Volume xx (200y), Number z, pp. 1–15

Realtime Performance Driven Physical Simulation for Facial Animation

V. Barrielle[1,2] and N. Stoiber[2]

[1]CentraleSupélec, France
[2]Dynamixyz, France
vincent.barrielle@m4x.org, nicolas@stoiber.fr

Abstract

We present the first realtime method for generating facial animations enhanced by physical simulation from realtime performance capture data. Unlike purely data-based techniques, our method is able to produce physical effects on the fly through the simulation of volumetric skin behavior, lip contacts and sticky lips. It remains however practical as it does not require any physical/medical data which are complex to acquire and process, and instead relies only on the input of a blendshapes model. We achieve realtime performance on the CPU by introducing an efficient progressive Projective Dynamics solver to efficiently solve the physical integration steps even when confronted to constantly changing constraints. Also key to our realtime performance is a new Taylor approximation and memoization scheme for the computation of the Singular Value Decompositions required for the simulation of volumetric skin. We demonstrate the applicability of our method by animating blendshape characters from a simple webcam feed.

CCS Concepts

•***Computing methodologies*** → ***Animation; Physical simulation;*** *Motion capture;*

Keywords: facial animation, physical simulation, performance capture

1. Introduction

Producing convincing facial animation is a difficult task, which despite years of research effort from the graphics community, still cannot produce results that would fool an observer. Human beings are so used to interpreting subtle variations of facial expressions that any missing detail can drastically reduce the perceived quality of the animation.

The now widespread usage of ever more advanced performance capture systems makes it possible to capture every subtle detail of an actor's performance, but transferring that performance to a virtual character's animation can lead to losing the subtlety of the movements. Indeed, the industry standard for facial rigs, the *blendshapes*, cannot represent the full complexity of facial expressions [LAR*14]. As an affine model, blendshapes are ill-suited for representing the inherently non-linear deformation of the skin.

Simulating the true physical behavior of facial elements does lead to high quality facial animations [SNF05], but building an adequate physical model is a daunting task that requires costly data acquisition, as well as tedious manual work. A recent trend attempts to alleviate these requirements by building physical simulation models on top of blendshapes [BSC16, IKNDP16]. Even though these approaches show good results, are able to simulate non-linear deformations and fine-grained skin dynamics and can resolve lip contacts, their runtime requirements rules out realtime uses and limits them to offline applications.

Both works build upon Projective Dynamics [BML*14], an efficient real-time physics simulation framework, but its direct application to facial simulation does not yield real-time performance, with computation times on the order of one second per frame [IKNDP16]. Simulating the face requires handling a system where interactions between components are in constant evolution due to contact between lips, or transient effects such as sticky lips. These ever-changing interactions prevent optimizations crucial to the computational performance of Projective Dynamics.

In this paper, we propose a performance driven facial animation system able to simulate these physical effects in realtime on the CPU. Our method is based on the *blendforces* paradigm introduced in Barrielle et al. [BSC16]. This paradigm proposes to interpret the geometric deformation patterns of blendshapes as force vectors, that can be activated through time to put the face in motion within a Newtonian physical simulation framework. We enhance this system by deriving a set of volumetric internal forces for a better modeling of skin behavior, and add new forces to model sticky lips (Section 4). We show how we can drive this physical system by performance capture in realtime using a simple webcam (Sec-

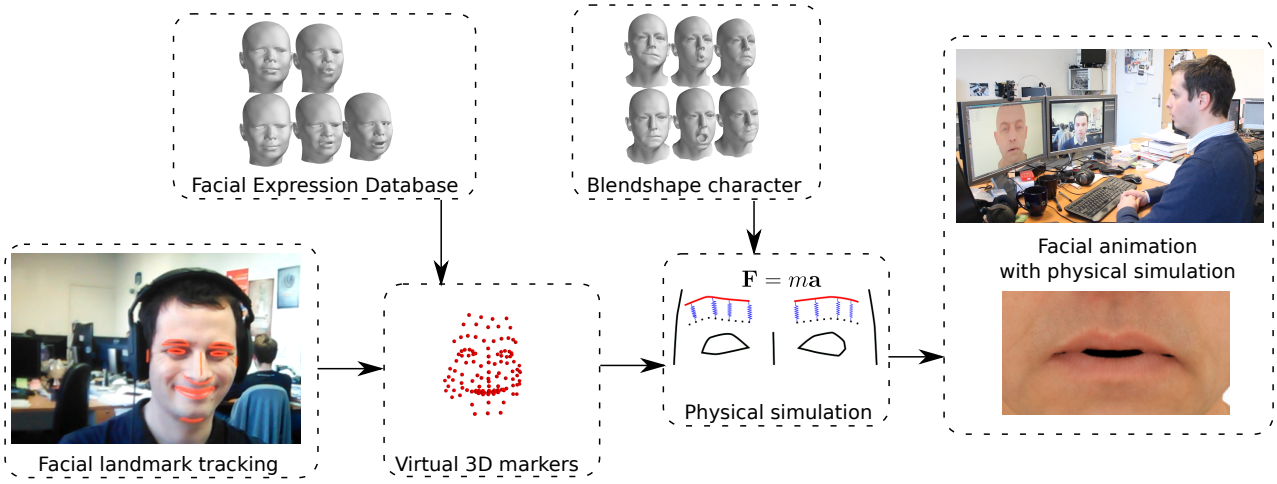

Figure 1: Our system creates physics-based facial animation from an RGB camera, enriching the animation with sticky lips effects.

tion 3). We tackle the problem of realtime simulation by introducing a new Projective Dynamics solver that requires no expensive sparse matrix decomposition and adapts its computational budget to the desired level of accuracy (Section 5). Additionaly, we show how to further reduce the computational cost of volumetric forces by introducing a Taylor approximation and memoization scheme for the Singular Value Decomposition (SVD) those forces require (Section 6). Note that although we build on the blendforces framework, our new simulation mechanisms are general and should be applicable to other physics-based facial animation methods such as Ichim et al. [IKNDP16]. Our main contributions are:

- a method to build a volumetric physical system from a set of facial blendshapes
- a progressive Projective Dynamics solver that preserves its performance in the face of changing constraints and adapts its computational budget to the requested accuracy/time trade-off.
- a Taylor approximation and memoization scheme for SVD computations that drastically reduces the computational resources required for simulating volumetric forces
- a practical facial animation system that integrates this optimized physical simulation with video-based facial performance capture to automatically produce facial animation with physical effects in realtime.

2. Related Work

High-quality facial performance capture Using multiple cameras, Bradley et al. [BHPS10] and later on Beeler and colleagues [BHB*11] captured the precise movements of the skin by leveraging advanced optical flow techniques. Fyffe and coworkers [FJA*14] perfectionned the technique by dynamically choosing the best anchors to avoid drift. Cao and colleagues [CBZB15] demonstrated high-fidelity realtime facial performance capture by combining advanced machine learning techniques and GPU-implemented optical flow. Wu et al. [WBGB16] achieved high-quality reconstructions with a single camera by leveraging an anatomically-inspired model to constrain the tracking.

These methods achieve impressive facial performance capture results, however, exception taken of the method of Cao and colleagues, they require huge computational resources, preventing them from running in real time. They are also designed to capture movements and do not provide retargeting solutions. In this work, we seek to achieve high-quality performance driven animation in real time by leveraging physical simulation to produce the fine-scale details that cannot be captured online. In particular, the optical flow leveraged by these methods frequently fails to capture precise lips contacts.

Recently, deep learning techniques have shown interesting potential for high-quality facial performance capture. Olszewski et al. [OLSL16] used convolutionnal neural networks to recover blendshape weights corresponding to the mouth expression of virtual reality headset users. Laine et al. [LKA*17] leveraged deep learning to learn a mapping from an actor's image to the corresponding high-quality performance captured mesh, allowing for the convenient capture of additional high-quality data. Thanks to their machine learning formulation, these methods can infer coherent data during lips contacts if such information was present in the training set. They cannot however generalize to capture inertial movements that typically lie outside their training space.

Realtime performance driven animation Realtime facial performance capture and animation has been extensively studied. The commoditization of RGBD sensors such as the Kinect led to many capture systems based on RGBD images, a trend pioneered by Weise et al. [WBLP11]. Subsequent works enabled online customization of the facial tracking model [BWP13, LYYB13], robustness to occlusions [HMYL15] and the ability to work on a Virtual Reality setup [LTO*15].

Another trend has been realtime performance driven animation from an RGB camera. The availability of the FaceWarehouse database [CWZ*14] has enabled robust systems from a simple webcam, starting with the person specific 3D regressor from Cao et al. [CWLZ13], later extended to handle multiple identities and lighting conditions [CHZ14]. Thies et al [TZS*16] enhance the ac-

curacy of facial tracking using an inverse illumination approach. To robustify the animation, audio can be used when the facial tracking is failing [LXC*15]. Wang and colleagues [WSXC16] enhance the animation results by capturing the gaze direction as well.

We propose to further improve facial animation results by relying on a physical simulation system driven by performance capture (Section 3), that produces effects difficult to otherwise capture, such as lip contacts and sticky lips.

Physical simulation for facial animation Early attempts at producing facial animation through physical simulation relied on mass-spring systems [PB81, Wat87, TW93]. The quality of the produced animation stepped up with the introduction of finite element models [KGPG96, KGC*96]. Sifakis et al. [SNF05] demonstrated that an anatomically accurate facial simulation model could be animated from motion capture data. However, the acquisition cost of the model restricted the technique to high-end applications. To reduce the cost of building such detailed models, Cong and coworkers [CBE*15] developped a method to automatically transfer a physical system to a new character, and increased artistic control over the results [CBF16].

A recent trend aimed at mixing physical simulation with handcrafted data such as blendshapes. Ma et al. [MWF*12] built a mass-spring system with rest-lengths dynamically controlled by blendshapes activations. In the same spirit, Ichim and colleagues [IKNDP16, IKKP17] interpolated deformation gradients to animate a template-derived volumetric system. Kozlov and coworkers [KBB*17] used physical simulation to enrich an existing physical simulation with inertial effects. Li et al. [LXB16] fitted a tetrahedral mesh to an existing surfacic facial animation to handle self-collisions. Barrielle et al. [BSC16] introduced the blendforces framework where delta-blendshapes are interpreted as a force basis corresponding to muscle activations and animate this physical system from motion capture data. Their physical system is however limited to surfacic forces. Our work expands on it by automatically constructing a volumetric face physical system from a set of blendshapes, thus enabling the simulation of volumetric forces, and by enhancing contact resolution with sticky lips simulation (Section 4).

Realtime physical simulation The Projective Dynamics framework [LBOK13, BML*14] is a robust and fast framework for systems with constant constraints, but has not yet been successfully applied to realtime facial animation. Realtime physically-based facial animation is challenging: state of the art methods can simulate facial performances at around one frame per second [IKNDP16, IKKP17, LXB16].

The first obstacle to performance is the continuously changing constraints due to lips contacts that prevents efficient physical simulation techniques such as Projective Dynamics to use a prefactorized solver. Recent works handled changing systems by using iterative solvers such as an accelerated Jacobi [Wan15, WY16] or a parallel Gauss-Seidel [FTP16]. However, these methods are designed to take advantage of the computational power of the GPU and are likely to be less efficient on the CPU. Narain and colleagues [NOB16] showed that Projective Dynamics could be interpreted as a special case of an ADMM-based implicit Euler integration, and leveraged this fact to accelerate the convergence and simulate non-linear elasticity. Unfortunately, the performance of their method relies on a pre-factorized solver, making the approach unsuitable for the simulation of sticky lips. Liu et al. [LBK17] generalized Projective Dynamics to enable simulation of hyperelastic materials, improving the convergence of Projective Dynamics in the process, but their computational performance still suffers when constraints change. Our approach leverages an iterative solver integrated into the L-BFGS acceleration technique of Liu et al. (Section 5).

The second obstacle to performance lies in the computation of the forces projections. In particular, most volumetric forces require performing numerous SVDs, which are computationally expensive, even with the fast SVD technique of McAdams and colleagues [MST*11]. Fast polar decomposition methods have been used to simulate volumetric elasticity [RJ07, Wan15, MBCM16], however the polar decomposition cannot be used to simulate volume preservation, which is crucial to simulate the behavior of facial skin.

We propose to take advantage of the iterative nature of Projective Dynamics to memoize SVD computations, using a Taylor approximation to enhance the precision of our memoization strategy (Section 6). In this respect, our method shares some similarity with the warm-started fast polar decomposition of Rivers and James [RJ07]. However, our method differs in two important aspects: we compute a full singular value decomposition, and our Taylor analysis gives rise to an adaptive memoization scheme allowing to completely avoid some SVD computations.

3. Performance Driven Physical Simulation

Our facial animation method consists of simulating a physical face system through time. In this section, we recall the principles underlying physical simulation and describe how we can drive a physical face system with performance capture data. An overview of the animation system is presented in Figure 1.

3.1. Physical Simulation

Given a face mesh and its vertices $\mathbf{x}$, we can simulate the effect of Newton's second law of motion on its evolution by performing an implicit Euler integration with for a timestep h:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + h\mathbf{v}_{t+1} \tag{1}$$

$$\mathbf{v}_{t+1} = \mathbf{v}_t + h\mathbf{M}^{-1}\mathbf{f}_{t+1} \tag{2}$$

where $\mathbf{M}$ is the mass matrix of the system, and $\mathbf{f}_t$ is the sum of internal and external forces acting on the system at timestep $t+1$. We use the Projective Dynamics framework [BML*14], where internal forces derive from potentials of the form:

$$W_k(\mathbf{x}) = \frac{w_k}{2} \|\mathbf{G}_k\mathbf{x} - \mathbf{H}_k\mathbf{p}_k\|^2, \tag{3}$$

where $\mathbf{G}_k$ and $\mathbf{H}_k$ are constant differential operators, for instance Laplacian matrices or deformation gradient operators, and $\mathbf{p}_k$ is a projection of $\mathbf{x}$ on the constraint manifold. Our exact choice of internal forces is detailed in Section 4. The external forces acting on our face mesh are gravity and the action of muscles. In this work, we consider the latter to be modeled as vector forces lying in the

linear span of the blendshape matrix $\mathbf{B}$. These *blendforces* are thus expressed as $\mathbf{f}_{\text{muscles}} = \mathbf{Bu}$.

The time integration can be recast as an energy minimization problem [MTGG11]. Indeed, one can combine equations (1) and (2) as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\left(\mathbf{Bu} + \mathbf{f}_{\text{gravity}} - \sum_k \nabla W_k(\mathbf{x}_{t+1})\right). \quad (4)$$

Solving equation (4) is equivalent to minimizing

$$E_{\text{PD}}(\mathbf{x}_{t+1}) = \frac{1}{2h^2}\left\|\mathbf{M}^{1/2}(\mathbf{x}_{t+1} - \mathbf{y}_t)\right\|_F^2 + \sum_k W_k(\mathbf{x}_{t+1}), \quad (5)$$

where $\mathbf{y}_t = \mathbf{x}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\left(\mathbf{Bu} + \mathbf{f}_{\text{gravity}}\right)$ is the predicted state of the system in the absence of internal forces. Projective Dynamics solves this minization problem by alternating between *local steps* which compute the constraint projections $\mathbf{p}_k$ for each internal force and a *global step* that solves the linear problem associated to minimizing equation 5:

$$\mathbf{x}_{t+1} = \left(\frac{\mathbf{M}}{h^2} + \mathbf{Z}\right)^{-1}\left(\frac{\mathbf{M}\mathbf{y}_t}{h^2} + \mathbf{Yp}\right), \quad (6)$$

with $\mathbf{Z} = \sum_k w_k \mathbf{G}_k^T \mathbf{G}_k$, $\mathbf{p}$ a stacking of all the $\mathbf{p}_k$ and $\mathbf{Y} = \sum_k \mathbf{G}_k^T \mathbf{H}_k \mathbf{S}_k$, $\mathbf{S}_k$ being a selection matrix such that $\mathbf{p}_k = \mathbf{S}_k\mathbf{p}$.

Using the Projective Dynamics framework guarantees that the global step matrix $\mathbf{A} = (\mathbf{M}/h^2 + \mathbf{Z})$ is symmetric positive definite (SPD), and constant as long as the support of internal forces does not change. This property allows one to precompute the global step matrix inversion through a sparse Cholesky decomposition, which triggers excellent computational performance up to realtime rates. However, the assumption of constant support is not valid for some internal forces we wish to consider, such as contact response forces on the lips or transient forces like sticky lips. We show how we handle the global step for those cases in Section 5.

3.2. Performance Driven Control

We drive our physical simulation by computing the blendforces actuation parameter $\mathbf{u}$ that reproduces as faithfully as possible facial performances captured by an RGB camera. The blendforces framework enables computing these actuations from motion capture markers, however no markers are available in our setup. We overcome this issue by generating "virtual markers" matching the captured facial performance.

From the video frames we extract facial landmarks $\mathbf{l}$ using a state-of-the-art method [KS14]. Using the 3D blendshape database FaceWarehouse [CWZ*14], we build a parameteric 3D facial shape model capable of modeling variations of identity and expression as in Thies et al [TZN*15]:

$$\mathbf{s}(\beta_{\text{id}}, \beta_{\text{exp}}) = \mu + \mathbf{\Psi}_{\text{id}}\beta_{\text{id}} + \mathbf{\Psi}_{\text{exp}}\beta_{\text{exp}}, \quad (7)$$

where $\mathbf{\Psi}_{\text{id}}$ and $\mathbf{\Psi}_{\text{exp}}$ are matrices storing the principal modes of facial shape variation with respect to identity and expression, respectively. We fit this model to the 2D landmarks by minimizing the energy:

$$E_{\text{fit}} = E_{\text{proj}} + w_{\text{id}}E_{\text{id}} + w_{\text{exp}}E_{\text{exp}}, \quad (8)$$

where E_{id} and E_{exp} are PCA priors preventing the identity and expression parameters from leaving their validity range, and

$$E_{\text{proj}} = \sum_{\mathbf{v}\in S(\mathbf{s})} \|\Pi(\mathbf{Qv} + \mathbf{t}) - \mathbf{l}_{\mathbf{v}}\|_F^2, \quad (9)$$

where Π is the camera projection operator, $S(\mathbf{s})$ is the set of vertices in our parametric face model that have a corresponding landmark point, $\mathbf{l}_{\mathbf{v}}$ is the landmark corresponding to vertex $\mathbf{v}$, and $\mathbf{Q}$ and $\mathbf{t}$ represent the rigid motion of the face model.

We then apply the computed expression parameters β^*_{exp} to identity parameters β^+_{id} corresponding to the neutral mesh of our face physical system, and select some "virtual markers" $\mathbf{t}$ on the resulting mesh:

$$\mathbf{t} = \mathbf{T}\left(\mu + \mathbf{\Psi}_{\text{id}}\beta^+_{\text{id}} + \mathbf{\Psi}_{\text{exp}}\beta^*_{\text{exp}}\right), \quad (10)$$

where $\mathbf{T}$ is a selection matrix for our virtual markers. These markers are then used to compute the blendforces actuation parameters $\mathbf{u}^*$ that bring the facial system closest to the markers:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \left\|\mathbf{t} - \mathbf{T}'\mathbf{x}(\mathbf{u})\right\|^2, \quad (11)$$

where $\mathbf{T}'$ selects vertices corresponding to the virtual markers. Following the control strategy of Barrielle et al [BSC16], we leverage the fact that, for fixed constraint projections $\mathbf{p}$, $\mathbf{x}$ is a linear function of $\mathbf{u}$. In this setup, the control problem of equation 11 can be solved using a linear least-squares solve, which features a constant left-hand side when only considering internal forces with constant support. For this reason, we solve for $\mathbf{u}^*$ without taking transient forces such as lips collisions or sticky lips into account. We justify this approximation by noting that a realistic facial mesh configuration featuring lips self-intersection is typically really close in euclidean distance to a configuration with no self-intersection. Therefore, blendforces actuations computed without considering collision forces should be very close to those computed while taking these forces into account. A similar argument can be used for sticky lips forces. In Barrielle et al, the projections $\mathbf{p}$ and actuations $\mathbf{u}$ are iteratively refined by simulating the face physical system in this contact-free setup. We found that, for realtime purposes, as few as two iterations of this control procedure led to visually satisfying blendforces actuations, freeing some computational budget for physical simulation including lips contacts.

4. Face Physical System

In this section we describe how we build a volumetric physical face system from a set of facial blendshapes, and the nature of internal forces that act in the Newtonian physical simulation introduced in Section 3.

Contrary to some previous approaches, where building a facial physical system required specific and complex data aquisition, we chose to construct one automatically from blendshapes. Not only is this process less tedious, it also renders our approach more practical given wide availability of blendshapes models in real-world applications.

Given a model defined by its neutral shape $\mathbf{x_0}$ and a blendshape matrix $\mathbf{B}$, we interpret $\mathbf{B}$ as a basis of external forces that mimic muscle activations (Section 3). These forces deform a facial mesh

that resists deformation due to internal forces such as skin elasticity or the presence of the skull. In this work, we wish to take the thickness of the flesh into account. To this end, based on the neutral mesh we compute an *hypodermal* surface that represent the skin's thickness, and build a tetrahedral mesh connecting this new surface with the neutral meshs's surface.

4.1. Hypodermal Surface Construction

We build an hypodermal surface by offsetting the neutral mesh's surface and smoothing its high frequency components. We assign to each vertex $\mathbf{v_i}$ a skin thickness value τ_i. These thickness values can be either equal for all vertices or painted on the mesh's surface. We compute an extruded surface $\mathbf{t}$ by subtracting the thicknesses along the normal directions:

$$\mathbf{t}_i = \mathbf{v}_i - \tau_i \mathbf{n}_i, \tag{12}$$

where $\mathbf{n_i}$ is the normal direction at vertex $\mathbf{v_i}$. We then compute the hypodermal surface by minimizing the following energy:

$$\arg\min_{\mathbf{x}_h} \|\mathbf{x}_h - \mathbf{t}\|^2 + w_{\text{smooth}} \|\mathbf{L}\mathbf{x}_h\|^2 + w_{\text{lap}} \|\mathbf{L}\mathbf{x}_h - \mathbf{L}\mathbf{x}_0\|^2, \tag{13}$$

where $\mathbf{L}$ is the laplacian matrix associated with the model's triangle mesh. The parameters w_{smooth} and w_{lap} respectively control the smoothness of the hypodermal surface and how closely its curvature matches that of the neutral mesh.

As each vertex of the hypodermal surface has a corresponding *epidermal* vertex, it is straightforward to join these two surfaces so as to form a tetrahedral mesh. Each triangle in the epidermal surface has a correspong triangle in the hypodermal surface. The volume defined by this extrusion structure can be discretized as 3 tetrahedrons. Since there is more than one possiblity to create these tetrahedrons, we enforce a coherent structure by propagating an initial choice during a traversal of the triangle adjacency graph.

This hypodermal surface construction method provides a simple mean to create a volumetric face mesh, but is not without defects. If the chosen skin thickness values τ_i are too large, badly shaped or intersecting tetrahedra could be created, especially in areas with high curvature. We worked around this issue by specifying lower thickness values in problematic areas.

4.2. Skin Modelling

Based on the volumetric face mesh presented in Section 4.1, we now define the internal forces that will govern how the face deforms in reaction to external forces. As in Ichim et al [IKNDP16] or Barrielle et al [BSC16], our forces are modelled using the Projective Dynamics framework. The skin's elasticity is modelled using an as-rigid-as-possible potential. For each tetrahedron, its potential is written as:

$$W_{\text{strain}}(\mathbf{x}) = w_{\text{strain}} \left\| \mathbf{D}\mathbf{x} - \mathbf{U}\mathbf{V}^{\mathbf{T}} \right\|^2, \tag{14}$$

where $\mathbf{D}$ is the matrix to form the deformation gradient of the tetrahedron with respect to the rest configuration [Wan15], and $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}} = \mathbf{F} = \mathbf{D}\mathbf{x}$ is the SVD of the the deformation gradient. Intuitively, this potential constrains tetrahedron to stay close to a rotated rest configuration.

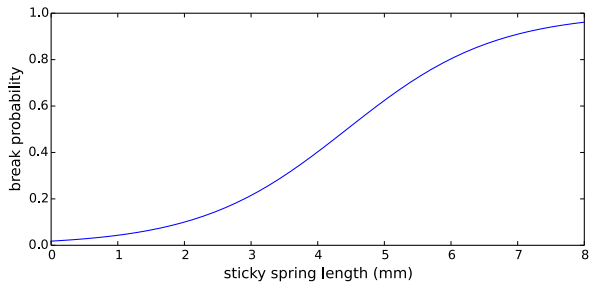

Figure 2: Probability of a sticky lips spring to break as a function of its length.

Like most biological soft tissues, human skin is nearly incompressible [WMG96]. We model this with a potential that penalizes volume changes:

$$W_{\text{vol}}(\mathbf{x}) = w_{\text{vol}} \left\| \mathbf{D}\mathbf{x} - \mathbf{U}\mathbf{\Sigma}^*\mathbf{V}^{\mathbf{T}} \right\|^2, \tag{15}$$

where $\mathbf{\Sigma}^*$ is the closest diagonal matrix of determinant 1 to $\mathbf{\Sigma}$.

We model skin attachment to the skull by adding springs with zero rest-length to the hypodermal vertices, attracting them to their rest positions. As in Barrielle et al [BSC16], all forces's stiffnesses are automatically determined from the blendshapes by solving for the stiffnesses that enable static equilibrium of all forces on each blendshape expression.

4.3. Lips Contacts

Our physical system prevents interpenetration of lips by detecting collisions and adding non-collision constraints to the system. Additionaly, we introduce a new force to handle the sticky lips phenomenon. For each lips collision, we instantiate springs with zero rest-length between the colliding points. These springs remain active on the following frames, but break when their length gets too long, which corresponds to breaking the sticky lips constraint when it is subject to too much tension. More specifically, the probability for a sticky lips spring of length l to break is

$$p_{\text{break}}(l) = \frac{1 - \tanh(2 - rl)}{2} \tag{16}$$

where r is a constant chosen to ensure that the break probability for a caracteristic length l_0 is 0.1. We experimentally set l_0 to 2mm. At the end of each simulated frame, random numbers in the $[0,1]$ range are generated for each sticky lips spring, and we break the springs where the generated random number is lower than $p_{\text{break}}(l)$. The break probability as a function of the sticky spring length is displayed in figure 2.

The actual sticky lips phenomenon depends on lips humidity, making each sticky lips occurence different. Our random breaking scheme reproduces the non-predictible impression given by this natural process. For instance, note that the non-zero probability of

breaking sticky springs of length zero accounts for the fact that sticky lips do not always occur.

5. Progressive Projective Dynamics Solver

5.1. Shortcomings of Projective Dynamics Solvers

The computational performance of Projective Dynamics comes from having a constant system matrix $\mathbf{A}$ in the least-squares problem of equation 5. This enables the use of efficient pre-factorized solvers, typically a Cholesky solver. However, in the face of changing constraints, the global step matrix is no longer constant, and alternative methods have to be explored. Wang [Wan15] solves the global step system using one Jacobi iteration, and relies on Chebyshev multipliers to accelerate convergence, therefore requiring no pre-factorization of the system. However, Jacobi iterations have no convergence guarantees for matrices that are not diagonally dominant, and the system matrix for tetrahedral strain and volume constraints contains strong off-diagonal components. Wang proposes to use under relaxation to overcome this issue, but in our case an under relaxation factor of 0.3 was necessary to avoid divergence, making the convergence too slow to be usable in practice. Besides, Wang recommends turning Chebyshev multipliers off for the first 10 iterations, requiring even more iterations to provide a benefit over Jacobi iterations, which in practice is only affordable on the GPU. Ichim et al [IKNDP16] propose to handle collisions by augmenting the global step system with equality constraints using Lagrange multipliers, however this method cannot handle other kinds of changing constraints. Most notably, this method is not suited to express the additional spring constraints we introduce to simulate sticky lips.

An effective method to boost the convergence properties of Projective Dynamics has been devised by Liu et al [LBK17]. They identify Projective Dynamics as a quasi-Newton method, showing that equation 6 is equivalent to updating $\mathbf{x}_t$ by $-\mathbf{A}^{-1}\nabla E_{\text{PD}}$. They thus see $\mathbf{A}^{-1}$ as an initial inverse Hessian approximation, and propose to improve upon this approximation by integrating the limited-memory BFGS (L-BFGS) procedure into the Projective Dynamics procedure. They show that integrating L-BFGS enables more diverse simulation capabilities and accelerated convergence. Their method shows excellent results, but is not directly suited for the simulation of systems with constraint changes, as it still relies on a pre-factorization of the global step matrix $\mathbf{A}$ for optimal performance. We propose to extend their method to handle systems with constraint changes. To this end, we rely on the Gauss-Seidel iterative process as a replacement of the pre-factorized Cholesky solver. We thereby take advantage of the fact that Projective Dynamics guarantees that the global step matrix is SPD and Gauss-Seidel iterations are guaranteed to converge on SPD matrices.

5.2. Adaptive Gauss-Seidel Iterations

Since our face physical system features changing constraints, we cannot directly employ the L-BFGS accelerated Projective Dynamics method of Liu and colleagues [LBK17]. We propose a variation of their method where we integrate iterations of the Gauss-Seidel solver within the L-BFGS procedure itself to play the role of $\mathbf{A}^{-1}$. We call this optimization scheme Gauss-Seidel-L-BFGS (algorithm 1).

ALGORITHM 1: L-BFGS Gauss-Seidel procedure

Function `gsLbfgs` $(\mathbf{A}, n)(\mathbf{g}, m)$
  Data:
  $\mathbf{A}$: global step system matrix
  n: number of Gauss-Seidel iterations
  $\mathbf{g}$: gradient of E_{PD}
  m: history size
  Result: $\mathbf{d}$, the descent direction
  $\mathbf{q} \leftarrow$ `forwardLbfgs`$(\mathbf{g}, m)$
  `/* `$n-1$` GS iterations, starting from` $\mathbf{q}$ `*/`
  $\mathbf{s} \leftarrow$ `Gauss-Seidel`$(\mathbf{A}, n-1, \mathbf{q}, \mathbf{q})$
  `/* Ensure we computed a descent direction */`
  repeat
    $\mathbf{s} \leftarrow$ `Gauss-Seidel`$(\mathbf{A}, 1, \mathbf{s}, \mathbf{q})$
    $\mathbf{d} \leftarrow$ `backwardLbfgs`$(\mathbf{s}, m)$
  until $\mathbf{g}^T\mathbf{d} < 0$
  return $\mathbf{d}$
end

ALGORITHM 2: Simulation with adaptive Gauss-Seidel solver

for $t \in [1, N_{timesteps}]$ **do**
  $\mathbf{y}_t \leftarrow \mathbf{x}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}f_{\text{ext}}$
  $\mathbf{x}_{t+1,1} \leftarrow \mathbf{y}_t$
  $n \leftarrow 2$
  for $k \in [1, N_{iterations}]$ **do**
    $\mathbf{p} \leftarrow$ projectOnConstraints$(\mathbf{x}_{t+1})$
    $e_0 \leftarrow E_{\text{PD}}(\mathbf{x}_{t+1,k}, \mathbf{p})$
    $\mathbf{g} \leftarrow \nabla E_{\text{PD}}(\mathbf{x}_{t+1,k}, \mathbf{p})$
    $\mathbf{d} \leftarrow$ `gsLbfgs`$(\mathbf{A}, n)(\mathbf{g}, m)$`/* See alg. 1 */`
    $n_{\text{ls}} \leftarrow 0$
    $\alpha \leftarrow 1$
    repeat
      $\mathbf{x}_{t+1,k+1} = \mathbf{x}_{t+1,k} + \alpha\mathbf{d}$
      $e \leftarrow E_{\text{PD}}(\mathbf{x}_{t+1,k+1}, \mathbf{p})$
      $\alpha \leftarrow \alpha/2$
      $n_{\text{ls}} \leftarrow n_{\text{ls}} + 1$
    until $e \leq e_0 + c\alpha\mathbf{g}^T\mathbf{d}$ `/* Armijo condition */`
    if $n_{ls} \neq 1$ **then**
      $n \leftarrow 2n$
    end if
    `updateLbfgsHistory`$(\mathbf{x}_{t+1,k+1}, \mathbf{g})$
  end for
  $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_{t+1,k+1}$
end for

Integrating Gauss-Seidel into L-BFGS The L-BFGS procedure requires an initial Hessian approximation defined only by its ability to be multiplied by a vector. The sparse Cholesky solver used by Liu and colleagues satisfies this condition, and so does an iterative solver such as the Gauss-Seidel solver we propose. However, integrating an iterative solver presents additional difficulties. With an iterative solver used for a *limited* number of iterations, there is no guarantee that the method will effectively produce a descent direction.

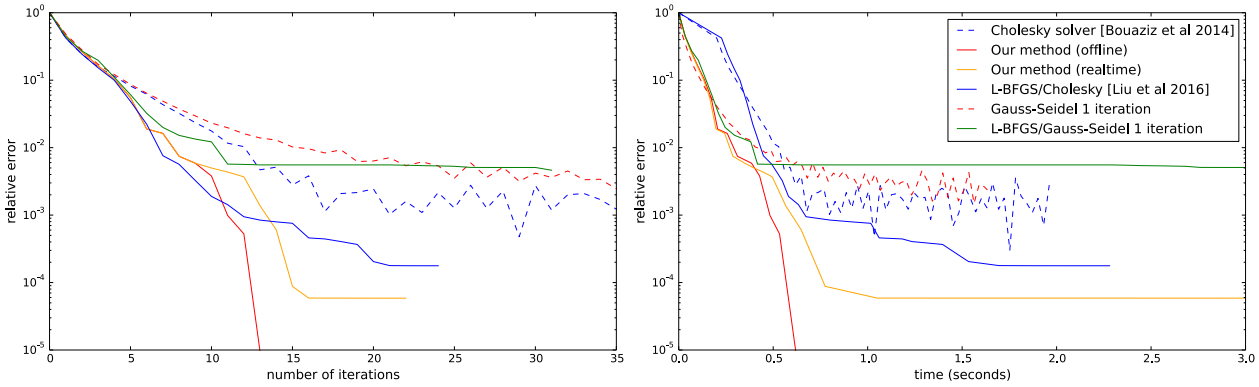

Figure 3: Decrease of the relative error on a typical simulation timestep involving collisions and sticky lips. The relative error is measured with respect to the best converged state among all methods. Timings were measured without using our SVD memoization strategy (Section 6).

To see how we can make sure we get a descent direction, let's briefly recall the functionning of L-BFGS. L-BFGS first transforms the gradient ∇E_{PD} using curvature information accumulated in previous iterations (procedure `forwardLbfgs()` in algorithm 1). Then the initial inverse Hessian is applied, and another transformation by the curvature information (procedure `forwardLbfgs()`) yields the update. For details on these procedures, see algorithm 9.1 from Nocedal and Wright [NW06]. The first loop of this algorithm corresponds to `forwardLbfgs()`, and the second loop corresponds to `forwardLbfgs()`. The update is guaranteed to be a descent direction if the initial inverse Hessian is SPD. Since the Gauss-Seidel iterative method is guaranteed to converge on SPD matrices, given any inital value, there exists a Gauss-Seidel iteration count such that the update obtained is a descent direction. We take advantage of the cheap computational cost of the `forwardLbfgs()` procedure to iteratively check if, after an initial number of iterations, the computed update is a descent direction, augmenting the number of Gauss-Seidel iterations otherwise.

To ensure that this procedure does not result in too many additional Gauss-Seidel iterations, we introduce an heuristic for the choice of the starting point of the iterative method. We found experimentally that using the result of the `forwardLbfgs()` procedure as a starting point leads to a descent direction without additional iterations most of the time.

Number of Gauss-Seidel iterations We place our Gauss-Seidel-L-BFGS procedure as the global step optimization procedure within Projective Dynamics (algorithm 2). Besides avoiding the need for re-factoring matrix $\mathbf{A}$ in the face of changing constraints, using the Gauss-Seidel iterative method allows skipping computations that yield unnecessary precision. Indeed, solving $\mathbf{A}^{-1}\mathbf{q}$ to full Gauss-Seidel convergence wastes computations, since computing new constraint projections in each local step implies solving on a *different* right hand side $\mathbf{q}$ on the following global step. A more efficient scheme is to compute an approximate solution, while ensuring that this approximation is accurate enough to represent an improvement for the overall simulation. We contribute an heuristic to determine an appropriate number of Gauss-Seidel iterations, balancing computational resource usage and accuracy of the solve. Our heuristic rests on the Armijo condition of sufficient decrease. This condition requires that, for a descent direction $\mathbf{d}$:

$$E_{\text{PD}}(\mathbf{x}+\alpha\mathbf{d}) \leq E_{\text{PD}}(\mathbf{x}) + c\alpha\nabla E_{\text{PD}}(\mathbf{x})^T\mathbf{d}, \tag{17}$$

where α is the line search step length, and $c \in [0,1]$ a constant (we use $c = 0.1$). Our L-BFGS scheme includes a backtracking line search step that checks if the decrease of E_{PD} verifies the Armijo condition. We interpret a number of line search iterations greater than one as a hint that the number of Gauss-Seidel iterations was not sufficient. Consequently, we increase the number of iterations for the next global steps in the current timestep. The complete procedure is outlined in algorithms 1 and 2. The procedure `updateLbfgsHistory()` records the curvature information for the latest m iterations. As Liu and colleagues, we choose $m = 5$.

While a full Cholesky factorization is not an option for realtime scenarios, in offline cases, we might find that increasing the number of Gauss-Seidel iterations could lead to a solve slower than a Cholesky solve, albeit less precise. While this issue does not arise in practice for realtime simulation, handling this case is important to use our solver for offline simulations. In offline scenarios, we propose a simple strategy, consisting of computing the Cholesky factorization of the new system matrix in a background thread, and using the Cholesky solve instead of the Gauss-Seidel solve once the factorization is complete. This way, the factorization time has little influence on the time to compute a timestep.

5.3. Convergence Properties

We assert the effectiveness of our solver by measuring its convergence properties on a typical simulation timestep involving collisions and sticky lips. We measure the relative error to the converged state $\mathbf{x}^*$ of the method that achieved the lowest energy. The decrease of the relative error $\varepsilon = (E_{\text{PD}}(\mathbf{x}) - E_{\text{PD}}(\mathbf{x}^*))/(E_{\text{PD}}(\mathbf{x}_0) - E_{\text{PD}}(\mathbf{x}^*))$ is shown in Figure 3. We compare against Projective Dynamics with a Cholesky solver [BML*14] and Projective Dynamics with a solver made of 1 iteration of Gauss-Seidel. Due to the non-smooth derivative of the collision term, these methods without a line search present oscillations once the system gets close to

its converged state. We showcase the convergence behavior of two variants of our method. In the realtime variant, the varying number of Gauss-Seidel iterations is used, while in the offline variant, as described previously, we switch to a Cholesky solver once the offloaded factorization has been computed, at iteration 8 in this experiment. In the first iterations of the process, the convergence behavior is identical. However, after 10 iterations, switching to the offloaded Cholesky solver is beneficial. To assess the importance of our Gauss-Seidel iteration strategy, we compare our method to a version with a single Gauss-Seidel iteration per solve. After a few global iterations, a single Gauss-Seidel iteration does not provide a good descent direction, causing more line search steps to be performed and a stall in relative error reduction.

We also compare to the method of Liu and colleagues [LBK17]. The offline variant of our method is exactly the method of Liu and colleagues after a few iterations (once the Cholesky factorization has been computed), therefore the difference between the two methods stems from a different starting point. Our method enjoys a performance advantage in the early iterations since it does not pay the price of recomputing a Cholesky factorization. One can notice that our offline method achieved the lowest energy here, but that is not always the case, and on other frames the method of Liu and colleagues may get a lower energy. Still, the difference in relative error is always on the order of 10^{-4}, and the convergence curves still exhibit similar shapes in the first iterations. We interpret this phenomenon as an artifact of the line search procedure. Indeed, as Liu and coworkers, we chose a computationally efficient backtracking line search procedure based on Armijo conditions, however the recommended line search strategy for L-BFGS line search should also check the Wolfe conditions, which prevent too short updates, but comes at higher computational cost. While Liu et al did not find issues with this approach, we think our setup, and more particularly collision forces, can lead to very small updates, which cause the procedure to stall sooner than it would with a better line search scheme. For our realtime approach, we still advocate for the simple backtracking line search approach, but for offline simulation integrating Wolfe conditions could be interesting.

These results show that the convergence properties of our method are particularly interesting for realtime use, since it consistently achieves the best relative error reduction in the early parts of the simulation.

6. Fast SVD Approximation

Simulating skin in the Projective Dynamics framework requires computing expensive per-tetrahedron SVDs. Indeed, volume preservation is an important component for the realistic simulation of skin, and cannot be handled without computing the singular values, as opposed to volumetric strain which can be computed with a lighter polar decomposition [Wan15]. Wang noted that, even on the GPU with a fast global step solver, SVD computation would dominate the simulation time. This problem is even more relevant on the CPU, with less computational power than the GPU for the highly parallel local steps. To tackle this issue, we note that, since Projective Dynamics iteratively computes the vertices positions, the deformation gradients of tetrahedrons are unlikely to have changed much between two iterations. It is therefore possible to take advantage of SVDs evaluated on previous iterations to reduce the computational burden. We propose to leverage previous computations with a differential scheme using Jacobians of the SVD to form its first-order Taylor approximation.

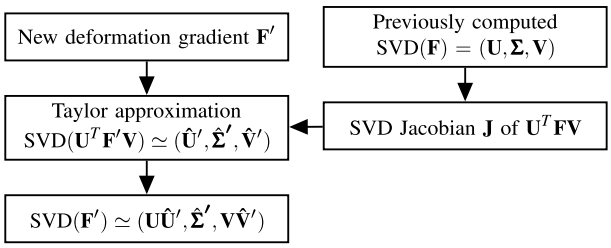

Figure 4: Differential SVD approximation process

6.1. SVD Taylor Expansion

From the SVD decomposition $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ of a matrix $\mathbf{F}$ given by its indices $(F_{ij})_{i,j}$, it is possible to compute the Jacobians of the decomposition $\frac{\partial \mathbf{U}}{\partial F_{ij}}, \frac{\partial \mathbf{\Sigma}}{\partial F_{ij}}, \frac{\partial \mathbf{V}}{\partial F_{ij}}$ [PL00]. However, computing these Jacobians is an $O(n^4)$ operation, even more than the already too expensive $O(n^3)$ of the SVD. While this could suggest that using these Jacobians for a first-order Taylor approximation is too expensive for real-time needs, we show that it is possible to formulate a really cheap procedure to compute them in our setup. In the following, please note that we consider an unusual version of the SVD, where the $\mathbf{U}$ and $\mathbf{V}$ matrices are actual rotations, meaning that the diagonal of $\mathbf{\Sigma}$ can have negative elements.

Consider the deformation gradient of a tetrahedron $\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. Instead of forming the Jacobians of this SVD, we compute the Jacobians of matrix $\mathbf{\Sigma} = \mathbf{U}^T\mathbf{F}\mathbf{V}$. In the following Projective Dynamics iteration, given a new deformation gradient $\mathbf{F}'$, we use these Jacobians in a first-order Taylor expansion to approximate all three SVD matrices of $\mathbf{U}^T\mathbf{F}'\mathbf{V}$. We can then recover an aproximate SVD for F' as:

$$\begin{aligned} \text{SVD}\left(\mathbf{U}^T\mathbf{F}'\mathbf{V}\right) &\simeq \left(\hat{\mathbf{U}}', \hat{\mathbf{\Sigma}}', \hat{\mathbf{V}}'\right) \\ \text{SVD}\left(\mathbf{F}'\right) &\simeq \left(\mathbf{U}\hat{\mathbf{U}}', \hat{\mathbf{\Sigma}}', \mathbf{V}\hat{\mathbf{V}}'\right). \end{aligned} \tag{18}$$

The process for computing an approximate SVD this way is outlined in figure 4. With this setup, we have reduced our problem to computing the SVD Jacobians on a diagonal matrix, $\mathbf{U}^T\mathbf{F}\mathbf{V}$. It turns out that this problem is significantly less expensive, as many steps of the process can be simplified. In fact, the SVD Jacobians turn out to have a close form solution and to exhibit strong sparsity. We only present the resulting Jacobians here, but interested readers are invited to read the complete derivation in appendix A. For a 3×3 diagonal matrix $\mathbf{C}$, let $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T$ be its SVD. The Jacobian for the $\tilde{\mathbf{\Sigma}}$ SVD matrix coefficients is given by

$$\frac{\partial \tilde{\mathbf{\Sigma}}}{\partial \mathbf{C}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \tag{19}$$

where we vectorize matrix $\mathbf{C}$ in row-order fashion. Recalling equation 18, this means that we approximate the $\hat{\mathbf{\Sigma}}'$ component of the

ALGORITHM 3: SVD Jacobian approximation

Function `svdApprox` $(\mathbf{U},\mathbf{\Sigma},\mathbf{V},\mathbf{J},\mathbf{F}')$
  Data:
  $(\mathbf{U},\mathbf{\Sigma},\mathbf{V})$: SVD of previous deformation gradient $\mathbf{F}$
  $\mathbf{J}$: components of the Euler-SVD Jacobian
  $\mathbf{F}'$: new deformation gradient
  Result: $(\tilde{\mathbf{U}},\tilde{\mathbf{\Sigma}},\tilde{\mathbf{V}})$, the approximate SVD
  $\mathbf{C} \leftarrow \mathbf{U}^T\mathbf{F}'\mathbf{V}$
  $\tilde{\mathbf{\Sigma}} \leftarrow$ `diag`$(\mathbf{C})$
  $\theta_0 \leftarrow j_0 C_{12} + j_1 C_{21}$
  $\theta_1 \leftarrow j_2 C_{02} + j_3 C_{20}$
  $\theta_2 \leftarrow j_4 C_{01} + j_5 C_{10}$
  $\theta'_0 \leftarrow j_0 C_{21} + j_1 C_{12}$
  $\theta'_1 \leftarrow j_2 C_{20} + j_3 C_{02}$
  $\theta'_2 \leftarrow j_4 C_{10} + j_5 C_{01}$
  $\tilde{\mathbf{U}} \leftarrow \mathbf{U}\mathbf{R}(\mathbf{\Theta})$
  $\tilde{\mathbf{V}} \leftarrow \mathbf{V}\mathbf{R}(\mathbf{\Theta}')$
  return $(\tilde{\mathbf{U}},\tilde{\mathbf{\Sigma}},\tilde{\mathbf{V}})$
end

SVD of $\mathbf{F}'$ by the diagonal of $\mathbf{U}^T\mathbf{F}'\mathbf{V}$. The Jacobians of $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ have a similar sparsity pattern and similar nonzero values, given by

$$\frac{\partial \tilde{\mathbf{U}}}{\partial \mathbf{C}} \equiv \frac{\partial \tilde{\mathbf{V}}}{\partial \mathbf{C}} \equiv \begin{pmatrix} \bullet & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \bullet & 0 & \bullet & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \bullet & 0 & 0 & 0 & \bullet & 0 & 0 \\ 0 & \bullet & 0 & \bullet & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \bullet & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \bullet & 0 & \bullet & 0 \\ 0 & 0 & \bullet & 0 & 0 & 0 & \bullet & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \bullet & 0 & \bullet & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \bullet \end{pmatrix}, \tag{20}$$

where the bullets $\bullet$ represent nonzero values, which have the forms

$$\bullet \equiv \frac{1}{2}\left(\frac{\pm 1}{\tilde{d}_k + \tilde{d}_l} + \frac{1}{\tilde{d}_k - \tilde{d}_l}\right) \text{ or } \bullet \equiv \frac{\pm 1}{\tilde{d}_k - \tilde{d}_l}, \tag{21}$$

with $\tilde{d}_0, \tilde{d}_1, \tilde{d}_2$ the diagonal values of $\tilde{\mathbf{\Sigma}}$.

The SVD Jacobians in our case thus only require to compute about 15 values, much less than the 108 values required by the method described by Papadopoulo and Lourakis [PL00]. As shown by equation 21, these values have simple close-form values, while the full method would require 2×2 linear solves for each value. As the next section will show, we can further reduce the number of Jacobian values computed to 6, leading to an even more efficient implementation. Timings of our methods show that computing our sparse Jacobian values takes around 9.13ns, whereas computing the full Jacobians takes around 456ns.

6.2. Euler Angles SVD Expansion

Using a first-order Taylor expansion to approximate rotation matrices yields non-orthogonal matrices. To overcome this issue, we formulate our approximations of the SVD's rotation matrices in the Euler angles parameter space, ensuring that we only output true rotation matrices. Since we compute our SVD Jacobians on $\mathbf{U}^T\mathbf{F}\mathbf{V}$, we need to compute the Euler angles Jacobian around the identity. Let us recall the Euler angles definition. For Euler angles $\mathbf{\Theta} = (\theta_0, \theta_1, \theta_2)$, we form the rotation matrix:

$$\mathbf{R}(\mathbf{\Theta}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_0 & s_0 \\ 0 & -s_0 & c_0 \end{pmatrix} \begin{pmatrix} c_1 & 0 & -s_2 \\ 0 & 1 & 0 \\ s_2 & 0 & c_1 \end{pmatrix} \begin{pmatrix} c_3 & s_3 & 0 \\ -s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{22}$$

with c_i, s_i the cosine and sine of θ_i. Inverting the relation and differentiating around $\mathbf{R} = \mathbf{I}$ yields

$$\frac{\partial \mathbf{\Theta}}{\partial \mathbf{R}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{23}$$

Combining this Euler angles Jacobian to our SVD Jacobians yields a sparse matrix, detailed in Appendix A, that can be defined by six values $\mathbf{J} = (j_0, j_1, j_2, j_3, j_4, j_5)^T$ that are computed using equation 29. Once Euler angles for $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ have been computed, we reconstruct the rotation matrices with equation 22. For additional performance gain we approximate the cosine and sine functions with their second and first order Taylor expansions, respectively. Our complete procedure for computing an approximate SVD is described in algorithm 3.

6.3. SVD Memoization Strategy

In the iterative scheme of Projective Dynamics vertex positions update progressively, which opened the way for our Taylor-based approximation of section 6.1. In case of very small changes, we argue that the SVD may not need to be updated at all. We therefore opt for a strategy where we choose for each tetrahedron whether to keep the previous SVD as is, use our Taylor approximation of section 6.1, or compute a full SVD anew. We propose to determine the precision of our Taylor expansion using the matrix norm of our Euler-SVD Jacobian. Since the spectral norm of a matrix is always smaller than its Frobenius norm, we can safely use the Frobenius norm $||\mathbf{J}||_F$ as an approximation of the spectral norm. We can then measure an upper bound on the Euler angles change:

$$\|\Delta\mathbf{\Theta}\| \leq \|\mathbf{J}\|_F \left\|\mathbf{F}' - \mathbf{F}\right\|_F. \tag{24}$$

Using this upper bound, we can adapt at runtime the required computation for a given tetrahedron. We consider two thresholds $\tau_0 < \tau_1$. If $\|\mathbf{J}\|_F\|\mathbf{F}' - \mathbf{F}\|_F < \tau_0$, we keep the results of the previous SVD computation. If $\tau_0 \leq \|\mathbf{J}\|_F\|\mathbf{F}' - \mathbf{F}\|_F < \tau_1$, we use our Taylor approximation. Otherwise, we compute a full SVD decomposition of $\mathbf{F}'$. To avoid drifting, we only compute our approximation with respect to states for which the regular SVD has been computed. Full SVDs are computed for all tetrahedrons at the beginning of a timestep. For improved precision, we progressively reduce the values of τ_0 and τ_1, halving them after each Projective Dynamics iteration.

6.4. Accuracy of the SVD Approximation

To assess the accuracy of our SVD memoization scheme, we sample random deformation gradient-like matrices in the following manner: we sample the Euler angles of the $\mathbf{U}$ and $\mathbf{V}$ matrices, and sample the diagonal matrix $\mathbf{\Sigma}$ by drawing from a normal distribution of mean 1 and standard deviation 0.5. We then sample per-

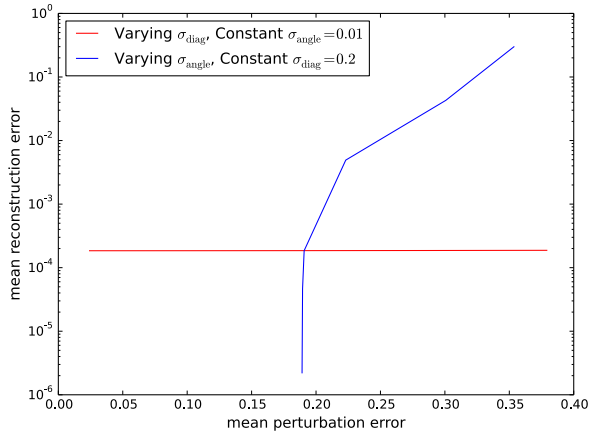

Figure 5: SVD approximation error as a function of the distance to the reference matrix.

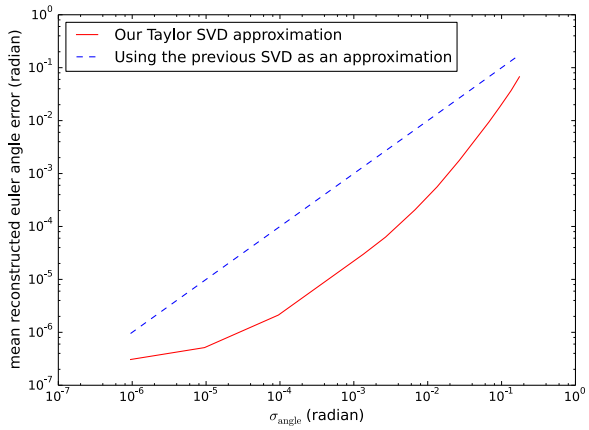

Figure 6: SVD approximation error in terms of Euler angles.

turbations to the Euler angles and the diagonal values by drawing from normal distributions of mean 0 and standard deviations σ_{angle} and σ_{diag}, yielding perturbations $\delta\mathbf{U}, \delta\mathbf{\Sigma}, \delta\mathbf{V}$. We compute our SVD Jacobian on the unperturbed matrices, apply our method to the perturbed matrices, yielding reconstructed matrices $\hat{\mathbf{U}}, \hat{\mathbf{\Sigma}}, \hat{\mathbf{V}}$, and measure the error. In Figure 5, we measure the mean reconstruction Frobenius error $\|\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T - \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\|_F$ as a function of the initial Frobenius error $\|(\mathbf{U}+\delta\mathbf{U})(\mathbf{\Sigma}+\delta\mathbf{\Sigma})(\mathbf{V}+\delta\mathbf{V})^T - \boldsymbol{U\Sigma V}^T\|_F$. Since forming $(\mathbf{U}+\delta\mathbf{U})(\mathbf{\Sigma}+\delta\mathbf{\Sigma})(\mathbf{V}+\delta\mathbf{V})^T$ is linear in $\delta\mathbf{\Sigma}$, we expect the precision of our order 1 approximation to not depend on σ_{diag}, which is confirmed in Figure 5. Up to a certain amount of deviation, our Taylor expansion method yields a satisfying approximation of the SVD. This comforts us in our use of the estimated Euler angle change as a way to predict the approximation error of our method. The reconstructed Euler angle error as a function of σ_{angle} is shown in Figure 6. We can observe that our approximation leads to a substantial amelioration over plain memoization.

We measure the performance of our method by applying it to

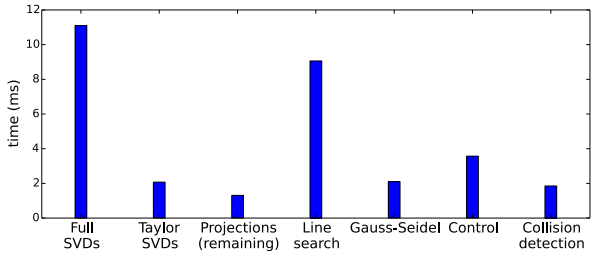

Figure 7: Breakdown of computation times within a timestep involving 50 collision constraints and 110 sticky lips constraints.

2^{24} randomly sampled matrices. The time to compute our approximated SVD for all those matrices amount to 281ms, translating to 16.7ns per SVD. In comparison, the method of McAdams et al [MST*11], which consists of hand-written SIMD intrinsics, yields 369ms for the same number of matrices, translating to 22.0ns per SVD. The per-SVD cost of our method is thus 25% lower.

In addition to this raw computation gain, our method brings further performance improvements with the automatic determination of which SVD computations need and need not be performed (Section 6.3). When used inside our simulation, for a typical timestep, around 48% of tetrahedrons do not need recomputation of the SVD, 34% are computed using our Taylor SVD approximation, and the rest requires a full SVD computation. Without halving the τ_0 and τ_1 thresholds after each iteration, the computed state has a relative error of 10^{-3} compared to the converged solution, but halving them on each of the first five iterations gives a computed state with relative error smaller than 10^{-5}.

7. Results

7.1. Performance Evaluation

We evaluate the performance of our system by simulating a physical face system with 14k vertices and 45k tetrahedrons. The simulation is performed on a desktop computer with a 3.2 GHz Intel Xeon E5-1650 processor, 6 physical cores and 16GB of main memory. All computations are parallelized using OpenMP where applicable, most notably the collision detection and the forces projection. For realtime performance, we simulate the system with 8 Projective Dynamics iterations. Simulating a frame without collisions or sticky lips interaction takes 28ms, while frames with collisions and sticky lips (around 120 additional constraints) take up to 40ms, making it possible to achieve 25fps, the capture rate of our RGB camera. We provide a breakdown of our simulation timings in Figure 7. Our two main bottlenecks are the computation of projections, and the evaluation of E_{PD} in the line search procedure. The projections cost is still dominated by full SVD computations. Most full SVD computations take place in the first Projective Dynamics iterations of a timestep. It should be noted that for these SVD computations, we use the SVD method provided by the Eigen C++ library, which is not as optimized as that of McAdams et al. [MST*11]. Therefore, switching to a more performant SVD implementation could bring even more performance.

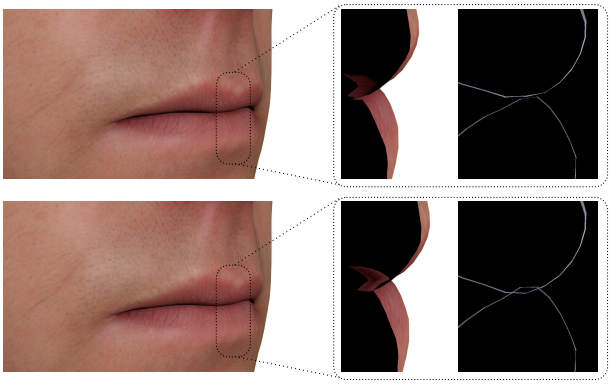

Figure 8: Prevention of lips self-intersections. Top: results of our system. The lips are touching but not intersecting. Bottom: results with collision detection turned off. The lips are intersecting.

Figure 9: Comparison of the shapes of natural sticky lips phenomenon and our simulated sticky lips. The latter generates plausible sticky lips shapes, matching what can be observed in nature.

Figure 10: Comparison of the timing of natural sticky lips phenomenon and our simulated sticky lips. Left: a natural sticky lips sequence, with frames separated by 120ms. Right: a simulated sticky lips sequence, at the same rate.

7.2. Simulation Results

We evaluate our performance driven facial physical simulation system by capturing the performance of individuals in front of a webcam and displaying the simulation results in realtime. Example simulations are shown in the accompanying video. An overview of our results is shown in Figures 14 and 15. Not only does our system faithfully convey the expression of the user, but contrary to previous realtime facial animation systems, our results showcase physical simulation effects as well.

Lips self intersection One of the benefits enabled by physical simulation is the prevention of lips self-intersections, as is shown in Figure 8. Preventing these self-intersections issues has been showcased before [IKNDP16, BSC16], but our system is able to eliminate them while maintaining realtime performance. While it can be argued that lips interpenetration is a subtle effect hard to notice, detecting the contacts is also crucial for the simulation of sticky lips.

Sticky lips Our system produces sticky lips shapes matching those observable in the real world, as can be seen in Figure 9. Furthermore, the timing of our sticky lips is also realistic, as demonstrated in Figure 10. Since our system does not *capture* sticky lips but *simulates* the phenomenon, one cannot expect the system to produce the same sticky lips that would happen in the input sequence. However, we observe that the timing of the phenomenon, as well as its characteristics, closely matches natural sticky lips. Dynamic results of our simulated sticky lips can be observed in the accompanying video.

Non-human characters Our physical system can be used to simulate non-human faces, enabling entertaining inertial effects. We experimented our system on a character with tentacles on the face. Since the tentacles are not attached to the skull and are only a passive element, we disabled the skull attachment force (Section 4.2) for tentacle vertices and excluded the tentacles from the blend-forces matrix $\mathbf{B}$. We demonstrate the interest of physical simulation for such a character by showing that our method takes the effects of gravity and inertial forces into account (Figure 11). We show the results of our method versus a simple blendshape animation, and observe that using physical simulation gives the impression of a flesh-made character while the blendshape animation looks plastic-made. Since the interaction with gravity can cause the tentacles to collide, we use our contact forces to prevent self-intersection, as shown in Figure 12. We can also use our sticky lips force (Sec-

Figure 11: Our method enables integrating facial animation in a realistic environment, allowing the tentacles of the character to react to gravity (top). By comparison, with a plain blendshape animation (bottom), the character seems to be plastic-made rather than flesh made. This effect is best seen in the accompanying video.

tion 4.3) to let tentacles stick to each other for additional artisitic effects (Figure 13).

7.3. Limitations and Future Work

Currently, the precision of our system is limited by the quality of the underlying facial landmark tracker. When the face landmark tracking fails, our physical simulation will maintain a natural facial expression, but the system will not be able to match the subject's expression. We note that very recent facial landmarks trackers such as OpenPose [SJMS17] could alleviate this issue. The landmark tracker's results suffer from jittering issues which also affect the final animation. To the best of our knowledge, current off-the-shelf generic facial tracking methods all suffer from this limitation, and addressing this issue is still an actively researched topic. This jittering issue could be addressed using more intrusive, calibrated motion capture systems, but these would reduce the applicability of our proposed method.

Contrary to the work of Ichim and colleagues, our volumetric simulation does not take into account the full volume of the face. While this allows us to handle non-human characters, this prevents us from explicitly considering the articulated nature of the jaw.

In the future, we would like to investigate further performance enhancement to our systems. Indeed, sticky lips are a phenomenon that is best observed at higher framerates. This could be achieved by parallelizing our Gauss-Seidel solver as in Vivace [FTP16]. Recently, Yu et al. [YJL*16] demonstrated physical simulation for internal articulators such as the tongue, and including this kind of effects would improve the realism of our method. Another interesting improvement would be to handle the contacts between the eyelids and the eyes, which would enforce consistency between skin deformation and gaze direction. As the L-BFGS method of Liu and colleagues [LBK17] generalizes to more general elastic forces, it would be interesting to see if our adaptive Gauss-Seidel scheme can be used with more general materials. We also note that our blendforces do not encode muscular compression effects such as those involved in a lip pucker. These kind of effects could be produced with more advanced blendshapes-derived forces, as seen in the very recent work of Ichim et al [IKKP17], but using this kind of forces in a realtime setup still seems challenging.

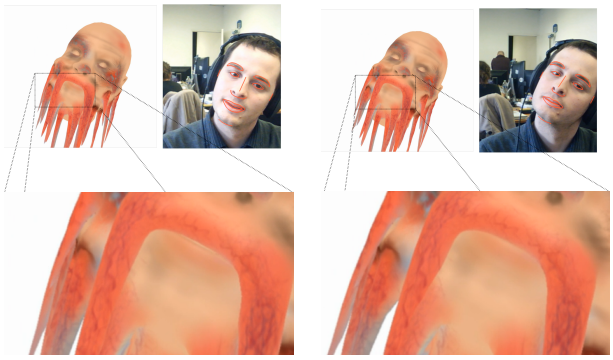

Figure 12: Our system can also prevent self collisions between the tentacles of a stylized character. Left: with collision handling. Right: without collision handling. Without collision handling, the tentacle would penetrate the character's face.

8. Conclusion

We have presented the first performance driven realtime facial animation system based on physical simulation. We introduced improvements to the Projective Dynamics physical simulation framework, enabling the realtime simulation of a volumetric physical face system with changing constraints. This enabled us to simulate complex yet perceptually essential elements of the facial performance. We demonstrated the applicability of our system by driving the physical simulation with realtime performance captured using a simple RGB camera. The resulting simulation contains effects that could not be achieved in other realtime performance driven facial animation methods, such as sticky lips or inertial effects.

Acknowledgements

We would like to thank Carole Garnier for her implementation of the 2D face landmark tracker, and Maxime Thomas Le Déoré and Azuline Teulié for the blendshape models. We also wish to thank Renaud Séguier for proof-reading the paper. We thank the reviewers as well for their insightful comments.

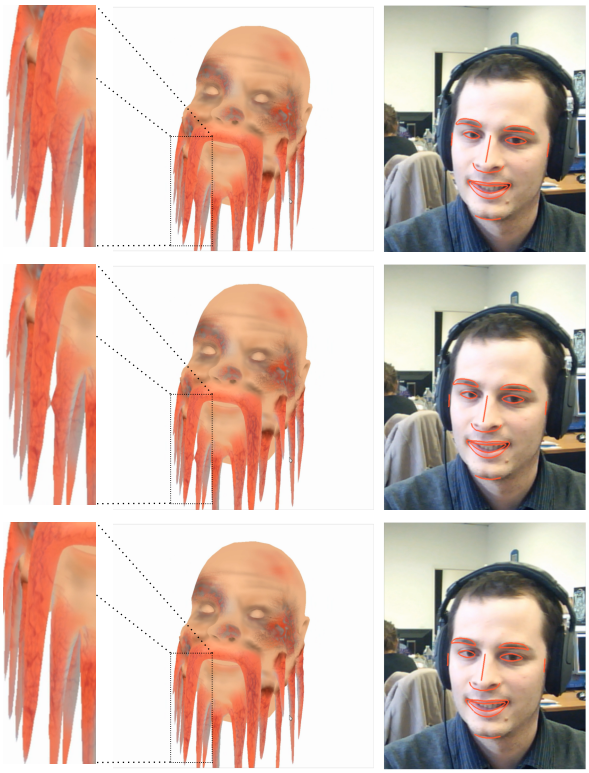

Figure 13: Applying our sticky lips process to tentacles. Gravity causes tentacles to collide, and later on, our sticky force delays the separation between the tentacles. This effect is best seen in the accompanying video.

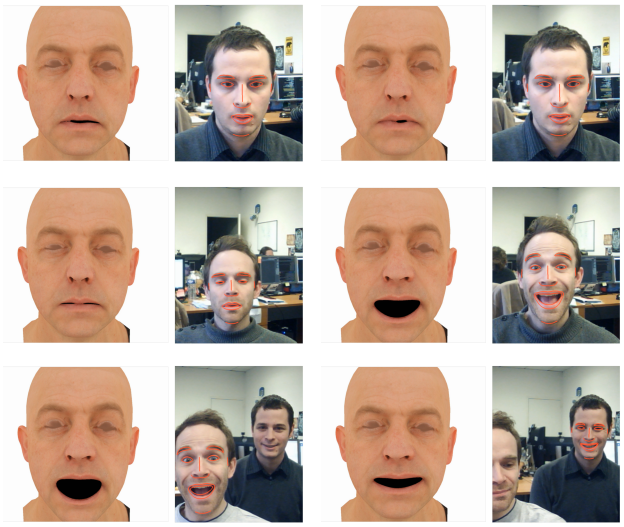

Figure 14: Our system in action, targeting a hyman character. Our method enables the realtime transfer of facial performance from a webcam feed, while enhancing the realism of the resulting animation with physical simulation. The simulated physical effects are the prevention of lips self intersection, the sticky lips effect.

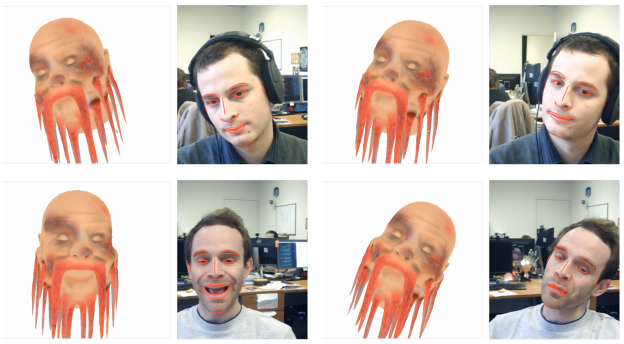

Figure 15: Our system in action, targeting a non-human character. Our physical simulation enables the prevention of tentacles self intersection, sticky tentacles, and models the influence of gravity on tentacles.

References

[BHB*11] BEELER T., HAHN F., BRADLEY D., BICKEL B., BEARDSLEY P., GOTSMAN C., SUMNER R. W., GROSS M.: High-quality passive facial performance capture using anchor frames. In *ACM SIGGRAPH 2011 papers* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 75:1–75:10. 2

[BHPS10] BRADLEY D., HEIDRICH W., POPA T., SHEFFER A.: High resolution passive facial performance capture. In *ACM SIGGRAPH 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH '10, ACM, pp. 41:1–41:10. 2

[BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. 33*, 4 (July 2014), 154:1–154:11. 1, 3, 7

[BSC16] BARRIELLE V., STOIBER N., CAGNIART C.: Blendforces: a dynamic framework for facial animation. *Computer Graphics Forum (Proceedings of Eurographics 2016) 35*, 2 (2016). 1, 3, 4, 5, 11

[BWP13] BOUAZIZ S., WANG Y., PAULY M.: Online modeling for realtime facial animation. *ACM Trans. Graph. 32*, 4 (2013), 40:1–40:10. 2

[CBE*15] CONG M., BAO M., E. J. L., BHAT K. S., FEDKIW R.: Fully automatic generation of anatomical face simulation models. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (New York, NY, USA, 2015), SCA '15, ACM, pp. 175–183. 3

[CBF16] CONG M., BHAT K. S., FEDKIW R.: Art-directed muscle simulation for high-end facial animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2016), SCA '16, Eurographics Association, pp. 119–127. 3

[CBZB15] CAO C., BRADLEY D., ZHOU K., BEELER T.: Real-time high-fidelity facial performance capture. *ACM Trans. Graph. 34*, 4 (July 2015), 46:1–46:9. 2

[CHZ14] CAO C., HOU Q., ZHOU K.: Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Trans. Graph.* (2014). 2

[CWLZ13] CAO C., WENG Y., LIN S., ZHOU K.: 3d shape regression for real-time facial animation. *ACM Trans. Graph. 32*, 4 (2013), 41:1–41:10. 2

[CWZ*14] CAO C., WENG Y., ZHOU S., TONG Y., ZHOU K.: Facewarehouse : a 3d facial expression database for visual computing. *IEEE TVCG 20*, 3 (2014), 413–425. 2, 4

[FJA*14] FYFFE G., JONES A., ALEXANDER O., ICHIKARI R., DEBEVEC P.: Driving high-resolution facial scans with video performance capture. *ACM Trans. Graph. 34*, 1 (Dec. 2014), 8:1–8:14. 2

[FTP16] FRATARCANGELI M., TIBALDO V., PELLACINI F.: Vivace a practical gauss-seidel method for stable soft body dynamics. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 214:1–214:9. 3, 12

[HMYL15] HSIEH P.-L., MA C., YU J., LI H.: Unconstrained realtime facial performance capture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1675–1683. 2

[IKKP17] ICHIM A.-E., KADLEČEK P., KAVAN L., PAULY M.: Phace: Physics-based face modeling and animation. *ACM Transactions on Graphics (TOG) 36*, 4 (2017). 3, 12

[IKNDP16] ICHIM A.-E., KAVAN L., NIMIER-DAVID M., PAULY M.: Building and animating user-specific volumetric face rigs. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2016), SCA '16, Eurographics Association, pp. 107–117. 1, 2, 3, 5, 6, 11

[KBB*17] KOZLOV Y., BRADLEY D., BÃĎCHER M., THOMASZEWSKI B., BEELER T., GROSS M.: Enriching facial blendshape rigs with physical simulation. *Computer Graphics Forum (Proc. Eurographics) 36*, 2 (2017). 3

[KGC*96] KOCH R. M., GROSS M. H., CARLS F. R., VON BÃIJREN D. F., FANKHAUSER G., PARISH Y. I. H.: Simulating facial surgery using finite element models. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 421–428. 3

[KGPG96] KEEVE E., GIROD S., PFEIFLE P., GIROD B.: Anatomy-based facial tissue modeling using the finite element method. In *Proceedings of the 7th Conference on Visualization '96* (Los Alamitos, CA, USA, 1996), VIS '96, IEEE Computer Society Press, pp. 21–ff. 3

[KS14] KAZEMI V., SULLIVAN J.: One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition* (Washington, DC, USA, 2014), CVPR '14, IEEE Computer Society, pp. 1867–1874. 4

[LAR*14] LEWIS J., ANJYO K., RHEE T., ZHANG M., PIGHIN F., DENG Z.: Practice and theory of blendshape facial models. *Eurographics* (2014). 1

[LBK17] LIU T., BOUAZIZ S., KAVAN L.: Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph. 36*, 3 (2017), 23:1–23:16. 3, 6, 8, 12

[LBOK13] LIU T., BARGTEIL A. W., O'BRIEN J. F., KAVAN L.: Fast simulation of mass-spring systems. *ACM Transactions on Graphics 32*, 6 (Nov. 2013), 209:1–7. Proceedings of ACM SIGGRAPH Asia 2013, Hong Kong. 3

[LKA*17] LAINE S., KARRAS T., AILA T., HERVA A., SAITO S., YU R., LI H., LEHTINEN J.: Production-level facial performance capture using deep convolutional neural networks. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (New York, NY, USA, 2017), SCA '17, ACM, pp. 10:1–10:10. 2

[LTO*15] LI H., TRUTOIU L., OLSZEWSKI K., WEI L., TRUTNA T., HSIEH P.-L., NICHOLLS A., MA C.: Facial performance sensing head-mounted display. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2015) 34*, 4 (July 2015). 2

[LXB16] LI Y., XU H., BARBIČ J.: Enriching triangle mesh animations with physically based simulation. *IEEE Transactions on Visualization and Computer Graphics* (2016). 3

[LXC*15] LIU Y., XU F., CHAI J., TONG X., WANG L., HUO Q.: Video-audio driven real-time facial animation. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 182:1–182:10. 3

[LYYB13] LI H., YU J., YE Y., BREGLER C.: Realtime facial animation with on-the-fly correctives. *ACM Trans. Graph. 32*, 4 (2013), 42:1–42:10. 2

[MBCM16] MÃIJLLER M., BENDER J., CHENTANEZ N., MACKLIN M.: A robust method to extract the rotational part of deformations. In *Proceedings of the 9th International Conference on Motion in Games* (New York, NY, USA, 2016), MIG '16, ACM, pp. 55–60. 3

[MST*11] MCADAMS A., SELLE A., TAMSTORF R., TERAN J., SIFAKIS E., STUDIOS W. D. A.: *Computing the Singular Value Decomposition of* 3×3 *matrices with minimal branching and elementary floating point operations*. Tech. rep., 2011. 3, 10

[MTGG11] MARTIN S., THOMASZEWSKI B., GRINSPUN E., GROSS M.: Example-based elastic materials. In *ACM SIGGRAPH 2011 Papers* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 72:1–72:8. 4

[MWF*12] MA W.-C., WANG Y.-H., FYFFE G., CHEN B.-Y., DEBEVEC P.: A blendshape model that incorporates physical interaction. *Comput. Animat. Virtual Worlds 23*, 3-4 (2012), 235–243. 3

[NOB16] NARAIN R., OVERBY M., BROWN G. E.: ADMMâŁĞ projective dynamics: fast simulation of general constitutive models. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2016), Eurographics Association, pp. 21–28. 3

[NW06] NOCEDAL J., WRIGHT S. J.: *Numerical Optimization*. Springer Verlag, 2006. 7

[OLSL16] OLSZEWSKI K., LIM J. J., SAITO S., LI H.: High-fidelity facial and speech animation for VR HMDs. *ACM Transactions on Graphics (Proceedings SIGGRAPH Asia 2016) 35*, 6 (Dec. 2016). 2

[PB81] PLATT S. M., BADLER N. I.: Animating facial expressions. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1981), SIGGRAPH '81, ACM, pp. 245–252. 3

[PL00] PAPADOPOULO T., LOURAKIS M. I.: Estimating the jacobian of the singular value decomposition: Theory and applications. In *European Conference on Computer Vision* (2000), Springer, pp. 554–570. 8, 9, 15

[RJ07] RIVERS A. R., JAMES D. L.: FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph. 26*, 3 (July 2007). 3

[SJMS17] SIMON T., JOO H., MATTHEWS I., SHEIKH Y.: Hand keypoint detection in single images using multiview bootstrapping. In *CVPR* (2017). 12

[SNF05] SIFAKIS E., NEVEROV I., FEDKIW R.: Automatic determination of facial muscle activations from sparse motion capture marker data. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 417–425. 1, 3

[TW93] TERZOPOULOS D., WATERS K.: Analysis and synthesis of facial image sequences using physical and anatomical models. *IEEE Trans. Pattern Anal. Mach. Intell. 15*, 6 (1993), 569–579. 3

[TZN*15] THIES J., ZOLLHÃŰFER M., NIESSNER M., VALGAERTS L., STAMMINGER M., THEOBALT C.: Real-time expression transfer for facial reenactment. *ACM Transactions on Graphics (TOG) 34*, 6 (2015). 4

[TZS*16] THIES J., ZOLLHÃŰFER M., STAMMINGER M., THEOBALT C., NIESSNER M.: Face2face: Real-time face capture and reenactment of rgb videos. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* (2016). 2

[Wan15] WANG H.: A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 246:1–246:9. 3, 5, 6, 8

[Wat87] WATERS K.: A muscle model for animation three-dimensional facial expression. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 17–24. 3

[WBGB16] WU C., BRADLEY D., GROSS M., BEELER T.: An anatomically-constrained local deformation model for monocular face capture. *ACM Trans. Graph. 35*, 4 (July 2016), 115:1–115:12. 2

[WBLP11] WEISE T., BOUAZIZ S., LI H., PAULY M.: Realtime performance-based facial animation. *ACM Transactions on Graphics (Proceedings SIGGRAPH 2011) 30*, 4 (2011). 2

[WMG96] WEISS J. A., MAKER B. N., GOVINDJEE S.: Finite element implementation of incompressible, transversely isotropic hyperelasticity. *Computer methods in applied mechanics and engineering 135*, 1 (1996), 107–128. 5

[WSXC16] WANG C., SHI F., XIA S., CHAI J.: Realtime 3d eye gaze animation using a single rgb camera. *ACM Trans. Graph. 35*, 4 (July 2016), 118:1–118:14. 3

[WY16] WANG H., YANG Y.: Descent methods for elastic body simulation on the gpu. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 212:1–212:10. 3

[YJL*16] YU J., JIANG C., LI R., LUO C. W., WANG Z. F.: Realtime 3d facial animation: From appearance to internal articulators. *IEEE Transactions on Circuits and Systems for Video Technology PP*, 99 (2016), 1–1. 12

Appendix A: SVD Jacobian derivation

In this annex we derive the closed form expressions of the SVD Jacobian for a "centered" deformation gradient. To this end, we recall results from Papadopoulo and Lourakis [PL00]. For a 3×3 matrix $\mathbf{C}$ with elements $(C_{ij})_{i,j}$, let $\tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^T$ be its SVD. The Jacobian for the SVD diagonal is defined by:

$$\frac{\partial \tilde{d}_k}{\partial C_{ij}} = \tilde{u}_{ik}\tilde{v}_{jk}. \tag{25}$$

The Jacobians for the rotation matrices are given by:

$$\begin{aligned}\frac{\partial \tilde{\mathbf{U}}}{\partial C_{ij}} &= \tilde{\mathbf{U}}\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}}\\ \frac{\partial \tilde{\mathbf{V}}}{\partial C_{ij}} &= -\tilde{\mathbf{V}}\mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}},\end{aligned} \tag{26}$$

where the matrices $\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}}$ and $\mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}}$ are defined by the linear systems:

$$\begin{cases}\tilde{d}_l\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}kl} + \tilde{d}_k\mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}kl} &= \tilde{u}_{ik}\tilde{v}_{jl}\\ \tilde{d}_k\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}kl} + \tilde{d}_l\mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}kl} &= -\tilde{u}_{il}\tilde{v}_{jk}.\end{cases} \tag{27}$$

In our case, $\mathbf{C} = \mathbf{U}^T\mathbf{F}\mathbf{V}$, the matrices $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are the identity, which gives a trivial diagonal Jacobian:

$$\frac{\partial \tilde{\mathbf{\Sigma}}}{\partial \mathbf{C}} = \begin{pmatrix}1&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&0\\0&0&0&0&0&0&1&0&0\end{pmatrix}, \tag{28}$$

where we vectorize the matrix $\mathbf{C}$ in row-order fashion. Similarly, equation 27 is greatly simplified, with only four possible right-hand sides: $(0,0)^T, (1,0)^T, (0,-1)^T, (1,-1)^T$. For right-hand sides $(1,0)^T, (0,-1)^T$, the solutions are of the form:

$$\begin{cases}\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}kl} &= \frac{1}{2}\left(\frac{s}{\tilde{d}_k+\tilde{d}_l} + \frac{1}{\tilde{d}_k-\tilde{d}_l}\right)\\ \mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}kl} &= \frac{1}{2}\left(-\frac{s}{\tilde{d}_k+\tilde{d}_l} + \frac{1}{\tilde{d}_k-\tilde{d}_l}\right),\end{cases} \tag{29}$$

for k and l such that $|\tilde{d}_k| \neq |\tilde{d}_l|$ and with $s = \pm 1$. For the $(1,-1)^T$ right-hand side, the solution is:

$$\begin{cases}\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}kl} &= \frac{1}{\tilde{d}_k-\tilde{d}_l}\\ \mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}kl} &= -\frac{1}{\tilde{d}_k-\tilde{d}_l},\end{cases} \tag{30}$$

again with diagonal values of different magnitudes. As suggested by Papadopoulo and Lourakis [PL00], the equality case must be handled with a least-squares solve. This least-squares solve has a closed form solution as well. Suppose $\tilde{d}_k = \pm\tilde{d}_l$. Then only the $(1,0)^T, (0,-1)^T$ right-hand sides yield nonzero solutions, of the form:

$$\mathbf{\Omega}^{ij}_{\tilde{\mathbf{U}}kl} = \mathbf{\Omega}^{ij}_{\tilde{\mathbf{V}}kl} = \pm\frac{1}{2}\frac{1}{\tilde{d}_k \pm \tilde{d}_l}. \tag{31}$$

In practice, since we're only interested in the Euler angles Jacobian $\mathbf{J}$ defined in Section 6.2, we don't need to compute the solutions from equation 30 (these values correspond to all-zeros columns in equation 23). Putting aside the equality case, the close form for the values $\mathbf{J} = (j_0, j_1, j_2, j_3, j_4, j_5)^T$ is:

$$\begin{cases}j_0 = &\frac{1}{2}\left(\frac{1}{\tilde{d}_2+\tilde{d}_1} + \frac{1}{\tilde{d}_2-\tilde{d}_1}\right)\\ j_1 = &\frac{1}{2}\left(-\frac{1}{\tilde{d}_2+\tilde{d}_1} + \frac{1}{\tilde{d}_2-\tilde{d}_1}\right)\\ j_2 = &\frac{1}{2}\left(-\frac{1}{\tilde{d}_2+\tilde{d}_0} - \frac{1}{\tilde{d}_2-\tilde{d}_0}\right)\\ j_3 = &\frac{1}{2}\left(\frac{1}{\tilde{d}_2+\tilde{d}_0} - \frac{1}{\tilde{d}_2-\tilde{d}_0}\right)\\ j_4 = &\frac{1}{2}\left(\frac{1}{\tilde{d}_1+\tilde{d}_0} + \frac{1}{\tilde{d}_1-\tilde{d}_0}\right)\\ j_5 = &\frac{1}{2}\left(-\frac{1}{\tilde{d}_1+\tilde{d}_0} + \frac{1}{\tilde{d}_1-\tilde{d}_0}\right).\end{cases} \tag{32}$$

Equation 31 informs us that in case of equal diagonal values, the fraction that would become undefined should be replaced by zero.