Surface and Curve Skeletonization of Large 3D Models on the GPU

Andrei C. Jalba$^{(a)}$, Jacek Kustra$^{(b)}$, and Alexandru C. Telea$^{(c)}$

***Abstract*— We present a GPU-based framework for extracting surface and curve skeletons of 3D shapes represented as large polygonal meshes. We use an efficient parallel search strategy to compute point-cloud skeletons and their distance and feature transforms with user-defined precision. We regularize skeletons by a new GPU-based geodesic tracing technique which is orders of magnitude faster and more accurate than comparable techniques. We reconstruct the input surface from skeleton clouds using a fast and accurate image-based method. We also show how to reconstruct the skeletal manifold structure as a polygon mesh and the curve skeleton as a polyline. Compared to recent skeletonization methods, our approach offers two orders of magnitude speed-up, high precision, and low memory footprints. We demonstrate our framework on several complex 3D models.**

***Index Terms*— Medial axes, Geodesics, Skeleton regularization.**

I. Introduction

Skeletons, or medial axes, are shape descriptors used in virtual navigation, shape matching, shape reconstruction, and shape processing [62]. 3D shapes admit two types of skeletons. *Surface skeletons* are 2D manifolds which contain the loci of maximally-inscribed balls in a shape [50], [62]. *Curve skeletons* are 1D curves which are locally centered in the shape [16]. Surface-skeleton points, together with their distance to the shape and closest-shape points, define the medial surface transform (MST) which is used for animation, smoothing, and matching [4], [6], [17].

Computing *surface* skeletons of complex 3D objects represented by polygonal meshes is still a difficult problem. Challenges include: (i) finding accurate skeletons; (ii) removing spurious skeleton branches (*i.e.*, regularizing skeletons); (iii) producing skeleton-mesh models for use in subsequent algorithms; and (iv) efficient computation with respect to speed and memory.

In this paper, we present a framework for computing surface and curve skeletons which fulfills the above requirements, with the following contributions:

- a refinement of the skeleton extraction method in [40] which exploits CPU and GPU parallelism for increased performance;
- a general-purpose GPU geodesic tracing method which is two magnitude orders faster, and more accurate, than similar techniques. We use this method to efficiently apply global regularization [55] to large skeletons;
- an extension of the above regularization method with a new detector to compute robust curve skeletons;
- a new image-space method to reconstruct shapes directly from the (regularized) MST in real time;
- accurate extraction of skeleton manifolds from MST clouds.

Section II reviews related work. We next present our skeleton cloud extraction (Sec. III), image-based reconstruction from skeleton clouds (Sec. IV), geodesic tracing for regularization (Sec. V), curve skeleton extraction (Sec. VIII), and skeleton-mesh reconstruction from skeleton clouds (Sec. VI). Section VII compares our work with a recent surface skeleton extractor. Section IX discusses our results. Section X concludes the paper.

The authors are with: (a) Institute for Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands; (b) Philips Research, Eindhoven, The Netherlands; (c) Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, The Netherlands. E-mail: {a.c.jalba@tue.nl, jacek.kustra@gmail.com, a.c.telea@rug.nl}.

II. Related work

Given a shape $\Omega \subset \mathbb{R}^3$ with boundary $\partial\Omega$, we first define its distance transform $DT_{\partial\Omega} : \mathbb{R}^3 \to \mathbb{R}^+$

$$DT_{\partial\Omega}(\mathbf{x} \in \Omega) = \min_{\mathbf{y} \in \partial\Omega} \|\mathbf{x} - \mathbf{y}\|. \tag{1}$$

The skeleton, or medial axis, of Ω is next defined as

$$\begin{aligned} S(\Omega) = \{\mathbf{x} \in \Omega \,|\, \exists \mathbf{f}_1, \mathbf{f}_2 \in \partial\Omega, \mathbf{f}_1 \neq \mathbf{f}_2, \\ \|\mathbf{x} - \mathbf{f}_1\| = \|\mathbf{x} - \mathbf{f}_2\| = DT_{\partial\Omega}(\mathbf{x})\}, \end{aligned} \tag{2}$$

where $\mathbf{f}_1$ and $\mathbf{f}_2$ are the contact points with $\partial\Omega$ of the maximally inscribed ball in Ω centered at $\mathbf{x}$ [28], [55]. The contact points $\mathbf{f}_1$ and $\mathbf{f}_2$ are also called *feature transform* (FT) points [65]. The vectors $\mathbf{f} - \mathbf{x}$ determined by skeleton points and their corresponding feature points are also called *spoke vectors* [63]. $S(\Omega)$ is a set of manifolds with boundaries which meet along a set of Y-intersection curves [13], [18], [37]. $S(\Omega)$ can be computed by various methods, as follows.

Voxel-based methods include thinning, distance-field, and general-field methods. Thinning removes $\partial\Omega$ voxels (or pixels in 2D) while preserving connectivity [7], [48]. Voxel removal in distance-to-boundary order enforces centeredness [53]. Distance-field methods find $S(\Omega)$ along singularities of $DT_{\partial\Omega}$ [27], [31], [35], [38], [56], [60], [71], [74] and can be efficiently done on GPUs [11], [65], [67], [72]. General-field methods use fields smoother (with fewer singularities) than distance transforms [1], [4], [16], [30]. Such methods are more robust for noisy shapes. Foskey *et al.* compute the θ-SMA, an approximate simplified medial axis, using the angle between feature vectors [24]. The θ-SMA can get disconnected along the so-called ligature branches. The θ-HMA refines the θ-SMA by computing medial axes which are homotopy-equivalent to the input surface [67].

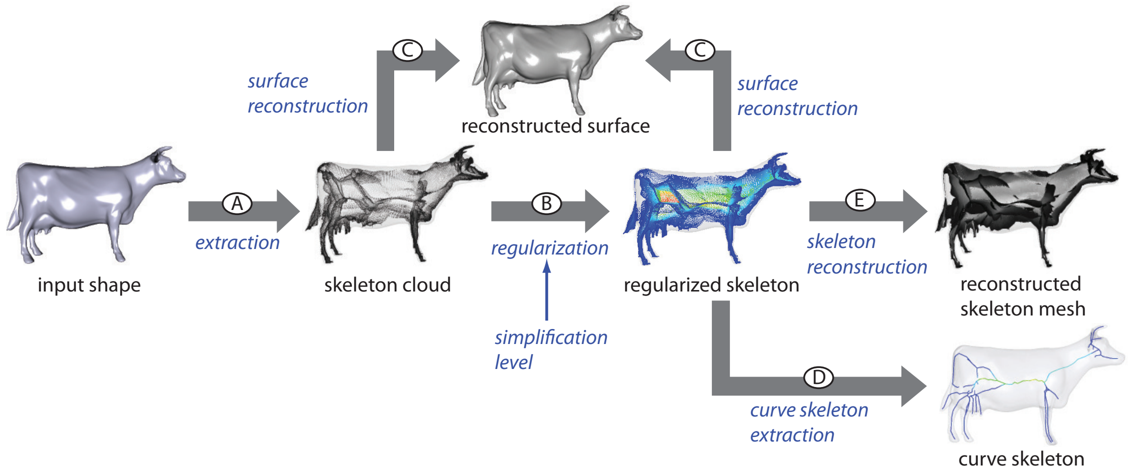

Fig. 1. Our pipeline: skeleton cloud extraction (A), regularization (B), surface reconstruction (C), curve-skeleton extraction (D), and skeleton-mesh reconstruction (E). All steps except E (CPU-only) are implemented on both the CPU and GPU.

Mesh-based methods often use Voronoi diagrams to compute polygonal skeletons [21]. Amenta *et al.* compute the Power Crust, an approximation of a surface and its medial axis by a subset of Voronoi points [3]. Other methods use edge collapses [39], starting from a mesh segmentation [33], or sphere sweeping [44]. Mesh-based methods compute precise 3D skeletons, can handle non-uniformly sampled surfaces, and use much less memory – typically $O(n^2)$ as compared to $O(n^3)$ needed for a n^3 voxel volume. However, such methods are quite complex to implement, can be numerically unstable, and are harder to parallelize [66].

Clean skeletons are extracted from noisy shapes by thresholding *importance measures* to prune points caused by small details [58]. We distinguish between local and global measures [43], [55]. Local measures cannot separate locally-identical, yet globally-different, contexts (see *e.g.* [55], Fig. 1). Thresholding local measures can disconnect skeletons. Reconnection needs extra work [41], [50], [60], [67], and makes pruning less intuitive [58]. Local measures include the angle between the feature points and distance-to-boundary [3], [24], divergence-based [9], [60] and first-order moments [56]. Stolpner *et al.* find skeleton voxels where the gradient of the shape's distance transform is multi-valued [63], [64]. Precision and speed are increased by a voxel-and-point-cloud approach which subdivides voxels close to the skeleton. Leymarie and Kimia topologically simplify point-cloud skeletons to capture Y intersection curves and skeleton sheet boundaries in medial scaffolds [37]. Good surveys of skeletonization theory are given in [62], [64].

Global measures monotonically increase from the skeleton boundary inwards. Thresholding them yields connected skeletons. Miklos *et al.* approximate shapes by a union of balls (UoB) and use UoB medial properties [29] to simplify skeletons [43]. Dey and Sun present the medial geodesic function (MGF) which is equal to the shortest-geodesic length between feature points [20], [52]. Related work on curve skeletons is given in Sec. VIII. Reniers *et al.* [55] extend the MGF for surface and curve skeletons using geodesic lengths and surface areas between geodesics, respectively, inspired by the so-called 2D collapse metric [17], [47], [71]. Besides monotonicity, the MGF and its 2D collapse metric counterpart have an intuitive geometric meaning: They assign to a skeleton point the amount of shape boundary that corresponds, or 'collapses to', the respective skeleton point. Hence, skeleton points with small metric values correspond to small-scale shape details, and skeleton points with large metric values correspond to large-scale shape details. This allows an easy simplification of the skeleton: Thresholding by a value τ eliminates all skeleton points which encode less than τ boundary length units. However, for large 3D meshes, computing the geodesics required to evaluate the MGF is an expensive process.

Recently, Ma *et al.* proposed arguably the fastest method to extract surface skeletons from oriented point clouds [40]. However, this method produces a 'raw' surface-skeleton *point cloud* which, as the authors note too, are of limited use if one requires a compact skeleton surface or a curve skeleton. We extend their proposal in several directions (see Fig. 1): (A) faster skeleton-cloud extraction, (B) regularization, (C) object reconstruction, (D) curve-skeleton extraction, and (E) surface-skeleton mesh reconstruction. These steps are described next.

III. Surface skeleton extraction

Given an input shape $\partial\Omega$, we extract a skeleton point-cloud from an oriented point-cloud model $C = \{(\mathbf{p}_i, \mathbf{n}_i)\}$ containing points $\mathbf{p}_i$ and their surface normals $\mathbf{n}_i$, where the point coordinates are normalized to $[-1,1]^3$ for simplicity. Our method is based on the ball shrinking algorithm of Ma *et al.* [40] which works as follows. For each point $\mathbf{p} \in C$, a (large) ball $B(\mathbf{s}, r_0)|\mathbf{s} = -r_0\mathbf{n} + \mathbf{p}$, tangent at $\mathbf{p}$, is created. By definition, $\mathbf{f}_1 \equiv \mathbf{p}$ is the first feature point of $\mathbf{s}$. The algorithm shrinks B by searching the closest point $\mathbf{f}_2$ to $\mathbf{s}$ and iteratively moving $\mathbf{s}$ so that B passes through $\mathbf{f}_1$ and $\mathbf{f}_2$ until $\mathbf{s}$ converges. At that moment, B is maximally inscribed, and its center $\mathbf{s}$ yields a new skeleton point with inscribed radius $r_{ins} = \|\mathbf{s} - \mathbf{f}_1\| = \|\mathbf{s} - \mathbf{f}_2\|$; for full details see [40]. We next propose two performance improvements for this method.

A. CPU Parallelization

Ma *et al.* propose an efficient sequential CPU implementation. Key to their method is a heuristic that sets the initial radius r_0 for a ball $B(\mathbf{s}, r_0)$ being shrunk to the radius of a

TABLE II
PERFORMANCE OF THE METHOD BY MA *et al.* [40].

Model	Points (surface)	**CPU timings** (seconds)	**GPU timings** (seconds)
Armadillo	106289	4.20	0.39
Dragon	437645	20.23	2.64
Horse	48485	1.62	0.29

skeleton point already found for a surface point $\tilde{\mathbf{p}}$ close to $\mathbf{p}$. This greatly reduces the number of shrinking steps (see [40], Sec. 3). However, this requires *sequential* processing of the cloud C in a global distance-based ordering, computed by in-order visiting a *kd* tree containing C.

We parallelize this idea as follows. We use a single global value r_0, initially set to 2. Next, we divide C into N equal-sized chunks (without any point ordering) and process one chunk per thread. When a thread finds a new value r_{ins}, we set r_0 to $(r_0+r_{ins})/2$, *i.e.* adapt r_0 in a moving-average fashion. Additionally, we stop shrinking the ball when two consecutive center positions differ by a value less than an user-specified value τ. Finally, we use an *approximate* nearest-neighbor (NN) scheme based on *kd* trees [45] to search for the closest point $\mathbf{f}_2$ to $\mathbf{s}$ with precision ε.

Table I shows our skeleton extraction timings t and average number k of *kd*-tree searches per point for different τ and ε values, on several models, on a 4-core 2.4 GHz CPU with $N=8$ threads. For the first three models, timings by the method of Ma *et al.* are also given in Table II. Smaller τ values need more iterations; larger τ values yield less accurate skeletons quicker. We see that k is quite stable for all models. For a skeleton centeredness precision $\tau = 10^{-3}$, our method is roughly four times faster than the sequential method of Ma *et al.* which use also a 2.4 GHz CPU. Given our 4-core CPU, this implies a very efficient parallelization. Relaxing ε slightly accelerates this search (Tab. I, column $\varepsilon = 10^{-3}$) with little accuracy loss.

Ball-shrinking yields a surface-skeleton cloud $S = \{(\mathbf{s},\mathbf{f}_1,\mathbf{f}_2)_i\}$ with two feature points $\mathbf{f}_1$, $\mathbf{f}_2$ per skeleton point $\mathbf{s}$ (see Fig. 6). The local density of S is proportional with the input cloud density times the shape curvature [62]. Thus, we get more skeleton points where the surface changes rapidly and/or is finely sampled, see, *e.g.*, the cow's horns, ears, and pig's snout in Fig. 6. Controlling this density is easy: For more points, we refined the initial mesh, using the Yams package [25]. For fewer points, we uniformly subsample S by removing all points closer to each skeleton point than some distance δ. Figure 2 shows the skeleton of a cow model for four different δ values. Conceptually, our method is similar to Leymarie *et al.* [37]: We regard each input point $\mathbf{f_1}$ as a medial axis 'generator', try to pair it with another point $\mathbf{f}_2$, and test maximality of the resulting balls. However, while [37] explicitly compute pairs $(\mathbf{f}_1,\mathbf{f}_2)$ *and* check for ball maximality using search strategies based on visibility constraints, we implicitly compute the pairs *while* shrinking balls.

B. GPU parallelization

For an efficient transfer of ball shrinking to GPUs, we need (a) an efficient GPU nearest-neighbor (NN) search and (b) an effective load balancing between GPU threads.

To these ends, Ma *et al.* proposed a mixed CPU-GPU approach [40]. For NN search, they use the GPU algorithm in [76]. For load balancing, ball-shrinking iterations are done in parallel on the GPU. After *each* GPU iteration, threads ask the CPU whether each ball needs more iterations. If so, the CPU invokes the GPU kernel for the next iteration only for the not-yet-converged balls. As Ma *et al.* mention, this achieves good performance, but cannot use the initial radius heuristic, as that heuristic was designed for a sequential algorithm.

Our GPU proposal directly parallelizes the CPU algorithm with one thread per skeleton point. Since our initial radius heuristic works in a parallel setting, we transfer it directly to the GPU.

In contrast to Ma *et al.*, we use a GPU-only load balancing scheme. This poses some subtle constraints on the choice of the NN technique used. For this, we investigated several options. Garcia *et al.* showed a GPU brute-force NN algorithm [26] which turned to be 20 to 30 times slower than our CPU KD-tree search [45]. Cayton's NN algorithm covers the input set of N points by a randomly-distributed set of $m \ll N$ overlapping balls which are searched in parallel on the GPU [12]. However, this method cannot do several NN searches in parallel which we need to parallelize ball extraction. Left-balanced GPU KD-trees also perform poorly. Such trees are rather deep (maximum depth $D = \log N$ for N input points). Also, the unfavorable distribution of query points (potential skeleton points having at least two shape points at equal distances) causes many tree node visits during a NN search, *i.e.* many random, uncached, memory accesses which seriously degrade GPU performance.

For our context, a KD-tree with alternating splitting dimensions and median-based pivoting proved best. We limit the tree depth to a small value (10 to 12). Leaf nodes store more than one point and are linearly searched. Linear search performs very well on the GPU [26] (more cache hits and coalesced memory accesses), yielding a better overall NN speed.

Table I shows the speed of our GPU method on an Nvidia GTX 280. Since the GPU NN search is exact, these timings should be compared with CPU timings for $\varepsilon = 0$. Thus, our GPU method is 4 to 10 times faster than its CPU variant. Compared to the GPU method of Ma *et al.* (see Table II), we are about 3 times faster, as we use an initial radius heuristic which their method does not support.

IV. IMAGE-BASED SURFACE RECONSTRUCTION

We now present an efficient and simple algorithm for reconstruction of a model from its skeleton cloud (Fig. 3). For each skeleton point $\mathbf{s}_i$ with radius r_i, we build a viewplane-aligned quad Q, or billboard, of world-space edge size 2. We texture Q scaled to $(r_i,r_i,1)$ with a $D \times D$ texture whose texels $T(u,v)$ encode both depth and shading. If fixed-point texturing is available, T uses a 32-bit RGBA format: The first 3 bytes of $T(u,v)$ encode the height at $(u/D,v/D)$ of a half-ball of radius 1 centered in the texture (see insets in Fig. 3). The fourth texture byte (A) encodes the ball color computed, *e.g.*, with Phong shading. If floating-point textures are available,

TABLE I
PERFORMANCE OF OUR SKELETON EXTRACTION METHOD ON BOTH CPU AND GPU (SEE SEC. III).

Model	Points (surface)	Points (skeleton)	**CPU timings** (t seconds, k iterations) $\tau=10^{-4},\varepsilon=0$ k	t	$\tau=10^{-3},\varepsilon=0$ k	t	$\tau=\varepsilon=10^{-3}$ t	**GPU timings** $\tau=10^{-3},\varepsilon=0$ t
Armadillo	106289	106289	5	1.29	3	0.87	0.87	0.21
Dragon	437645	436439	5	6.53	2	5.46	5.44	1.21
Horse	48485	48485	5	0.77	3	0.58	0.58	0.09
Cow	185882	185703	5	3.63	3	2.82	2.77	0.64
Bird	46866	46862	5	0.44	3	0.35	0.34	0.03
Horse 2	193934	193887	5	7.51	3	5.77	5.77	1.32
Asiandragon	231574	230964	4	2.83	2	2.19	2.19	0.22
Asiandragon 2	954027	951010	5	17.12	3	13.10	11.82	1.57
Hand	197245	196920	4	3.32	2	2.50	2.50	0.51
Elephant	50485	50422	5	0.62	3	0.48	0.48	0.05
Buddha	543652	541762	4	6.76	3	5.47	5.37	1.42
Mouse	403652	402301	6	5.61	3	3.07	3.02	0.34
Pig	225282	224539	5	7.48	3	4.21	4.17	1.47
Armadillo 2	172974	172955	6	3.29	3	1.86	1.86	0.47
Rabbit	124998	123811	6	1.31	3	0.85	0.83	0.06

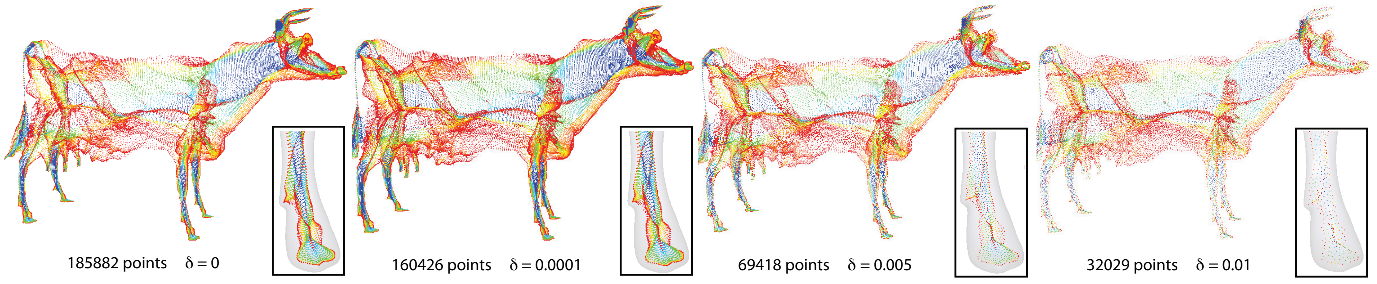

Fig. 2. Uniform skeleton sampling for different δ values (see Sec. III). Colors show angles of feature vectors (θ-SMA detector).

we encode the height and shading in the L and A channels of a luminance-alpha texture. This leaves two texture channels for future potential use. The texture size D is set to 512. This yields highly accurate shading and height encoding even for very large balls. For maximal rendering speed, we store T on $log_2(D)$ precomputed mip-map levels.

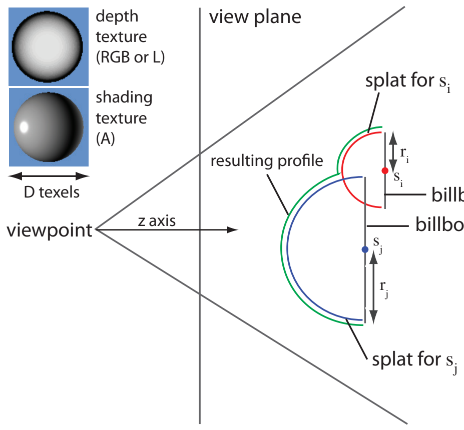

Fig. 3. Skeleton splatting for surface reconstruction.

We render the quads by a simple ARB fragment program shader (7 operations) which gets the z_{NDC} normalized device coordinate (NDC) of the current skeleton point $\mathbf{s}_i$ via the current color. The shader then computes the final NDC depth $z_{NDC}+h$ at the current pixel from the incoming z_{NDC} and ball height h (from the current texel). If $z+h$ passes the depth test, the current texel's color is copied to the fragment output. The overall effect is as if 3D balls centered at the skeleton points, and scaled to the respective skeleton-point radii, are rendered with hidden surface removal.

Our method renders shaded models directly from skeleton clouds of 500K points at 15 frames/second on a GT 330M card. If lighting changes, we only recompute one shading texture. The x,y splat sizes are pixel-accurate (by OpenGL scaling). Depth values are either 24-bit fixed-point of floating point, hence one can use whichever of the two is readily available on one's platform. Overall, our splatting-based reconstruction method delivers images nearly identical to the original mesh (see Fig. 4).

Progressive rendering, like the Qsplat method [57], is easily done by drawing billboards sorted by a skeleton importance metric, *e.g.* ball radius or our geodesic metric in Sec. V. If we use the geodesic metric, this always produces compact shapes. This result cannot be guaranteed by pure surface splatting like Qsplat. Alternatively, using a simplified skeleton as input for the reconstruction allows us to obtain simplified renderings of the input shape. For example, if we use a surface skeleton simplified by the geodesic-based importance metric presented next in Sec. V, we obtain a shape where edges have been smoothed out; if we use the curve skeleton presented next in Sec. VIII, we obtain a tubular approximation of the input shape. Other shape simplification effects can be achieved by choosing suitable skeleton simplification metrics.

Thickness estimation: Our splatting can also be used to estimate the so-called local shape thickness, also called wall thickness, a known task in 3D metrology [5], [22], [36], [62], [70]. Given a shape $\Omega\subset\mathbb{R}^3$, the thickness at a point $\mathbf{p}\in\partial\Omega$ is defined as the distance from $\mathbf{p}$ to the closest point on the

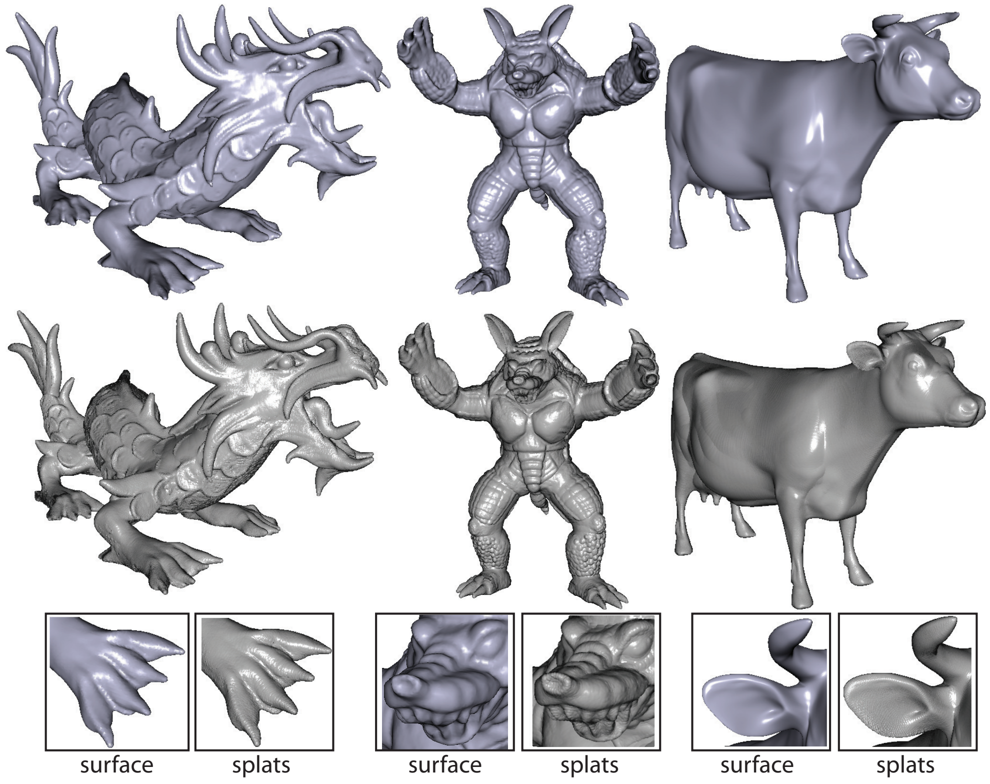

Fig. 4. Comparison of surface rendering (top row) and skeleton image-based surface reconstruction (middle row). Insets show details.

skeleton $S(\Omega)$ to $\mathbf{p}$. We can efficiently evaluate (and visualize) the thickness at all points on $\partial\Omega$ by simply mapping the skeleton points' radii r_i to their splat colors via a gray-to-red (thin-to-thick) colormap (Fig. 5). Reconstructing the shape by ball splatting will now show thin surface areas as red and thick areas as white (see Fig. 5). Compared to typical curvature estimation used for the same task, we need no differentiation and work directly on a point cloud. Our method is fast: The models in Fig. 5 take under 0.2 seconds with our method, and several seconds on identical hardware with a recent voxel-based thickness estimator [70].

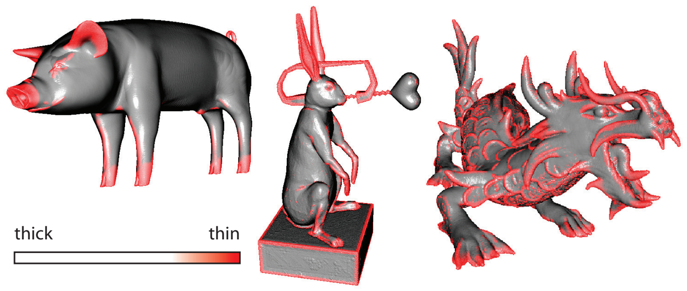

Fig. 5. Thickness estimation using image-based skeleton splatting.

Union of balls (UoB): Our image-based skeleton splatting delivers the same result as an UoB model, *e.g.*. [43], [64]. Our splatting could be used as drop-in shape reconstruction for any method that delivers an MST point cloud. As we shall see in Sec. VII, our method is roughly one to two orders of magnitude faster than [43] and over two orders of magnitude faster than [64].

V. Skeleton regularization

Skeleton regularization assigns an importance value $\rho : S \to \mathbb{R}^+$ to skeleton points (Sec. II). If ρ monotonically increases from the skeleton boundary inwards, thresholding ρ yields a connected skeleton which captures shape details at a given scale. Such metrics are proposed in [20], [52], [55]: For a skeleton point $\mathbf{s}$ with feature points $\mathbf{f}_1$, $\mathbf{f}_2$, $\rho(\mathbf{s})$ is the length of the shortest path γ on $\partial\Omega$ between $\mathbf{f}_1$ and $\mathbf{f}_2$. Finding such paths can be done using Dijkstra's algorithm [55], computing the distance map DT_{f_1} of $\mathbf{f}_1$ by the Fast Marching Method (FMM) and then tracing γ in $-\nabla DT_{f_1}$ from $\mathbf{f}_2$ [49], or hybrid search techniques [68], [73]. However, such methods are very slow, as we shall see next.

A. Shortest and straightest geodesics

We compute the skeleton importance ρ using discrete *straightest geodesics* on polyhedral surfaces [32], [51] which generalize straight lines to arbitrary manifolds. Given a point $\mathbf{p} \in \partial\Omega$ and a tangent vector $\mathbf{v} \in T_p$ at $\mathbf{p}$, the (discrete) straightest geodesic γ_S is the unique solution of the (discrete) initial-value problem $\gamma_S(0) = \mathbf{p}$, $\gamma_S'(0) = \mathbf{v}$ [51]. We extend this to define the (discrete) *shortest, straightest geodesic* (SSG) γ_{se} between two points $\mathbf{s}, \mathbf{e} \in \partial\Omega$, over tangent vectors $\mathbf{v}_i \in T_s$ at $\mathbf{s}$, as the solution of the discrete boundary-value problem

$$\begin{aligned} &\gamma_{S,i}(0) = \mathbf{s},\ \gamma_{S,i}'(0) = \mathbf{v}_i \\ &\gamma_{S,i}(\|\gamma_{S,i}\|) = \mathbf{e} \\ &\gamma_{se} = \underset{i}{\operatorname{argmin}} \|\gamma_{S,i}\|, \end{aligned} \tag{3}$$

where $\|\gamma_{S,i}\|$ is the length of the discrete geodesic $\gamma_{S,i}$. Thus, γ_{se} is the shortest among all straightest geodesics between $\mathbf{s}$ and $\mathbf{e}$. Solving Eqns. 3 is not easy. Speed-wise, the cost is proportional to the number of tangent directions $\mathbf{v}_i$ considered. Also, current algorithms for computing straightest geodesics [32], [51] estimate γ' by evaluating the (discrete) Gaussian curvature at the mesh vertices visited while tracing. The tangent vectors to γ may change directions especially when this curvature is not exactly 2π, so geodesics may not reach their endpoints $\mathbf{e}$, which is critical for our goal. All these concerns are addressed next.

B. Efficient SSG computation

For a skeleton point $\mathbf{s}$, we trace M straightest geodesics $\gamma_{S,i}$, $1 \le i \le M$ in parallel on the CPU or GPU between the feature points $\mathbf{f}_1$ and $\mathbf{f}_2$ of $\mathbf{s}$ on $\partial\Omega$, with starting angles $\alpha_i = (2\pi i)/M$ uniformly spread around the vector $\mathbf{f} = \mathbf{f}_1 - \mathbf{f}_2$ at $\mathbf{f}_1$. For each direction $\mathbf{v}_i$, we intersect the edges of the mesh faces visited while tracing by the plane with normal $\mathbf{n}_i = \mathbf{f} \times \mathbf{v}_i$, and set $\rho(\mathbf{s}) = \min_i \|\gamma_{S,i}\|$ *i.e.* the length of the SSG between $\mathbf{f}_1$, $\mathbf{f}_2$.

We speed up tracing by early termination: we stop tracing a path if its length exceeds the current $\rho(\mathbf{s})$. For a closed mesh, we consider both paths from $\mathbf{f}_1$ to $\mathbf{f}_2$ given by the mesh-plane intersection (closed) curve. When one of these paths is computed, we stop tracing the other path if its current length exceeds the first path's length. For a computed γ_{se}, we also store its tangent vectors $\mathbf{t}_s$ and $\mathbf{t}_e$ in $\mathbf{f}_1$ and $\mathbf{f}_2$ respectively, and use $\mathbf{t}_s$ as starting direction when tracing SSGs for the next skeleton point. Since neighbor skeleton points usually have similar geodesics [20], [54], early termination occurs sooner. This further speeds up tracing.

As shown in Fig. 6, ρ monotonically increases from the skeleton boundary (blue) inwards (red). Thresholding ρ removes skeleton points given by small shape details (Fig. 6 e,f,h,i). Such details can be surface noise (Fig. 6 d), but also appear in locally-tubular shapes (Fig. 6 f). In contrast, thresholding non-monotonic metrics such as θ-SMA (Fig. 2) would disconnect skeletons.

C. Performance and accuracy of SSG tracing

Table III shows the speed of our SSG method on a GTX 280 card *vs* a 2.8 GHz 4-core PC for $M = 20$ directions, one thread per SSG, and the speed-up due to the heuristics in Sec. V-B.

TABLE III
TIMINGS FOR COMPUTING THE GEODESIC IMPORTANCE.

Model	Timing CPU (seconds)	Timing GPU (seconds)	Optimization speed-up (%)
Cow	61.7	18.2	67.4
Bird	10.3	3.2	33.4
Horse	78.9	13.6	57.4
Asiandragon	61.5	13.8	60.0
Asiandragon 2	541.2	115.1	57.8
Hand	62.2	3.1	48.1
Elephant	7.1	3.3	57.1
Buddha	327.6	43.4	62.3
Mouse	210.3	52.1	69.0
Dragon	198.3	30.4	57.2
Pig	96.9	24.0	64.5
Armadillo	63.4	11.2	59.3
Rabbit	76.4	10.8	49.2

Speed: We compared our GPU SSG to FMM [49], the Dijkstra algorithm with A^* heuristics [55], the Surazhky-approximate (SA) and Surazhky-exact (SE) geodesic tracing [68], and CPU SSG (Sec. V-B). Of these, only SA and Dijkstra were used to regularize skeletons [20], [55]. GPU SSG is 3 to 10 times faster than CPU SSG (higher speed-ups for larger models, Tab. III); 10 times faster than Dijkstra; 100 times faster than FMM; 500 times faster than SA; and *thousands* of times faster than SE. This is not surprising: For a n-vertex mesh, Dijkstra is $O(n^2 \log n)$; FMM computes one distance field per vertex ($O(n \log n)$) and traces a geodesic in this field ($O(\sqrt{n})$, proportional to the shape diameter), *i.e.* is $O(n^2 \log n + n^{3/2})$. SA has the same complexity as FMM. SSG traces all geodesics for a point in parallel. As we have more GPU cores than M directions, GPU SSG is $O(n^{3/2})$.

Accuracy: All above methods, except SE, compute approximate geodesics. The starting angle sampling M (Sec. V-B) means that SSG may miss very narrow surface dents falling between two consecutive paths $\gamma_{S,i}$ and $\gamma_{S,i+1}$, *i.e.* may overestimate SSG length. Overestimation is not an issue for skeleton regularization: Long geodesics yield anyway high-importance points which are to be kept (red points, Fig. 6). Short geodesics, caused by surface noise which we want to eliminate by importance thresholding (blue points, Fig. 6), allow only much narrower concavities to fall between them, and thus are less affected by length overestimation.

TABLE IV
ACCURACY AND TIMING COMPARISON FOR GEODESIC TRACING METHODS (FMM, SE, AND SSG FOR DIFFERENT NUMBERS OF DIRECTIONS M).

Method	Relative error (%)		Timing (seconds)	
	Bird	Pig	Bird	Pig
Surazhky-exact	0	0	11209	426.2
FMM	1.87	0.85	339.6	20.9
SSG ($M = 5$)	1.90	0.36	0.28	0.05
SSG ($M = 10$)	0.36	0.07	0.56	0.08
SSG ($M = 20$)	0.14	0.02	1.04	0.14
SSG ($M = 30$)	0.08	0.009	1.54	0.22
SSG ($M = 50$)	0.04	$8 \cdot 10^{-4}$	2.43	0.32
SSG ($M = 100$)	0.02	$2 \cdot 10^{-4}$	4.83	0.62
SSG ($M = 500$)	0.01	$4 \cdot 10^{-5}$	23.24	2.95

Figure 7 and Table IV show the median relative error of SSG geodesic-lengths (for different M values), FMM, and SE. We used downsampled versions of *bird* and *pig* (11718 and 3522 points, respectively), since SE is extremely slow. SSG is more accurate than FMM for $M > 10$, both for median and maximum errors. For $M = 100$, SSG practically gets exact geodesics at a tiny cost *vs* SE. Comparing median errors with SA, SSG is more accurate for $M \geq 30$, see Tab. 1 in [68] where an upper relative error bound of 0.1% is used. This is equivalent to SSG with $M = 10$ directions for *bird* and $M = 30$ directions for *pig*. As mentioned, the cost of SA is the same as FMM, *i.e.* hundreds of times slower than SSG.

Memory: For a *single* geodesic tracing on a n vertex model, Dijkstra with A^* is $O(n)$; FMM is $O(n \log n)$; Surazhky *et al.* is $O(n^{3/2})$. SSG is $O(M)$. Verma and Snoeyink improved upon Surazhky *et al.* by combining Dijkstra with A^* with the original method [73]. This reduces the memory cost to $O(n)$, yields a speed-up of 8, but overestimates geodesic lengths

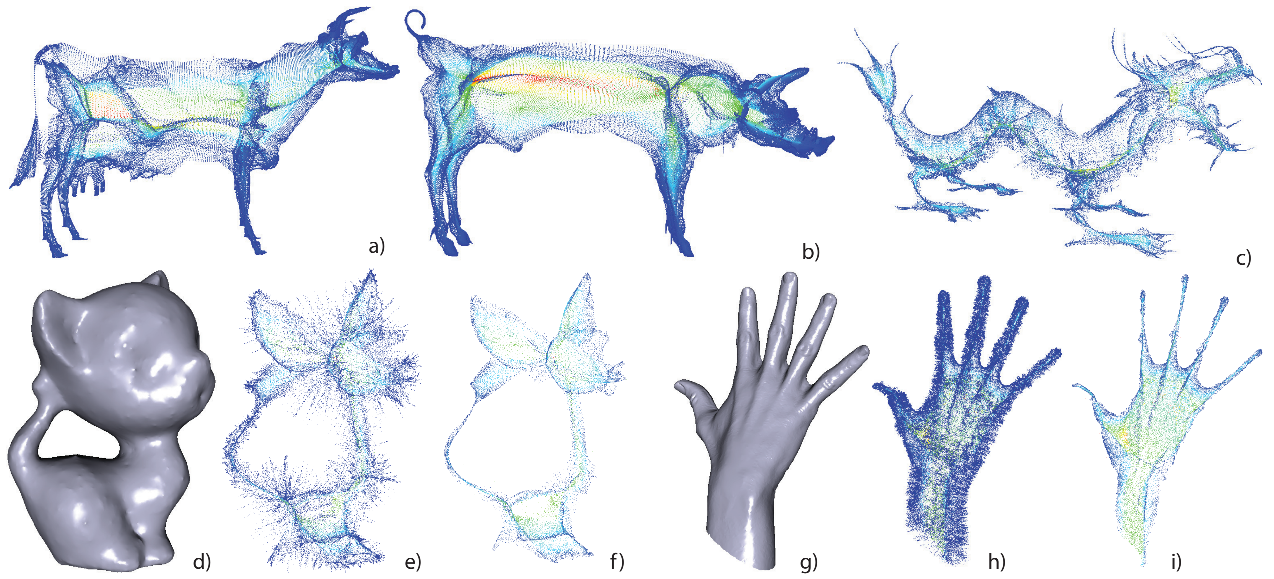

Fig. 6. Skeleton cloud regularization by geodesic importance. Red points are the most important. Blue points correspond to small surface features (Sec. V).

by 10% on average. Concluding, our GPU SSG method is a good trade-off: It is nearly as accurate as exact geodesics, and hundreds of times faster than approximate geodesic methods.

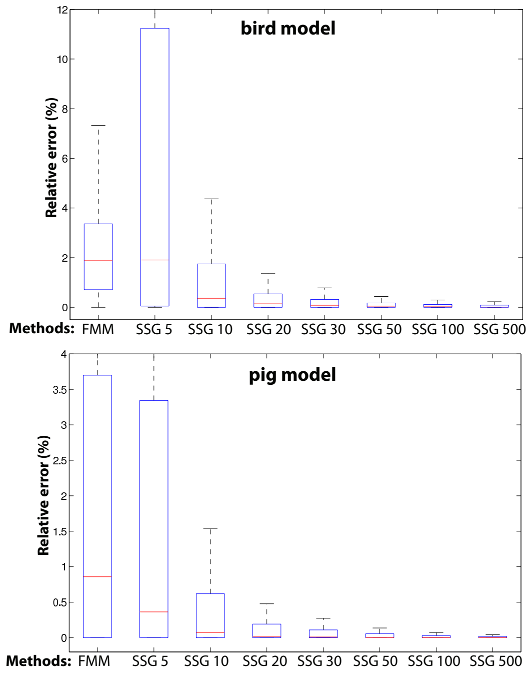

Fig. 7. Accuracy comparison: FMM *vs* SSG geodesic tracing (see Sec. V-C).

VI. Surface skeleton reconstruction

Shape comparison, topological analysis, or segmentation tasks require a mesh skeleton, not a point cloud. 3D skeletons contain many self-intersecting, closely-spaced, manifolds whose boundaries are the skeleton end-curves and Y-intersection curves. Hence, point cloud reconstruction methods for locally smooth and/or non-intersecting and/or watertight surfaces cannot be used [2], [19], [34]. Reconstruction of open, non-manifold, and self-intersecting surfaces [13], [75] is relatively slow and non-trivial. We present next two methods for reconstructing *skeleton* surfaces from point clouds based on specific skeleton properties.

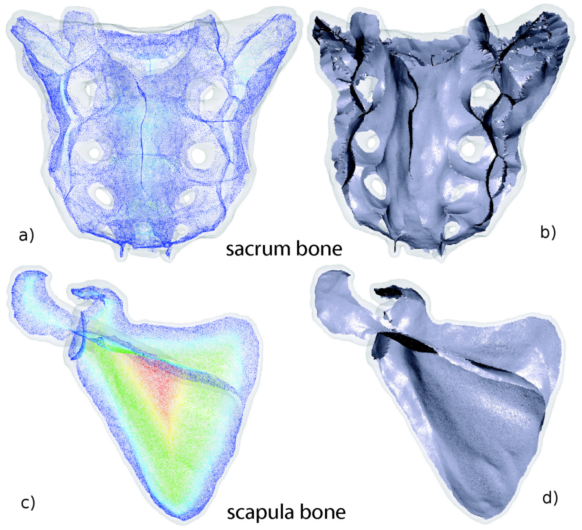

Fig. 9. Anatomic shapes: point clouds (a,c) and surface skeletons (b,d).

Delaunay reconstruction: For each triangle $F = \{\mathbf{f}_i\} \subset \partial\Omega$, we use FT^{-1}, the inverse of the feature transform (FT) computed at skeleton extraction (Sec. III) to gather all skeleton points $S(F)$ having $\mathbf{f}_i$ as feature points. Obtaining FT^{-1} is for free: For each skeleton point $\mathbf{s}$ found by the ball-shrinking algorithm in Sec. III, the ball shrinking also gives us its two feature points $\mathbf{f}_1$ and $\mathbf{f}_2$. By adding $\mathbf{s}$ to the shape points $\mathbf{f}_1$ and $\mathbf{f}_2$, we obtain FT^{-1} at the end of the ball shrinking, *i.e.*, for any shape point $\mathbf{p} \in \partial\Omega$, all its skeleton points $\mathbf{s} \in S(\Omega)$.

Next, we project all points in $S(F)$ on the plane of F, triangulate these projections [59], and use the resulting mesh patch to connect the points in $S(F)$. The reason for 'lifting' the connectivity from 2D into 3D is that locally planar surface patches (*i.e.*, triangles F) create, by definition, locally-planar skeleton patches (*i.e.*, triangulation of $S(F)$). This creates duplicate skeleton-mesh triangles, since close model faces have common skeleton points in convex areas. To remove

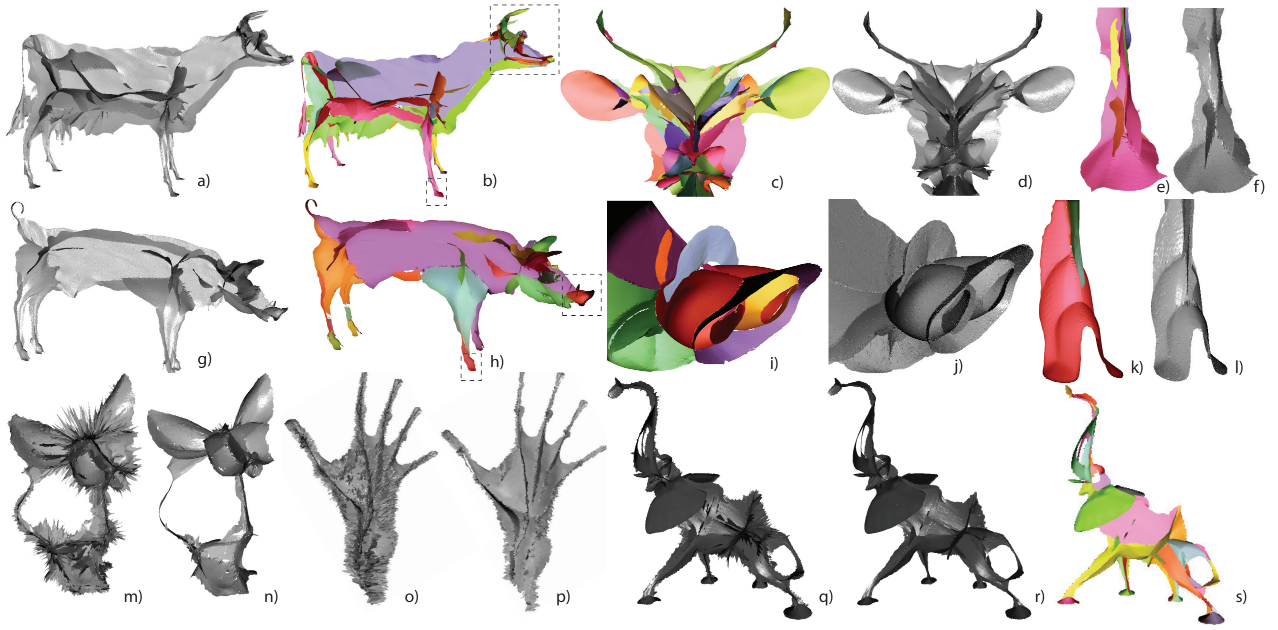

Fig. 8. Delaunay method (a,g; details d,f,j,l; simplified clouds (m-r) and per-manifold method (b,h,s; details c,e,i,k) for skeleton reconstruction (Sec. VI).

these, we mark all model vertices which map via the FT only to *internal* triangles, *i.e.*, which do not have edges on the boundary of a Delaunay triangulation [59], and skip faces having only marked vertices. The method is $O(N)$ for N skeleton points, since we triangulate small point sets $S(F)$ of size $O(1)$. This takes under 3 seconds for all shapes in this paper. We use only *local* information (skeleton points of a small surface neighborhood), so we can do out-of-core reconstruction of large skeletons, like [13].

Figures 8,9 show our results. All small details (cow eyes, hooves, horns, and pig snout) create skeletal manifolds. Noisy skeletons have no 'stitches' between close ligature sheets (Fig. 8 m,o,q). It is challenging to reconstruct such manifolds *only* from point clouds: Ligatures match surface concavity pairs [50], so their cloud density can be arbitrarily small even for densely sampled models (Sec. III). The inverse FT links ligatures to the input surface and thus reconstructs them well. Simplified skeleton meshes are easily created by filtering low-importance points (compare Figs. 8 n,p,r to the raw skeletons in Figs. 8 m,o,q).

Per-manifold reconstruction: The Delaunay method leaves a few tiny holes in the skeleton mesh (Fig. 8 d,f,j,l). Our second reconstruction method fixes these. First, we cluster skeleton points into separate manifolds. For this, we observe that a small ball $\nu_\varepsilon(\mathbf{s}) \subset S$ of radius ε around a skeleton point $\mathbf{s}$ maps to one or more small vicinities $\nu^i_{\partial\Omega}(\mathbf{s})$ of the input shape [50]. Points on the manifold end-curves, inside a manifold, and on Y-intersection curves have one, two, and three or more $\nu^i_{\partial\Omega}$ respectively. We cluster skeleton points into manifolds by a simple flood fill which groups adjacent vicinities $\nu_\varepsilon(\mathbf{s})$ having average feature vectors that differ less than an angle α (in practice, $\alpha = 5°$). The fill stops when we find a vicinity with a highly different feature vector, *i.e.*, at Y-intersection curves. We repeat the fill from a random unclustered point until all points are clustered. Finally, we reconstruct each cluster with the ball pivoting algorithm (BPA) [8] with ball radius ρ_{ball} set to the average inter-point distance in the cluster. BPA incrementally grows a triangle mesh surface, as follows: Starting with a seed triangle, a ball of a given size ρ_{ball} pivots around each triangle edge $(\mathbf{e}_1, \mathbf{e}_2)$, *i.e.*, revolves around the edge while keeping contact with the edge's endpoints, until it touches a point $\mathbf{p}$ from our point-cluster, and without containing any other cluster point except $\mathbf{p}$ and the edge's endpoints. When $\mathbf{p}$ is found, a new triangle $(\mathbf{p}, \mathbf{e}_1, \mathbf{e}_2)$ is added to the mesh. The process is repeated from the edges of the newly found triangle, until all possible edges have been considered. For full details, we refer to [8], and to the public MeshLab implementation of BPA we have used here [42].

Figure 8 b,h,s show results, with manifolds colored differently for illustration. Per-manifold reconstruction removes the small holes and inter-manifold stitches at Y-intersection curves of the Delaunay method (compare Figs. 8 c,e,i,k and d,f,j,l), at a higher execution time: 12 seconds for the cow and pig models.

VII. Surface Skeleton Comparison

We compare our point-cloud surface skeletons (PCS) with the discrete scale axis (DSA) method [43], one of the best methods for extracting detailed surface skeletons (Fig. 10). For similar skeleton simplification levels, PCS and DSA create similar skeletons (Fig. 10 g-h,m-n). Yet, differences exist (Fig. 10, red marks). These have two causes: geometry (different skeleton points found) and topology (same skeleton points found but connected differently). Geometry differences imply topology differences. Note how DSA found many skeleton points outside the hand model (Fig. 10 e-f) and connected these to points inside the hand. In the cow model (Fig. 10 c-d), skeleton points are largely identical, but DSA wrongly connects the tail and rump skeletons. Such issues, due to strong model concavities, are noted in [43].

DSA can skip large parts of the skeleton periphery, see *e.g.* the pig and cow spine and belly and elephant spine

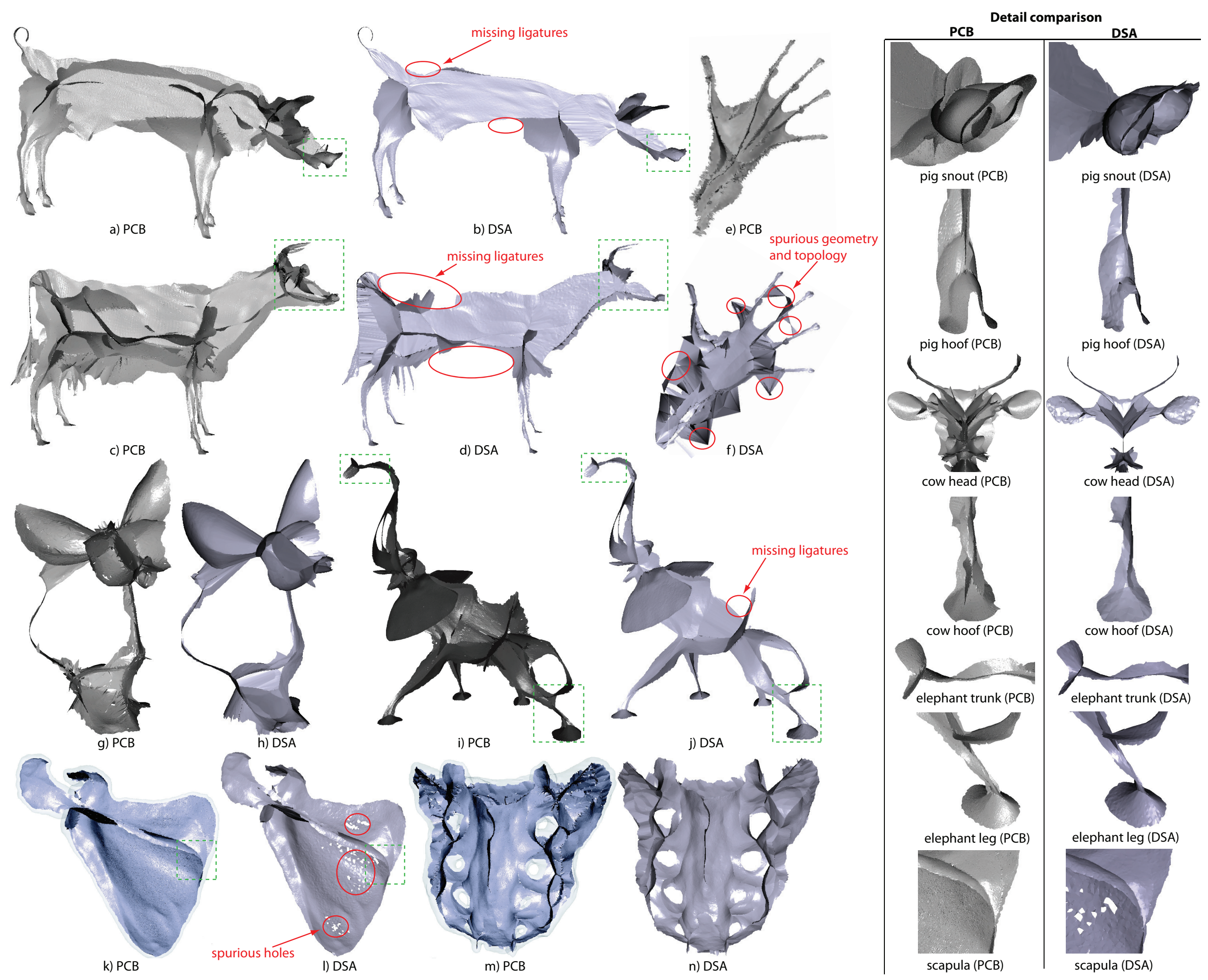

Fig. 10. Comparison of PCS and DSA methods (Sec. VII). Skeleton parts wrongly added/missed by DSA are marked red. Green-marked details are shown right.

(Fig. 10 b,d,j). These parts, found by our PCS, are *ligature* sheets between the core skeleton and faraway skeleton points in shallow surface cusps [50]. The reason hereof is that PCS and DSA use different skeleton *scale* metrics: PCS uses geodesic importance (a global metric); DSA computes simplified skeletons by up-scaling the input shape (a local operation). DSA also creates many small holes in skeletal sheets, see Fig. 10 l. These artifacts (for a genus 0 shape) are likely due to the numerical degeneracies listed in [43].

TABLE V
TIMING COMPARISON OF PCS AND DSA SKELETONIZATION METHODS.

Model	PCS (our method)		DSA(Miklos *et al.* [43])		
	time (sec.)	points	time (sec.) $\delta = 0.007$	time (sec.) $\delta = 0.005$	points
pig	27.9	224539	485	975	145153
cow	19.4	185703	448	919	64240
bird	3.3	46862	201	421	104140
horse	17.5	193887	369	744	63297
asiandragon	14.9	230964	n/a	n/a	n/a
hand	4.8	196920	378	778	93301
elephant	3.5	50422	505	1031	166089
armadillo	13.2	172955	426	853	105167

Table V shows timing and size statistics for PCS and DSA. For PCS, we used an accuracy $\tau = 10^{-3}$ (Sec. III), and also added regularization time. For DSA, we used approximation thresholds δ for the mesh to union-of-balls (UoB) conversion of 0.007 and 0.005 ([43], Sec. 2). Note that $\tau = 10^{-3}$ for PCS is similar to $\delta = 10^{-3}$ for DSA as for skeleton accuracy. PCS is up to 100 times faster than DSA. More accurate UoB settings (smaller δ) make DSA much slower: 45 minutes for a 313K-point skeleton of a 177K-triangle shape at $\delta = 0.002$ [43]. We could not test such settings as $\delta \leq 0.003$ made DSA crash on our models. The same was true for the asiandragon model, $\delta = 0.007$. If we replace our SSG regularization by simpler metrics, *e.g.* θ-SMA, PCS becomes much faster, basically identical to the timings in Tab. I.

PCS and DSA create skeletons of different sizes. PCS creates one skeleton point per input point (Sec. III). DSA uses a Voronoi diagram, which has a different point count. Yet, the skeletal *detail* created by PCS is similar to DSA (see Fig. 10).

We also compared PCS with the method of Stolpner *et al.* [64] which approximate medial axes as UoB point clouds. On a 3.4 GHz PC, PCS is over 100 times faster (Tab. V *vs* Tab. 1 [64]).

VIII. Curve skeleton extraction

Curve skeleton points are surface skeleton points which have two or more SSGs between their feature points [20], [55]. The above definition for curve skeleton points can be quite easily applied to detect such points for voxel-based models, as shown in [55]. For surface skeletons represented as point clouds, as in our case, the above criterion cannot be immediately used, mainly due to the typically non-uniform density of skeleton point clouds (see Sec. III-A). Therefore, we propose to extract curve skeletons by a method akin to the technique of Siddiqi *et al.* [60], [61]. Our proposal is based on the following observation: In a small vicinity $\mathcal{N}$ of a curve-skeleton point, geodesic tangents, mapped to $\mathcal{N}$, abruptly change directions (Fig. 11). Given this observation, we extract curve-skeleton points by looking for high-response points of weight-averaged tangent directions in vicinities around each surface-skeleton point, by a three-step method: (i) find candidates close to the curve skeleton; (ii) filter and regularize candidates; and (iii) reconstruct the curve skeleton.

A. Detecting candidate curve skeleton points

For each surface skeleton point $\mathbf{s}_i$, we compute the average (projected) tangent direction of its SSG path, *i.e.*,

$$\begin{aligned} \mathbf{t}'_{s,i} &= \mathbf{t}_{s,i} - \mathbf{f}_i(\mathbf{f}_i \cdot \mathbf{t}_{s,i}) \\ \mathbf{t}'_{e,i} &= \mathbf{t}_{e,i} - \mathbf{f}_i(\mathbf{f}_i \cdot \mathbf{t}_{e,i}) \\ \mathbf{t}_i &= \frac{\mathbf{t}'_{s,i} + \mathbf{t}'_{e,i}}{\|\mathbf{t}'_{s,i} + \mathbf{t}'_{e,i}\|}, \end{aligned} \tag{4}$$

where $\mathbf{t}_{s,i}$, $\mathbf{t}_{e,i}$ are the tangent vectors at its feature points, and $\mathbf{f}_i$ is the normalized feature vector of $\mathbf{s}_i$. Projection improves the detector reliability (see below) close to Y-intersection curves, *i.e.*, where several skeletal manifolds intersect [18].

Tangent vectors $\mathbf{t}_i$ span a vector field $\mathbf{T}$ over the surface skeleton (Fig. 11). One can evaluate the divergence of $\mathbf{T}$ or its more numerically-stable flux [60]. Points close to the curve skeleton have large positive flux/divergence values. However, along with these, this approach may yield also points close to the Y-intersection curves.

We find candidate curve-skeleton points differently. For each skeleton point $\mathbf{s}_i$, let $\mathcal{N}_i$ be its set of neighbor skeleton points (we use 10 nearest-neighbors in practice). We measure the likelihood of $\mathbf{s}_i$ to be a curve-skeleton point as

$$I(\mathbf{s}_i) = \rho_i - \rho_i \left\| \frac{\sum_{j\in\mathcal{N}_i} w_{ij}\mathbf{t}_j}{\sum_{j\in\mathcal{N}_i} w_{ij}} \right\|, \; w_{ij} = |\mathbf{f}_i \cdot \mathbf{f}_j|\, e^{-\frac{(\rho_i-\rho_j)^2}{2\sigma_1^2} - \frac{\|\mathbf{s}_i-\mathbf{s}_j\|^2}{2\sigma^2}}$$

with $\rho_i \equiv \rho(\mathbf{s}_i)$. The importance $I_i \equiv I(\mathbf{s}_i)$ averages tangent vectors in $\mathcal{N}_i$ with weights given by a 2D Gaussian kernel and σ_1, σ_2 set to the median of the distances $\|\mathbf{s}_i - \mathbf{s}_j\|_{j\in\mathcal{N}_i}$. The weight w_{ij} lowers the impact of tangent vectors $\mathbf{t}_j$ of skeleton points $\mathbf{s}_j$ which: (i) have feature vectors $\mathbf{f}_j$ not parallel to $\mathbf{f}_i$, (ii) have geodesics of different lengths (following [20], [55]), and (iii) are far from $\mathbf{s}_i$. Points close to Y-intersection curves meet conditions (i) and (ii), so they contribute weakly to I. In contrast, points close to the curve skeleton yield large weights, and have tangent vectors pointing outwards in all directions (see Fig. 11). Such points have large I values, so we find the set $\mathscr{C}$ of candidate curve-skeleton points by thresholding I at a small value $T_I > 0$.

B. Regularization of candidate curve-skeleton points

We assign an importance μ_i to each point $\mathbf{s}_i \in \mathscr{C}$ to prune spurious curve skeleton details. We set μ_i to the smallest surface area between two SSGs of *nearby* skeleton points. This is similar to the metric in [55] which was computed by a flood-fill on a voxel surface. Our case is more complicated as we have two curves on an unstructured *mesh*. We efficiently *approximate* this area using only the angle between the two geodesics and the lengths of a few additional straightest geodesics, as follows.

For each candidate $\mathbf{s}_i \in \mathscr{C}$, we find a neighbor $j^\star \in \mathscr{M}_i$, with $\mathscr{M}_i$ a neighborhood of $\mathbf{s}_i$ (10 nearest-neighbors), for which

$$J_j = (1 + \mathbf{t}_i \cdot \mathbf{t}_j)\, e^{-\frac{(I_i-I_j)^2}{2\sigma_1^2} - \frac{\|\mathbf{s}_i-\mathbf{s}_j\|^2}{2\sigma_2^2}} \tag{5}$$

is minimal, *i.e.*, $j^\star = \operatorname{argmin}_{j\in\mathscr{M}_i} J_j$. Since $\mathbf{s}_i$ and $\mathbf{s}_{j^\star}$ are spatially close, we assume that their feature points coincide, and use $\mathbf{s}_i$ for these. Let $\theta = \angle(\mathbf{t}_i, \mathbf{t}_{j^\star}) \in [0, \pi]$ be the angle between the tangent vectors of $\mathbf{s}_i$ and $\mathbf{s}_{j^\star}$. The pair of SSGs $\gamma_{se,i}$ and $\gamma_{se,j^\star}$ between the feature points $\mathbf{f}_{1,i}$ and $\mathbf{f}_{2,i}$ divide the surface $\partial\Omega$ in two parts, the smallest area of which we want to estimate. For this, we trace P straightest geodesics $\gamma_{S,i,k}$, $1 \le k \le P$ from $\mathbf{f}_{1,i}$ to $\mathbf{f}_{2,i}$ on $\partial\Omega$, with uniformly spread starting angles $\alpha_k = 2\pi k/P$ around the vector $\mathbf{f}_i = \mathbf{f}_{1,i} - \mathbf{f}_{2,i}$ at $\mathbf{f}_{1,i}$. Assuming that each geodesic is half of an *ellipse*, with minor axis $\mathbf{f}_i$, the ellipse radii a_k and $b_k \ge a_k$ are given by a simple approximation formula for an ellipse perimeter, *i.e.*, $a_k = \|\mathbf{f}_i\|/2$ and $b_k = (2\|\gamma_{S,i,k}\| - \pi a_k)/2$. Next, we approximate $\partial\Omega$ between two consecutive geodesics by an *oblate spheroid* with radii a_k and $c_k = (b_k + b_{k+1})/2$, so its area is that of an oblate spheroid wedge with angle $\beta = 2\pi/P$, *i.e.*,

$$S_k = \beta c_k^2 + \frac{\beta a_k^2}{2e} \ln \frac{1+e}{1-e}, \tag{6}$$

with $e = \sqrt{1 - a_k^2/c_k^2}$. Assuming that the starting direction $\alpha_0 = 0$ corresponds to the tangent of $\gamma_{se,i}$, we compute μ_i as

$$\mu_i = \min\left(\sum_{k,\alpha_k<\theta} S_k, \sum_{k,\alpha_k\ge\theta} S_k \right). \tag{7}$$

Thresholding μ removes short curve-skeleton branches to yield the final candidate set $\mathscr{C}'$. We use more paths $P = 50$ than for the surface skeleton metric ($M = 20$, Sec. V) to limit area estimation errors. σ_1 and σ_2 are set to the median distance in $\mathscr{M}_i$.

C. Curve skeleton reconstruction

To get the final curve skeleton CS, we connect points in $\mathscr{C}'$ by line segments, by adapting the ball-pivoting method [8]. We start from the point with largest importance $\max_{\mathscr{C}'}(\mu)$, find its neighbor in $\mathscr{C}'$ within a radius r with largest μ value, and add a new line segment to CS. Next, we try to extend CS by searching neighbors of its end vertices $\mathbf{e}_i$. To become a new end vertex, a point $\mathbf{x}$ must (i) be within distance r from an $\mathbf{e}_i$;

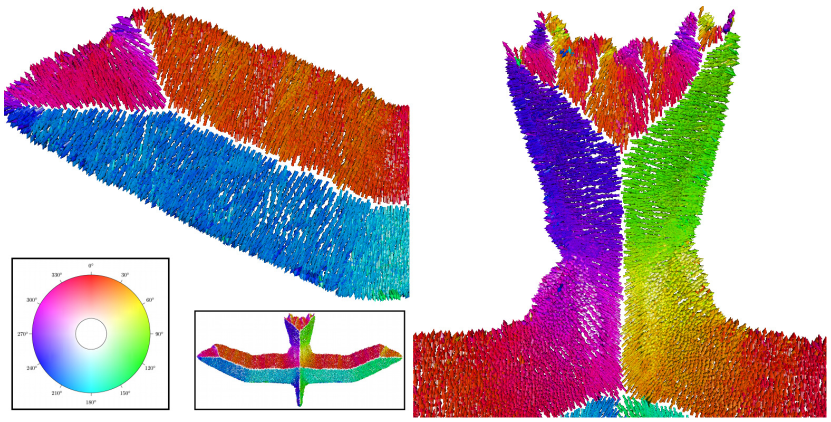

Fig. 11. Tangent vector field **T** (shown with directional color-coding)

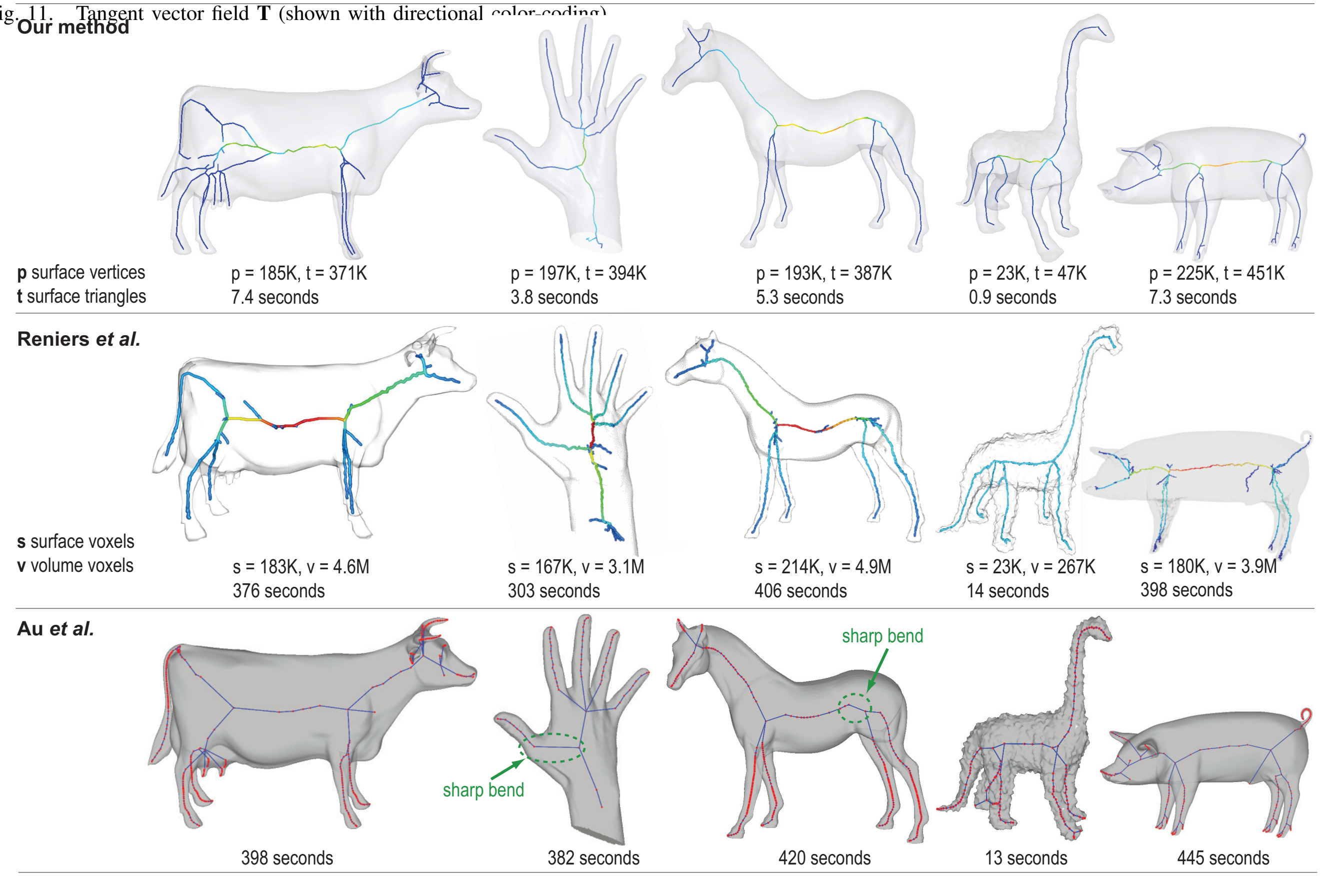

Fig. 12. Curve skeleton extraction: our method (top row), voxel-based method [55] (middle row), and mesh collapse method [5] (bottom row).

and (ii) the segment $(\mathbf{x}, \mathbf{e}_i)$ must be well-aligned with the curve tangent. When CS cannot be extended, we backtrack and try to extend from vertices of previous segments. This captures the curve skeleton branching.

D. Comparison

Regularization (Sec. VIII-B) is the costliest step of our curve-skeleton extraction. $\mathscr{C}$ is a small subset of the surface skeleton, but tracing $P = 50$ straightest geodesics $\gamma_{S,i,k}$ per point $\mathbf{s}_i \in \mathscr{C}$ is still expensive, as no early termination can be used (Sec. V). Still, we only need to compute geodesic lengths. This results in a very efficient CUDA mapping (see Tab. VI).

Figure 12 compares our results with Reniers *et al.* [55] and with of Au *et al.* who compute curve-skeletons by shape collapse via Laplacian smoothing [5]. Although our point count is smaller than, or at most equal to, the surface voxel count of Reniers *et al.*, we find more skeleton branches, *e.g.* the cow udder and horns. Our skeletons, unlike Au *et al.*, do not have artificial straight-line branches and sharp bends. Au *et al.* added these in a 'surgery' step to connect disjoint skeleton parts (see green markers, hand and horse model, Fig. 12 bottom). The skeletons of Au *et al.* extend deeper into surface cusps, *e.g.* the pig's hooves and snout. Such branches are shortened by our regularization (Sec. VIII-B). Our method is on average 50 times faster than Au *et al.* and over one order of magnitude faster than Reniers *et al.* Interestingly, the costs of the latter two are similar, since both methods 'visit' the

TABLE VI
CURVE-SKELETON EXTRACTION TIMINGS.

Model	Detect (seconds)	Regularize (seconds)	Reconstruct (seconds)	Total (seconds)
Cow	3.1	3.5	0.8	7.4
Bird	0.5	1.2	0.1	1.8
Horse	2.5	2.6	0.2	5.3
Asiandragon	2.6	3.7	1.7	8.0
Hand	1.3	1.9	0.6	3.8
Elephant	0.5	1.5	0.5	2.5
Buddha	3.3	5.3	0.8	9.4
Mouse	3.0	4.6	0.9	8.5
Dragon	3.5	4.7	0.9	9.1
Pig	2.5	4.0	0.8	7.3
Armadillo	1.8	1.5	0.6	3.9
Rabbit	0.5	1.1	0.5	2.1

entire input volume: Au *et al.* while collapsing the mesh, and Reniers *et al.* while computing its voxel skeleton detectors.

E. Relation to surface skeletons

As stated at the beginning of Sec. VIII, we detect curve skeleton points as a subset of surface skeleton points, so our curve skeleton is always a subset of the corresponding surface skeleton point-cloud. Both curve and surface skeletons can be simplified (regularized), and the relation between these two regularization types is as follows. First, we can regularize a surface skeleton, using the SSG-length importance metric ρ (Sec. V), and then compute the curve skeleton of the regularized surface skeleton. This approach, however, may eliminate entire curve-skeleton terminal branches which correspond to thin and narrow shape parts, such as the pig's tail or cow's horns in Fig. 12, since the surface skeleton has a low importance in such regions. Alternatively, we can extract the curve skeleton from the full surface skeleton, and regularize the former by thresholding its own importance μ (Eqn. 7). This has the advantage of simplifying curve-skeleton terminal branches more uniformly.

A mixed curve-and-surface skeleton can be easily extracted too. For this, we replace the importance ρ of the surface skeleton points which have been detected to be also on the curve skeleton by the corresponding curve skeleton importance μ. Note that, for all curve skeleton points $\mathbf{p} \in CS$, $\mu(\mathbf{p}) > \rho(\mathbf{p})$, since the former measures an area whose boundary length is the latter. Given this, and as already shown for the voxel case in [55], thresholding ρ with increasing values will now deliver a mixed curve-and-surface skeleton, where the surface skeleton is first progressively simplified towards the curve skeleton, followed by the simplification of the higher-importance curve skeleton.

IX. DISCUSSION

We next discuss our framework *vs* several related methods.

Input: We use (non-uniform) mesh models instead of voxel models. Our input can be of any genus (see the rabbit, cat, and dragon models), self-intersecting (cow model), and non-closed (hand model). Skeleton extraction and shape rendering only require an oriented point cloud input. We need connectivity data only for the regularization and Delaunay-based reconstruction.

Accuracy: Voxel-based skeletons are limited by the voxel resolution [11], [31], [55]. Like Stolpner *et al.* [64], our skeletons are point clouds close to the true medial axis within a user-prescribed precision in world space. Our image-based shape reconstruction from its (simplified) skeleton is real-time and near-pixel accurate.

Scalability: A 1024^3 distance-and-feature-transform volume needs at least 4 GB RAM [11], [30], [70]. An equivalent mesh, roughly 1 M triangles, needs only 24 MB RAM, which is essential for typical 1 GB GPU RAM limits. Voxelization has also large speed costs and is delicate for certain meshes [23], [24], [46]. Multiresolution voxel schemes reduce memory costs but complicate algorithms and reduce GPU speedups. Table I (Sec. III) ($\tau = 10^{-3}$, $\varepsilon = 0$, equivalent to a 1024^3 volume) shows that our method is over 100 times faster than [10], [20], [37], [43], [55], [64], even without voxelization.

Geodesic computation: Our GPU computation of shortest, straightest geodesics (SSGs) is over two orders of magnitude faster, and more accurate, than state-of-the-art techniques [49], [68], [73]. This makes global skeleton regularization practical for large models. Our SSG method is also usable for other applications requiring fast, near-exact, geodesics on meshes.

Simplicity: Our framework has no complex computational geometry operations or degenerate cases, unlike [43], [55], [64]. Its only user parameters are the skeleton centeredness τ and number of geodesic directions M, explained in Secs. III and V.

Curve skeletons: Existing *curve* skeleton extractors have widely different speed, accuracy, and curve skeleton definitions [5], [14], [15], [20], [30]. Our curve skeletons cannot replace all such methods. Our main novelty is the fast extraction of curve skeletons from a *surface* skeleton cloud. The closest methods to ours are the medial geodesic function (MGF) [20] and ROSA [69]. Yet, we use a different angle-based criterion than MGF and also than ROSA which computes curve skeletons as centers of point cloud projections on a cut plane found by optimizing for circularity. We are two orders of magnitude faster than ROSA and MGF, and on average 50 times faster than Au *et al.* [5] (Sec. VIII-C). Compared to the variational method of Hassouna *et al.* [30], we are 20 times faster (Fig. 12, Tab. VI *vs* Fig. 10 in [30]).

X. CONCLUSION

We have presented a GPU-based framework for extracting surface and curve skeletons from large meshes. Using GPUs to parallelize several extraction steps (maximal ball computation, regularization, and surface reconstruction), we obtain similar or higher-quality surface and curve skeletons with two orders of magnitude speed-up over state-of-the-art methods. Skeleton clouds, extracted with user-desired accuracy, are regularized by a new parallel computation of shortest, straightest geodesics which is over two orders of magnitude faster, and more accurate, than similar schemes. From the skeleton cloud, we reconstruct the input shapes in real-time by an image-based method or extract high-quality mesh and curve skeletons for further use in applications such as shape analysis, classification, and matching.

References

[1] N. Ahuja and J. Chuang. Shape representation using a generalized potential field model. *IEEE TPAMI*, 19(2):169–176, 1997.

[2] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *JCGA*, 12:125–141, 2002.

[3] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *Proc. SMA*, pages 65–73. ACM, 2001.

[4] C. Aslan, A. Erdem, E. Erdem, and S. Tari. Disconnected skeleton: Shape at its absolute scale. *IEEE TPAMI*, 30(12):2188–2203, 2008.

[5] O. K. C. Au, C. Tai, H. Chu, D. Cohen-Or, and T. Lee. Skeleton extraction by mesh contraction. In *Proc. ACM SIGGRAPH*, pages 441–449, 2008.

[6] X. Bai and L. Latecki. Path similarity skeleton graph matching. *IEEE TPAMI*, 30(7):1282–1292, 2008.

[7] X. Bai, L. Latecki, and W.-Y. Liu. Skeleton pruning by contour partitioning with discrete curve evolution. *IEEE TPAMI*, 3(29):449–462, 2007.

[8] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE TVCG*, 5(4):349–359, 1999.

[9] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven automatic centerline extraction. *Medical Image Analysis*, 9(3):209–221, 2005.

[10] S. Bouix, K. Siddiqi, A. Tannenbaum, and S. Zucker. Medial axis computation and evolution. In *Statistics and analysis of shape*, chapter 1, pages 1–28. 2006.

[11] T. Cao, K. Tang, A. Mohamed, and T. Tan. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. SIGGRAPH I3D Symp.*, pages 134–141, 2010.

[12] L. Cayton. A nearest neighbor data structure for graphics hardware. In *Proc. ADMS*, pages 192–197, 2010. `people.kyb.tuebingen.mpg.de/lcayton`.

[13] M. Chang, F. Leymarie, and B. Kimia. Surface reconstruction from point clouds by transforming the medial scaffold. *CVIU*, (113):1130–1146, 2009.

[14] J. Chuang, C. Tsai, and M. Ko. Skeletonization of three-dimensional object using generalized potential field. *IEEE TPAMI*, 22(11):1241–1251, 2000.

[15] N. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE TVCG*, 13(3):87–95, 2007.

[16] N. Cornea, D. Silver, X. Yuan, and R. Balasubramanian. Computing hierarchical curve-skeletons of 3D objects. *Visual Comput.*, 21(11):945–955, 2005.

[17] L. Costa and R. Cesar. *Shape analysis and classification*. CRC Press, 2000.

[18] J. Damon. Global medial structure of regions in $\mathbf{R}^3$. *Geometry and Topology*, 10:2385–2429, 2006.

[19] T. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *IJCGA*, 35(1):340–355, 2006.

[20] T. Dey and J. Sun. Defining and computing curve skeletons with medial geodesic functions. In *Proc. SGP*, pages 143–152. IEEE, 2006.

[21] T. Dey and W. Zhao. Approximating the medial axis from the Voronoi diagram with a convergence guarantee. *Algorithmica*, 38:179–200, 2003.

[22] R. Dougherty and K. Kunzelmann. Computing local thickness of 3D structures with ImageJ. *Microscopy and Microanalysis*, (12):1678–1679, 2007. `www.optinav.com/LocalThicknessEd.pdf`.

[23] E. Eisemann and X. Decoret. Fast scene voxelization and applications. In *Proc. SIGGRAPH I3D Symp.*, pages 71–78, 2006.

[24] M. Foskey, M. Lin, and D. Manocha. Efficient computation of a simplified medial axis. In *Proc. Shape Modeling*, pages 135–142, 2003.

[25] P. Frey. YAMS: a fully automatic adaptive isotropic surface remeshing procedure. tech. rep. 0252, INRIA, Nov. 2001. `www.ann.jussieu.fr/~frey`.

[26] V. Garcia, E. Debreuve, and M. Barlaud. Fast *k* nearest neighbor search using GPU. In *Proc. CVGPU*, pages 77–83, 2008.

[27] Y. Ge and J. Fitzpatrick. On the generation of skeletons from discrete euclidean distance maps. *IEEE TPAMI*, 18:1055–1066, 1996.

[28] P. Giblin and B. Kimia. A formal classification of 3D medial axis points and their local geometry. *IEEE TPAMI*, 26(2):238–251, 2004.

[29] J. Giesen, B. Miklos, M. Pauly, and C. Wormser. The scale axis transform. In *Proc. Annual Symp. Comp. Geom.*, pages 106–115, 2009.

[30] M. Hassouna and A. Farag. Variational curve skeletons using gradient vector flow. *IEEE TPAMI*, 31(12):2257–2274, 2009.

[31] W. Hesselink and J. Roerdink. Euclidean skeletons of digiral image and volume data in linear time by the integer medial axis transform. *IEEE TPAMI*, 30(12):2204–2217, 2008.

[32] I. Hotz and H. Hagen. Visualizing geodesics. In *Proc. IEEE Visualization*, pages 311–318, 2000.

[33] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM TOG*, 22(3):954–961, 2003.

[34] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proc. SGP*, pages 61–70, 2006.

[35] R. Kimmel, D. Shaked, N. Kiryati, and A. Bruckstein. Skeletonization via Distance Maps and Level Sets. *Computer Vision and Image Understanding: CVIU*, 62(3):382–391, 1995.

[36] J. Lambourne, D. Brujic, Z. Djuric, and M. Ristic. Calculation and visualisation of the thickness of 3D CAD models. In *Proc. SMI*, pages 107–112, 2005.

[37] F. Leymarie and B. Kimia. The medial scaffold of 3d unorganized point clouds. *IEEE TVCG*, 29(2):313–330, 2007.

[38] F. Leymarie and M. Levine. Simulating the grassfire transform using an active contour model. *IEEE TPAMI*, 14(1):56–75, jan 1992.

[39] X. Li, T. Woon, T. Tan, and Z. Huang. Decomposing polygon meshes for interactive applications. In *Proc. I3D Symp.*, pages 35–42, 2001.

[40] J. Ma, S. W. Bae, and S. Choi. 3D medial axis point approximation using nearest neighbors and the normal field. *Visual Comput.*, 28(1):7–19, 2012.

[41] G. Malandain and S. Fernandez-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16(5):317–327, 1998.

[42] MeshLab. MeshLab geometry processing software, 2012. `meshlab.sourceforge.net`.

[43] B. Miklos, J. Giesen, and M. Pauly. Discrete scale axis representations for 3D geometry. In *Proc. ACM SIGGRAPH*, pages 394–493, 2010.

[44] M. Mortara, G. Patanet, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: A method for multiscale decomposition of 3D shapes into tubular primitives and bodies. In *Proc. ACM SMA*, pages 339–344, 2004.

[45] D. Mount and S. Arya. Approximate nearest neighbor search software. 2011. `www.cs.umd.edu/~mount/ANN`.

[46] F. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE TVCG*, 9(2):191–205, 2003. see also `www.cs.princeton.edu/~min/binvox`.

[47] R. L. Ogniewicz and O. Kubler. Hierarchic Voronoi skeletons. *Pattern Recognition*, (28):343–359, 1995.

[48] K. Palagyi and A. Kuba. Directional 3D thinning using 8 subiterations. In *Proc. DGCI*, volume 1568, pages 325–336. Springer LNCS, 1999.

[49] G. Peyre and L. Cohen. Geodesic computations for fast and accurate surface remeshing and parameterization. In *Progress in Nonlinear Differential Equations and Their Applications*, volume 63, pages 151–171. Springer LNCS, 2005. `www.ceremade.dauphine.fr/~peyre`.

[50] S. Pizer, K. Siddiqi, G. Szekely, J. Damon, and S. Zucker. Multiscale medial loci and their properties. *IJCV*, 55(2-3):155–179, 2003.

[51] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *ACM SIGGRAPH Courses*, pages 30–38, 2006.

[52] S. Prohaska and H. C. Hege. Fast visualization of plane-like structures in voxel data. In *Proc. IEEE Visualization*, page 2936, 2002.

[53] C. Pudney. Distance-ordered homotopic thinning: A skeletonization algorithm for 3D digital images. *CVIU*, 72(3):404–413, 1998.

[54] D. Reniers and A. Telea. Part-type segmentation of articulated voxel-shapes using the junction rule. *CGF*, 27(7):1837–1844, 2008.

[55] D. Reniers, J. J. van Wijk, and A. Telea. Computing multiscale skeletons of genus 0 objects using a global importance measure. *IEEE TVCG*, 14(2):355–368, 2008.

[56] M. Rumpf and A. Telea. A continuous skeletonization method based on level sets. In *Proc. VisSym*, pages 151–158, 2002.

[57] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH*, pages 230–237, 2000.

[58] D. Shaked and A. Bruckstein. Pruning medial axes. *CVIU*, 69(2):156–169, 1998.

[59] J. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, pages 203–222. Springer LLNC, 1996.

[60] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. Hamilton-Jacobi skeletons. *IJCV*, 48(3):215–231, 2002.

[61] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. W. Zucker. The Hamilton-Jacobi skeleton. In *Proc. of the International Conference on Computer Vision - Volume 2*, ICCV '99, pages 828–, Washington, DC, USA, 1999. IEEE Computer Society.

[62] K. Siddiqi and S. Pizer. *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 2009.

[63] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci and boundary differential geometry. In *Proc. IEEE 3DIM*, pages 87–95, 2009.

[64] S. Stolpner, S. Whitesides, and K. Siddiqi. Sampled medial loci for 3D shape representation. *CVIU*, 115(5):695–706, 2011.

[65] R. Strzodka and A. Telea. Generalized distance transforms and skeletons in graphics hardware. In *Proc. VisSym*, pages 221–230, 2004.

[66] A. Sud. *Efficient computation of discrete Voronoi diagram and homotopy-preserving simplified medial axis of a 3D polyhedron*. PhD thesis, UNC Chapel Hill, 2006.

[67] A. Sud, M. Foskey, and D. Manocha. Homotopy-preserving medial axis simplification. In *Proc. SPM*, pages 103–110, 2005.

[68] V. Surazhsky, T. Surazshky, D. Kirsanov, S. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *Proc. ACM SIGGRAPH*, pages 130–138, 2005.

[69] A. Tagliasacchi, H. Zhang, and D. Cohen-Or. Curve skeleton extraction from incomplete point cloud. In *Proc. SIGGRAPH*, pages 541–550, 2009.

[70] A. Telea and A. Jalba. Voxel-based assessment of printability of 3D shapes. In *Proc. ISMM*, pages 393–404. Springer LNCS, 2011.

[71] A. Telea and J. J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym*, pages 251–259, 2002.

[72] M. van Dortmont, H. van de Wetering, and A. Telea. Skeletonization and distance transforms of 3D volumes using graphics hardware. In *Proc. DGCI*, pages 617–629. Springer LNCS, 2006.

[73] V. Verma and J. Snoeyink. Reducing the memory required to find a geodesic shortest path on a large mesh. In *Proc. ACM GIS*, pages 227–235, 2009.

[74] M. Wan, F. Dachille, and A. Kaufman. Distance-field based skeletons for virtual navigation. In *Proc. IEEE Visualization*, pages 239–246, 2001.

[75] J. Wang, M. Oliveira, and A. Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. In *Proc. SMA*, pages 139–147, 2007.

[76] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time KD-tree construction on graphics hardware. *ACM TOG*, 27(5):111, 2008.

Andrei C. Jalba received his B.Sc. (1998) and M.Sc. (1999) in Applied Electronics and Information Engineering from "Politehnica" University of Bucharest, Romania. He obtained a Ph.D. degree at the Institute for Mathematics and Computing Science of the University of Groningen, the Netherlands, in 2004. Currently he is assistant professor at the Eindhoven University of Technology. His research interests include computer graphics and vision, shape processing and parallel computing.

Jacek Kustra received his Licenciatura degree in Electronics and Telecommunications and M.Sc. in Biomedical Engineering from the University of Aveiro, Portugal, in 2004 and 2008 respectively. In parallel to his studies he worked as software development freelancer. He is employed as a Scientist in Philips Research and is working towards a PhD degree at the University of Groningen. His research interests are in shape and surface processing with applications towards healthcare systems.

Alexandru C. Telea received his PhD (2000) in Computer Science from the Eindhoven University of Technology, the Netherlands. Until 2007, he was assistant professor in visualization and computer graphics at the same university. Since 2007, he is professor of computer science at the University of Groningen, the Netherlands. His interests include 3D multiscale shape processing, scientific and information visualization, and software analytics.