to define the action of each deformer from their constituent velocity terms. Our squashy and floppy deformers showcase this process in the next section.

4. Squashy and Floppy Effects

While the velocity skinning framework is general and can be used for any deformation triggered by the motion of an articulated character, we illustrate it in this paper through its application to *squashy* and *floppy* expressive effects, motivated by the classic principles of animation [TJ81].

Both effects can be defined in terms of their translational and rotational input, ψ^T and ψ^R, respectively, through simple sums of the corresponding terms:

$$\begin{aligned} \psi_{\text{squash}}(\mathbf{v}_i^{T,u}, \mathbf{v}_i^{R,u}) &= \psi^T_{\text{squash}}(\mathbf{v}_i^{T,u}) + \psi^R_{\text{squash}}(\mathbf{v}_i^{R,u}) \\ \psi_{\text{floppy}}(\mathbf{v}_i^{T,u}, \mathbf{v}_i^{R,u}) &= \psi^T_{\text{floppy}}(\mathbf{v}_i^{T,u}) + \psi^R_{\text{floppy}}(\mathbf{v}_i^{R,u}) \end{aligned} \tag{13}$$

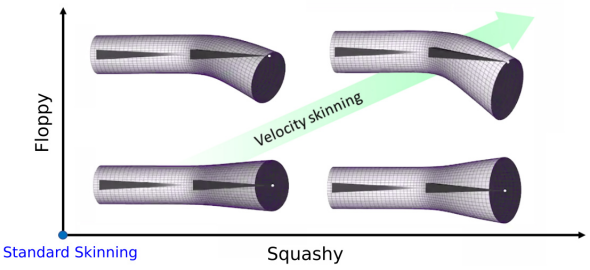


Figure 3: *The deformers of squashy and floppy velocity skinning reveal information in the still about the speed and direction of the motion that is not visible in the standard skinning surface.*

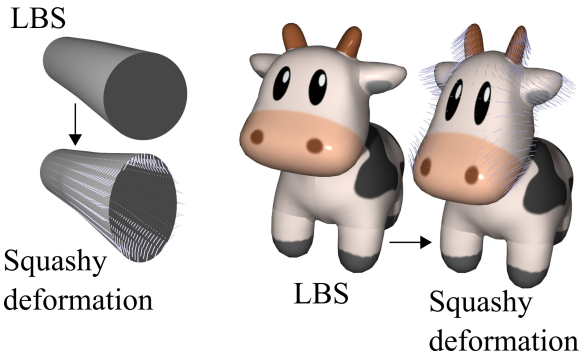


Figure 4: *The "squashy" deformer applies an anisotropic scaling to the moving parts of the shape compared to the shape obtained by LBS. Left: Result when the front-part of the cylinder rotates downward; Right: Result when the cow's head rotates downward.*

with these added to the set of the deformer functions of Equation (12). The respective scales of these effects can be tuned down (or nullified) using "floppiness" and "squashiness" gain values, k_{squash} and k_{floppy} respectively (see Figure 3). These values can be defined per-vertex and "painted" on the surface (see Section 5).

4.1. Squashy deformations

Inspired by one of the most well known of animation principles *squash & stretch*, our squash effect aims to deform an object to produce a local elongation in the direction of motion [TJ81]. At the same time, it is important that this deformation approximately preserves volume.

We define the squash effect through controlled scalings. Let the *centroid* $\mathbf{c}_i$ of bone i (the purple dot in Figure 8) be the center for the portion of the mesh that is affected by bone i. Because our system is not physically based, this is not a strict definition, and $\mathbf{c}_i$ can be freely customized by the user (see Section 5). By default, we position $\mathbf{c}_i$ at the barycenter of the vertices in the bone's region of influence (computed as described in Appendix B). Unless stated otherwise, we use this setting in all our examples.

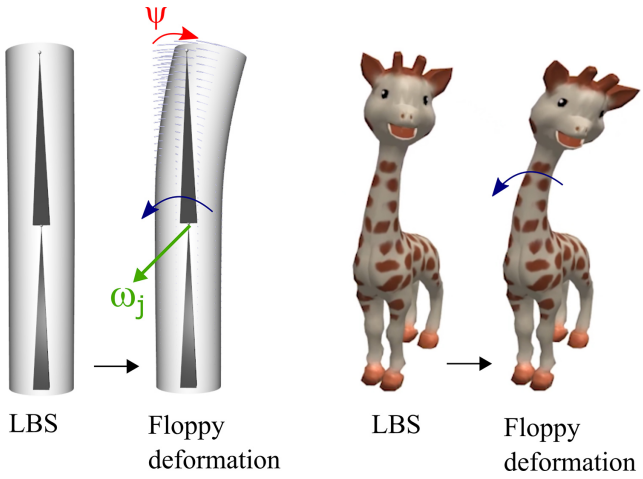


Figure 5: *Left: "Floppy" deformation induced by the counter-clockwise rotation of the top bone. In rest pose, the shape is a straight cylinder, while the deformer bends the cylinder in the opposite direction. Right: Same deformer when a neck bone of the giraffe rotates.*

For linear bone motions, the scaling is simply centered in $\mathbf{c}_i$. The formula for the displacement is given by

$$\psi^{T}_{\text{squash}}(\mathbf{v}_i^{T,u}) = (\mathrm{R}\,\mathrm{S}\,\mathrm{R}' - \mathrm{Id})(\mathbf{p}^u - \mathbf{c}_i)\,, \tag{14}$$

where Id is the identity matrix, R is any rotation matrix which maps the $x-$axis to the direction of the bone velocity $\mathbf{v}_i$ (and R′ denotes its transpose); S is the following anisotropic, volume-preserving scaling matrix:

$$\mathrm{S} = \begin{bmatrix} 1+s & 0 & 0 \\ 0 & 1/\sqrt{1+s} & 0 \\ 0 & 0 & 1/\sqrt{1+s} \end{bmatrix}$$

where the scaling value s is proportional to the speed of the velocity component:

$$s = k_{\text{squash}}\,\|\mathbf{v}_i^{T,u}\|$$

For rotating bone motions, the scaling resembles the effect of a centrifugal force generated by a spin. In this case, for each bone i, we identify a *medial axis*, as, intuitively, an axis expected to run approximately through the interior of the shape; as a default, we define it as the axis connecting $\mathbf{c}_i$ with $\mathbf{p}_i$ (magenta line in Figure 8), but again this choice can be tuned. Then, we want the squash effect not to produce any elongation or shrinking along this axis. Let Pr be the operator projecting a point into this medial axis. The formula for the displacement is then given by

$$\psi^{R}_{\text{squash}}(\mathbf{v}_i^{R,u}) = (\mathrm{R}\,\mathrm{S}\,\mathrm{R}' - \mathrm{Id})(\mathbf{p}^u - Pr(\mathbf{p}^u))\,, \tag{15}$$

where R is the rotation that maps the y-axis parallel to medial axis, and maps the z-axis as close as possible to ω_i When ω_i is parallel to the medial axis, then R is undefined, and ψ^{R}_{squash} is simply zeroed.

S is the following anisotropic, volume-preserving scaling matrix:

$$S = \begin{bmatrix} 1+s & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/(1+s) \end{bmatrix}$$

where the scaling value s is proportional to the magnitude of of the relevant velocity component:

$$s = k_{\text{squash}} \, \|\mathbf{v}_i^{R,u}\|$$

4.2. Floppy deformations

The floppy deformer is inspired by an exaggeration effect described by classic animators [TJ81] as "*The loose flesh [...] will move at a slower speed than the skeletal parts. This trailing behind in an action is sometimes called 'drag', and it gives a looseness and a solidity to the figure that is vital to the feeling of life.*" which is taken from a passage about their rule for Follow Through and Overlapping Action. We adopt the term "floppy" to describe this effect. Intuitively, this effect stems from the skin vertices appearing to move with a delay (or *drag*) with respects to the initiating bone movement; we obtain this by displacing vertices opposite to the velocity induced by each individual bone.

For linear bone motions, this is achieved by simply displacing the vertex in the opposite direction of the linear velocity induced by bone i:

$$\psi^T_{\text{floppy}}(\mathbf{v}_i^{T,u}) = -k_{\text{floppy}} \, \mathbf{v}_i^{T,u} \tag{16}$$

with the "floppiness" value, k_{floppy}, reflecting how pronounced the effect must be (this value can be defined per vertex, see Section 5).

For rotating bone motions, the delay produces a bending (rotation) around the current axis rotation of bone i (that is, the axis passing through p_i and aligned with ω_i), in the opposite direction:

$$\psi^R_{\text{floppy}}(\mathbf{v}_i^{R,u}) = (\mathrm{R} - \mathrm{Id})(\mathbf{p}^u - Pr(\mathbf{p}^u)) \tag{17}$$

where Pr is the operator projecting $\mathbf{p}^u$ on the rotation axis, and R is the rotation matrix around said axis with angle

$$\theta = -k_{\text{floppy}} \, ||\mathbf{v}_i^{R,u}|| \, .$$

Crucially, as $||\mathbf{v}_i^{R,u}||$, and thus θ, increase linearly with the distance to the bone (as per Equation (10)), the displacements induced by Equation (17) is not linear, resulting in naturally looking deformations that curve the profiles (as exemplified in Figure 5). In addition, the advantage of defining our deformation from the angular velocity of the joint, instead of using per-vertex velocities, is the ability to bend the shape, even by large angles, without introducing unwanted stretch (see Fig. 6).

5. Deformer Controls

In this section we introduce interactive mesh deformation tools which allow simple customization of parameters with simultaneous visualization of their effect on the character.

Per-vertex painting. k_{floppy} and k_{squash} weights (see Section 4) can be set non-uniformly over the surface in defining them as a per-vertex scalar value. These weights can be interactively painted over the surface using a readily available brush-like tool in standard

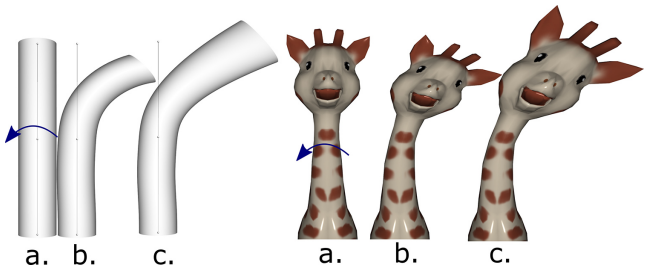


Figure 6: *For a tube (left) and the giraffe model (right), (a.) is the initial shape with an arrow indicating motion. The drag deformation using velocity-skinning (b.) adequately models bending, while the use of per-vertex velocities produces a stretching effect (c.).*

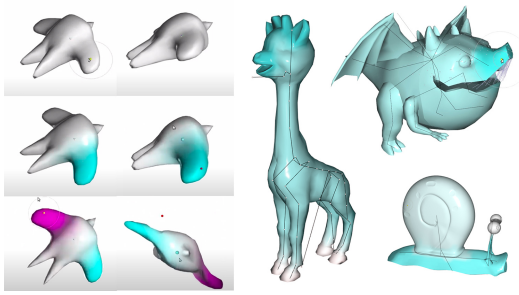


Figure 7: *Weight painting: left column, painting interface allows positive (cyan) and negative (magenta) weights to dictate the influence of the skinning effects. Left center column, representative image after weights are applied. Note top row equivalent to standard skinning with no effects on the outcome. Right, example paintings that show weights in practice on a three different models.*

modeling software. Non-uniform weights allow the user to express local effects that could not necessarily be captured by the animation skeleton. Auxiliary "phantom" bones (a.k.a. helper bones) are not required, we just need to set high weights values to make parts of the mesh look softer, or zero weight values to make them look rigid. Non-uniform weights can bring stylization effects even in the extreme case of a mesh associated with a single joint. This is illustrated by the bird in Figure 7-left, experiencing a rigid translation of its single joint, positioned at its center. The non-uniform floppy-deformation weights used for the wings (blue color) locally enhance motion. An extra degree of freedom can be used in associating negative values to weights – thus making deformation occur in *anticipation* of bone motion, instead of having skin parts *drag* behind. Lastly, the combination of positive and negative weights enables either synchronized motion or may provide a sense of action/reaction such as a birds flapping its two wings in one direction and in the opposite one (see bird wings in the accompanying video and in the interactive online demonstration).

Per-bone control. A coarse-level controller is added to manage which bones are affected by each effect. In this manner, entire branches can be removed (e.g. the legs in a walking motion) to assure the intent of the keyframe animation is not impacted negatively. Further, the bones that receive the effect can be controlled

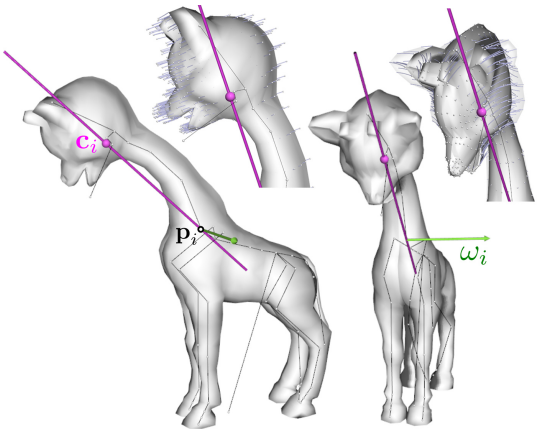


Figure 8: *Squashy deformer applied to the neck of a giraffe model. The magenta dot is the* centroid $\mathbf{c}_i$ *for the lowest neck bone, and the magenta line, its* medial axis*. These artist-tunable parameters are the centers for translation and rotation, for the squash effect.*

with finer granularity, by setting the weights of all bones, separately. This includes a split of the rotational and translational components of each. The artist supplemental video highlights more of the distinctions that can be made through this level of control.

Squashy-specific controls. We provide interactive control for the squash effect, in allowing the user to interactively move the centroid $\mathbf{c}_i$ associated with a bone (see Section 4.1 and Figure 8). This enables a fine tuning of the center or the axis around which the elongation and compression are applied for each bone, similar to the scale-pivot interface of standard 3D packages. The user-displacement applied to $\mathbf{c}_i$ is stored as an offset expressed in the local coordinate frame of the bone, and is therefore automatically adjusted during the animation. In addition, the user can force the squashy effect to act with respect to a single position instead of an axis. While the squash effect around an axis is well adapted for rotating limbs, the user may rather apply the effect of end-effector bones with respect to the local centroid in specific cases.

Floppy-specific controls. The floppy effect is similar to the surface-bending effect in some 3D modeling packages. However, ours is automatically scaled with the speed (to add the appearance of dynamics effects) and the floppy weight values (to give higher controllability). When high speed animation is used, this bending deformation may get exaggerated above plausible (desired) range. We integrate a simple interactive limiter defined as a maximal allowed bending angle for θ from Section 4.2, illustrated in Figure 9. This allows the user to model salient floppy effects for a character exhibiting both slow and high speed motions, while ensuring that high speed related deformations remains in the admissible range.

Deformation Visualizer. Interactive tuning of time-dependant deformation can be a complex task, while fine tuning is easier in static poses. The deformations we generate, however, fundamentally rely on velocity, although not directly on a time variable. Therefore, constant linear and angular velocity values can be artificially attached to bones, allowing a visualization of the induced deformation in a static pose/velocity space. The velocity (as well as the different parameters described previously) can then be interactively modified, while the deformation is updated accordingly. To ease the visualization of the deformation with respect to the input skin mesh, as well providing insight about the underlying motion and speed, our visualizer traces the trajectory that vertices would follow when the velocity magnitude of bones increase from 0 to the set values (see Figure 10). This tool offers an enhanced, visual understanding of the modeled deformation, independent from the final skeletal animation, and provides visual cues enabling tuning of velocity skinning effects from a static pose.

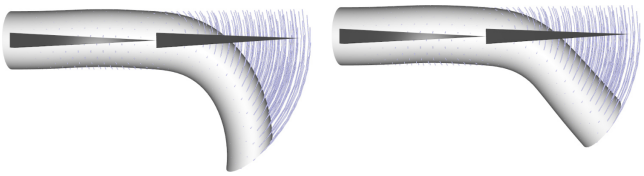


Figure 9: *Deformation of a straight cylinder, when the right-bone rotates counter-clockwise. Left: High speed or large floppy weights may lead to exaggerated bending deformation. Right: A maximal bending angle (in this case* 45°*) is set to bound the deformation.*

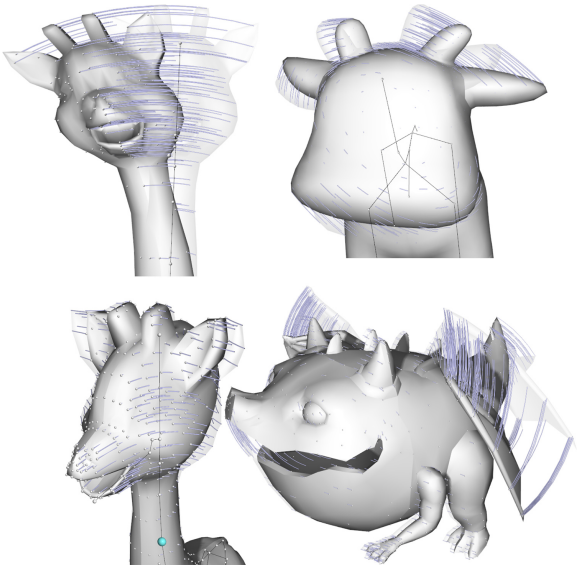


Figure 10: *Trajectory followed by skin vertices moving from standard LBS (depicted as a transparent surface), to velocity skinning deformation (opaque surface). Top: Floppy effect after bending the giraffe neck (left), and twisting the cow head (right). Bottom: Squashy effect after twisting the giraffe neck (left), and multiple floppy deformations applied on different parts of the dragon (right).*

6. Data flow of Velocity Skinning

Velocity skinning is well suited to standard real-time animation pipelines, since it follows the same setup both at pre-processing and at run-time stages. In particular, the independence between animation steps is well preserved, as all the data needed at run-time (in addition to skinning) can be stored separately for each skinned

model (independently from the animations), and for each skeletal animation (independently from the models).

Pre-processing a skinned model. At this stage and for each model, we propagate original bone weights along the skeleton structure to compute velocity weights using Equation (8), and store them per vertex. User-defined *drag* bounds and weightings (see Section 5) are also stored as vertex attributes of the skinned model. We also compute the centroid of each bone (Equation (18)), and store the manually tuned offset the user may add (see Section 5).

Pre-processing a skeletal animation. Given an animation (defined as a skeleton pose per keyframe) per-bone angular and linear velocities are extracted using derivatives of the joint trajectory curves as explained in Section 3.2. Our approach can accommodate arbitrary animation inputs such as parametric key-framed animations, procedural motions, as well as interactive run-time manipulations. To avoid discontinuity artifacts coming from sudden changes of position, these velocities can be smoothed-out by time averaging.

At run-time. After per-animation and per-model data are prepared or generated, final deformations are computed on-the-fly thanks to their closed-form formulation. In contrast to a physically-based approach, this fully kinematic procedural approach requires no book-keeping of previous states, resulting in a single-pass method, fully applied at run-time. In our experimentation, we applied velocity skinning on top of two standard skinning approaches, LBS and DQS, and it has a comparable memory footprint and workload to non-velocity skinning implementations.

7. Implementations and Results

For testing purposes, we implemented Velocity Skinning in three working prototypes, which were used to produce all the rendering in the figures in this paper and in the accompanying videos. The source code of all three prototypes is provided in the supplemental material.

The first implementation is a stand-alone, CPU-based desktop application, serving as a reference implementation for Velocity Skinning. It displays Velocity-Skinned animations, and doubles as an tool for authoring Velocity-Skinned meshes, by allowing an user to set the controlling parameters for a given model, and see the resulting deformation with with a given animations.

In order to explore the range of effect which can be achieved, and as a preliminary test of the authoring interface, we solicited the assistance of a set of animators: two novices (first-year digital-art graduate students) and two professionals (with 5-10 years experience in the commercial sector). The animations appear within the primary video along with the representative research results, but the process of adding effects is also highlighted in a supplemental video with side-by-side animations that showcase the input and output of our system. This video is narrated by one of the (first-year) animators describing the addition of effects in a “tutorial” style. The findings suggest our tools can add controllable effects to animations from both categories, novice and professional. Namely, the novice animations benefited from adding coarse effects (e.g. the whole head of the cow model) while the more refined professional animations only needed small, more subtle additions to push the existing animations further. An example of the latter would be using weight painting to isolate floppy effects to the ears and tail of the cow which do not include rig-bones in our stylized model (see Figure 1-left). Another example is illustrated in Figure 13 by a snail with different weights on its respective eyes, body and shell parts. In a post interview, the (professional) animator of the walk-cycle animation stated that it would take him an approximated three hours in Maya, based on his experience, to add the effects we introduced through our tools. It took a less-skilled animator less than an hour to produce the shown result. See the video for more details of this hands-on experimentation.

The second implementation is an interactive web-based demo that displays animations on a small set of character models, intended to showcase the effects that can be achieved with Velocity Skinned. It enables users to compare Velocity Skinning with standard blend skinning, and to test different pre-made set of parameters, which are customized for specific characters. The web-demo is based on WebGL, but computes the deformation on the CPU side. It can be accessed at https://velocityskinning.com.

The third implementation is a fully GPU-based desktop application, implementing Velocity Skinning in a vertex shader. It is intended to empirically evaluate the scalability of Velocity Skinning in a video-game or similar context. In this implementation, the different pre-processed parameters, including rig-weights and its propagated version as well as skeleton poses, are sent to the GPU memory as buffer objects, and accessed in the shader.

We tested this system on different scenarios: scenes composed by a single large model, with 300k, 500k and 1.5 millions of triangles (three resolution levels of a “Cthulhu” model [Gri18]) illustrated in Figure 11, and a scene with 4000 individually animated instances of a low-res "cow" models of 3200 vertices each (see Figure 1-right). To isolate the impact of the number of bones, we also compared the performances with the large model using only one animated bone. In each case, we compare against the case of Linear Blend Skinning. Table 1 reports total GPU memory usage, and render times measured on a consumer laptop in all scenes (GPU: NVIDIA Quadro P3200; CPU: Intel Core i7 2.6GhZ CPU).

This experiment indicates that velocity skinning can easily fit to low budget computation, allowing for real-time rendering even in presence of high-resolution mesh, and complex scenes.

8. Discussion and Conclusion

We introduced a new technique, velocity skinning, which enriches standard skinning with automatic stylization, without significantly impacting computational times. The main idea is to add a displacement at each vertex, computed from the velocities of the bones with non-zero local LBS weights. We demonstrated the usefulness of the method with two specialized deformers generating squashy and floppy deformations, designed in accordance with two classic animation principles. The resulting technique is finely controllable via a set of easily edited controls, such as “painting” the extent of the individual effects on the 3D model.

A key of the simplicity and thus efficiency of velocity skinning is the additive nature of the applied displacement (i.e. all the procedural displacements derived by individual bone motions, across

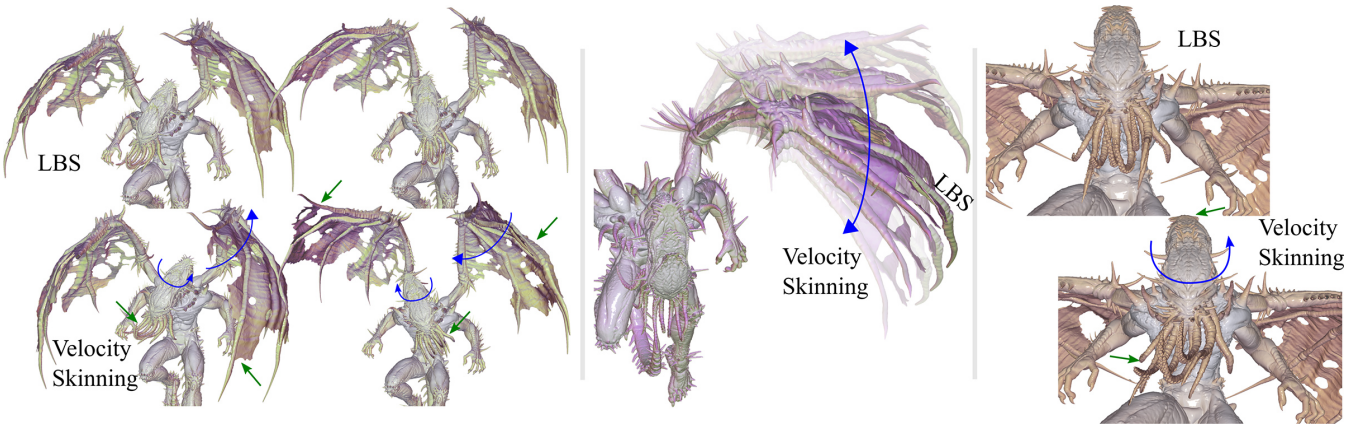


Figure 11: *Velocity skinning applied on a detailed mesh, thanks to our GPU implementation. The blue arrows indicates the main motions, while the green arrows highlight some noticeable effects of our deformers. Left: Comparison of LBS (top) and velocity skinning (bottom) on two extreme poses when constant floppy weights are used. Velocity skinning automatically enhances the motion in bending the wings and the tentacles based on their respective bones motion. Middle: Shape variations of the wing applied by the floppy deformer, the "neutral"-LBS pose being the middle one. Right: Computing automatic per-vertex weights based on local shape diameter allows to apply more deformation on mesh details, thus mimicking a lower stiffness on small protruding elements.*

Scene	Complexity (number of)				GPU memory		Computing time (per frame)	
	Models	per model vert	tri	bones	LBS	VS	LBS	VS
Flying Cthulhu	1	150k	300k	1	13 MB	21 MB	3.9 ms	8.0 ms
				12				8.7 ms
	1	250k	500k	1	22 MB	35 MB	6.6 ms	13.9 ms
				12				14.4 ms
	1	750k	1500k	1	64 MB	106 MB	18.7 ms	34.8 ms
				12				36.7 ms
Cow's meadow	4000	3.2k	5.8k	7	1.7 MB	2.4 MB	17.2 ms	39.0 ms

Table 1: *Timings and memory usage measured on different scenes, animated on GPU with standard Linear Blend Skinning ("LBS") and Velocity Skinning ("VS"). GPU memory includes the storage all precomputed parameters. Average timings are shown, but minimal and maximal stay in a range of ± 0.5ms. Bones indicates the actual number of bones that have a non-zero motion - thus generating a Velocity Skinning deformation on the mesh.*

all stylization effects, are simply added up). This is justified by the central observation that vertex velocities can be decomposed as a weighted sum of skeletal influences. However, because the deformation functions ψ are in general not linear with velocity, this is only an approximation which degrades as deformations grow. Thus, velocity skinning cannot be expected to always give good results, especially for extreme deformations or special conditions, for example when mixing large deformations induced by velocities with different directions. In spite of this, our experiments indicate that velocity skinning is able to produce expressive stylized animations under many useful scenarios, such as when the skinned mesh is animated by a rig with only a few joints (see Figure 12 or the web demonstration).

To be clear, our method is not physically based and therefore the animations we produce will not necessarily be realistic. Velocity skinning is not intended as a replacement for dynamic simulation, although it may look similar to it when well tuned. A benefit of our method compared to simulation is that any per-vertex displacement is applied instantaneously, and can be computed independently for each frame. We hypothesize that accounting for acceleration terms in addition to velocity will narrow the gap, and we plan to experiment with this in future work. Another future research direction would be to apply data-driven training to the models, in order to make the velocity skinning weights match a physical simulation. Visual appearance may degrade under various settings, especially when summing different contributions in opposite directions. For example, applying fast and opposite rotations on successive joints can result in undesirable scaling. In our experiments, such artifacts were only visible in carefully crafted and exaggerated examples such as the one shown in Fig 14. Otherwise, the animation remained surprisingly well-behaved, without visible artifacts, even

Figure 12: *This dragon has a minimal number of bones in its wings and tail, left column shows the rest post and skeleton. Through velocity skinning, a rich flapping and wagging animation (right, frames) can be added automatically to augment the limited skeleton animation. See accompanying video for more detail.*

when the deformation was quite large. An additional limitation of the current implementation is that motion applied to the end effectors of a character will not propagate backward within the hierarchy - this effect would require dynamic adaptation of the skeleton hierarchy. Therefore, our method cannot achieve contact-like effects and IK-guided motion (contrasting further with [ZBLJ20]).

From the *end-application* point of view, velocity skinning adds a only limited overhead on top of standard LBS: both are single-pass techniques that fit interactive applications as well as a GPU pipeline. The largest impact stems from the fact that weights $\tilde{a}_i^u$ are not as sparse as a_i^u, because they are propagated along the skeleton hierarchy. As such, velocity skinning cannot offer the same range of optimizations available for standard skinning, where the number of bone-links per vertex is commonly limited to a constant between 2 and 8. Still, the technique is amenable in practice for real-time resource critical settings, as indicated by our implementations. Further research could be developed to allow a fast, closed-form update of velocity-skinned normals for accurate illumination, as can be done with basic skinning [TPSH14].

For velocity skinning to be generally adopted, it also needs to be amenable to be included in existing *asset-creation pipelines*. As a first exploratory step, we made efforts to offer specialized interface handles for animation control. While we were able to solicit limited animator help in the production of some animations in the corresponding video, and we received very positive feedback, more extensive testing in the field would help identify bottlenecks and give stronger evidence for the utility and benefits of the approach. While we feel the introduction of the fundamental approach is a stand-alone contribution and we have made some effort to offer a practical implementation for animation control, extensive user testing is an important next step.

In summary, velocity skinning offers a new path to include automatic, stylized deformations and holds great potential for both offline and interactive future animation effects.

Acknowledgement

We acknowledge and thank Paul Kry, Kenny Erleben, and all attendees of the 2020 McGill University Bellairs workshop, the origin of this work, for the fruitful discussions and input. We would also like to thank the artists, Rodney Florencio Da Costa, Kayla Rutherford, and Mohammad Saffar that offered their help and feedback in our usability study.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 764644.
This paper only contains the author's views and the Research Executive Agency and the Commission are not responsible for any use that may be made of the information it contains.

References

[AS07] ANGELIDIS A., SINGH K.: Kinodynamic skinning using volume-preserving deformations. In *SCA* (2007). 3

[BCK*13] BÉNARD P., COLE F., KASS M., MORDATCH I., HEGARTY J., SENN M. S., FLEISCHER K., PESARE D., BREEDEN K.: Stylizing Animation By Example. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 32*, 4 (2013). 3

[BKLP16] BAI Y., KAUFMAN D. M., LIU K., POPOVIC J.: Artist-Directed Dynamics for 2D Animation. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 35*, 4 (2016). 3

[BLCD02] BREGLER C., LOEB L., CHUANG E., DESHPANDE H.: Turning to the Masters: Motion Capturing Cartoons. In *SIGGRAPH* (2002). 3

[BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 33*, 4 (2014). 3

[CBC*05] CAPELL S., BURKHART M., CURLESS B., DUCHAMP T., , POPOVIC Z.: Physically based rigging for deformable characters. In *SCA* (2005). 3

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIC Z.: Interactive Skeleton-Driven Dynamic Deformations. *ACM SIGGRAPH* (2002). 3

[CMT*12] COROS S., MARTIN S., THOMASZEWSKI B., SCHUMACHER C., SUMNER R., GROSS M.: Deformable Objects Alive! *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 31*, 4 (2012). 3

[dASTH10] DE AGUIAR E., SIGAL L., TREUILLE A., HODGINS J. K.: Stable Spaces for Real-time Clothing. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 29*, 4 (2010). 3

[DB13] DEUL C., BENDER J.: Physically-based character skinning. In *VRIPhys* (2013). 3

[DBB*17] DVOROZNAK M., BÉNARD P., BARLA P., WANG O., SYKORA D.: Example-Based Expressive Animation of 2D Rigid Bodies. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 36*, 4 (2017). 1, 3

[FOKGM07] FORSTMANN S., OHYA J., KROHN-GRIMBERGHE A., MCDOUGALL R.: Deformation Styles for Spline-based Skeletal Animation. In *SCA* (2007). 2

[GDO07] GARCIA M., DINGLIANA J., O'SULLIVAN C.: A Physically Based Deformation Model for Interactive Cartoon Animation. In *VRIPHYS* (2007). 3

[Gri18] GRIPPIN A.: *Flying Cthulhu*, 2018. URL: https://sketchfab.com. 9

[HTCS13] HAHN F., THOMASZEWSKI B., COROS S., SUMNER R. W.: Efficient Simulation of Secondary Motion in Rig-Space. In *SCA* (2013). 3

[JBK*12] JACOBSON A., BARAN I., KAVAN L., POPOVIC J., SORKINE O.: Fast Automatic Skinning Transformations. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 31*, 4 (2012). 2

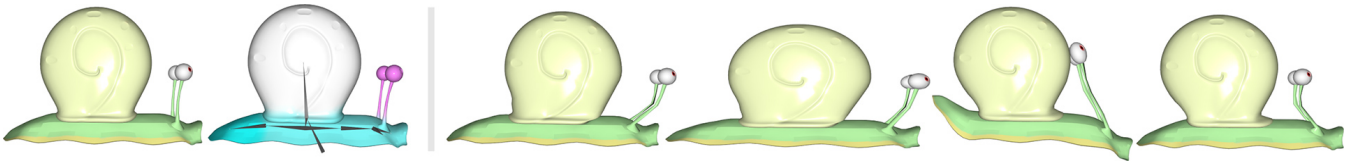


Figure 13: *Example of deformations generated using time-varying skeletal velocities from a single skeleton pose. Floppy weights are shown in the second image: zero on the shell; negative weights on the eyes; and positive weights on the rest of the body. The squash effect is applied uniformly on all parts of the shape.*

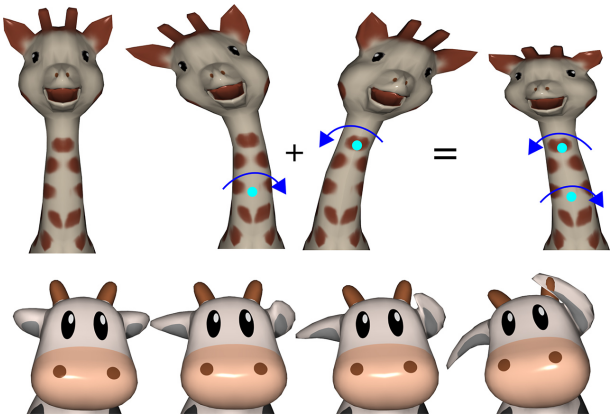


Figure 14: *Limitations of the deformers. Top: applying two consecutive and opposite rotations to the giraffe neck induces scaling artifacts on its head. Bottom: The highly floppy ears of the cow self intersect with the geometry when increasing the velocity of motion.*

[JDKL14] JACOBSON A., DENG Z., KAVAN L., LEWIS J.: Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses* (2014). 2

[JP02] JAMES D. L., PAI D. K.: DyRT: dynamic response textures for real time deformation simulation with graphics hardware. *ACM Trans. on Graphics 21*, 3 (2002). 3

[JS11] JACOBSON A., SORKINE O.: Stretchable and twistable bones for skeletal shape deformation. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (2011), pp. 1–8. 2

[KA08] KASS M., ANDERSON J.: Animating Oscillatory Motion With Overlap: Wiggly Splines. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH* (2008). 3

[KB18] KOMARITZAN M., BOTSH M.: Projective Skinning. *PACM, Proc. I3D 1*, 1 (2018). 3

[KCGF14] KAZI R. H., CHEVALIER F., GROSSMAN T., FITZMAURICE G.: Kitty: Sketching Dynamic and Interactive Illustrations. In *User Interface Software & Technology* (2014). 3

[KCJL06] KIM J.-H., CHOI J.-J., JOON H., LEE I.-K.: Anticipation Effect Generation for Character Animation. *LNCS, Proc. CGI*, 4035 (2006). 3

[KCŽO08] KAVAN L., COLLINS S., ŽÁRA J., O'SULLIVAN C.: Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG) 27*, 4 (2008), 1–23. 2

[KGUF16] KAZI R. H., GROSSMAN T., UMETANI B., FITZMAURICE G.: Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation. *ACM CHI* (2016). 3

[KJP02] KRY P. G., JAMES D. L., PAI D. K.: EigenSkin: real time large deformation character skinning in hardware. In *SCA* (2002). 2

[KL08] KWON J., LEE I.: Exaggerating Character Motions Using Sub-Joint Hierarchy. *Computer Graphics Forum. 27*, 6 (2008), 1677–1686. 3

[KL12] KWON J.-Y., LEE I.-K.: The Squash-and-Stretch Stylization for Character Motions. *IEEE TVCG 18*, 3 (2012). 1, 3

[KS12] KAVAN L., SORKINE O.: Elasticity-Inspired Deformers for Character Articulation. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH Asia 31*, 6 (2012). 2

[Las87] LASSETER J.: Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics. Proc. ACM SIGGRAPH 21*, 4 (1987). 1

[LCa05] LARBOULETTE C., CANI M.-P., ARNALDI B.: Dynamic Skinning: Adding Real-time Dynamic Effects to an Existing Character Animation. In *Spring Conference on Computer Graphics* (2005). 3

[LCF00] LEWIS J., CORDNER M., FONG N.: Pose Space Deformation: A Unified Approach to Shape Interpolation andSkeleton-Driven Deformation. *ACM SIGGRAPH* (2000). 2

[LGXS03] LI Y., GLEICHER M., XU Y.-Q., SHUM H.-Y.: Stylizing Motion with Drawings. In *SCA* (2003). 3

[LL19] LE B. H., LEWIS J.: Direct Delta Mush Skinning and Variants. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 38*, 4 (2019). 2

[LYKL12] LEE S.-Y., YOON J.-C., KWON J.-Y., LEE I.-K.: CartoonModes: Cartoon stylization of video objects through modal analysis. *Graphical Models 74* (2012), 51–60. 3

[May18] MAYA: *Squash and Jiggle deformers*. Autodesk, 2018. https://knowledge.autodesk.com. 3

[MDR*14] MANCEWICZ J., DERKSEN M. L., RIJPKEMA H., , WILSON. C. A.: DeltaMush: Smoothing Deformations While Preserving Detail. In *DigiPro* (2014). 2

[MG03] MOHR A., GLEICHER M.: Building Efficient, Accurate Characer Skins from Examples. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 22*, 3 (2003). 2

[MHTG05] MULLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 24*, 3 (2005). 3

[MK16] MUKAI T., KURIYAMA S.: Efficient Dynamic Skinning with Low-Rank Helper Bone Controllers. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 35*, 4 (2016). 3

[MMC16] MACKLIN M., MULLER M., CHENTANEZ N.: XPBD: Position-Based Simulation of Compliant Constrained Dynamics. *MIG* (2016). 3

[MMG06] MERRY B., MARAIS P., GAIN J.: Animation space: A truly linear framework for character animation. *ACM Transactions on Graphics (TOG) 25*, 4 (2006), 1400–1423. 2

[MTLT88] MAGNENAT-THALMANN N., LAPERRIRE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *In Proceedings on Graphics interface'88* (1988), Citeseer. 2

[MZSE11] MCADAMS A., ZHU Y., SELLE A., EMPEY M.: Efficient elasticityfor character skinning with contact and collisions. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 30*, 4 (2011). 3

[Nao15] NAOYA IWAMOTO AND HUBERT P.H. SHUM AND LONGZHI YANG AND SHIGEO MORISHIMA: Multi-layer Lattice Model for Real-Time Dynamic Character Deformation. *Computer Graphics Forum. Proc. Pacific Graphics 34*, 7 (2015). 3

[NFB16] NIETO J. R., FACEY T., BRUGNOT S.: A Flexible Rigging Framework for VFX and Feature Animation. In *ACM SIGGRAPH Talks* (2016). 2

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A Sketch-Based Interface for Detail-Preserving Mesh Editing. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 24*, 3 (2005). 3

[NT06] NOBLE P., TANG W.: Automatic Expressive Deformations for Stylizing Motion. In *GRAPHITE* (2006). 3

[OM94] OPALACH A., MADDOCK S.: Disney Effects Using Implicit Surfaces. In *Workshop on Animation and Simulation* (1994). 3

[OZ10] OKUN J. A., ZWERMAN S.: *The VES Handbook of Visual Effects. Industry Standard VFX Practices and Procedures*. Focal Press, 2010. 1, 2

[PMRMB15] PONS-MOLL G., ROMERO J., MAHMOOD N., BLACK M. J.: Dyna: A Model of Dynamic Human Shape in Motion. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 34*, 4 (2015). 3

[RF14] RUMMAN N. A., FRATARCANGELI M.: Position based skinning of skeleton-driven deformable characters. In *Proceedings of the 30th Spring Conference on Computer Graphics* (2014), SCCG '14, p. 83–90. 3

[RHC09] ROHMER D., HAHMANN S., CANI. M.-P.: Exact volume preserving skinning with shape control. In *SCA* (2009). 3

[RL13] RAMOS J., LARBOULETTE C.: A Muscle Model for Enhanced Character Skinning. *Journal of WSCG 21*, 2 (2013). 3

[RM13] ROBERTS R., MALLETT B.: A Pose Space for Squash and Stretch Deformation. In *Int. Conf. on Image and Vision Computing* (2013). 1, 3

[RPM] RUHLAND K., PRASAD M., MCDONNEL R.: Data-driven approach to synthesizing facial animation using motion capture. 3

[RRC*18] ROUSSELET V., RUMMAN N. A., CANZIN F., MELLADO N., KAVAN L., BARTHE L.: Dynamic implicit muscles for character skinning. *Computer & Graphics 77* (2018). 3

[TJ81] THOMAS F., JOHNSTON O.: *Disney Animation: The illusion of life*. Disney Editions, 1981. 1, 2, 3, 5, 6, 7

[TPSH14] TARINI M., PANOZZO D., SORKINE-HORNUNG O.: Accurate and efficient lighting for skinned models. *Comput. Graph. Forum 33*, 2 (2014), 421–428. 11

[VBG*13] VAILLANT R., BARTHE L., GUENNEBAUD G., CANI M.-P., ROHMER D., WYVILL B., GOURMEL O., , PAULIN M.: Implicit Skinning: Real-time Skin Deformation with Contact Modeling. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 32*, 4 (2013). 2

[VGB*14] VAILLANT R., GUENNEBAUD G., BARTHE L., WYVILL B., CANI M.-P.: Robust iso-surface tracking for interactive character skinning. *ACM Transactions on Graphics 33*, 6 (Nov. 2014), 1 – 11. 2

[WDAC06] WANG J., DRUCKER S. M., AGRAWALA M., COHEN M. F.: The cartoon animation filter. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 25* (2006). 3

[Wil01] WILLIAMS R.: *The animator's survival kit*. Faber and Faber, 2001. 1

[WJBK15] WANG Y., JACOBSON A., BARBIC J., KAVAN L.: Linear Subspace Design for Real-Time Shape Deformation. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 34*, 4 (2015). 2

[WP02] WANG X. C., PHILLIPS C.: Multi-weight enveloping: least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2002), pp. 129–138. 2

[WWB*19] WANG Y., WEIDNER N. J., BAXTER M. A., HWANG Y., KAUFMAN D. M., SUEDA S.: REDMAX: Efficient & flexible approach for articulated dynamics. *ACM Trans. Graph. 38*, 4 (July 2019). 3

[XB16] XU H., BARBIC J.: Pose-Space Subspace Dynamics. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH 35*, 4 (2016). 3

[YSZ06] YANG X., SOMASEKHARAN A., ZHANG J. J.: Curve skeleton skinning for human and creature characters. *Computer Animation & Virtual Worlds* (2006). 2

[ZBLJ20] ZHANG J. E., BANG S., LEVIN D., JACOBSON A.: Complementary Dynamics. *ACM Trans. on Graphics. Proc. ACM SIGGRAPH Asia* (2020). 3, 11

Appendix A: Derivation of Equation (7)

Starting from Equation (6):

$$\begin{aligned}
\mathbf{v}^u &= \sum_i a_i^u \left(\sum_{j \in \mathcal{A}(i)} \mathbf{v}_j^u \right) \\
&= \sum_i \sum_{j \in \mathcal{A}(i)} a_i^u \mathbf{v}_j^u \\
&= \sum_i \sum_j B(\, j \in \mathcal{A}(i)\,)\, a_i^u \mathbf{v}_j^u \\
&= \sum_j \sum_i B(\, j \in \mathcal{A}(i)\,)\, a_i^u \mathbf{v}_j^u \\
&= \sum_j \sum_i B(\, i \in \mathcal{D}(j)\,)\, a_i^u \mathbf{v}_j^u \\
&= \sum_j \sum_{i \in \mathcal{D}(j)} a_i^u \mathbf{v}_j^u \\
&= \sum_j \mathbf{v}_j^u \left(\sum_{i \in \mathcal{D}(j)} a_i^u \right)
\end{aligned}$$

Where B is the conditional function, such that $B(\text{true}) = 1$ and $B(\text{false}) = 0$, and, the 5th line uses the equivalence

$$j \in \mathcal{A}(i) \Leftrightarrow i \in \mathcal{D}(j)$$

At the end, renaming (i, j) as (j, i), and substituting with Equation (8), gives Equation (7).

Appendix B: Bone-centroids as barycenters

Using the simplifying assumption that the mass of the model is only concentrated at its surface, we approximate the barycenter of the portion of the model affected by a given bone i as:

$$\mathbf{c}_i = \sum_u \bar{a}_i^u m_u \mathbf{p}^u \Big/ \sum_u \bar{a}_i^u m_u \tag{18}$$

where m_u is the area of the Voronoi cell associated to vertex u (that is, one third of the areas of all triangles associated to u). and $\bar{a}_i^u$ is the downward propagated vector (see Figure 2, bottom) given by $\bar{a}_i^u = \sum_{j \in \mathcal{A}(i)} a_j^u$