Interactive Modelling of Volumetric Musculoskeletal Anatomy

RINAT ABDRASHITOV, University of Toronto, Canada
SEUNGBAE BANG, University of Toronto, Canada
DAVID LEVIN, University of Toronto, Canada
KARAN SINGH, University of Toronto, Canada
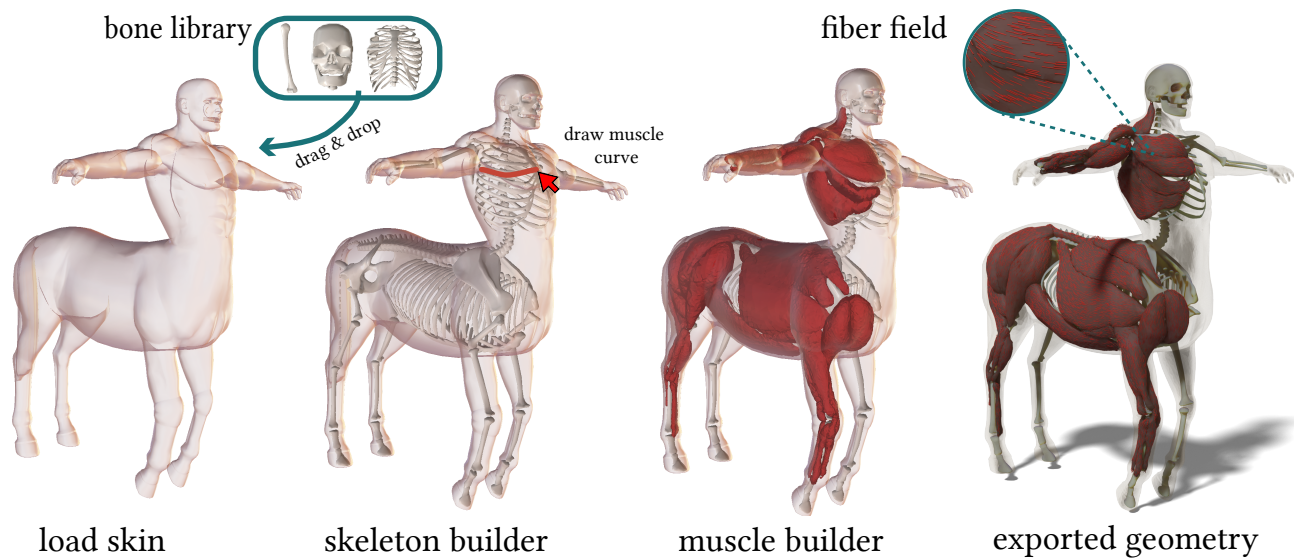ALEC JACOBSON, University of Toronto, Canada

Fig. 1. Given a skin surface mesh, a library of bone parts is used to quickly create a skeleton in our skeleton builder tool. The user then draws curves to generate the muscle shapes which are visualized using a volume rendering. Once all muscles are created, we can export the geometry of the muscles, automatically compute fiber fields and use the result in downstream applications. Centaur model is part of the TOSCA dataset [?]. Horse bone models were obtained from https://3dassets.store/. Used under permission.

We present a new approach for modelling musculoskeletal anatomy. Unlike previous methods, we do not model individual muscle shapes as geometric primitives (polygonal meshes, NURBS etc.). Instead, we adopt a volumetric segmentation approach where every point in our volume is assigned to a muscle, fat, or bone tissue. We provide an interactive modelling tool where the user controls the segmentation via muscle curves and we visualize the muscle shapes using volumetric rendering. Muscle curves enable intuitive yet powerful control over the muscle shapes. This representation allows us to automatically handle intersections between different tissues (muscle-muscle, muscle-bone, and muscle-skin) during the modelling and automates computation of muscle fiber fields. We further introduce a novel algorithm for converting the volumetric muscle representation into tetrahedral or surface geometry for use in downstream tasks. Additionally, we introduce an interactive skeleton authoring tool that allows the users to create skeletal anatomy starting from only a skin mesh using a library of bone parts.

CCS Concepts: • **Computing methodologies → Mesh geometry models**; **Volumetric models**; **Graphics systems and interfaces**.

Additional Key Words and Phrases: anatomy modelling, 3D interface, diffusion curves

Authors' addresses: Rinat Abdrashitov, University of Toronto, Toronto, Canada, rinat@dgp.toronto.edu; Seungbae Bang, University of Toronto, Toronto, Canada, seungbae@cs.toronto.edu; David Levin, University of Toronto, Canada, diwlevin@cs.toronto.edu; Karan Singh, University of Toronto, Toronto, Canada, karan@dgp.toronto.edu; Alec Jacobson, University of Toronto, Toronto, Canada, jacobson@cs.toronto.edu.

1 INTRODUCTION

Digital characters are a driving force in the entertainment industry allowing artists to tell stories limited only by their imagination. A lot of effort goes into reaching a point where digital characters are indistinguishable from the real ones. Characters are often modeled by only considering their skin [Jacobson et al. 2014], disregarding

underlying volumetric muscle, fat, and bone structure. Animating physically realistic effects like muscle bulging, skin sliding, wrinkles, and volume preservation, without an explicit musculoskeletal structure is challenging and, and requires skilled and tedious manual effort, to achieve high quality results. Therefore, a truly accurate portrayal of digital characters requires the creation of biologically representative musculoskeletal anatomy.

Different solutions that allow artists to automate tedious manual tasks like the creation of the skin [Yoshiyasu et al. 2014], hair [Saito et al. 2018], rigs [Xu et al. 2020] and others have been explored over the years. However, user-friendly solutions to the problem of creating a musculoskeletal structure that is suitable for character modelling and animation are relatively unexplored. The current solutions either require artists to model every muscle using sculpting software, go through tedious parameter tweaking of geometric primitives, or create a detailed template for retargeting to new geometries. These methods make it hard to produce complex intersection-free muscle shapes that conform to the skin surface. Additionally, defining muscle fiber directions requires users to manually specify attachment points for every muscle. Making incremental changes using these approaches is tedious and hinders the fast exploration of character design.

We propose an interactive modelling tool, that adopts the *outside−in* [Pratscher et al. 2005] approach and enables the creation of a musculoskeletal system starting from a skin mesh. The user starts by arranging the skeleton from pre-existing templates of bones. Then the user simply sketches curves inside a volume constrained by the skin and our system automatically infers the muscle shapes. Inspired by Orzan et al. [2008], we utilize a diffusion process to segment the volume into muscle and fat tissues based on the user-created curve network. The resulting muscles are intersection free and conform to the skin geometry. We show how to utilize volume rendering to visualize the muscle shapes and hence avoid the need for explicit meshing every time the user edits a curve. We further propose an algorithm for extracting the manifold muscle meshes from our volumetric segmentation for the use in downstream tasks. To the best of our knowledge, our system is the first user-friendly interactive modelling tool, capable of creating intersection free geometry for the musculoskeletal system.

2 BACKGROUND AND RELATED WORK

We provide a brief background on muscle anatomy followed by review of related work categorized by muscle representations in physically-based animation; skinning methods that geometrically attempt to "emulate" the physics of muscle deformation; anatomic templates to aid character modelling, setup and transfer; and interactive interfaces for volumetric and character modelling.

2.1 Musculoskeletal Anatomy:

Muscle is a soft tissue (of type skeletal, cardiac or smooth), whose function is to produce force and motion. A large body of research in Computer Graphics, biomechanics and robotics is focused on studying the physiological properties and and function of skeletal muscles [Ng-Thow-Hing and Fiume 1997; Scheepers et al. 1997]. For the rest of the paper, we will refer to "skeletal muscles" simply
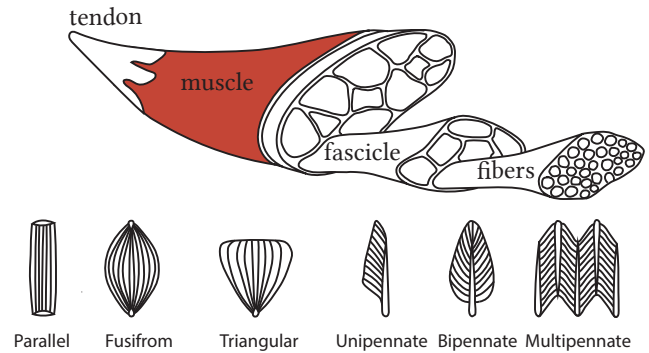

Fig. 2. Muscle structure. Image courtesy of [Lee et al. 2010]

as "muscles". Internally, the muscle is composed of numerous muscle *fiber* bundles, called *fascicles*. Large muscles, such as the biceps brachii or the sartorius have *fascicles* arranged parallel to one another along the length of the muscle. Other muscles exhibit fascicles with a *pennation* angle, between their tendinous attachments and the longitudinal axis of the muscle (Fig 2 bottom). Skeletal muscle is anchored by tendons to bone. Tendons transmit forces produced by the attached muscle to the bone, enabling locomotion and maintaining posture (Fig. 2 top). We refer the reader to the survey by Lee et al. [2010] which provides a thorough overview of modelling and simulation of skeletal muscles.

2.2 Surface-based muscle primitives:

Musculoskeletal primitives for geometric character skinning is at least three decades old [Chadwick et al. 1989]. Early research has explored the formulation of muscles as collections of ellipsoids [Pratscher et al. 2005; Scheepers et al. 1997; Singh et al. 1995], generalized cylinders [Simmons et al. 2002; Wilhelms and Van Gelder 1997], polygon meshes [Albrecht et al. 2003], extruded parametric curves [Tsang et al. 2005], NURBS [Autodesk 2021], and implicit models [Roussellet et al. 2018]. These surface-based muscle primitives typically serve as proxy geometry to bind and geometrically deform a geometric skin. While these primitives can be imbued with simplified muscle dynamics, they are ill-suited to general purpose anatomic simulation [Ziva Dynamics 2021]. Our representation, based on muscle curves and anisotropically induced muscle volumes, provides the high-level geometric control of muscle shape and skin deformation of these surface-based primitives, but can also produce various muscle shapes (parallel, convergent, pennate) and automate the computation of fiber fields. Being an inherently volume-based representation, it is also well-suited to handle general muscle-muscle, muscle-bone, muscle-skin intersections and muscle fiber bundle computations.

2.3 Volume-based muscle primitives:

Physically-based simulation of the skin layered over volumetric muscle primitives [Li et al. 2013] is a desirable solution to producing the subtle details of skin motion [Weta Digital 2021; Ziva Dynamics 2021]. The initial musuloskeletal setup of a character as

comprised of skin, fat, muscle and bone, in a simulation pipeline is tedious and requires multiple iterations of laboriously rebuilding hand-crafted bone and muscle geometry [Deepak Rajan 2021], to elicit the desired simulation behavior from the musculoskeletal anatomy. These sculpted geometries then need to be processed to resolve intersections, and define fiber fields to support anisotropic muscle contraction. While MRI/CT scan data can aid the reconstruction of accurate live anatomy [Jacobs et al. 2016; Teran et al. 2005], such data must be artist-imagined for fictional characters. Muscles have also been built by physically simulating inflatable 3D patches defined by a user on a character's skin [Turchet et al. 2017], or as parametric solid volumes [Ng-Thow-Hing and Fiume 1997], but these muscles tend to leave undesirable gaps between muscles, bones and other internal structures. Angles et al. [2019] models a muscle as a bundle of position-based rods augmented with isotropic scale to enable simulation of volumetric effects. However, their rod-based representation requires users to either manually create bundles or acquire pre-existing tetrahedral geometry of muscles which is then automatically converted to their representation. Yu et al. [2020] introduces an efficient algorithm for (self)-repulsion of space curves that can be used to design biologically-inspired curve networks such as muscle fibers. However, their optimization-based approach is not suitable for interactive modelling of a large number of muscles.

2.4 Anatomic Templates:

The first semi-automatic method for creating anatomical structures, such as bones, muscles, viscera, and fat tissues was proposed by Ali-Hamadi et al. [2013]. Their method can be seen as a partial registration process, where skin surfaces are first registered based on the data, and the interior tissues are estimated using interpolation and anatomical rules. Saito et al. [2015] create a wide range of human body shapes from a single input 3D anatomy template by simulating biological processes responsible for human body growth. Kadleček et al. [2016] use a set of 3D scans of an actor in various poses to compute subject-specific and pose-dependent parameters of an anatomical template model, to explain the captured 3D scans as closely as possible. Our method is complementary to these approaches and can be used to produce the initial template.

2.5 Interactive volumetric and character modelling:

Takayama et al. [2010] proposed a novel diffusion surface (DS) representation to model the smooth color variation seen in fruit and vegetables. User input to their approach, and others [Owada et al. 2008; Pietroni et al. 2007] is based on cross-sections, which are ill-suited to modelling complex muscle geometry and connectivity. Solid texture synthesis [Pietroni et al. 2010] is focused on modelling homogeneous material like wood or marble and Cutler et al. [2002] uses scripting to define internal volumetric structure of mesh(es). Yuan et al. [2012] do facilitate solid modelling of heterogeneous objects with multiple internal regions using multiphase implicit functions. However, these approaches are not artist-centric or require segmented and labeled 3D biomedical images as input. Wang et al. [2011] represent complex internal 3D structure using multiscale vector volumes. The object is decomposed into components
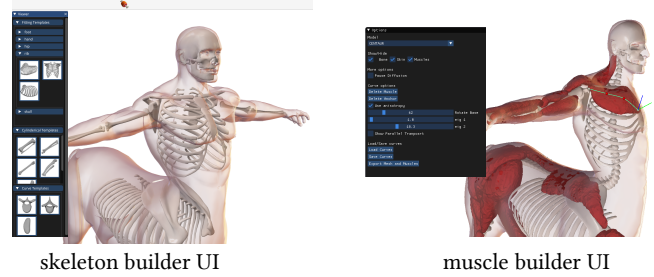

skeleton builder UI   muscle builder UI

Fig. 3. The skeleton and muscle builder UIs.

modelled as SDF trees. However, the user needs to actually create the "building-blocks" of an object, such as SDF instances and region definitions, and then assemble them together into linked SDF trees.

Several sketch-based interfaces for character modelling [De Paoli and Singh 2015; Nealen et al. 2007; Schmid et al. 2011; Takayama et al. 2013] use a 3D geometric skin as a canvas on and around which to project 2D sketch strokes. Our work is similar in spirit to [De Paoli and Singh 2015; Schmid et al. 2011] in that we are focused on drawing curves, around the skin, and specifically within a volumetric domain constrained by the surface of the skin.

The actual shape of the muscles is inferred from the scalar values defined on the curves via a volumetric rendering approach. We do not explicitly generate the geometry (triangle or tetrahedral meshes) of the muscles until the user completes the modelling session. Unlike previous methods, our approach allows the user to rapidly explore and experiment with both the topological connectivity and shape of musculoskeletal structures, with a guarantee of precisely constrained and intersection-free structures that conform outside-in to the given skin surface.

3 OUR SYSTEM

The user starts by providing the skin mesh of a model. The bone meshes can also either be provided by the user or built using our interface (Fig.3 left). We generate the tetrahedral mesh $\mathbf{T} \in \mathbb{R}^{|\mathbf{T}|\times 4}$, $\mathbf{V} \in \mathbb{R}^{|\mathbf{V}|\times 3}$ from combined skin and bone geometries to make sure our tetrahedralization conforms to the bone geometry. We remove all tetrahedra belonging to the bone geometry. This simultaneously induces desirable *natural boundary conditions* (see, e.g., [Stein et al. 2018]) and increases computational performance. A tablet or a mouse can be used to draw an open 2D stroke that starts and ends over the bone surface. The muscle surface corresponding to the drawn curve is presented to the user. The user can continue to draw more strokes or edit the existing ones to create a full muscular system (Fig.3 right). The results can then be exported as surface or tetrahedral meshes and further edited in other software packages. In the following sections, we describe each part of our muscle modelling system and discrete implementation in detail.

3.1 Skeleton Authoring

If pre-existing skeletal geometry is not available, we provide a tool for creation of the skeletal system from the pre-existing library of bones. We define a three categories for the types for bones: cylinder

bone, curve bone, shape bone. The user can select a pre-defined templates for each of the categories. Then each of template bones are properly placed inside the body mesh with the algorithm described below.

3.1.1 Cylinder bone. A cylinder bone is a type of bone that can be represented as a line segment (e.g. arm or leg bone). When a user clicks on a point on the surface of the skin mesh, we cast a ray through the mesh and record the first two hits, which correspond to the ray entering and exiting the mesh, respectively. We take the midpoint of these two intersections as one endpoint of the cylinder bone, and the other endpoint is determined interactively using the same ray-casting procedure while the mouse button is held down. A cylinder bone template is pre-rigged with two point handles and it is deformed accordingly as its two point handles are attached with the endpoints of the line the user draw.

3.1.2 Curve bone. Curve bone is a type of bone that can be represented as a curve. For example, a spine can be described as a sequence of vertebrae bones placed along a curve. When the user draws a curve, it is projected onto a user-defined plane of symmetry. Then for easy editing of the curve, we fit a Catmull-Rom spline to a given points on plane. Finally, a user selected template is distributed along the spline.

3.1.3 Shape bone. A shape bone is a type that cannot be represented as a line segment nor a curve. Essentially, it describes all the bones with complex shapes. We place the shape bone as local fitting on the user-specified region. We determine the center of the template with a mouse click, using the same raycasting procedure used to compute the endpoints of a cylinder bone. Then from that initial shape, the template is iteratively fitted to its local region of the skin mesh.

3.1.4 Local fitting. We pre-rigged the template with a cage deformer. Using data that already has skin and corresponding bone mesh, we define a cage by cutting a local region of the skin mesh with high decimation and with manual editing. We fit the cage using both step of rigid ICP (Iterative Closest Point) and nonrigid ICP, and the templates are deformed using this registered cage. We first perform rigid ICP to find an optimal transformation of its closest corresponding target points. After it has converged within the threshold, we then perform nonrigid ICP by deforming the cage with an additional squared Laplacian smoothness term to prevent abrupt deformation. Corresponding target points are determined by finding the closest points to cage vertices on the skin mesh. We discard the correspondence point whose normal directions are almost opposite to their closest projected points

3.2 Muscle Curve Authoring

The user draws a curve for each muscle. Skeletal muscles require an origin and insertion points where the muscle tendons are being attached to the bone. In our interface we expect the user to always begin and end the stroke over the bone surface and provide the necessary visual feedback to achieve that. The first and last points of the curve are automatically projected via ray-casting onto the surface of the bone to find their corresponding 3d coordinates.

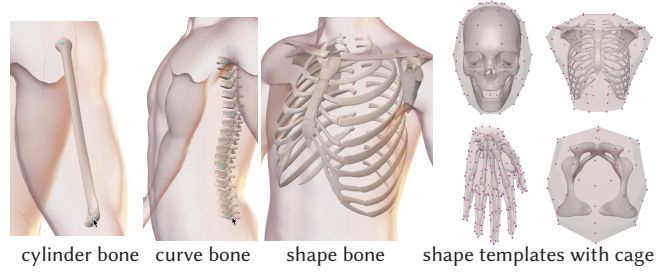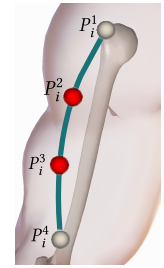cylinder bone   curve bone   shape bone   shape templates with cage

Fig. 4. three types of bone in our skeleton authoring interface, and pre-rigged on cage deformer of shape templates.

The depth value for all the points drawn in between is ambiguous, so we simply linearly interpolate the depths of the first and last points. Because we want the curves to be easily editable we fit a Catmull–Rom spline S_i in a least squares manner with 4 control points $P_i^1, ..., P_i^4$ by default: one for each end point and two along the curve (see inset). We denote the resulting muscle curve network as a set of splines $\{S_1, S_2, ..., S_m\}$. Each control point P_i^p is augmented with an additional attribute representing a *tissue* value D_i^p. Users can adjust the tissue value to shape the muscle: a larger value results in a "thicker" shape around the control point.

3.3 Muscle and Fat Functions

Let $\Omega \in \mathbb{R}^3$ denote the volumetric domain defined by the tetrahedral mesh $\mathbf{V}, \mathbf{T}$. Our goal is to find a scalar muscle function $f_i : \Omega \to \mathbb{R}$ for each muscle i that describes the likelihood of any point $p \in \Omega$ to belong to a muscle i. Similarly we define a fat function f_s that describes the likelihood of any point to belong to a fat layer. We propose to define f_i and f_s as minimizers of the Dirichlet energy subject to constraints:

$$\underset{f_s, f_i, i=1,...,m}{\text{argmin}} \sum_{i=1}^{m} \int_{\Omega} \nabla f_i^T \mathbf{A}_i \nabla f_i + \int_{\Omega} ||\nabla f_s||^2 dV \tag{1}$$

$$\text{subject to} \quad f_i|_{\delta\Omega_i = D_i} \tag{2}$$

$$f_i|_{\delta\Omega_j = 0} \quad j \in \{1, ..., m\}, j \neq i \tag{3}$$

$$f_i|_{\delta\Omega = 0} \tag{4}$$

$$f_s|_{\delta\Omega = d_{\text{fat}}} \tag{5}$$

$$f_s|_{\delta\Omega_j = 0} \quad j \in \{1, ..., m\} \tag{6}$$

where $\mathbf{A}_i$ is an optional user-defined diffusion tensor field which biases the directions in which material flows at a point in space. Intuitively, we "diffuse" each muscle curve such that the tissue values at the curve points ($\delta\Omega_i$) are set by the user (via interpolation of tissue values D_m^p at the control points) and the tissue values at all other muscle curves ($\delta\Omega_j$) and skin ($\delta\Omega$) are set to zero. We additionally diffuse from the skin to represent the fat layer (fat function f_s, where "s" stands for skin) by setting its tissue value to a user-defined d_{fat} and constraining the values at all muscle curves
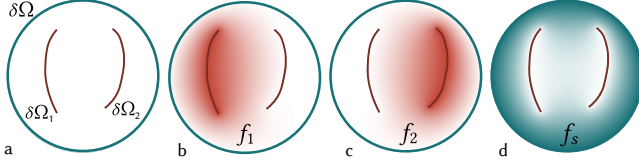
Fig. 5. 2D example of two muscle curves in red and skin layer in aquamarine (a). User defined tissue values at each curve ($\delta\Omega_1$, $\delta\Omega_2$) are diffused (b,c) to compute corresponding muscle functions (f_1, f_2). Additionally fat tissue values defined at skin vertices are diffused to compute the fat function f_s (d).

to be zero (Fig. 5). The fat function always diffuses isotropically and hence is written as a separate term. The vertices that belong to the bone surface do not "diffuse" bone tissue material but instead participate in the optimization as natural (zero normal derivative) boundary conditions.

3.4 Discretization

The muscle splines usually do not coincide with the vertices of the tetrahedral mesh and to discretize the splines we need to identify a set of tetrahedrons that contain each muscle curve S_i. For each tetrahedron t and each muscle curve i passing through it, we identify a point $c_i^t \in \mathbb{R}^3$ on the curve that is contained inside the tetrahedron (see inset).

In practice, we choose the point to be the midpoint of the curve segment bounded by the tetrahedron. We call this point a *collocation* point. All collocation points can be found efficiently by finding the tetrahedron containing the first curve point and then finding all the

other tetrahedrons by tracing the curve, given that the adjacency is computed beforehand. This amounts to at most three ray triangles intersections per tetrahedron and therefore very fast.

For each curve i, we stack all *collocation* points in a matrix $C_i \in \mathbb{R}^{|C_i| \times 3}$ and compute a tissue d_i^t value for each *collocation* point by interpolating values at control points P_i^p. One of the primary constraints to be satisfied (Eq. 2, 3) are the curve constraints, i.e.,the tissue value at the vertices V must be determined to agree with the values at the collocation points. We use barycentric coordinates to interpolate the tissue values for each tetrahedron, such that the value d_i^t of the collocation point c_i^t is expressed as:

$$d_i^t = \sum_{j=1}^{4} b_j * d_j \tag{7}$$

where d_j denotes the tissue value of a vertex j of a tetrahedron t (for $j = 1, 2, 3, 4$) to which the collocation point c_i^t belongs to, and b_j are the barycentric coordinates of c_i^t with respect to the vertex j. Stacking the barycentric equation (7) for a set of desired values of the collocation points into a matrix constitutes a linear equality

constraint equation:

$$B_i f_i = \begin{bmatrix} B_1 \\ \vdots \\ B_m \end{bmatrix} f_i = \begin{bmatrix} d_1 \\ \vdots \\ d_m \end{bmatrix} = d_i \in \mathbb{R}^{|C|} \tag{8}$$

where for every curve $i = \{1, ..., m\}$ we have $B_i \in \mathbb{R}^{|C| \times |V|}$ which is a sparse matrix of stacked barycentric coordinates of collocation points for all curves with non-zero entry $B_i(k, j)$ being a barycentric coordinate of the *collocation* point k with respect to vertex j, $d_i \in \mathbb{R}^{|C|}$ is a vector of stacked tissue values of collocation points s.t. $d_i(j)$ is nonzero only if collocation point j belongs to curve i and $f_i \in R^{|V|}$ are values of muscle function f_i at each vertex of the tetrahedral mesh. The fat tissue constraint (Eq. 6) can be similarly discretized as

$$B_s f_s = \begin{bmatrix} B_1 \\ \vdots \\ B_m \end{bmatrix} f_s = 0 \in \mathbb{R}^{|C|} \tag{9}$$

The Dirchlet Energy in Eq.1 is discretized as

$$\min \sum_{i=1}^{m} (f_i^T \tilde{L}_i f_i + \alpha ||B_i f_i - d_i||^2) + f_s^T L_c f_s + \alpha ||B_s f_s||^2 \tag{10}$$

$$\text{subject to} \quad f_i|_{skin} = 0 \tag{11}$$

$$f_s|_{skin} = d_{fat} \tag{12}$$

$$\tilde{L}_i = G^T \tilde{M} A_i G \tag{13}$$

where G is the gradient matrix (see [Botsch et al. 2010] for derivation) and $\tilde{M}$ is a mass matrix representing an inner-product accounting for the volume associated with each tetrahedron, $G^T \tilde{M} A G$ is the anisotropic cotangent Laplacian ([Andreux et al. 2014]), L_c is the standard (isotropic) cotangent Laplacian, α parameter that defines the tradeoff between smoothness of the resulting scalar field and respecting the tissue values at the collocation points (set to 5 in our experiments). Because the collocation points are not necessarily located at mesh vertices and many may appear in the same element, using soft constraints avoids overshooting. Vertex values f_i of each muscle function can be computed separately and hence computation of Eq.10 is easily parallelizable. Compared to regular-grid-based methods, the boundary-conforming tetrahedral mesh makes it easy to set precise boundary conditions.

3.5 Segmentation

Each point $p \in \Omega$ in our volume will either belong to a muscle, bone, or fat. The fat is visually represented as an "empty" space between muscles, skin, and bones. The points that belong to the bone tissue are simply all the points that are contained inside tetrahedrons comprising the bone geometry. So we only need to differentiate between muscles and fat. We treat the muscle and fat functions as probabilities and assign the point p to the tissue with the highest probability (Fig. 6):

$$tissue(p) = \underset{i}{\text{argmax}}(\{f_1(p), ... f_m(p), f_s(p)\}) \tag{14}$$
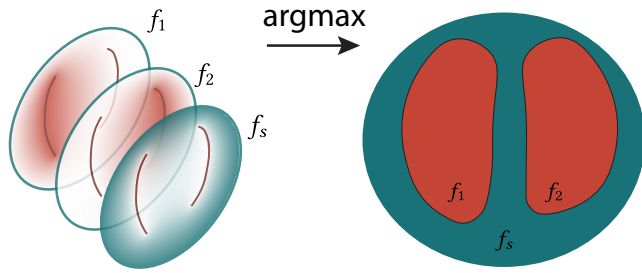
Fig. 6. Each point in our domain has a vector of tissue values that represent the likelihood of this point to belong to one of the tissues (left). We assign the point to the tissue with the highest probability (right).

In the discrete case we find the tetrahedron t that contains the point p and use barycentric interpolation to determine tissue values for muscles and fat at p given tissue values $(f_1^t .. f_m^t, f_s^t)$ at the vertices of the tetrahedron t. We skip the normalization step of our probabilities because the muscle visualization (Sec 3.6) and muscle extraction (Sec 4) are invariant to normalization.

3.6 Visualization

In the end, we only want to visualize all points belonging to the muscles, while points belonging to fat should be invisible. A common solution to this problem in scientific visualization and computer graphics is volume rendering. We take inspiration from the literature on volume rendering on unstructured grids ([Silva et al. 2005; Weiler et al. 2003]) which deals with rendering isosurfaces of a scalar function defined on vertices of the tetrahedral mesh. To achieve interactive rates we perform ray casting on the graphics hardware via a ray propagation approach and perform all computation inside a fragment shader.

To render the model we split each tetrahedron into 4 triangles and submit them for rendering on the GPU. For each vertex of the triangle, we assign a vertex attribute with the value of the tetrahedron index it belongs to and set it to not be interpolated when moving from vertex to fragment shader. That way each fragment can be traced back to the tetrahedron it belongs to. We store all the information (vertex positions, normals, muscle and fat functions) about each tetrahedron on the GPU and access it inside the shader.

Inside the fragment shader, we start with computing the entry point into the tetrahedron that the current fragment belongs to by simply converting the fragment from screen space into camera space. We determine the corresponding exit point by computing three intersection points of the ray with the planes containing faces of the entered tetrahedron and choosing the intersection point that is closest to the eye point but not on a face that is visible from the eyepoint. With the eye point e, and the normalized direction r of the viewing ray, the intersection points with the faces of the tetrahedron are $e + \lambda_i r$ with $0 \leq i < 4$:

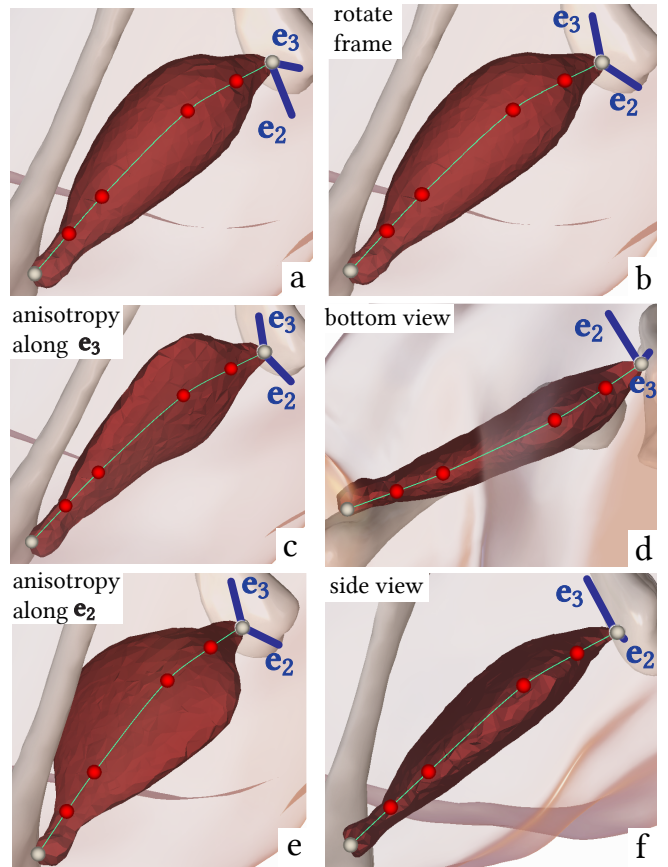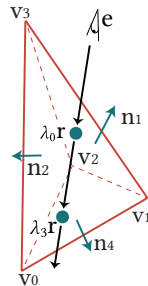$$\lambda_i = \frac{(v_{3-i} - e) \cdot n_i}{r \cdot n_i} \qquad (15)$$

Fig. 7. The user has additional control over the muscle shape by changing the rate of diffusion along the certain directions.

where $i \in \{0, 1, 2, 3\}$ denote the face index, v_i is the vertex opposite to the i-th face, n_i is the normal vector of the face i pointing outside of the tetrahedron (see inset). A face is visible if the denominator in the previous equation is negative; thus, this test comes almost for free. If λ_i is set to an appropriately large number for all visible faces, $\min\{\lambda_i | 0 \leq i < 4\}$ identifies the exit point. Once the minimum λ_i and its face i are identified, the intersection point x may be computed as $x = e + \lambda_i r$.

The muscle surface is only potentially visible if the entry point belongs to fat in which case we ray march through the single tetrahedron from the entry point to the exit point until we detect that the tissue changed from fat to muscle. At which point we stop and compute the normal to shade the surface of the muscle. If the whole tetrahedron belongs to fat we call the GLSL discard command and the GPU will call the shader again until we find a tetrahedron that contains the surface of a muscle (if one exists). Figures 1, 7 show an example of a volume-rendered muscles.

3.7 Anisotropy

To provide an additional control over the muscle shape we allow users to change the rate of diffusion along the certain directions. This is achieved by introducing a tensor field that biases the directions in

which tissues diffuse at a point in space. Tensor field is represented by a tensor matrix $\mathbf{A}_i \in \mathbb{R}^{|G| \times |G|}$ in Eq. 13.

We provide an easy way for the user to construct the tensor fields for the individual muscles. The first point of each muscle curve is assigned a 3D frame $\mathbf{Q} = [\mathbf{e}_1 \mathbf{e}_2 \mathbf{e}_3] \in \mathbb{R}^{3 \times 3}$ representing eigenvectors of a tensor. The first vector in the frame $\mathbf{e}_1$ is always aligned with the curve tangent while the other two eigenvectors lie in its null space. The user has control over the rotation of $\mathbf{e}_2, \mathbf{e}_3$ along the axis defined by $\mathbf{e}_1$ (Fig. 7ab). Additionally, the user can control the magnitude of eigenvalues λ_2, λ_3 to bias diffusion rate along $\mathbf{e}_2, \mathbf{e}_3$ directions (Fig. 7cdef). We use the method of [Hanson and Ma 1995] to compute the frame $\mathbf{Q}_i^t$ at each collocation point c_i^t, and assign frames to each vertex of the tetrahedrons containing collocation points (using the frame of the closest collocation point).

Each frame $\mathbf{Q}$ can be considered as a set of nine scalars and the problem of propagating values from the fixed vertices of the tetrahedrons containing collocation points to the remaining vertices is thus same as computing nine scalar fields over the mesh. To find the unknown values at the remaining vertices, we use the Laplacian smoothing framework, i.e. for each scalar field, we create a harmonic vertex-based scalar field on $\mathbf{T}$ given the boundary conditions (from the values at the fixed vertices) [Palacios et al. 2016]. Once all 9 scalar fields have been obtained, we compute the per-vertex tensor $\mathbf{Q}^T \Lambda \mathbf{Q}$ where Λ contains eigenvalues $\lambda_1 \lambda_2 \lambda_3$ along the diagonal and assemble the tensor field matrix $\mathbf{A}_i$ for the muscle i.

4 MUSCLE EXTRACTION

Given the complete muscle curve network and the corresponding muscle functions defined over the volumetric domain, we need to extract the geometry of the muscles suitable for downstream applications.

Isosurface extraction methods [Chentanez et al. 2009; Labelle and Shewchuk 2007; Lorensen and Cline 1987] do not trivially solve our problem, because we are not simply extracting an isosurface of a scalar function. Instead, we want compute a *pointwise* maximum of multiple scalar functions inside each tetrahedron and extract a boundary separating each function while ensuring the final output is a manifold tetrahedral mesh. In other words, inside each tetrahedron, each tissue function can be thought of as a point-wise vote for ownership. We want to split the tetrahedron along boundaries that delineate changes in the maximum vote and assign each sub-tet to the tissue with maximum value. We cast the problem of extracting the muscle surface geometry as a solution to the computation of the upper envelope of scalar functions representing each tissue over each tetrahedron. First, lets look at a simple example of the upper envelope problem in Fig. 9. Four line segments are defined over the 1D domain $[a, b]$ (Fig. 9, left). The upper envelope is the pointwise maximum of all segments over the domain. The maximization diagram is a subdivision of the domain $[a, b]$ into cells, where each cell's identity is induced by the upper envelope. Alternatively, we can think of the maximization diagram as a projection of the upper envelope onto the domain (Fig. 9, right).

Let us now define the general upper envelope problem. Let $S = \{s_1, s_2 ...., s_n\}$ be n d-simplices in $(d + 1)$-dimensional space. A d-simplex has $(d + 1)$ vertices, i.e. $d = 1$ is a line segment, $d = 2$ is a

triangle and $d = 3$ is a tetrahedron. We can thus view each s_i, as the graph of a partially defined linear function $x_{d+1} = f_i(x_1, x_2, ..., x_d)$, whose domain of definition is a d-simplex, namely the orthogonal projection of s_i, onto the hyperplane $x_{d+1} = 0$. The upper envelope, M, of the given simplices is the pointwise maximum of these functions [Edelsbrunner et al. 1989], that is,

$$M(x_1, x_2, ..., x_d) = \max_{1 \le i \le n} f_i(x_1, x_2, ..., x_d) \qquad (16)$$

The maximization diagram M_S of S is the subdivision of R^d into connected cells obtained by the projection of the upper envelope of S in the x^d direction. The example in Fig. 9 corresponds to $d = 1$ where each line segment is a 1D simplex in 2D space. Fig. 8a shows an example of two 2-dimensional simplices in 3-dimensional space whose maximization diagram is shown in (Fig. 8b). Solutions for solving the general upper envelope problem for $d = 1$ and $d = 2$ ([Agarwal et al. 1996; Meyerovitch 2006]) and computing their corresponding maximization diagrams have been proposed. Abdrashitov et al. [2019] computes $d = 2$ upper envelope of "part" functions to extract the smoothed part boundaries.

We consider tissue functions $\mathcal{T} = \{f_1, ..., f_m, f_s\}$ (combination of muscle and fat functions) defined as scalar fields over the vertices of our tetrahedral mesh and we notice that Equation 14 is the pointwise maximum (Eq.16) of tissue functions. In our problem ($d = 3$) we are interested in finding maximization diagrams of all tissue functions over the volume constrained by the surface of the skin. This problem can be solved by considering finding the maximization diagrams of the tissue functions over each tetrahedron. In which case in contrast to the general upper envelope problem where our domain is a continuous hyperplane, we restrict our domain by a 3-simplex on that hyperplane. Fig. 8c shows an example of the maximization diagram of the functions defined by the scalar values at the vertices of the tetrahedron and (Fig.8d) shows the tetrahedralization of each cell. As a result, the tetrahedron is split into more sub-tets where each sub-tet has only one function that is the maximum. In other words, we can "assign" the sub-tet to one of the tissues. Performing this operation for every single tetrahedron results in a new tetrahedral mesh where each tetrahedron is assigned to one tissue. We can extract individual tissue shapes by simply combining all tetrahedrons that are assigned to that tissue. Fig. 8e shows an example of taking a uniform 3x3 tetrahedral mesh with two scalar functions defined over it and splitting it into two tetrahedral meshes (Fig.8f). The split is the result of computing tetrahedralized maximization diagrams of every tetrahedron. However, the resulting tetrahedral meshes representing each muscle are not guaranted to be manifold unless we are consistent with how we tessellate adjacent tetrahedrons (Fig. 10). In reality, we would just be getting a tetrahedral "soup" that is difficult to work with. Instead we want our resulting mesh to be manifold and hence we can easily use it for downstream applications. We propose an algorithm for computing manifold tetrahedralized maximization diagrams of functions defined as scalar fields over the vertices of a tetrahedral mesh. We first define auxiliary operations in Sections 4.1, 4.2 and then discuss the main algorithm in Sections 4.3.
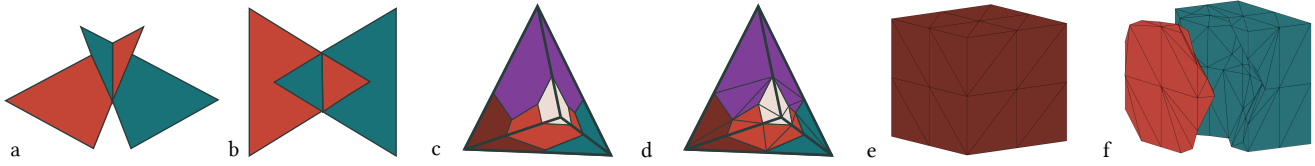
Fig. 8. Two triangles (a) and the corresponding maximization diagram (b). Example of the maximization diagram cells of 5 different functions over the tetrahedron (c) and tetrahedralization of each cell (d). Tetrahedral mesh (e) with 2 different functions defined over its vertices and the resulting tetrahedralized maximization diagram (one of the parts is slightly moved in the figure to show the internal tessellation)
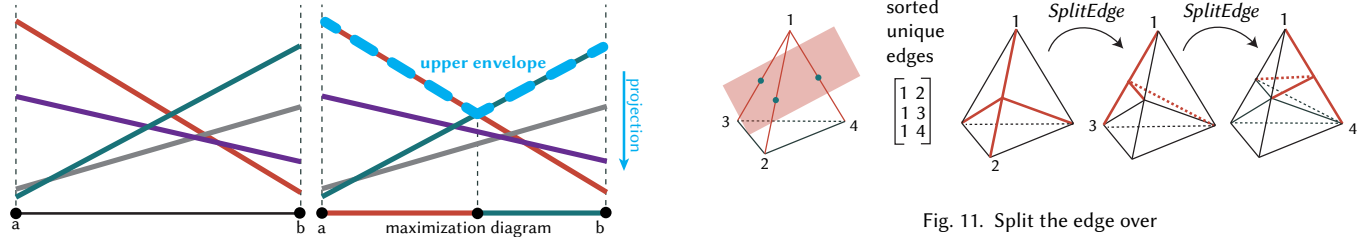

Fig. 9. Four line segments (red, green, purple, gray) are defined over a single 1D element $[a, b]$ (left). The upper envelope (light blue) and the maximization diagram (right).
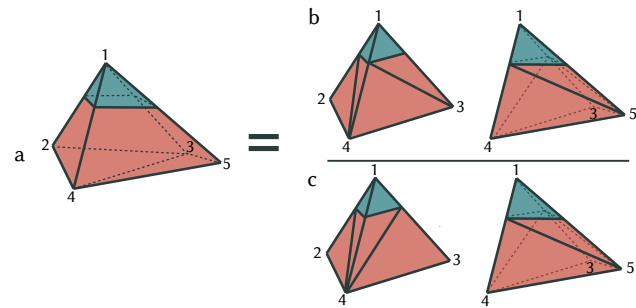

Fig. 11. Split the edge over

Fig. 10. Maximization diagram is defined over two adjacent tetrahedrons *1234* and *1345* (a). During the tessellation there is no guarantee that the shared face *134* is split the same way (b). By making sure the order of edge splits is consistent we can guarantee that the face is split the same way (c).

4.1 Prune tissues

We notice that if the tissue function $f_i \in \mathcal{T}$ is strictly below any other tissue function f_j, then it will not be part of the maximization diagram and hence can be ignored and help avoid unnecessary computations. We define the sparse matrix $\mathbf{W} \in \{0, 1\}^{m+1 \times m+1}$ where $\mathbf{W}_{ij} = 1$ if and only if there exist at least one tetrahedron in $\mathbf{T}$ whose maximization diagram contains cells assigned to tissues i and j.

4.2 Split tetrahedron

Given only two different tissue functions f_1 and f_2 over a tetrahedron we need to find the tetrahedralized maximization diagram that results from computing the upper envelope of those two functions.

This problem is similar to the *Marching tetrahedra* [Doi and Koide 1991] algorithm that finds an isosurface of a scalar field $f_1 - f_2$ passing through isovalue of 0. However, in our case we are not interested in extracting the isosurface but rather splitting the tet along the isosurface and tetrahedralizing the resulting polyhedrons.

Input: tetrahedron defined by 4 vertices $\mathbf{V} \in R^{4 \times 3}$, per-vertex values $\mathbf{f} \in R^4$ of a scalar function f.
Output: tet mesh $\mathbf{T}_{split}, \mathbf{V}_{split}$ resulted from splitting the input tet along the isosurface of f at isovalue of 0.

Algorithm: We find a set of **sorted** unique edges $\mathbf{E}$ of the tetrahedron that are intersected by the isosurface of f. The number of edges $|E|$ is either 3 or 4. We then define a *split_edge* operator that given a tetrahedron t and an edge e first splits the edge at a vertex v and then splits t into two tetrahedrons by connecting v with two vertices of the edge opposite e (Fig. 11). Similarly to *Marching tetrahdera* the location of v is determined by interpoloating vertices of e using weights defined by f. We simpy run *split_edge* recusively on each $e \in \mathbf{E}$ to produce the resulting tetrahedral mesh $\mathbf{T}_{split}$. The order of edges in $\mathbf{E}$ determines the order in which we split the edges and hence determines the final topology of the tessellation. By simply sorting the interested unique edges we solve the problem of inconsistent tessellation (Fig.10b) between adjacent tetrahedrons (Fig.10c). We additionally maintain a history of unique edge splits before splitting an edge e at vertex v. If the edge e was split before at vertex $\hat{v}$ by a call to *split_edge* on one of the adjacent tetrahedron we do not create a new vertex v but simply make it point to $\hat{v}$. To improve the robustness of the algorithm, when v almost coincides with one of the vertices in $\mathbf{V}$ (when one of the values in $\mathbf{f}$ is close to zero within some thereshold) we simply remove that edge from $\mathbf{E}$ so it will not be split.

4.3 Tetrahedralized Maximization Diagram

We iterate through every tissue function $f_i \in \mathcal{T}$ and use $\mathbf{W}$ to find a set of tissue funtions $\hat{\mathcal{T}}$ such that for any $f_j \in \hat{\mathcal{T}}, \mathbf{W}(i, j) > 0$. In other words $\hat{\mathcal{T}}$ contains all tissue functions that appear in the maximization diagrams with f_i in at least one tetrahedron. We compute all pairs $\{f_i, f_j\}$ of the tissue f_i with tissues in the set $\hat{\mathcal{T}}$. For every pair we iterate through all tetrahedrons $t \in \mathbf{T}$ with corresponding 4 vertices $\mathbf{V}_t$ and call the *SplitTet* subroutine. Algorithm 1 summarizes our divide and conquer approach to computing tetrahedralized maximimzation diagram of tissue functions $\hat{\mathcal{T}}$ over the tetrahedral mesh $\mathbf{T}, \mathbf{V}$.

Algorithm 1: Tetrahedralized Maximization Diagram

Input: Tet mesh $\mathbf{T}, \mathbf{V}$ and per-vertex tissue values
      $\mathbf{X} \in \mathbb{R}^{|\mathbf{V}| \times (m+1)}$

Output: Tet mesh $\mathbf{T}_{out}, \mathbf{V}_{out}$ and per-tet tissue labeling
      $\mathbf{L}_{out}$

$\mathbf{W}$ = PruneTissues(X); # Section 4.1 **for** *each tissue f_i* **do**

   $\hat{\mathcal{T}}$=FindIntersectingTissues(f_i, $\mathbf{W}$) **for** *every pair*

   $(f_i, f_j) \in \hat{\mathcal{T}}$ **do**

      for $t \leftarrow 1$ *to* $|\mathbf{T}|$ **do**

         $\mathbf{V}_{st}, \mathbf{T}_{st}$ = SplitTet($\mathbf{V}_t$, $\mathbf{X}(t, i) - \mathbf{X}(t, j)$);

         //using barycentric interpolation to compute all
           tissues values for all $v \in \mathbf{V}_{st}$

         $\mathbf{X}_{st}$ = InterpolateDiffusion($\mathbf{V}_{st}$, $\mathbf{T}$, $\mathbf{V}$, $\mathbf{X}$);

         //replace t with first tetrahedron in $\mathbf{T}_{st}$, append
           the rest

         UpdateT($\mathbf{T}$, $\mathbf{T}_{st}$);

         $\mathbf{V} = [\mathbf{V}; \mathbf{V}_{st}]$;

         $\mathbf{X} = [\mathbf{X}; \mathbf{X}_{st}]$;

      end

   end

end

4.4 Fiber direction

Fiber directions play an important role in simulation by defining the directions the muscle contracts. Fibers tend to point in the same directions as gradients of a hypothetical flow from one end of the muscle to the other [Choi and Blemker 2013; Saito et al. 2015]. Given muscles tetrahedral mesh $\mathbf{V}_m, \mathbf{T}_m$, we leverage our muscle curve to guide the global flow of the fiber field. The flow should align with the tangent space of the muscle curve. This can be formulated via constraint that for each tetrahedron along the curve we would like the projection of the gradient onto the normal space to be zero.

Let $\mathbf{T}_{mc} \in \mathbf{T}_m$ be a subset of tetrahedrons that the muscle curve intersects. Then the desired "flow" can be found by solving the energy minimization problem:

$$\min \mathbf{u}^T \mathbf{L}_c \mathbf{u} + \alpha (\mathbf{NGu})^T (\mathbf{NGu}) \tag{17}$$

$$\text{subject to } \mathbf{u}_{start} = 0 \tag{18}$$

$$\mathbf{u}_{end} = 1 \tag{19}$$

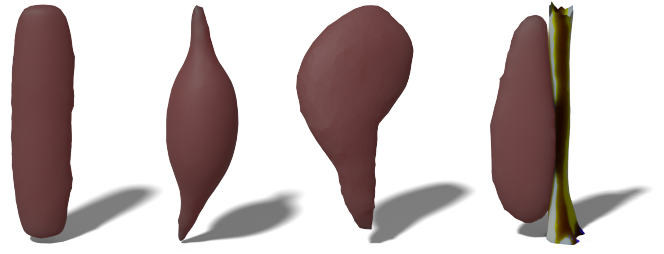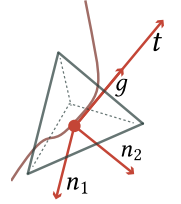$$\frac{\partial \mathbf{u}_{skin}}{\partial \hat{\mathbf{n}}} = 0 \tag{20}$$

Fig. 12. Parallel, Fusiform, Triangular, Unipennate.

where $\mathbf{G} \in R^{3|\mathbf{T}_{mc}| \times |\mathbf{V}_m|}$ is a gradient matrix, $\mathbf{N} \in \mathbb{R}^{2|\mathbf{T}_{mc}| \times 3|\mathbf{T}_{mc}|}$ is sparse matrix containing 2 vectors per gradient vector representing the null space of the curves tangent vector, $\mathbf{u}_{start}$ and $\mathbf{u}_{end}$ are a subset of vertices within a threshold distance of the curve endpoints. This term forces the gradient of the scalar field in $\mathbf{T}_{mc}$ to align with the curve's tangent vector. The inset figure shows an example of a tetrahedron that intersects the muscle curve where gradient $\mathbf{g}$ of the scalar field is aligned with the tangent vector $\mathbf{t}$ via constraining $\mathbf{g}$ to be orthogonal to the null space of $\mathbf{t}$ which is represented by $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$. The fiber field can then be computed as the normalized gradient of $\mathbf{u}$, i.e., $\nabla \mathbf{u} / ||\nabla \mathbf{u}||$.

5 IMPLEMENTATION AND RESULTS

Skeletal and muscle curve authoring tools are implemented as separate standalone C++ applications using Eigen [Guennebaud et al. 2010], libigl [Jacobson et al. 2013], and, for the user interface, ImGui. The output of the skeletal authoring tool are skin and bone triangular meshes. We exploit the fact that due to symmetry one only needs to use half of the mesh for creating the muscles. Therefore, we cut both skin and bone meshes along the half-plane. The mesh vertices belonging to the cut boundary can be excluded from the constraints in Eq. 11, 12 to help create muscles that are suppose to cross the cut boundary. The half-skin and half-bone meshes are loaded by the muscle-curve authoring tool and we use TetGen [Si 2015] to generate a tetrahedral mesh. The quadratic solver is implemented using the modified version of libIGL's *min_quad_with_fixed* function with Eigen's PardisoSupport module which provides significant speed ups. The tool was tested on a computer with Intel Xeon CPU @ 2.40GHZ, Nvidia GTX1080 and 64GB of RAM and can perform at interactive rates for tetrahedral meshes with > 500k tetrahedrons. The geometry stays intact between curve editing operations, and we only update GPU buffers containing per-vertex diffusion values which is fast.

Figure 16 showcases the results created by the authors using our tool. We informally evaluated our tool with a professional animator who has experience with using Maya Muscles [Autodesk 2021] for creating musculoskeletal systems. They mentioned that using existing tools requires a significant amount of tedious work for ensuring that muscle geometry respects the geometry of its surroundings (bones, skin, other muscles), and the overall workflow is "unintuitive" and "awkward". They liked the fact that our tool
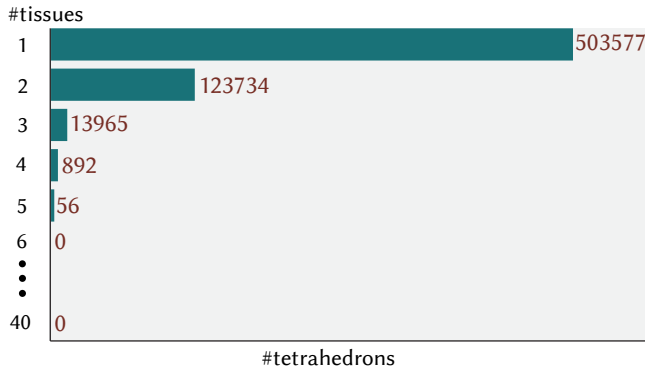
Fig. 13. We consider the Lion model in Fig. 16 with 39 muscles and 642224 tetrahedrons. The plot shows the number of tetrahedrons and the maximum number of tissue functions that can appear in their corresponding maximization diagram. This shows that for the vast majority of tetrahedrons (78%) computing their tetrahedralized maximization diagrams requires no splitting or only trivial non-recursive splitting of 2 tissues (19%). No tetrahedrons require resolving intersection of 6 or more tissues.

allows one to simply draw the location of the muscle while handling these issues automatically and provides enough intuitive controls (changing tissue values, moving control points, enabling anisotropy) to create complex muscle shape (Fig.12). One feature they wanted to see added is volume preservation of the muscle: when a muscle is being flattened in one direction, there is a corresponding stretching in another direction so that the volume stays the same. They also wanted an ability to specify multiple attachment points instead of just two to enable finer control, especially for triangular types of muscle shapes. We further informally evaluated our tool with an orthopedic surgeon with extensive knowledge of human anatomy but who does not have experience with 3D modelling applications. They see the potential of using this tool to explain the surgery process to their patients. By using a pre-existing anatomical muscle model made with our tool, they would be able to edit the muscles to explain their medical procedure. However, similar to the animator, they mentioned that to create truly anatomically correct muscles, they need to be able to specify multiple attachment points.

The muscle extraction algorithm in Section 4 uses the divide and conquer approach by considering pairs of tissue functions. In Figure 13, using the example of the Lion model (Fig.16), we show that in practice the vast majority of tetrahedrons only contain 1 tissue function in their maximization diagram and therefore are not split. While the vast majority of tetrahedrons that do need to be split only contain 2 tissue functions, which means they are only split once.

Because our volumetric domain is represented as an unstructured mesh, the resulting topology of the muscle meshes ends up containing *skinny* tetrahedrons as shown in Figure 14. However, because we guarantee that the resulting meshes are manifold, it is straightforward to extract the boundary faces (faces that belong to only one tetrahedron). The resulting muscle surface triangular meshes can be easily remeshed and tetrahedralized again. In practice, we simply run TetWild [Hu et al. 2018] with default parameters on the surface mesh to improve the topology.
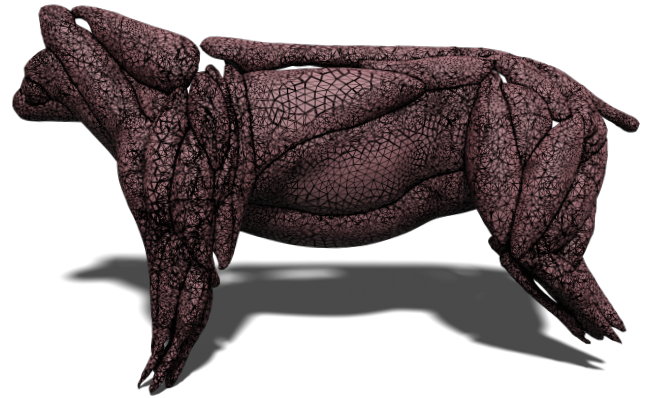

Fig. 14. Example of the final tessellation produced by the upper envelope algorithm.
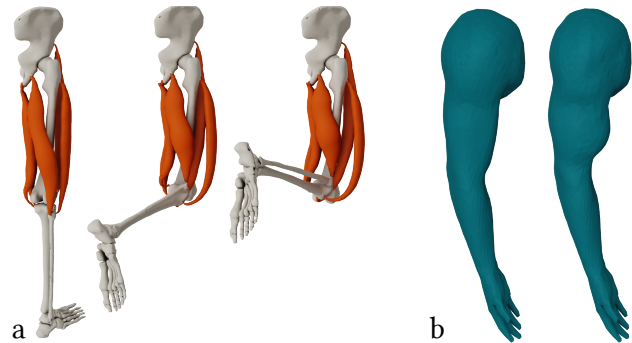

Fig. 15. Simulating the motion of a leg using muscle geometry created with our tool without contact handling. Motion is induced by contracting the hamstring using the method of Modi et al. [2020] (a). Dynamic Finite Element simulation of isometric contraction of the bicep using linear tetrahedral finite elements. Muscles are modeled as Neohookean elastic solids using the fiber model of Teran et al. [2003]. Time integration is performed via Implicit Euler time stepping using the Bartels library [Levin 2020]. We refer the reader to the provided *Additional Modelling and Simulation Demos* (07:30m) video to see the animated examples. Bone geometry (left) is provided by ©Ziva Dynamics. Used under permission.

During the modelling process the muscles seen by the user are directly volume rendered upper-envelopes: rendering does not require performing any meshing operations during the modeling process. Every time the user creates or edits muscle curves we recompute values of the tissue functions (Eq. 10) for each vertex of the tetrahedral mesh and update those values in the corresponding GPU buffer. These values are later accessed by the fragment shader to perform the volumetric rendering using the ray-marching method (supported by all modern GPUs). The muscle visualization using fragment shaders is fast (>30fps) on our system. The bottleneck of our system during modeling is computing the solution to (Eq. 10) for which we currently use the Eigen library with Pardiso enabled. This solve immediately benefits from any generic Poisson solving optimizations which are orthogonal to our main contributions. We only perform a meshing operation (Sec. 4) after the modelling session is
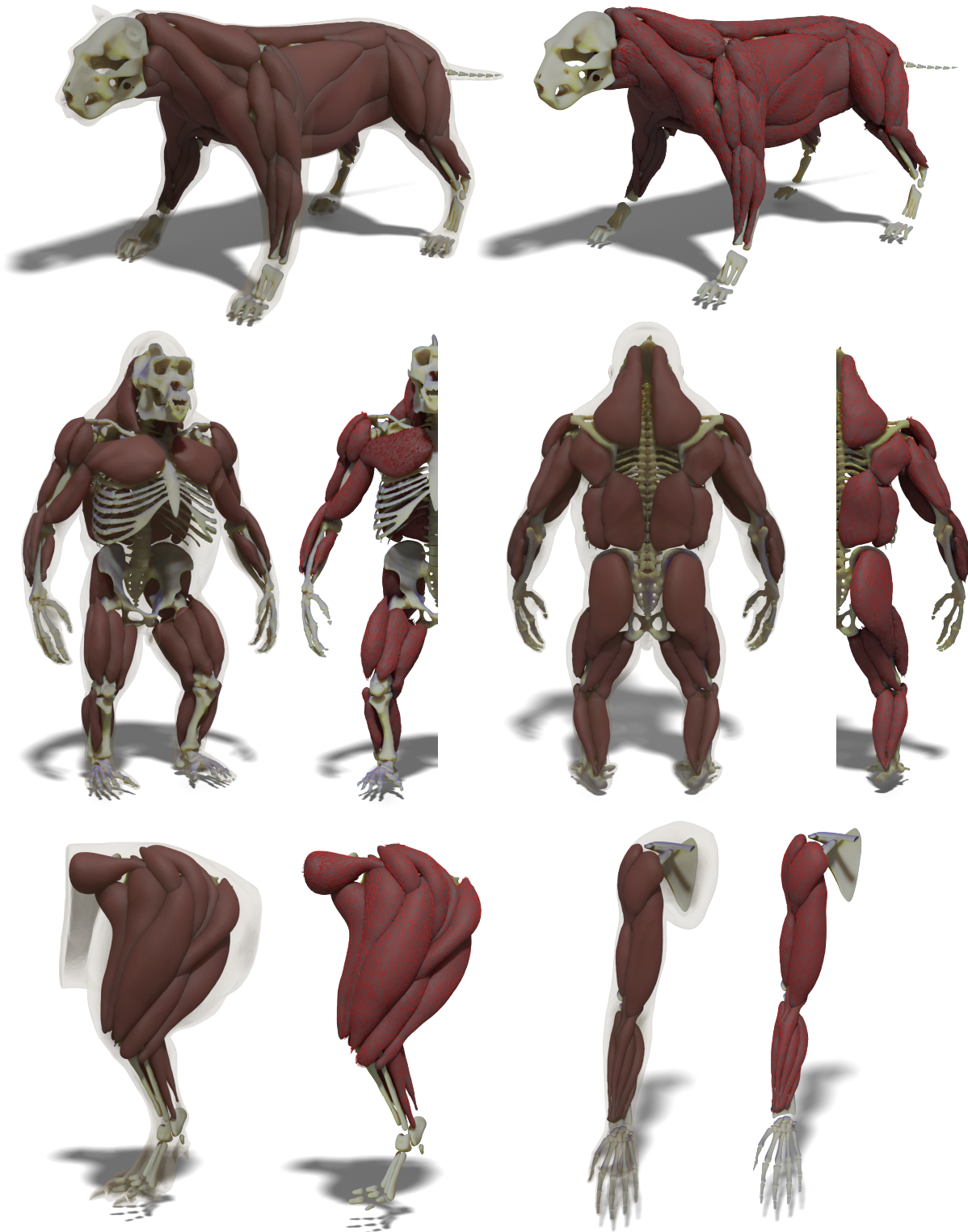
Fig. 16. Results. We refer the reader to the provided supplemental video to see the modelling process of the Ape model. We refer the reader to the provided *Additional Modelling and Simulation Demos* video to see the modelling process of the Lion model. Lion (top) and Dinosaur Leg (bottom left) are provided by ©SideFx Software. Used under permission. Ape model (middle) is obtained from https://www.3dscanstore.com/ecorche-3d-models/gorilla-ecorche-3d-model. Used under permission. Bone geometry of the arm (bottom right) is provided by ©Ziva Dynamics. Used under permission.

complete, to export muscles as tetrahedral meshes for downstream applications. Using muscle extraction every time we create/edit a muscle would negatively impact performance, which is the motivation for our volumetric rendering approach.

6 LIMITATIONS AND FUTURE WORK

Although we demonstrated that our tool enables users to create complex muscular-skeletal geometries, there are limitations, subject to future work.

The quality of the muscle shapes depends on the quality of the tetrahedrazation (**V**,**T**) of the input skin and bone meshes. On one hand, coarse meshes prevent the creation of smooth and thin muscles; on the other hand, very high quality meshes can affect the performance of computing muscle functions and hinder the interactive experience. This makes it hard to create muscles in the areas where bones are too close to the skin, unless the area is densely tessellated. We also notice that often a large portion of the model ends up being unused (the intestines area), yet it's still being used for computation of muscle functions which wastes computation. In practice, for all models, we were able to find a balance between the quality of the tetrahedralization and performance. In the future, we want to explore adaptive boundary conforming tessellation of the volume with increased tessellation quality only around boundaries of the muscles.

We use a simple heuristic (Sec. 3.2) for creating initial muscle curves, which leads to multiple editing operations of control points to get the muscle curve shape just right. This makes it challenging to create muscle curves around complex bone geometry, since control points must be edited from multiple viewpoints. In the future, we would like to explore a more robust curve creation techniques that simultaneously utilizes skin and bone geometry to create the best guess for the initial shape of the muscle curve.

Currently, each muscle is represented by a single curve. Based on the feedback we received from users, we plan to explore how we can use multiple curves to represent a single muscle. This would allow us to create more anatomically correct and complex muscle shapes with multiple origin and insertion points. Additionally, there are many muscles in which fiber direction does not match the flow between origin and insertion and therefore we plan to explore ways to specify the pennation angle (the angle between the longitudinal axis of the entire muscle and its fibers) as it is an important parameter in muscle contraction dynamics.

We have investigated the potential use of the resulting muscle geometry for simulation (Fig. 15) but more work needs to be done to test simulations with the complex and densely packed full-body muscle systems. With the help of professional artists, we plan to evaluate how animation-ready our resulting muscle geometry is using industry-standard software like Houdini [SideFx 2021] and Ziva Dynamics [2021].

Our system is designed with the idea of creating muscle geometry from scratch but we have experimented with extracting curve networks from pre-existing artist-made muscle shapes (using methods for computing curve skeletons of 3D shapes) and importing them into our system, which can provide a good starting point.

We currently do not support locking of the muscle shape. However, we could extract the mesh of a muscle, and treat it similarly to bones so that other muscles wrap around it. This requires to redo the initial tessellation to make sure it conforms to imported muscle geometries but TetGen only takes a few seconds. Similarly an artist can import existing muscle meshes and continue to create muscles around them.

REFERENCES

Rinat Abdrashitov, Alec Jacobson, and Karan Singh. 2019. A system for efficient 3D printed stop-motion face animation. *ACM Transactions on Graphics (TOG)* 39, 1 (2019), 1–11.

Pankaj K Agarwal, Otfried Schwarzkopf, and Micha Sharir. 1996. The overlay of lower envelopes and its applications. *Discrete & Computational Geometry* 15, 1 (1996), 1–13.

Irene Albrecht, Jörg Haber, and Hans-Peter Seidel. 2003. Construction and Animation of Anatomically Based Human Hand Models. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03)*. Eurographics Association, Goslar, DEU, 98–109.

Dicko Ali-Hamadi, Tiantian Liu, Benjamin Gilles, Ladislav Kavan, Francois Faure, Olivier Palombi, and Marie-Paule Cani. 2013. Anatomy transfer. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 188.

Mathieu Andreux, Emanuele Rodola, Mathieu Aubry, and Daniel Cremers. 2014. Anisotropic Laplace-Beltrami operators for shape analysis. In *European Conference on Computer Vision*. Springer, 299–312.

Baptiste Angles, Daniel Rebain, Miles Macklin, Brian Wyvill, Loic Barthe, JP Lewis, Javier Von Der Pahlen, Shahram Izadi, Julien Valentin, Sofien Bouaziz, et al. 2019. VIPER: Volume invariant position-based elastic rods. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 2, 2 (2019), 1–26.

Autodesk 2021. Maya Muscle. http://download.autodesk.com/us/support/files/muscle.pdf..

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. CRC press.

J. E. Chadwick, D. R. Haumann, and R. E. Parent. 1989. Layered Construction for Deformable Animated Characters. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '89)*. Association for Computing Machinery, New York, NY, USA, 243–252. https://doi.org/10.1145/74333.74358

Nuttapong Chentanez, Ron Alterovitz, Daniel Ritchie, Lita Cho, Kris K Hauser, Ken Goldberg, Jonathan R Shewchuk, and James F O'Brien. 2009. Interactive simulation of surgical needle insertion and steering. In *ACM SIGGRAPH 2009 papers*. 1–10.

Hon Fai Choi and Silvia S Blemker. 2013. Skeletal muscle fascicle arrangements can be reconstructed using a laplacian vector field simulation. *PloS one* 8, 10 (2013), e77576.

Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller, and Robert Jagnow. 2002. A procedural approach to authoring solid models. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 302–311.

Chris De Paoli and Karan Singh. 2015. SecondSkin: sketch-based construction of layered 3D models. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–10.

Deepak Rajan 2021. Cassowary | Ziva Dynamics | Anatomy Modeling | Part 3. https://www.youtube.com/watch?v=AD6TAQHnZwk&ab_channel=DeepakRajan.

Akio Doi and Akio Koide. 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems* 74, 1 (1991), 214–224.

Herbert Edelsbrunner, Leonidas J Guibas, and Micha Sharir. 1989. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete & Computational Geometry* 4, 4 (1989), 311–336.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Andrew J Hanson and Hui Ma. 1995. Parallel transport approach to curve framing. *Indiana University, Techreports-TR425* 11 (1995), 3–7.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Trans. Graph.* 37, 4 (2018), 60–1.

James Jacobs, Jernej Barbic, Essex Edwards, Crawford Doran, and Andy van Straten. 2016. How to build a human: practical physics-based character animation. In *Proceedings of the 2016 Symposium on Digital Production.* 7–9.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and John P Lewis. 2014. Skinning: Real-time shape deformation (full text not available). In *ACM SIGGRAPH 2014 Courses.* 1–1.

Alec Jacobson, Daniele Panozzo, et al. 2013. libigl: A simple C++ geometry processing library. http://igl.ethz.ch/projects/libigl/.

Petr Kadleček, Alexandru-Eugen Ichim, Tiantian Liu, Jaroslav Křivánek, and Ladislav Kavan. 2016. Reconstructing personalized anatomical models for physics-based body animation. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–13.

Francois Labelle and Jonathan Richard Shewchuk. 2007. Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles. *ACM Transactions on Graphics* 26, 3 (July 2007), 57.1–57.10. http://graphics.cs.berkeley.edu/papers/Labelle-ISF-2007-07/ Special issue on Proceedings of SIGGRAPH 2007.

Dongwoon Lee, Michael Glueck, Azam Khan, Eugene Fiume, and Ken Jackson. 2010. A survey of modeling and simulation of skeletal muscle. *ACM Transactions on Graphics* 28, 4 (2010), 1–13.

David I.W. Levin. 2020. Bartels: A lightweight collection of routines for physics simulation. https://github.com/dilevin/Bartels.

Duo Li, Shinjiro Sueda, Debanga R Neog, and Dinesh K Pai. 2013. Thin skin elastodynamics. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

Michal Meyerovitch. 2006. Robust, generic and efficient construction of envelopes of surfaces in three-dimensional spaces. In *European Symposium on Algorithms.* Springer, 792–803.

Vismay Modi, Lawson Fulton, A Jacobson, S Sueda, and David IW Levin. 2020. EMU: Efficient Muscle Simulation in Deformation Space. In *Computer Graphics Forum.* Wiley Online Library.

Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. 2007. FiberMesh: designing freeform surfaces with 3D curves. In *ACM SIGGRAPH 2007 papers.* 41–es.

Victor Ng-Thow-Hing and Eugene Fiume. 1997. Interactive display and animation of B-spline solids as muscle shape primitives. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation 1997, Budapest, Hungary, September 2-3, 1997 (Eurographics)*, Daniel Thalmann and Michiel van de Panne (Eds.). Springer, 81–97. https://doi.org/10.1007/978-3-7091-6874-5_6

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–8.

Shigeru Owada, Takahiro Harada, Philipp Holzer, and Takeo Igarashi. 2008. Volume Painter: Geometry-Guided Volume Modeling by Sketching on the Cross-Section.. In *SBM.* 9–16.

Jonathan Palacios, Chongyang Ma, Weikai Chen, Li-Yi Wei, and Eugene Zhang. 2016. Tensor field design in volumes. In *SIGGRAPH ASIA 2016 Technical Briefs.* 1–4.

Nico Pietroni, Paolo Cignoni, Miguel A Otaduy, and Roberto Scopigno. 2010. A survey on solid texture synthesis. *IEEE Computer Graphics and Applications* 30, 4 (2010), 74–89.

Nico Pietroni, Miguel A Otaduy, Bernd Bickel, Fabio Ganovelli, and Markus Gross. 2007. Texturing internal surfaces from a few cross sections. In *Computer Graphics Forum*, Vol. 26. Wiley Online Library, 637–644.

Michael Pratscher, Patrick Coleman, Joe Laszlo, and Karan Singh. 2005. Outside-in anatomy based character rigging. In *Proceedings of the 2005 ACM SIG-GRAPH/Eurographics symposium on Computer animation.* ACM, 329–334.

Valentin Roussellet, Nadine Abu Rumman, Florian Canezin, Nicolas Mellado, Ladislav Kavan, and Loïc Barthe. 2018. Dynamic implicit muscles for character skinning. *Computers & Graphics* 77 (2018), 227–239.

Shunsuke Saito, Liwen Hu, Chongyang Ma, Hikaru Ibayashi, Linjie Luo, and Hao Li. 2018. 3D hair synthesis using volumetric variational autoencoders. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–12.

Shunsuke Saito, Zi-Ye Zhou, and Ladislav Kavan. 2015. Computational bodybuilding: Anatomically-based modeling of human bodies. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. 1997. Anatomy-Based Modeling of the Human Musculature. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 163–172. https://doi.org/10.

1145/258734.258827

Johannes Schmid, Martin Sebastian Senn, Markus Gross, and Robert W Sumner. 2011. Overcoat: an implicit canvas for 3d painting. In *ACM SIGGRAPH 2011 papers.* 1–10.

Hang Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)* 41, 2 (2015), 1–36.

SideFx 2021. Houdini. https://www.sidefx.com/.

Cláudio Teixeira Silva, Joao Luiz Dihl Comba, Steven Paul Callahan, and Fabio Fedrizzi Bernardon. 2005. A survey of GPU-based volume rendering of unstructured grids. *Revista de informática teórica e aplicada. Porto Alegre, RS. Vol. 12, n. 2 (out. 2005), p. 9-29* (2005).

Maryann Simmons, Jane Wilhelms, and Allen Van Gelder. 2002. Model-Based Reconstruction for Creature Animation. In *Proceedings of the 2002 ACM SIG-GRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. Association for Computing Machinery, New York, NY, USA, 139–146. https://doi.org/10.1145/545261.545284

Karansher Singh, Jun Ohya, and Richard Parent. 1995. Human figure synthesis and animation for virtual space teleconferencing. In *1995 Virtual Reality Annual International Symposium, VRAIS '95, Research Triangle Park, North Carolina, USA, March 11-15, 1995.* IEEE Computer Society, 118–126. https://doi.org/10.1109/VRAIS.1995.512487

Oded Stein, Eitan Grinspun, Max Wardetzky, and Alec Jacobson. 2018. Natural Boundary Conditions for Smoothing in Geometry Processing. *ACM Trans. Graph.* 37, 2, Article 23 (May 2018), 13 pages. https://doi.org/10.1145/3186564

Kenshi Takayama, Daniele Panozzo, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. 2013. Sketch-based generation and editing of quad meshes. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–8.

Kenshi Takayama, Olga Sorkine, Andrew Nealen, and Takeo Igarashi. 2010. Volumetric modeling with diffusion surfaces. In *ACM SIGGRAPH Asia 2010 papers.* 1–8.

J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. 2003. Finite Volume Methods for the Simulation of Skeletal Muscle. In *Proceedings of the 2003 ACM SIG-GRAPH/Eurographics Symposium on Computer Animation (SCA '03)*. Eurographics Association, Goslar, DEU, 68–74.

Joseph Teran, Eftychios Sifakis, Silvia S Blemker, Victor Ng-Thow-Hing, Cynthia Lau, and Ronald Fedkiw. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics* 11, 3 (2005), 317–328.

Winnie Tsang, Karan Singh, and Eugene Fiume. 2005. Helping Hand: An Anatomically Accurate Inverse Dynamics Solution for Unconstrained Hand Motion. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '05)*. Association for Computing Machinery, New York, NY, USA, 319–328. https://doi.org/10.1145/1073368.1073414

Fabio Turchet, Oleg Fryazinov, and Sara C Schvartzman. 2017. Physically-based Muscles and Fibers Modeling from Superficial Patches. (2017).

Lvdi Wang, Yizhou Yu, Kun Zhou, and Baining Guo. 2011. Multiscale vector volumes. *ACM Transactions on Graphics (TOG)* 30, 6 (2011), 1–8.

Manfred Weiler, Martin Kraus, Markus Merz, and Thomas Ertl. 2003. Hardware-based ray casting for tetrahedral meshes. In *IEEE Visualization, 2003. VIS 2003.* IEEE, 333–340.

Weta Digital 2021. Weta Digital. https://www.wetafx.co.nz/research-and-tech/technology/tissue/.

Jane Wilhelms and Allen Van Gelder. 1997. Anatomically Based Modeling. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 173–180. https://doi.org/10.1145/258734.258833

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: Neural Rigging for Articulated Characters. *arXiv preprint arXiv:2005.00559* (2020).

Yusuke Yoshiyasu, Wan-Chun Ma, Eiichi Yoshida, and Fumio Kanehiro. 2014. As-conformal-as-possible surface registration. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 257–267.

Christopher Yu, Henrik Schumacher, and Keenan Crane. 2020. Repulsive Curves. *arXiv preprint arXiv:2006.07859* (2020).

Zhan Yuan, Yizhou Yu, and Wenping Wang. 2012. Object-space multiphase implicit functions. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–10.

Ziva Dynamics 2021. Ziva VFX. https://zivadynamics.com.