THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

délivré par
l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

---

présentée et soutenue par

Valentin ROUSSELLET

le 25 juin 2018

Implicit muscle models for interactive character skinning

École doctorale et spécialité :
MITT : Image, Information, Hypermédia
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5055)
Directeur de Thèse :
Loïc BARTHE, *Professeur à l'Université Toulouse III Paul Sabatier*

JURY

Raphaëlle CHAINE	*Professeur à l'Université Lyon I Claude Bernard*	Examinatrice
Paul G. KRY	*Professeur à l'Université McGill*	Rapporteur
Maud MARCHAL	*Maître de Conférences à l'INSA Rennes*	Examinatrice
Damien ROHMER	*Professeur à l'École Polytechnique*	Rapporteur
Mathias PAULIN	*Professeur à l'Université Toulouse III Paul Sabatier*	Examinateur

Implicit muscle models for interactive character skinning

Valentin Roussellet

Aux carabistouilles,
aux pièces de théâtre,
aux notes de musique,
aux éclats de rire.

Abstract

Surface deformation, or *skinning* is a crucial step in 3D character animation. Its role is to deform the surface representation of a character to be rendered in the succession of poses specified by an animator. The quality and plausiblity of the displayed results directly depends on the properties of the skinning method. However, speed and simplicity are also important criteria to enable their use in interactive editing sessions.

Current skinning methods can be divided in three categories. Geometric methods are fast and simple to use, but their results lack plausibility. Example-based approaches produce realistic results, yet they require a large database of examples while remaining tedious to edit. Finally, physical simulations can model the most complex dynamical phenomena, but at a very high computational cost, making their interactive use impractical.

The work presented in this thesis are based on, *Implicit Skinning* a corrective geometric approach using implicit surfaces to solve many issues of standard geometric skinning methods, while remaining fast enough for interactive use. The main contribution of this work is an animation model that adds anatomical plausibility to a character by representing muscle deformations and their interactions with other anatomical features, while benefiting from the advantages of Implicit Skinning. Muscles are represented by an extrusion surface along a central axis. These axes are driven by a simplified physics simulation method, introducing dynamic effects, such as jiggling. The muscle model guarantees volume conservation, a property of real-life muscles.

This model adds plausibility and dynamics lacking in state-of-the-art geometric methods at a moderate computational cost, which enables its interactive use. In addition, it offers intuitive shape control to animators, enabling them to match the results with their artistic vision.

Résumé

En animation de personnages 3D, la déformation de surface, ou *skinning*, est une étape cruciale. Son rôle est de déformer la représentation surfacique d'un personnage pour permettre son rendu dans une succession de poses spécifiées par un animateur. La plausibilité et la qualité visuelle du résultat dépendent directement de la méthode de skinning choisie. Sa rapidité d'exécution et sa simplicité d'utilisation sont également à prendre en compte pour rendre possible son usage interactif lors des sessions de production des artistes 3D.

Les différentes méthodes de *skinning* actuelles se divisent en trois catégories. Les méthodes géométriques sont rapides et simples d'utilisation, mais leur résultats manquent de plausibilité. Les approches s'appuyant sur des exemples produisent des résultats réalistes, elles nécessitent en revanche une base de données d'exemples volumineuse, et le contrôle de leur résultat est fastidieux. Enfin, les algorithmes de simulation physique sont capables de modéliser les phénomènes dynamiques les plus complexes au prix d'un temps de calcul souvent prohibitif pour une utilisation interactive.

Les travaux décrits dans cette thèse s'appuient sur *Implicit Skinning*, une méthode géométrique corrective utilisant une représentation implicite des surfaces, qui permet de résoudre de nombreux problèmes rencontrés avec les méthodes géométriques classiques, tout en gardant des performances permettant son usage interactif. La contribution principale de ces travaux est un modèle d'animation qui prend en compte les effets des muscles des personnages et de leur interactions avec d'autres éléments anatomiques, tout en bénéficiant des avantages apportés par *Implicit Skinning*. Les muscles sont représentés par une surface d'extrusion le long d'axes centraux. Les axes des muscles sont contrôlés par une méthode de simulation physique simplifiée. Cette représentation permet de modéliser les collisions des muscles entre eux et avec les os, d'introduire des effets dynamiques tels que rebonds et secousses, tout en garantissant la conservation du volume, afin de représenter le comportement réel des muscles.

Ce modèle produit des déformations plus plausibles et dynamiques que les méthodes géométriques de l'état de l'art, tout en conservant des performances suffisantes pour permettre son usage dans une session d'édition interactive. Elle offre de plus aux infographistes un contrôle intuitif sur la forme des muscles pour que les déformations obtenues se conforment à leur vision artistique.

Acknowledgements

A doctorate curriculum is often seen a solitary endeavour. In my case, this could not be further from the truth.

I wish to thank, first and foremost, my thesis supervisor Loïc Barthe, who has been very supportive, positive, and benevolent from the very beginning of our work together, through the ups and downs of research and publication. His thoughts, his scientific guidance, and his constant backing of my own ideas were invaluable in the production of this work.

I am also greatly indebted to Nicolas Mellado. Besides being a skilled young researcher, Nicolas has helped me countless times in my struggles with difficult maths, in debugging complex code, and in writing our papers, and I would not have gotten far without his help.

I am as much indebted to the other Storm team professors, David Vanderhaeghe and Matthias Paulin, who provided an excellent work environment, whose discussions were always useful and who encouraged me to invest my efforts for the team.

Working with my co-author and colleague Nadine Abu Rumman was very inspiring. Her skills, her great scientific outlook and her unfaltering motivation were instrumental in bringing our work to publication. In addition to his scientific contributions on which much of my work relies, I would also like to thank fellow team-mate Florian Canezin, who helped me out of technical difficulties on many occasions, and accepted to work with the massive amount of research code without (much) protesting.

Members of the research team have all been very supportive during my work: heartfelt thanks to Charly, whose code is so perfect it must never be fixed, to Rodolphe, who started it all, to Maurizio, who worked tirelessly on ibl, to Thomas and Céline, who welcomed me as an aspiring Ph.D. student, to Caroline, who took on the burden of the weekly seminars with me, to Thibault, who keeps it now, and to Anahid, who shared my fate from day one and never failed to entertain and support me.

The regular visits of Professor Brian "Blob" Wyvill in Toulouse always were a source of inspiration, never running out of great research insight and funny historical anecdotes. He was also very kind to correct the English writing of my thesis.

I am very grateful to Professor Ladislav Kavan, who welcomed me at the University of Utah. I spent a very fruitful and helpful summer in his team, and in addition to his great contribution to shape our common paper, he sharpened my interest in many areas of computer graphics. I also wish to thank Professor Daniel Sýkora, who gave me the idea of Discrete Fourier Transform for the sketch-based interface. In the Utah team, I am particularly indebted to Petr Kadleček, whose friendly welcome, and wise advice largely contributed to make this visit enjoyable, and to Dimitar Dinev, who offered me accommodation in Salt Lake City.

I am very grateful to the thesis committee who accepted to review my work, and in particular, I would like to thank the *rapporteurs*, Professor Paul Kry and Professor Damien Rohmer, for their helpful comments and remarks which helped me improve my manuscript.

This work would not have been possible without the funding provided by the Laboratoire d'Excellence CIMI, scholarship (ANR-11-LABX-0040) and the ANR FOLD-Dyn project (ANR-16-CE33-0015-01). I also wish to thank Professor François Faure and his company Anatoscope who gave us access to their anatomic model which was very helpful for our experiments.

♠

"Truly there would be reason to go mad, were it not for music": these words from Tchaikowsky express my feelings towards the amazing student orchestra of Toulouse. I joined OSET soon after starting my Ph.D. curriculum, and their musical talent and *joie de vivre* was one of the greatest source of happiness I found while in Toulouse.

I wish to thanks my flatmates, who put up with the hectic lifestyle of a grad student for a surprising long time, and Morgane, a dedicated Ph.D. student and great musician. I count myself lucky to be friends with Alif, and with the ever faithful Bronze Knights travel buddies, with whom many adventures were shared, and with Momo and our great conversations about science, the universe and everything. I owe many heartfelt thanks to Élise, for her unconditionnal love and support.

Finally, my whole family has always been a source of confidence and encouragement, through the good times and the bad. My most special thanks go to my father Jean-François, who always brought the best in me, and to Guillaume, my little brother, who I look up to in everything that matters.

Short contents

NOTATIONS

Common notations

Unless specified otherwise, the following conventions are adopted in this work:

Type	Notation	Examples
angle	lowercase Greek	α_i, γ, θ
scalar	lowercase italic	a, b, k, w_{ij}
point in affine space $\mathbb{R}^3$	lowercase bold	$\mathbf{p}$, $\mathbf{q}$, $\mathbf{m}$
vector in vector space $\mathbb{R}^3$	lowercase bold with arrow	$\vec{\mathbf{x}}$, $\overrightarrow{\mathbf{d_0}}$
matrix	capital bold	$\mathbf{M}$, $\mathbf{R}_\mathbf{x}(\theta)$
curve	capital cursive	$\mathscr{C}$, $\mathscr{P}(u)$

The following section summarizes the specific notations defined specifically in each chapter.

General geometry

$\mathbf{R}$ rotation matrix

$\hat{\mathbf{q}}$ dual-quaternion

$\vec{\mathbf{n}}(\mathbf{p})$ normal of a surface at point $\mathbf{p}$

Animation and skinning

$\mathbf{v}_i$ vertex i of a 3D mesh

$\mathbf{v'}_i$ updated position of $\mathbf{v}_i$

$\mathbf{v_{ref}}_i$ reference position of $\mathbf{v}_i$

$\mathcal{N}(\mathbf{v}_i)$ neighbourhood (set of adjacent vertices) of $\mathbf{v}_i$

$\mathcal{S}$ animation skeleton

$P(t)$ pose of the skeleton at time t

P_{ref} reference pose of the skeleton.

$\mathbf{M}_j$ model-space transform of joint j

$\overline{\mathbf{M}_j}$ parent-space transform of joint j

$\mathbf{M_{ref}}_j$ model-space transform of joint j in the reference pose

$\mathbf{B}_j$ model-space transform of joint j relative to the reference pose

$\overline{\mathbf{B}_j}$ parent-space transform of joint j relative to the reference pose

w_{ij} skinning weight of vertex i relatively to bone j

Scalar fields and implicit surfaces

C^n class of functions whose derivatives are continuous up to order n.

ϕ, ψ numerical functions ($\mathbb{R} \to \mathbb{R}$)

κ gradient-based operator controller function ($[0, \pi] \to [0, 1]$)

f, g 3D scalar fields ($\mathbb{R}^3 \to \mathbb{R}$)

$\mathbf{p}, \mathbf{q}$ point at which the function is evaluated

w 3D warping function ($\mathbb{R}^3 \to \mathbb{R}^3$)

∇f gradient of f

$\mathbf{J}_w$ Jacobian matrix of function w.

S_C C-iso surface of a scalar field ($f^{-1}(C)$)

Ω arbitrary 3D volumic object, subset of $\mathbb{R}^3$

$\partial\Omega$ the frontier surface of Ω

G composition operator

F composite 3D scalar field.

Implicit skinning

f_H HRBF field function

N_H number of HRBF points

f_j Compact-support field function representing mesh vertices associated to bone j

F_t Top-level skin field function at time t.

$\mathbf{v}_i$ Starting position of the i^{th} vertex

$\vec{\mathbf{h}}$ Displacement vector during the projection step

λ Projection step factor

e_i Tracked iso-surface of $\mathbf{v}_i$

$\mathbf{v}_i^{(\text{p})}$ Position of $\mathbf{v}_i$ after the projection step.

c_{ik} Cotangent weight between $\mathbf{v}_i$ and $\mathbf{v}_k$

$\mathbf{v}_i^{(\text{j})}$ Position of $\mathbf{v}_i$ after the j^{th} ARAP Jacobi iteration

$\mathbf{v}_i^{(\text{r})}$ Position of $\mathbf{v}_i$ after the ARAP relaxation step.

N_A Number of ARAP Jacobi steps.

$\mathcal{T}$ Tangent plane at a given vertex

N number of Implicit Skinning iterations

Implicit muscle model

f_M Muscle scalar field function

$\mathscr{C}$ Central polyline axis of the muscle

s Curvilinear coordinate along the central axis

$\mathbf{m}_0, \mathbf{m}_1$ End points (origin and insertion) of the muscle central axis

$\vec{\mathbf{n}}_{\mathbf{m}_0}, \vec{\mathbf{n}}_{\mathbf{m}_1}$ Normals orienting the respective end points of the central axis

$\mathbf{p}_i$ Intermediate control point of the central axis

$\vec{\mathbf{n}}_i$ Intermediate control normal of the central axis

$\mathbf{q}$ Point at which the muscle field function is evaluated

$\mathbf{h}$ Projection of $\mathbf{q}$ on the central axis

$\mathbf{h}_i$ Projection of $\mathbf{q}$ on the i^{th} segment of the polyline $\mathscr{C}$

θ Angle between the normal of the axis at $\mathbf{h}$ and the projection vector $\overrightarrow{\mathbf{qh}}$

Φ Part of the profile function depending on s

r Part of the profile function depending on θ (cross-section)

V Volume of the muscle

w Width scale factor of the profile function

l Length of polyline $\mathscr{C}$

e Eccentricity of the elliptic cross-section

u, v Semi-axis length of elliptic cross-section

a Activation level

α, β Shape parameters of the beta profile function

$\mathcal{L}^2$ Space of square-integrable functions.

Position-based dynamics

$C(\mathbf{p})$ Constraint function evaluated for the particles at position $\mathbf{p}$.

ρ_M Average density of muscle tissue

m Total mass of the muscle

m_i Mass of particle i

$\vec{\mathbf{v}}_i(t_n)$ Velocity of particle i at time step t_n

D_i Collision radius of particle i

k Stiffness of elastic distance constraint

d_0 Rest distance of elastic distance constraint

μ Velocity damping coefficient

η Friction coefficient

Introduction

Anatomical studies by Leonardo Da Vinci. © The Royal Collection.

Animated characters are our digital doubles in a virtual world. They are initially designed as 3D models: a surface mesh made of polygons. These meshes are then set in motion by computer artists in several steps: *rigging*, *animation* and *skinning*. Rigging creates a set of controllers that function as an intermediate representation for the animator to set the model of the character in different poses. A dedicated 3D artist, the *rigger*, creates this control layer or *rig*. This rig often takes the form of a *skeleton* articulated around joints, imitating the real life skeletal system, and its control parameters are the joints transforms. It is then up to the animators to manipulate these controllers to set the characters into motion, and imbue the characters with life, purpose, and meaning. Skinning is the function linking the rig parameters and the deformation of the character model. Thus, the properties of the skinning method have a great influence on the visual appearance of the animated character.

Virtual characters are only limited by the imaginations of their creators: they can be dragons, unicorns, or impossibly muscular super-heroes. They can fly, become invisible, or dodge bullets. Thus *realism*, the imitation of reality, is not the primary goal of skinning. The main purpose of skinning is rather *plausibility*: the visual result must *look* correct to our eyes, however unrealistic the characters or the situation. Additionally, animators must

be able to tune the attitude of a character much like a live-action actor would do. As a consequence, a good skinning method must be predictable and offer intuitive controls to artists, allowing them to align the resulting animation to their vision. Speed of evaluation is also a desirable property of skinning to speed up the production cycle: animators wish to see the effect of a change of rig parameters instantaneously, so they can adjust it interactively and converge quickly to the desired result. The speed requirement becomes a hard constraint in video games, where the animation is played in real-time.

Among skinning methods, the fastest and simplest are purely geometric: they directly apply the skeletal transforms to the vertices of the character mesh. These methods may be blazingly fast, but they often lack plausibility in complex animations. This shortfall in realism is essentially due to geometric methods only taking into account the surface of the character's skin, while the effects visible on the character are often a result of what happens underneath the skin.

The two other families of skinning methods attempt to fully replicate these effects to produce more realistic deformations. Example-based methods reproduce the deformations from a database of skin surfaces for a given character. Physics-based methods simulate the mechanics of the underlying tissues in a character's body to deform the skin. These methods produce quality results, but require a high investment to be useful: example databases require models be acquired or hand-crafted; physical models are notoriously difficult to configure and computationally intensive. This restricts their use to high-budget productions which can afford the time and manpower to use these complex methods.

There is a growing body of work dedicated to extending geometric skinning to replicate some effects of more complex methods, trading off a small computational cost for improvement on the plausibility and liveliness of the character animations, by adding volumetric effects to the surfacic representation of the mesh. Implicit surfaces are especially useful for this purpose, because of their properties to represent a surface along with the enclosed volume in a succinct mathematical form. A recent research direction, known as Implicit Skinning, has shown that using an implicit representation of the skin along with the standard polygonal mesh handles contact between body parts. This contact handling prevents mesh self-intersection on the resulting skin, an issue that has plagued geometric skinning methods for a long time.

I chose to extend this research by using implicit surfaces to model the volume occupied by muscles under the skin. Muscles make up a large portion of the body, and the effect of their action is particularly visible on the skin. In many cases, the precise shape of the skin in a limb is primarily dictated by the muscles underneath. Traditional media artists have since long studied anatomy in order to picture humans and animals more realistically, as illustrated by the anatomic studies of Leonardo Da Vinci at the opening of this chapter.

Because they set the body in motion, the appearance of muscles sets emphasis on the action of the character: for example, characters flexing their arms under efforts or taking a strong hit. Currently, most animators work with hand-crafted muscle deformers using ad-hoc models to create muscle deformations on the skin, which is a tedious process.

In this work, I present an animation method for representing muscles using implicit surfaces to model the volumetric effects they cause on the skin, producing high-quality skin deformations, including contact handling. This model takes several important phenomena contributing to the muscle deformations into account: volume conservation in muscle tissue, activation and isometric deformations. Integrating this model with a physics simulation produces dynamic effects such as jiggling motions and to detect and resolve muscle-muscle and muscle-bone collisions.

Overview of contributions

The main contributions detailed in this thesis are twofold. First, the **specification of an implicit muscle model**, giving a family of shapes able to represent most skeletal muscles in the human body and their deformation modes. These muscles are defined as curve-sweep surfaces, and dynamically driven by Position Based Dynamics, a fast approximate physical simulation method. Second, the **integration of this model with Implicit Skinning**, in order to benefit from its skin contact resolution on the final skinning solution. The implementation of this model offers a small set of intuitive parameters to control the shape of the muscles and their dynamic behaviour, avoiding the need for tedious sculpting of custom deformers. Setting the muscle parameters can be done interactively during the rigging process, and the total compute time of our method remains small enough for interactive editing.

This thesis is divided in two parts. The first part presents the work upon which the animation method is built. In Chapter 1, I survey the three different categories of skinning methods: geometric, example-based and physics-based. The next chapter (Chapter 2) is an introduction to the formalism of implicit surfaces, which will be used in the following chapters. Chapter 3 introduces *Implicit Skinning*, using implicit surfaces to resolve collisions in the skin of a character.

In the second part, I present a muscle model made of implicit surfaces which represent the muscles of an animated character. Chapter 4 presents the muscle model and discusses the rationale behind the choices in this approach. In Chapter 5, a physics simulation is added to the muscle model, enabling to represent dynamic phenomena. Finally, Chapter 6 describes how to implement this model within a skinning pipeline using *Implicit Skinning* and discusses the practical implications of such an implementation.

I

Skinning with implicit surfaces

1 Character animation and skinning

And all the carnall beauty of my wife,
Is but skin-deep, but to two senses known
— Thomas Overbury (1581 – 1613), *A Wife*

This chapter presents *skinning* as the last stage of the character animation process, following the sculpting of a character model, or *modelling*, and the definition of animation controls, or *rigging*. After the elementary concepts and notations are exposed in Section 1.1, the subsequent sections review the different families of skinning methods: physically-based (Section 1.2), example-based (Section 1.3), and geometry-based (Section 1.4).

1.1 The skeletal animation pipeline

1.1.1 Models, rigs, and skeletons

Animated objects and characters are usually represented as *polygon meshes*, which are fast to display with current rendering hardware. Animating these objects thus requires to deform the mesh to the desired shape at each frame.

Describing the movement of the millions of vertices of a modern display mesh cannot be expected to be done manually, hence the idea of *rigging*: setting a mesh with a set of simple spatial transform parameters. Animators only manipulate these high-level parameters, whose evolution in time guides the mesh vertices to their desired position. A natural idea for rigging a character is to use an *animation skeleton*, which was initially proposed by Magnenat-Thalmann et al. [MLT88]. The skeleton is defined as a hierarchic kinematic chain of *joints*, as shown in Figure 1.1.

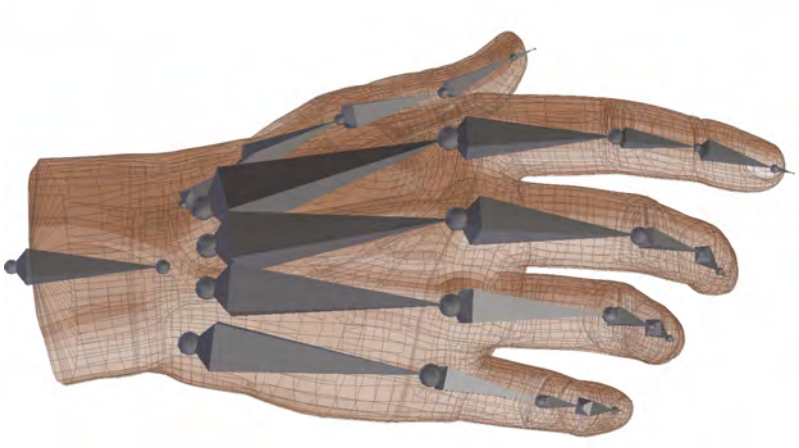


Figure 1.1: A model of a hand with its skeleton rig.

Definition. An animation skeleton $\mathcal{S}$ is given by a *tree* $\mathcal{G}_\mathcal{S}$ of m joint nodes, the root being conventionally the first node; and a set of rigid transforms $\{\mathbf{M}_j\}_{j=0..m-1}$ giving the local frame of each joint.

In this model, *bones* are nominally associated with the edges of the graph, linking two joints: the *proximal joint* being the closest from the root and the *distal joint* the farthest. For humanoid characters, the root joint is generally located on the spine, near the centre of gravity.

The transforms at joints $\mathbf{M}_j$ are the key parameters of the skeleton rig. They are usually set by manipulating the bones in an animation tool, hence the very common abuse of terminology where a bone is associated to its proximal joint. Thus expressions such as "bone transform" must be understood as referring to the transform of the rigid bones applied at the proximal joint.

Each joint transform defines a local space, in which the bone usually occupies one of the axis. When manipulating bones to setup a skeleton in an application, it is much more intuitive to modify the *parent-space* transform of a joint, defined relatively to its parent joint in the tree, rather than the absolute transform. This way, when a bone is moved, all the subsequent bones in the chain follow the movement: for example, raising the arm by modifying the shoulder joint's transform will also raise the forearm and hand bones.

Definition. The parent-space transform of joint j, whose parent joint is indexed by p is

$$\overline{\mathbf{M}}_j = \mathbf{M}_p^{-1}\mathbf{M}_j \ .$$

For the root joint, the parent-space transform of the root joint is the same its model transform: $\mathbf{M}_0 = \overline{\mathbf{M}}_0$.

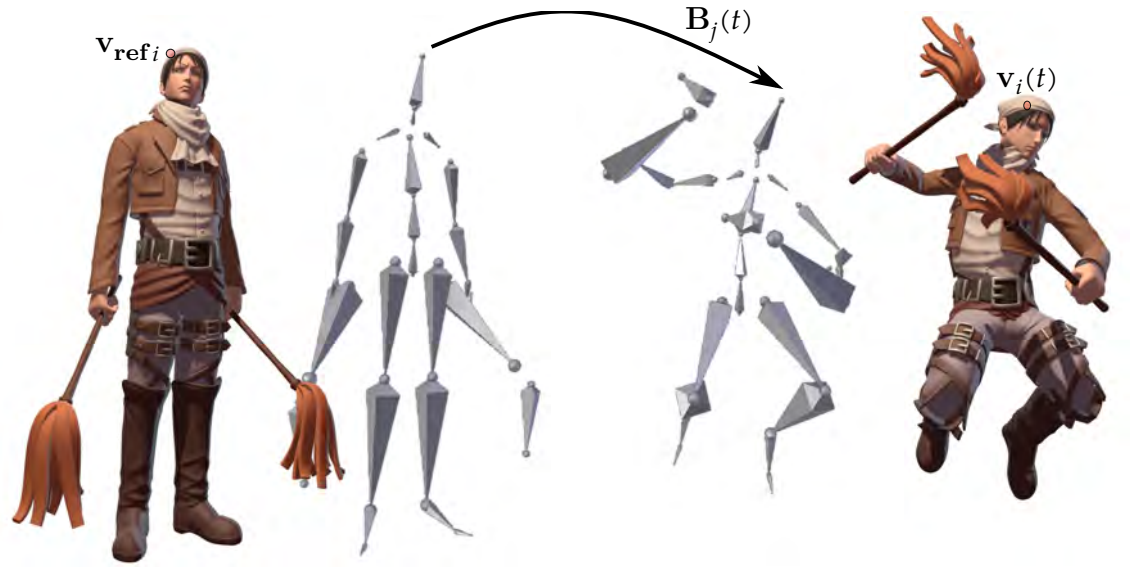


Figure 1.2: Animation and skinning of a character. Left: the initial model, represented by a polygon mesh, is rigged with a skeleton in its *reference pose*. Right: the animator moves the skeleton to the desired pose. A *skinning* algorithm moves each vertex of the initial mesh $\mathbf{v}_{\mathbf{ref}\,i}$ to its new position $\mathbf{v}_i$.

The animation skeleton is a very useful tool for character animation setup because of its resemblance, albeit imperfect, with its real-life counterpart. Many other rigging systems have been devised for specific usages. Simply using independent *points* is especially useful for simple objects or 2D animations. Using points linked in a coarse lattice which envelops the object to deform is another fruitful approach introduced by SEDERBERG and PARRY [SP86]: *cage-based deformation*. NIETO and SUSÍN [NS13] provide a recent survey of these methods. This thesis focuses on skeletal-based skinning which is still the ubiquitous rigging method for character animation.

1.1.2 Animating a rigged character

Displaying an animation is then done in two logically separate steps: an *animation system* is responsible for providing the rig parameters at each time step t, and then an algorithm deforms the mesh given these parameters. In the case of an animated skeleton, the rig parameters are the joint transforms $\mathbf{M}_j$.

Definition. Given an animation skeleton $\mathcal{S}$, an animation function associates a set of rigid transforms $P(t) = \{\mathbf{M}_j(t)\}_{j=0\ldots m-1}$ to each time step t. The set $P(t)$ is called the *pose* of the skeleton at time t. Transforms $\mathbf{M}_j$ represent the position and orientation of the joints in world space.

Usually, the skeleton is set up by an artist on the undeformed mesh. This initial pose of the skeleton, associated with the initial state of the mesh, is known as the *reference* or *rest pose*: $P_{\text{ref}} = \{\mathbf{M}_{\mathbf{ref}_j}\}_{j=0..m-1}$. A deformation of the mesh vertices is usually expressed more easily with transforms from the reference pose to the current pose, leading to the following notation.

Definition. The *relative bone transform* $\mathbf{B}_j(t)$ of bone j at time t is the transform of joint j relatively to the reference pose, given by:

$$\mathbf{B}_j(t) = (\mathbf{M}_{\mathbf{ref}_j})^{-1}\mathbf{M}_j\,.$$

Generating the right succession of poses to express motion is both an artistic field and an entire research area. The animations can be created by an animator (and usually stored as a set of *key poses* through which the others are interpolated), acquired from an actor equipped with a *motion capture* device, or even synthesized procedurally with machine learning methods [HSK16].

Once the rig parameters are determined for a given time t, the goal is to deform the display model of the character according to these parameters. As the display model usually represents the surface of the character (or skin), this step is naturally called *skinning*, as illustrated by Figure 1.2.

Definition. Given a set of N vertices $\{\mathbf{v}_{\mathbf{ref}_i}\}$ representing the object's surface in the reference pose $\{\mathbf{M}_{\mathbf{ref}_j}\}$, a *skinning function* gives for each pose $P(t) = \{\mathbf{M}_j(t)\}_{j=0...m-1}$, the new position of each vertex $\mathbf{v}_i(t)$.

Skinning is a vastly under-constrained problem. A typical animation skeleton has about 30 to 40 joints, each with 3 to 6 degrees of freedom, while a character mesh usually has from tens of thousands to millions of vertices.

Moreover, it is hard to quantify what constitutes a good deformation. We are used to see human characters evolve in real-life, and thus can quickly tell when a computer-generated character does not look right. Yet, finding which geometric properties of the resulting mesh must be constrained for the result to look correct is challenging even for simple cases.

1.1.3 Primary and secondary motion

Animators often stress the difference between *primary* and *secondary motion*, exposed as a major principle of animation by Disney animators THOMAS and JOHNSTON [TJ95, pp. 64–65]. The limits between what constitutes primary and secondary motion is somewhat

ill-defined. In principle, primary motion is the direct result of the controlled action of the character and makes up the bulk of the character's appearance at a given pose. Secondary motion refers to all movements and effects which appear as a reaction to the character's primary motion. This includes many dynamic effects such as body parts jiggling when a person is moving, muscles contracting under effort, strands of hair or cloth flying in the wind, etc.

While primary motion conveys the action of the character, secondary motion adds more life and more dimension to the character's animation, significantly increasing the realism of an animated scene. Movements with secondary motion seem more natural, and artists often use it to emphasise the character's action and create more expressive scenes.

In skeletal animation, secondary motion is often understood as motion on the skin that is not the result of the direct manipulation of the joints. This usually requires to increase the complexity of the rig by adding an extra layer between the rig parameters and the final effect on the skin. The high number of degrees of freedom in these effects adds to the difficulty of animating secondary motion. Methods that generate secondary motion are often computationally expensive and expose many unintuitive parameters which require hand-tuning. These shortcomings burden the animators with complexity and only high-budget animation productions can afford to use them. In this work, we focus on secondary motion caused by the anatomy of a character, and most prominently on the effect of muscles on its visual aspect.

The next sections provide an overview of the different families of skinning algorithms. Recent reviews on the field include the tutorial by JACOBSON et al. [Jac+14], which provides a good introduction, and the recent survey by ABU RUMMAN and FRATARCANGELI [AF16] for a complete coverage of skinning techniques. In addition, a specific review dedicated to muscle simulations in computer graphics can be found in the survey by LEE et al. [Lee+12].

Methods able to reproduce secondary motion effects on the skin can be classified in three categories. First, *physics-based* methods (Section 1.2) use mechanical simulation of deformable bodies to compute the motion of the different parts of the character's anatomy (muscles, bones, soft tissues and skin). In contrast, *example-based* methods (Section 1.3) seek to reproduce the deformations from example data, either crafted by artists or acquired from the real world. Finally, *geometric* skinning (Section 1.4) directly uses the skeleton transforms to compute the final positions of the vertices, while secondary motion is generally added post-hoc with specific deformers.

Figure 1.3: Early mesh-based muscle simulation.

(Picture from [NT98])

1.2 Physically-based skinning

Deformations appearing on the skin are the result of the complex biomechanical processes happening in a moving character. To reproduce these results in computer graphics, it is possible to see skinning as the result of a biomechanical simulation of a virtual human body. Physical simulation methods were introduced early in computer graphics [SDN84; GM85; CHP89] and have since taken an important role in realistic animation methods. They offer representations for complex dynamic deformations such as collision resolution, bulging and jiggling of soft tissues.

The following section gives an overview of the main families of physics-based approaches. Detailed reviews of the field include Nealen et al.'s [Nea+06] and the survey by Abu Rumman and Fratarcangeli [AF16, Section 4].

1.2.1 Force-based muscle models

Methods used in biomechanical research usually focus on computing the forces exerted by the muscles. To that effect, muscle models are often reduced to a *line of action* through which the forces are transmitted. A model of the physical behaviour of the human musculature for computer simulation was presented by Zajac [Zaj89], modelling the muscle as an actuator and describing mechanical properties of muscle tissues and tendons.

However, a simulation of muscles for computer graphics must also take into account the shape of each muscle and their volume. CHEN and ZELTZER [CZ92] introduced the simulation of forces through a finite-element method (FEM) on a mesh representation of the muscle. At the time, this method was too slow for interactive visualisation of the results, which led THALMANN et al. [TSC96] and NEDEL and THALMANN [NT98] to introduce a model based on a mass-spring system, used over a simplified mesh of the muscle. Their method, illustrated on Figure 1.3, was fast enough for the simulation of one muscle in real-time.

Several approaches attempted to tackle muscle simulation more efficiently. TERAN et al. [Ter+03; Ter+05] used finite-volume methods to compute the stress tensor inside the muscles, coupled with a spline-based fibre model, while work by PAI et al. [PSW05] attempts to blend the line-based methods used in biomechanical research and volumetric approaches by using a *strand* representation. Their work was extended by SUEDA et al. [SKP08; Sue+11] specifically for skin deformations of the hand.

1.2.2 Simulation of anatomic models

With the growing of computing power, new methods were developed to model not only muscles, but also other soft tissues and skin to represent a fully simulated human body. The OpenSIM software [Del+07] or the full upper body model of LEE et al. [LST09] are examples of methods striving for a general simulation model of the human body. The most recent musculoskeletal models now include complex multibody rigid dynamics for the skeleton coupled with shape-varying muscles whose motion takes into account inertia and mass transfer [Mur+14; HHP15].

A key advantage of physically-based anatomic methods is their ability to model skin elasticity and contact [McA+11]. The ability to handle self-contact of the skin, a situation that happens often in animation (for example, with strong bending of the elbow and knee) is especially difficult to model accurately with other methods. FAN et al. [FLP14] presented an Eulerian-on-Lagrangian approach to model musculoskeletal systems, including muscle and bone contact. This work was later extended by SACHDEVA et al. [Sac+15] to take tendons into account.

These approaches however require intensive modelling work. A volumetric map of the initial position of muscles, bones, tendons, fat tissues and skin is generally required as the input to the simulation. Propositions to tackle this problem include the transfer from an existing anatomic model [Ali+13] or the simulation of muscle growth [SZK15]. The most recent work by KADLEČEK et al. [Kad+16] (Figure 1.4) proposes to adapt an anatomy

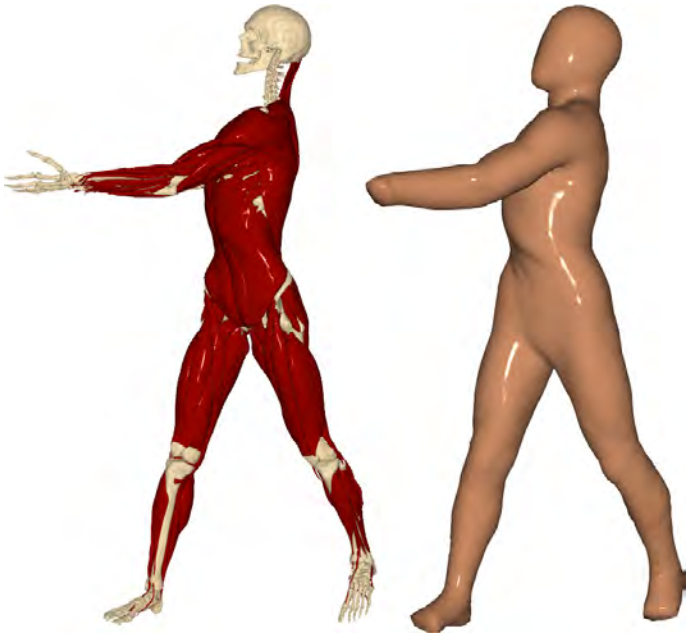


Figure 1.4: A physics-based anatomic template with volumetric muscles and fat tissue.

(Picture from [Kad+16])

template to user-supplied body scan, bridging the gap between physical simulation and data-driven methods (Section 1.3).

1.2.3 Simulation space reduction

An interesting venue to reduce the high computational cost of a full physical simulation on a detailed model is to try to restrict the state-space to a smaller dimensional space, based on prior knowledge of the result. Since only the character's skin surface is visible in an animation, it makes sense to focus on the skin deformation only, e.g. by using thin-shell models [Li+13] or by reproducing the effect of corotational elasticity on the surface [GMS14].

Another animation-specific space reduction was presented by HAHN et al. [Hah+12]. Their method restricts the simulation's state-space to the subspace of deformations allowed by the animation rig. This method has the extra advantages of fitting in the standard animation pipeline and enabling the animators to exert control over the type of deformation, and edit post-hoc the resulting animated rig parameters.

Recently, XU and BARBIČ [XB16] obtained real-time frame rates running a finite-element method in the pose-space, i.e. the space defined by interpolation between example poses (see Section 1.3.1).

1.2.4 Simulation control and coupling

A key issue in physically-based animation and skinning is to interface the force-based physics simulation with the kinematic input of the skeleton. In real life, motion is transmitted from the muscles to the bones, but in character animation, the animation skeleton bones are directing the character's motion, and thus drive the position of the muscles. The works of Capell et al. [Cap+02; Cap+05] showed how to generate forces from the animation skeleton as inputs to a finite-elements methods. Nevertheless, these methods may violate conservation of momentum and thus become unstable for large time steps.

Shinar et al. [SSF08] exposed a two-way coupling model between soft and rigid bodies. In this model, the deformable bodies could exert force on the rigid bones. Two-way simulation has also been used to represent effects of external bodies on the characters [KP11; Liu+13], e.g. throwing a ball at a character.

1.2.5 Discussion

Physical simulation methods use the most detailed and realistic models of a character to replicate the complex phenomena that govern the aspect of a body in motion. Unfortunately, this complexity comes with a cost. FEM simulations are very computationally expensive, and the high resolution necessary to reproduce the finest details of human anatomy makes a full-fledged simulation very slow. State-of-the-art methods take from 3 to 4 seconds per frame for simple models such as the arm of Fan et al. [FLP14] (4 muscles and 3 bones); complete anatomical models such as the one presented by Kadleček et al. [Kad+16] (60 bones and more than 100 muscles) take about 30 seconds to a minute to compute one frame of animation.

In addition, a soft-body simulation exposes many parameters: material constants such as stiffness, Lamé constants or Young's modulus, as well as simulation parameters: integration time step, solving method, etc. These parameters must be chosen carefully to ensure a plausible result and remain within some boundaries, lest the simulation becomes unstable.

The nature of a physical simulation also makes it hard to predict beforehand what the result will look like at a given point in time without running the simulation itself. As a consequence, animating a character through physical simulation is a long and tedious process. Animators must be familiar with the technicalities of elastic material models and their properties, and often proceed by trial-and-error to set the parameters that achieve the results that match their artistic direction. The time and manpower required for using such methods thus restricts their practical use to high-budget productions, and forbids their use in real-time applications.

Figure 1.5: Typical blendshape expressions. From left to right: half-smile, full smile and open-mouth expression.

(Picture from [Lew+14])

1.3 Data-driven skinning

While physical simulation attempts to model the body in order to simulate its motion, data-driven approaches attempt to imitate the results obtained by a set of example deformations. They provide an easy way to generate consistent solutions for a problem with such a large state-space such as skinning, but without the tedious work of finding the correct parameters of a physical simulation. Mukai [Muk16] presents a recent survey of example-based methods.

1.3.1 Pose-space deformation

Introduced by Lewis et al. [LCF00], *pose-space deformation* (PSD) has the artist manually position the skeleton in a set of key poses, and sculpt the mesh in each of these poses, usually as a correction over a direct geometric skinning method (see Section 1.4). Subsequently, during animation, the current pose is expressed in terms of the example poses and the shape of the mesh is interpolated from the associated sculpted shapes.

The pose-space lends itself naturally to space reduction using principal component analysis [KJP02] which helps speed up the computations. This method was later extended by Kurihara and Miyata [KM04] and Rhee et al. [RLN06] to support weights, reducing the number of example poses. Weighted pose-space deformation (WPSD) can then handle a sparser pose-space but at the cost of an increased computational complexity.

Pose-space deformation is especially popular in facial animation [LH09] where it is generally known as *blendshapes*. Key poses of the face are often mapped to different emotions (such as fear, excitement, happiness or surprise, as illustrated by Figure 1.5) enabling the artists to generate compelling faces showing transitions between emotions [Lew+14].

Example-based deformation algorithms critically depend on the interpolation method used to reconstruct the blended mesh deformation from the key shapes. Covering a high-dimensional space such as the pose space by hand-tuned mesh shapes would be intractable, so the interpolation methods must behave well with scattered data points [see Jac+14, section III.4].

1.3.2 Pose and shape capture

In the last decades, 3D data acquisition methods from real-world objects and persons became increasingly available. As a consequence, subsequent research investigated capturing and exploiting shapes acquired from real world data instead of relying on an artist sculpting the example poses. The earliest work by ALLEN et al. [ACP02] directly computes interpolations between skin shapes captured from a range scanner.

Later work benefited from improved acquisition methods: medical imagery, [KM04], motion capture systems [PH06; PH08], depth sensors [Cas+16] and even smartphone-based capture [IBP15]. In particular, the work of NEUMANN et al. [Neu+13] and LOPER et al. [LMB14] showed that it was possible to capture the soft tissue deformations from standard motion capture markers.

1.3.3 Statistical shape models

To support the acquisition of several animations of the same person, or even of persons of different body type led to the development of models able to separate deformations caused by the change of pose from deformation due to varying body types, e.g. characters that vary in gender, size, body weight or musculature. The SCAPE model by ANGUELOV et al. [Ang+05] can represent shape variations caused by both position and body type as a function of the pose. This amounts to defining a function S such as each vertex $\mathbf{v}_i$ of the body's surface can be written as

$$\mathbf{v}_i = S(\beta, P(t), \mathbf{v}_{\mathbf{ref}\,i}) \ ,$$

where $P(t)$ is the current pose, β is the *shape parameter* representing the body type variation of the character relatively to a default *mean shape*, and $\mathbf{v}_{\mathbf{ref}\,i}$ is the vertex position in the reference pose.

Processing scans from different capture sessions with different people suffers from issues associated with captured 3D data: noise and registration. Efforts to mitigate this problem led ALLEN et al. [All+06] to use statistical models of correlation between shapes to account for missing or noisy scan data, and WEBER et al. [Web+07] to use harmonic

Figure 1.6: Data-driven soft tissue model showing physical parameters learned from capture data and extrapolated to new poses and external forces.

(Picture from [Kim+17])

interpolation to increase the number of joints influencing a vertex. Later developments include the co-registration method BlendSCAPE [Hir+12] which incrementally registers various input meshes together while learning a deformation model.

Later development in statistical shape models include SMPL [Lop+15] which uses a wide database of scanned shapes to separate body type and pose deformation directly as a linear function of the pose P, helping its integration into a production skinning pipeline.

1.3.4 Learning dynamics

As the amount of available scanned data increased, it became possible to use machine learning algorithms on body shape deformations to fit the parameters of a physical simulation from learned data.

The Dyna model by PONS-MOLL et al. [Pon+15] was the first to present a statistical model of dynamics on the surface of the skin based on example data. LOPER et al. [Lop+15] used a similar approach with SMPL to learn an additional dynamic deformation term on top of their statistical body type and shape deformer.

The most recent work by KIM et al. [Kim+17] goes beyond the surfacic representation and sucessfully fits a full volumetric model of a human body with the help of a large database of registered skin deformations (Figure 1.6).

1.3.5 Discussion

Example-based skinning provides high-level context for surface deformations. While high realism can be achieved by using scanned data, the final result is conditionned by the availability of specific poses in the database or the capacity to record new key poses.

(**a**) Rigid skinning (**b**) LBS (**c**) DQS

Figure 1.7: Geometric skinning methods

Cases often arise when scanned data is not desirable or possible: for unrealistic characters such as monsters, or impossible poses (e.g. exaggerated fighting stances). It is then up to the artists to generate the key poses and then hand-tune the weights until the results of the deformation are good in all poses of the animation, which is a tedious process. In particular, even with high quality example data, avoiding self-intersection of the deformed mesh in every pose is not guaranteed unless they are separately resolved using collision detection.

Lastly, data-based approaches can be problematic for interactive applications (such as video games) which usually require to display a large number of different characters (sometimes appearing only in a few scenes). The memory consumption of the pose database, requiring to store a set of meshes per character, can quickly become prohibitive.

1.4 Geometric skinning

On the other side of the realism versus complexity trade-off, geometric methods are the most straightforward because they rely only on the kinematic input, provided by the skeleton, to compute the new vertices positions.

These approaches are very popular for interactive applications as their simplicity makes them very efficient. They remain the standard skinning method in most production applications, from feature film animation to video games [AHH08, pp. 80–85].

1.4.1 Linear blend skinning and skinning weights

One of the simplest skinning method which can be considered would associate each vertex $\mathbf{v}_i$ of the mesh to one of the joints j and move it according to this joint's transform only:

$$\mathbf{v}_i(t) = \mathbf{B}_j(t)\mathbf{v}_{\mathbf{ref}\,i}\,.$$

The results of this method (sometimes called *rigid skinning*, as the vertices are rigidly transformed) are of very poor quality. Yet, most vertices which lie in the middle sections of the limbs are correctly placed, and the problematic areas are located around the joints, as illustrated by Figure 1.7(a).

It is clear that vertices around a joint should be influenced by more than just one bone. Each vertex $\mathbf{v}_i$ is thus given a set of *skinning weights* $\mathbf{w}_i = \{w_{ij}\}_{j=0..m-1}$ which are used to combine transforms of multiple bones. Each vertex has partition of unity weights ($\sum_j w_{ij} = 1$). Vertices in the middle of the limbs move with only one bone (thus have one weight equal to 1 and the others to 0); while vertices around joints smoothly transition between two bones (and sometimes more for complex joints). The vertex position $\mathbf{v}_i(t)$ is simply expressed as the linear combination of the transforms of each bone weighted by its weight, applied to its reference position $\mathbf{v}_{\mathbf{ref}\,i}$:

$$\mathbf{v}_i(t) = \sum_{j=0}^{m-1} w_{ij}\mathbf{B}_j(t)\mathbf{v}_{\mathbf{ref}\,i}\,.$$

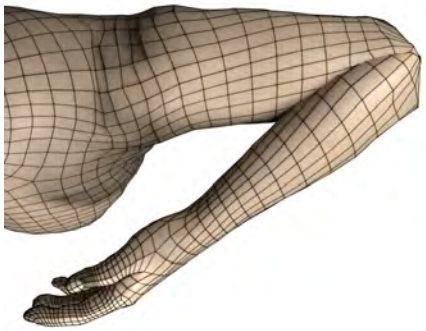
This method was introduced by MAGNENAT-THALMANN et al. [MLT88] and is now known as *linear blend skinning* or LBS [1].

This method is easy to implement on current graphics hardware, and has therefore proven to be very efficient and successful. Even now, it is the *de facto* standard for real-time skinning. Nonetheless, it suffers from major drawbacks.

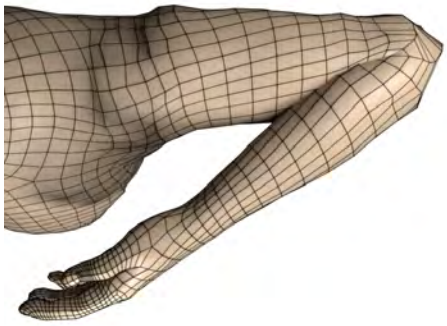
Firstly, while the result is visually plausible for small deformations, the quality of the deformed mesh degrades quickly when the joints bend or twist too widely. These visual artefacts are frequent enough to have earned their own nickname in the animation community: a joint bent too far will make the surface thinner on the exterior of the bend, leading to the *collapsing elbow* artefact (Figure 1.8(a)) while a twist of 180° will project all nearby points towards the center of the joint causing the dreaded *candy-wrapper* artefact (Figure 1.8(c)).

Secondly, weight-based methods such as LBS also rely on user input, as the animator usually provides the weights by painting on the mesh, as depicted in Figure 1.9. Getting

[1] In fact, this skinning technique has been used many times and under many different names such as *stitching* or *vertex blending* [Woo00] , *skeletal subspace deformation* or *enveloping* [LCF00], etc.



(**a**) LBS of bent elbow



(**b**) DQS of bent elbow

(**c**) LBS of twisted elbow

(**d**) DQS of twisted elbow

Figure 1.8: Typical artefacts of geometric skinning methods. Notice the loss of volume with LBS when bending (a), and the bulging with DQS (b). Figure (c) shows the *candy-wrapper* artefact with LBS and its mitigation by DQS (d).

the weights right for a given animation usually takes time and practice, and relies a lot on trial-and-error. Jacobson et al. [Jac+11] provided a method to automatically compute weights from a given mesh with its animation skeleton, and a set of target poses. Yet subsequent research by Kavan and Sorkine [KS12] showed that merely optimizing the weights is not enough to produce good deformations in the most complex cases and that non-linear deformers are required.

1.4.2 Dual quaternion skinning

Artefacts such as the candy-wrapper or the collapsing elbow appear as LBS tries to apply a linear combination in the space of rigid transforms, whose topology is spherical. These issues can be avoided by combining the transforms using non-linear blending. Several non-linear approaches were subsequently investigated, such as *log-matrix skinning* [Ale02] or *spherical blend skinning* [KŽ05]. The most successful non-linear method was proposed by Kavan et al. [Kav+07], who used the space of *unit dual quaternions* to represent rigid transforms, similarly to how the space of unit quaternions can represent rotations [Ham44].

Figure 1.9: Weight painting in Blender, showing the influence of the arm bone over the jacket vertices from $w_{ij} = 1$ in red (maximal influence) to $w_{ij} = 0$ in blue (no influence).

Dual quaternion skinning (DQS) uses the same input as linear blend skinning but converts the transforms $\mathbf{B}_j$ to their unit dual quaternions representation $\hat{\mathbf{q}}_j$ and computes the blended transform in the dual-quaternion space by linear combination and normalization.

$$\hat{\mathbf{q}}_i(t) = \frac{\sum_{j=0}^{m-1} w_{ij}\hat{\mathbf{q}}_j(t)}{\left\|\sum_{j=0}^{m-1} w_{ij}\hat{\mathbf{q}}_j(t)\right\|}$$

The resulting unit dual quaternion $\hat{\mathbf{q}}_i$ represents a rigid transform $\mathbf{T}_{\hat{\mathbf{q}}_i}$ which is then applied to the vertex.

$$\mathbf{v}_i(t) = \mathbf{T}_{\hat{\mathbf{q}}_i}(t)\mathbf{v}_{\mathbf{ref}\,i}$$

The topological properties of the space of dual quaternions avoid the candy wrapper and elbow collapse artefacts (Figures 1.8(b) and 1.8(d)), and the performance of a typical implementation of DQS can be almost as fast as LBS, which made dual-quaternion skinning increasingly popular in real-time applications.

However, DQS is not exempt of pitfalls. First, care must be taken when implementing dual quaternion skinning, especially to avoid antipodality problems when composing the dual quaternions [see Kav+08, section 4].

Second, DQS introduces an unwanted bulge around bent joints, because vertices moving around a joint are constrained to share a common centre of rotation located exactly at that

joint. This has led Kim and Han [KH14] to introduce a correction technique to mitigate this effect.

A more recent work by Le and Hodgins [LH16] systematically removes these artefacts by computing the optimal centre of rotation for each vertex. For each vertex $\mathbf{v}_{\mathbf{ref}\,i}$ of the mesh in reference pose, an optimal rotation centre $\mathbf{p}_i^*$ is pre-computed by averaging all vertices of the original meshes over the whole surface, weighted by a weight similarity function. During skinning, the vertex's rotation is computed by spherical linear interpolation around the transformed centre of rotation. This method avoids most of the artefacts of other geometric skinning methods, and only requires an expensive precomputation step for the centre of rotations $\mathbf{p}_i^*$.

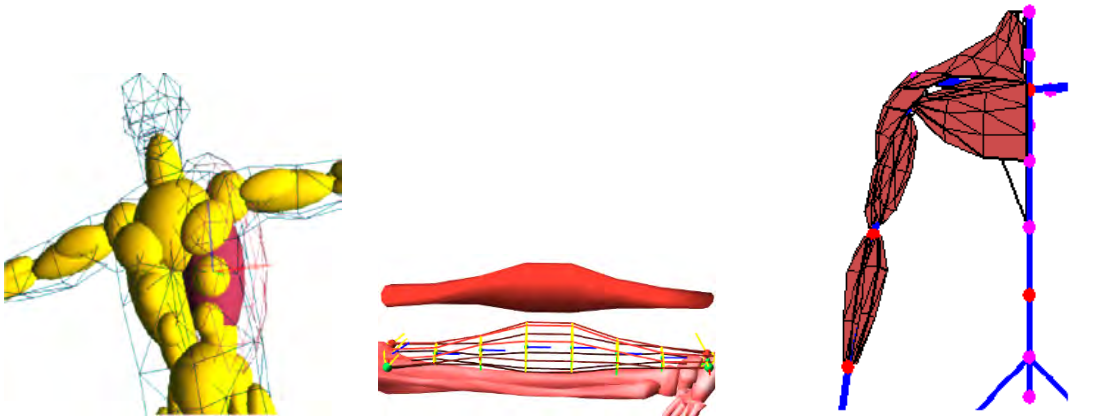
1.4.3 Improving geometric skinning

The case for improved geometric skinning arises as the simplistic nature of these methods exposes their weakness. The basic deformations generated by geometric methods cannot hope to reproduce the richness of the complex phenomena of skin deformation. Characters often seem lifeless or made out of plastic, and body parts are self-intersecting as soon as joints bend at sharp angles.

A natural extension to these standard methods is to increase the number of weights per-vertex. Instead of one weight per transform as in LBS or DQS, one could take several weights, one per principal direction of the transform. This was the approach described by Wang and Phillips [WP02], which uses twelve weights per vertex-bone pair. Setting so many weights manually by painting becomes unfeasible. Their method proposes instead to optimize them from example poses, similarly to pose-space deformation methods (see Section 1.3.1).

Principal component analysis can also be used to reduce the number of weights [AM00]. Merry et al. [MMG06b; MMG06a] proposed to use four weights per pair by solving for rotational invariants (mimicking an elastic deformation), while Jacobson and Sorkine [JS11] used two weights, one for bending and one for twisting.

In production settings, animators often increase the range of possible deformations by adding extra joints who do not necessarily correspond to real-life joints. This is often done by hand on a case-by-case basis, but several efforts were made to generalize this approach. For example, Mohr and Gleicher [MG03] systematically adds several joints to each existing bone. Wang et al. [WPP07] proposed to learn the position of extra joints from example poses, and to set the weights accordingly, and Kavan et al. [KCO09] to approximate the non-linear deformations with linear skinning using only the input animation as a prior. While at runtime, the skinning computation remains a linear weighted sum of transforms,



(**a**) Ellipsoids (**b**) Generalized cylinders (**c**) Skeleton implicit surfaces

Figure 1.10: Examples of geometric muscle deformers

(Pictures from [LAG01; WV97; Min+00])

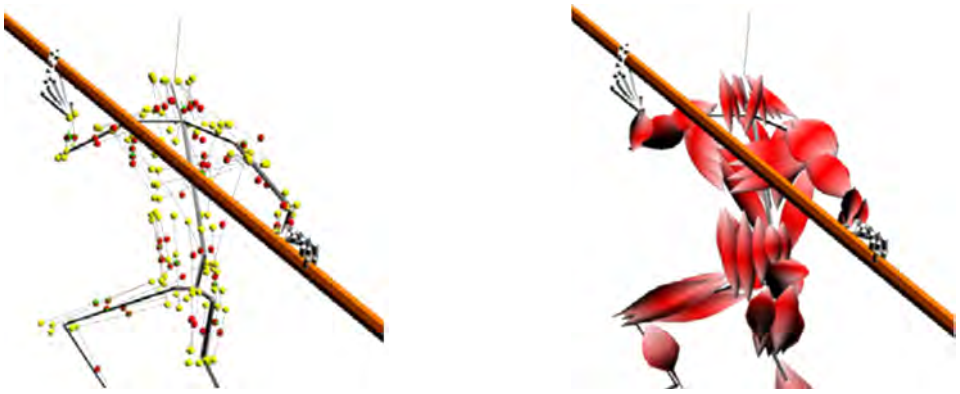
these approaches will suffer from the same drawbacks as example-based methods, i.e. the necessity to obtain sculpted or scanned data beforehand.

1.4.4 Shape-based muscle deformers

Muscle deformations are especially prominent in skinning, and the ability to reproduce muscle bulging is often the principal type of secondary deformation that many methods seek to reproduce [LCF00; SRC01; KJP02; MG03; WPP07; MK16]. To avoid the need for extra weights and extra bones and offer a more intuitive interface to the animator, a fruitful approach is to add muscle-specific deformers to joint-centred geometric skinning.

An important property to consider when modelling muscle shapes is *volume preservation*. Muscles generally combine two types of deformation: *isotonic* and *isometric*. Isotonic deformation happens when the limbs move and the muscles endpoints (*origin* and *insertion*) are drawn closer together or pushed farther apart, while isometric deformation is caused by the *activation* of the muscle (the increase of tension in the muscle fibres) without motion of the endpoints. In real life, the conservation of volume is what makes muscles bulge when contracted or activated [Kar90]. It is therefore crucial for the shapes representing the muscles to deform at constant volume and to be able to reproduce these two modes of deformation.

WILHELMS [Wil94] modeled muscles, bones and soft tissues with ellipsoids, including deformable muscles which maintain their volume when contracting, using the analytic formula for the volume of the ellipsoid. Her method works by directly linking the mesh vertices to the nearest ellipsoid primitive with a mass-spring model which pulls the



(**a**) Control points (**b**) Muscle shapes

Figure 1.11: A dynamic muscle shape model based on Bézier curves. Figure (a) shows the two end points (yellow) and a middle point (red) defining the curves. Figure (b): the width of the muscle is sinusoidal, yielding deformable fusoid shapes.

(Pictures from [LA07])

vertices when they move too far from their primitive. LECLERCQ et al. [LAG01] described a similar approach with ellipsoid primitives to model muscles (Figure 1.10(a)). They used a ray-marching method to project the vertices of the mesh to the desired distance of the ellipsoids.

Subsequent methods sought to improve the geometric models with a better representation of the natural shape of muscles. SCHEEPERS et al. [Sch+97] and WILHELMS and VAN GELDER [WV97] used generalized cylinders with varying width along a single axis (Figure 1.10(b)). MIN et al. [Min+00] use skeleton-based implicit surfaces, which model fusiform muscles and more complex shapes such as pectorals, as shown in Figure 1.10(c), and used the gradient of the implicit field function to project the vertices to the correct distance to the surface.

More recently, LEE and ASHRAF [LA07] represented muscles with an axis made of quadratic Bézier curbes defined by two end points (origin and insertion) and a middle point (Figure 1.11(a)). While the origin and insertion are kinematically driven by the animation skeleton, the middle point is controlled by a dynamic mass-spring system linking it to the end points. The muscles width along the axis is given by a sinusoidal function, which is then sampled to create a low-resolution mesh representation of the muscle.

In the reference pose, the vertices $\mathbf{v_{ref}}_i$ are bound to the joint deformations with the standard LBS skinning weights $\mathbf{w}_i$ (as described in Section 1.4.1). They are given an additional set of weights $\mathbf{w'}_i$ for each neighbouring muscle vertex. During the animation,

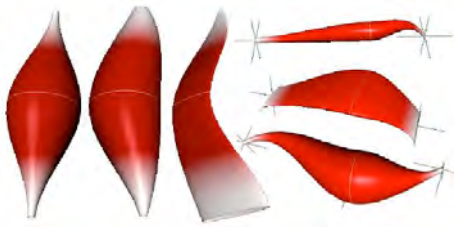


Figure 1.12: Muscles shapes obtained with an axis curve and a thickness curve.

(Picture from [RL13])

the position of the vertices is first computed by LBS

$$\mathbf{v}_i^{(\text{LBS})} = \sum w_{ij} \mathbf{B}_j \mathbf{v}_{\mathbf{ref}\,i}\,.$$

Then, the muscle deformation is transformed back in the reference pose, and the vertex is skinned with the muscle weights relatively to the deformed muscle vertex.

$$\mathbf{v}_i^{(\text{muscle})} = \sum w'_{ik} \mathbf{T}_k^{(\text{muscle})} \mathbf{v}_{\mathbf{ref}\,i}\,,$$

where $\mathbf{T}_k^{(\text{muscle})}$ is the transform of mesh vertex k relatively to its reference pose position. The computed point $\mathbf{v}_i^{(\text{muscle})}$ is interpreted as the *displacement* generated by the deformed muscle. The two are then combined by simply adding the muscle displacement to the result of LBS:

$$\mathbf{v}_i = \mathbf{v}_i^{(\text{LBS})} + \mathbf{v}_i^{(\text{muscle})}\,.$$

A similar approach, presented by RAMOS and LARBOULETTE [RL13], can represent more shapes by using two Bézier curves, one for the muscle axis as in the previous method, and one to design the muscle profile. A sample of these shapes can be seen in Figure 1.12. These two methods extend geometric skinning by having the vertices deformed both by the skeleton and the nearby muscles, by using extra weights to blend the deformation transform of the muscle with the skeleton joints transforms. However, as with other geometric methods relying on extra bones, this only makes the rigging process more cumbersome, as the number of weights for each vertex increases by a factor of two or three.

1.4.5 Discussion

Direct geometric methods based on skinning weights reign supreme when speed and efficiency are required, as in real-time applications. Nonetheless, deformations that they can reproduce in practice is limited. The indirect link between the skinning weights and

the final skinning results makes weight painting a long and tedious process which requires a lot of trial and error, as correcting the weights to improve the aspect of the skin in one pose can destroy the appearance of another unrelated pose.

Being purely kinematic, direct methods cannot reproduce *dynamic effects* such as elastic or plastic deformations. To create more complex effects such as a muscle bulging, the only options are either to add more bones and control points to the rig, which in turn increases the number of weights to set, limiting these effects in practical applications, or to use dedicated muscle deformers.

Finally, these methods fail to take self-collision into account. Often, when bending a character's arm or leg at a sharp angle, parts of the skinned mesh will traverse each other, creating a final skin which intersects itself. This becomes especially problematic with shape-based muscle deformers that work on top of standard geometric skinning: the bulging muscles often worsen the problem of self-collisions.

Removing these intersections post-hoc is a costly operation. Despite the availability of spatial acceleration structures (k-d tree, spatial hashing), solving mesh-mesh collision for complex character meshes often makes framerates drop below the second [AF15].

A recent effort to avoid self intersection in the mesh was presented by Vaillant et al. [Vai+13; Vai+14]. Their method uses implicit surfaces to model the skin along with dedicated contact operators, representing an interaction surface between body parts. This approach models contact in the skin efficiently and solves the volume loss and bulging artefacts of LBS and DQS.

This technique lends itself to extension, because it makes no assumption over the nature of the implicit surfaces used in the skin representation. In this thesis we present implicit deformers integrated into this skin representation that can represent muscles and other anatomic elements while elegantly solving the self-intersection problem.

This thesis presents a model which brings together Implicit Skinning with an implicit muscle shape deformer. While the muscle model is in line with the approaches described in Section 1.4.4, its integration with implicit skinning avoids the complex weight setup required by these methods, while reaping the benefits of Implicit Skinning. The next chapters provide a general background on implicit modelling and specifically on Implicit Skinning, defining the concepts used to design the model and integrate it with Implicit Skinning.

2 Introduction to implicit surfaces

No need to ask, he's a smooth operator

— Sade, *Smooth Operator*

This chapter presents the elemental concepts used to build implicit surfaces for 3D modelling, which we leverage for skinning.

Firstly, we start with the basic definitions of scalar fields and their iso-surfaces in Section 2.1. The following section (Section 2.2) presents the primitive models used in implicit modelling. We describe the main orientation and iso-value conventions: global support functions (Section 2.2.1) and compact support functions (Section 2.2.2). We then focus on extrusion surfaces, which are specifically used to model muscles in this work (Section 2.2.3).

Secondly, we introduce composition operators (Section 2.3) and transformation of scalar fields (Section 2.4) as means to assemble objects from several implicit primitives. Lastly, we expose the equations derived when using implicit surfaces in animation (Section 2.5).

2.1 Scalar fields and implicit surfaces

Implicit surfaces are geometric surfaces defined by an equation: they provide a convenient representation of many 3D objects. Historically, algebraic surfaces (implicit surfaces whose equation is polynomial) where among the first three-dimensional objects studied [HC32, chapter I and IV]. In computer graphics, implicit surfaces were used as representations for models in CAD software as early as the 1960s [Sab68].

Their strengths and weaknesses are complementary to other common surface representations in computational geometry [Hug+13; Wyv15]. In contrast to polygonal meshes, implicit surfaces can represent infinitely smooth surfaces; and, unlike parametric surfaces, they can define objects with arbitrarily complex topology.

Figure 2.1: A sample of implicit surfaces. From left to right: a sphere, a torus, a blend of two blobs, a Klein bottle, an inflated lemmniscate, an ellipsoid and a capsule.

Definition. A 3D scalar field is a real-valued function f defined over $\mathbb{R}^3$. This function f can then be used to define a surface in 3D space, by considering the preimage of a value $C \in \mathbb{R}$, i.e. the set of points $\mathbf{p}$ such that $f(\mathbf{p})$ is equal to C:

$$S_C = f^{-1}(C) = \left\{\mathbf{p} \in \mathbb{R}^3 \,|\, f(\mathbf{p}) = C\right\}.$$

Scalar fields are also known as *potential fields* by analogy with their use in physics as potential energy fields from which some forces derive.

In this work, it is assumed that functions f are at least C^1 almost everywhere (discontinuities are limited to a null-measure subset) and that S_C contains only *regular points*, i.e. points where the gradient ∇f is non-zero. Under these conditions, the set S_C is a 2D-manifold and is called the C-*iso-surface* of the scalar field f. Iso-surfaces are implicitly defined by the equation $f(\mathbf{p}) = C$, rather than by an explicit list of points or parametrization. They are thus known as *implicit surfaces*.

A scalar field thus represents an infinite family of implicit surfaces, one for each value of C (though some can be degenerate – e.g. reduced to a point – or empty). If f is differentiable, the scalar field not only gives the set of points on the surface but also a normal vector on each of these points:

$$\vec{\mathbf{n}}(\mathbf{p}) = \frac{\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|}.$$

From the first partial derivatives of f it is also possible to obtain closed formulae for the tangent and binormal vectors. Similarly, if f is twice differentiable, the Hessian matrix of f gives informations about the implicit surface's curvature [Gol05].

If C is a regular value of f, it divides the space $\mathbb{R}^3$ into two subsets, Ω_+ and Ω_-, where

$$\Omega_+ = \left\{\mathbf{p} \in \mathbb{R}^3 \,|\, f(\mathbf{p}) > C\right\} \text{ and } \Omega_- = \left\{\mathbf{p} \in \mathbb{R}^3 \,|\, f(\mathbf{p}) < C\right\} .$$

Which of these two subsets represents the inside of the object and which is the outside depends on the function. In general, the interior is often a bounded subset (and the exterior is unbounded), but this is not always the case (e.g. if S_C is an infinite plane). In practice, a convention on the surface orientation along with a standard value for the constant C is chosen in advance.

Example. The simplest implicit surface is perhaps the *sphere*; by defining

$$f(\mathbf{p}) = \|\mathbf{p}\|^2 = x^2 + y^2 + z^2 ,$$

the iso-surface where $f(\mathbf{p}) = R^2$ (for $R > 0$) is a sphere of radius R centred on the origin.

The sphere belongs to the family of algebraic surfaces called *quadrics*, where the function f is a second order polynomial. Other quadrics include ellipsoids, paraboloids, hyperboloids, cones and cylinders. Higher degree algebraic surfaces generate a wider array of shapes: Möbius strips (degree 3), tori (degree 4), Klein bottles (degree 12), as illustrated by Figure 2.1.

2.2 Implicit shape models

2.2.1 Distance fields and global support functions

Distance fields are another important family of scalar fields, where the function $f(\mathbf{p})$ is defined as the distance between $\mathbf{p}$ and a given geometric set (sometimes called *skeleton* or *source* of the field).

Example. With a given point $\mathbf{p_0} \in \mathbb{R}^3$ and

$$f(\mathbf{p}) = \|\mathbf{p} - \mathbf{p_0}\| ,$$

the iso-surface where $f(\mathbf{p}) = R$ is a sphere of radius R centred on $\mathbf{p_0}$.

Distance fields have useful mathematical properties of their own: they are continuous, and their gradient is unitary, i.e. $\|\nabla f(\mathbf{p})\| = 1$. The generated shapes depend on the type

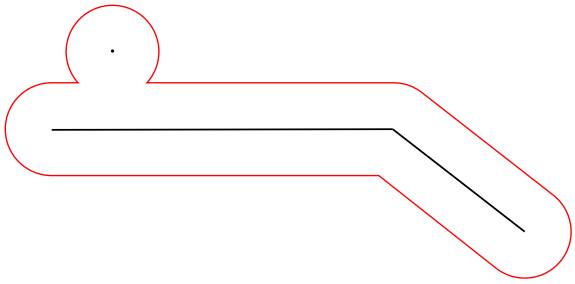


Figure 2.2: Skeleton-based surface (in red) defined relatively to a skeleton made of a point and two line segments.

of the skeleton: the previous example of the sphere is a distance field where the source is a single point $\mathbf{p_0}$, while using a segment as the skeleton will result in a *capsule* shape.

Distance fields can be easily extended to *signed distance fields* or SDF, where the sign of the field function at point $\mathbf{p}$ determines whether p is inside or outside the object, and the object's surface is conventionally defined as $f(\mathbf{p}) = 0$.

Example. Defining a sphere as a signed distance field from its centre point $\mathbf{p_0}$:

$$f(\mathbf{p}) = \|\mathbf{p} - \mathbf{p_0}\| - R\,.$$

A SDF is thus naturally oriented, using the sign of f to tell the inside from the outside of the object: with the previous example, a point is inside the sphere if $\|\mathbf{p} - \mathbf{p_0}\| < R$ i.e. where $f(\mathbf{p}) < 0$. This convention (referred in the following as the *global support convention*) can be extended to give a standard orientation to many 3D objects limited by implicit surfaces:

- the surface $\partial\Omega$ is the set S_0 where $f(\mathbf{p}) = 0$.
- the *inside* is the region Ω_- where $f(\mathbf{p})$ is negative
- the *outside* is the region Ω_+ where $f(\mathbf{p})$ is positive

More generally, *skeleton-based surfaces* are defined by functions whose value depend on the distance to their skeleton. The dependency to the distance can be any arbitrary function $\phi : \mathbb{R} \to \mathbb{R}$. Distance fields are a special case where ϕ is linear. Common examples using non-linear functions include radial-basis-functions (RBF), where the skeleton is only a point $\mathbf{c}$:

$$f(\mathbf{p}) = \phi\left(\|\mathbf{p} - \mathbf{c}\|\right).$$

More often, skeleton-based surfaces use segments or curves as their skeleton, as shown in Figure 2.2. Note that the use of *skeleton* in this context is unrelated to the *animation skeleton* defined in Chapter 1.

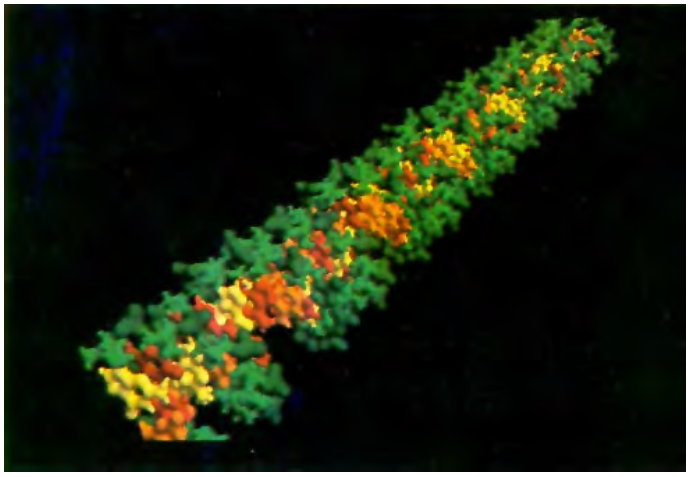


Figure 2.3: BLINN's *blobby molecules*: each atom is represented by a spherical primitive, the resulting molecule is rendered as a blending between all its atoms.

(Picture from [Bli82])

	Global support	Compact support
Iso-surface	$f(\mathbf{p}) = 0$	$f(\mathbf{p}) = 0.5$
Inside	$f(\mathbf{p}) < 0$	$f(\mathbf{p}) > 0.5$
Outside	$f(\mathbf{p}) > 0$	$f(\mathbf{p}) < 0.5$
Normal direction	∇f	$-\nabla f$
Union operator	MIN	MAX
Intersection operator	MAX	MIN
Complement	$-f$	$1 - f$

Table 2.1: A summary of the two main convention of implicit surfaces

2.2.2 Compact support functions

Definition. The *support* $\text{supp}(f)$ of a scalar field f is the set of points where f is non-zero:

$$\text{supp}(f) = \left\{\mathbf{p} \in \mathbb{R}^3 \,|\, f(\mathbf{p}) \neq 0\right\} .$$

SDF and similar functions usually have *global support*, which means $\text{supp}(f)$ is not bounded. Moreover, the absolue value of $f(\mathbf{p})$ grows higher as the distance from $\mathbf{p}$ to the surface increases.

From a computational point of view, this is an undesirable property: when evaluating several field functions, the value of distant objects will influence the result, and could create problems such as numerical instability and loss of precision, especially when blending several implicit surfaces together (see Section 2.3). On the other hand, when $\text{supp}(f)$ is

bounded, the function f is said to have *compact support*. In practice, this means their region of influence is limited to the neighbourhood of the implicit surface. This helps improving the efficiency of evaluating the composition of several implicit surfaces at a given point $\mathbf{p}$, as only functions having $\mathbf{p}$ in their support need to be evaluated, while others can be discarded based on a simple spatial test (e.g. a bounding box). This is also helpful for storing and retrieving the values in a spatial data structure (e.g. in a grid) by discretising the field's bounding box.

Compact support functions were introduced early on independently by several seminal papers under many different names such as BLINN's *Blobs* [Bli82], NISHIMURA et al.'s *Metaballs* [Nis+85] and WYVILL et al.'s *Soft Objects* [WMW86].

Given their nature, these functions follow a different convention than the global support convention:

- values of f are usually bounded between 0 and 1.
- the surface $\partial\Omega$ is the set $S_{\frac{1}{2}}$ where $f(\mathbf{p}) = \frac{1}{2}$.
- the *inside* is the region Ω_+ where $f(\mathbf{p}) > \frac{1}{2}$
- the *outside* is the region Ω_- where $f(\mathbf{p}) < \frac{1}{2}$

Compact support functions thus have the opposite orientation of SDF and similar fields. The normal (pointing outwards) of an implicit surface following the compact support convention is:

$$\vec{\mathbf{n}}(\mathbf{p}) = -\frac{\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|} .$$

A summary of the differences between conventions can be found in Table 2.1.

Skeleton-based compact support functions provide somewhat of an equivalent to the globally supported skeleton surfaces. Their field functions are defined as a decreasing function K of the distance d from $\mathbf{p}$ to their skeleton.

$$f(\mathbf{p}) = K(d)$$

Similarly to distance fields, the nature of the skeleton will change the overall shape (point skeletons create spheres, segments create capsules, etc.).

The choice of the *fall-of filter function* K has an influence on the properties of the field. The first function proposed for point-based surfaces was a Gaussian kernel, which was drawn from actual physical models of electromagnetic fields of molecules to help with

their graphical representation [Bli82]:

$$K(d) = \exp\left(-\frac{d^2}{\sigma^2}\right),$$

of which an illustration can be seen in Figure 2.3.

This function tends to zero as d increases, yet never reaches it (it is not a proper compact support function). Even so, for numerical applications, the value of K can be considered to vanish after a certain distance threshold (for example $K(4\sigma) \approx 10^{-7}$). Many different fall-of filter functions subsequently published explicitly set a threshold R (the *radius* of the support) after which K vanishes.

Metaballs [Nis+85; NN94], degree 2:

$$K(d) = \begin{cases} 1 - 3\left(\frac{d}{R}\right)^2 & \text{if } d < \frac{R}{3} \\ \frac{3}{2}\left(1 - \frac{d}{R}\right)^2 & \text{if } d \in [\frac{R}{3}, R] \\ 0 & \text{if } d > R \end{cases}$$

Soft objects [WMW86], degree 6:

$$K(d) = \begin{cases} 1 - \frac{4}{9}\left(\frac{d}{R}\right)^6 + \frac{17}{9}\left(\frac{d}{R}\right)^4 - \frac{22}{9}\left(\frac{d}{R}\right)^2 & \text{if } d < R \\ 0 & \text{if } d \geq R \end{cases}$$

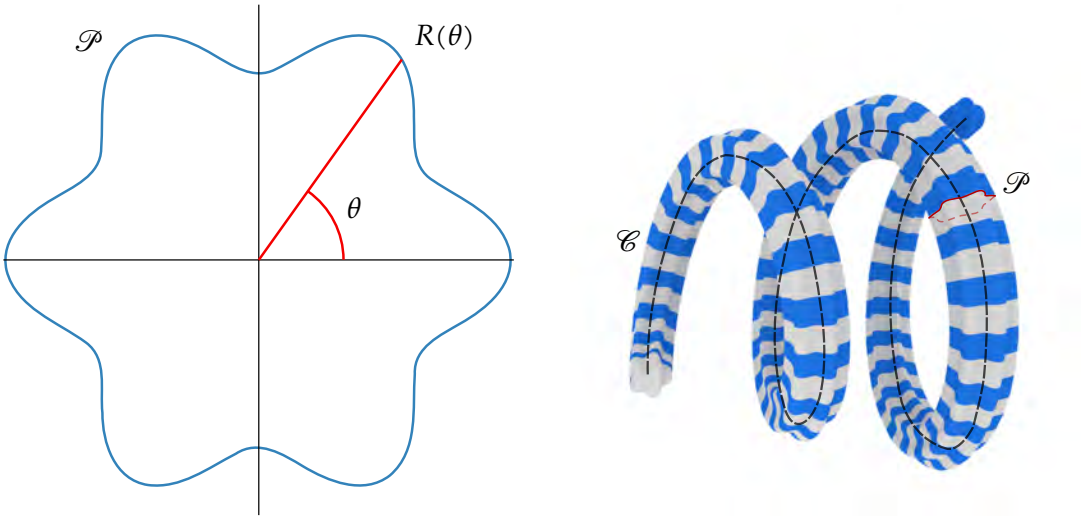
Blobby model [Blo97], degree 6:

$$K(d) = \begin{cases} \left(1 - \left(\frac{d}{R}\right)^2\right)^3 & \text{if } d < R \\ 0 & \text{if } d \geq R \end{cases}$$

These functions define compact support scalar fields, and usually use a polynomial formula for faster evaluation. Using a higher degree polynomial guarantees a higher order of continuity of the field at the threshold.

2.2.3 Extrusion surfaces

Extrusion surfaces, also known as generalized cylinders or sweep surfaces, are natural extensions of skeleton-based implicit surfaces [CBS96; GH99]. They start from a curve $\mathscr{C}(s)$, s being the curvilinear parameter of the curve, and sweep a *profile* curve $\mathscr{P}$ along the axis curve $\mathscr{C}$.



(**a**) Polar profile function. (**b**) Extrusion along an helix curve.

Figure 2.4: Example of a sweep surface. Figure (a) shows the polar profile $\mathscr{P}$. Figure (b) shows its extrusion along an helix curve $\mathscr{C}$

Skeleton surfaces, described in the previous section, can be seen as a special case where $\mathscr{P}$ is a circle of constant radius. The profile can however depend on the curvilinear coordinate of the axis curve s, for example, by making the radius grow and shrink with s. Circles of varying radius can produce worm-like shapes, which are suitable for modelling organic objects (animals, human limbs, etc.). Anisotropic profile curves extend the range of possible objects even more: by providing a frame orthogonal to the axis curve, the profile function can be described by a polar curve in the plane orthogonal to the axis curve, as shown in Figure 2.4(a). This gives a polar curve $\mathscr{P}$ defined by a radial function $R(s, \theta)$ in its most general form. This allows to define the extrusion of the polar curve along the axis curve, as illustrated by Figure 2.4(b).

Evaluating the field function at any query point $\mathbf{q}$ requires the projection of this point on the axis curve and computing the polar curve frame $(\vec{\mathbf{u}}, \vec{\mathbf{v}})$ which is locally orthogonal to $\mathscr{C}$ at this projected point. After that, it is possible to evaluate the radius R of the profile curve in the angular direction θ. The function evaluates to the difference between this radius, and the distance between $\mathbf{q}$ and the axis curve. This process is described in detail in Algorithm 2.1 and illustrated by Figure 2.5.

This definition yields a global support function but can be converted to a compact support function by any of the fall-off filter functions mentioned in Section 2.2.2.

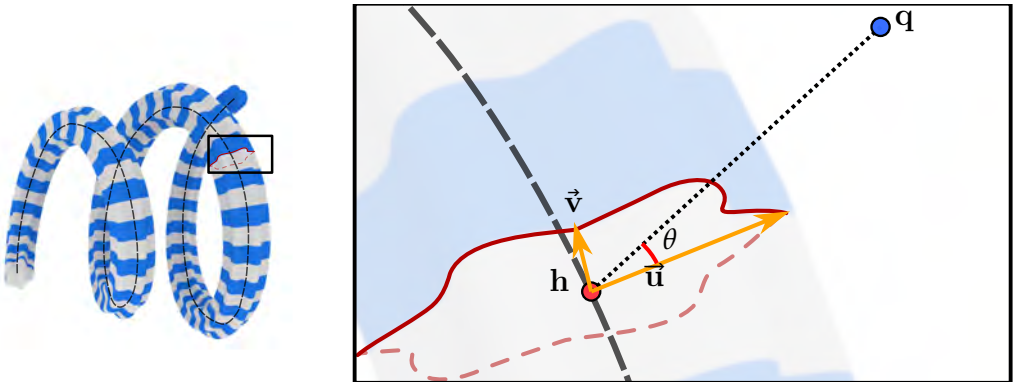


Figure 2.5: Evaluation of an extrusion surface field function: the point $\mathbf{q}$ is projected on the axis curve $\mathscr{C}$ on point $\mathbf{h}$. The evaluation is the difference between the distance $\left\|\overrightarrow{\mathbf{qh}}\right\|$ and the radius of the profile curve $R(s(\mathbf{q}),\theta)$, defined in the orthogonal plane $(\vec{\mathbf{u}},\vec{\mathbf{v}})$.

Algorithm 2.1 F_EXTRUSION: field function evaluation for an extrusion surface

---

input: a query point $\mathbf{q}$, an axis curve $\mathscr{C}$ and a profile curve defined along the axis $\mathscr{P}(s)$.
output: the value of the field function $f(\mathbf{q})$
$\mathbf{h} = \text{PROJECT}(\mathbf{q},\mathscr{C})$
Let $s(\mathbf{q})$ be the curvilinear parameter of $\mathbf{h}$ on $\mathscr{C}$
Evaluate the local normal frame $(\vec{\mathbf{u}},\vec{\mathbf{v}})$ at $\mathbf{h}$
Let $\theta(\mathbf{q})$ be the angle between $\vec{\mathbf{u}}$ and $\overrightarrow{\mathbf{qh}}$
Evaluate the radius of the polar curve $R(s(\mathbf{q}),\theta)$
return $\left\|\overrightarrow{\mathbf{qh}}\right\| - R(s(\mathbf{q}),\theta)$

2.3 Composition of implicit surfaces

A key feature of implicit surfaces is the ability to combine several field functions to define a new surface whose properties are defined by the underlying fields.

2.3.1 Composition operators

Definition. Given n field functions $\{f_i\}_{i=1..n}$, a n-ary composition operator is a function $G : \mathbb{R}^n \to \mathbb{R}$ which defines a new scalar field F such as

$$F(\mathbf{p}) = G(f_1(\mathbf{p}),...,f_n(\mathbf{p}))\,.$$

The gradient of this new field ∇F is easily computed by the chain rule:

$$\nabla F(\mathbf{p}) = \sum_{i=1}^{n} \frac{\partial G}{\partial x_i}\Big|_{f_i(\mathbf{p})} \nabla f_i(\mathbf{p})\,.$$

In general, the continuity class of F is the minimum of the continuity of all the f_i and the continuity of G itself. This means discontinuities can appear if the operator is non-smooth even if all the underlying fields are.

Example. Set-theoretic operators $G_{\min}$ and $G_{\max}$ can be defined as:

$$G_{\min}(f_1, ..f_n) = \min(f_1, ..f_n)$$
$$G_{\max}(f_1, ..f_n) = \max(f_1, ..f_n)$$

In the global support convention, the field resulting from applying the MIN operator defines the geometric union of the two objects while the MAX operator defines the intersection. The opposite is true for the compact support convention (see Table 2.1).

It is easy to express the usual Constructive Solid Geometry (CSG) operations such as geometric union, intersection and difference in the implicit surfaces framework [Req80]. It is natural to define complex objects by repeated assembly of implicit primitives recursively composed by operators which leads to define tree structures much like a classical CSG tree [WMW86]. This composition tree, referred to as the *blob-tree* in the subsequent literature, has become the standard for implicit modelling.

Composition operators have been extensively studied, as they are the key to modelling complex objects as assemblies of implicit surfaces. To have more control on the assembly result, it is convenient to restrict the blending operators to binary: while it is possible to add a few parameters to control a binary composition, extending it to n-ary would require too many due to the combinatorial explosion.

An example of such a controllable binary operator for global support scalar fields was proposed by PASKO et al. [Pas+95]:

$$G(f_1, f_2) = f_1 + f_2 - \sqrt{f_1^2 + f_2^2} + \frac{a_0}{1 + \left(\frac{f_1}{a_1}\right)^2 + \left(\frac{f_2}{a_2}\right)^2}.$$

This operator provides three parameters: a_0 controls the global strength of the blend while a_1 and a_2 can give more weight to f_1 or f_2, creating an *asymmetric* composition operator. With $a_0 = 0$ this operator behaves as a *clean union*: it generates an exact (sharp) union between the two implicit surfaces and a C^1 field everywhere else, as demonstrated by Figure 2.6. This operator helps to maintain the continuity of the field over recursive compositions while maintaining the sharp features of the modelled surfaces. BARTHE et al. [BWG04] and BERNHARDT et al. [Ber+10] adapted these operators to the compact support convention.

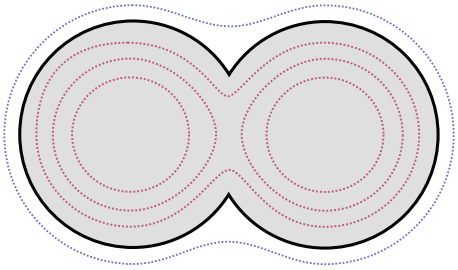
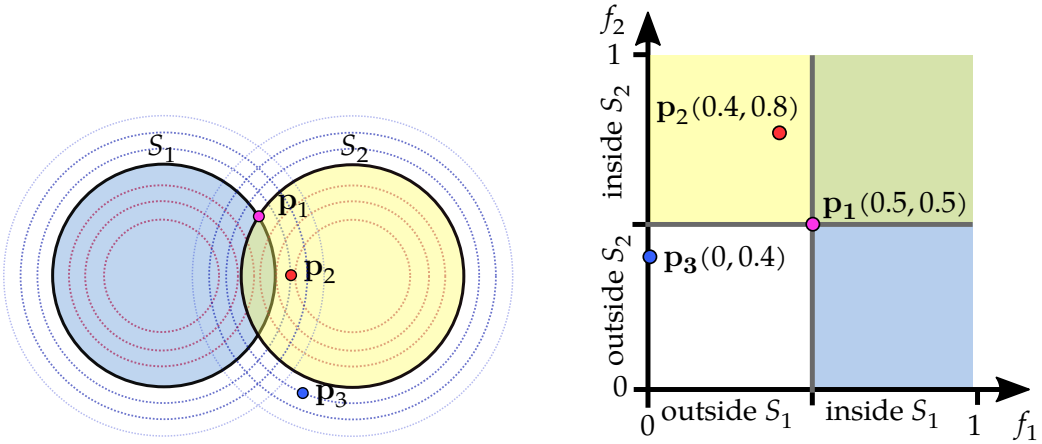


Figure 2.6: Clean union of two spheres. The implicit surface (black line) is the union of two spheres generating a sharp edge at their interface. The other iso-values (dashed lines) are smooth.



(**a**) Two implicit spheres

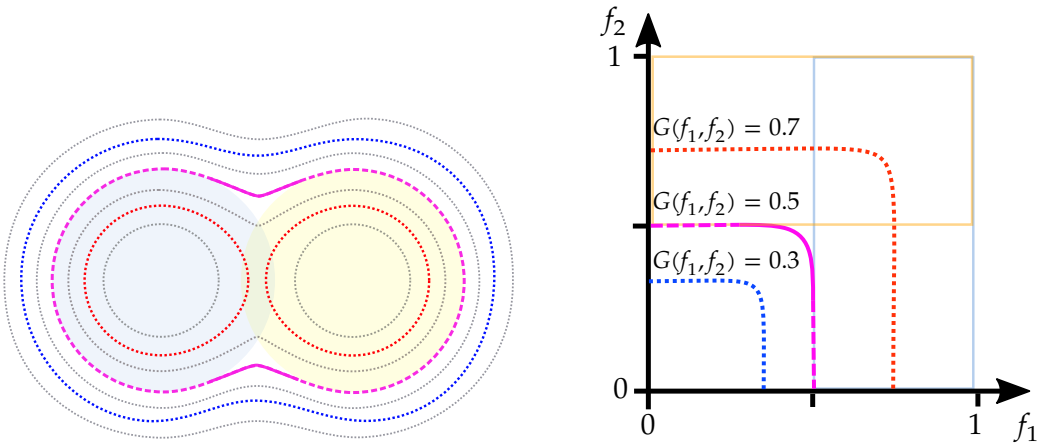
(**b**) Plane of scalar fields values

Figure 2.7: Two implicit spheres and 2D space of field values. Figure (a) shows two implicit spheres S_1 and S_2 defined by $f_1 = 0.5$ and $f_2 = 0.5$. Figure (b) shows the 2D space defined by (f_1, f_2). Points $\mathbf{p}_i$ in 3D space are mapped to the points of coordinates $(f_1(\mathbf{p}_i), f_2(\mathbf{p}_i))$.

2.3.2 Graphical representation of operators

At this point, it is necessary to introduce a visualisation method to illustrate the operators discussed in this section. Graphical representations are extremely useful to better understand, control and design composition operators. For binary operators, it is possible to illustrate the operator by displaying the graph of the function $G(f_1, f_2)$ on a 2D plane [HH85].

On this 2D graph, the x-axis plots values of f_1 and the y-axis values of f_2. To each point $\mathbf{p}$ in 3D space corresponds a point on the 2D plane where the coordinates are $f_1(\mathbf{p})$ and $f_2(\mathbf{p})$. If functions follow the compact support convention, all possible values lie in the square $[0, 1] \times [0, 1]$. An example of this representation is shown in Figure 2.7.



(**a**) Composition of two spheres (**b**) Iso-lines of operator

Figure 2.8: Graphical representation of binary operators. Figure (a) shows the composition of the two spheres of the previous figure, whith highlighted iso-surfaces. Curves on Figure (b) are the iso-lines of the operator function G depicted in the 2D-space of values corresponding the higlighted iso-surfaces. The purple curve corresponds to $G(f_1,f_2) = 0.5$, i.e. the implicit surface defined by the composition using G. Two other curves are shown, the red curve $G(f_1,f_2) = 0.7$ belongs to the inside of the composition, while the blue curve $G(f_1,f_2) = 0.3$ belongs to the outside. These colours are adopted throughout this work.

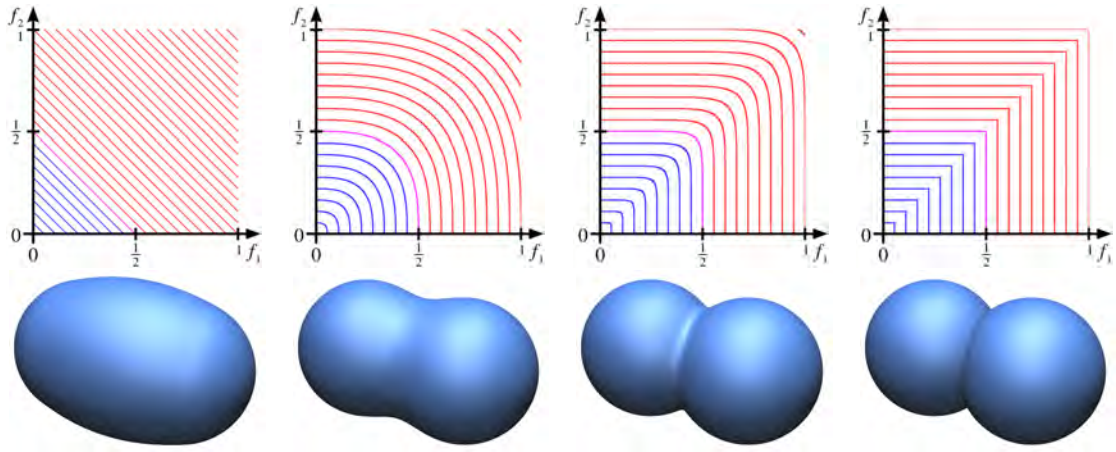
Operators are shown by plotting different curves corresponding to iso-values of G. For example, the resulting surface of the composed field is represented by the curve $G(f_1,f_2) = 0.5$ in the graph. Other iso-values curves depict the result of the composition on the inside and the outside of the object. For example, Figure 2.8 depicts an operator which smoothly blends two spheres.

2.3.3 Blending operators

Union and clean-union operators are often used in computer-assisted design to model objects with sharp edges. Blending operators are another important family of operators, enabling to assemble implicit objects and maintain the smoothness of the resulting assembled surface. To achieve this result, the operator itself must be continuous (ideally C^∞) as recursive blending with low-order continuity operators can lead to discontinuities in the final result. This property is especially useful to model organic objects or fluids.

When introducing compact-support blobby objects for rendering atoms and molecules, BLINN [Bli82] naturally defined the sum operator for field functions representing the probability of presence of electrons (see Figure 2.3):

$$G_+(f_1,...f_n) = \sum_{i=1}^{n} f_i \,.$$



(**a**) Sum operator (**b**) Ricci-3 operator (**c**) Ricci-10 operator (**d**) Union operator

Figure 2.9: Top row: example of composition operators, showing their graphical representation on a 2D plane. Bottom row: result of their application to blend two spheres.

(Pictures from [Can16])

An early attempt on expanding the set-theoretic operations by Ricci [Ric73] led to defining a family of operators:

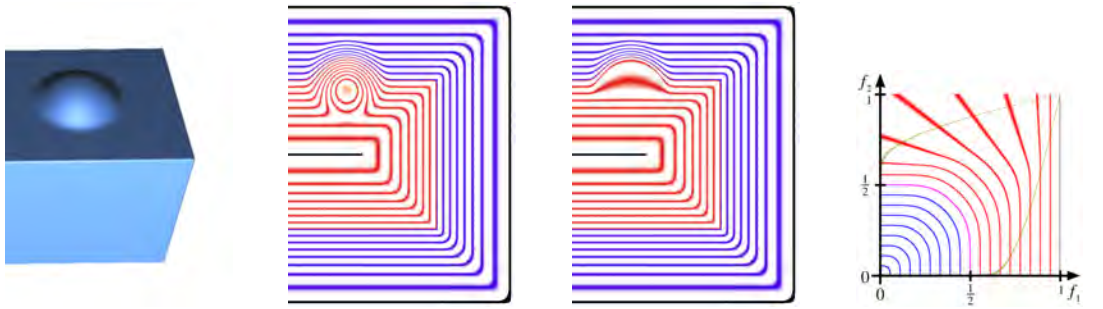
$$G_k(f_1, ..f_n) = \sqrt[k]{\sum_{i=1}^{n} f_i^k}.$$

The parameter k controls the behaviour of the blending; G_1 is actually the sum operator, but as k increases, G_k tends towards the max operator, while remaining C^∞. In Figure 2.9, we illustrate the result of composition with several operators, by showing their 2D graph, along with their application to two spheres.

The blending of implicit surfaces is useful for modelling, yet blending operators can behave quite unintuitively or lead to unwanted artefacts. As a consequence, many research efforts were directed towards providing more control on the final result to the designer and avoiding several pitfalls associated with blending [BGC01; Bar+03].

While compact-support functions are especially useful for composition, the blending operators described above will sometimes generate a composed function whose values go higher than 1. This can be problematic when chaining several composition operations together. It is always possible to clamp the results to 1, but this has an adverse effect on the metric in the inside parts of the objects. Canezin et al. [CGB13] developed an efficient solution by crafting specific operators whose value stays between 0 and 1.

In addition, they provide two new operators: a *detail operator* and a *difference operator* which respectively enable to add or remove small parts of a main object without creating critical



(**a**) Blending scene (**b**) Regular blend (**c**) Detail blend (**d**) Graph of operator

Figure 2.10: Figure (a): effect of detail operator on the blending of a spherical detail on a box. Figure (b): using regular blending generates critical points inside the surface. Figure (c): using the detail operator improves the quality of the resulting field. Figure (d) shows the operator itself.

(Pictures from [CGB13])

points near the border. As shown on Figure 2.10, the resulting surface of the detail blending is not modified but the regularity of the scalar field inside is improved, by eliminating the critical point at the centre of the sphere detail.

Blending operators have been also been plagued by three common types of unwanted features which arise in modelling.

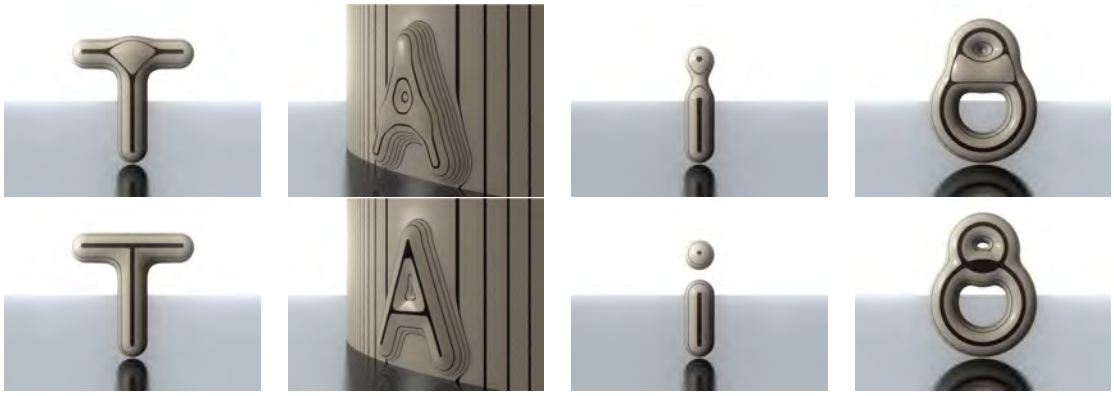
Bulging is the appearance of unwanted bumps on the surfaces. The blending operators add volume around the edges to create a smooth surface, but the addition is not localized and can create a bulge, e.g. in the case of the T-junction shown in Figure 2.11(a).

Blurring of details is caused by the difference of metric in the scalar fields which can make small objects deformed when blended with a larger object. This case is illustrated by Figure 2.11(b).

Changes of topology include blending at a distance when two separate objects are blended because of their proximity (Figure 2.11(c)), and the closing of holes, changing the surface's genus (Figure 2.11(d)).

2.3.4 Gradient-based operators

GOURMEL et al.'s [Gou+13] *gradient-based operators* provide a solution to the blending issues by defining an operator whose result depends not only on the blended fields values but also on their gradients. The key idea of gradient-based operators is to define a family of operators G_α controlled by a parameter $\alpha \in [0, 1]$. Variations of α interpolate G_α between a clean union and a blending operator. While BARTHE et al. [BWG04] and BERNHARDT et al.



(**a**) Bulging (**b**) Blurring of details (**c**) Blending at distance (**d**) Closing of holes

Figure 2.11: Top row: common blending artefacts. Bottom row: expected solutions.

(Pictures from [Gou+13])

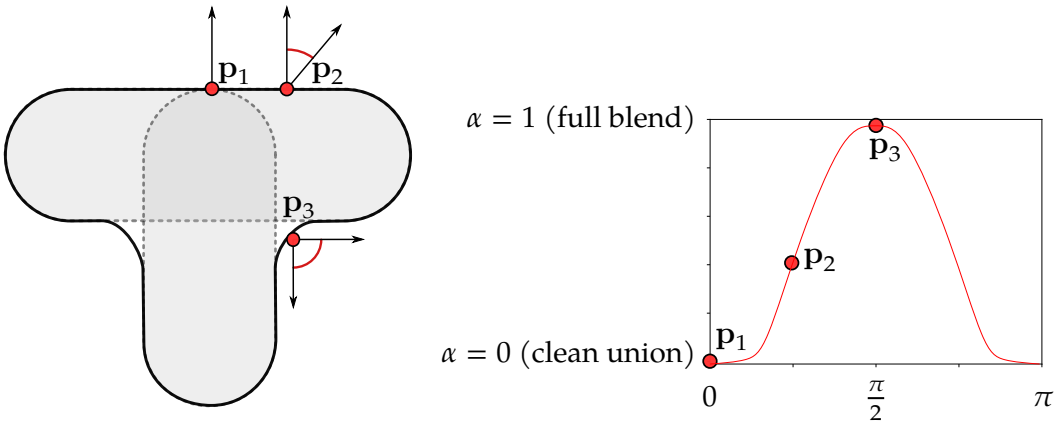
[Ber+10] already defined such families of blending functions; the contribution of Gourmel et al. [Gou+13] is to tie the value of the parameter α to the angle θ between the gradients $\nabla f_1(\mathbf{p})$ and $\nabla f_2(\mathbf{p})$ of the two composed fields. The function $\alpha = \kappa(\theta)$ is specified by the user as a *controller function*.

Computing the composition of two functions f_1 and f_2 is thus a multi-step process.

1. Evaluate $f_1(\mathbf{p}), f_2(\mathbf{p}), \nabla f_1(\mathbf{p})$ and $\nabla f_2(\mathbf{p})$;
2. Compute $\theta = \arccos(\nabla f_1(\mathbf{p}) \cdot \nabla f_2(\mathbf{p}))$;
3. Compute $\alpha = \kappa(\theta)$;
4. Return $F(\mathbf{p}) = G_\alpha(f_1(\mathbf{p}), f_2(\mathbf{p}))$.

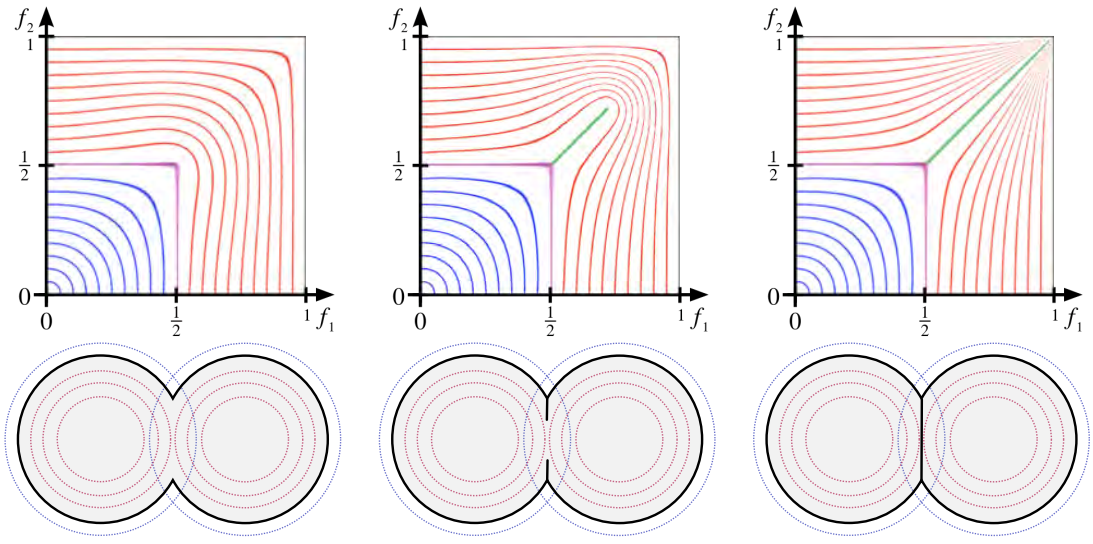
Gradient-based operators provide an extra degree of freedom to model complex interactions between two scalar fields, with the specification of the controller function to enable the blending only at selected angles. With a specific controller, each of the three aforementioned blending artefacts can be avoided. For example, Figure 2.12 shows how a gradient-based operator resolves the bulging artefact of the T-junction.

For practical applications in modelling, it is often preferable to specify the operator in terms of the desired result instead of using complex closed-formulae. For example, the *contact operators* by Vaillant et al. [Vai+14] create a contact surface between two colliding implicit surfaces, shown on Figure 2.13. This family of operators was designed by imposing boundary conditions on the operator: $G(f_1, 0) = f_1$ and $G(0, f_2) = f_2$. In addition, the 0.5-isosurface must be the union of the two initial surfaces, with an added *contact surface* at the intersection of the two objects, yielding the pattern seen in Figure 2.14. The remainder



(**a**) Gradient-based blending (**b**) Controller function

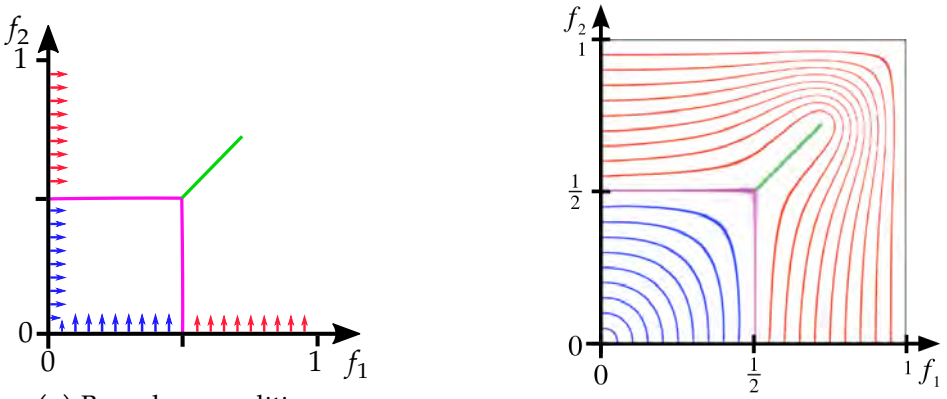
Figure 2.12: Solving the bulging artefact with gradient-based blending. Figure (a) shows the T-junction of two surfaces. Three points are shown with the gradients of the two primitive fields. On Figure (b), a plot of the controller is shown with the corresponding value of α for the three points.



(**a**) $\alpha = 0$ (clean union) (**b**) $\alpha = 0.5$ (contact) (**c**) $\alpha = 1$ (full contact)

Figure 2.13: Top row : Contact operator for various values of α, from clean union without contact to full contact surface. Bottom row : Scalar fields of two spheres (dashed lines) and surface resulting from the blend.

(Pictures from [Vai+14])



(**a**) Boundary conditions

(**b**) Result of interpolation

Figure 2.14: Construction of the contact operator for $\alpha = 0.5$. Figure (a): boundary conditions are imposed on the sides (when f_1 or $f_2 = 0$) and at the 0.5-isosurface, which must include the union of the two surfaces (in purple) and generate a contact surface when $f_1 = f_2$ inside the intersecting object (in green). Figure (b) shows the resulting operator after biharmonic interpolation.

of the field is computed by biharmonic interpolation, guaranteeing a C^2-smooth field outside of the imposed conditions.

They also describe a *bulge-in-contact* operator which artificially inflates the surface around the contact region. The most recent work in operator design enables specification of the blending result by sketching the desired surface [Ang+17].

2.4 Transforming implicit surfaces

2.4.1 Spatial transformation of a scalar field

Implicit objects can easily be deformed by spatial transforms, which is especially useful for modelling as it makes it possible to create complex shapes by twisting, bending or squeezing simple objects [Bar84; WO97].

Definition. If f is a scalar field defining an input implicit surface, and $w : \mathbb{R}^3 \to \mathbb{R}^3$ a continuous spatial transform or *warp function*. The transformed field $\bar{f}$ is defined as $\bar{f}(\mathbf{p}) = f(w^{-1}(\mathbf{p}))$.

The warp function maps the input implicit surface's space to the space where it is deformed. When computing the value of $\bar{f}(\mathbf{p})$, the query point must be transported back from this deformed space to the initial space for f to be computed, hence the inverse of w appearing in the formula (see Figure 2.15 for an illustration of this process). Warping

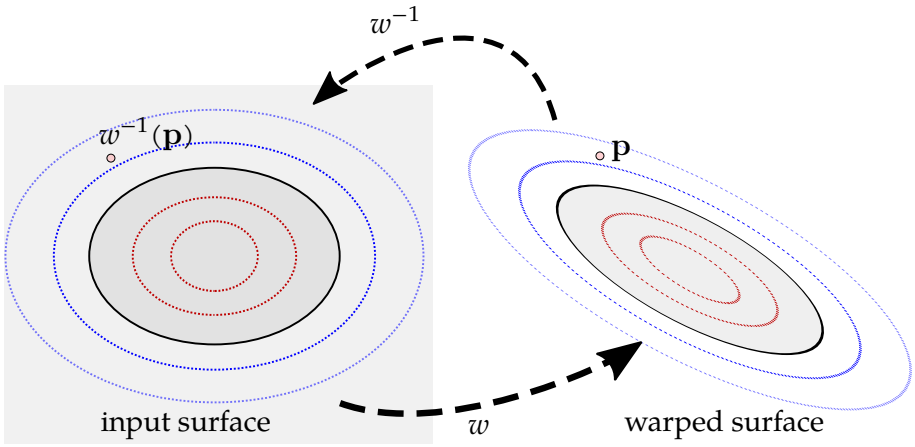


Figure 2.15: Warping of an implicit surface.

requires that w is at least *locally invertible*, meaning that w is $\mathcal{C}^1$ and that its Jacobian matrix is nonsingular, i.e. $|\mathbf{J}_w| \neq 0$.

2.4.2 Gradient of a deformed field

The gradient of a transformed implicit function $\nabla\bar{f}$ can be computed by the chain rule:

$$\nabla\bar{f}(\mathbf{p}) = (\mathbf{J}_{w^{-1}}(\mathbf{p}))^{\mathrm{T}} \nabla f(w^{-1}(\mathbf{p}))\,,$$

where $\mathbf{J}_{w^{-1}}(\mathbf{p})$ is the *Jacobian matrix* of w^{-1} at $w^{-1}(\mathbf{p})$, which is the inverse of the Jacobian matrix of w at $w^{-1}(\mathbf{p})$:

$$\mathbf{J}_{w^{-1}}(\mathbf{p}) = \left(\mathbf{J}_w(w^{-1}(\mathbf{p}))\right)^{-1}.$$

The matrix that locally transforms gradients (and thus normals) is the *inverse transpose* of the Jacobian matrix of w:

$$\nabla\bar{f}(\mathbf{p}) = (\mathbf{J}_w(w^{-1}(\mathbf{p})))^{-\mathrm{T}} \nabla f(w^{-1}(\mathbf{p}))\,.$$

Example. In the simple case where w is an invertible affine transform $\mathbf{T}(\mathbf{p}) = \mathbf{M}\mathbf{p} + \vec{\mathbf{t}}$, the Jacobian matrix $\mathbf{J}_\mathbf{T}$ is constant and is equal to the linear part of $\mathbf{T}$: $\mathbf{J}_\mathbf{T} = \mathbf{M}$. This yields

$$\nabla\bar{f}(\mathbf{p}) = \mathbf{M}^{-\mathrm{T}} \nabla f(\mathbf{T}^{-1}(\mathbf{p}))\,.$$

If $\mathbf{T}$ is actually a rigid transform (i.e. an isometry), then $\mathbf{M}$ is a rotation $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{R}^{-\mathrm{T}} = \mathbf{R}$, which simplifies the formula even further :

$$\nabla\bar{f}(\mathbf{p}) = \mathbf{R} \nabla f(\mathbf{T}^{-1}(\mathbf{p}))\,.$$

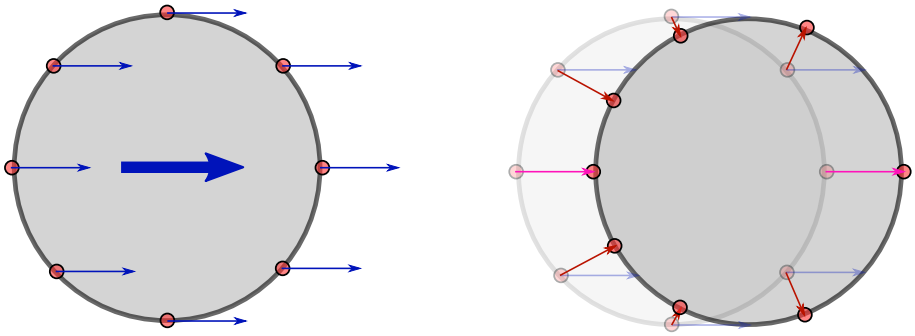


Figure 2.16: Tracking points on a translating implicit sphere. Left: points on a translating sphere theoretically have the same velocity everywhere. Right: after advancing the sphere the tracking points will project radially to the surface and slide towards the back of the sphere.

2.5 Animated implicit surfaces

Computer-generated animation has often made use of implicit surfaces since their appearance in computer graphics. They provide a simple representation that handles situations such as soft objects, objects changing topology or fluids much better than the ubiquitous polygon meshes [YT13; CGB16].

Desbrun and Cani [DC98] started investigating on the issues raised by time-varying scalar fields $f(\mathbf{p}, t)$, most importantly the trajectories of points on the surface. Considering a point $\mathbf{p}(t)$ which is constrained on the C-iso-surface of f, can be expressed as:

$$\forall t\,,\, f(\mathbf{p}(t), t) = C.$$

Differentiating this equation in t yields:

$$\frac{\mathrm{d}f}{\mathrm{d}t}(\mathbf{p}(t), t) = \frac{\partial f}{\partial t}(\mathbf{p}(t), t) + \nabla f(\mathbf{p}(t), t) \cdot \frac{\mathrm{d}\mathbf{p}(t)}{\mathrm{d}t} = 0\,.$$

This equation is of great importance for the *tracking* of animated implicit surfaces [WH94], i.e. attempting to find the trajectory of a set of points initially on the surface at $t = 0$.

Tracking is used, for example, to keep a point cloud or a polygon mesh in sync with an implicit surface. Initial points can be user-supplied or obtained by any polygonisation method (e.g. *marching cubes* [LC87; Nie04]), and then track the surface during the animation, maintaining a consistent representation of the moving surface.

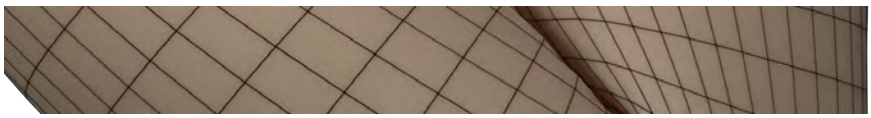
As a consequence, $f(\mathbf{p}(t), t)$ is constant (as $\mathbf{p}$ stays on the same iso-surface during motion), but $\mathbf{p}(t)$ is not, thus the previous relationship gives a differential equation on the velocity of $\mathbf{p}$:

$$\frac{\mathrm{d}\mathbf{p}(t)}{\mathrm{d}t} \cdot \nabla f(\mathbf{p}(t), t) = -\frac{\partial f}{\partial t}(\mathbf{p}(t), t)$$

This differential equation gives only a *radial* constraint on the velocity of $\mathbf{p}$ (along ∇f), leaving the *tangential* components unknown. This is a common issue with implicit surfaces, as the scalar field itself has no tangential parametrization *per se*. For example, the field of a sphere spinning on itself would be identical to a static field.

Solving the tracking problem thus requires to supply these tangential components with additional hypotheses. Ignoring the tangential component will lead to errors even in simple cases such as a translating sphere: the tracking points which are not aligned with the velocity vector will simply slide along the surface until they are concentrated to the points aligned with the translation vector, as can be seen in Figure 2.16. Stam and Schmidt [SS11] proposed to avoid these errors by maintaining the normals at the tracking points constant during the animation, which works for objects in translation, but not in rotation or for deforming objects. Fujisawa et al. [FMM13] presented a more robust approach based on curvature invariance.

This highlights how implicit surfaces and meshes complement each other in animation. Vertices of a mesh can track a moving implicit surface by combining normal information from the field and its gradient, and use the tangential information of the original mesh. This idea is at the core of the Implicit Skinning algorithm, which uses the tools presented in this chapter to solve several issues of geometric skinning.



3 Implicit Skinning

All problems in computer graphics can be solved with a matrix inversion.

— James F. Blinn, *Jim Blinn's Corner: Dirty Pixels*

With their capacity to define smooth organic-looking surfaces with only a small number of parameters, implicit surfaces were considered early on for character animation. Their low memory footprint was seen as a key point for storage and transmission of digital animated characters in applications such as video-conferences [SOP95] or virtual reality [TSC96].

As GPUs became increasingly common, the cost of rendering polygon meshes became lower while the display of implicit surfaces remained computationally expensive. Implicit surfaces thus fell out of favour in mainstream animation applications. Yet they remain a useful complement surface representation to polygon meshes. This complementarity was exploited by Vaillant et al. [Vai+13; Vai+14]. Their method is able to correct many issues of geometric skinning while remaining fast enough for interactive use.

In the following sections, we detail the necessary steps to define the skin scalar field and describe how this representation is used for skinning.

A first step, described in Section 3.1, constructs the implicit surface representation of the skin from the character mesh in the reference pose and maintains its consistency with the skeleton during the animation. The mesh is segmented according to the skeleton bones, and each part of the mesh is represented by a separate scalar field (Section 3.1.1). These primitive fields are then combined with composition operators (Section 3.1.2). The result of this process is an implicit surface which closely matches the initial character mesh. During the animation, the implicit surfaces are transformed rigidly with the animation skeleton, and the resulting surface obtained by composition of these primitives thus adapts to the current animation pose, as described in Section 3.1.3.

(**a**) Mesh partition

(**b**) Implicit surfaces of the parts and reconstructed skin

Figure 3.1: Partition and reconstruction of the mesh. Vertices are separated in parts associated to skeleton bones (a) and each part is represented by a scalar field, which are composed together to form the implicit skin (b)

(Pictures from [Vai+14])

The next sections detail the Implicit Skinning algorithm. As a first step, DQS (described in Section 1.4.2) moves the vertices to their starting position. From this initial guess, the vertices are alternatively projected towards their target iso-surface on the implicit skin and displaced tangentially to minimize a local deformation energy. We first present each operation independently: the iso-surface tracking in Section 3.2 and the energy minimization in Section 3.3. Finally, we detail how these two steps are interleaved to make the vertices converge to their final position in Section 3.4.

3.1 Implicit skin representation

3.1.1 HRBF primitives

The first step to setup Implicit Skinning is to generate an implicit surface which approximates the character mesh, while being able to deform with the skeletal movements. To this effect, the character mesh (in rest pose) is partitioned by associating each vertex with the skeleton bone which has the most influence on its movement (Figure 3.1(a)). If skinning weights are available, vertices are associated with the highest weighted bone; if not, automatic partition methods or a user-supplied partition can still be used.

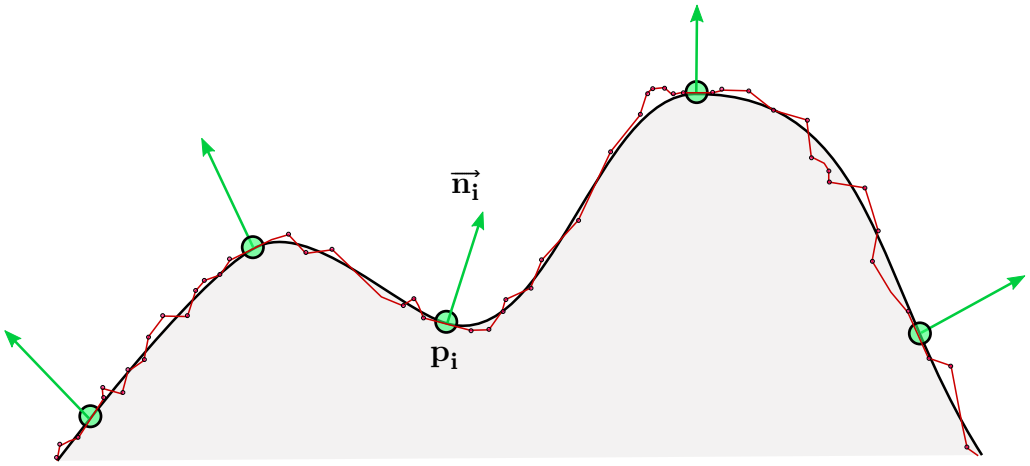


Figure 3.2: HRBF surface reconstruction: Points $\mathbf{p_i}$ and normals $\overrightarrow{\mathbf{n_i}}$ are sampled from the mesh surface (red line). The implicit surface interpolates the input to reconstruct a smooth approximation of the mesh.

Each part of the mesh is then approximately reconstructed by an implicit surface using Hermite Radial Basis Functions or HRBF [MGV11], as shown on Figure 3.1(b). Starting from a set of N_H points $\mathbf{p_i}$ and normals $\overrightarrow{\mathbf{n_i}}$, in our case sampled from the mesh's surface by Poisson disc sampling [Bri07], HRBFs generates a scalar field f for which $f(\mathbf{p_i}) = 0$ and $\nabla f(\mathbf{p_i}) = \overrightarrow{\mathbf{n_i}}$, as shown in Figure 3.2.

Definition. Given a basis function $\phi : \mathbb{R} \to \mathbb{R}$, an HRBF is defined as

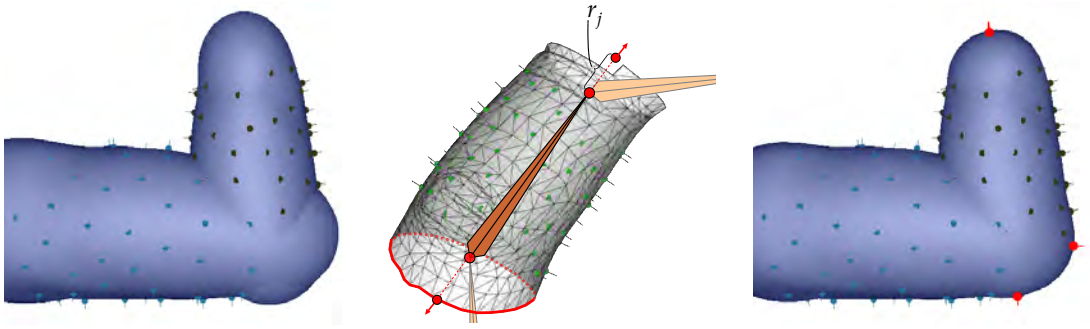
$$f_\mathrm{H}(\mathbf{p}) = \sum_{i=1}^{N_\mathrm{H}} a_i \phi\left(\|\mathbf{p} - \mathbf{p}_i\|\right) + \vec{\mathbf{b}}_i \cdot \nabla\left(\phi\left(\|\mathbf{p} - \mathbf{p}_i\|\right)\right).$$

This field f_H is expressed as a linear combination of radial-basis functions ϕ, whose value only depend on the distance to a center point, and their gradient. Computing the coefficients a_i and $\overrightarrow{\mathbf{b_i}}$ of the linear combination thus requires to solve the equations:

$$\begin{cases} f_\mathrm{H}(\mathbf{p_i}) & = 0 \\ \nabla f_\mathrm{H}(\mathbf{p_i}) & = \overrightarrow{\mathbf{n_i}} \end{cases}$$

which can be done by solving a linear system of size $4N_\mathrm{H}$. The function ϕ used in Implicit Skinning is $\phi(r) = r^3$.

The Poisson sampling gives generally good results but its random nature sometimes makes the HRBF's shape quite different from the initial mesh part it is supposed to represent. Such problems arise generally when sampling high-frequency areas of the mesh where the sampled point can lie on sharp details and thus distort the shape representation.



(**a**) Without extra end points (**b**) Adding the extra points (**c**) With extra end points

Figure 3.3: Effect of extra endpoints on the implicit skin. The reconstructed HRBF shape is often elongated near the joints, yielding unwanted bulging when the joint bends (a). By adding extra points at each end along the skeleton bone (b), this bulge is averted (c).

(Pictures from [Vai+13])

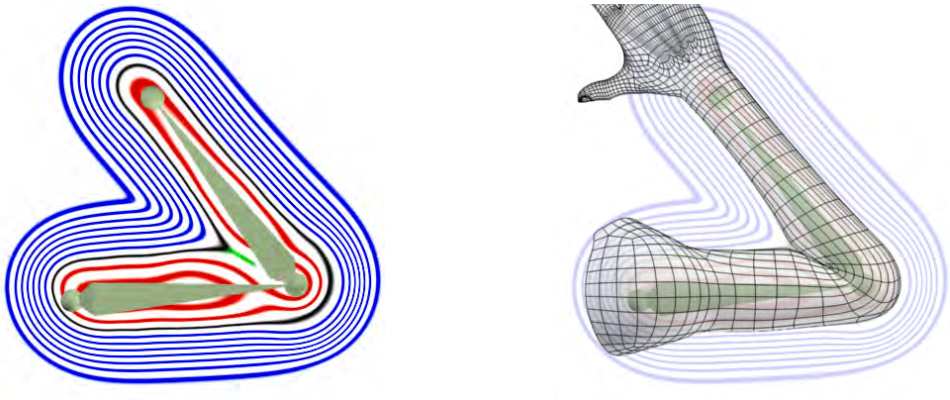
Fortunately, after HRBF points are sampled, they can still be manually edited by adding, deleting points or changing the sampled points' position or normal, to better capture the shape without the high-frequency details.

Because the mesh vertices belonging to one part often exhibit a cylindrical shape, the reconstructed HRBF fails to capture the shape at the end of this part, where the mesh connects with the next part over a skeleton joint. This yields an unwanted bulge when the joint bends, as shown in Figure 3.3(a). To correct this artefact, two points are added at each end of the shape. Computing the distance between the proximal and distal end points of the bone and the group of vertices $\mathbf{v_i}$ gives a radius r_j for each endpoint. A new HRBF sample point is then added at this distance, following the direction of the bone segment, as illustrated in Figure 3.3(b).

Once the HRBF surface closely matches the input mesh, the result can be saved alongside the model and reused in subsequent animations. Each HRBF field is converted to a compact support scalar field with the following fall-of filter function:

$$K(d) = \begin{cases} 1 & \text{if } d < -R \\ -\frac{3}{16}\left(\frac{d}{R}\right)^5 + \frac{5}{8}\left(\frac{d}{R}\right)^3 - \frac{15}{16}\left(\frac{d}{R}\right) + \frac{1}{2} & \text{if } d \in [-R, +R] \\ 0 & \text{if } d > R \end{cases}, \tag{3.1}$$

which guarantees C^2 continuity at $K(d) = 0$ and $K(d) = 1$. The support radius R is set at the highest distance from the sampled points $\mathbf{p_i}$ to the bone axis, given that limbs usually exhibit a rough rotational symmetry around their animation bone.



(**a**) Field composition with contact operator (**b**) Result after skinning

Figure 3.4: Effect of the contact operator on skinning: two scalar fields (arm and forearm) are blended with a contact operator, generating a contact surface visible on Figure (a) in green. Skin vertices project on this surface, resulting in a mesh without interpenetration (b).

(Pictures from [Vai+14])

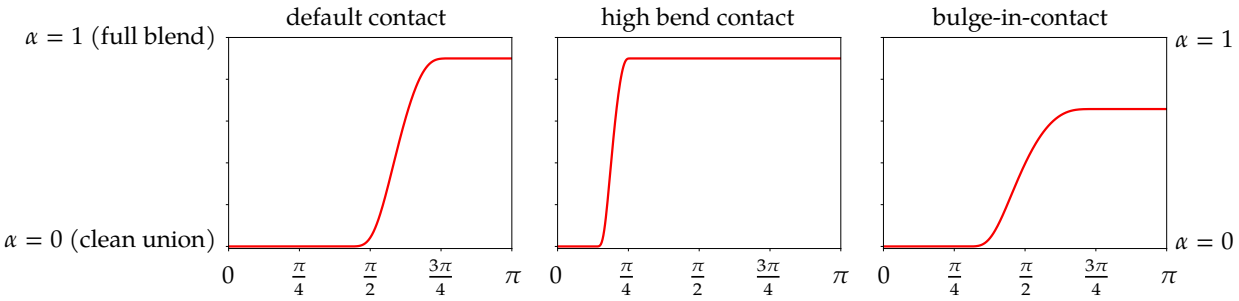
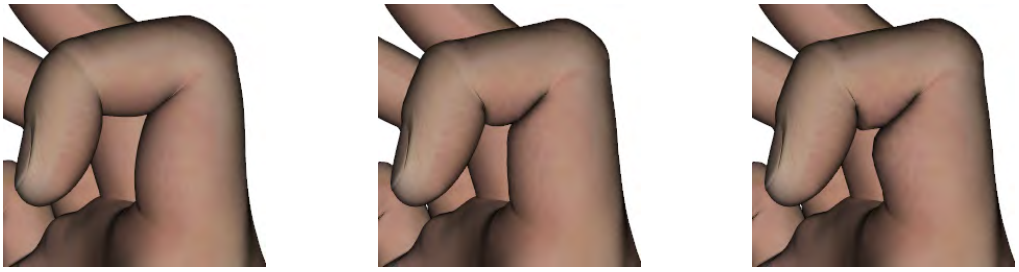


Figure 3.5: Controller functions used for Implicit Skinning.

The definition of the skin part field f_j associated to the animation bone j is thus a compact-support HRBF: $f_j(\mathbf{p}) = K(f_{\mathrm{H}}(\mathbf{p}))$, i.e. a 15-degree polynomial. To speed up the evaluation of the function, each field f_j is sampled in a 3D grid. Subsequent calls to the value of f_j are computed by trilinear interpolation inside the sampled grid.

3.1.2 Composition operators

Each vertex group associated to bone j now has a compact-support field function f_j; all these functions have to be combined together to provide a unique skin field F using composition operators. In skinning, the contact operator defined in Section 2.3.4 is employed for its ability to model a contact surface between two intersecting body parts, such as the arm and forearm (Figure 3.4).



(a) DQS (b) IS with contact operator (c) IS with bulge operator

Figure 3.6: Implicit Skinning on fingers, showing geometric skinning with no contact handling (a), the surface obtained with the contact operator (b), and the extra volume gained with the bulge operator (c).

(Pictures from [Vai+14])

For this operator, the controller function κ, which maps the gradients angle with the operator's parameter α is set to a default function shown in Figure 3.5. This default controller smoothly interpolates between the union and the contact around a gradients angle of $\theta = \frac{\pi}{2}$. This controller works for most joints of the character. For joints that can bend to sharp angles (such as the elbow or the knee), a sharper transition, starting at wider angles, will increase the consistency of the contact surface. Another option is to use the bulge-in-contact operator, which inflates the surfaces in the contact areas. This behaviour increases the plausibility of the deformed surface at the finger joints, as illustrated by Figure 3.6. In this case, a specific controller is specified for the bulge operator.

The final surface building algorithm proceeds recursively, following the tree structure of the skeleton. Functions associated with leaf bones are first composed with their parent bone's function. Then, the resulting composed function is composed with its parent, and so on. Typically, in humanoid skeletons, some joints have more than one child: for example the hips or the neck. Because the composition operators are binary, separate branches are assembled two by two, until all branches are taken into account. This process is illustrated in Figure 3.7. At the end, the topmost composition operator defines the scalar field F which represents the whole skin. With this definition of F, the evaluation of its value at a given point $\mathbf{p}$ might appear costly, because it would require the evaluation of each primitive function $f_j(\mathbf{p})$, and the recursive composition of all these values with the tree of operators. However, because the f_j are compact-support functions, only the surfaces closest to $\mathbf{p}$ will influence the result, as the other primitives will evaluate to 0. The composition tree also includes a bounding volume hierarchy (BVH) tree to speed up the evaluation by discarding any request at a point outside the node's bounding volume.

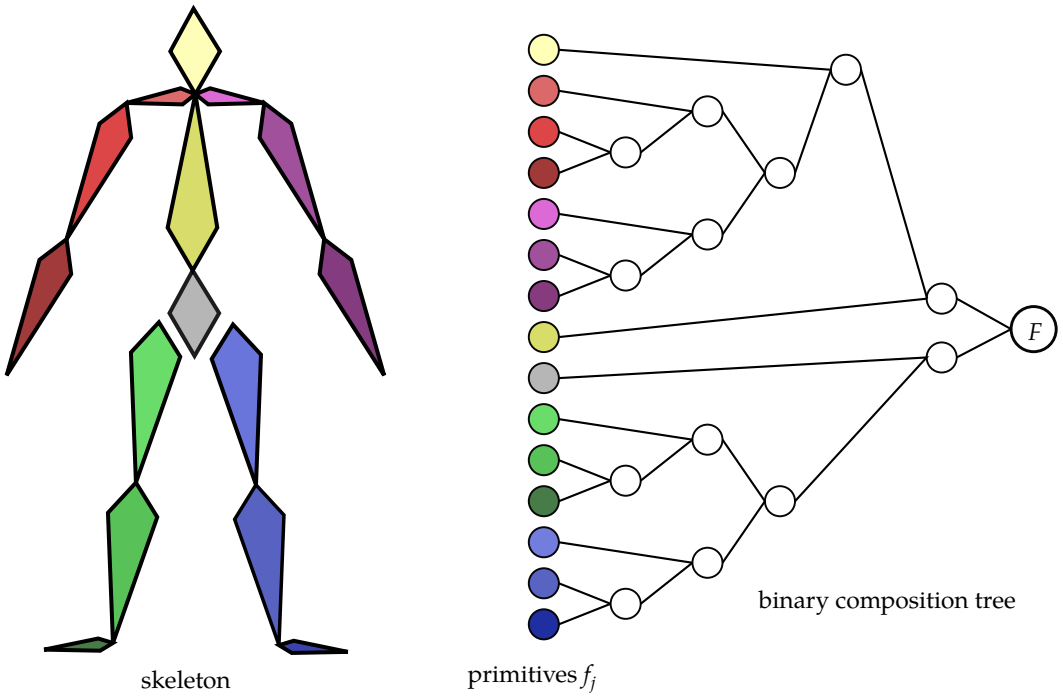


Figure 3.7: Building of the skin composition tree. Each color node correspond to the primitive scalar field f_j associated with skeleton bone j, and each white node is a composition operator. The topmost operator is the skin scalar field F.

3.1.3 Animation of the implicit surface

During animation, at each time step t, the scalar fields are transformed by their associated bone transform $\mathbf{B}_j(t)$. Using the field spatial transformation formula defined in Section 2.4,

$$\begin{aligned} f_j(\mathbf{p}, t) &= f_j\left((\mathbf{B}_j(t))^{-1}\mathbf{p}\right) \\ \nabla f_j(\mathbf{p}, t) &= \mathbf{R}_j(t)\,\nabla f_j\left((\mathbf{B}_j(t))^{-1}\mathbf{p}\right) , \end{aligned}$$

where $\mathbf{R}_j(t)$ is the rotational part of $\mathbf{B}_j(t)$.

This yields a time-varying topmost skin field F_t which represents the recursive composition of the transformed primitive fields. The skin field follows the motion of the skeleton and, thanks to the contact operator, generates a contact surface inside the two colliding fields, as shown in Figure 3.4(a).

Algorithm 3.1 PROJECTION: Implicit surface tracking step

input: a scalar field F_t, a vertex $\mathbf{v}_i$ tracking its iso-value e_i and a step factor λ
output: an updated position $\mathbf{v}_i^{(p)}$

$f_i = F_t(\mathbf{v}_i)$
$\vec{\mathbf{g_i}} = \nabla F_t(\mathbf{v_i})$
$d = f_i - e_i$
if $(|d| < \epsilon)$ **then** *// Stop case: the point is already at e_i*
  return $\mathbf{v}_i$
end if
$\vec{\mathbf{h}} = -\lambda d \frac{\vec{\mathbf{g_i}}}{\|\vec{\mathbf{g_i}}\|^2}$ *// Compute displacement vector*
$\mathbf{v'}_i = \mathbf{v}_i + \vec{\mathbf{h}}$ *// Advance the point along the displacement direction*
$d' = f(\mathbf{v'}_i) - e_i$
if $(d(d - d') < 0)$ **then** *// Stop case: the iso-value difference has increased*
  return $\mathbf{v}_i$
end if
if $(dd' < 0)$ **then** *// Stop case: intersection found along the displacement direction*
  return DICHOTOMY_SEARCH $(f, \mathbf{v}_i, \mathbf{v'}_i, e_i)$
end if
return $\mathbf{v'}_i$ *// Advance by the full length of $\vec{\mathbf{h}}$*

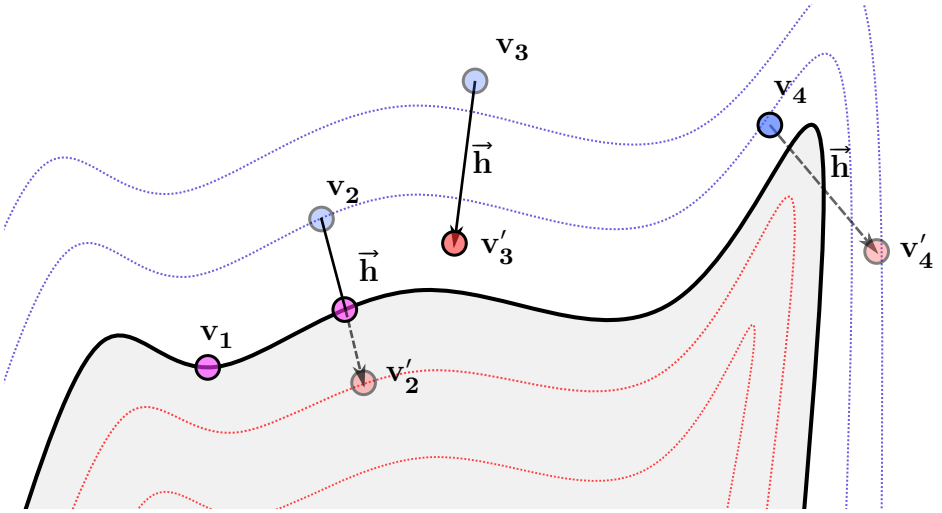


Figure 3.8: Implicit Skinning projection step, illustrating all possible cases. $\mathbf{v_1}$ is already on its target iso-surface, and does not move. $\mathbf{v_2}$ is advanced along the gradient direction and crosses its target iso-surface. The points moves to the intersection. $\mathbf{v_3}$ is advanced along the gradient, but falls short of the target surface. The points moves the full length of the displacement vector $\vec{\mathbf{h}}$. $\mathbf{v_4}$ is advanced along the gradient, but its new position is further away from the surface. It remains at the same position.

3.2 Surface tracking

As seen in Figure 3.2, the skin field's surface does not perfectly matches the mesh, but instead attempts to capture the overall low-frequency shape. This field is used as a guideline to correct the shortcomings of a geometric skinning approach.

Before animating the mesh, the value of the field $e_i = F_{t=0}(\mathbf{v_{ref}}_i)$ is evaluated at each vertex of the mesh in its reference position. During animation, the vertices of the mesh are first deformed using a geometric skinning method to a starting position $\mathbf{v}_i$. Comparing the value of the animated field F_t at these new positions to the initial value identifies which vertices positions need to be corrected.

These vertices are displaced to pull them back to their correct iso-surface of the scalar field F_t; in essence each vertex $\mathbf{v}_i$ is tracking the e_i iso-surface S_{e_i} of F_t. This process, described in Algorithm 3.1, is implemented as an iterative gradient descent, until the target iso-value is reached. At each tracking step, the value and gradient of F_t at the current position v_i is evaluated, and a *displacement vector* $\vec{\mathbf{h}}$ is computed. This vector's direction is given by the gradient of F_t and its length is proportional to the difference d between the current value $F_t(\mathbf{v}_i)$ and the target value e_i, and a step scale factor λ which controls the speed at which the points converge to their iso-surface.

If the displacement of $\mathbf{v}_i$ crosses the target iso-surface, the precise location of the intersection is located by dichotomy search, and the point is only moved to this intersection. Otherwise, the point is moved along the displacement vector, but only if the new position has a value closer to the target. If the move would increase the target value difference, the point stays in the same position, waiting for a subsequent step (with a smaller step factor λ) to find the correct moving position. Figure 3.8 gives an illustration of the possible cases of the tracking step.

Being a gradient-descent step, the tracking will fail to compute a displacement direction if the gradient is 0 at the current position of the vertex. For that reason, it is crucial to maintain a field without singular points near the surface of the mesh, or some vertices can get trapped. HRBFs used in modelling character body parts usually have singular points near their axis (i.e. the skeleton bone). This is averted by the use of DQS as a first geometric skinning method as the candy-wrapper artefact of LBS would bring the point too close to the central axis. In contrast, DQS has a natural tendency to push the vertices outwards which prevents the vertices from being to close from the inside. Additionally, since F_t is a compact support function, the field and gradient are both null outside of its support. The choice of the support radius R (as defined in Section 3.1.1) is therefore important to prevent the vertices from getting stuck too far away from the surface.

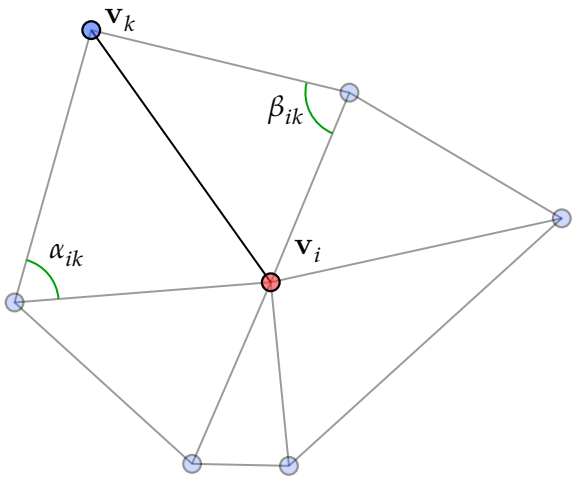


Figure 3.9: Neighbourhood of vertex $\mathbf{v}_i$ and angles α_{ik} and β_{ik} involved in the computation of cotangent weight c_{ik}.

By following the gradient, each vertex will converge towards the nearest point on their target iso-surface. Yet, as mentioned is Section 2.5, this only corrects the vertex *normal* position without considering problems that arise in the *tangential* direction. In practice, the vertices will slide along the surface, distorting the mesh's triangles and yielding a visually unappealing deformation, calling for a tangential scheme to avoid these distortions.

3.3 Tangential relaxation and skin elasticity

To mitigate the distortions, the structure of the mesh can be exploited to correct the position of the vertices $\mathbf{v}_i^{(\mathrm{p})}$ after the projection step in the *tangent plane* $\mathcal{T}$, i.e. the plane normal to $\nabla F\left(\mathbf{v}_i^{(\mathrm{p})}\right)$. Vaillant et al. [Vai+13] initially proposed to use a relaxation scheme based on barycentric coordinates but later moved on [Vai+14] to a formulation based on the as-rigid-as-possible (ARAP) energy introduced by Sorkine and Alexa [SA07].

The ARAP energy is a function defined on the mesh which measures its deformation relatively to its reference shape. For any vertex $\mathbf{v}_i$, the local deformation energy E_i is defined as:

$$E_i = \sum_{\mathbf{v}_k \in \mathcal{N}(\mathbf{v}_i)} c_{ik} \left\| (\mathbf{v}_i - \mathbf{v}_k) - \mathbf{R}_i(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) \right\|^2$$

where $\mathcal{N}(\mathbf{v}_i)$ is the *neighbourhood* of $\mathbf{v}_i$, that is to say, the vertices immediately connected to $\mathbf{v}_i$ through an edge of the mesh, and $\mathbf{R}_i$ is a rotation matrix. The weights c_{ik} are the classical *cotangent weights* of the discrete Laplace-Beltrami operator [Bot+10], i.e. $c_{ik} = \frac{1}{2}(\cot(\alpha_{ik}) + \cot(\beta_{ik}))$, with α_{ik} and β_{ik} the angles opposite to the edge $[\mathbf{v}_i\mathbf{v}_k]$ (see Figure 3.9). The weights c_{ij} are precomputed for all neighbouring vertices on the reference mesh. The total

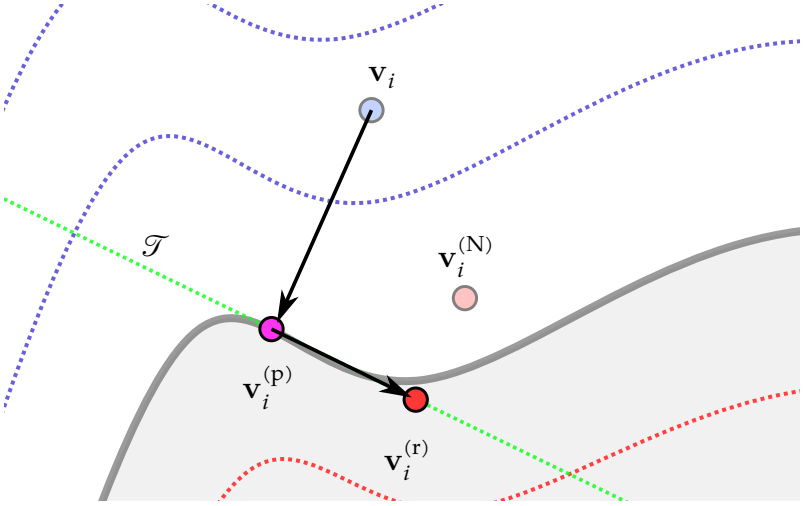


Figure 3.10: Projection and tangential relaxation. Vertex $\mathbf{v}_i$ is projected to $\mathbf{v}_i^{(\mathrm{p})}$ lying on the iso-surface, then is moved by the relaxation to $\mathbf{v}_i^{(\mathrm{r})}$, which is the projection on the tangent plane of the result of N_A ARAP Jacobi iterations $\mathbf{v}_i^{(\mathrm{N_A})}$

ARAP energy E is the sum of all individual vertex deformation energies over the whole mesh:

$$E = \sum_{i=0}^{n-1} E_i \ .$$

In the standard ARAP formulation, the deformation energy is minimised by iterating a two step process. The first step finds the optimal rotation matrices $\mathbf{R}_i$ which transform $\mathcal{N}(\mathbf{v}_{\mathbf{ref}\,i})$ into $\mathcal{N}(\mathbf{v}_i)$. The second step moves the vertices $\mathbf{v}_i$ to minimize the energy. As the energy is quadratic in terms of the vertex positions, the local minimum can be found with Newton's method by solving a $3n \times 3n$ linear system.

Because of the first step, the overall algorithm is non-linear and thus requires an iterative approach to find a satisfying solution. Additionally, computing the optimal rotations requires a polar decomposition of the Jacobian of the local deformation per vertex, which is a costly operation. Moreover, this estimation is prone to introduce errors in the subsequent mesh such as triangle inversion or edge collapse [BN07]. However, in the case of skinning, this expensive step can be avoided by using the skeleton transforms as an additional source of information. The rotations $\mathbf{R}_i$ are set *a priori* to the rotational part of the dual-quaternion transform from P_{ref} to P affecting $\mathbf{v}_i$, in other words, a blend of the skeleton bones transform for which $\mathbf{v}_i$ has a non-zero skinning weight (as described in Section 1.4.2).

As said above, with the rotation fixed, it is possible to compute the optimal positions of the vertices $\hat{\mathbf{v}}_i$ from their position after the projection step $\mathbf{v}_i^{(\mathrm{p})}$ by solving the linear

system:

$$\begin{pmatrix} \hat{\mathbf{v}_0} \\ \hat{\mathbf{v}_1} \\ \vdots \\ \hat{\mathbf{v}_{n-1}} \end{pmatrix} = \begin{pmatrix} \mathbf{v}_0^{(\mathrm{p})} \\ \mathbf{v}_1^{(\mathrm{p})} \\ \vdots \\ \mathbf{v}_{n-1}^{(\mathrm{p})} \end{pmatrix} - \mathbf{H}_E{}^{-1} \nabla E \ ,$$

where $\mathbf{H}_E$ is the Hessian of E.

The chosen implementation for the relaxation step is the Jacobi method, an iterative method converging to the optimal solution. At a given Jacobi step j, the positions of the vertices is updated with the following formula

$$\mathbf{v}_i^{(\mathrm{j}+1)} = \sum_{k \in \mathcal{N}(\mathbf{v}_i)} \overline{c_{ik}} \mathbf{v}_k^{(\mathrm{j})} + \mathbf{b}_i \ ,$$

where $\overline{c_{ik}}$ are the normalized cotangent weights

$$\overline{c_{ik}} = \frac{c_{ik}}{\sum_{k=0}^{n} c_{ik}} \ ,$$

and $\mathbf{b_i}$ is defined as

$$\frac{1}{2} \sum_{k=0}^{n} \overline{c_{ik}} (\mathbf{R}_i + \mathbf{R}_k)(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) \ .$$

The derivation of the previous expressions is detailed in Appendix A.1.

Yet, when moving the vertex towards the computed optimal position, it could move away from its iso-surface, defeating the purpose of the projection step. The relaxation is therefore constrained to the tangent plane $\mathcal{T}$, by projecting the last point $\mathbf{v}_i^{(\mathrm{N_A})}$ onto the tangent plane, as shown in Figure 3.10. The resulting point $\mathbf{v}_i^{(\mathrm{r})}$ is the output of the ARAP step described in Algorithm 3.2.

The tangent plane projection is a linearisation of the iso-surface constraint, and thus does not guarantees that $\mathbf{v}_i^{(\mathrm{r})}$ will satisfy $F_t(\mathbf{v}_i^{(\mathrm{r})}) = e_i$. This suggests to use an iterative process interleaving the projection step with the relaxation step, repeated until the point has reached a position that both minimizes the deformation energy and is on the target iso-surface.

3.4 Implicit Skinning algorithm

The success of Implicit Skinning requires a careful balance between the surface tracking and the relaxation scheme. The progress of the projection step towards the target iso-

Algorithm 3.2 ARAP_ITER: Jacobi iterations of the ARAP energy minimization.

input: starting positions of vertices $\mathbf{v}_i^{(0)}$, reference positions $\mathbf{v}_{\mathbf{ref}\,i}$, rotations $\mathbf{R}_i$, normalized cotangent weights $\overline{c_{ik}}$, and number of ARAP steps N_A
output: updated position $\mathbf{v}_i^{(\mathrm{r})}$
$\mathbf{b}_i = \frac{1}{2}\sum_{k=0}^{n} \overline{c_{ik}}(\mathbf{R}_i + \mathbf{R}_k)(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k})$
for $j = 0$ **to** N_A **do**
  $\mathbf{v}_i^{(\mathrm{j}+1)} = \mathbf{b}_i + \sum_{k\in\mathcal{N}(\mathbf{v}_i)}^{n} \overline{c_{ik}}\mathbf{v}_k^{(\mathrm{j})}$
end for
return $\mathbf{v}_i^{(\mathrm{r})} = \textsc{project}\,(\mathbf{v}_i^{(\mathrm{N_A})}, \mathscr{T})$ *// project the last point on tangent plane*

surface is controlled by the step scale parameter λ, while the convergence of the ARAP optimization is driven by the number of ARAP Jacobi steps N_A.

The main algorithm consists in a central loop which successively applies the projection step and the ARAP Jacobi steps, until the prescribed number of iterations is reached. For each of these central loop iterations, a value of λ and N_A is supplied, determining the balance between the normal and the tangential progression of the vertices. The sequence of λ is decreasing, yielding smaller and smaller step sizes. This behaviour stabilizes the march of the vertices, first by moving in large steps and then by advancing towards their target surface with a higher precision. Between the projection step, only one ARAP Jacobi iteration is run (i.e. $N_A = 1$). This way the algorithm prioritizes the normal displacement to the tangential displacement. After the central loop has run for the prescribed number of iterations, tangential ARAP Jacobi iterations are run until the mesh converges to the optimal solution.

Algorithm 3.3 Implicit Skinning algorithm

input: a vertex $\mathbf{v}_{\mathbf{ref}\,i}$, a sequence of λ and N_A
output: a final skinning position $\mathbf{v}_i^{(\mathrm{final})}$
$\mathbf{v}_i = \mathbf{T}_{\hat{\mathbf{q}}_i}\mathbf{v}_{\mathbf{ref}\,i}$ *// Initial DQS solution*
for $n = 0$ **to** N **do**
  $\mathbf{v}_i^{(\mathrm{p})} = \textsc{projection\_step}\,(F_t, \mathbf{v}_i, e_i, \lambda_n)$ *// Projection step*
  $\mathbf{v}_i^{(\mathrm{r})} = \textsc{arap\_iter}(\mathbf{H}, \mathbf{v}_i^{(\mathrm{p})}, N_{A,n})$ *// ARAP Jacobi step*
  $\mathbf{v}_i = \mathbf{v}_i^{(\mathrm{r})}$ *// Update the positions and start over*
end for
repeat
  $\mathbf{v}_i = \textsc{arap\_iter}(\mathbf{H}, \mathbf{v}_i, 1)$
until convergence

3.4.1 Time-dependency

Implicit Skinning is a corrective algorithm: it starts with an initial solution and moves the vertices to a position that fits the iso-surface constraint and the ARAP energy minimization.

In practice, a better consistency can be achieved at the cost of the independence of history. In this case, for the initial solution DQS is not applied from the reference vertices, but from the vertices of the *previous* frame of animation. In other words, the previous frame is treated as the reference mesh for the next frame. This time-dependency means that the result of skinning depends not only on the current pose, but also on the previous poses: two different animations will yield different results even if they reach the exact same pose. This has a positive effect on performance, because it limits the number of projection and ARAP iteration steps, as the vertices start closer from the optimal solution. In addition to the performance gain, it renders the result of the skinning almost insensitive to the skinning weights distribution. In their implementation, Vaillant et al. [Vai+14] achieve smooth deformations even with rigid weights (i.e. the weight of the nearest bone is 1 and all the others are 0). The loss of history-independence can however be a problem to attain reproducible results interactively. The method is still deterministic when playing the same animation twice.

3.5 Discussion

Implicit Skinning greatly improves the results of its initial geometric solution by focusing on the modelling of the joints. Because the points track their iso-surface, collapsing or bulging at joints is averted. Moreover, the use of operators to generate a contact surface on which the vertices project resolves self-penetration of the mesh, a problem that was difficult to avoid with purely geometric methods.

Despite its complexity, this method is able to run at interactive frame-rates. Several design and implementation choices of Implicit Skinning are dictated by the performance requirements. Since the function and its gradient are evaluated several times for each vertex, inefficiency in this evaluation can degrade the frame-rate dramatically. As seen in Section 3.1, memory is traded off to increase the function evaluation efficiency by storing the values in a 3D grid, and using a BVH tree for the global field evaluation. Moreover, the two main subroutines of the algorithm can be processed in parallel for each vertex $\mathbf{v}_i$ either on a GPU or on a multithreaded CPU.

On a more theoretical standpoint, the usefulness of the approach, arises from its separation between the volumic effects, which are represented by the implicit surfaces and their interaction, and the surfacic effects, which are computed from the mesh. However,

the modelling approach using HRBFs and contact operators limits the kind of effects that can be represented around the joints. This produces some sort of dynamic effects with the bulge-in-contact operation on the fingers of Figure 3.6(c), but this is a very limited use-case. Except for their rigid transformations, the surfaces do not deform and cannot reproduce dynamic deformations happening in the limbs, in particular, those due to muscles.

However, because the tracking algorithm does not rely on any assumption on the underlying scalar field, it is possible to enrich the implicit skin representation in order to account for dynamic anatomic effects localized in the different parts of the body. Our contribution, described in the next part, is the definition of such a model to simulate the dynamics of muscles on the skin, thus increasing the liveliness of the characters animated with this method.

II

Implicit muscle deformers

4 Implicit muscle models

We admire the skillful construction of the fibers in each muscle; how much more then ought we to admire it in the brain.

— Niels Steensen (1638 – 1686), *Discours sur l'anatomie*

In this chapter, we present an implicit model which can represent muscle shapes. First, we examine the physical properties of real-life muscles that the model must take into account to produce plausible shapes and deformations. Second, we introduce a family of extrusion surfaces (presented in Section 2.2.3) defined by a scalar field function f_M whose iso-surface represents a muscle. We then describe the parameters of this model, how they are constrained to represent plausible muscle shapes and ensure volume preservation. This chapter concludes by a discussion on the practical consequences of the choices made for this model.

4.1 Muscle anatomy

4.1.1 The different types of muscles

The human body has more than 400 muscles divided in two categories. *Skeletal muscles*, or voluntary muscles, set the body in motion by contracting or extending, generating a force that moves the bones of the skeleton. These motions happen either consciously, e.g. when walking, or unconsciously as reflexes, e.g. when maintaining balance. *Involuntary muscles*, on the other hand, are muscles that are only controlled by unconscious processes. They include the walls of the heart and smooth muscle tissues that surround other organs (such as the stomach and intestine). These muscles lie deep in the human body and have no visible effect on the skin. We thus focus our study on skeletal muscles. As an example,

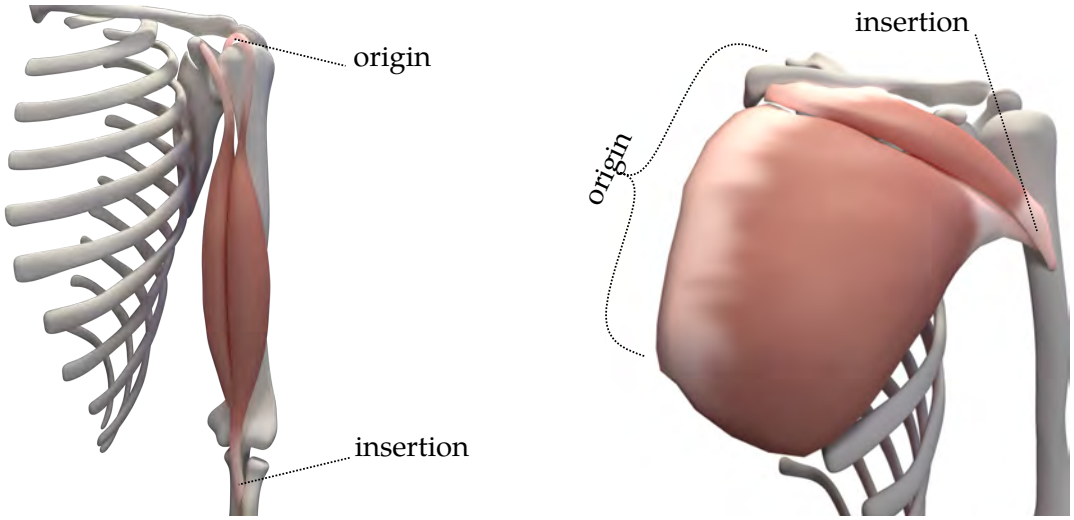
Figure 4.1: A 3D anatomical model of the left arm.

Model by Anatoscope

Figure 4.1 shows all 56 skeletal muscles involved in the motion of the left arm, hand and fingers.

Skeletal muscles exert force on the skeleton through tendons, located at their extremities. Tendons are made of stiff tissue and connect the muscular system to the bones. The attachment which is static during the muscle's contraction is the *origin*, while the attachment which tends to be moved is the *insertion*. Typically, the origin is proximal (i.e. closer to the body's centre of gravity) and the insertion is distal, such as in the biceps and the pectoral of Figure 4.2.

Muscles come in many shapes in the body and are usually classified according to their architecture: the arrangement of fibers within the muscle. Muscles driving large movements generally have fibers running parallel to the muscle axis, yielding the familiar *fusiform* shape found in the biceps, triceps or hamstrings. In more complex muscles, the fibers are arranged diagonally in one, two, or more rows and are known as *pennate* muscles (unipennate, bipennate or even multipennate). The architecture of these various types of muscles produces different physical aspect, as illustrated by Figure 4.3. They nonetheless



(**a**) Biceps (**b**) Pectoral

Figure 4.2: Figure (a): endpoints of the biceps, with its origin on the humerus and its insertion on the radius. Figure (b): endpoints of the pectoral, its origin on the sternum and its endpoint on the humerus.

Model by Anatoscope

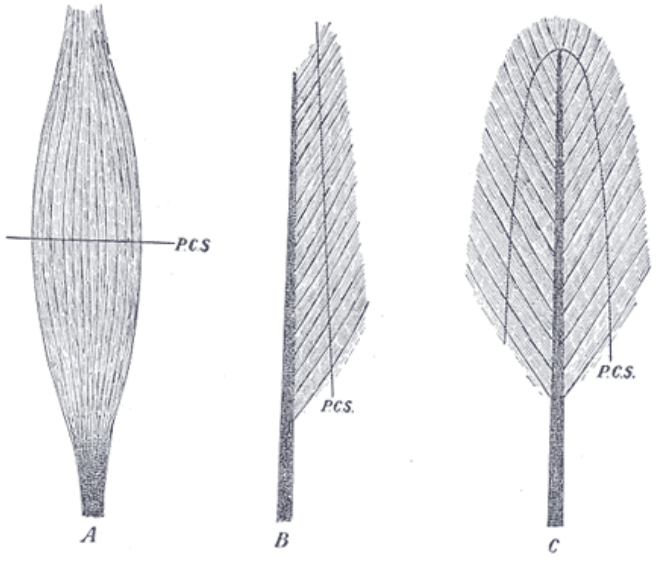


Figure 4.3: The different types of muscles. A: fusiform, B: unipennate, C: bipennate. (PCS: physical cross section).

(Picture from [Gra18])

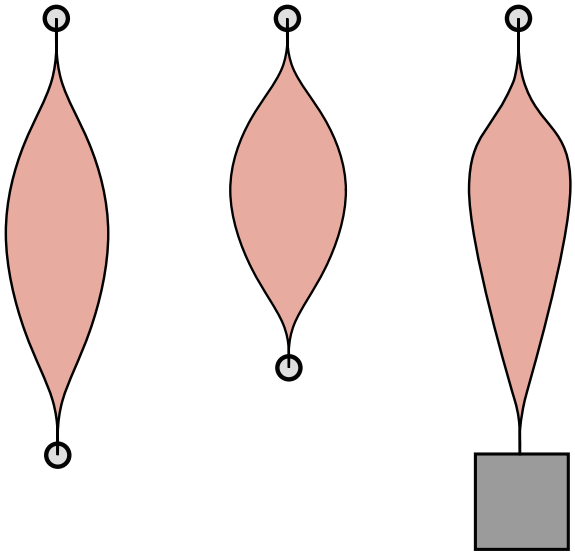


Figure 4.4: Deformation modes of a skeletal muscle. From left to right: muscle in a rest state, isotonic contraction and isometric deformation.

share common shape attributes: a roughly circular cross-section, a wide belly and tapered extremities.

The pectoral shown in Figure 4.2(b) belongs to a separate category: *convergent* muscles. Muscles in this category are characterized by a wide origin and a relatively smaller insertion point. Their cross-section is flat, and their longitudinal shape is roughly triangular.

4.1.2 Deformations

A key property of muscle deformation is the *conservation of volume* [Ste64]. When moving, the shape of the muscle changes, but its volume remains constant. This is, of course, a property that is not valid for very long timescales as muscles grow with physical exercise or shrink with inactivity. For the range of motions we consider in animation, however, this assumption is largely verified.

Muscles have two modes of deformation: *isotonic* and *isometric*. Isotonic deformation happens when the muscle contracts or extends, setting into motion the bones to which it is attached. For example, the biceps will drive the forearm closer to the arm to bend it. Because the muscle's length is shortening, its width will increase, keeping its volume constant, creating the familiar bulging of the arm. In isometric deformation, the muscle's length does not change, but the tension of the muscle increases, for example to counteract the weight of an heavy object. This phenomenon is known as *activation*. In that case, the muscle's shape changes to increase the force exerted by the muscle but the endpoints stay static. These two modes are illustrated by Figure 4.4. Except in dedicated physical

exercises, the deformations observed during human motion are often a combination of these two modes.

4.1.3 A muscle model for skinning

Skeletal muscles are made of fibers which can contract and expand. In real life, the muscles exert force on the bones to set them in motion. However, in the case of animation, the skeleton rig drives the position of the character. Our goal is therefore not to model the physical behaviour of the muscles as in physics-based animation methods (as described in Section 1.2), but to define a model which produces shapes similar to the shapes of real muscles. Our muscles have to be set up on the reference pose of the mesh, typically by their end points and initial shape, and deform with the animation skeleton according to a set of rules inspired from real muscles. This model should be able to account for isotonic deformations caused by the skeleton motion, but also offer the possibility to be activated to reflect isometric deformations. As seen above, volume conservation is a crucial constraint which affects the muscle's shape and it must be preserved during deformation.

Our muscles should be represented by an implicit surface, as our goal is to integrate them within the implicit skin representation. We thus seek to define a continuously differentiable scalar field f_M which can produce the different shapes of muscles described above. We define several parameters specifying the shapes of muscles at rest and in the deformation modes discussed in this section, maintaining volume conservation as the shape of the muscle deforms over time. Given the roughly circular cross-section of most muscles, the surfaces we define are sweep surfaces around a central axis representing the line of action of the muscle. Additionally, we model the elasticity of muscle tissue by enabling deformations on this central axis.

4.2 Muscle model

A muscle is defined as a 3D scalar field $f_M : \mathbb{R}^3 \to \mathbb{R}$ which represents an *extrusion surface*, as defined in Section 2.2.3. The axis curve is a polyline $\mathscr{C}$, which defines the central axis of the muscle, and the shape is defined by a profile function R. A summary of notations is given in Figure 4.5.

The initial state of the axis is defined by positioning the two end points, origin and insertion, and attaching them to the animation skeleton. The shape of the muscle is controlled by the parameters of the profile function R.

The computation of the value of the scalar field f_M at any given point $\mathbf{q} \in \mathbb{R}^3$ is broken down in the following steps:

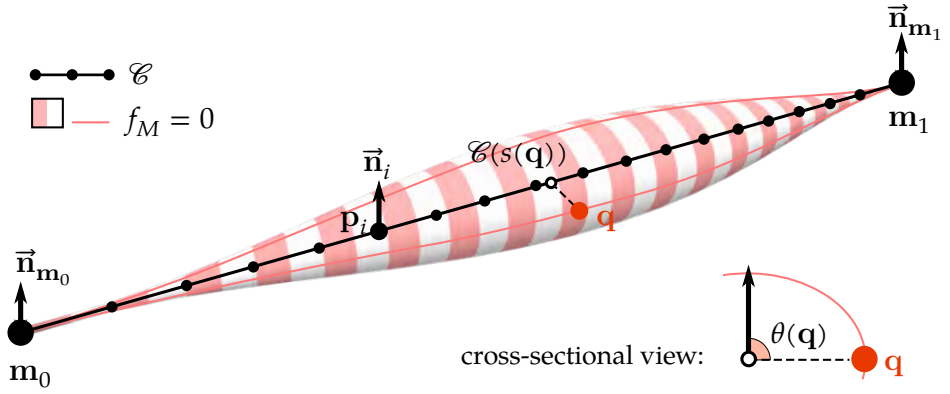


Figure 4.5: Schematic view of muscle primitive with notations.

- construction of the polyline $\mathscr{C}$,
- projection of $\mathbf{q}$ on $\mathscr{C}$, yielding the projection point $\mathbf{h} = \text{proj}(\mathbf{q})$ with its curve coordinate s,
- evaluation of the normal on the axis at s and the angle θ within the normal plane,
- evaluation of the sweeping profile function $R(s, \theta)$.

The value of the field at $\mathbf{q}$ is then given by

$$f_M(\mathbf{q}) = \left\|\overrightarrow{\mathbf{qh}}\right\| - R(s, \theta) \ .$$

4.2.1 Construction of the central axis

The muscle is defined by its two endpoints $\mathbf{m}_0$ and $\mathbf{m}_1$, each attached to an animation bone: they move kinematically during the animation. The segment $[\mathbf{m}_0\mathbf{m}_1]$ is divided into N_M parts with intermediate control points $\mathbf{p}_i$. The resulting polyline $\mathscr{C}$ is parametrized by $s \in [0, 1]$, and we note $s(\mathbf{q})$ the curvilinear parameter of the projection $\mathbf{h}$ of $\mathbf{q}$ on $\mathscr{C}$. While the control points $\mathbf{p}_i$ start on the straight line between the end points, they are allowed to move during the animation, as described in Chapter 5.

In order to define the polar profile along the central axis, the polyline is oriented at each endpoint by given normal vectors $\vec{\mathbf{n}}_{\mathbf{m}_0}$ and $\vec{\mathbf{n}}_{\mathbf{m}_1}$. From these two endpoint normals, we define a normal vector at each point of $\mathscr{C}$ as follows. We associate to each control point of the polyline $\mathbf{p}_i$ a normal vector $\vec{\mathbf{n}}_i$. Firstly, we interpolate the end points normal by spherical linear interpolation at each intermediate point $\mathbf{p}_i$. In order to account for the local deformation of the polyline, these interpolated vectors are projected on the plane defined by $\mathbf{p}_{i-1}$, $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ when the points are not aligned. If the three points are aligned, we interpolate the normals of the two closest points for which they are defined.

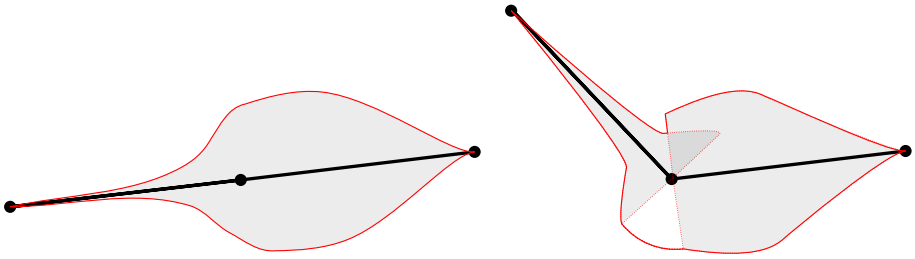


Figure 4.6: Discontinuities of the orthogonal projection. Left: a muscle profile on an undeformed axis. Right: Surface discontinuities appear after the deformation of the axis due to the orthogonal projection.

Secondly, the polyline normal at $s(\mathbf{q})$ is then computed by spherical linear interpolation of the control points normals $\vec{\mathbf{n}}_i, \vec{\mathbf{n}}_{i+1}$ of the polyline segment it belongs to.

4.2.2 Projection on the axis

In the evaluation of the function, the projection operation is crucial to the continuity of the implicit surface. Projecting the query point orthogonally on the closest segment yields discontinuities when the polyline deforms, as is illustrated by Figure 4.6. Discontinuities appear in the inner part of the dihedral angles of two consecutive segments, where the discrepancy in distance caused by the variation of the profile's radius leads to a sudden jump. This is especially prominent if there is a local twist defined on the segment, as the orientation of the polar curve, which depends on the normal, will not vary continuously. On the other hand, the outer part of the dihedral angles always projects to a spherical wedge, maintaining a continuous surface. This issue, which arises from the angles of the polyline, is similar to the problem of interpolating a smooth normal on a polygon mesh [KVS99; Pan+13].

We solve this problem by detecting when the point is in the inside part of an angle, and reparametrizing the projection in this case. We start by computing the closest point $\mathbf{h}_i$ from $\mathbf{q}$ to each segment $[\mathbf{p}_i\mathbf{p}_{i+1}]$. Standard orthogonal projection would then return the nearest point on the polyline , i.e. the point which minimizes $\left\|\overrightarrow{\mathbf{qh_i}}\right\|$, creating the discontinuities mentioned above. In our case, we detect cases when the point is in the inner part of the dihedra between two successive segment by examining if the projected $\mathbf{h}_i$ point lies on the *interior* of the segment or on its extremities. The re-projection is applied when the query point's projection on two consecutive segments is an interior point, as shown in Figure 4.7.

If the point is indeed in the problematic area, we reparametrize the projection point using the cotangent of the angles λ_i and λ_{i+1} formed between the point and the two segments (as depicted in Figure 4.8). This process is summarized in Algorithm 4.1. The

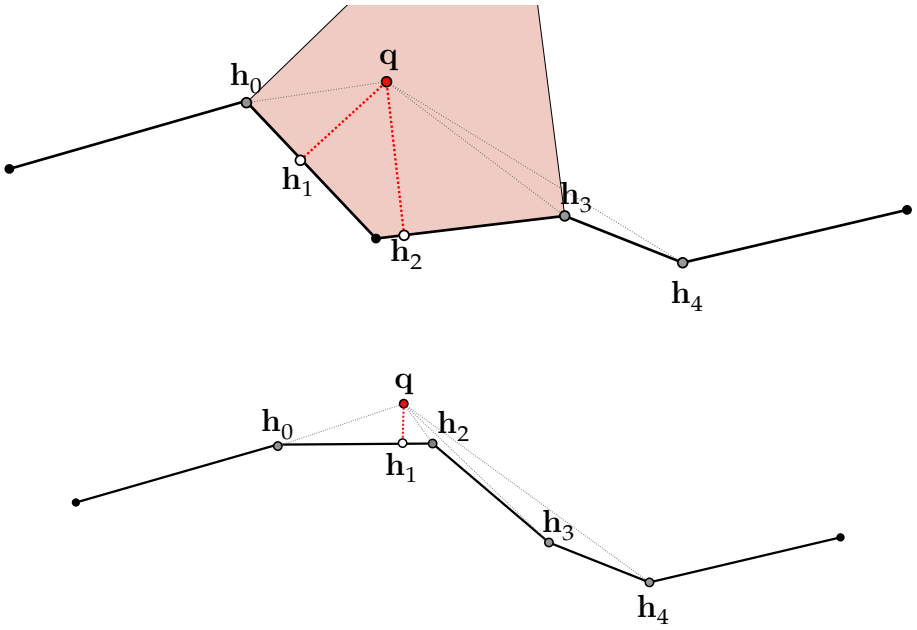


Figure 4.7: Projection on a polyline. The query point $\mathbf{q}$ is projected on each segment of the polyline. Each point $\mathbf{h}_i$ is the nearest point from $\mathbf{q}$ on segment i. Top figure: When two consecutive nearest points $\mathbf{h}_1$ and $\mathbf{h}_2$ belong to the *interior* of their segment, the query point is in the inner part of the dihedra (red area). Bottom figure: when the query point is in the outer part of an angle, at most one of the nearest point will not be an extremity of its segment.

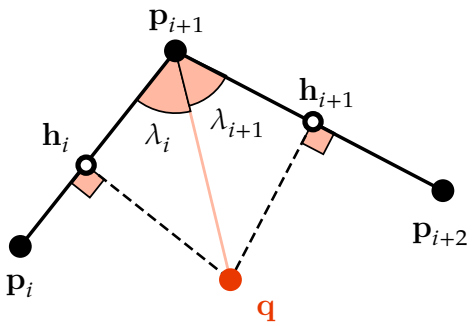
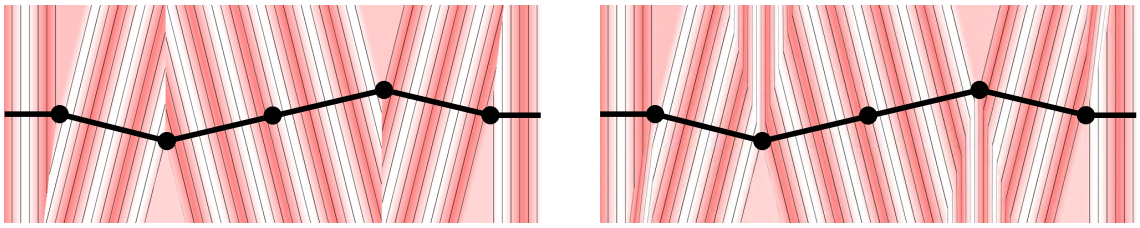


Figure 4.8: Reparametrization of inner angle. The projection of $\mathbf{q}$ is computed by weighting the curvilinear coordinate of the projected points on each segment $\mathbf{h}_i$ and $\mathbf{h}_{i+1}$ by the cotangent of the associated angles λ_i and λ_{i+1}.

Algorithm 4.1 Re-parametrization of the polyline projection

input: A polyline $\mathscr{C}$ defined by its control points $\mathbf{p}_i$ and a query point $\mathbf{q}$
output: A projected point $\mathbf{h}$ on $\mathscr{C}$ and the corresponding parameter $s(\mathbf{q})$
for all segments $[\mathbf{p}_i, \mathbf{p}_{i+1}]$ of $\mathscr{C}$ **do**
  compute $\mathbf{h}_i$, the nearest point from $\mathbf{q}$ to the segment and s_i its parameter
end for
Let $\mathbf{h}^*$ be the nearest point from $\mathbf{q}$ among all $\mathbf{h}_i$, and s^* its parameter.
// Check if the point projects on the interior of two consecutive segments
if $\exists\ i$ such as $((\mathbf{h}^* = \mathbf{h}_i$ or $\mathbf{h}^* = \mathbf{h}_{i+1})$ **and** (none of $\mathbf{h}_i, \mathbf{h}_{i+1}$ belongs to $\{\mathbf{p}_0, \dots \mathbf{p}_{N_M}\}$ $))$
then
  $s(\mathbf{q}) = \frac{s_i \cot\lambda_i + s_{i+1}\cot\lambda_{i+1}}{\cot\lambda_i + \cot\lambda_{i+1}}$
else
  $s(\mathbf{q}) = s^*$
end if
return $\mathbf{h} = \mathscr{C}(s(\mathbf{q}))$



Orthogonal projection | Our projection operator

Figure 4.9: Comparison between standard point-to-segment orthogonal projection, and ours. Colors are computed from the curvilinear coordinates $s(\mathbf{q})$. Note the discontinuities appearing when using standard orthogonal projection are avoided with our operator.

projection algorithm guarantees a smooth transition of the projection instead of a jump from the area of influence of one segment to the next. The result of this reprojection is shown in Figure 4.9.

4.2.3 Function evaluation

The extrusion surface is defined by sweeping a polar curve over the central axis $\mathscr{C}$. To describe the shape we specify a radial function $R(\mathbf{q})$ which depends on:

- The curvilinear parameter $s(\mathbf{q})$,
- The angle $\theta(\mathbf{q})$ between the polyline normal at $s(\mathbf{q})$ and the vector $\overrightarrow{\mathbf{qh}}$.

We specify R as a separable function in each parameter:

$$R(\mathbf{q}) = w\Phi(s(\mathbf{q}))r(\theta(\mathbf{q})).$$

This definition distinguishes between the shape of cross-section of the muscle, defined by $r(\theta)$, and the evolution of its width along its axis, specified with $\Phi(s)$. The separability is also useful to simplify the evaluation of the volume of the muscle to ensure its conservation, as will be shown in the next section.

4.3 Shape parameters and volume preservation

The model presented above is defined by several parameters which control the shape of the muscle:

- The central polyline $\mathscr{C}$,
- The width scale factor w,
- The profile function Φ,
- The cross-section function r.

These parameters evolve during the animation as the result of the motion of the character or as prescribed by an animator. We first show how they must be constrained in order to model volume preservation. We then define functions which respect these constraints and show how to use them to model the two deformation modes of the muscle.

4.3.1 Evaluation of volume

When $\mathscr{C}$ is a straight line, we can evaluate the volume of the muscle in cylindrical coordinates (ρ, θ, s) as

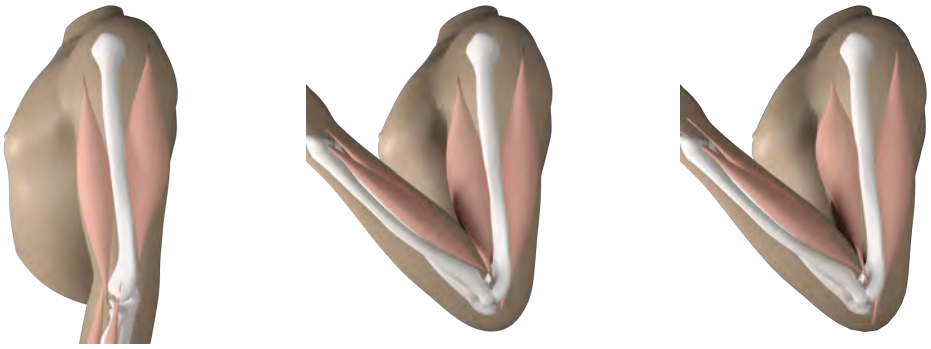
$$V = \iiint \mathrm{d}\rho\ \rho \mathrm{d}\theta\ l \mathrm{d}s\ ,$$

where l is the total length of the polyline, and ρ is the radial integration variable ($\rho \in [0, R(s,\theta)]$). This can be further developed by integrating successively each variable:

$$V = \int_{s=0}^{1} \int_{\theta=0}^{2\pi} \int_{\rho=0}^{R(s,\theta)} \rho \mathrm{d}\rho\ \mathrm{d}\theta\ l \mathrm{d}s\ .$$

Integrating in ρ yields:

$$V = l \int_{s=0}^{1} \int_{\theta=0}^{2\pi} \frac{(R(s,\theta))^2}{2} \mathrm{d}\theta \mathrm{d}s\ .$$



(**a**) At rest (**b**) Contracted (**c**) Contracted and activated

Figure 4.10: Effect of muscle contraction on the muscle shape. Figure (a): rest shape of the muscle. Figure (b): shape of the contracted muscle. The shortening of the axis caused the width of the muscle to increase to preserve volume. Figure (c): shape of the contracted muscle at full activation.

Using the separability of R:

$$V = w^2 l \int_{s=0}^{1} (\Phi(s))^2 \, \mathrm{d}s \int_{\theta=0}^{2\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta \, .$$

We thus require that the profile function Φ and the cross-section function r each satisfy:

$$\int_{s=0}^{1} (\Phi(s))^2 \, \mathrm{d}s = \text{constant} \, , \tag{4.1}$$

$$\int_{\theta=0}^{2\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta = \text{constant.} \tag{4.2}$$

This constant can be arbitrarily chosen by scaling the functions with a multiplicative factor, but we set it to 1 to further simplify the calculus.

Under these conditions, the muscle's volume can be written as

$$V = \pi w^2 l \, . \tag{4.3}$$

A detailed derivation is exposed in Appendix A.2. As stated in Section 4.2.1, during the animation, the muscle endpoints will move with their respective animation bones. This may cause the axis to shorten or lengthen, l will change to a new value l'. Equation (4.3) shows that to keep the volume constant, w must be changed to a new value w' such that

$$w' = w\sqrt{\frac{l}{l'}} \, .$$

This rule causes the muscle to inflate when it contracts and shrink when it is stretched while preserving its volume, thus representing isotonic contraction, as can be seen in Figure 4.10.

4.3.2 Shape profile and activation

The profile function Φ must be able to represent various tapered shapes to model the different muscles of the human body. Additionally, it must be capable of interpolating smoothly between the rest shape and the activated shape of the muscle during isometric deformations. To this effect, we introduce a definition of Φ inspired by the beta probability distribution:

$$\Phi(s) = \phi(\alpha, \beta; s),$$

where α and β are scalar parameters controlling the shape of the profile.

As per Equation (4.1), the conservation of volume is guaranteed only if the integral of the square of Φ is always constant, regardless of α and β. We thus define ϕ as a function of unit norm in the $\mathcal{L}^2$ space of square-integrable functions:

$$\phi(\alpha, \beta, s) = \frac{\phi_0(\alpha, \beta; s)}{\|\phi_0(\alpha, \beta; s)\|_2},$$

where ϕ_0 is defined for $s \in [0, 1]$ as:

$$\phi_0(\alpha, \beta; s) = s^{\alpha-1}(1-s)^{\beta-1}.$$

As such $\phi(\alpha, \beta; s)$ can be explicitly written as

$$\phi(\alpha, \beta; s) = \frac{s^{\alpha-1}(1-s)^{\beta-1}}{\sqrt{\int_0^1 y^{2(\alpha-1)}(1-y)^{2(\beta-1)}\mathrm{d}y}}. \tag{4.4}$$

In principle, α and β can be any positive numbers. For muscle profiles we consider only integer values for efficiency of evaluation. We additionally impose $\alpha > 1$ and $\beta > 1$ to yield a function where $\phi(0) = \phi(1) = 0$, and $\alpha \leq 9$ and $\beta \leq 9$. Larger values leading first to very sharp profiles that do not correspond to realistic muscle shapes, and second, to numerical precision issues due to the high values of both the numerator and the denominator. The denominator of Equation (4.4) is in fact independent of s and can be pre-computed for all allowed values of α and β using the Euler *beta function* [AR10].

The ratio α/β controls the asymmetry of the shape (with the distribution being symmetric for $\alpha = \beta$) while the individual values of α and β control the sharpness of the function's rise on each side, as illustrated by Figure 4.11.

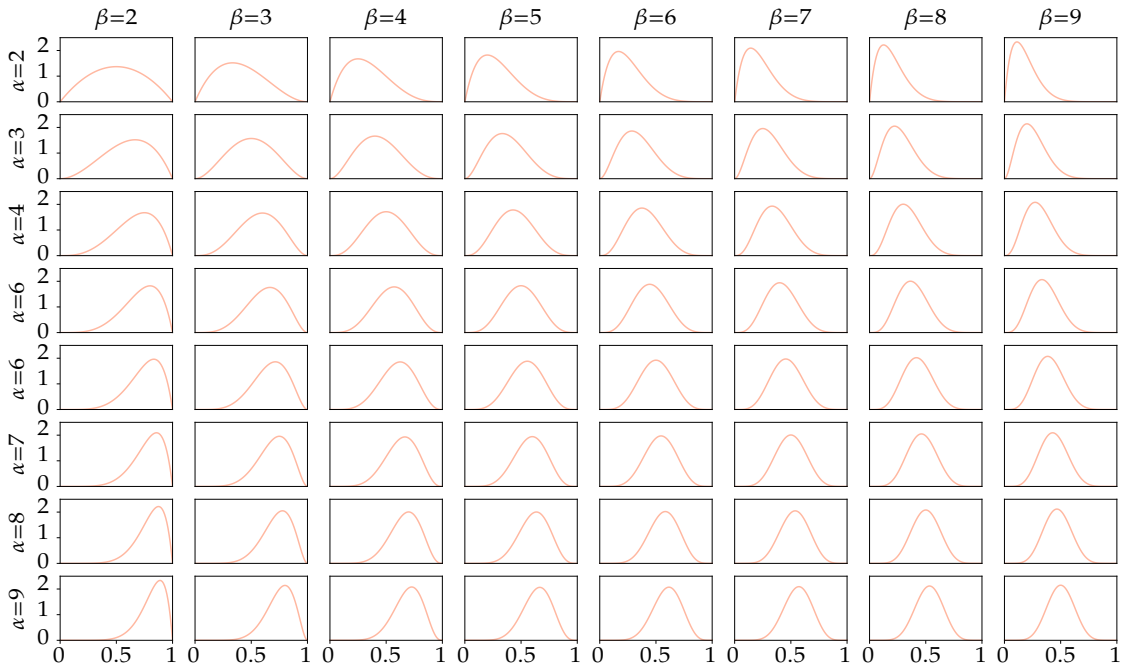


Figure 4.11: Profiles of the ϕ function for values of α and β.

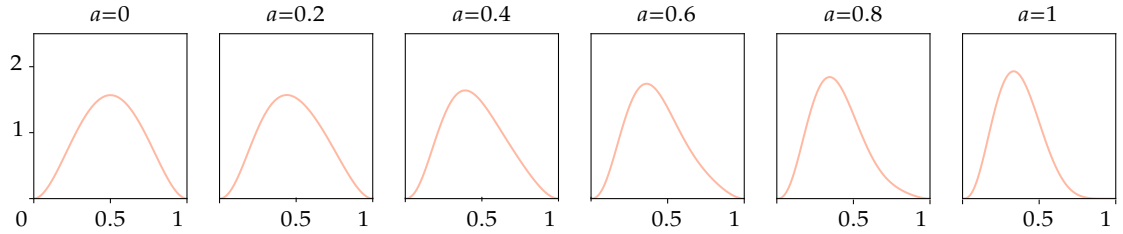


Figure 4.12: Profile function interpolation between $\alpha_0 = \beta_0 = 3$ and $(\alpha_1, \beta_1) = (4, 7)$.

To account for isometric deformation, our model must represent the muscle in its rest shape and its activated shape, and to interpolate smoothly from one to the other. This is represented by picking two pairs of parameters (α_0, β_0) and (α_1, β_1) representing respectively the rest shape and activated shape.

To interpolate between these two shapes while preserving volume, it is necessary to ensure that all interpolated functions also verify Equation (4.1) by ensuring their square integrate to 1. It can also be seen as spherical interpolation along the unit sphere of $\mathcal{L}^2$.

Let a be the interpolation parameter between $\phi(\alpha_0, \beta_0; s)$ and $\phi(\alpha_0, \beta_0; s)$. The rest shape corresponds to $a = 0$ and full activation to $a = 1$. The interpolated function is defined as

$$\Phi_a(s) = \frac{(1-a)\phi(\alpha_0, \beta_0, s) + a\phi(\alpha_1, \beta_1, s)}{\sqrt{F(a)}},$$

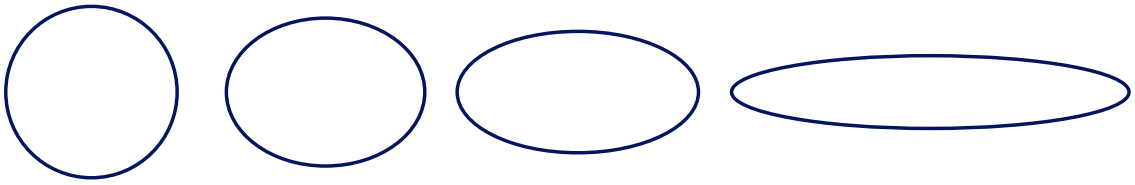


Figure 4.13: Ellipses of constant area and various eccentricities.

where

$$F(a) = \int_0^1 ((1-a)\phi(\alpha_0, \beta_0; s) + a\phi(\alpha_1, \beta_1; s))^2 \, \mathrm{d}s$$

is a normalization term. While the value of F might appear to be expensive to compute for each a, it can be formulated as

$$F(a) = (1-a)^2 + a^2 + 2a(1-a)\ K(\alpha_0, \alpha_1, \beta_0, \beta_1)\ ,$$

where K is a constant term which can be expressed in terms of the Euler *beta function* B

$$K(\alpha_0, \alpha_1, \beta_0, \beta_1) = \frac{B(\alpha_0 + \alpha_1 - 1, \beta_0 + \beta_1 - 1)}{\sqrt{B(2\alpha_0 - 1, 2\beta_0 - 1) B(2\alpha_1 - 1, 2\beta_1 - 1)}}\ ,$$

as shown in Appendix A.3. In practice, K can be tabulated prior to the execution, resulting in fast runtime evaluations. Figure 4.12 shows an example family of functions at various levels of interpolation a.

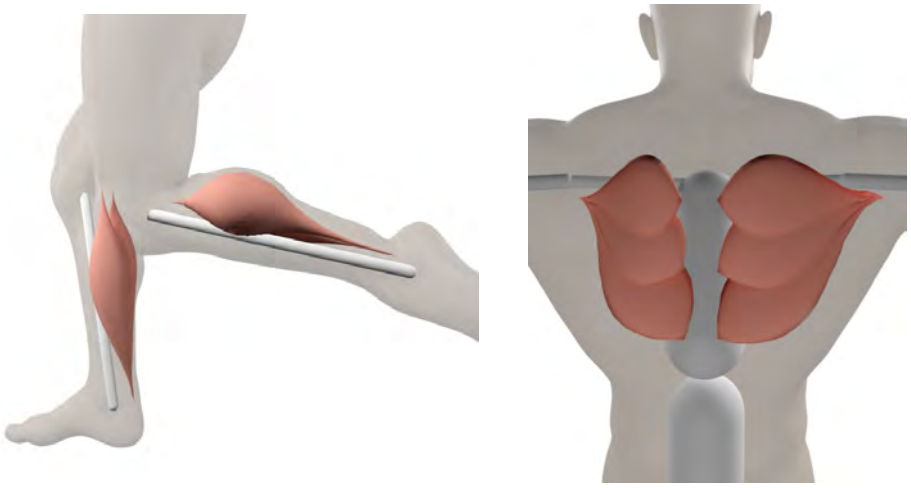
4.3.3 Cross-section

To model the cross section of muscles, we set $r(\theta)$ to be an ellipse defined by semi-length axis u and v:

$$r(\theta) = \frac{uv}{\sqrt{u^2 \cos^2\theta + v^2 \sin^2\theta}}\ .$$

The aspect ratio of the ellipse is defined by one parameter, the *eccentricity*, denoted $e \in [0, 1[$. Figure 4.13 shows the effect of eccentricity on the shape of the ellipse. An eccentricity of $e = 0$ defines a circle, and eccentricities closer to one define wider and flatter ellipses. The semi-axis lengths u and v are computed directly from the eccentricity e:

$$u = \sqrt[4]{(1-e^2)}\ , \qquad\qquad v = \frac{1}{u}\ .$$



(**a**) Calf muscles

(**b**) Dorsals

Figure 4.14: Figure (a): Calf muscles, one in its rest state ($\alpha_0 = 3, \beta_0 = 3$) and one in its activated state. ($\alpha_1 = 4, \beta_1 = 8$) Figure (b): Dorsals muscles in their rest state ($\alpha_0 = 2, \beta_0 = 3$).

Thus, we ensure that $uv = 1$, that is to say, the area of the ellipse is equal to that of a circle of radius 1. This property guarantees that $\int r^2 \mathrm{d}\theta$ stays constant, so that the volume does not change as e varies in time, because it satisfies Equation (4.2).

4.3.4 Summary

The muscle defined in the previous section can be completely described by the following parameters:

- The shape parameters of the muscle in rest state (α_0, β_0) and in activated state (α_1, β_1),
- The initial position of the endpoints $\mathbf{m}_0$ and $\mathbf{m}_1$ and the animation bone to which they are attached,
- The normals $\vec{\mathbf{n}}_{\mathbf{m}_0}$ and $\vec{\mathbf{n}}_{\mathbf{m}_1}$ orienting the muscle at each extremity,
- The width scale factor w,
- The eccentricity of the cross-section e,
- The activation level a.

The shape parameters are picked beforehand to model the desired shapes of muscles. For example, the biceps of Figure 4.10 use $\alpha_0 = \beta_0 = 3$; yielding a smooth symmetric

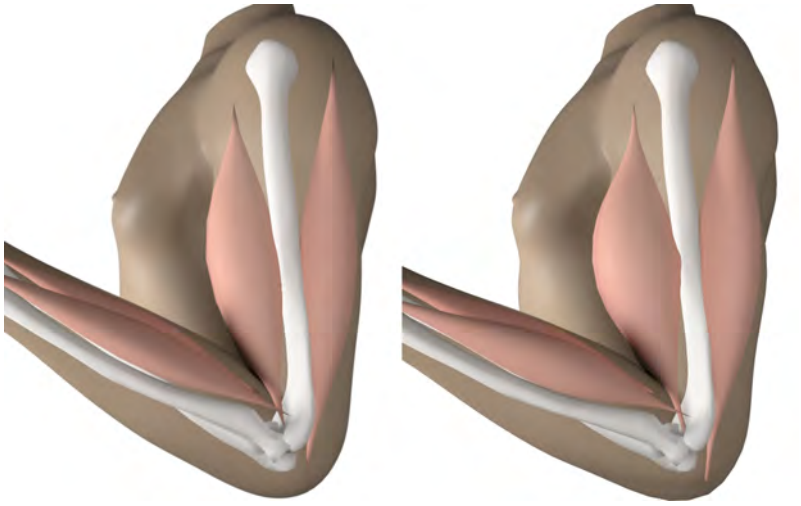


Figure 4.15: Amplification of muscle width. Left : normal biceps curl. Right : muscle width increased by 30 %.

shape in its rest state and $\alpha_1 = 4, \beta_1 = 7$ for a significant bulge in its activated state. The calf muscles of Figure 4.14(a) uses a slightly sharper activated state ($\alpha_1 = 4, \beta_1 = 8$). We also model convergent muscles such as the dorsals or pectorals with highly asymmetrical shapes, as shown in Figure 4.14(b), also using a high eccentricity $e = 0.96$ to generate flat muscles.

The endpoints positions and normals are set on the reference shape of the mesh and move kinematically with the skeleton, their motion being prescribed by the animation. In contrast, the activation and eccentricity can change freely during the motion, as our model ensures that the volume of the muscle is preserved regardless of their value. We found useful to keyframe them with the animation key poses to simulate the activation of muscles during the animations used in our experiments. It is also possible to override the volume preservation by setting the width parameter, amplifying the shape of the muscle bulge, as shown in Figure 4.15.

4.4 Discussions

4.4.1 Alternatives to the beta function

The beta function profile is carefully chosen to satisfy both the need for a variety of shapes to represent different muscles and efficiency of evaluation. The muscle field function will be evaluated several times per vertex, as part of the global skin representation used by the projection algorithm of Implicit Skinning. Using trigonometric or exponential functions, which are slower than polynomials, can dramatically decrease the Implicit

Function	Speed
Beta (2,2,4,7)	2522
Beta (8,8,9,9)	1337
Sine	895
Bump	1353
Piecewise cubic	3960

Table 4.1: Average number of evaluations per microsecond of several muscle profile functions (higher is faster).

Skinning performance. On the other hand, simplistic functions cannot account for the variety of shapes required for all muscles.

Several other function models were considered, and ultimately discarded for these reasons. For example, Lee and Ashraf [LA07] uses a sine function profile:

$$\Phi_{\text{sin}}(s) = \sin(\pi s).$$

Another similar function is the exponential bump function:

$$\Phi_{\text{bump}}(s) = \exp\left(\frac{-1}{4s(1-s)}\right).$$

These functions yield a symmetrical shape for the muscle (as $\Phi(s) = \Phi(1-s)$), however, to represent activated muscles or asymmetrical muscles, the model needs to be able to produce skewed shapes, while maintaining constant volume, i.e. $\int \Phi^2 \mathrm{d}s = 1$. When introducing assymetry in the profile either by piecewise definition or non-uniform scaling or the parameter space, the conservation of volume becomes either impossible to formulate in closed-form, or the scaling conditions become rapidly too expensive to evaluate, as opposed to the polynomial $F(a)$ in the derivation of the beta function.

Instead of trying to introduce skewness post-hoc, it is better to design a family of functions with a skewness parameter whose square integral is constant. An example would be a piecewise cubic spline designed with constraints on the end points ($s = 0$ and $s = 1$) and the position of the maximum at $s = a$, as shown on Figure 4.16:

$$\begin{aligned} \Phi_{\text{cubic}}(0) &= 0 & \Phi'_{\text{cubic}}(0) &= 0 \\ \Phi_{\text{cubic}}(1) &= 0 & \Phi'_{\text{cubic}}(1) &= 0 \\ \Phi_{\text{cubic}}(a) &= 1 & \Phi'_{\text{cubic}}(a) &= 0\,. \end{aligned}$$

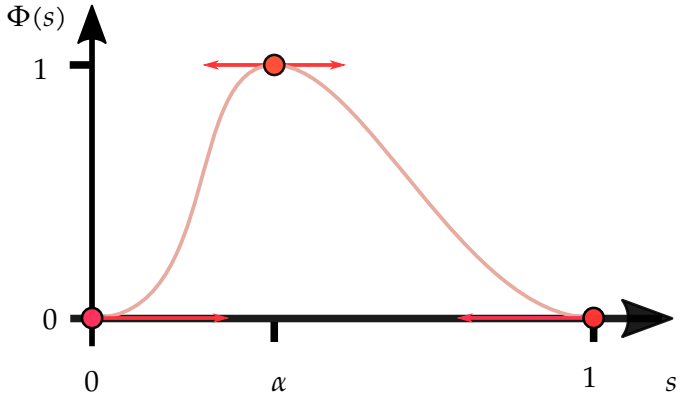


Figure 4.16: Piecewise cubic muscle profile.

These constraints define a family of piecewise cubic polynomial parametrized by a. In addition, the square integral of any such function Φ_{cubic} does not depend on a:

$$\int_0^1 (\Phi_{\text{cubic}}(s))^2 \, \mathrm{d}s = \frac{13}{35}.$$

The derivation of this property is found in Appendix A.4. As can be seen in Table 4.1, this function is very fast to evaluate. However this profile function is limited in the types of shapes that it can represent, with only one degree of freedom to represent both the extremal shapes (rest shape and activated) and their interpolation.

This led to the choice of the beta function family, as its performance is close to the cubic profile but offers a much wider shape choice.

4.4.2 Sketching profile

Our criteria for the choice of Φ are dictated by the application in interactive skinning, hence the need for a trade-off between the variety of shapes and evaluation speed.

Another type of possible input is sketching, which would be artist-friendly and could enable them to draw muscles fitting a character to the desired artistic effect, in a manner similar to *écorchés* in traditional media drawing. In our model, this can be seen as an optimization problem where one fits a muscle shape to a given sketch, such as the ones depicted in Figure 4.17, and tries to position it within the existing 3D model. A related approach is the method of TURCHET et al. [TFS17] which grows muscles for FEM simulation from painted patches on the mesh.

Given the closed-formula of the profile function, it is possible to optimize the shape parameters directly by minimizing the absolute error between the projection of the profile on the sketching plane and the sketch itself. Our proposed model of radial profile is differentiable in all its continuous parameters, making it possible to evaluate the gradient

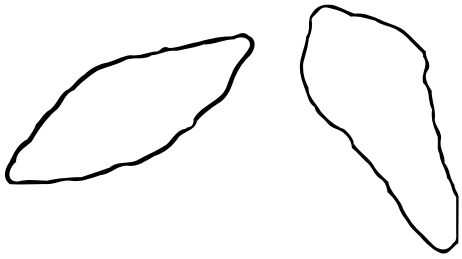


Figure 4.17: Example sketches of muscle profile.

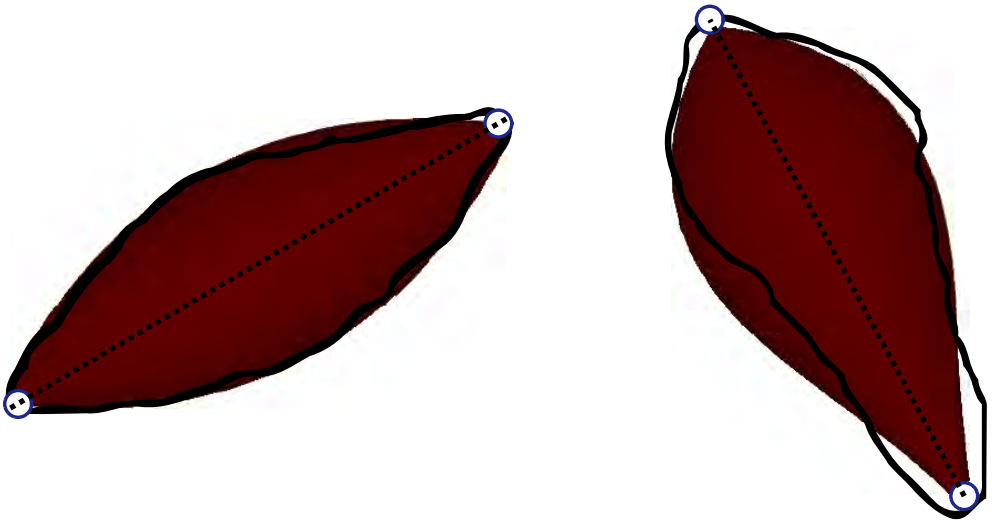


Figure 4.18: Fitting the beta function profile on sketches. Left: the optimization finds a good solution for the muscle endpoints and profile. Right: while the endpoints are correctly placed, the best fitting profile still fails to capture the sketch faithfully.

of Φ relatively to $\mathbf{m}_0, \mathbf{m}_1, w$ and a. The discrete parameters α and β, however, must be optimized separately.

Our implementation extracts the shape from a 2D sketch (stored in a 100×100 pixels black-and-white image). The inital step is to determine the muscle axis as the first principal direction in the pixel space, locating the muscle endpoints by intersecting the axis with the oriented bounding box of the muscles as a starting point. Given that there are only 81 possible pairs of (α, β) parameters, we run a continuous optimization of all the other parameters for all possible pairs, and select the best fit at the end of the process.

While the definition of the beta profile in terms of discrete parameters was an advantage to speed up its evaluation, it is a drawback to fit with a sketch, as it complicates the optimization process. Moreover, even the optimal profile fails to fit with some input sketches, as shown on Figure 4.18. To fit this profile, we would need to allow α and β to vary continuously, or to let them increase over 9, which in both cases result in numerical evaluation problems.

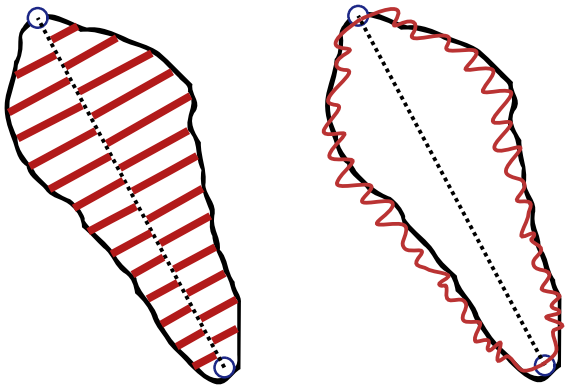


Figure 4.19: Sampling values on the sketch. Left: we sample alternatively each side of the axis to find the width given by the input sketch. Right: the DCT profile obtained with the samples.

For this use case, better results are obtained by sampling the muscle width along the axis, taking N_{samples} values x_k on each side of the axis, as shown on Figure 4.19. We then compute the Discrete Sine Transform (DST) as the profile function to create an interpolating function:

$$\Phi_{\text{DST}}(s) = \sum_{i=1}^{N_{\text{samples}}} c_i \sin(i\pi s),$$

where the coefficients c_i are computed from the sample values x_k:

$$c_i = \sum_{k=0}^{N_{\text{samples}}} x_k \sin\left(\frac{ki\pi}{N_{\text{samples}}+1}\right).$$

As seen on Figure 4.19, the full DCT overfits the sampled data. It can be simplified by applying a low-pass filter, keeping only the N_c first values of c_i and discarding the higher harmonics, i.e:

$$\Phi_{\text{DST}}(s) = \sum_{i=1}^{N_c} c_i \sin(i\pi s),$$

with $N_c < N_{\text{samples}}$. The result of this process is depicted on Figure 4.20, with $N_{\text{samples}} = 200$ and $N_c = 10$.

While the DST profile function is slower to evaluate than the other profile functions presented in this section, they are better suited for applications deriving the shape from user input, such as sketching. They still offer a closed-formula evaluation for their volume, as:

$$\int_{s=0}^{1} \Phi_{\text{DST}}(s)\mathrm{d}s = \frac{1}{N_c^2} \sum_{i=1}^{N_c} c_i^2,$$

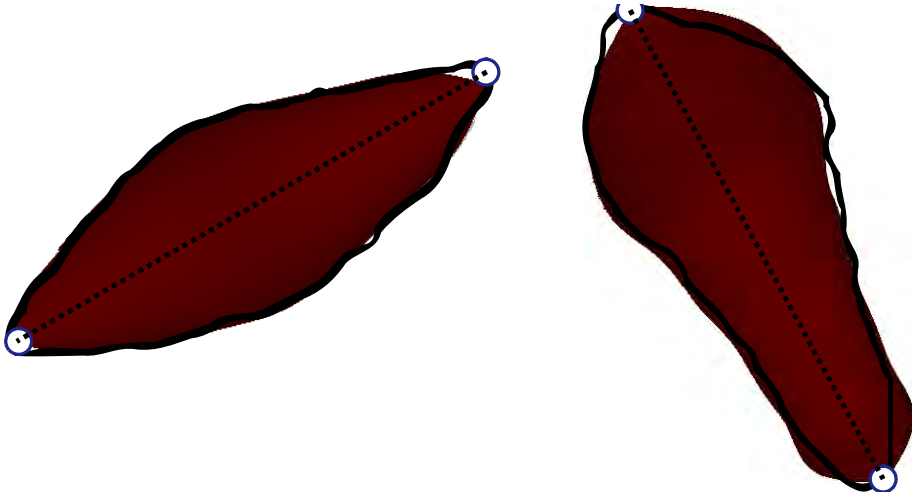


Figure 4.20: Muscle profiles of sketches fitted with DST with 10 harmonics, from 200 samples.

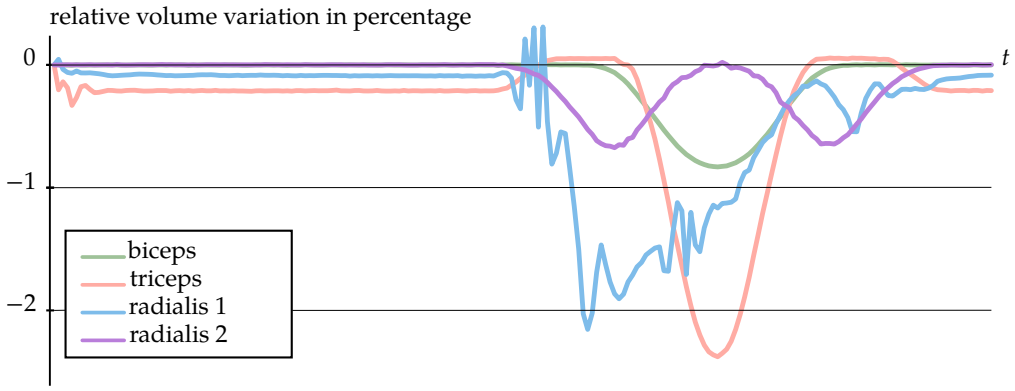


Figure 4.21: Relative variation (in %) of muscle volume in the biceps curl scene.

defining a unit $\mathcal{L}^2$ norm DST profile function with the same approach as with the beta functions. This illustrates the flexibility of our model which is independent on the choice of profile function, given that it satisfies Equation (4.1) to maintain constant volume. The specific profile function can thus be designed to answer the specific criteria of the application or use case.

4.4.3 Volume conservation and non-fusiform muscles

The derivation of Section 4.3 holds for a straight central axis $\mathscr{C}$, but when the axis deforms, the volume might slightly change even if the volume constraints are satisfied. Due to the discontinuous nature of the polyline and the addition of the reprojection operator, a closed formula for the volume becomes inaccessible. We nonetheless evaluate the volume numerically, using the volume of the representative mesh as a proxy for the implicit

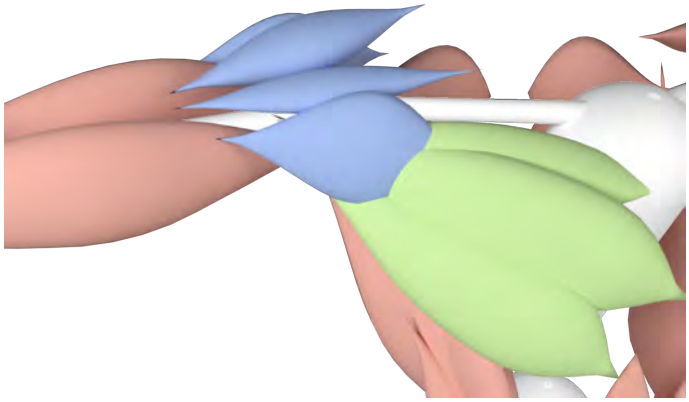


Figure 4.22: Pectoral (green) and shoulder (blue) muscles represented by sets of fusiform fibers. In this case, shoulder muscles collide against each other, while the pectorals are allowed to overlap to better approximate the desired shape.

surface volume. We record only a minimal variation of the volume, at most 2%, as shown in Figure 4.21 for the biceps curl scene.

The model presented in this chapter produces fusiform muscles such as biceps and triceps of the upper limb or quadriceps and calf muscles from the lower limb. To model more complex muscles such as the pectoral, we follow a method similar to the one proposed by Scheepers et al. [Sch+97] and Murai et al. [Mur+16]. We represent these muscles by instancing several fusiform shapes, each integrating a set of muscle fibers. Figure 4.22 shows an example of pectoral and shoulder muscles. Depending on the desired deformation effects, muscle collisions can be ignored (Figure 4.22 between green muscles) or enabled as explained in Section 5.3 (Figure 4.22 between blue muscles).

While we were able to capture the shape of fusiform muscles in the whole body, the motion of these complex convergent muscles (such as the pectoral) or multipennate muscles (e.g. the deltoid) is imperfectly represented with such a model. In particular, it becomes difficult to predict the type of deformations created by these muscles on the skin surface. The placement of these muscles and the setup of their parameters requires much time to achieve the correct looking skin deformations.

Increasing the number of fibers can lead to better results, but also increases the simulation time, and the number of parameters to set, slowing down the rigging process and the animation frame rate. To help with this setup, we define origin and insertion *curves* and sample the parameters of the muscles along the curve for the individual fibers instead of setting each independently. In addition, to provide a muscle representation that creates smoother shapes for the whole muscle, it could be possible to use blending operators (Section 2.3) to assemble the fusoid fibers in a coherent muscle shape before using it as

a deformer, but keeping the volume of the resulting shape constant requires dedicated operators that preserve volume of the resulting surface.

5 Dynamic muscle deformations

It's simple: if it jiggles, it's fat
— Arnold Schwarzenegger, *Pumping Iron*

In the previous chapter, we introduced a muscle model which was kinematically driven by the animation skeleton and deforms at constant volume. We now extend this model to deform the muscles by reacting dynamically to the animation motion. More specifically, we let the muscle axis control points $\mathbf{p}_i$ be driven by a physical simulation, which we present in Section 5.1. This physics-based approach enables us to model dynamic effects such as inertia or jiggling, as described in Section 5.2. We then show how to implement collision detection within the simulation to resolve collision between the muscle and other anatomic elements: bones, skin and other muscles (Section 5.3). We finish by a discussion on the consequences of using such an approximate method, and how some issues can be mitigated by recent work.

5.1 Position Based Dynamics

The sweep surface presented in the previous chapter is defined in terms of the position of the control points $\mathbf{p}_i$ defining its central polyline $\mathscr{C}$. While these points could be animated directly by specifying their trajectory, we rather use a physics simulation to generate complex motions such as inertial effects and jiggling.

Standard physical simulation is generally force-based and explicitly solve Newton's laws of motion by computing the force and integrating twice to update the position of objects. Force-based methods are physically realistic and simulate a variety of dynamical systems, but suffer from instability depending on the chosen integration method and time step. In our case, we seek to compute the position of the control points, modelled as

particles, to simulate the elasticity between the particles of a given muscle axis, and to avoid collisions between anatomy primitives.

Introduced by Müller et al. [Mül+07], Position Based Dynamics (PBD) is a real-time approximate method for physical simulation modelling dynamic particles tied by constraints. This section describes briefly the concepts underlying a PBD simulation. Further details can be found in the surveys by Bender et al. [Ben+14; BMM15].

PBD models a physical system by simulating particles with masses. Contrarily to standard physical simulation methods, the external and internal forces are not modeled directly, but represented as constraint functions $C(\mathbf{p}) = 0$ (equality constraint) or $C(\mathbf{p}) \geq 0$ (inequality constraint) depending only on the position of the particles $\mathbf{p}$. Time-integration updates the position and velocities of the particles by taking into account only inertial effects, using a semi-implicit Euler scheme. A constraint solver then iteratively attempts to satisfy all constraints, or at least to minimize the total deviation from the constraints. Non-linear constraints are linearised by a first-order approximation:

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p}.$$

The solver then finds the correction $\Delta\mathbf{p}$ to apply to the positions which minimizes the constraint in the direction of the constraint gradient $\nabla_{\mathbf{p}} C(\mathbf{p})$. The proposed solver works by solving iteratively each constraint using a Gauss-Seidel approach.

This physics simulation method offers many advantages over force-based methods. It is unconditionally stable, meaning that the energy of the system is never increasing regardless of the simulation conditions. The constraint functions are generic, their only requirement being the definition of function C and its gradient. This simplifies the implementation of many phenomena which were otherwise modelled separately within the same framework, such as rigid bodies [DCB14], fluid simulations [MM13] or deformable bodies and cloth [Mac+14]. While this method has no ambition to simulate such dynamic phenomena realistically, it produces a visually plausible approximation.

To imbue the muscles with dynamic behaviour, we use PBD to control the position of the control points $\mathbf{p}_i$ of the central axis $\mathscr{C}$. We leverage the generality of PBD constraints to avoid collision between the skin, bones and muscles which are all represented by implicit surfaces.

5.2 Elasticity and inertial effects

5.2.1 Particles setup

The control points $\mathbf{p}_i$ of each muscle axis are represented by particles of a Position Based Dynamics simulation. They behave as point masses obeying the differential equations of motion. The first step of the simulation setup is thus to set the mass of each particle.

As discussed in the previous chapter, the endpoints move with their respective skeleton bones. From the point of view of the physical simulation, these points are set as *kinematic* particles with an infinite mass (so that other particles and constraints cannot move it). Their motion is set at the beginning of each physics step by setting their velocity to follow their animation bone.

The total mass m of the muscle is distributed among the other particles $\mathbf{p}_i$ to give each particle a mass m_i. First, the muscle mass m is computed from its total volume V:

$$m = V\rho_M.$$

We use an average density of muscle tissue of $\rho_M = 1.06\,\mathrm{g.cm}^{-3}$ [Urb+01], while the value of V is directly obtained from Equation (4.3). Each intermediate particle $\mathbf{p}_i$ is given a mass m_i in proportion to the width of the muscle at the initial position of the particle:

$$m_i = \frac{\Phi(s(\mathbf{p}_i))}{\sum_j \Phi(s(\mathbf{p}_j))} m\,.$$

To represent the fact that the muscle evolves along soft tissues which restain its movement, we introduce a global damping coefficient μ on the velocities at each integration step of PBD. The semi-implicit Euler integration step is thus written, for each particle:

$$\begin{aligned}\vec{\mathbf{v}}(t_{n+1}) &= \mu\vec{\mathbf{v}}(t_n) + \vec{\mathbf{a}}(t_n)\Delta t\\ \mathbf{p}(t_{n+1}) &= \mathbf{p}(t_n) + \vec{\mathbf{v}}(t_{n+1})\Delta t\,,\end{aligned}$$

where $\vec{\mathbf{v}}(t_n)$ is the velocity of a particle, $\mathbf{p}(t_n)$ its position, and $\vec{\mathbf{a}}(t_n)$ its acceleration at the time step t_n. This damping coefficient (set to $\mu = 0.9$ in our experiments) prevents the jiggling motions to generate long-term oscillations in the muscles by dissipating their velocity quickly.

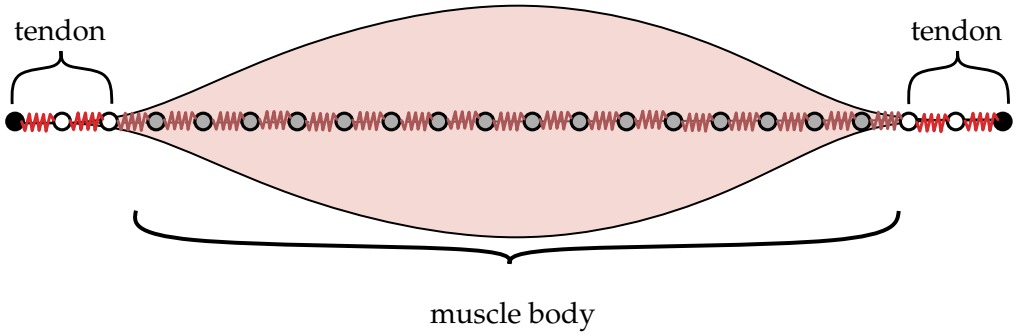


Figure 5.1: Position of tendons in the dynamic muscle model. The tendons particles have stiffer links, while the muscle body particles have looser distance constraint stiffness but smaller rest lengths.

5.2.2 Elastic distance constraints

To model the tension and elasticity of the muscle, we introduce *elastic distance constraints* [Jak01; Mül+07] between two successive particles $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$:

$$C(\mathbf{p}_i, \mathbf{p}_{i+1}) = \|\overrightarrow{\mathbf{p_i p_{i+1}}}\| - d_0,$$

where d_0 is the *rest distance*. This constraint's strength is controlled by a *stiffness* parameter k, which is a multiplicative factor applied to the computed position update $\Delta\mathbf{p}$. It is used to increase the time it takes for the constraint to bring back the particles to their rest length, simulating a looser tie between the two particles, thus increasing the effect of inertia. Constraints are set up differently depending on their position in the muscle. On each extremity of the muscle axis, the first and last 10% of its length is considered as tendon, as shown in Figure 5.1. Tendons constraints are stiff ($k \approx 0.9$) and have a rest length d_0 equal to their initial length, to model their tendency to bend rather than stretch. Constraints belonging to the middle section are assigned a much smaller rest length (approximately 2 % of their starting length) to represent the tension in the muscle fibers. The stiffness of muscle body controls the strength of the muscle tone and thus the amount of influence between inertial effects and muscle tension. In our experiments, a small value of k (e.g. $k \approx 0.1$) generates a muscle following its attachment bones with visible inertia, producing a noticeable jiggle. Conversely, a high value of the stiffness ($k > 0.7$) produces a very tense muscle which reacts quickly to motion changes. Figure 5.2 shows a comparison of the same animation played with different stiffness values on the muscles.

Similarly to the shape parameters of the previous chapter, the stiffness of the muscle body constraints can evolve over time. They can be keyframed by an animator to create different muscle behaviours depending on the action of the character. For example, in the

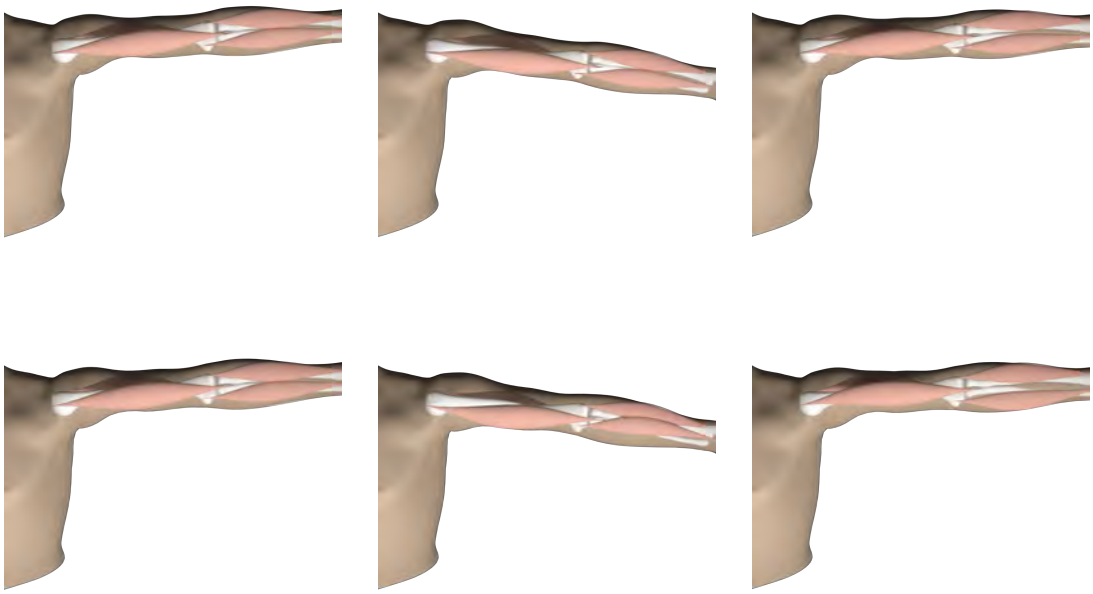


Figure 5.2: Comparison between stiff and loose muscles. Top row: stiff muscles ($k = 0.9$). Bottom row: loose muscles ($k = 0.4$).

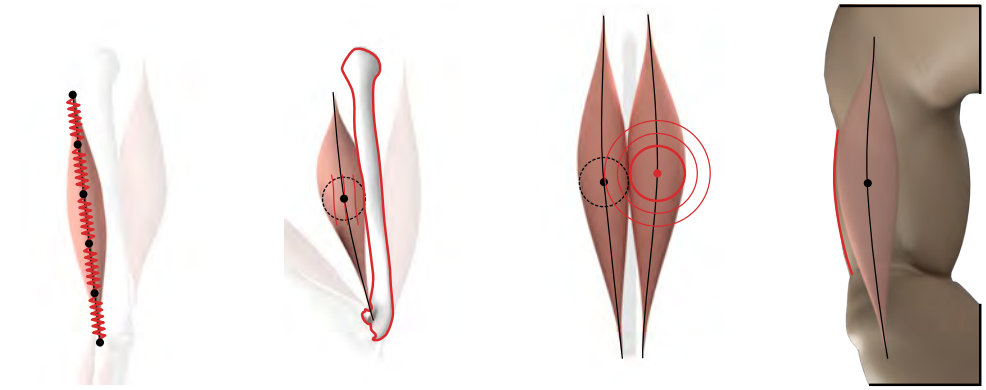
jump scene, we lower the stiffness of the calf muscles during the landing to emphasise the jiggling created by the contact of the legs with the ground, giving a more noticeable visual cue to the muscular reaction.

5.3 Collision resolution

In our body, the shape of muscles is constrained by the presence of rigid bones, other muscles, soft tissues and skin. To generate more plausible muscle deformation, and to better capture the effect of volume preservation, it helps to resolve collisions among the individual organs. The use of scalar fields to define the muscles provides an efficient way to solve the collision in PBD by defining a collision constraint against an implicit surface.

Definition. Let $\mathbf{p}_i$ be a particle, f a globally-supported 3D signed distance field and D_i the collision radius of the particle. The *implicit surface constraint* which maintains the particle at a distance D_i outside of the implicit surface defined by $f(\mathbf{p}) = 0$ is:

$$C(\mathbf{p}_i) = f(\mathbf{p}_i) - D_i \geq 0 \, .$$



Muscle elasticity Muscle-Bone Muscle-Muscle Muscle-Skin

Figure 5.3: PBD constraints in the dynamic muscle model used to represent muscle properties and interactions with surrounding elements.

The constraint gradient is immediately derived:

$$\nabla_{\mathbf{p}_i} C(\mathbf{p}_i) = \nabla f(\mathbf{p}_i) \ .$$

The collision radius D_i effectively treats the particle as a sphere for collision with other elements. It is by default initialized to the average width of the muscle at the position of the particle:

$$D_i = w\Phi(s(\mathbf{p}_i)).$$

Because this radius evolves with time (for example, due to activation of the muscle, as described in Section 4.3.2), it must be updated at each frame. With this constraint, we can resolve collision between muscles and other anatomic elements represented by scalar fields, such as bones, skin and other muscles (Figure 5.3).

5.3.1 Muscle-bone and muscle-muscle collision

The muscle axis constraint has a tendency to keep the muscle axis straight. When a limb such as the arm or the leg bends, this draws the end point of the muscle further inside the skin. This has the effect of burrowing the muscle under the skin, limiting its influence despite the bulging caused by the contraction. In the body, muscles are prevented to move away from the skin surface by the rigid bones. We seek to capture this effect by adding bones to our model.

To represent anatomically correct bones, we capture their shape from an anatomic model by using distance fields from bone meshes (as can be seen on the model pictured in Figures 4.10, 4.21 and 5.2). Each anatomic bone is attached to an animation bone, with

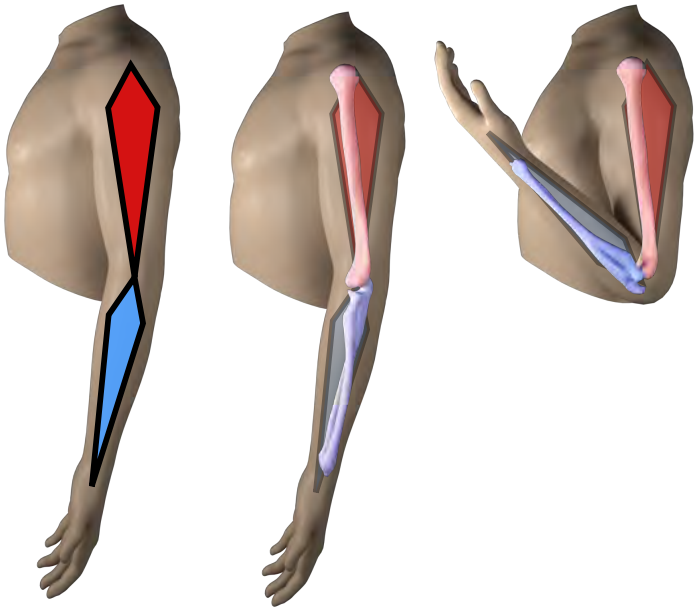


Figure 5.4: Anatomic bones and animation bones. Left: skin and animation skeleton in rest state. Center: Anatomic bones are added and tied to the animation skeleton. The humerus (in red) moves with the arm bone, and the radius and ulna (blue) with the forearm bone. Left: Anatomic bones move with their skeleton bones.

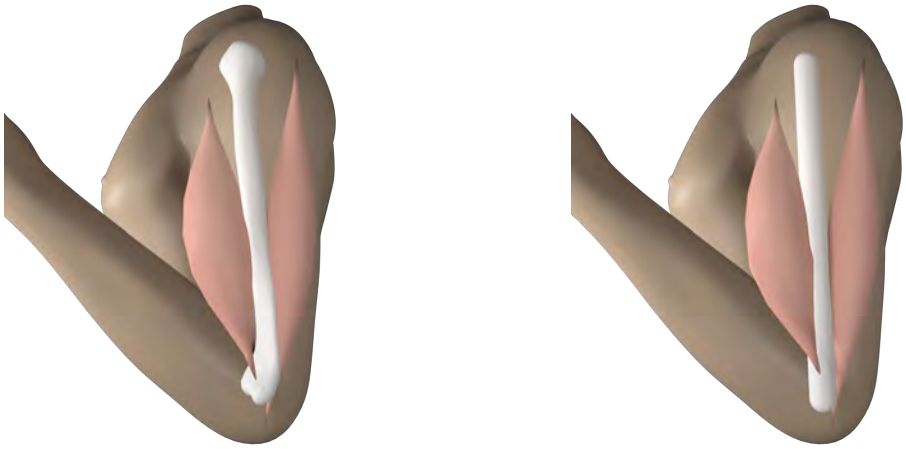
which it moves rigidly, as shown in Figure 5.4. The distance field computation against a mesh is expensive: to evaluate the distance from a point to a mesh, one must compute the distance from the query point to each triangle of the mesh [BA02]. To speed up the computation, we discretize the distance field and store each value in a grid of the same size as the HRBF grid of the associated animation bone (as defined in Section 3.1.1).

When anatomic bone shapes are not available, we use a cylinder as a proxy shape. The cylinder, defined by a constant radius R_{cyl} around the animation bone, is evaluated analytically:

$$f_{\text{bone}}(\mathbf{p}) = d(\mathbf{p}, [\mathbf{b}_j, \mathbf{b}_{j+1}]) - R_{\text{cyl}} \,,$$

where $[\mathbf{b}_j, \mathbf{b}_{j+1}]$ is the segment defined by the animation joints of the current bone and the next and d is the distance function. Given that muscles often collide on the shaft of the bones which has a roughly cylindrical shape, the results produced with the proxy shapes are very close to the ones using an anatomical bone model, as shown in Figure 5.5.

We optionally model collision between muscles with the same constraint. In this case, each particle of a given muscle will collide against the implicit surface of the other muscles. This is especially useful to model multipennate muscles such as the deltoid, as seen in Figure 4.22. In our model, this muscle is represented by several fusiform shapes. If they are not prevented to collide, the complex motion of the shoulder can draw all fibers together



(**a**) With anatomical bone. (**b**) With cylinder proxy.

Figure 5.5: Comparison between anatomical bones and proxy. Figure (a): biceps and triceps colliding against a distance field from an anatomical bone mesh. Figure (b): biceps and triceps colliding against a cylinder proxy shape.

on one side of the body, yielding unrealistic behaviour. We keep this feature optional because the computational cost of muscle-muscle collision grows as the square of the number of muscles in a given limb, which slows down the physics simulation.

5.3.2 Muscle-skin interaction

To represent the effect of the skin containing the muscles, we add a third field-based constraint to keep the particles inside the surface of the nearest HRBF f_j (Section 3.1). In this case, the constraint is meant to keep the particles *inside* an implicit surface. This can be achieved by simply reversing the sign of the constraint function C. In this case, the collision radius of the particle is 0: this constraint should not prevent the muscle from pushing the skin outwards, but the axis must remain inside the initial skin shape.

This effect is especially useful for muscles going across complex joints, such as the pectoral across the shoulder, since the effect of the muscle axis will otherwise tend to form a straight line and stick out the body. It also prevents inertial effects from dragging the particles outside the skin in fast motions such as running or jumping.

5.3.3 Friction

We model the friction in a similar manner to rigid body PBD [DCB14], by adding a tangential friction term to each particle which collides against a bone or a muscle. If $\Delta\mathbf{p}_i$ is the position correction of the i^{th} particle computed by the constraint solver, then the

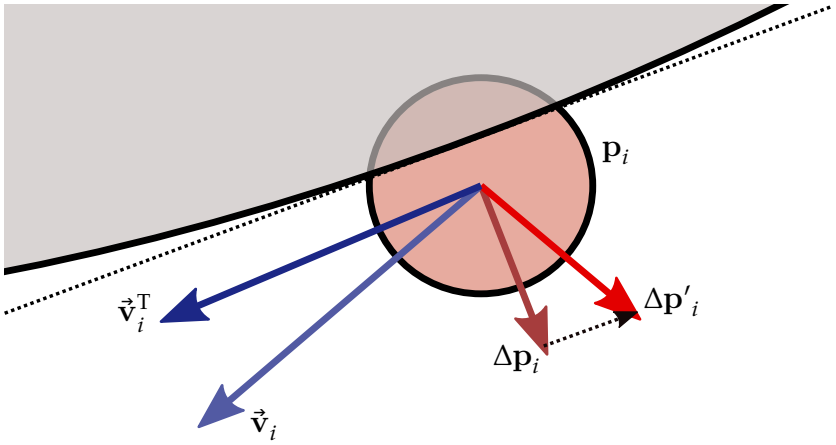


Figure 5.6: Friction in collision resolution. Particle $\mathbf{p}_i$ is colliding against the grey object. The constraint correction computed by the solver $\Delta\mathbf{p}_i$ is normal to the surface of the colliding object. Friction adds a correction term which is proportional to the tangential velocity $\vec{\mathbf{v}}_i^{\mathrm{T}}$, the projection of the velocity on the tangent plane.

final position variation $\Delta\mathbf{p}'_i$ is given by:

$$\Delta\mathbf{p}'_i = \Delta\mathbf{p}' - \eta\vec{\mathbf{v}}_i^{\mathrm{T}},$$

where η is a friction coefficient and $\vec{\mathbf{v}}_i^{\mathrm{T}}$ is the tangential velocity of the particle, i.e. the projection of the particle velocity $\vec{\mathbf{v}}_i$ on the tangent plane of the surface. This plane is normal to ∇C and $\Delta\mathbf{p}$, as the initial correction is colinear to the constraint gradient. This friction is opposed to the tangential velocity of the particle with relation to the contact surface, simulating the loss of energy when the two objects interact. These notations are detailed in Figure 5.6.

5.4 Discussion

5.4.1 Anatomic bones

For most animation models, anatomic bones are not available. While our method proposes to use cylinders in the absence of anatomically correct bone shapes, it is also possible to import a generic anatomic model of the human body to any humanoid character. This method, presented by Ali-Hamadi et al. [Ali+13] is known as *Anatomy Transfer*.

In general, animation bones correspond more or less to an anatomic bone or group of bones, as shown in Figure 5.4. While this model works for swing around a joint (such as the elbow or the leg bending), a more complex behaviour is required to model twist properly, especially for group of bones such as the ulna and radius in the forearm. A

study undertaken by Zhu et al. [ZHK15] shows that it is possible to control anatomic bones with skeleton bones by using skinning weights to map the animation transforms to realistic bone twist and swing.

5.4.2 Particles collision shape

As stated above, the particles have a spherical collision shape when they collide against bones or other muscles. Assuming spheres greatly simplifies the constraint evaluation, and is a valid approximation when the eccentricity of the muscle cross-section is close to 0. However, it fails to represent the muscle shape when the eccentricity parameter e becomes closer to 1. It is possible to represent ellipsoids with the same excentricity instead of spheres, but this greatly increases the evaluation cost of constraints, as this requires to compute the local frame of the particle and the angle, for a minimal effect on the visual result on the skin.

5.4.3 Stiffness in PBD

A major limitation of Position Based Dynamics is the dependency between the constraint solver time step, the number of solver iterations and the overall behaviour of the simulation. Despite the stiffness parameter k, the solver eventually converges towards an infinitely stiff simulation as the number of iterations increase. This coupling is problematic when trying to achieve a specific results with muscles of various stiffnesses, as changing the solver parameters (e.g. reducing iteration count to speed up the simulation) requires to adjust all the stiffnesses.

This issue was recently solved by Macklin et al. [MMC16], and the implementation recently published in the Flex GPU physics engine. The integration of this method in our implementation could improve the predictability of the results, and exhibit a more meaningful stiffness parameter.

6 Integration with Implicit Skinning

Every program is a part of some other program, and rarely fits.

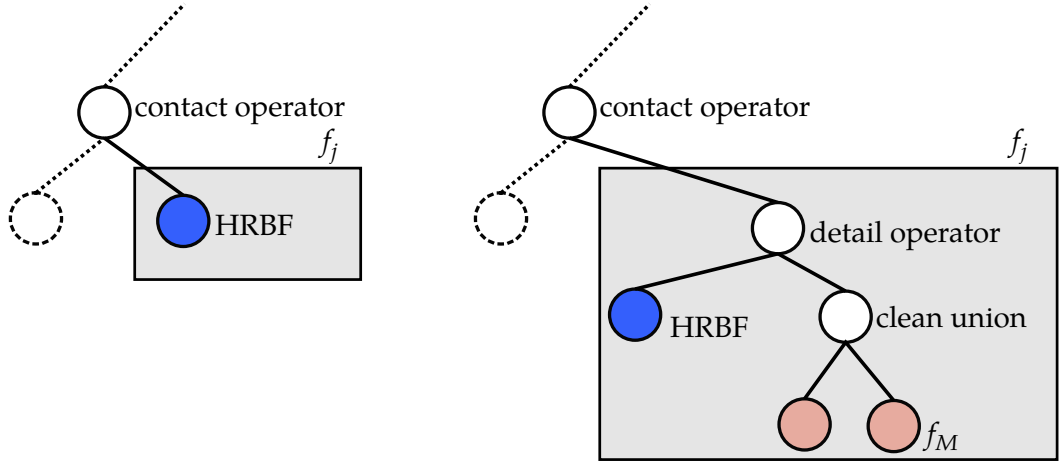
— Alan J. Perlis, *Epigrams on programming*

This chapter discusses the integration of the muscle model within the Implicit Skinning algorithm. In Section 6.1, we show how the muscles influence the result of skinning by using implicit composition operators to blend the muscles with the implicit skin representation defined in Section 3.1. Next, we describe the full pipeline of operations necessary to add dynamic muscle deformations to Implicit Skinning in Section 6.2. Finally, Section 6.3 presents the results obtained with our method and discusses how the performance of our implementation scales with the complexity of the model and the parameters of the method.

6.1 Scalar field composition

As detailed in Section 3.1, the shape of each part of the mesh associated with an animation bone j is captured by the scalar field f_j. All partial skin fields f_j are combined together to create the implicit skin representation F. To apply the deformations of the muscle shapes into the skin, we must integrate the muscle fields f_M within the composition tree, so that their motion is reflected in F and eventually, in the skin vertices through the surface tracking process.

Each muscle in our model is set to affect only one part of the skin, by associating it with one animation bone j and the related mesh part. For example, the biceps and triceps muscles should deform the skin representing the arm bone, while the *brachialis* and *brachoradialis* must be associated with the forearm. It is crucial to integrate the muscles at the skin part level, and not later in the implicit composition process, in order to benefit



(**a**) Composition without muscles (**b**) Composition with muscles

Figure 6.1: Integration of muscle field in the composition tree. Both figures detail the field of a single skin part f_j (greyed out area). Figure (a): without muscles, the skin part field f_j is only made of the HRBF field. Figure (b): when muscles are enabled, the muscles field f_M are blended together with a clean union operator, then blended with the HRBF thanks to the detail operator. The result of this blend is the skin part field f_j.

from the contact surface generated by the contact operator (Section 3.1.2). When body parts are in contact, the effect of the muscles must be taken into account before the contact operator is applied.

We are thus led to replace the HRBF-only representation of body part j by defining a new f_j blending the HRBF and the muscles primitives associated with this part, as depicted in Figure 6.1. Note that the animation bone j with which we associate the muscle is not necessarily one of the bones associated with the muscles' attach points (defined in Section 4.1). For example, the biceps will influence the skin shape of the arm bone, but its attach points are moving with the shoulder bone and the forearm bone respectively.

The sweep surface f_M defined in Chapter 4 is a distance field with global support. To be able to blend it with the HRBF scalar fields, it is necessary to adopt a homogeneous representation [BWG04]. The first step is thus to convert this field into a compact support function (see Section 2.2.2). We use the $\mathcal{C}^2$ fall-of filter function $K(d)$ defined in Equation (3.1), similarly to how the HRBF fields are converted from global support to compact support. The compact muscle function is thus defined as $K(f_M(\mathbf{p}))$. The fall-of function's radius R is chosen to be equal to the maximal width radius of the muscle:

$$R = \max_{s \in [0,1]} (wu\Phi(s)).$$

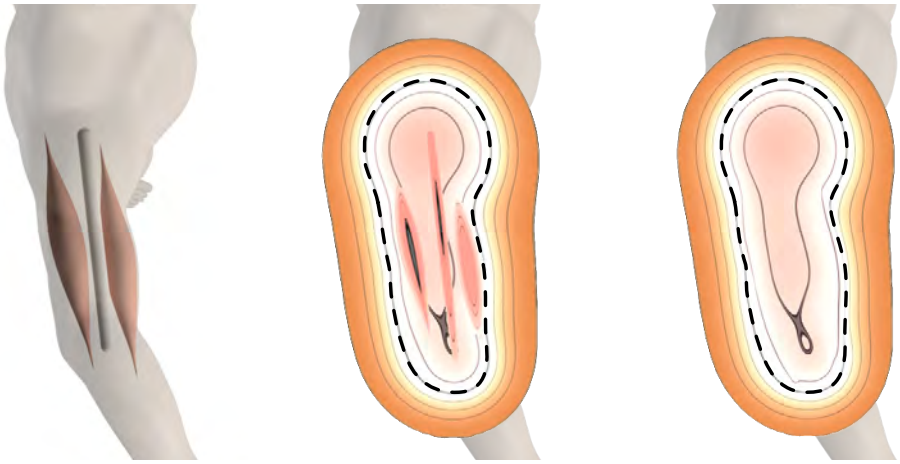


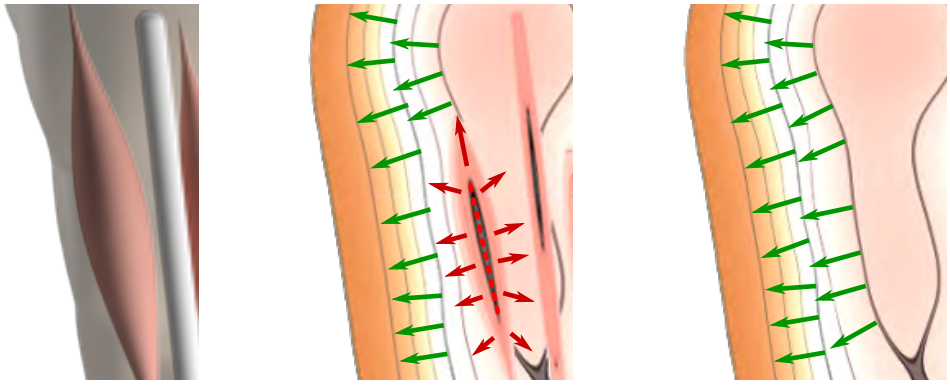
Figure 6.2: Blending muscle and bones with a HRBF. Left: muscle and bones are added to the skin representation of the arm. Center: using a clean union operator yields an irregular field, as shown on the level curves. Right: using Canezin et al.'s [CGB13] operator avoids these problems and creates a regular field. See also the enlargement in Figure 6.3.

We then assemble all muscular fields associated with a given animation bone to form the *anatomic field*. We stressed in Section 3.2 the importance, for the skin field F to be free of gradient discontinuities and critical points (i.e. points where $\nabla F = 0$) near the skin surface. The standard union operator on field functions is known to produce such gradient discontinuities. For this reason, we use the clean union operator defined in Section 2.3.1 which does not produce discontinuities.

The next step is to blend the anatomic field with the HRBF partial skin field. Unfortunately, the muscle field's gradient is by definition null on the muscle axis. Using a union or a clean union operator will introduce singular points on the resulting field inside the skin's surface, as well as gradient pointing in the wrong direction, as shown in Figures 6.2 and 6.3(b).

This problem of implicit modelling, raised by Canezin et al. [CGB13] happens when adding small details to comparatively larger objects. The solution is to use Canezin et al.'s [CGB13] *detail operator* described in Section 2.3.3 and Figure 2.10. This operator blends only the outside part of the muscle fields without including the central singular points and the bone-facing gradients, generating a suitable field for surface tracking, as can be seen in Figure 6.2 (right) and Figure 6.3(c).

During the animation, we want the skin to capture the underlying muscle's shape when it bulges out, but also when it contracts. In our approach, muscles are added to the initial HRBF skin approximation using the blending operator mentioned above. However, since the shape of the muscles is often apparent in the reference pose mesh, the HRBF captures the muscle shape with the rest of the skin, as can be seen in Figure 6.4(a). This means that



(**a**) Skin and primitives (**b**) Union operator field (**c**) Detail operator field

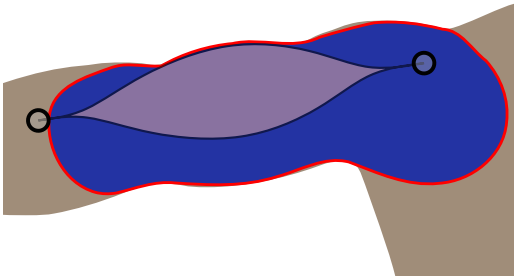
Figure 6.3: Gradient field of blended muscle field: detail of Figure 6.2 displaying the gradient field over the level curves. Figure (a): setup with skin and muscle primitives. Figure (b): with the union operator, the gradient field is influenced by the muscle, yielding gradients with erroneous directions or length (red arrows) and an area with singular points (dotted line). Figure (c): the detail operator guarantees a regular gradient field and removes the singular points near the surface.

if the muscle deforms completely inside the HRBF (for example, when it contracts and becomes thinner), it will not modify the blended surface's shape, leaving the resulting surface of the skin skin, represented by f_j unchanged (Figure 6.4(b)). In order to correctly deform the skin represented by this field with visible muscle deformations, we reduce the HRBF surface along muscles, as shown in Figure 6.4(c). This is done by directly manipulating the control points of the HRBF to push its surface towards the inside of the skin around the region of influence of the muscle. As a result, the muscle primitive stands out, and defines the muscular rest surface. After the blending, the muscle primitive becomes directly responsible for the deformations of the skin shape in this area, letting the vertices follow the deformation of the muscle (Figure 6.4(d)).

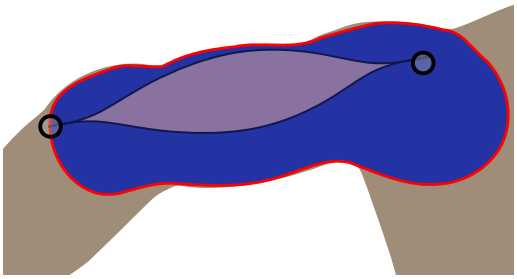
6.2 Integration in the Implicit Skinning pipeline

From an algorithmic perspective, the operations relative to the muscle model update must be incorporated in the Implicit Skinning pipeline. The implementation of this pipeline is illustrated in Figure 6.5.

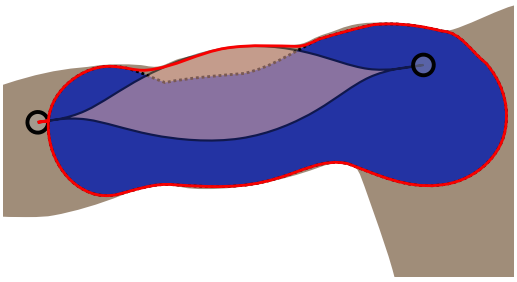
At each new frame, the animation skeleton is transformed to its new position, and the relative transforms of each bone are computed. The next step updates the data which is kinematically bound to the animation bones: the individual HRBF fields, the muscle endpoints and the fields representing the rigid anatomic bones are all updated after the



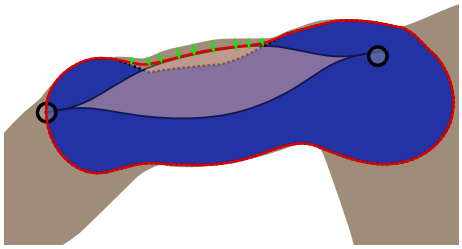
(**a**) Rest state with initial HRBF



(**b**) Bent arm with initial HRBF



(**c**) Rest state with thinned HRBF



(**d**) Bent arm with thinned HRBF

Figure 6.4: HRBF modification to integrate muscles. Figure (a): The initial HRBF obtained by sampling the mesh part (in blue), with a muscle representing the triceps. Figure (b): When the forearm bends, the triceps gets thinner, but the blending of the two surfaces (in red) does not change, as the HRBF itself did not move. The effect of the muscle extension cannot be seen on the skin. Figure (c): In this case the HRBF is thinned to leave the muscle responsible for the skin shape around its area of influence. Figure (d): When the forearm bends, the triceps gets thinner. This time, the effect on the skin is visible, as the vertices will track the muscle's surface (green arrows).

bone transforms. Keyframed per-muscle shapes parameters, such as the eccentricity e, activation a, stiffness k or width scale factor w are also updated during this phase.

Next, the PBD solver computes the new positions of the muscle central axis particles $\mathbf{p}_i$. The PBD time step is set to a tenth of the animation frame rate. When the particles position is computed, the muscles update their sampled normals $\vec{\mathbf{n}}_i$, as described in Section 4.2.1, and scales its width according to the new length of its axis, according to the formula derived in Section 4.3.1. At this point, the muscle primitives are updated and the composite skin field F reflects the moving skin parts, the contracting or extending muscles and the contact areas between body parts. This field is used in the mesh tracking steps described in Chapter 3 to skin the character mesh vertices.

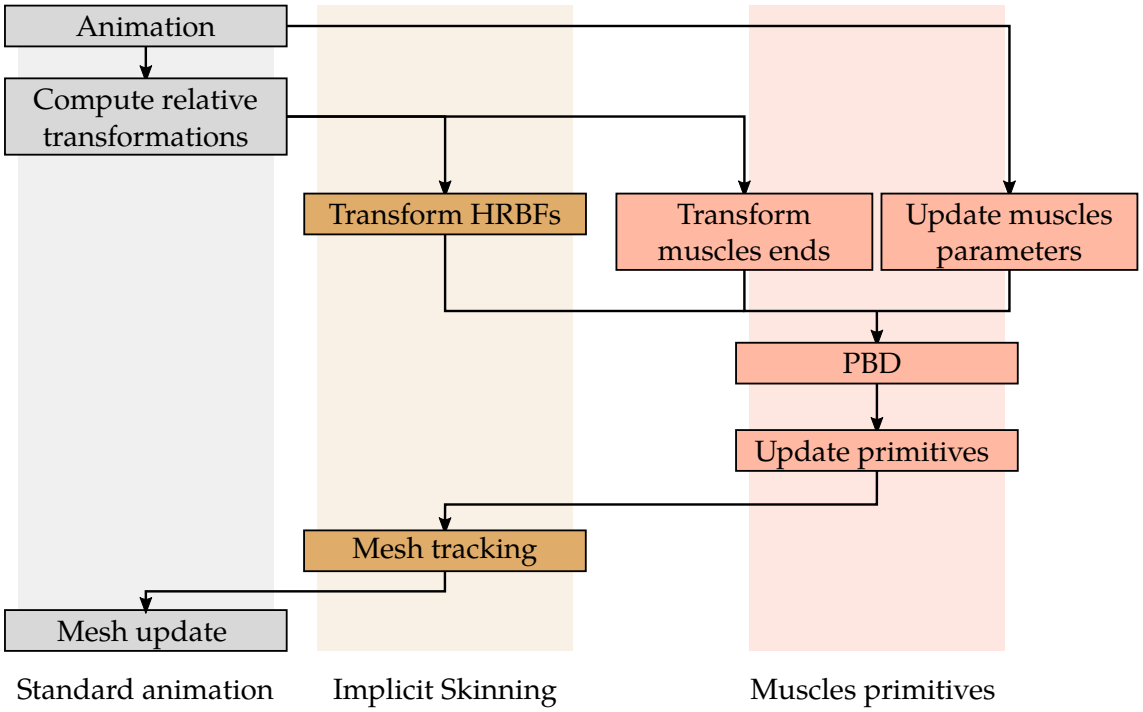
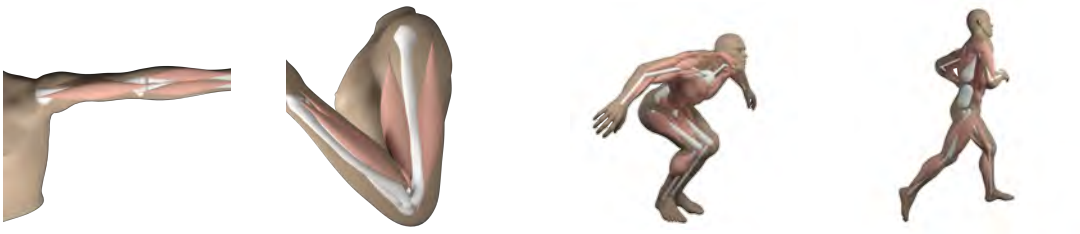


Figure 6.5: Breakdown of the Implicit Skinning pipeline with muscles. Steps pictured in grey are the standard geometric skinning pipeline, steps in bown are the Implicit Skinning correction algorithm, and steps in salmon are our new anatomic-related workflow. Arrows represent direct dependencies between steps.

6.3 Skinning with muscles

6.3.1 Results and discussion

To demonstrate the effects of our method, we set up a variety of scenes, ranging from simple motions, such as an arm shake or a biceps curl, involving only a few muscles, to more challenging motions, such as jumping or running, with a fully rigged model.

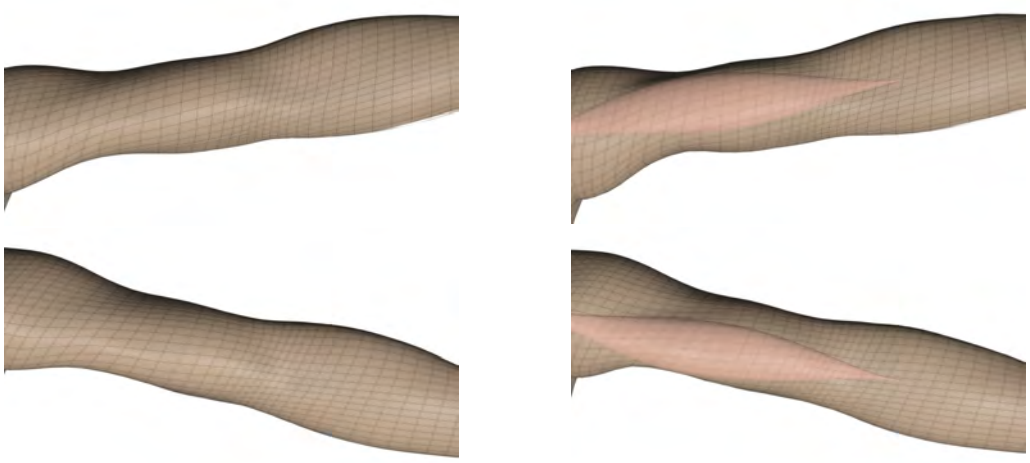


(**a**) Arm shake (**b**) Biceps curl (**c**) Jump (**d**) Run

Figure 6.6: Test scenes used for our results (see description in Table 6.1).

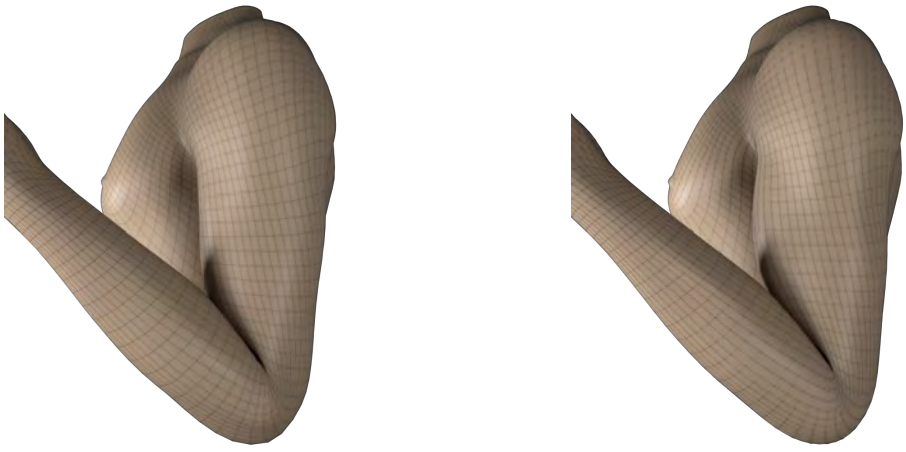
Scene	# vertices	# bones	# muscles	# constraints
Arm shake	2172	3	4	456
Biceps curl	2172	3	4	456
Jump	19296	12	50	4250
Run	19296	12	50	4250

Table 6.1: Summary of the scenes used for our results. The number of PBD constraints is given for 30 particles per muscle.



(**a**) Implicit Skinning (**b**) Implicit Skinning with muscles

Figure 6.7: Detail of the arm during the arm shake scene. Figure (a): Standard Implicit Skinning solution. Figure (b): Implicit Skinning with muscles.

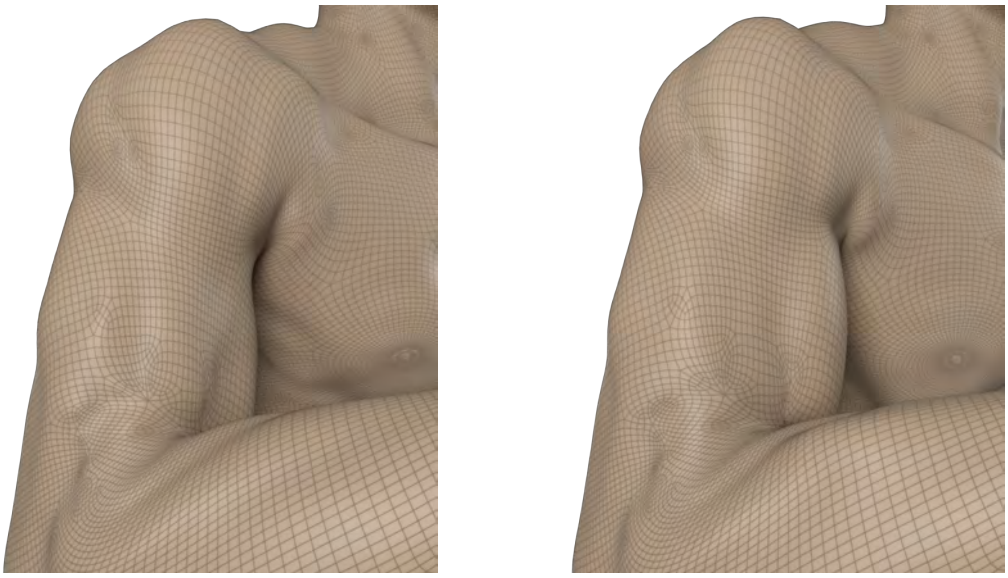


(**a**) Implicit Skinning (**b**) Implicit Skinning with muscles

Figure 6.8: Detail of the arm during the biceps curl scene. Figure (a): Standard Implicit Skinning solution. Figure (b): Implicit Skinning with muscles.

(**a**) Implicit Skinning (**b**) Implicit Skinning with muscles

Figure 6.9: Detail of the legs during the jump scene. Figure (a): Standard Implicit Skinning solution. Figure (b): Implicit Skinning with muscles. The scene is captured just before the impulse when the muscles are the most contracted. Note also the contact handling between the calves.



(**a**) Implicit Skinning (**b**) Implicit Skinning with muscles

Figure 6.10: Detail of the arm during the run scene. Figure (a): Standard Implicit Skinning solution. Figure (b): Implicit Skinning with muscles, showing the bulged biceps, deltoid and pectoral.

Table 6.1 and Figure 6.6 summarize the setup of each scene, and side-by-side comparison are illustrated in Figures 6.7 to 6.10.

The arm shake (Figures 6.6(a) and 6.7) shows a very fast motion of the arm, which allows to exhibit the inertial effects of our muscle model and the effect of the stiffness of the muscles, as detailed in Section 5.2. The biceps curl scene (Figures 6.6(b) and 6.8) is a prime example of the effect of muscle contraction (for the biceps) and extension (for the triceps), as well as showing the contact handling between the arm and the forearm. These scenes use skin and bones meshes derived from a commercial anatomic model (Anatoscope).

The jump and run scenes (Figures 6.6(c), 6.6(d), 6.9 and 6.10) are designed to be stress tests of our model. The animations show fast-changing motion (the jumping impulsion and reception, the run contact and up/down poses), which could be challenging for an inertial system to follow without the bone and skin collision constraints described in Section 5.3. The model used in these scenes is also a higher resolution mesh and has been rigged with 50 muscles with all types of constraints (elasticity, muscle-bone, muscle-muscle and muscle-skin) resulting in 1 500 particles and more than 4 000 constraints. The model is a regular animation model using cylindrical bone proxies for muscle-bone collision.

6.3.2 Performance

Our study is both qualitative, to show the effect of the muscles primitive on the skinning result, and quantitative, to report on the timings of our method and the CPU and memory overhead it incurs on the default Implicit Skinning implementation. In particular, we study the average time per frame of our scenes with regard to the complexity of the model, and study the influence of the parameters of the simulation on the frame rate. These timings were generated on a 3.6 GHz Intel Xeon E5-1650 CPU with 64 GB of memory.

By far, the two more intensive steps of the computation are the PBD simulation step and the mesh tracking step from Implicit Skinning. A breakdown of the timings can be found in Table 6.2.

While Implicit Skinning runs at real-time frame rates even in more demanding scenes, adding more muscles or increasing the number of particles per muscle increases both the physics simulation time, and the tracking time. The former is caused by the increased numbers of particles to simulate and of constraints to solve. The latter occurs because the Implicit Skinning algorithm evaluates the skin field F several times per vertex in a frame. The cost of computing F itself depends on the number of primitive fields functions to evaluate. While the use of compact support function with a BVH tree prunes out many

Scene	Standard IS	IS + Muscles		
		Tracking	PBD	Total
Arm shake	0.008	0.013	0.055	0.068
Biceps curl	0.005	0.015	0.042	0.057
Jump	0.050	0.462	0.096	0.558
Run	0.051	0.473	0.096	0.569

Table 6.2: Average times in seconds per frame for our different scenes and 30 particles per muscle. Times are given for standard Implicit Skinning without muscle, and for Implicit Skinning with muscles detailing the time spent by the IS tracking, and by the PBD simulation.

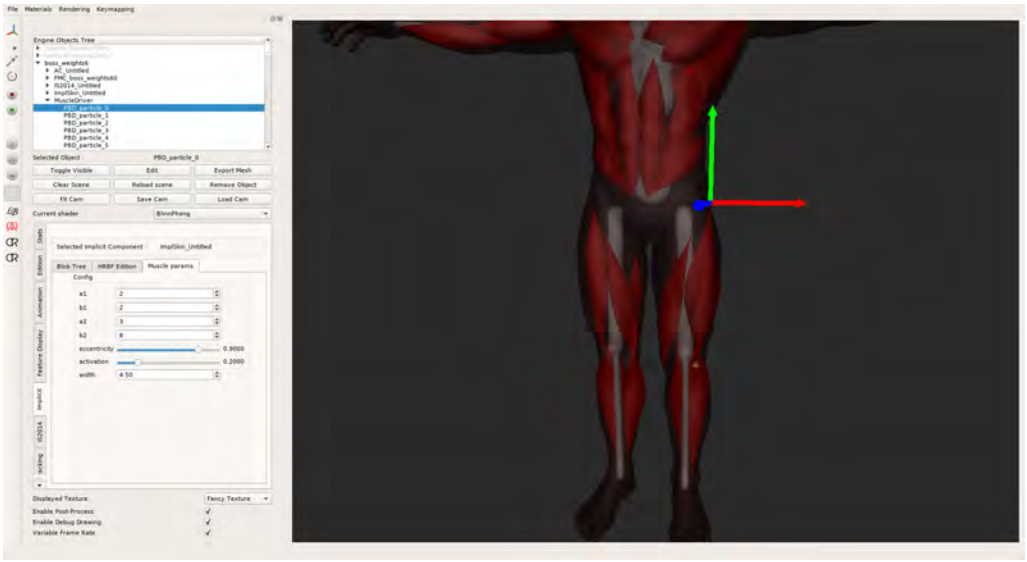


Figure 6.11: Interactive muscle editing session in our animation and modelling application, showing the exposed muscle shape parameters and the end points position manipulation.

useless computations, the muscle function evaluation and composition incurs a penalty on the overall function evaluation for vertices which are affected by one or several muscles. In particular, the time spent evaluating the muscle function f_M is proportional to the number of particles per muscle, because of the projection operator which projects the query point on every segment (as seen in Section 4.2.2).

In our most complex scene with 50 muscles each including 30 particles, we maintain a total frame time below the second. It is possible to improve the performance in cases where more interactive frame rates are required, for example when editing the muscle parameters (Figure 6.11). In this case, the skinning can be deactivated to reach interactive response times. The number of particles can be safely decreased for muscles which are only subject to small deformations, increasing the simulation speed, as shown in Figure 6.12.

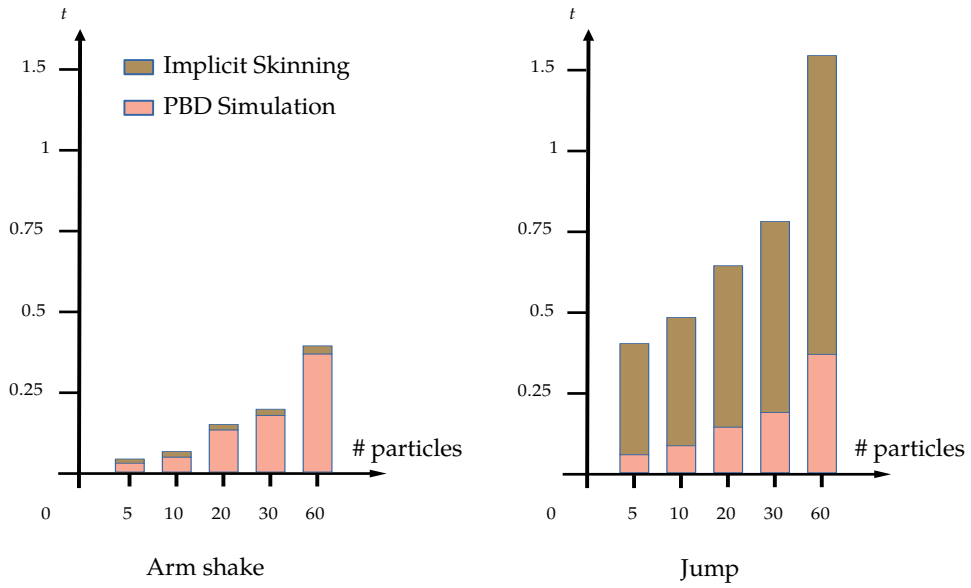


Figure 6.12: Average times in seconds for the computation of an animation frame, for different number of particles per muscle. The brown bars represents the time spent in computation of mesh tracking, the salmon bars represents the time spent in PBD simulation.

There is also room for improvement in the evaluation of the field functions, which quickly becomes the main bottleneck when the number of vertices of the skinned model increases. Specifically, a smarter spatial data structure (e.g. spatial hashing) or value caching could improve the function evaluation time. Our current implementation in a multi-threaded application handles parallelism at the task level, but the costliest steps (PBD and IS) have to be handled sequentially. Nonetheless, both PBD and IS are heavily parallelizable. The PBD solver is designed to be order independent so that particles position can be solved in parallel. Similarly, the Implicit Skinning algorithm steps can be run simultaneously per-vertex. While our implementation leverages CPU multi-threading, better performances could thus be achieved with a GPU implementation of both PBD and IS.

From a memory standpoint, the overhead to Implicit Skinning is minimal, due to the compact representation of muscles in memory, as the function and its gradient are evaluated in closed form from the muscle parameters (position and normals of the polyline points and shape parameters). Each muscle thus occupies only about one kilobyte of memory (with 30 particles per muscle), assuming single-precision floats. The look-up table for the 81 possible beta function values (one for each pair of (α, β)), which is common to all muscles, requires an additional 324 bytes. The blending also requires storing the detail operator in a 2D grid, adding 200 kilobytes. Using anatomical bone shapes, each

stored as a discretized distance field in a $128 \times 128 \times 128$ 3D grid requires approximately 8 megabytes per bone shape. If memory consumption becomes an issue, the resolution of the 3D grid can be lowered, or the bone distance fields replaced by proxy cylinders, as detailed in Chapter 5.

6.3.3 Limitations

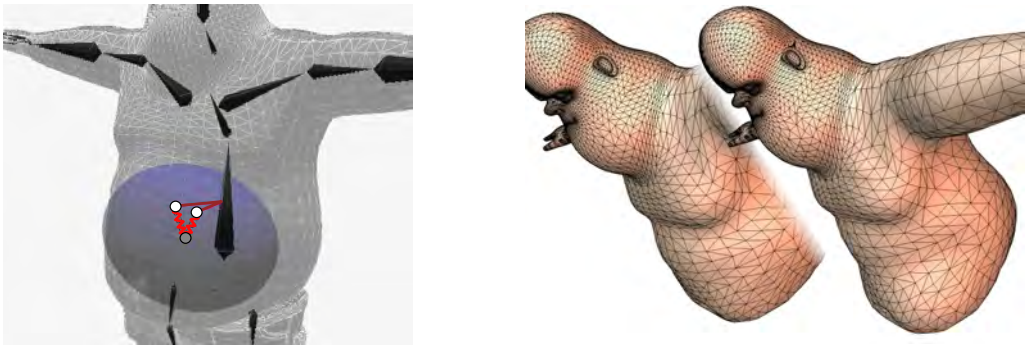
The main limitations of our method come from the limits of the muscle model itself which were exposed in Section 4.4.3, in particular, the difficulty to model complex muscles such as the pectorals, the buttocks, or the deltoids. This issue is aggravated by the high degree of freedom of the related joints. While the elbow or knee function mostly as mechanical hinges and rotate only around one axis, the shoulder and hips have several rotational degrees of freedom, which makes it difficult to predict the muscle's motion in all cases.

Another limitation is the necessity to manually edit the HRBFs around the muscles to let the vertices be deformed only by their nearest muscles. While it might be possible to do this editing automatically, this might be the sign that another skin representation is necessary to replace the HRBFs with a more flexible representation. Another way of overcoming this limitation would be to define a local scalar field around the vertices, in order to let each vertex be deformed by the most relevant anatomic primitives.

Finally, the dynamic behaviour of the muscles, being governed mostly by the scaleless stiffness parameter k sometimes yields counter-intuitive inertial deformations and requires tuning. In some cases it is necessary to keyframe it to obtain a stiffer or looser behaviour than expected. For example, during the jump scene, we artificially lowered the stiffness to exhibit a more visible jiggle at the reception of the jump. While this might be useful to represent the varying tension force in the muscle depending on its action, there is no guideline for the setting of this parameter except by trial and error, because it is not tied to a physical value which could be used as a reference. This issue is tied to the nature of PBD as an approximation of physics, and could be solved by using a more realistic physics method for the dynamic simulation, at the expense of an increased computational cost.

6.4 Towards implicit anatomic volumes

We showed that our model can blend the skin contact and elasticity modelling of Implicit Skinning with deformations generated by muscle primitives. While we focused our study on muscles, the integration with the Implicit Skinning algorithm is in fact generic and could take into account primitives representing other volumetric effects from the character's anatomy. Using similar models to define other dynamic elements induces different types



(**a**) Model setup.

(**b**) Effect of the fat lump.

Figure 6.13: Implicit fat tissue jiggling. Figure (a): setup of a jiggling belly. The center of the ellipsoid is a PBD particle (in grey) attached by elastic distance constraints (in red) to kinematic particles(in white) attached to a skeleton bone. Figure (b): side-by-side comparison of the model without the fat primitive (left) and with the fat primitive(right), showing the deformation induced on the skin.

of secondary motion, for example, fat tissues jiggling. We experimented this approach by using an ellipsoid field f_{fat} centered on one particle $\mathbf{p}$:

$$f_{\text{fat}}(\mathbf{q}) = \frac{\left(x_{\mathbf{q}} - x_{\mathbf{p}}\right)^2}{a^2} + \frac{\left(y_{\mathbf{q}} - y_{\mathbf{p}}\right)^2}{b^2} + \frac{\left(z_{\mathbf{q}} - z_{\mathbf{p}}\right)^2}{c^2} - 1 \,.$$

Similarly to the setup of a muscle, this particle is tied to the animation through kinematic particles and elastic distance constraints, as seen in Figure 6.13(a). We set the mass of the particle to $m = 20$ kg and set the distance constraints rest length to their initial length, and their stiffness to a low value ($k = 0.1$). This setup creates a high inertia fat lump implicit primitive which is integrated in the implicit skin model like the muscle primitives, as described in Section 6.1. The effect of this primitive creates a belly jiggle effect on the skin, shown in Figure 6.13(b).

The behaviour of fat tissues and their resulting action on the skin's surface is surely not completely captured by this model, and would require a dedicated study to the same extent as our modelling of muscle behaviour in Chapter 4. The ability to integrate fat tissues shows that our proposed integration of implicit primitives into the Implicit Skinning pipeline is sufficiently flexible to integrate primitives representing not only muscles but other anatomic elements.

Conclusion

That's a very nice rendering, Dave. I think you've improved a great deal.

— HAL 9000, *2001: A Space Odyssey*

Through this dissertation, my goal was to broaden the scope of the core idea behind Implicit Skinning: the joint use of meshes and implicit surfaces for character animation. In particular, the implicit approach to modelling has proven useful to design families of shapes representing muscles, giving freedom to choose the functional definition adapted to the use case, while deriving mathematical constraints to ensure the plausibility of muscle deformations. Moreover, the scalar fields defining the implicit surfaces enable the seamless integration of these shapes into a physics simulation, elegantly resolving collisions between muscles and bones. The combination of these two advantages, coupled to the existing Implicit Skinning algorithm, has enabled me to replace the many disparate deformers, whose configuration is often burdensome and ad-hoc for the artists, with a unified representation of the skin, and a way to blend in their dynamic effects. The focus of this work has been specifically on muscles because of their great role in the appearance of a character and its actions. The generic principles used in the definition of the muscle model within the skinning pipeline, however, are generic enough to allow subsequent research for models of other anatomic elements or other secondary animation effects. Besides the fat tissue representation discussed in the last chapter, scalar fields can also be used to generate wrinkles and folds in the skin, or to simplify collision detection, for example, with a cloth simulation. Of course, this warrants further research, in order to derive the correct mathematical properties of such applications to be implemented in a skinning pipeline.

My work brings together concepts from skinning methods, implicit surfaces modelling and composition, and dynamic simulation, in order to achieve an efficient implementation of the proposed rigging and animation method. To this effect, I implemented these various systems under a common interface in in my animation engine, with the ability to edit interactively the parameters of the muscles and to run concurrently each component of the animation pipeline. This implementation extracts the benefits of each of its components and provides ways for extensibility by limiting the coupling between them. It also suffers from the induced drawbacks inherent to each component: the difficulty to model complex

muscles and joints with the muscle model, the unintuitive material parameters of Position Based Dynamics, and the necessity to manually edit the implicit skin representation to enable all muscle effects. In addition, the performances of the implementation degrade as the complexity of the model increases, which effectively limits the interactive use-case to one complex character at most. Many of the functional limitations of the method can be overcome by replacing the proposed model with more complex approaches, for example, replacing PBD with a more accurate physics simulation. This would in turn decrease the performances of the implementation. Conversely, the implementation could be made faster, settling for a degraded solution by doing without some of the features of the model or limiting the accuracy of the simulation. In computer graphics, there is often a trade-off between performance and quality, and the right setup depends on the application and the time budget an implementation can afford to spend on animation effects.

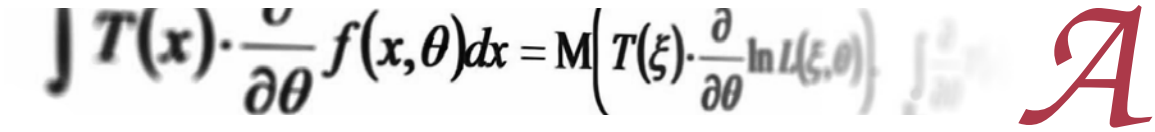
While the method presented in this thesis makes the case for a wider use of implicit surfaces in skinning, it remains to be seen whether they will be adopted by animators in the future, and how they could be used for future research in geometric skinning.

Because of the high learning curve of computer animation (which requires a great deal of both artistic and technical skill) and the imperatives of production, animators often prefer to deal with tried-and-tested methods, even if it means handling their shortcomings manually. Animators are often not familiar with implicit surfaces and the mathematical background behind them. Thus, if implicit objects are to see more use in future animation software, it would require to devise controls that animators can use with familiarity and simplicity. The genericity of methods based on scalar fields, however, is a advantage: meshes, cages or parametric surfaces can be used as proxies and converted to an implicit representation internally, exposing only the geometrically meaningful parameters to the artists. They can also be designed by artist-friendly interfaces such as sketching. The processes presented in this thesis make no assumption on the nature of the scalar fields but only on mathematical properties that are simple to enforce. This loose coupling between the surface representation and the skinning algorithm makes it easier to use the methods discussed in this dissertation as building blocks for better animation tools. However, a major limitation of basing an approach on Implicit Skinning is that it works only as the last skin deformation in the sequence of skinning operations. Our muscle deformers are built with this requirement in mind. This will be a problem for a practical implementation in animation software: one cannot expect every skin deformation operation available in Maya or Blender to be rewritten specifically for Implicit Skinning. Moreover, it is particularly inefficient to provide an implicit representation of the whole skin when only some areas are affected by the implicit-based deformers and corrections. An elegant solution to these two issues would provide a local implicit model and blend smoothly the implicit-based

deformations with any other skinning method, paying the computational cost only where needed and to integrate the deformations caused by implicit surfaces at any point in the skinning process.

The latest work in high-quality skinning research show a convergence of physics-based and data-based methods, either by fitting a generic physics template to a specific model, or by extrapolating a physics simulation from captured data. These progress is due in part to the recent advances in machine learning and specifically the application of deep learning methods to computer animation. The applicability of machine learning to geometric skinning methods is still an ongoing research topic. While neural networks can learn to reproduce complex non-linear operations such as skinning, their users often need to make sense of the parameters of models. Besides, learning algorithms benefit from models with compact representations, which reduces the combinatorial explosions of parameters in the learning process. Indeed, one of the goals of my work was to define a muscle with a compact representation and meaningful parameters, which humans – and machine learning algorithms – find easier to manipulate. Thus, a complexity-reducing model such as the one presented in this thesis could be a stepping stone towards the next generation of geometric skinning algorithms.

Appendix



A Proofs

A.1 ARAP Jacobi iteration

Recall the expression of the ARAP energy:

$$\begin{aligned} E(\mathbf{v}_0, ...\mathbf{v}_n) &= \sum_{i=0}^{n} E_i(\mathbf{v}_0, ...\mathbf{v}_n) \\ &= \sum_{i=0}^{n} \sum_{k=0}^{n} c_{ik} \left\| (\mathbf{v}_i - \mathbf{v}_k) - \mathbf{R}_i(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) \right\|^2 , \end{aligned}$$

assuming the weights c_{ik} are the cotangent weights if $\mathbf{v}_i$ and $\mathbf{v}_k$ are neighbours and 0 if not.

We can derive the gradient of E relatively to a given vertex $\mathbf{v}_u$:

$$\nabla_u E = \nabla_u \sum_{i=0}^{n} \sum_{k=0}^{n} c_{ik} \left\| (\mathbf{v}_i - \mathbf{v}_k) - \mathbf{R}_i(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) \right\|^2 .$$

This yields two family of non-null terms:

$$\begin{aligned} \nabla_u E = &\sum_{k=0}^{n} c_{uk} \nabla_u \left\| (\mathbf{v}_u - \mathbf{v}_k) - \mathbf{R}_u(\mathbf{v}_{\mathbf{ref}\,u} - \mathbf{v}_{\mathbf{ref}\,k}) \right\|^2 \\ &+ \sum_{i=0}^{n} c_{iu} \nabla_u \left\| (\mathbf{v}_i - \mathbf{v}_u) - \mathbf{R}_i(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,u}) \right\|^2 . \end{aligned}$$

Now differentiating the norm-squared terms yields:

$$\begin{aligned} \nabla_u \left\| \mathbf{v}_u - \mathbf{v}_k - C \right\|^2 &= 2(\mathbf{v}_u - \mathbf{v}_k - C) \\ \nabla_u \left\| \mathbf{v}_i - \mathbf{v}_u - C \right\|^2 &= -2(\mathbf{v}_i - \mathbf{v}_u - C), \end{aligned}$$

where C is a constant term.

This yields a formula for the gradient:

$$\nabla_u E = 2\left(\sum_{k=0}^{n} c_{uk}\left((\mathbf{v}_u - \mathbf{v}_k) - \mathbf{R}_u(\mathbf{v_{ref}}_u - \mathbf{v_{ref}}_k)\right) - \sum_{i=0}^{n} c_{iu}\left((\mathbf{v}_i - \mathbf{v}_u) - \mathbf{R}_i(\mathbf{v_{ref}}_i - \mathbf{v_{ref}}_u)\right)\right) .$$

We rewrite the two sums in one:

$$\nabla_u E = 2\left(\sum_{k=0}^{n} c_{uk}\left((\mathbf{v}_u - \mathbf{v}_k) - \mathbf{R}_u(\mathbf{v_{ref}}_u - \mathbf{v_{ref}}_k)\right) - c_{ku}\left((\mathbf{v}_k - \mathbf{v}_u) - \mathbf{R}_k(\mathbf{v_{ref}}_k - \mathbf{v_{ref}}_u)\right)\right) .$$

By definition $c_{uk} = c_{ku}$, thus it yields:

$$\begin{aligned}\nabla_u E &= 2\sum_{k=0}^{n} c_{uk}\left(2(\mathbf{v}_u - \mathbf{v}_k) - \left(\mathbf{R}_u(\mathbf{v_{ref}}_u - \mathbf{v_{ref}}_k) - \mathbf{R}_k(\mathbf{v_{ref}}_k - \mathbf{v_{ref}}_u)\right)\right) \\ &= 2\sum_{k=0}^{n} c_{uk}\left(2(\mathbf{v}_u - \mathbf{v}_k) - (\mathbf{R}_u + \mathbf{R}_k)\,(\mathbf{v_{ref}}_u - \mathbf{v_{ref}}_k)\right) .\end{aligned}$$

We differentiate once more in v to get the 3×3 block (u, v) of the Hessian matrix $\mathbf{H}_E$: $\nabla^2_{uv} E = \nabla_v \nabla_u E$. First, we derive the diagonal blocks (i.e. assuming $u = v$)

$$\begin{aligned}\nabla^2_{uu} E &= 2\sum_{k=0}^{n} c_{uk}\nabla_u\left(2(\mathbf{v}_u - \mathbf{v}_k) - (\mathbf{R}_u + \mathbf{R}_k)\right)(\mathbf{v_{ref}}_u - \mathbf{v_{ref}}_k)\big) \\ &= 4\sum_{k=0}^{n} c_{uk}\mathbf{I}_3 .\end{aligned}$$

Now the non-diagonal blocks where $u \neq v$:

$$\nabla^2_{uv} E = 2\sum_{k=0}^{n} c_{uk}\nabla_v\left(2(\mathbf{v}_u - \mathbf{v}_k) - (\mathbf{R}_u + \mathbf{R}_k)\right)(\mathbf{v_{ref}}_u - \mathbf{v_{ref}}_k)\big) .$$

leaving only one non-zero term of the sum where $k = v$:

$$\nabla^2_{uv} E = -4c_{uv}\mathbf{I}_3 .$$

Since the cotangent coefficients c_{uv} are 0 when $\mathbf{v}_u$ and $\mathbf{v}_v$ are not neighbours, the resulting Hessian matrix is very sparse and diagonal-dominant.

We are looking for the positions of vertices $(\hat{\mathbf{v}_1}, ...\hat{\mathbf{v}_n})$ minimizing E. Because E is quadratic, the optimal solution can be found in one iteration of Newton's method. Given a starting mesh $(\mathbf{v}_1^{(p)}, ..., \mathbf{v}_n^{(p)})$, we define

$$(\mathbf{v}_1^{(p)}, ..., \mathbf{v}_n^{(p)}) + (\Delta\mathbf{v}_1, ..., \Delta\mathbf{v}_n) = (\hat{\mathbf{v}_1}, ..., \hat{\mathbf{v}_n}) ,$$

where $\Delta\mathbf{V} = (\Delta\mathbf{v}_1, ..., \Delta\mathbf{v}_n)$ is the solution to the linear problem

$$[\mathbf{H}_E(\mathbf{V})]\,\Delta\mathbf{V} = -\nabla E(\mathbf{V}) .$$

Because $\mathbf{H}_E$ is diagonal-dominant, the Jacobi method finds an iterative solution. Writing $\mathbf{H}_E = \mathbf{D} + \mathbf{N}$ where $\mathbf{D}$ is diagonal ($\mathbf{D} = \text{diag}(4\sum_k c_{ik})$) and $\mathbf{N}$ contains the non-diagonal elements.

$$\Delta\mathbf{V}^{(\text{k}+1)} = -\mathbf{D}^{-1}(\nabla E + \mathbf{N}\Delta\mathbf{V}^{(\text{k})}) \ .$$

With the previous formula we can write the iterative equation for each vertex$\mathbf{v}_i$:

$$\mathbf{v}_i^{(\text{j}+1)} = \frac{1}{4\sum_{k=0}^{n} c_{ik}} \left(2 \left(\sum_{k=0}^{n} c_{ik}(2(\mathbf{v}_i^{(\text{j})} - \mathbf{v}_k^{(\text{j})}) - (\mathbf{R}_i + \mathbf{R}_k)(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) \right) \right) - \sum_{k=0}^{n} 4c_{ik}\mathbf{v}_k^{(\text{j})} \right)$$

$$\mathbf{v}_i^{(\text{j}+1)} = \frac{1}{\sum_{k=0}^{n} c_{ik}} \left(\frac{1}{2} \sum_{k=0}^{n} c_{ik}(\mathbf{R}_i + \mathbf{R}_k)(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) + \sum_{k=0}^{n} c_{ik}\mathbf{v}_k^{(\text{j})} \right) \ .$$

We now introduce the normalized cotangent weights $\overline{c_{ik}}$ for each vertex $\mathbf{v}_i$:

$$\overline{c_{ik}} = \frac{c_{ik}}{\sum_{u=0}^{n} c_{iu}} \ ,$$

which simplifies the notation:

$$\mathbf{v}_i^{(\text{j}+1)} = \frac{1}{2} \left(\sum_{k=0}^{n} \overline{c_{ik}}(\mathbf{R}_i + \mathbf{R}_k)(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) + \sum_{k=0}^{n} \overline{c_{ik}}\mathbf{v}_k^{(\text{j})} \right) \ .$$

Finally, we define

$$\mathbf{b_i} = \frac{1}{2} \sum_{k=0}^{n} \overline{c_{ik}}(\mathbf{R}_i + \mathbf{R}_k)(\mathbf{v}_{\mathbf{ref}\,i} - \mathbf{v}_{\mathbf{ref}\,k}) \ ,$$

which yields the update equation proposed in Section 3.3.

A.2 Volume of extrusion surface

The volume enclosed by the muscle's extrusion surface is given by

$$V = \pi w^2 l,$$

where w is the width factor and l the length of the axis.

Proof.

$$V = \int_{s=0}^{1} \int_{\theta=-\pi}^{\pi} \int_{\rho=0}^{w\Phi(s)r(\theta)} \rho \text{d}\rho \ \text{d}\theta \ l \text{d}s \ .$$

Integrating in ρ yields:

$$V = l \int_0^1 \int_{-\pi}^{\pi} \frac{(w\Phi(s)r(\theta))^2}{2} \text{d}\theta \text{d}s$$

$$= w^2 l \int_0^1 (\Phi(s))^2 \, \mathrm{d}s \int_{-\pi}^{\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta \, .$$

The integral in θ is the area of an ellipse of semi-axis length u and v:

$$\begin{aligned}\int_{\theta=-\pi}^{\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta &= \frac{1}{2} \int_{-\pi}^{\pi} \frac{u^2 v^2}{u^2 (\cos(\theta))^2 + v^2 (\sin(\theta))^2} \mathrm{d}\theta \\ &= \int_{\frac{-\pi}{2}}^{\frac{\pi}{2}} \frac{u^2 v^2}{u^2 (\cos(\theta))^2 + v^2 (\sin(\theta))^2} \mathrm{d}\theta,\end{aligned}$$

since the function is π-periodic.

We compute this integral by noticing that on $]\frac{-\pi}{2}, \frac{\pi}{2}[$,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\theta}\left[uv \arctan\left(\frac{v\tan(\theta)}{u}\right)\right] &= uv \frac{v}{u(\cos(\theta))^2} \left(\frac{1}{\left(\frac{v\tan(\theta)}{u}\right)^2 + 1}\right) \\ &= \frac{uv^2}{u(\cos(\theta))^2 \left(\frac{v^2}{u^2}(\tan(\theta))^2 + 1\right)} \\ &= \frac{u^2 v^2}{u^2 (\cos(\theta))^2 + v^2 (\sin(\theta))^2} \, .\end{aligned}$$

The integral in θ is expressed as a limit:

$$\begin{aligned}\int_{\pi}^{\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta &= \lim_{x \to \frac{\pi}{2}^-} uv \left(\arctan\left(\frac{v\tan(x)}{u}\right) - \arctan\left(\frac{v\tan(-x)}{u}\right)\right) \\ &= 2uv \lim_{x \to \frac{\pi}{2}^-} \arctan\left(\frac{v\tan(x)}{u}\right) \\ &= 2uv \lim_{x \to \frac{\pi}{2}^-} \frac{\pi}{2} - \arctan\left(\frac{u}{v\tan(x)}\right).\end{aligned}$$

This limit is now determined, because

$$\begin{aligned}\lim_{x \to \frac{\pi}{2}^-} \arctan\left(\frac{u}{v\tan(x)}\right) &= \arctan(0) \\ &= 0,\end{aligned}$$

thus,

$$\lim_{x \to \frac{\pi}{2}^-} \arctan\left(\frac{v\tan(x)}{u}\right) = \frac{\pi}{2}.$$

Finally,

$$\int_{-\pi}^{\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta = \pi uv \, .$$

We substitute the integral in θ by the expression above in the computation of V:

$$V = w^2 l \int_0^1 (\Phi(s))^2 \, \mathrm{d}s \; \pi u v \; .$$

Since we enforce $uv = 1$, it yields

$$V = \pi w^2 l \int_0^1 (\Phi(s))^2 \, \mathrm{d}s \; .$$

We also constrain $\int \Phi^2 = 1$, thus we have

$$V = \pi w^2 l \; .$$

□

A.3 Interpolation of beta function

The Euler *beta function* B is defined as

$$B(\alpha, \beta) = \int_0^1 u^{\alpha-1}(1-u)^{\beta-1} \mathrm{d}u \; .$$

Let us define

$$I(\alpha, \beta) = \int_0^1 u^{2(\alpha-1)}(1-u)^{2(\beta-1)} \mathrm{d}u \; = B(2\alpha - 1, 2\beta - 1) \; .$$

which appears in the denominator of Equation (4.4).

We define the linear interpolation $\Phi(s)$ between $\phi(\alpha_0, \beta_0; s)$ and $\phi(\alpha_1, \beta_1; s)$, governed by the activation parameter a, such that $\int_0^1 (\Phi(s))^2 \, \mathrm{d}s = 1, \forall a \in [0, 1]$ as:

$$\Phi_a(s) = \frac{(1-a)\phi(\alpha_0, \beta_0; s) + a\phi(\alpha_1, \beta_1; s)}{\sqrt{F(a)}} \; ,$$

where $\sqrt{F(a)}$ is the $\mathcal{L}^2$-norm of the numerator:

$$F(a) = \int_0^1 ((1-a)\phi(\alpha_0, \beta_0; s) + a\phi(\alpha_1, \beta_1; s))^2 \mathrm{d}s \; .$$

We show that $F(a)$ can be expressed as a second-order polynomial in a whose coefficients depend only on the chosen values for $\alpha_1, \alpha_2, \beta_1$ and β_2.

Proof.

$$\begin{aligned} F(a) = & \int_0^1 ((1-a)\phi(\alpha_0, \beta_0; s) + a\phi(\alpha_1, \beta_1; s))^2 \, \mathrm{d}s \\ = & (1-a)^2 \int_0^1 (\phi(\alpha_0, \beta_0; s))^2 \, \mathrm{d}s \\ & + a^2 \int_0^1 (\phi(\alpha_1, \beta_1; s))^2 \, \mathrm{d}s \end{aligned}$$

$$
\begin{aligned}
&+ 2a(1-a)\int_0^1 \phi(\alpha_0,\beta_0;s)\phi(\alpha_1,\beta_1;s)\mathrm{d}s \\
=&(1-a)^2 + a^2 + 2a(1-a)K(\alpha_0,\alpha_1,\beta_0,\beta_1),
\end{aligned}
$$

where $K(\alpha_0,\alpha_1,\beta_0,\beta_1)$ is the constant term equal to:

$$
\frac{\int_0^1 s^{\alpha_0+\alpha_1-2}(1-s)^{\beta_0+\beta_1-1}\mathrm{d}s}{\sqrt{\int_0^1 y^{2(\alpha_0-1)}(1-y)^{2(\beta_0-1)}\mathrm{d}y \int_0^1 y^{2(\alpha_1-1)}(1-y)^{2(\beta_1-1)}\mathrm{d}y}},
$$

which can be expressed in terms of the *beta function* B as:

$$
K(\alpha_0,\alpha_1,\beta_0,\beta_1) = \frac{B(\alpha_0+\alpha_1-1,\beta_0+\beta_1-1)}{\sqrt{B(2\alpha_0-1,2\beta_0-1)B(2\alpha_1-1,2\beta_1-1)}}.
$$

□

A.4 Square integral of piecewise cubic profile

Let $a \in]0,1[$ Recall the definition of Φ_{cubic} through Hermite constraints:

$$
\begin{aligned}
\Phi_{\text{cubic}}(0) &= 0 & \Phi'_{\text{cubic}}(0) &= 0 \\
\Phi_{\text{cubic}}(1) &= 0 & \Phi'_{\text{cubic}}(1) &= 0 \\
\Phi_{\text{cubic}}(a) &= 1 & \Phi'_{\text{cubic}}(a) &= 0\,.
\end{aligned}
$$

We can show that there is only one piecewise third-degree polynomial Φ_{cubic} fitting these constraints.

$$
\Phi_{\text{cubic}}(s) = \begin{cases} x_0 + x_1 s + x_2 s^2 + x_3 s^3 & \text{if } s \in [0,a] \\ y_0 + y_1 s + y_2 s^2 + y_3 s^3 & \text{if } s \in [a,1] \end{cases}.
$$

The constraints above can be written as 8 equations (as the constraints at a yield one equation for each piece), yielding a linear system:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & a & a^2 & a^3 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2a & 3a^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 1 & a & a^2 & a^3 \\ 0 & 0 & 0 & 0 & 0 & 1 & 2a & 3a^2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

The determinant of the system is $a^4(a-1)^4$. This system is therefore invertible for any $a \in]0,1[$, yielding the following piecewise polynomial:

$$\Phi_{\text{cubic}}(s) = \begin{cases} -\frac{2}{a^3}s^3 + \frac{3}{a^2}s^2 & \text{if } s \in [0,a] \\ -\frac{2}{(a-1)^3}s^3 + \frac{3(a+1)}{(a-1)^3}s^2 - \frac{6a}{(a-1)^3}s + \frac{3a-1}{(a-1)^3} & \text{if } s \in [a,1] \end{cases}.$$

We then show that this family of functions also guarantee volume conservation regardless of a. More precisely we show that

$$I = \int_0^1 (\Phi_{\text{cubic}}(s))^2 \, \mathrm{d}s = \frac{13}{35}.$$

$$\begin{aligned} \int_0^1 (\Phi_{\text{cubic}}(s))^2 \mathrm{d}s &= \int_0^a (\Phi_{\text{cubic}}(s))^2 \mathrm{d}t + \int_a^1 (\Phi_{\text{cubic}}(s))^2 \mathrm{d}s \\ &= \int_0^a \left(-\frac{2}{a^3}s^3 + \frac{3}{a^2}s^2 \right)^2 \mathrm{d}s \\ &\quad + \int_a^1 \left(-\frac{2}{(a-1)^3}s^3 + \frac{3(a+1)}{(a-1)^3}s^2 - \frac{6a}{(a-1)^3}s + \frac{3a-1}{(a-1)^3} \right)^2 \mathrm{d}s \, . \end{aligned}$$

Integrating the first term is simple:

$$\begin{aligned} I_1 = \int_0^a \left(-\frac{2}{a^3}s^3 + \frac{3}{a^2}s^2 \right)^2 \mathrm{d}s &= \int_0^a \left(\frac{4}{a^6}s^6 - \frac{12}{a^5}s^5 + \frac{9}{a^4}s^4 \right) \mathrm{d}s \\ &= \frac{4a^7}{7a^6} - \frac{12a^6}{6a^5} + \frac{9a^5}{5a^4} \\ &= \frac{13a}{35} \, . \end{aligned}$$

The second term is more complex but still straightforward by expanding the polynomial in x, integrating each term and then expanding again the polynomial in a:

$$
\begin{aligned}
I_2 &= \int_a^1 \left(-\frac{2}{(a-1)^3}s^3 + \frac{3(a+1)}{(a-1)^3}s^2 - \frac{6a}{(a-1)^3}s + \frac{3a-1}{(a-1)^3} \right)^2 \mathrm{d}s \\
&= \frac{1}{(a-1)^6} \int_a^1 4s^6 - (12a+12)s^5 + (9a^2+42a+9)s^4 - (36a^2-48a-4)s^3 \\
&\quad + (54a^2+12a-6)s^2 - (36a^2-12a)s + (9a^2-6a+1) \\
&= \frac{1}{(a-1)^6}\Big(\frac{4(1-a^7)}{7} + \frac{(-12a-12)(1-a^6)}{6} \\
&\quad + \frac{(9a^2+42a+9)(1-a^5)}{5} + \frac{(-36a^2-48a+4)(1-a^4)}{4} \\
&\quad + \frac{(54a^2+12a-6)(1-a^3)}{3} + \frac{(-36a^2+12a)(1-a^2)}{2} + (9a^2-6a+1)(1-a) \Big) \\
&= \frac{1}{(a-1)^6} \left(-\frac{13}{35}a^7 + \frac{13}{5}a^6 - \frac{39}{5}a^5 + 13a^4 - 13a^3 + \frac{39}{5}a^2 - \frac{13}{5}a + \frac{13}{35} \right) \\
&= \frac{13}{35}\frac{(1-a)^7}{(1-a)^6} \\
&= \frac{13}{35}(1-a) \ .
\end{aligned}
$$

Finally, for the whole integral, a vanishes:

$$
\begin{aligned}
I &= I_1 + I_2 \\
&= \frac{13}{35}(a + (1-a)) \\
&= \frac{13}{35} \ .
\end{aligned}
$$

B

Résumé en français

B.1 Introduction

L'animation de personnages est un composant central des médias numériques modernes : films, jeux vidéos ou réalité virtuelle. Produire des déformations de qualité sur un modèle de personnage, ce qui est crucial pour rendre celui-ci crédible aux yeux du spectateur, demeure néanmoins un défi technique. La complexité du corps humain, composé d'os, de muscles, et de divers tissus organiques est en effet difficile à reproduire sous forme numérique tout en maintenant des vitesses d'exécution élevées et une simplicité de paramétrisation. Idéalement, le calcul des déformations du personnage s'exécuterait en temps réel sur une machine standard, et offrirait à son utilisateur des paramètres intuitifs permettant un contrôle précis du résultat. Le modèle de base est celui de la déformation basée sur un squelette articulé ; l'addition de déformations causées par les muscles sur la peau améliore significativement la plausibilité du résultat. Pour prendre en compte ces déformations musculaires, de nombreuses techniques ont été développées, allant des méthodes géométriques rapides en calcul aux simulations physiques détaillées mais coûteuses. Les travaux de cette thèse explorent l'adjonction de déformations musculaires aux méthodes géométrique de skinning par squelette. Parmi ces approches, trois familles ont chacune étendu le Linear Blend Skinning (LBS) [MLT88] et le Dual Quaternion Skinning [Kav+07]. La première de ces familles regroupe les méthodes utilisant les poses clefs [LCF00] sculptées directement avec des logiciels de modélisation. Ces méthodes sont très générales, mais cette étape de modélisation manuelle est particulièrement fastidieuse. La deuxième famille utilise des primitives musculaires, positionnées à l'intérieur du corps du personnages, agissant comme déformeurs sur la peau [WV97]. Ils sont généralement contrôlables à l'aide de paramètres intuitifs et éditables en temps réels, mais souffrent

des limitations inhérentes aux méthodes de skinning sous-jacentes (LBS et DQS), c'est à dire le manque de prise en compte des contacts de la peau et du volume au niveau des articulations. La troisième famille utilise des base de données capturées pour inférer les déformations sur les maillages [MG03 ; WPP07]. Les déformations produites sont toutefois limités par la taille des données d'apprentissages et la difficulté pour les artistes de contrôler le résultat.

Nous décrivons une approche qui permet de cumuler les avantages des primitives musculaires avec une méthode de skinning géométrique produisant des résultat de bonne qualité, notamment au niveau de sa gestion des contacts : l'Implicit Skinning [Vai+13 ; Vai+14]. Cette méthode permet de corriger les auto-intersections du maillage et de représenter l'élasticité de la peau. Nous y ajoutons la modélisation des muscles, de leur interaction avec d'autres éléments anatomiques comme les os, et des effets dynamiques, crées par la mise en mouvement du squelette d'animation. En particulier, nous présentons de nouvelles primitives musculaires qui imitent les formes des muscles et reproduisent leur déformations : flexion, extension et activation, en maintenant leur volume constant, à l'instar des muscles réels. Ces primitives musculaires sont représentées par des surfaces de balayage autour d'un axe central. La dynamique des muscles est induite sur lesdits axes par Position Based Dynamics [Mül+07], une méthode de simulation physique approximative. L'utilisation de champs scalaires 3D pour nos primitives musculaires nous permet de l'intégrer dans le cadre de l'Implicit Skinning, et de détecter simplement les collisions entre muscles ou avec les os. La mise en œuvre de ces primitives permet de contrôler les formes des muscles virtuels et leur comportement dynamique à l'aide d'un petit nombre de paramètres intuitifs, évitant ainsi de fastidieuses phases de modélisation de formes correctives ou de coûteuses simulations physiques.

B.2 Primitive musculaire

Un muscle est défini comme un champ scalaire $f_M : \mathbb{R}^3 \to \mathbb{R}$ construit par balayage d'une fonction profil R le long d'une ligne polygonale $\mathscr{C}$. Ce champ scalaire continu définit en sa 0-isovaleur une surface implcite représentant la surface du muscle.

L'évaluation de f_M en un point $\mathbf{q}$ consiste en trois étapes : la construction de la ligne polygonale centrale $\mathscr{C}$, la projection de $\mathbf{q}$ sur celle-ci, ce qui permet de calculer la distance d entre le point et la courbe, et l'évaluation du profil de balayage $R(\mathbf{q})$. La valeur du champ scalaire en $\mathbf{q}$ est alors :

$$f_M(\mathbf{q}) = d - R(\mathbf{q}) \ .$$

Construction de l'axe central Les deux points extrémaux $\mathbf{m}_0$ et $\mathbf{m}_1$ de la primitive musculaire sont attachés à un os d'animation, et y sont liés rigidement pendant l'animation. Le segment $[\mathbf{m}_0\mathbf{m}_1]$ est alors divisé en N_M segments égaux, dont les points intermédiaires sont notés $\mathbf{p}_i$. La ligne polygonale (polyligne) $\mathscr{C}$ est paramétrée par $s \in [0,1]$ et on note $s(\mathbf{q})$ le paramètre correspondant à la projection d'un point $\mathbf{q}$ quelconque sur $\mathscr{C}$. Afin de modéliser des profils de coupe elliptiques, la polyligne est orientée à chaque extrémité par un vecteur normal, respectivement $\vec{\mathbf{n}}_{\mathbf{m}_0}$ et $\vec{\mathbf{n}}_{\mathbf{m}_1}$. Chaque point de contrôle $\mathbf{p}_i$ se voit associer une normale $\vec{\mathbf{n}}_i$ de la manière suivante :

1. Interpolation sphérique des normales par rapport aux extrémités

2. projection du vecteur interpolé sur le plan tangent formé par les deux segments ayant leur extrémité en $\mathbf{p}_i$.

La normale en un point quelconque de la polyligne est interpolée sphériquement entre les normales des points de contrôle du segment sur lequel il se trouve.

Opérateur de projection La projection d'un point $\mathbf{q}$ sur $\mathscr{C}$ est une étape critique de l'évaluation de la fonction f_M car elle influe sur les propriétés du champ scalaire final, notamment la continuité. La projection orthogonale n'est pas satisfaisante car elle génère des discontinuités dans les régions intérieures des angles formés par les segments consécutifs (voir Figure 4.6). Nous proposons une nouvelle paramétrisation de la projection d'un point $\mathbf{q}$ lorsqu'il se projette orthogonalement à l'intérieur d'un angle de deux segments consécutifs. Le schéma de la Figure 4.8 illustre le procédé : le paramètre curviligne du point projeté sur $\mathscr{C}$ est une combinaison linéaire des paramètres du point projeté sur chacun des segments pondérés par la cotangente de l'angle λ_i formé avec les segments respectifs.

Surface de balayage La surface est définie en extrudant le profil R le long de l'axe $\mathscr{C}$. Pour définir la forme des muscles, la fonction de profil R est paramétrée par l'abscisse curviligne s et l'angle θ formé entre le vecteur reliant $\mathbf{q}$ et son projeté d'une part et le vecteur normal en ce point d'autre part. R est définie comme étant séparable en chacune des variables, c'est à dire :

$$R = wr(\theta)\Phi(s) \ .$$

La fonction Φ représente la distribution de la masse musculaire le long de l'axe et r sa coupe transversale, tandis que w est un facteur d'échelle. Cette séparabilité permet d'établir que le volume du muscle reste constant si et seulement si les fonctions Φ et r

vérifient :

$$\int_{s=0}^{1} (\Phi(s))^2 \, \mathrm{d}s = \text{constante} \, , \tag{B.1}$$

$$\int_{\theta=0}^{2\pi} \frac{(r(\theta))^2}{2} \mathrm{d}\theta = \text{constante} \, . \tag{B.2}$$

Sous ces conditions, en choisissant les constantes égales à 1, le volume du muscle lorsque son axe est une ligne droite s'écrit :

$$V = \pi w^2 l, \, ,$$

où l est la longueur de l'axe, comme démontré dans l'Annexe A.2. L'équation précédente permet de modéliser la contraction et l'extension des muscles. Les points extrémaux étant liés à des os d'animations différents, la longueur de l'axe varie au fil des mouvements du personnage. Lorsque celle-ci se raccourcit d'un facteur γ, maintenir le volume constant nécessite donc de faire grossir la largeur w d'un facteur $\sqrt{\gamma}$, et inversement lorsque l'axe musculaire s'allonge, reproduisant ainsi les contractions et extensions isotoniques du muscle.

Fonctions de profil Nous présentons ci-dessous des fonctions vérifiant ces équations et disposant de paramètres permettant de contrôler efficacement la forme résultante. Premièrement, la fonction de distribution de masse Φ, est inspirée de la loi bêta de probabilités :

$$\Phi(s) = \phi(\alpha, \beta; s),$$

où α et β sont deux paramètres entiers (> 1) contrôlant la forme du profil. La fonction ϕ est définie de sorte à ce que son carré s'intègre à 1, c'est à dire une fonction unitaire au sens de la norme $\mathcal{L}^2$:

$$\phi(\alpha, \beta; s) = \frac{s^{\alpha-1}(1-s)^{\beta-1}}{\sqrt{\int_0^1 y^{2(\alpha-1)}(1-y)^{2(\beta-1)}\mathrm{d}y}}.$$

Le ratio α/β contrôle l'asymétrie de la distribution (les profils symétriques sont pour $\alpha = \beta$) et les valeurs individuelles de α et β déterminent l'aspect de la fonction à chaque extrémité.

Nous proposons de modéliser l'activation en interpolant entre deux paires de paramètres (α_0, β_0) et (α_1, β_1) selon un paramètre d'interpolation $a \in [0, 1]$. Le premier jeu de paramètres représente la forme du muscle au repos et le second la forme du muscle au maximum de son activation (contraction isométrique). Pour préserver le volume du

muscle et respecter les équations, on emploie l'interpolation sphérique-linéaire (au sens de $\mathcal{L}^2$) entre les deux fonctions $\phi(\alpha_0, \beta_0; s)$ et $\phi(\alpha_1, \beta_1; s)$. On montre à l'Annexe A.3 que la normalisation de la fonction interpolée est équivalente à un polynôme du second degré en a, permettant d'évaluer cette fonction à un coût d'évaluation faible.

Le profil de section transversale est une ellipse contrôlée par un seul paramètre d'excentricité e, les longueurs des demi-axes u et v de l'ellipse s'en déduisant déduisant par l'Équation (B.2).

$$u = \sqrt[4]{(1 - e^2)} \quad v = 1/u.$$

B.3 Déformations dynamiques

Le modèle de muscle décrit précédemment est purement cinématique. Dans cette section nous présentons l'addition d'effets dynamiques par l'introduction d'une simulation physique. Les points de contrôle $\mathbf{p}_i$ de l'axe central $\mathscr{C}$ peuvent maintenant se déplacer selon les résultats d'un moteur physique simplifié : Position based dynamics (PBD) [Mül+07]. Chaque point est représenté dans ce cadre par une particule de masse m_i représentant une fraction de la masse du muscle(les extrémités demeurent liées cinématiquement à leur os d'animation). L'approche PBD prend en compte non pas les forces agissant sur les particules, mais des fonctions de contraintes, qui sont optimisées par le solveur Gauss-Seidel de PBD. Ce solveur converge itérativement vers une position des particules satisfaisant aux contraintes. Notre approche utilisant les champs scalaires est ici un atout, puisqu'elle permet de définir des contraintes modélisant efficacement les collisions entre objets.

Contraintes d'élasticité Nous introduisons une contrainte de distance élastique entre les particules de l'axe musculaire se comportant comme des ressorts, caractérisés par une raideur k et une distance de repos d_0, ces paramètres déterminant le comportement dynamique du muscle : une faible raideur créera beaucoup d'effets inertiels et des secousses visibles, alors qu'une raideur forte maintient l'axe du muscle quasi immobile relativement à ses extrémités.

Détection des collisions Les collisions sont évitées en introduisant des contraintes entre une particule et un champ scalaire, c'est à dire en forçant une particule à demeurer à une certaine distance de la surface implicite correspondante. Chaque particule se comporte alors comme une sphère, et les os sont représentés soit par des champs de distance discrétisés par rapport à un modèle anatomique existant ou comme de simple cylindres. La prise en compte des os donne des déformations musculaires plus réalistes. De même

nous proposons de modéliser la collision entre muscles : chaque particule de l'axe d'un muscle est repoussée par le champ scalaire correspondant aux autres muscles proches. Enfin, nous proposons une modélisation de l'interaction entre le muscle et la peau, en contraignant les particules de muscles à rester à une certaine distance à l'intérieur de la peau telle que définie dans la position initiale du personnage.

B.4 Intégration à Implicit Skinning

Opérateurs de mélange Les déformations générées par les muscles sont intégrées dans la représentation implicite de la peau proposée par Implicit Skinning [Vai+13; Vai+14]. Chaque champ scalaire musculaire f_M est un champ de distance à support global. Pour être intégrés avec les champs scalaires de l'Implicit Skinning, il est tout d'abord nécessaire de les convertir en fonction à support compact [BWG04]. Nous utilisons pour cela la fonction compactifiante utilisée par Vaillant et al. [Vai+13]. Les différents muscles appartenant à une partie du corps distincte associée à un os d'animation sont alors mélangées au champ scalaire HRBF représentant la surface de la peau, pour produire un nouveau champ scalaire.

Ces différents champs scalaires, représentant chacun une partie mobile du corps du personnages, sont ensuite combinés au moyen de l'opérateur de contact décrit par Vaillant et al. [Vai+14] pour représenter le personnage entier. Durant l'animation, les sommets du maillage représentant le personnage suivent cette surface implicite se déformant selon les mouvements du squelette, ce qui permet à la fois d'éviter les auto-intersections, et de prendre en compte les effets des muscles.

Toutefois, le mélange entre les muscles et la HRBF requiert une attention particulière. En utilisant un opérateur d'union, le champ scalaire résultant présente des discontinuités et des points critiques à proximité de la surface de la peau. Or, les points critiques induisent l'algorithme de suivi d'Implicit Skinning en erreur, et provoquent des artefacts visuels qu'il convient d'éviter. Pour cela, nous utilisons l'opérateur de détail décrit par Canezin et al. [CGB13], qui évite l'apparition de ces points singuliers.

Pour que l'effet du muscle soit visible en extension comme en flexion, les HRBFs représentant la peau du modèle sont modifiées de sorte à ce que ce soit le muscle qui capture la forme de la peau et non la HRBF statique. Ainsi lorsque le muscle se dégonfle, la peau est visiblement creusée.

Intégration à l'algorithme d'animation La mise à jour de l'algorithme Implicit Skinning tenant compte des muscles est illustrée par la Figure 6.5. À chaque nouvelle trame, le squelette d'animation est transformé pour atteindre sa pose cible. Les éléments liés cinématiquement

au squelette d'animation (HRBF et particules extrémales) sont mis à jour. Les paramètres de forme du muscle (α, β, l'excentricité ou l'activation) peuvent être éventuellement spécifiés sous forme de *keyframes* et sont calculés à ce moment. Ensuite, le solveur PBD calcule la nouvelle position des particules, et les champs scalaires représentant les muscles sont mis à jour. Cela permet d'atteindre la représentation implicite de la peau pour cette trame, qui sera utilisé pour corriger la position des sommets par l'algorithme Implicit Skinning.

B.5 Résultats

Pour tester cette méthode d'animation, nous avons utilisé plusieurs scènes allant de mouvements simples comme une flexion de l'avant-bras ou un mouvement oscillant du bras, à des mouvements plus complexes comme la course ou le saut. Les résultats obtenus démontrent la possibilité d'exécuter cette méthode à des vitesses permettant son usage interactif, allant d'environ 30 images par secondes dans les cas les plus simples à 2 images par secondes dans les plus complexes. La complexité des scènes dégrade les performances assez rapidement, mais lors des sessions d'édition, il est possible de favoriser la vitesse au détriment de la précision en réduisant le nombre de particules par muscles, ou en simulant uniquement les muscles en cours d'édition. De plus, la mise en œuvre de notre méthode pourrait également être rendue plus rapide en exécutant les tâches les plus coûteux (la simulation physique et l'algorithme de suivi d'Implicit Skinning) sur un GPU. En effet, ces deux calculs sont essentiellement parallélisables, et bénéficieraient grandement d'une implémentation sur carte graphique.

B.6 Conclusion

Les travaux présentés dans cette thèse mettent donc en évidence qu'il est possible d'améliorer la plausibilité des résultats d'Implicit Skinning par l'adjonction de primitives représentant l'effet des muscles en conservant des performances suffisantes pour l'interactivité. La formulation implicite des formes des muscles est un atout pour l'évaluation du volume en formule fermée et donc pour dériver des règles permettant la déformation des muscles à volume constant. De plus, cette formulation s'intègre élégamment dans le modèle physique de Position based dynamics, permettant d'ajouter des effets dynamiques dans le mouvement des muscles à moindre frais calculatoires. Enfin, ces travaux augurent de l'applicabilité de méthodes inspirées de l'Implicit Skinning dans un contexte de production infographique.

Bibliography

[ACP02] Brett Allen, Brian Curless, and Zoran Popović. "Articulated Body Deformation from Range Scan Data". In: *ACM Transactions on Graphics* 21.3 (July 2002), pp. 612–619.

[AF15] Nadine Abu Rumman and Marco Fratarcangeli. "Position-Based Skinning for Soft Articulated Characters". In: *Computer Graphics Forum* 34.2 (2015). Proceedings of Eurographics, pp. 240–250.

[AF16] Nadine Abu Rumman and Marco Fratarcangeli. "State of the Art in Skinning Techniques for Articulated Deformable Characters". In: *Proceedings of the International Conference on Computer Graphics Theory and Applications*. GRAPP 2016. Rome, Italy, Feb. 2016.

[AHH08] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-time rendering*. 3rd ed. Natick, Massachusetts: A K Peters Ltd., 2008.

[Ale02] Marc Alexa. "Linear Combination of Transformations". In: *ACM Transactions on Graphics* 21.3 (July 2002), pp. 380–387.

[Ali+13] Dicko Ali-Hamadi, Tiantian Liu, Benjamin Gilles, Ladislav Kavan, François Faure, Olivier Palombi, and Marie-Paule Cani. "Anatomy Transfer". In: *ACM Transactions on Graphics* 32.6 (Nov. 2013), 188:1–188:8.

[All+06] Brett Allen, Brian Curless, Zoran Popović, and Aaron Hertzmann. "Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '06. Eurographics Association. 2006, pp. 147–156.

[AM00] Marc Alexa and Wolfgang Müller. "Representing Animations by Principal Components". In: *Computer Graphics Forum* 19.3 (2000), pp. 411–418.

[Ang+05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. "SCAPE: Shape Completion and Animation of People". In: *ACM Transactions on Graphics* 24.3 (July 2005), pp. 408–416.

[Ang+17] Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi. "Sketch-based Implicit Blending". In: *ACM Transactions on Graphics* 36.6 (Nov. 2017), 181:1–181:13.

[AR10] Richard A. Askey and Ranjan Roy. "The Euler Beta Function". In: *NIST Digital Library of Mathematical Functions*. Ed. by Frank W. J. Olver, Adri B. Olde Daalhuis, Daniel W. Lozier, Barry I. Schneider, Ronald F. Boisvert, Charles W. Clark, Bruce R. Miller, and Bonita V. Saunders. Release 1.0.18. Cambridge University Press, 2010. Chap. 5.12.

[BA02] J Andreas Bærentzen and Henrik Aanæs. *Generating signed distance fields from triangle meshes*. Tech. rep. 2002-21. DK-2800 Kongens Lyngby - Denmark: Informatics and Mathematical Modeling, Technical University of Denmark, 2002.

[Bar+03] Loïc Barthe, Neil. A. Dodgson, Malcolm A. Sabin, Brian Wyvill, and Véronique Gaildrat. "Two-dimensional Potential Fields for Advanced Implicit Modeling Operators". In: *Computer Graphics Forum* 22.1 (2003), pp. 23–33.

[Bar84] Alan H. Barr. "Global and Local Deformations of Solid Primitives". In: *SIGGRAPH Computer Graphics* 18.3 (Jan. 1984), pp. 21–30.

Bibliography

[Ben+14] Jan Bender, Matthias Müller, Miguel A. Otaduy, Matthias Teschner, and Miles Macklin. "A Survey on Position-Based Simulation Methods in Computer Graphics". In: *Computer Graphics Forum* 33.6 (2014), pp. 228–251.

[Ber+10] Adrien Bernhardt, Loïc Barthe, Marie-Paule Cani, and Brian Wyvill. "Implicit blending revisited". In: *Computer Graphics Forum* 29.2 (2010), pp. 367–375.

[BGC01] Loïc Barthe, Véronique Gaildrat, and René Caubet. "Extrusion of 1D implicit profiles: Theory and first application". In: *International Journal of Shape Modeling* 7.2 (2001), pp. 179–199.

[Bli82] James F. Blinn. "A Generalization of Algebraic Surface Drawing". In: *ACM Transactions on Graphics* 1.3 (July 1982), pp. 235–256.

[Blo97] Jules Bloomenthal, ed. *Introduction to implicit surfaces*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 1997.

[BMM15] Jan Bender, Matthias Müller, and Miles Macklin. "Position-Based Simulation Methods in Computer Graphics". In: *Tutorial Proceedings of Eurographics*. Zurich, Switzerland, Apr. 2015.

[BN07] Antoine Bouthors and Matthieu Nesme. "Twinned Meshes for Dynamic Triangulation of Implicit Surfaces". In: *Proceedings of Graphics Interface*. GI '07. Montreal, Canada: ACM, 2007, pp. 3–9.

[Bot+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. Ed. by A K Peters. 5 Commonwealth Road, Suite 2C, Natick, Massachusetts.: CRC press, 2010.

[Bri07] Robert Bridson. "Fast Poisson Disk Sampling in Arbitrary Dimensions". In: *ACM SIGGRAPH 2007 Sketches*. SIGGRAPH '07. San Diego, California: ACM, 2007.

[BWG04] Loïc Barthe, Brian Wyvill, and Erwin de Groot. "Controllable binary CSG operator for "soft objects"". In: *International Journal of Shape Modeling* 10.2 (2004), pp. 135–154.

[Can16] Florian Canezin. "Study of the Composition Models of Field Functions in Computer Graphics". PhD thesis. Université de Toulouse, Sept. 2016.

[Cap+02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. "Interactive Skeleton-driven Dynamic Deformations". In: *ACM Transactions on Graphics* 21.3 (July 2002), pp. 586–593.

[Cap+05] Steve Capell, Matthew Burkhart, Brian Curless, Tom Duchamp, and Zoran Popović. "Physically Based Rigging for Deformable Characters". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '05. Los Angeles, California: ACM, 2005, pp. 301–310.

[Cas+16] Dan Casas, Andrew Feng, Oleg Alexander, Graham Fyffe, Paul Debevec, Ryosuke Ichikari, Hao Li, Kyle Olszewski, Evan Suma, and Ari Shapiro. "Rapid Photorealistic Blendshape Modeling from RGB-D Sensors". In: *Proceedings of the 29th International Conference on Computer Animation and Social Agents*. CASA '16. Geneva, Switzerland: ACM, 2016, pp. 121–129.

[CBS96] Benoît Crespin, Carole Blanc, and Christophe Schlick. "Implicit Sweep Objects". In: *Computer Graphics Forum* 15.3 (1996), pp. 165–174.

[CGB13] Florian Canezin, Gaël Guennebaud, and Loic Barthe. "Adequate inner bound for geometric modeling with compact field functions". In: *Computers & Graphics* 37.6 (2013). Shape Modeling International (SMI) Conference 2013, pp. 565–573.

[CGB16] Florian Canezin, Gael Guennebaud, and Loïc Barthe. "Topology-Aware Neighborhoods for Point-Based Simulation and Reconstruction". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '16. Zurich, Switzerland, July 2016.

[CHP89] J. E. Chadwick, David. R. Haumann, and Richard. E. Parent. "Layered Construction for Deformable Animated Characters". In: *SIGGRAPH Computer Graphics* 23.3 (July 1989), pp. 243–252.

[CZ92] David T. Chen and David Zeltzer. "Pump It Up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method". In: *SIGGRAPH Computer Graphics* 26.2 (July 1992), pp. 89–98.

[DC98] Mathieu Desbrun and Marie-Paule Cani. "Active Implicit Surface for Animation". In: *Proceedings of Graphics Interface*. GI 1998. Published under the name Marie-Paule Cani-Gascuel. The Canadian Human-Computer Communications Society. Vancouver, Canada, June 1998, pp. 143–150.

[DCB14] Crispin Deul, Patrick Charrier, and Jan Bender. "Position-Based Rigid Body Dynamics". In: *Computer Animation and Virtual Worlds* 27.2 (2014), pp. 103–112.

[Del+07] Scott L. Delp, Franck C. Anderson, Allison S. Arnold, Peter Loan, Ayman Habib, John T. Chand, Eran Guendelman, and Darryl G. Thelen. "OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement". In: *IEEE Transactions on Biomedical Engineering* 54.11 (Nov. 2007), pp. 1940–1950.

[FLP14] Ye Fan, Joshua Litven, and Dinesh K. Pai. "Active Volumetric Musculoskeletal Systems". In: *ACM Transactions on Graphics* 33.4 (July 2014), 152:1–152:9.

[FMM13] Makoto Fujisawa, Yojiro Mandachi, and Kenjiro T. Miura. "Calculation of Velocity on an Implicit Surface by Curvature Invariance". In: *Journal of Information Processing* 21.4 (Oct. 2013), pp. 674–680.

[GH99] Cindy Grimm and John Hughes. "Implicit generalized cylinders using profile curves". In: *Proceedings of Implicit Surfaces*. Bordeaux, France: ACM, Sept. 1999, pp. 33–41.

[GM85] Michael Girard and Anthony A. Maciejewski. "Computational Modeling for the Computer Animation of Legged Figures". In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '85. New York, NY, USA: ACM, 1985, pp. 263–270.

[GMS14] Ming Gao, Nathan Mitchell, and Eftychios Sifakis. "Steklov-Poincaré Skinning". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '14. Copenhagen, Denmark: Eurographics Association, 2014, pp. 139–148.

[Gol05] Ron Goldman. "Curvature formulas for implicit curves and surfaces". In: *Computer Aided Geometric Design* 22.7 (2005). Geometric Modelling and Differential Geometry, pp. 632–658.

[Gou+13] Olivier Gourmel, Loïc Barthe, Marie-Paule Cani, Brian Wyvill, Adrien Bernhardt, Mathias Paulin, and Herbert Grasberger. "A Gradient-based Implicit Blend". In: *ACM Transactions on Graphics* 32.2 (Apr. 2013), 12:1–12:12.

[Gra18] Henry Gray. *Anatomy of the Human Body*. Ed. by Warren H. Lewis. 20th ed. Philadelphia: Lea and Febiger, 1918.

[Hah+12] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. "Rig-space physics". In: *ACM Transactions on Graphics* 31.4 (July 2012), 72:1–72:8.

[Ham44] William Rowan Hamilton. "On quaternions; or on a new system of imaginaries in algebra". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 25.163 (1844), pp. 10–13.

[HC32] David Hilbert and Stephan Cohn-Vossen. *Anschauliche Geometrie*. Ed. by Julius Springer. Berlin: Heidelberg, 1932.

Bibliography

[HH85] Christoph Hoffmann and John Hopcroft. "Automatic surface generation in computer aided design". In: *The Visual Computer* 1.2 (1985), pp. 92–100.

[HHP15] Minyeon Han, Jisoo Hong, and F.C. Park. "Musculoskeletal dynamics simulation using shape-varying muscle mass models". In: *Multibody System Dynamics* 33.4 (2015), pp. 367–388.

[Hir+12] David A. Hirshberg, Matthew Loper, Eric Rachlin, and Michael J. Black. "Coregistration: Simultaneous Alignment and Modeling of Articulated 3D Shape". In: *Proceedings of the European Conference on Computer Vision*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. ECCV 2012 6. Florence, Italy: Springer Berlin Heidelberg, Oct. 2012, pp. 242–255.

[HSK16] Daniel Holden, Jun Saito, and Taku Komura. "A Deep Learning Framework for Character Motion Synthesis and Editing". In: *ACM Transactions on Graphics* 35.4 (July 2016), 138:1–138:11.

[Hug+13] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. "Implicit Representation of Shape". In: *Computer Graphics: Principles and Practice*. 3rd ed. Boston, MA, USA: Addison-Wesley Professional, July 2013. Chap. 24, p. 1264.

[IBP15] Alexandru Eugen Ichim, Sofien Bouaziz, and Mark Pauly. "Dynamic 3D Avatar Creation from Hand-held Video Input". In: *ACM Transactions on Graphics* 34.4 (July 2015), 45:1–45:14.

[Jac+11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. "Bounded Biharmonic Weights for Real-time Deformation". In: *ACM Transactions on Graphics* 30.4 (July 2011), 78:1–78:8.

[Jac+14] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and John P. Lewis. "Skinning: Real-time Shape Deformation". In: *ACM SIGGRAPH Courses*. SIGGRAPH 2014. Vancouver, Canada, Aug. 2014.

[Jak01] Thomas Jakobsen. *Advanced Character Physics*. gamastutra.com. Jan. 2001.

[JS11] Alec Jacobson and Olga Sorkine. "Stretchable and Twistable Bones for Skeletal Shape Deformation". In: *ACM Transactions on Graphics* 30.6 (Dec. 2011), 165:1–165:8.

[Kad+16] Petr Kadleček, Alexandru-Eugen Ichim, Tiantian Liu, Jaroslav Křivánek, and Ladislav Kavan. "Reconstructing Personalized Anatomical Models for Physics-based Body Animation". In: *ACM Transactions on Graphics* 35.6 (Nov. 2016), 213:1–213:13.

[Kar90] Troels Kardel. "Niels Stensen's geometrical theory of muscle contraction (1667): A reappraisal". In: *Journal of Biomechanics* 23.10 (1990), pp. 953–965.

[Kav+07] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. "Skinning with Dual Quaternions". In: *Proceedings of the Symposium on Interactive 3D Graphics and Games*. I3D '07. Seattle, Washington: ACM, Apr. 2007, pp. 39–46.

[Kav+08] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. "Geometric Skinning with Approximate Dual Quaternion Blending". In: *ACM Transactions on Graphics* 27.4 (Nov. 2008), p. 105.

[KCO09] Ladislav Kavan, Steven Collins, and Carol O'Sullivan. "Automatic Linearization of Nonlinear Skinning". In: *Proceedings of the Symposium on Interactive 3D Graphics and Games*. I3D '09. Boston, Massachusetts: ACM, 2009, pp. 49–56.

[KH14] YoungBeom Kim and JungHyun Han. "Bulging-free dual quaternion skinning". In: *Computer Animation and Virtual Worlds* 25.3-4 (2014), pp. 321–329.

[Kim+17] Meekyoung Kim, Gerard Pons-Moll, Sergi Pujades, Seungbae Bang, Jinwook Kim, Michael J. Black, and Sung-Hee Lee. "Data-driven Physics for Human Soft Tissue Animation". In: *ACM Transactions on Graphics* 36.4 (July 2017), 54:1–54:12.

[KJP02] Paul G. Kry, Doug L. James, and Dinesh K. Pai. "EigenSkin: Real Time Large Deformation Character Skinning in Hardware". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '02. San Antonio, Texas: ACM, July 2002, pp. 153–159.

[KM04] Tsuneya Kurihara and Natsuki Miyata. "Modeling Deformable Human Hands from Medical Images". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '04. Grenoble, France: Eurographics Association, 2004, pp. 355–363.

[KP11] Junggon Kim and Nancy S. Pollard. "Fast Simulation of Skeleton-driven Deformable Body Characters". In: *ACM Transactions on Graphics* 30.5 (Oct. 2011), 121:1–121:19.

[KS12] Ladislav Kavan and Olga Sorkine. "Elasticity-Inspired Deformers for Character Articulation". In: *ACM Transactions on Graphics* 31.6 (Nov. 2012), 196:1–196:8.

[KVS99] Leif Kobbelt, Jens Vorsatz, and Hans-Peter Seidel. "Multiresolution hierarchies on unstructured triangle meshes". In: *Computational Geometry* 14.1-3 (1999), pp. 5–24.

[KŽ05] Ladislav Kavan and Jiří Žára. "Spherical Blend Skinning: A Real-time Deformation of Articulated Models". In: *Proceedings of the Symposium on Interactive 3D Graphics and Games*. I3D '05. Washington, District of Columbia: ACM, Apr. 2005, pp. 9–16.

[LA07] Keng Siang Lee and Golam Ashraf. "Simplified Muscle Dynamics for Appealing Real-Time Skin Deformation". In: *Proceedings of the International Conference on Computer Graphics and Virtual Reality*. CGVR'07. Las Vegas, Nevada: CSREA Press, June 2007, pp. 160–168.

[LAG01] Antoine Leclercq, Samir Akkouche, and Éric Galin. "Mixing Triangle Meshes and Implicit Surfaces in Character Animation". In: *Proceedings of Eurographics Workshop on Computer Animation and Simulation*. 2001, pp. 37–47.

[LC87] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *SIGGRAPH Computer Graphics* 21.4 (Aug. 1987), pp. 163–169.

[LCF00] John P. Lewis, Matt Cordner, and Nickson Fong. "Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-driven Deformation". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. New Orleans, USA: ACM Press/Addison-Wesley Publishing Co., July 2000, pp. 165–172.

[Lee+12] Dongwoon Lee, Michael Glueck, Azam Khan, Eugene Fiume, and Ken Jackson. "Modeling and simulation of skeletal muscle for computer graphics: A survey". In: *Foundations and Trends in Computer Graphics and Vision* 7.4 (Apr. 2012), pp. 229–276.

[Lew+14] John P. Lewis, Ken Anjyo, Taehyun Rhee, Mengjie Zhang, Fred Pighin, and Zhigang Deng. "Practice and Theory of Blendshape Facial Models". In: *Eurographics 2014 - State of the Art Reports*. Ed. by Sylvain Lefebvre and Michela Spagnuolo. The Eurographics Association, 2014.

[LH09] Gene S. Lee and Frank Hanner. "Practical Experiences with Pose Space Deformation". In: *ACM SIGGRAPH Asia 2009 Sketches*. SIGGRAPH Asia '09. Yokohama, Japan: ACM, Dec. 2009, 43:1–43:1.

[LH16] Binh Huy Le and Jessica K. Hodgins. "Real-time Skeletal Skinning with Optimized Centers of Rotation". In: *ACM Transactions on Graphics* 35.4 (July 2016), 37:1–37:10.

[Li+13] Duo Li, Shinjiro Sueda, Debanga R. Neog, and Dinesh K. Pai. "Thin Skin Elastodynamics". In: *ACM Transactions on Graphics* 32.4 (July 2013), pp. 491–4910.

[Liu+13] Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. "Simulation and Control of Skeleton-driven Soft Body Characters". In: *ACM Transactions on Graphics* 32.6 (Nov. 2013), 215:1–215:8.

[LMB14] Matthew Loper, Naureen Mahmood, and Michael J. Black. "MoSh: Motion and shape capture from sparse markers". In: *ACM Transactions on Graphics* 33.6 (2014), 220:1–220:13.

[Lop+15] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. "SMPL: A Skinned Multi-Person Linear Model". In: *ACM Transactions on Graphics* 34.6 (Oct. 2015). Proceedings of SIGGRAPH Asia, 248:1–248:16.

[LST09] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. "Comprehensive Biomechanical Modeling and Simulation of the Upper Body". In: *ACM Transactions on Graphics* 28.4 (Sept. 2009), 99:1–99:17.

[Mac+14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. "Unified Particle Physics for Real-time Applications". In: *ACM Transactions on Graphics* 33.4 (July 2014), 153:1–153:12.

[McA+11] Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. "Efficient Elasticity for Character Skinning with Contact and Collisions". In: *ACM Transactions on Graphics* 30.4 (July 2011), 37:1–37:12.

[MG03] Alex Mohr and Michael Gleicher. "Building Efficient, Accurate Character Skins from Examples". In: *ACM Transactions on Graphics* 22.3 (July 2003), pp. 562–568.

[MGV11] Ives Macêdo, João Paulo Gois, and Luiz Velho. "Hermite Radial Basis Functions Implicits". In: *Computer Graphics Forum* 30.1 (2011), pp. 27–42.

[Min+00] Kyung-Ha Min, Seung-Min Baek, Gun Lee, Haeock Choi, and Chan-Mo Park. "Anatomically-based modeling and animation of human upper limbs". In: *Proceedings of the International Conference on Human Modeling and Animation*. Jan. 2000.

[MK16] Tomohiko Mukai and Shigeru Kuriyama. "Efficient Dynamic Skinning with Low-rank Helper Bone Controllers". In: *ACM Transactions on Graphics* 35.4 (July 2016), 36:1–36:11.

[MLT88] Nadia Magnenat-Thalmann, Richard Laperrire, and Daniel Thalmann. "Joint-Dependent Local Deformations for Hand Animation and Object Grasping". In: *Proceedings of Graphics Interface*. GI 1988. Edmonton, Canada, June 1988, pp. 26–33.

[MM13] Miles Macklin and Matthias Müller. "Position Based Fluids". In: *ACM Transactions on Graphics* 32.4 (July 2013), 104:1–104:12.

[MMC16] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. "XPBD: Position-based Simulation of Compliant Constrained Dynamics". In: *Proceedings of the International Conference on Motion in Games*. MIG '16. Burlingame, California: ACM, Oct. 2016, pp. 49–54.

[MMG06a] Bruce Merry, Patrick Marais, and James Gain. "Animation Space: A Truly Linear Framework for Character Animation". In: *ACM Transactions on Graphics* 25.4 (Oct. 2006), pp. 1400–1423.

[MMG06b] Bruce Merry, Patrick Marais, and James Gain. "Normal Transformations for Articulated Models". In: *ACM SIGGRAPH 2006 Sketches*. SIGGRAPH '06. Boston, Massachusetts: ACM, Aug. 2006.

[Muk16] Tomohiko Mukai. "Example-Based Skinning Animation". In: *Handbook of Human Motion*. Ed. by Bertram Müller, Sebastian I. Wolf, Gert-Peter Brueggemann, Zhigang

Deng, Andrew McIntosh, Freeman Miller, and William Scott Selbie. Springer International Publishing, 2016, pp. 1–21.

[Mül+07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. "Position based dynamics". In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118.

[Mur+14] Akihiko Murai, Kazunari Takeichi, Taira Miyatake, and Yoshihiki Nakamura. "Musculoskeletal modeling and physiological validation". In: *Proceedings of the IEEE International Workshop on Advanced Robotics and its Social Impacts*. ARSO 2014. Chicago, USA, Sept. 2014, pp. 108–113.

[Mur+16] Akihiko Murai, Q. Youn Hong, Katsu Yamane, and Jessica K. Hodgins. "Dynamic Skin Deformation Simulation Using Musculoskeletal Model and Soft Tissue Dynamics". In: *Pacific Graphics Short Papers*. Ed. by Eitan Grinspun, Bernd Bickel, and Yoshinori Dobashi. The Eurographics Association, 2016.

[Nea+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. "Physically Based Deformable Models in Computer Graphics". In: *Computer Graphics Forum* 25.4 (2006), pp. 809–836.

[Neu+13] Thomas Neumann, Kiran Varanasi, Nils Hasler, Markus Wacker, Marcus Magnor, and Christian Theobalt. "Capture and Statistical Modeling of Arm-Muscle Deformations". In: *Computer Graphics Forum* 32.2 (May 2013), pp. 285–294.

[Nie04] Gregory M. Nielson. "Dual Marching Cubes". In: *Proceedings of the IEEE Conference on Visualization*. VIS '04. Austin, USA: IEEE Computer Society, Oct. 2004, pp. 489–496.

[Nis+85] Hitoshi Nishimura, Makoto Hirai, Toshiyuki Kawai, Toru Kawata, Isao Shirakara, and Koichi Omura. "Object modeling by distribution functions and a Method of Image Generation". In: *The Transactions of the Institute of Electronics and Communication Engineers of Japan* J68-D.4 (1985). (in Japanese), pp. 718–725.

[NN94] Tomoyuki Nishita and Eihachiro Nakamae. "A Method for Displaying Metaballs by using Bézier Clipping". In: *Computer Graphics Forum* 13.3 (1994), pp. 271–280.

[NS13] Jesús R. Nieto and Antonio Susín. "Cage Based Deformations: A Survey". In: *Deformation Models: Tracking, Animation and Applications*. Ed. by Manuel González Hidalgo, Arnau Mir Torres, and Javier Varona Gómez. Dordrecht: Springer Netherlands, 2013, pp. 75–99.

[NT98] Luciana Porcher Nedel and Daniel Thalmann. "Real time muscle deformations using mass-spring systems". In: *Proceedings of Computer Graphics International*. CGI 98. Hannover, Germany, June 1998, pp. 156–165.

[Pan+13] Daniele Panozzo, Ilya Baran, Olga Diamanti, and Olga Sorkine-Hornung. "Weighted averages on surfaces". In: *ACM Transactions on Graphics* 32.4 (2013), 60:1–60:12.

[Pas+95] Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir Savchenko. "Function representation in geometric modeling: concepts, implementation and applications". In: *The Visual Computer* 11.8 (1995), pp. 429–446.

[PH06] Sang Il Park and Jessica K. Hodgins. "Capturing and Animating Skin Deformation in Human Motion". In: *ACM Transactions on Graphics* 25.3 (July 2006), pp. 881–889.

[PH08] Sang Il Park and Jessica K. Hodgins. "Data-driven Modeling of Skin and Muscle Deformation". In: *ACM Transactions on Graphics* 27.3 (Aug. 2008), 96:1–96:6.

[Pon+15] Gerard Pons-Moll, Javier Romero, Naureen Mahmood, and Michael J. Black. "Dyna: A Model of Dynamic Human Shape in Motion". In: *ACM Transactions on Graphics* 34.4 (July 2015), 120:1–120:14.

[PSW05] Dinesh K. Pai, Shinjiro Sueda, and Qi Wei. "Fast Physically Based Musculoskeletal Simulation". In: *ACM SIGGRAPH 2005 Sketches*. SIGGRAPH '05. Los Angeles, California: ACM, 2005.

[Req80] Aristides A. G. Requicha. "Representations of Rigid Solids: Theory, Methods, and Systems". In: *ACM Computing Surveys* 12 (1980), pp. 437–464.

[Ric73] Antonio Ricci. "A constructive geometry for computer graphics". In: *The Computer Journal* 16.2 (1973), pp. 157–160.

[RL13] Juan Ramos and Caroline Larboulette. "A Muscle Model for Enhanced Character Skinning". In: *Journal of WSCG* 21.2 (2013), pp. 107–116.

[RLN06] Taehyun Rhee, John P. Lewis, and Ulrich Neumann. "Real-Time Weighted Pose-Space Deformation on the GPU". In: *Computer Graphics Forum* 25.3 (2006), pp. 439–448.

[SA07] Olga Sorkine and Marc Alexa. "As-rigid-as-possible surface modeling". In: *Proceedings of the Symposium on Geometry processing*. Vol. 4. SGP 07. Barcelona, Spain, July 2007, pp. 109–116.

[Sab68] Malcolm A. Sabin. *The use of potential surfaces for numerical geometry*. Tech. rep. VTO/MS/153. Weybridge, United Kingdom: British Aerospace Corporation, 1968.

[Sac+15] Prashant Sachdeva, Shinjiro Sueda, Susanne Bradley, Mikhail Fain, and Dinesh K. Pai. "Biomechanical Simulation and Control of Hands and Tendinous Systems". In: *ACM Transactions on Graphics* 34.4 (July 2015), 42:1–42:10.

[Sch+97] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. "Anatomy-based Modeling of the Human Musculature". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 163–172.

[SDN84] Dennis C. Schneider, Terence M. Davidson, and Alan M. Nahum. "In vitro biaxial stress-strain response of human skin". In: *Archives of Otolaryngology* 110.5 (1984), pp. 329–333.

[SKP08] Shinjiro Sueda, Andrew Kaufman, and Dinesh K. Pai. "Musculotendon Simulation for Hand Animation". In: *ACM Transactions on Graphics* 27.3 (Aug. 2008), pp. 831–838.

[SOP95] Karansher Singh, Jun Ohya, and Richard E. Parent. "Human figure synthesis and animation for virtual space teleconferencing". In: *Proceedings of the Virtual Reality Annual International Symposium*. VRIS 95. Research Triangle Park, USA, Mar. 1995, pp. 118–126.

[SP86] Thomas W. Sederberg and Scott R. Parry. "Free-form Deformation of Solid Geometric Models". In: *SIGGRAPH Computer Graphics* 20.4 (Aug. 1986), pp. 151–160.

[SRC01] Peter-Pike J. Sloan, Charles F. Rose III, and Michael F. Cohen. "Shape by Example". In: *Proceedings of the Symposium on Interactive 3D Graphics*. I3D '01. Research Triangle Park, USA: ACM, 2001, pp. 135–143.

[SS11] Jos Stam and Ryan Schmidt. "On the Velocity of an Implicit Surface". In: *ACM Transactions on Graphics* 30.3 (May 2011), 21:1–21:7.

[SSF08] Tamar Shinar, Craig Schroeder, and Ronald Fedkiw. "Two-way Coupling of Rigid and Deformable Bodies". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '08. Dublin, Ireland: Eurographics Association, 2008, pp. 95–103.

[Ste64] Niels Stensen. *De musculis et glandulis observationum specimen: cum epistolis duabus anatomicis*. (Published under the name Nicolai Stenonis). Copenhagen: Matthiae Godicchenii, 1664.

[Sue+11] Shinjiro Sueda, Garrett L. Jones, David I. W. Levin, and Dinesh K. Pai. "Large-scale Dynamic Simulation of Highly Constrained Strands". In: *ACM Transactions on Graphics* 30.4 (July 2011), 39:1–39:10.

[SZK15] Shunsuke Saito, Zi-Ye Zhou, and Ladislav Kavan. "Computational Bodybuilding: Anatomically-based Modeling of Human Bodies". In: *ACM Transactions on Graphics* 34.4 (2015). Proceedings of SIGGRAPH 2015, 41:1–41:12.

[Ter+03] Joseph Teran, Silvia Blemker, Victor Ng-Thow-Hing, and Ronald Fedkiw. "Finite Volume Methods for the Simulation of Skeletal Muscle". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '03. San Diego, California: Eurographics Association, 2003, pp. 68–74.

[Ter+05] Joseph Teran, Eftichios Sifakis, Silvia Blemker, Victor Ng-Thow-Hing, Cynthia Lau, and Ronald Fedkiw. "Creating and simulating skeletal muscle from the visible human data set". In: *IEEE Transactions on Visualization and Computer Graphics* 11.3 (May 2005), pp. 317–328.

[TFS17] Fabio Turchet, Oleg Fryazinov, and Sarah C. Schvartzman. "Physically-based Muscles and Fibers Modeling from Superficial Patches". In: *Eurographics Short Papers*. 2017.

[TJ95] Frank Thomas and Ollie Johnston. *The illusion of life: Disney animation*. 3rd ed. New York: Hyperion, 1995.

[TSC96] Daniel Thalmann, Jianhua Shen, and Eric Chauvineau. "Fast realistic human body deformations for animation and VR applications". In: *Proceedings of Computer Graphics International*. CGI 96. Pohang. Korea, June 1996, pp. 166–174.

[Urb+01] Melanie G. Urbanchek, Elisa B. Picken, Loree K. Kalliainen, and William M. Kuzon Jr. "Specific Force Deficit in Skeletal Muscles of Old Rats Is Partially Explained by the Existence of Denervated Muscle Fibers". In: *The Journals of Gerontology: Series A* 56.5 (2001), B191–B197.

[Vai+13] Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. "Implicit Skinning: Real-time Skin Deformation with Contact Modeling". In: *ACM Transactions on Graphics* 32.4 (July 2013), 125:1–125:12.

[Vai+14] Rodolphe Vaillant, Gaël Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. "Robust Iso-surface Tracking for Interactive Character Skinning". In: *ACM Transactions on Graphics* 33.6 (Nov. 2014), 189:1–189:11.

[Web+07] Ofir Weber, Olga Sorkine, Yaron Lipman, and Craig Gotsman. "Context-Aware Skeletal Shape Deformation". In: *Computer Graphics Forum* 26.3 (2007).

[WH94] Andrew P. Witkin and Paul S. Heckbert. "Using Particles to Sample and Control Implicit Surfaces". In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. Orlando, USA: ACM, July 1994, pp. 269–277.

[Wil94] Jane Wilhelms. *Modeling Animals with Bones, Muscles, and Skin*. Tech. rep. University of California, Santa Cruz, 1994.

[WMW86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. "Data structure for soft objects". In: *The Visual Computer* 2.4 (Feb. 1986), pp. 227–234.

[WO97] Brian Wyvill and Kees van Overveld. "Warping as a modelling tool for CSG/implicit models". In: *Proceedings of the International Conference on Shape Modeling and Applications*. SMI 97. IEEE Computer Society. Aizu. Japan, Mar. 1997, pp. 205–213, 248.

[Woo00] Ryan Woodland. "Filling the Gaps – Advanced Animation Using Stitching and Skinning". In: *Game Programming Gems*. Ed. by Mark De Loura. Vol. 1. Game Programming Gems. Charles River Media, 2000. Chap. 4.15, pp. 476–483.

[WP02] Xiaohuan Corina Wang and Cary Phillips. "Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation". In: *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. SCA '02. San Antonio, Texas: ACM, 2002, pp. 129–138.

[WPP07] Robert Y. Wang, Kari Pulli, and Jovan Popović. "Real-time Enveloping with Rotational Regression". In: *ACM Transactions on Graphics* 26.3 (July 2007).

[WV97] Jane Wilhelms and Allen Van Gelder. "Anatomically-Based Modeling". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 173–180.

[Wyv15] Brian Wyvill. "Implicit Modeling". In: *Fundamentals of Computer Graphics*. Ed. by Steve Marschner and Peter Shirley. 4th ed. CRC Press, 2015. Chap. 22, pp. 585–612.

[XB16] Hongyi Xu and Jernej Barbič. "Pose-space Subspace Dynamics". In: *ACM Transactions on Graphics* 35.4 (July 2016), 35:1–35:14.

[YT13] Jihun Yu and Greg Turk. "Reconstructing Surfaces of Particle-based Fluids Using Anisotropic Kernels". In: *ACM Transactions on Graphics* 32.1 (Feb. 2013), 5:1–5:12.

[Zaj89] Felix E. Zajac. "Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control." In: *Critical reviews in biomedical engineering* 17 4 (1989), pp. 359–411.

[ZHK15] Lifeng Zhu, Xiaoyan Hu, and Ladislav Kavan. "Adaptable Anatomical Models for Realistic Bone Motion Reconstruction". In: *Computer Graphics Forum* 34.2 (May 2015). Proceedings of Eurographics, pp. 459–471.

Contents

List of Figures

♠

Colophon

This document is the
defense version of the thesis.

It was typeset with LuaLaTeX on June 20, 2018
using the `scrbook` class from the KOMA-Script package.

Pagella
is the main font used for the text and maths. A font in the style of Palatino, a humanist serif typeface, designed by Herman Zapf in 1949. This typeface comes from the TeX Gyre project maintained by the Polish TeX user group GUST. Additional glyphs from XITS-Math from STIX Fonts were used for the cursive math letters.

Fira Sans
designed by Erik Spiekermann and Carois Type Design for the Mozilla Foundation in 2013 is used for parts, chapters and section titles.
Fonts are released under open licences (GUST and SIL).

♠

All the material
used in the preparation
of this thesis was produced with free and open source software.

Résumé : En animation de personnages 3D, la déformation de surface, ou *skinning*, est une étape cruciale. Son rôle est de déformer la représentation surfacique d'un personnage pour permettre son rendu dans une succession de poses spécifiées par un animateur. La plausibilité et la qualité visuelle du résultat dépendent directement de la méthode de skinning choisie. Sa rapidité d'exécution et sa simplicité d'utilisation sont également à prendre en compte pour rendre possible son usage interactif lors des sessions de production des artistes 3D.

Les différentes méthodes de *skinning* actuelles se divisent en trois catégories. Les méthodes géométriques sont rapides et simples d'utilisation, mais leur résultats manquent de plausibilité. Les approches s'appuyant sur des exemples produisent des résultats réalistes, elles nécessitent en revanche une base de données d'exemples volumineuse, et le contrôle de leur résultat est fastidieux. Enfin, les algorithmes de simulation physique sont capables de modéliser les phénomènes dynamiques les plus complexes au prix d'un temps de calcul souvent prohibitif pour une utilisation interactive.

Les travaux décrits dans cette thèse s'appuient sur *Implicit Skinning*, une méthode géométrique corrective utilisant une représentation implicite des surfaces, qui permet de résoudre de nombreux problèmes rencontrés avec les méthodes géométriques classiques, tout en gardant des performances permettant son usage interactif. La contribution principale de ces travaux est un modèle d'animation qui prend en compte les effets des muscles des personnages et de leur interactions avec d'autres éléments anatomiques, tout en bénéficiant des avantages apportés par *Implicit Skinning*. Les muscles sont représentés par une surface d'extrusion le long d'axes centraux. Les axes des muscles sont contrôlés par une méthode de simulation physique simplifiée. Cette représentation permet de modéliser les collisions des muscles entre eux et avec les os, d'introduire des effets dynamiques tels que rebonds et secousses, tout en garantissant la conservation du volume, afin de représenter le comportement réel des muscles.

Ce modèle produit des déformations plus plausibles et dynamiques que les méthodes géométriques de l'état de l'art, tout en conservant des performances suffisantes pour permettre son usage dans une session d'édition interactive. Elle offre de plus aux infographistes un contrôle intuitif sur la forme des muscles pour que les déformations obtenues se conforment à leur vision artistique.

Abstract: Surface deformation, or *skinning* is a crucial step in 3D character animation. Its role is to deform the surface representation of a character to be rendered in the succession of poses specified by an animator. The quality and plausiblity of the displayed results directly depends on the properties of the skinning method. However, speed and simplicity are also important criteria to enable their use in interactive editing sessions.

Current skinning methods can be divided in three categories. Geometric methods are fast and simple to use, but their results lack plausibility. Example-based approaches produce realistic results, yet they require a large database of examples while remaining tedious to edit. Finally, physical simulations can model the most complex dynamical phenomena, but at a very high computational cost, making their interactive use impractical.

The work presented in this thesis are based on, *Implicit Skinning* a corrective geometric approach using implicit surfaces to solve many issues of standard geometric skinning methods, while remaining fast enough for interactive use. The main contribution of this work is an animation model that adds anatomical plausibility to a character by representing muscle deformations and their interactions with other anatomical features, while benefiting from the advantages of Implicit Skinning. Muscles are represented by an extrusion surface along a central axis. These axes are driven by a simplified physics simulation method, introducing dynamic effects, such as jiggling. The muscle model guarantees volume conservation, a property of real-life muscles. This model adds plausibility and dynamics lacking in state-of-the-art geometric methods at a moderate computational cost, which enables its interactive use. In addition, it offers intuitive shape control to animators, enabling them to match the results with their artistic vision.