Projective Skinning

MARTIN KOMARITZAN, Computer Graphics Group, Bielefeld University
MARIO BOTSCH, Computer Graphics Group, Bielefeld University

Fig. 1. Projective skinning avoids the artifacts of Linear Blend Skinning (LBS) and Dual Quaternion Skinning (DQS), yields dynamic physical effects, resolves local collisions, and is real-time capable. From left to right: LBS, DQS, ours.

We present a novel approach for physics-based character skinning. While maintaining real-time performance it overcomes the well-known artifacts of commonly used geometric skinning approaches, it enables dynamic effects, and it resolves local self-collisions. Our method is based on a two-layer model consisting of rigid bones and an elastic soft tissue layer. This volumetric model is easily and efficiently computed from an input surface mesh of the character and its underlying skeleton. In particular, our method neither requires skinning weights, which are often expensive to compute or tedious to hand-tune, nor a complex volumetric tessellation, which fails for many real-world input meshes due to self-intersections.

CCS Concepts: • **Computing methodologies → Physical simulation**; *Motion processing*; *Collision detection*;

Additional Key Words and Phrases: character animation, skinning, projective dynamics

ACM Reference Format:
Martin Komaritzan and Mario Botsch. 2018. Projective Skinning. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 12 (May 2018), 19 pages. https://doi.org/10.1145/3203203

Authors' addresses: Martin KomaritzanComputer Graphics Group, Bielefeld University; Mario BotschComputer Graphics Group, Bielefeld University.

2577-6193/2018/5-ART12 $15.00
https://doi.org/10.1145/3203203

1 INTRODUCTION

When we want to bring virtual characters to life, for example in movies, games, or virtual reality applications, using animation techniques becomes necessary. Independent of their geometric complexity, their movements are intuitively controlled through the joint angles of an embedded skeleton, much like for a real body, where the posture of the skeleton defines the shape of the skin. The virtual counterpart of the latter process is called *skinning*: computing the skin deformation from a change in skeletal pose.

Skinning methods can be roughly divided into three different categories. First, *geometric approaches*, like linear blend skinning [Magnenat-Thalmann et al. 1988], are very fast but can lead to artifacts for strong rotations and do not resolve collisions. Moreover, they require interpolation weights, whose quality greatly influence the resulting deformation. Because of that, determining these skinning weights (*rigging*) is often done by professional artists despite the existence of automatic approaches (e.g. [Baran and Popović 2007; Feng et al. 2015]). Second, *example-based methods*, such as [Lewis et al. 2000; Loper et al. 2015], provide fast and high quality skin deformations, but require a large number of example poses of the character, which have to be modeled by artists or to be obtained by scanning real humans. The third category are *physics-based approaches* (e.g. [Kadleček et al. 2016; McAdams et al. 2011]). They offer very convincing deformations, including collisions handling and dynamic effects like jiggling of soft tissue, but their computational cost is often prohibitive.

When it comes to real-time skinning in interactive applications, most existing methods either suffer from geometric artifacts, require professional artists or extensive training data, or simply are too slow. A promising alternative are methods building on *position-based dynamics*, a fast (game) physics simulation technique that has been successfully used in fluid, solid, and soft body dynamics [Bender et al. 2017]. Impressive performance and robustness are achieved by approximating true dynamic behavior, which in turn makes these methods less accurate and more parameter dependent.

In this paper we propose a skinning technique based on *projective dynamics*, which produces high quality skin deformations, handles local self-collisions, and thereby overcomes the typical skinning artifacts. We achieve real-time performance by using an elasticity formulation on volumetric elements and an efficient collision detection and resolution technique. A major benefit for the user is that our approach requires a minimum amount of input, namely the character's skin mesh and its embedded skeleton.

2 RELATED WORK

In this section we discuss the most relevant related work on geometric, example-based, and physics-based skinning. For more details on character skinning we refer the interested reader to the course notes of Jacobson et al. [2014].

Magnenat-Thalman et al. [1988] introduced Linear Blend Skinning (LBS), which became the standard method to compute skeleton-driven skin deformations. While easy to compute, LBS suffers from artifacts like volume loss in joint regions or even collapsing joints in case of twisting bones. Many approaches try to overcome these drawbacks, for instance by using example poses [Lewis et al. 2000] or additional weights [Wang and Phillips 2002]. The artifacts in LBS are due to its linear interpolation of rotation matrices, which, in general, does not result in a rotation. Dual Quaternion Skinning (DQS) [Kavan et al. 2008] uses dual quaternions instead of rotation and translation matrices to overcome this problem. However, DQS suffers from unnatural bulges in the presence of large rotations. Recently introduced more general skinning transformations [Jacobson et al. 2012; Kavan and Sorkine 2012] lead to better results than LBS and DQS, partly due to optimized skinning weights. Le and Hodgins [2016] avoid the artefacts of LBS and DQS by

finding an optimal center of rotation for each vertex. Since these centers can be precomputed, their performance is comparable to LBS and DQS.

A common drawback of geometric approaches is their dependence on high quality skinning weights, which either are complex to compute [Jacobson et al. 2011; Kavan and Sorkine 2012] or require hand-tuning by skilled artists. Automatic rigging methods [Baran and Popović 2007; Feng et al. 2015] avoid this problem, either by determining skeleton joints and skinning weights through optimization [Baran and Popović 2007] or by transferring them from a high-quality template model to the target model. While the former method requries a closed two-manifold target model, the auto-rigging method of Feng et el. [2015] can be applied directly to low-quality models, such as "dirty" raw 3D-scans.

An inherent problem of geometric skinning approaches is their lack of self-collision handling. Vaillant et al. [2013; 2014] tackle this problem by introducing Implicit Skinning. They describe the skin by an iso-surface of an implicit function and post-process the result of DQS or LBS by projecting vertices onto their initial iso-values. This approach can handle self-collisions but the projection can fail in some cases. Though the resulting skin deformation looks very convincing and can even incorporate muscle bulging, it is rather slow and does not support dynamic skin deformation effects.

Example-based methods try to learn the skinning deformation from a given set of training examples. While first approaches learn corrective terms to an underlying geometric skinning method [Lewis et al. 2000; Weber et al. 2007], recent method learn the whole skinning function [Anguelov et al. 2005; Loper et al. 2015], even including dynamic effects [Kim et al. 2017]. The drawback of example-based method is the need for training examples, which either have to be modeled by a professional artist or are obtained by 3D-scans using expensive equipment. Moreover, the trained model is specific to a certain skeleton topology and skin mesh, and changing either of those requires re-training.

Physics simulations have also been used to compute convincing skin deformations. Their main drawback are the high computational costs. For instance, the accurate biomechanical model of human bodies and motion of Kavan and colleagues [Kadleček et al. 2016; Saito et al. 2015] gives impressive results, but is far too complex to achieve real-time performance. The model of McAdams et al. [McAdams et al. 2011] produces convincing results through a multi-grid simulation on a regular hexahedral grid, but their frame-rates are also not feasible for interactive applications. Capell et al. [Capell et al. 2005, 2002] achieved interactive frame-rates, but only by using a rather coarse volumetric simulation mesh.

To handle higher resolutions at interactive rates, Position Based Dynamics (PBD) [Bender et al. 2017; Müller et al. 2005] has been used in the last decade. PBD simulates elasticity by individually decreasing local elastic energies and can thus be easily parallelized. Rumman et al. [2014; 2015] use PBD to enhance the result of LBS. Although their results overcome the artifacts of linear blending, they cannot handle self-collisions and their simulation produces non-smooth transitions near joints. Pan et al. [2017] extend this approach by additional constraints and collision handling, but they require a post-smoothing step, which inevitably removes surface details. Bender et al. [2013] use a combination of PBD and Shape Matching [Müller et al. 2005] to simulate a three-layer character model, featuring individual stiffness parameters for bone, fat, and skin tissue and handling collisions. While their method produces convincing deformations, the technique is too slow for real-time skinning. Moreover, their method depends on high-quality skinning weights to construct the tissue layers and an initial LBS step.

While PBD is very fast, it has the drawback that different constraints sharing the same vertex alternatingly project to different goal positions, resulting in jumps between those positions [Bouaziz et al. 2014]. Furthermore, the order of individual constraint projections can influence the result.

Bouaziz et al. [2014] introduce *Projective Dynamics* (PD) and show that PBD is a special case of PD. Their method overcomes the artifacts of PBD and converges in fewer iterations, however at the price of a global linear system solve. Fortunately, the global matrix is constant in most cases and hence can be pre-factored. Recently, Liu et al. [2017] and Narain et al. [2017] noticed that PD can be interpreted as a quasi-Newton approach or a special case of ADMM, respectively. Their method can handle more general nonlinear elastic models, but are more complex to implement. Although these methods show clear advantages over Position Based Dynamics, they have (to our best knowledge) not been used in *real-time* skinning simulation.

In this work we employ a simple-to-compute volumetric bone and tissue model (Section 3) and simulate it by projective dynamics, using an elasticity formulation for volumetric tissue elements (Section 4) and efficient collision detection and resolution (Section 5). Our model gives physically plausible results while still maintaining real-time frame-rates. We describe and discuss these individual conributions in the following sections.

3 GENERATING A VOLUMETRIC MESH

Given an input mesh of the character's skin surface and an embedded skeleton, the first step is to compute a volumetric simulation mesh consisting of rigid bones and soft tissue.

Related methods typically compute a tetrahedral mesh that interpolates the skin surface as its outer boundary and the skeleton's bone lines in the interior. We decided against this approach since tetrahedral mesh generation (e.g. [Si 2015]) requires the surface mesh to be closed and intersection-free, which often is not the case. For example, virtual characters are often 3D-scans of real people, where difficult-to-scan concave regions (e.g., crotch, armpits) can suffer from holes or self-intersections. To avoid tedious mesh repair sessions we aim at a mesh generation technique that also works in such real-world cases.

The main problem with LBS and DQS is the unnatural behavior near joints in presence of large rotations. In reality, the skin is stretched around the joint while pressing against it and slightly bulging out. Other approaches [Deul and Bender 2013; Rumman and Fratarcangeli 2014, 2015] try to approximate this behavior by correcting an initial LBS result through additional constraints that preserve the distance between skin and bone-line. In contrast, our method uses volumetric bones and joints that achieve this bulging in a more natural manner without any corrective constraints.

Our volumetric mesh generation consists of two steps. We first inflate the joint positions and line segments of the input skeleton to true volumetric bones and joints. Second, we shrink the skin surface onto the bone surface, such that each skin triangle and its copy on the bone surface span a prismatic element, which we then use for elasticity simulation.

3.1 Bones and Joints

We represent volumetric joints as spheres and volumetric bones as cylinders with spherical caps, which connect the spherical joints at both endpoints of the bone segment (see Figure 2b). With this representation both joints and bones are easily defined by a single radius, are simple to compute, and allow for efficient collision testing. While some joints would more accurately be described by ellipsoids, like for example the elbow, we decided for spherical joints due to their simplicity and computational efficiency.

The user may specify the joint and bone radii directly, but we can also derive them automatically from the skin mesh and the skeleton as follows: The radius of each bone is set to 75 % of the bone's closest distance to the skin surface, and each joint radius is the maximum of the radii of its incident

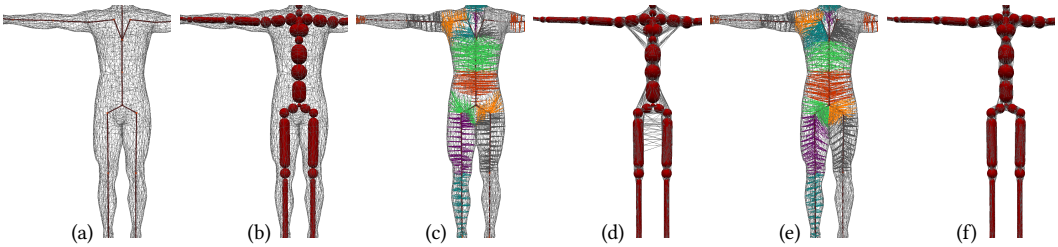

Fig. 2. Given a skeleton and a skin mesh (a), we automatically compute volumetric bones and joints (b). Matching every skin vertex with its closest point on the skeleton (c) leads to artifacts when shrinking the skin to those vertices (d). Laplacian smoothing of the foot-points on the skeleton gives better correspondences (e), which leads to good shrinking results (f).

bones. To ensure that two joints (radii r_1 and r_2) and the connecting bone (radius r_b) fit into one bone segment of length l_b, the following condition has to hold:

$$r_1 + r_2 + 2r_b \;\leq\; l_b. \tag{1}$$

This results in a minimal-length spherical bone in case of equality (see hip bones in Figure 2b). Wherever the volumetric bones do not fit the bone segment lengths, we consecutively scale down the bone and joint radii until the above condition is met. All examples shown in this paper have been produced using this automatic approach.

3.2 Skin Shrinking

In the second step we shrink the vertices of the surface mesh down to the volumetric bones and joints. Given a skin mesh with n vertices $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the goal is to compute for each vertex $\mathbf{x}_i$ a base point $\mathbf{q}_i$ on the volumetric skeleton. To this end we first find a point $\mathbf{s}_i$ on the initial (non-volumetric) input skeleton and then move each vertex $\mathbf{x}_i$ in the direction of $\mathbf{s}_i$ until it hits the volumetric bone, finally yielding the base point $\mathbf{q}_i$.

The trivial ansatz, using as $\mathbf{s}_i$ the point on the input skeleton closest to $\mathbf{x}_i$, is not sufficient, as depicted in Figure 2c,d. Incorporating vertex normals to determine or correct the shrinking direction can somewhat improve the result, but fails for non-smooth skin surfaces. In failure cases neighboring vertices $\mathbf{x}_i$, $\mathbf{x}_j$ on the skin mesh get strongly different points $\mathbf{s}_i$, $\mathbf{s}_j$ on the skeleton.

The latter, however, can easily be corrected by Laplacian smoothing of the base points $\mathbf{s}_i$. We iterate

$$\mathbf{s}_i \;\leftarrow\; \mathbf{s}_i + \lambda \Delta \mathbf{s}_i \;=\; \mathbf{s}_i + \frac{\lambda}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \left(\mathbf{s}_j - \mathbf{s}_i\right), \tag{2}$$

where $\mathcal{N}(i)$ denotes the one-ring neighbors (on the skin mesh) of vertex $\mathbf{x}_i$, $|\mathcal{N}(i)|$ the number of neighbors of $\mathbf{x}_i$, and $\lambda \in [0, 1]$ a parameter to control the smoothing speed (we use $\lambda = 1/2$). Note that (2) does not constrain the points $\mathbf{s}_i$ to lie on a bone-line after the smoothing iteration. We therefore project the updated $\mathbf{s}_i$ back to the skeleton after each iteration. In addition, we keep fixed all points $\mathbf{s}_i$ that lie on an extremal leaf node (e.g., hands, feet).

This smoothing process allows the base vertices to slide along the skeleton (also across joints) and is iterated until convergence. It yields a point distribution where neighboring surface vertices also have neighboring base points (see Figure 2e). With these corresponding base points the skin vertices can be move toward their base points until they lie on a volumetric bone or joint. The resulting shrunken skin is depicted in Figure 2f.

Fig. 3. When twisting the leg bones to turn the knees outwards with non-volumetric bone-lines, elements can twist around the bone-line and undo the transformation, as apparent at the knees and the trousers' seams (left). Volumetric bones (right) naturally resolve this issue.

3.3 Volumetric Tissue Mesh

The skin shrinking results in an inner (bone) and outer (skin) layer with identical vertex connectivity. Each outer face on the skin mesh together with its corresponding inner copy on the bones/joints spans a volumetric element, which we call *prism* with slight abuse of notation. Each prism can be split into three tetrahedra. For a surface mesh with K triangular faces, this leads to a volumetric mesh with $N = 2n$ vertices, K prism elements, and $T = 3K$ tetrahedra.

Our choice of constructing volumetric bones/joints has important benefits. First, attaching the inner tissue layer at a volumetric bone instead of on a non-volumetric bone-line naturally prevents the tissue from twisting around the bone. In contrast, when connecting the skin to a bone-line, as done for instance by Rumman [2014; 2015] and Capell [2005; 2002], a twisting of the bone-line is not noticed by connected elements, resulting in an unnatural deformation (see Figure 3) – unless particularly prevented [Capell et al. 2005]. Second, effects like skin bulging and stretching at bent joints can be achieved in a natural manner, as described in Section 4.2.

4 PROJECTIVE DYNAMICS

Projective Dynamics (PD) [Bouaziz et al. 2014] is a local-global solver for dynamic physical systems. The energy of the whole system is decreased by alternating (i) local projection steps and (ii) the global solution of a linear system of equations. The local step projects subsets of points to their individual closest configuration of vanishing energy. The global step finds a least-squares compromise between the (potentially conflicting) local projections. PD is highly efficient since the local projections can be parallelized and the global matrix is constant and sparse in most cases. Due to its efficiency and robustness, PD is well suited for our skinning simulation. Below we give the main equations of PD required to reproduce our approach, but refer to [Bouaziz et al. 2014] for more details.

We stack our vertex positions into a matrix $\mathbf{X} \in \mathbb{R}^{N\times 3}$ and define $\mathbf{V} \in \mathbb{R}^{N\times 3}$ to hold their velocities. In presence of forces $\mathbf{F} \in \mathbb{R}^{N\times 3}$, an implicit time-integration of Newton's laws of motion with time-step h leads to the update rules:

$$\mathbf{X}_{n+1} = \mathbf{X}_n + h\mathbf{V}_{n+1} \quad \text{and} \quad \mathbf{V}_{n+1} = \mathbf{V}_n + h\mathbf{M}^{-1}\mathbf{F}_{n+1}, \tag{3}$$

where $\mathbf{M}$ is a diagonal lumped mass matrix. Projective dynamics assumes constant external forces $\mathbf{F}_{\text{ext}}$ and position-dependent conservative internal forces induced by potentials $\mathbf{F}_{\text{int}}(\mathbf{X}) = -\sum_i \nabla W_i(\mathbf{X})$. Discretizing the weak form leads to the minimization of

$$\mathbf{X}_{n+1} \;=\; \arg\min_{\mathbf{X}} \frac{1}{2h^2} \left\| \mathbf{M}^{\frac{1}{2}} \left(\mathbf{X} - \tilde{\mathbf{X}}_{n+1} \right) \right\|_F^2 \;+\; \sum_i W_i(\mathbf{X}), \tag{4}$$

where $\tilde{\mathbf{X}}_{n+1} = \mathbf{X}_n + h\,\mathbf{V}_n + h^2\,\mathbf{M}^{-1}\mathbf{F}_{\text{ext}}$. The first term considers preservation of linear momentum and external forces, while the second term tries to minimize the internal potential energy.

The potentials $W_i(\mathbf{X})$ measure the squared deviation of a constrained subset of points, determined by a selector matrix $\mathbf{S}_i$, from the constraint manifold, measured as the distance of the points $\mathbf{S}_i\mathbf{X}$ from their projection $\mathbf{P}_i = \mathbf{P}_i(\mathbf{S}_i\mathbf{X})$ onto the constraint manifold. The potentials therefore have the special form

$$W_i(\mathbf{X}) \;=\; \frac{w_i}{2} \left\| \mathbf{A}_i\mathbf{S}_i\mathbf{X} - \mathbf{B}_i\mathbf{P}_i \right\|_F^2, \tag{5}$$

where w_i is weight for constraint i, and the matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ can be used, e.g., to mean-center the points $\mathbf{S}_i\mathbf{X}$ involved in constraint i or to map positions to gradients, which leads to faster convergence and allows more general constraint types. We employ three types of constraints respectively potentials W_i:

- Base points $\mathbf{q}_i$ are rigidly attached to their bone position, as described in Section 4.2.
- Tetrahedral deformation is penalized by strain constraints, as described in Section 4.1.
- Local collisions are resolved through unilateral constraints, as described in Section 5.

The vertices of the tissue mesh are updated by minimizing (4), which starts by translating all vertices as if there were no internal forces by setting $\mathbf{X}_{n+1} = \tilde{\mathbf{X}}_{n+1}$. Second, the vertex positions $\mathbf{X}_{n+1}$ are fixed and for each constraint the optimal projection $\mathbf{P}_i$ is computed (local step). In the global step, the projections $\mathbf{P}_i$ are fixed and a linear system is solved for the vertex position $\mathbf{X}_{n+1}$:

$$\left(\frac{\mathbf{M}}{h^2} + \sum_i w_i \mathbf{S}_i^\mathsf{T} \mathbf{A}_i^\mathsf{T} \mathbf{A}_i \mathbf{S}_i \right) \mathbf{X}_{n+1} \;=\; \frac{\mathbf{M}}{h^2} \tilde{\mathbf{X}}_{n+1} + \sum_i w_i \mathbf{S}_i^\mathsf{T} \mathbf{A}_i^\mathsf{T} \mathbf{B}_i \mathbf{P}_i. \tag{6}$$

Note that the system matrix is constant if no constraints have to be added or deleted. Therefore, it can be pre-factored once and the linear solve can be performed very efficiently at runtime. Local and global steps are repeated alternatingly to find a converged solution for the current time-step (we use a fixed number of 10 iterations). After computing the new vertex positions $\mathbf{X}_{n+1}$, the new velocities are computed by $\mathbf{V}_{n+1} = \frac{1}{h}(\mathbf{X}_{n+1} - \mathbf{X}_n)$.

4.1 Elastic Strain

Strain constraints penalize the difference between the deformed and undeformed state of an element. We propose to use the elasticity model of Chao et al. [2010], cast into the formalism of projective dynamics. We motivate this design choice by comparing it to the frequently employed as-rigid-as-possible (ARAP) energy of [Sorkine and Alexa 2007], corresponding to the rigidity constraint in [Bouaziz et al. 2012] or the tetrahedron strain in [Deuss et al. 2015]. To allow for a better comparison and discussion, we omit some term simplifications and describe both in the same formalism.

Given a deformation $\mathbf{f}\colon \mathbb{R}^3 \to \mathbb{R}^3$ that maps the undeformed state $\mathbf{x}$ to the deformed state $\mathbf{x}'$, i.e., $f(\mathbf{x}) = \mathbf{x}'$, Chao and colleagues [2010] measure the elastic potential by the deviation of the deformation's differential $\mathrm{d}\mathbf{f} =: \mathbf{F}$ (also called deformation gradient) to the rotation group SO(3). Discretizing with linear tetrahedral elements and casting the discrete strain into PD formalism yields an elastic potential W_i for each element. If a tetrahedral element is given as a matrix $\mathbf{E} \in \mathbb{R}^{3\times 4}$

containing its four mean-centered vertices as columns, as well as its deformed version $\mathbf{E}'$, the potential is

$$W(\mathbf{E}, \mathbf{E}') \;=\; \frac{1}{2}\, V(\mathbf{E}) \min_{\mathbf{R}\in SO(3)} \|\mathbf{F} - \mathbf{R}\|_F^2\,, \tag{7}$$

where V is the volume of the undeformed element, $\mathbf{F}$ the (constant) deformation gradient in this element, and $\mathbf{R}$ the rotation closest to $\mathbf{F}$. The deformation gradient $\mathbf{F}$ is the linear transformation that minimizes $\|\mathbf{FE} - \mathbf{E}'\|_F^2$ and can be found by solving

$$\mathbf{F} \;=\; \mathbf{E}'\mathbf{E}^\mathsf{T} \left(\mathbf{E}\mathbf{E}^\mathsf{T}\right)^{-1}. \tag{8}$$

Its closest rotation $\mathbf{R}$ can be determined via polar decomposition $\mathbf{F} = \mathbf{RS}$, where $\mathbf{S}$ is the symmetric stretch/shear component of the transformation $\mathbf{F}$. To ensure that $\mathbf{R}$ is a true rotation, i.e., $\det(\mathbf{R}) = 1$, the polar decomposition uses the singular value decomposition $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^\mathsf{T}$, sets $\Sigma' = \mathrm{diag}(1, 1, \det(\mathbf{F}))$, and computes $\mathbf{R} = \mathbf{U}\Sigma'\mathbf{V}^\mathsf{T}$.

Note that in case of tetrahedra $\mathbf{F}$ can more efficiently be determined from 3×3 matrices of edge vectors instead of from 3×4 matrices containing vertex positions [Bouaziz et al. 2014]. We use the above (mathematically equivalent) formulation because it is easier to compare to the ARAP approach.

Given $\mathbf{E}$ and $\mathbf{E}'$, computing first $\mathbf{F}$ and then $\mathbf{R}$ corresponds to the local step of PD. With respect to equations (5) and (6), the transposed rotation matrix $\mathbf{R}^\mathsf{T}$ corresponds to the projection $\mathbf{P}_i$ of the strain constraint i, and $\mathbf{B}_i$ is the identity. It hence remains to determine the matrices $\mathbf{A}_i$ for the global step of PD, such that $\mathbf{A}_i\mathbf{S}_i\mathbf{X}$ yields the deformation gradient $\mathbf{F}_i$ of prism $\mathbf{E}_i$. If the $4 \times N$ matrix $\mathbf{S}_i$ selects the four tetrahedron vertices from the $N \times 3$ matrix of unknown deformed vertices $\mathbf{X}$, and the 4×4 matrix $\mathbf{N}$ mean-centers them, then $\mathbf{E}_i'^\mathsf{T} = \mathbf{NS}_i\mathbf{X}$, and we get according to (8):

$$\mathbf{F}_i^\mathsf{T} \;=\; \left(\mathbf{E}_i'\mathbf{E}_i^\mathsf{T}\left(\mathbf{E}_i\mathbf{E}_i^\mathsf{T}\right)^{-1}\right)^\mathsf{T} \;=\; \left(\mathbf{E}_i\mathbf{E}_i^\mathsf{T}\right)^{-1}\mathbf{E}_i\mathbf{E}_i'^\mathsf{T} \;=\; \left(\mathbf{E}_i\mathbf{E}_i^\mathsf{T}\right)^{-1}\mathbf{E}_i\mathbf{NS}_i\mathbf{X},$$

which eventually reveals that $\mathbf{A}_i = (\mathbf{E}_i\mathbf{E}_i^\mathsf{T})^{-1}\mathbf{E}_i\mathbf{N}$.

The frequently used alternative for measuring the deformation of a volumetric element is the ARAP energy [Bouaziz et al. 2012; Deuss et al. 2015; Müller et al. 2005; Sorkine and Alexa 2007]. If the tetrahedra are mean-centered, this energy can be written as

$$W(\mathbf{E}, \mathbf{E}') \;=\; \frac{1}{2} \min_{\mathbf{R}\in SO(3)} \|\mathbf{E}' - \mathbf{RE}\|_F^2\,. \tag{9}$$

It measures the sum of squared differences of the deformed mean-centered points (or edges from the center to the vertices) to the optimally-rotated undeformed points (or edges) of the element. Finding the optimal rotation is the well known orthogonal Procrustes problem [Golub and Van Loan 2012] and is equivalent to

$$\frac{1}{2} \min_{\mathbf{R}\in SO(3)} \Big\| \underbrace{\mathbf{E}'\mathbf{E}^\mathsf{T}}_{\mathbf{G}} - \mathbf{R}\Big\|_F^2. \tag{10}$$

It can be solved via polar decomposition of $\mathbf{G}$. The local projection step yields the target positions $\mathbf{P}_i = \mathbf{R}_i\mathbf{E}_i$, and the global solve tries to match these target positions with the deformed elements' vertices in the least-squares sense (with $\mathbf{A}_i = \mathbf{N}$, $\mathbf{B}_i = \mathbf{I}$ in (6)).

Comparing (8) and (10) reveals that the two approaches decompose different matrices ($\mathbf{F}$ and $\mathbf{G}$) into their rotational ($\mathbf{R}$) and symmetric ($\mathbf{S}$) components. If we assume that $\mathbf{E}' = \mathbf{RS}\,\mathbf{E}$, we get

$$\mathbf{F} = \mathbf{RSE}\,\mathbf{E}^\mathsf{T}\left(\mathbf{E}\mathbf{E}^\mathsf{T}\right)^{-1} = \mathbf{RS} \quad \text{and} \quad \mathbf{G} = \mathbf{RSE}\,\mathbf{E}^\mathsf{T} \neq \mathbf{RS},$$

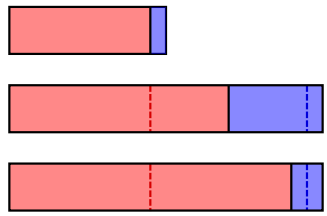

Fig. 4. A two-element bar is stretched by a factor two. When fitting positions (middle), edges minimize the *absolute* difference to their undeformed configuration. When fitting gradients, *relative* differences are minimized, preserving the size ratio of both elements (bottom).

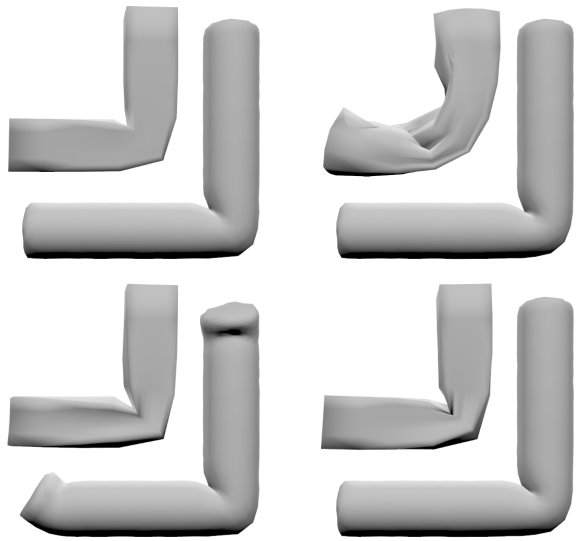

Fig. 5. A high-resolution cylinder and a coarsely triangulated cuboid with two bones each are deformed with different combinations of: decomposing **F** (top) or **G** (bottom) and fitting gradients (left) or positions (right).

i.e., decomposing **F** restores the original rotation **R**, while decomposing **G** in general results in a different rotation. Müller et al. [2007] argue that the factor $(\mathbf{E}\mathbf{E}^\mathsf{T})^{-1}$ is symmetric and therefore should be a part of **S** and *not* contribute to the rotation. They conclude that both variants result in the same rotation **R**. Myronenko et al. [2009] also state both variants to be equivalent. However, since the product of two symmetric matrices is in general not symmetric, this statement is not true, as pointed out earlier by Horn et al. [1988]. Moreover, **F** is independent of the element's shape while **G** is not, a drawback also mentioned by Chao et al. [2010].

Another interesting difference between the two approaches is that minimizing (7) tries to fit deformation gradients while minimizing (9) fits deformed positions/edges. The 2D example in Figure 4 shows the benefit of fitting gradients: When fitting positions/edges, the *absolute* differences between original and deformed edge lengths are minimized, while fitting gradients minimizes their *relative* differences. This difference can become more pronounced in case of irregular and inverted elements, where the changes in position can be small but gradients differ much [Chao et al. 2010]. An example is shown in Figure 5, which shows that decomposing **F** and fitting gradients yields

Fig. 6. Comparison to other skinning approaches on a challenging pose. From top to bottom: Linear blend skinning, dual quaternion skinning, skinning with optimized centers of rotation, skinning with position-based dynamics, projective skinning.

better results in strongly deformed regions. It also shows that exchanging the computation of $\mathbf{R}$ between gradient- and position-fitting does not work at all.

Using a physics-based simulation, our method overcomes the artifacts of LBS (joint collapse) and DQS (bulging), as shown in Figures 1 and 6 and in the accompanying video. For the extreme pose shown in Figure 6 even the CoR-optimized skinning [Le and Hodgins 2016] suffers from slight bulging artifacts in the chest region, which is challenging since it is influenced by more than two bones. We also tried using Position Based Dynamics for our simulation, based on the implementation of [Bender et al. 2017]. We add a spring constraint per tetrahedral edge and a volume-constraint, as proposed in [Rumman and Fratarcangeli 2015]. An example is shown in Figure 6. Our skin shrinking step can lead to tetrahedrons with high aspect ratios, which are problematic for PBD, while our projective dynamics simulation works robustly. Moreover, the PBD framework supports fitting positions only, causing the above-mentioned drawbacks, and PBD needs more iterations to converge than PD. Because each constraint updates its corresponding vertices directly, it is more difficult to parallelize than the independent local steps of PD. Rumman et al. [2015] use a graph coloring algorithm for that purpose. Furthermore, the order in which the constraints are processed can affect the solution and can lead to alternating jumps between different goal positions. PD therefore allows for easier parallelization of the local steps, converges in fewer iterations, and provides better numerical robustness. Its stability under extreme motions and deformations, as demonstrated by Bouaziz et al. [2012], is particularly important for skinning applications, where abrupt changes in joint angles can occur, e.g., due to noise and outliers in motion capture data. As shown in the accompanying video, projective skinnig robustly handles such application scenarios.

4.2 Skin Sliding

Changes in skeleton posture drive the skinning simulation by rigidly attaching the lower layer of the tissue mesh (i.e., the base vertices $\mathbf{q}_i$ of the shrunken skin) to the volumetric skeleton, as described in Section 3. While this anchoring is straightforward for vertices located on a cylindrical *bone*, vertices on a spherical *joint* require special treatment to achieve realistic skin sliding around the joint. When anchoring base vertices at spherical joints, the joint transformation should be chosen as the (quaternion-blended) average of the two incident bones' transformation, instead of as the full transformation of the child bone, as depicted in Figure 7.

However, a rigid motion just poorly mimics the natural behavior: At one side of the bent joint there is less and on the other side there is more space available for the vertices. We would therefore expect that the shrunken skin compresses on one side and stretches on the other side, and that the stretching should not be limited to the region between bone and joint but also affects vertices on the joint.

To achieve this, we define virtual anchor points on the child and parent bones as well as on the joint (2D example in Figure 8, blue dots). In 3D, those anchor points are determined as rings of evenly sampled vertices around the perimeter of the bone/joint. Every base vertex $\mathbf{q}_i$ in between the bone and joint anchors (red points in Figure 8) is represented as a linear combination of four anchor points (two on the nearest bone and two on the joint). Those anchors are rigidly transformed along with bones and the half-rotated joints, and their assigned vertices $\mathbf{q}_i$ are transformed afterwards from the linear combination followed by a projection onto the surface of the volumetric skeleton. This nicely mimics the desired skin sliding behavior around joints and also gives better results for clothed models (see Figure 9).

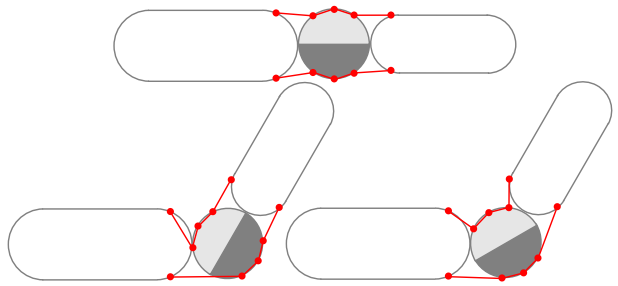

Fig. 7. When joints transform with the full transformation of their child bone, the shrunken skin (red vertices) experiences large stretching on one side (bottom left). Rotating the joint by the average of the incident bones' transformations avoids this problem (bottom right).

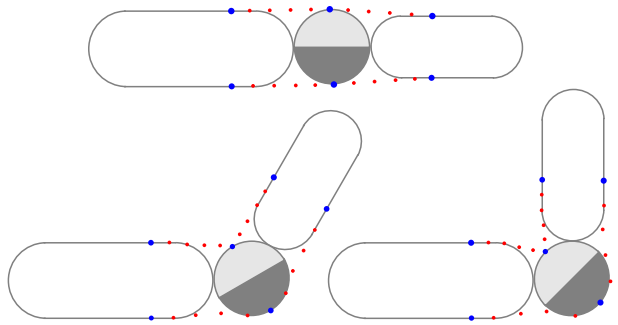

Fig. 8. Vertices of the shrunken skin near joints (red dots) are transformed by linear interpolation of anchor points on the skeleton (blue dots) followed by a projection onto the volumetric bones/joints.

5 COLLISIONS

Proper collision handling greatly improves the realism of any skinning method. It is commonly done in two steps: (i) detecting colliding vertices and (ii) moving those vertices to resolve the collisions. Despite the large body of research on fast collision detection and response, collision handling is still a very time consuming part of soft body simulations due to the $O(T^2)$ possible collision pairs.

We therefore pre-compute a conservative set of potentially colliding vertices and, during run-time, test those vertices only. However, human anatomy provides way too many possibilities for collisions, which result in a prohibitively large set of potential collisions pairs. To still allow for real-time animations, we focus on the handling of *local* collisions in our projective skinning framework. Those occur in joint regions and are (mostly) caused by two incident bones (e.g., elbow region). We neglect *global* collisions (e.g., hand colliding with upper body), since those are typically caused by unreasonable joint angle configurations and hence should be avoided in the character posing stage.

Local self-collisions most frequently occur because two local skin patches collide and interpenetrate. In a preprocessing step we find and store potential collision pairs, consisting of one vertex of each of these two patches, by simulating a set of extreme skeleton postures (Section 5.1). These potential collisions can then efficiently be tested and resolved at run-time (Section 5.2). For the latter we propose a method to enable/disable collision constraints in the PD simulation while avoiding the costly re-factorization of the global system matrix (Section 5.3).

Fig. 9. Comparison of sticking vertices of the shrunken skin to bones and joints (left) and our skin sliding (right).

5.1 Potential Collision Pairs

To collect potential collision pairs, we simulate a set of extreme poses that have been manually designed to provoke different types of collisions in the regions of interest (elbows, shoulders, hip/crotch, knees). During these (offline) simulations we use standard collision detection (accelerated by spatial hashing) and collision handling in our PD framework.

Each of these individual simulations has two responsible bones $\mathbf{b}_1$, $\mathbf{b}_2$ and one joint $\mathbf{j}$ (e.g., upper arm, lower arm, elbow joint). We detect collisions by vertex-in-tetrahedron tests and divide all colliding vertices into two groups $\mathcal{P}_1$, $\mathcal{P}_2$ corresponding to the two colliding skin patches. To this end, we first find the colliding skin vertex $\mathbf{x}_c$ that is closest to the joint $\mathbf{j}$. From $\mathbf{j}$ and $\mathbf{x}_c$ we compute a normal $\mathbf{n} = (\mathbf{x}_c - \mathbf{j}) \times (\mathbf{b}_1 \times \mathbf{b}_2)$ and split the colliding vertices into $\mathcal{P}_1$ and $\mathcal{P}_2$ using this plane. Note that the vertex $\mathbf{x}_c$ is identified in the extreme colliding pose, but the actual splitting is performed more robustly in a collision-free posture, typically the rest pose.

For the two patches $\mathcal{P}_1$ and $\mathcal{P}_2$ we determine collision pairs $(\mathbf{x}_i, \mathbf{x}_j)$ by finding for each vertex $\mathbf{x}_i \in \mathcal{P}_1$ the closest vertex $\mathbf{x}_j \in \mathcal{P}_2$, now again in the colliding pose.

5.2 Online Collision Detection

During the real-time skinning simulation, we test all stored potential collision pairs using a custom-tailored procedure aiming at high performance. For each potential collision pair entry $\{\mathbf{x}_i, \mathbf{x}_j, \mathbf{b}_i, \mathbf{b}_j, \mathbf{j}_{ij}\}$ we make use of the corresponding bones and joint. Let $\mathbf{b}_i$, $\mathbf{b}_j$ denote normalized bone direction vectors, as shown in Figure 10.

For our approximate collision test we project the points $\mathbf{x}_i$ and $\mathbf{x}_j$ into the plane spanned by $\mathbf{j}_{ij}$ and $\mathbf{b}_i, \mathbf{b}_j$, and denote those points by $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}_j$. We report a collision based on the angles in the 2D planar configuration show in Figure 10:

$$\delta c \;=\; \cos\alpha - \cos\beta \;=\; \frac{\mathbf{b}_i \cdot \left(\hat{\mathbf{x}}_i - \mathbf{j}_{ij}\right)}{\left\|\hat{\mathbf{x}}_i - \mathbf{j}_{ij}\right\|} - \frac{\mathbf{b}_j \cdot \left(\hat{\mathbf{x}}_j - \mathbf{j}_{ij}\right)}{\left\|\hat{\mathbf{x}}_j - \mathbf{j}_{ij}\right\|}.$$

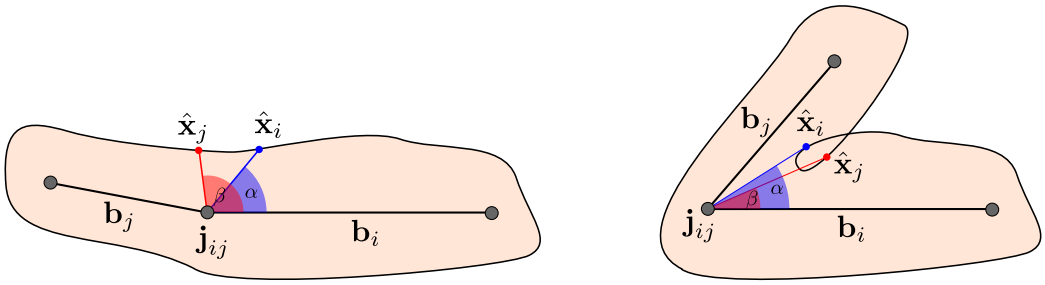

Fig. 10. Our approximate collision detection reports a potential collision if in the projected configuration $\alpha > \beta$.

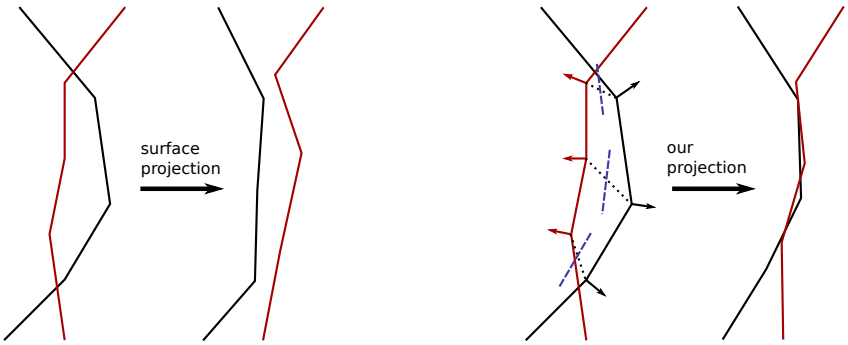

Fig. 11. Common surface projection of colliding vertices imposes a gap after the projection step (left). Our collision response acts between a vertex pair and incorporates vertex normals. Colliding vertices are projected onto a mid-plane (blue dashed lines). While not resolving collisions perfectly, the result is visually plausible and fast because it avoids the computation of a shortest mesh distance.

If δc is negative, i.e., $\alpha > \beta$ we report a collision and setup an unilateral collision constraint as described in the next subsection. Our approximate test might detect a collision in the projected 2D configuration where in 3D there is no collision. This is not a problem, however, since it will be detected by the unilateral collision constraint. The advantage of this collision test is that it is really fast: There is no noticeable difference in frame-rates when adding collision detection and collision constraints.

Another advantage is that this test can easily be generalized to situations where the two bones $\mathbf{b}_i, \mathbf{b}_j$ mainly responsible for the local collisions are not incident, but are connected by a chain of bones, as for instance the shoulder or hip/crotch region (see Figure 2). In such cases we choose the inner (w.r.t. the bone chain) endpoint of $\mathbf{b}_i$ as the joint $\mathbf{j}_{ij}$; everything else stays the same.

5.3 Unilateral Collision Constaints

The common approach for resolving collisions is to translate colliding vertices to the closest point on the object's surface. Bouaziz et al. [2014] suggests applying unilateral constraints to colliding vertices, defined by the plane containing the closest face of the surface. This constraint projects the vertex onto the plane if it is below (i.e., colliding) and otherwise uses its current position as the local projection $\mathbf{P}_i$. In case of self-collisions, however, this projection is not the closest configuration in which the collision is resolved. Instead, it causes a gap between the two skin parts (see Figure 11,

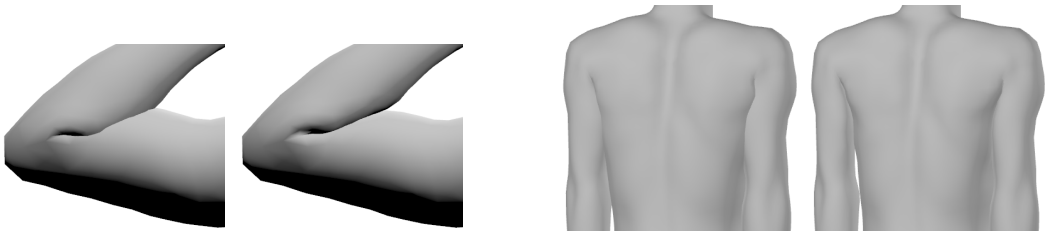

Fig. 12. Comparison of simulations with our collision constraint disabled (left) and enabled (right).

left). Moreover, the collision planes have to be updated because the skin moves and another face might become the closest one to the colliding vertex.

In contrast, we aim at the minimal change in vertex positions to resolve the collision. For each collision pair $(\mathbf{x}_i, \mathbf{x}_j)$ reported by the collision detection we compute an "in-between plane", which is defined by the average position $(\mathbf{x}_i + \mathbf{x}_j)/2$ and the average normal $(\mathbf{n}_i - \mathbf{n}_j)/\|\mathbf{n}_i - \mathbf{n}_j\|$. For collision response, we simply project both vertices onto this plane (if they are on the respectively wrong side). This approach leaves no gaps between the collision-resolved skin parts (see Figure 11, right).

Collision constraints are in general unilateral constraints, which means that the involved vertices will be projected only if a collision occurs. This behavior is difficult to implement efficiently in Projective Dynamics, because for the global solve one projection per constraint is expected. Bouaziz et al. [2014] propose to add a collision constraint to the solver when vertices collide and to remove it afterwards. But this results in many re-factorizations of the global system matrix and would considerably slow down the simulation. Collisions could also be solved in a post-processing step, but that makes them invisible for surrounding vertices. For example, when one vertex is pushed in some direction because of a collision, neighboring vertices should be affected due to elasticity.

One approach could be to always use an additional constraint per vertex that either projects the colliding vertices onto an intersection-free state or uses the current position as projection for non-colliding vertices. However, the latter seemingly "do-nothing" constraint leads to artifacts: If other constraints, like the strain constraint, project to a different position, the collision-related "do-nothing" projection would prevent the involved vertices from following the strains' projection. The high weight typically used for collision constraints even emphasizes this effect. Increasing the iteration count of the alternating local/global steps reduces the artifacts but slows down the simulation.

In our simulation we solve this problem by using two global matrices. The first one does not include collision constraints and the second one adds collision constraints for all potentially colliding vertex pairs. Assuming we would normally compute k local/global PD iterations per frame, we now perform $k/2$ iterations with the first matrix to get a preliminary skinning result without collisions. Afterwards we do $k/2$ iterations with the second global matrix to resolve possible collisions. This time, we have a good initialization for our "do-nothing" projection from the first solves without collisions. In that way, we can incorporate collisions in Projective Dynamics without the above-mentioned artifacts and without slowing down the simulation due to re-factorizations. For a full character with about 800 precomputed collision pairs, the computation of one PD time-step needs just about $2-4\%$ more time than without handling collisions. For comparison, re-factorizing the global matrix would result in an overhead of 300%. Figure 12 compares simulations without and with collision handling.

6 IMPLEMENTATION DETAILS

We implemented our skinning method in C++, use OpenGL for rendering, Shape-Op [Deuss et al. 2015] as a starting point for our Projective Dynamics solver, and Eigen [Guennebaud et al. 2018] for numerical linear algebra. Our examples using Position Based Dynamics are based on the PDB framework of Bender [2017].

In order to maintain real-time performance even for high-resolution character meshes, we selectively decimate the skin mesh using quadric error metric [Garland and Heckbert 1997] and compute the volumetric tissue mesh from the coarser skin surface. To preserve smooth deformations near joints we do not decimate those regions. Other parts, like the lower leg transform almost rigidly and do not suffer from decimation. To up-scale the coarser simulation results to the high-resolution surface meshes, we represent the latter as normal displacement of the simulation mesh [Botsch et al. 2010] and reconstruct it after each simulation time-step.

Regarding performance, the bottleneck of our simulation is the singular value decomposition used in the local projection steps to extract the rotational part of the elements' deformation gradients. To decrease the amount of SVDs per iteration, we perform a clustering of elements similar to [Jacobson et al. 2012]. Our strain constraint (Section 4.1) was purposely derived such that it can be used for *arbitrary* polyhedra by simply adjusting the column count of $\mathbf{E}$ and $\mathbf{E}'$. When applied to the original K prismatic elements (instead of the $T = 3K$ tetrahedra) the local step becomes roughly three times faster, which increases the overall performance by a factor of 1.7. This comes at the cost of decreasing the degrees of freedom of our volumetric representation, which reduces the quality of the skinning results. Fortunately, we can derive the rotation in the local step for the K prismatic elements, but perform the global step on the $T = 3K$ tetrahedra, where each tetrahedron is assigned the rotation of its prismatic element as local projection. This approach combines the smooth deformations of a full tetrahedral simulation with the performance benefits of a reduced simulation on prisms.

We further increase performance by using faster alternatives to SVD for computing the polar decomposition, as proposed by Chao et al. [2010]: We compute the SVD only in case of $\det(\mathbf{F}) < \epsilon$ with $\epsilon = 10^{-5}$, and use the Newton-based approach of Higham [1986] otherwise. Although we use the fast SVD implementation of Sifakis et al. [2011], this approach leads to a performance gain by another factor of 1.5. Benchmark timings for several resolutions of virtual characters are listed in Table 1, where timings were taken on a desktop PC with Intel Xeon CPU (6×3.5 GHz) and a Nvidia GTX 980 GPU. The local projections and the normal displacements for mesh up-scaling are parallelized on the CPU using OpenMP.

Table 1. Timings for several example meshes with #V vertices and #T tetrahedra. Shrinking time t_{shrink}, time to pre-compute collision pairs for one posture t_{col}, and complete pre-computation t_{pre} are given in seconds. Simulation time t_{sim} of one time-step with 10 iterations in milliseconds. Frames-per-second (FPS) also takes transformation of skeleton and shrunken skin plus rendering into account. HR lists frame-rate including up-scaling to the full-resolution visualization mesh (about 20k vertices).

Mesh	#V	#T	t_{shrink}	t_{col}	t_{pre}	t_{sim}	FPS	HR
cylinder	800	4.8k	0.037	-	0.1	3.8	230	-
low-res	1760	10.5k	4.8	1.2	24	8.5	98	85
mid-res	3010	17.7k	8.8	2	42.8	14.7	58	54
high-res	4316	25.2k	12.4	3.2	68	23	37	35

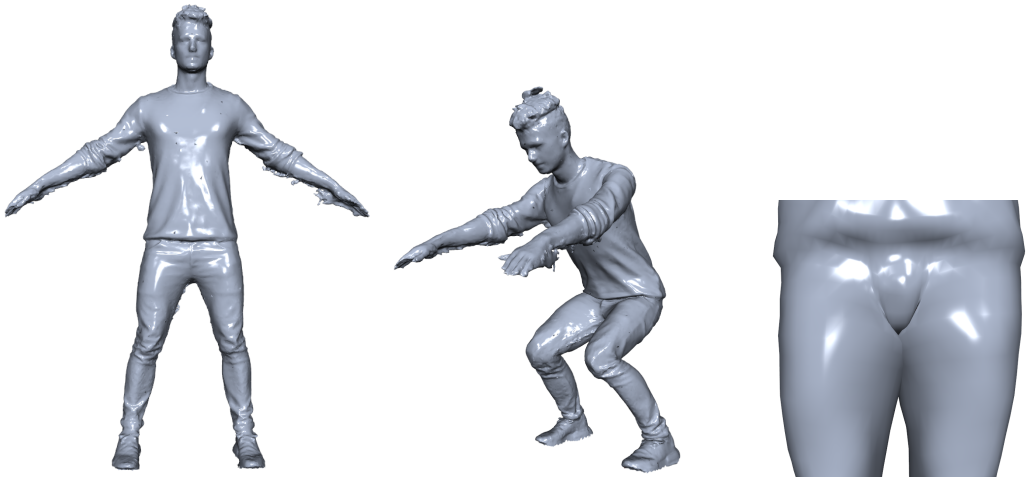

Fig. 13. Left: Raw scan obtained from Photoscan [Agisoft 2017], suffering from noise (arms) and holes due to missing data (feet). Right: All other character models were produced by fitting a human template model to photogrammetry scanner data [Achenbach et al. 2017]. These models are less noisy, but might have self intersections between the legs in the initial T-pose. Despite those challenging artifacts, projective skinning robustly handles these models.

7 DISCUSSION AND FUTURE WORK

Our Projective Skinning method overcomes the artifacts of Linear Blend Skinning and Dual Quaternion Skinning (see Figures 1, 6), resolves local collisions (Figure 11), and provides secondary motion effects—while maintaining real-time performance. Some of these effects are better visible in the accompanying video than in the examples shown in the paper.

Our volumetric skeleton and tissue model is computed automatically from an input skin mesh and a skeleton graph. In particular, our technique does not require any skinning weights, which makes it easy to setup and use. Our volumetric mesh genertion robustly handles non-clean meshes, as demonstrated in Figure 13 for a raw scan suffering from noise and small holes and for a character model with self-intersections. Both models could not be meshed with tetgen [Si 2015]; our projective skinning works robustly. Our volumetric skeleton provides effects like joint bulges and skin sliding in a more natural way than existing approaches, which require additional constraints. Comparing to Position Based Dynamics (Figure 6), our Projective Dynamics simulation is more robust and converges faster. This is crucial, since we do not just correct an initial geometric skinning, but instead compute the full deformation through a physics simulation, which in turn yields a more natural behavior. We efficiently detect and resolve collisions by pre-computing potential collision pairs and through novel collision constraints that do not require computationally expensive re-factorization of the system matrix.

Our method also has some limitations: It is not suitable for meshes with overhanging/overlapping parts, like big bulges of fat or clothing for example. In those cases the skin shrinking can lead to tetrahedra that are partially outside of the character's volume, which can cause unnatural dynamics and erroneous collisions. One solution could be to define additional bones lying inside those body parts. We plan to investigate this idea in future. A second problem is that our collision constraints do not work for collision pairs whose connecting line is roughly perpendicular to the plane spanned by their associated bones. This happens rarely, but it is not impossible. Moreover, because our

collision response incorporates current vertex normals, we observed rare cases where collision response and normal update lead to small oscillations of colliding vertices. We plan to experiment with other collision constraints to make them more robust. Another promising avenue for future work is to investigate recent solvers for nonlinear elasticity, such as ADMM [Narain et al. 2017] or the hyper-elastic solver of Liu et al. [2017]. Moreover, we plan to further improve performance by implementing a GPU version of our method.

ACKNOWLEDGMENTS

This work was supported by the Cluster of Excellence Cognitive Interaction Technology "CITEC" (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG). We are very grateful to Ulrich Schwanecke for his spectacular Taekwondo kicks that bring any skinning technique to its limits.

REFERENCES

Jascha Achenbach, Thomas Waltemate, Marc Erich Latoschik, and Mario Botsch. 2017. Fast Generation of Realistic Virtual Humans. In *Proc. of ACM Symposium on Virtual Reality Software and Technology*. 12:1–12:10.

Agisoft. 2017. Photoscan Pro. http://www.agisoft.com/.

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. *ACM Trans. Graph.* 24, 3 (2005), 408–416.

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Trans. Graph.* 26, 3 (2007).

Jan Bender, Matthias Müller, and Miles Macklin. 2017. A Survey on Position Based Dynamics. In *Eurographics Tuturials*.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon Mesh Processing*. AK Peters, CRC press.

Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. 2012. Shape-Up: Shaping Discrete Geometry with Projections. *Comput. Graph. Forum* 31, 5 (2012), 1657–1667.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4 (2014), 154:1–154:11.

Steve Capell, Matthew Burkhart, Brian Curless, Tom Duchamp, and Zoran Popović. 2005. Physically Based Rigging for Deformable Characters. In *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 301–310.

Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. 2002. Interactive Skeleton-driven Dynamic Deformations. *ACM Trans. Graph.* 21, 3 (2002), 586–593.

Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A Simple Geometric Model for Elastic Deformations. *ACM Trans. Graph.* 29, 4 (2010), 38:1–38:6.

Crispin Deul and Jan Bender. 2013. Physically-Based Character Skinning. In *Proc. of Virtual Reality Interactions and Physical Simulations (VRIPhys)*.

Mario Deuss, Anders Holden Deleuran, Sofien Bouaziz, Bailin Deng, Daniel Piker, and Mark Pauly. 2015. ShapeOp – A Robust and Extensible Geometric Modelling Paradigm. In *Proc. of Design Modelling Symposium*.

Andrew Feng, Dan Casas, and Ari Shapiro. 2015. Avatar Reshaping and Automatic Rigging Using a Deformable Model. In *Proc. of ACM SIGGRAPH Conference on Motion in Games*. 57–64.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *Proc. of Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 209–216.

Gene H. Golub and Charles F. Van Loan. 2012. *Matrix Computation* (4th ed.). John Hopkins University Press.

Gaël Guennebaud, Benoît Jacob, et al. 2018. Eigen v3. http://eigen.tuxfamily.org.

Nicholas J Higham. 1986. Computing the Polar Decomposition with Applications. *SIAM J. Sci. Stat. Comput.* 7, 4 (1986), 1160–1174.

Berthold KP Horn, Hugh M Hilden, and Shahriar Negahdaripour. 1988. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A* 5, 7 (1988), 1127–1135.

Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. 2012. Fast Automatic Skinning Transformations. *ACM Trans. Graph.* 31, 4 (2012), 77:1–77:10.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-time Deformation. *ACM Trans. Graph.* 30, 4 (2011), 78:1–78:8.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and J.P. Lewis. 2014. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH Courses*.

Petr Kadleček, Alexandru-Eugen Ichim, Tiantian Liu, Jaroslav Křivánek, and Ladislav Kavan. 2016. Reconstructing Personalized Anatomical Models for Physics-based Body Animation. *ACM Trans. Graph.* 35, 6 (2016), 213:1–213:13.

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.* 27, 4 (2008), 105:1–105:23.

Ladislav Kavan and Olga Sorkine. 2012. Elasticity-Inspired Deformers for Character Articulation. *ACM Trans. Graph.* 31, 6 (2012), 196:1–196:8.

Meekyoung Kim, Gerard Pons-Moll, Sergi Pujades, Seungbae Bang, Jinwook Kim, Michael J. Black, and Sung-Hee Lee. 2017. Data-driven Physics for Human Soft Tissue Animation. *ACM Trans. Graph.* 36, 4 (2017), 54:1–54:12.

Binh Huy Le and Jessica K. Hodgins. 2016. Real-time Skeletal Skinning with Optimized Centers of Rotation. *ACM Trans. Graph.* 35, 4 (2016), 37:1–37:10.

J.P. Lewis, Matt Cordner, and Nickson Fong. 2000. Pose Space Deformations: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. In *Proc. of SIGGRAPH*. 165–172.

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 3 (2017), 23:1–23:16.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-person Linear Model. *ACM Trans. Graph.* 34, 6 (2015), 248:1–248:16.

Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. 1988. Joint-dependent Local Deformations for Hand Animation and Object Grasping. In *Proc. of Graphics Interface*. 26–33.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph.* 30, 4 (2011), 37:1–37:12.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position Based Dynamics. *J. Vis. Comun. Image Represent.* 18, 2 (2007), 109–118.

Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. 2005. Meshless Deformations Based on Shape Matching. *ACM Trans. Graph.* 24, 3 (2005), 471–478.

Andriy Myronenko and Xubo B. Song. 2009. On the closed-form solution of the rotation matrix arising in computer vision problems. *CoRR* abs/0904.1613 (2009). http://arxiv.org/abs/0904.1613

Rahul Narain, Matthew Overby, and George E. Brown. 2017. ADMM ⊇ Projective Dynamics: Fast Simulation of Hyperelastic Models with Dynamic Constraints. *IEEE Transaction on Visualization and Computer Graphics* 23, 10 (2017), 2222–2234.

Junjun Pan, Lijuan Chen, Yuhan Yang, and Hong Qin. 2017. Automatic skinning and weight retargeting of articulated characters using extended position-based dynamics. *The Visual Computer* (2017), 1–13.

Nadine Abu Rumman and Marco Fratarcangeli. 2014. Position Based Skinning of Skeleton-driven Deformable Characters. In *Proc. of Spring Conference on Computer Graphics*. 83–90.

Nadine Abu Rumman and Marco Fratarcangeli. 2015. Position-Based Skinning for Soft Articulated Characters. *Computer Graphics Forum* 34, 6 (2015), 240–250.

Shunsuke Saito, Zi-Ye Zhou, and Ladislav Kavan. 2015. Computational Bodybuilding: Anatomically-based Modeling of Human Bodies. *ACM Trans. Graph.* 34, 4 (2015), 41:1–41:12.

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2 (2015), 11:1–11:36.

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible Surface Modeling. In *Proc. of Eurographics Symposium on Geometry Processing*. 109–116.

Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. 2013. Implicit Skinning: Real-time Skin Deformation with Contact Modeling. *ACM Trans. Graph.* 32, 4 (2013), 125:1–125:12.

Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. 2014. Robust Iso-surface Tracking for Interactive Character Skinning. *ACM Trans. Graph.* 33, 6 (2014), 189:1–189:11.

Xiaohuan Corina Wang and Cary Phillips. 2002. Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. In *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 129–138.

Ofir Weber, Olga Sorkine, Yaron Lipman, and Craig Gotsman. 2007. Context-Aware Skeletal Shape Deformation. *Computer Graphics Forum* 26, 3 (2007), 265–274.