STAG: Smart Tools and Applications in Graphics (2023)
K. Lupinetti, F. Banterle, N. Capece, and U. Erra (Editors)

User-Assisted Sphere-Mesh Construction

Davide Paolillo, Andrea Taroni, and Marco Tarini

University of Milan (Italy)

Abstract

In this study, we investigate the semi-automated generation of sphere-meshes as high-quality approximations for given three-dimensional shapes, originally represented as common triangular meshes. A sphere-mesh is a class of geometric proxy defined as the volume swept by spheres with linearly interpolated centers and radii, that potentially strikes a good balance between conciseness of representation, simplicity of spatial queries, and expressive power, and is amenable to animations. Despite these favorable characteristics, its broader adoption in applications such as video games, physical simulation, or robotics is hindered by the difficulty of its construction, which remains an open problem. Existing fully automatic algorithms, based on interactive coarsening of the input mesh, fail to consistently produce satisfactory results, especially when very coarse sphere-meshes are sought. We improve on this situation with a 3D interface specifically designed to permit users to easily and intuitively modify the automatically generated models. The two phases (existing automatic algorithm and novel interactive tool), used in cascade, constitute a viable semi-automatic way to produce sphere-meshes. We test our method on several inputs tri-meshes, assess their quality, and finally experiment with a few downstream applications to exemplify the usability of our results.

1. Introduction

In general terms, a *geometric proxy* is an approximation of a given solid 3D shape that can be employed in contexts such as physical simulations (especially collision detection and response), video games, robotics, vision, and others. Several types of geometric proxies have been proposed, striving to strike a good balance between expressive power, allowing for more faithful representations of a wider class of 3D shapes, and simplicity, allowing for coarse and compact representations. Simplicity implies that operations such as intersection testing with other proxies (of the same or different kinds), enforcing non-compenetration, or intersection testing with lines (e.g. for raycasting) are efficient.

A particularly appealing type of proxy is the *sphere-mesh* [TGB13]. A sphere-mesh is described as a collection of spheres S, and a collection T of segments and triangles connecting their centers. This defines a volume as the union of all the spheres having the center anywhere in the elements of T, and having the radius linearly interpolated between the radii of S (see Figure 1).

This schema exhibits advantageous qualities in both its expressive capacity and ease of representation, being potentially able to represent a variety of smooth shapes (convex or concave alike) with a small number of primitives. In addition, sphere-meshes can be easily animated, by changing sphere centers (and radii) over time. Sphere-meshes have been successfully employed, for example, in the context of image-based hand tracking [AMA16] and animation [JÉTE16]. On a sphere-mesh, overlap queries, and other similar tasks are made simpler by the observation that only one interpolated sphere is affected at a time, reducing the problem to the identification of that sphere [TGB13].

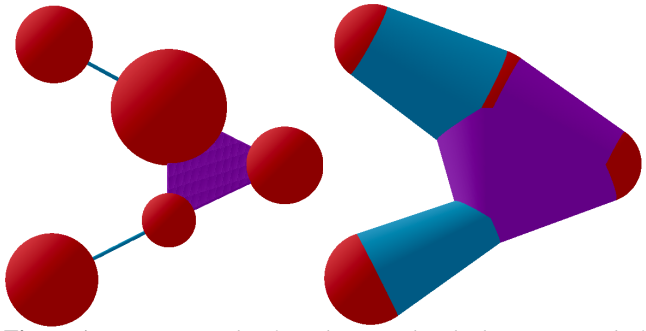

Figure 1: *A toy-example of a sphere-mesh, which is composed of $|S| = 5$ spheres, and $|T| = 3$ connecting elements (one triangle and two segments). Left: the connectivity structure. Right: the represented solid shape.*

Despite these favorable characteristics, sphere-meshes are not currently widely used, for example, in video games. We conjecture that this is because of the difficulties of their construction, with existing automatic solutions failing to produce sufficiently good instances, especially when low resolutions are sought (the resolution of a sphere-mesh being defined as the number of its elements).

Overview and Contributions. In this work, we revisit the only currently existing heuristic for fully automatic sphere-mesh construction [TGB13], which we recap for completeness in (Section

DOI: 10.2312/stag.20231303

3), and re-implement a close variation. We identify a set of intrinsic limitations of this heuristic that can result in the construction of sub-optimal, and typically too complex, sphere-meshes, hindering the potentiality of sphere-meshes as a representation.

Then, to address this issue, we propose an intuitive interactive tool that allows users to manually adjust a pre-existing sphere-mesh (Section 4), intended to be used on automatically generated results. This tool embeds a GUI to allow users to trigger additional coarsening operations (which include a fitting phase, just as in the automatic approach), tweak individual positions and radii, and so on. It also uses a real-time preview renderer able to deal with frequently changing sphere-meshes (Section 4.1)

This tool has two distinct motivations: first, there is a practical value in the ability to produce better and coarser sphere-meshes, ready for downstream applications (we exemplify and test a few such applications within the conclusions, in Section 7). Secondly, this tool is intended to help determine the potentiality of sphere-meshes as a shape representation, independently of any present or future automatic construction method.

We evaluate the semi-automated system by converting multiple real-world geometric structures into high-quality sphere-meshes (Section 6). To assess the results, we use and report the quality of a sphere-mesh approximation, which was computed as illustrated in Section 5. Finally, we field-test our result in a few demonstrative downstream applications (Section 7).

2. State of The Art

We provide a brief overview of the current state of geometric proxies, with a focus on the most relevant approaches.

Geometric Proxies and their construction. An analysis of literature concerning strictures similar to sphere-meshes as geometric proxies is beyond the scope of this paper, and we refer to the survey contained in [TGB13]. Here, we only refer to the most relevant parts.

Their closest relative of sphere-meshes is the Medial Axis Transform (MAT), a deeply studied subject [H67; NSR01; TW04; FA05; AMD07; BJM10; TGB13]. Unlike MAT, which aims to capture the symmetric skeleton of an object using its interior points, a sphere-mesh approach focuses on approximating the object's volume or boundary using a collection of spheres. While MAT serves as a tool for understanding the geometric and topological properties of a shape, sphere-meshes are geared toward various tasks, like collision detection, and animations.

Polygonal mesh simplification. A popular and simple kind of geometric proxy is, clearly, a coarse polygonal mesh. The construction of such coarse polygonal mesh falls in the domain of surface simplification (see [Lue01] for a survey), which is relevant to our work because the construction algorithm we employ stems from a modification of a polygonal simplification tool. Even the most aggressively coarsened polygonal meshes, however, have a primitive count that is much higher than what would be desirable for a proxy mesh (although advancements are being made in this area [CPW*23]). As we will discuss, sphere-mesh construction partially inherits this limitation, motivating us to resort to an interactive tool to improve sphere-meshes.

Manual editing of geometric proxies. Many suites for video games, such as Unity, offer tools to edit or tweak collision proxies of certain kinds (such as a collection of spheres or capsules); they, however, fail to include sphere-meshes. A much wider class of approaches and industrial products of approach, intended for professional modelers, targets authoring and detailed editing of complex 3D shapes represented as traditional boundary representations, which are not easily adaptable to sphere-mesh editing. Sketch-based approaches (see the survey in [ST14]) are more relevant, in that a higher-level interaction is leveraged.

In particular, a sphere-mesh can be considered a special case of an implicit surface; recently, interactive tools for manual editing implicit surfaces have been proposed [ATW*17], but focus primarily on customizing the blend of primitives which, in the case of sphere-meshes, is defined as the simple union. Another similarity is with [BMUS15], which offers an interactive tool to model curve skeletons (a data structure that can be seen to resemble sphere-meshes, but with curved segments and no triangle).

In short, no tools exist, to our knowledge, that allows direct interactive manipulation of a sphere-mesh.

3. Automatic Construction Phase

Given as input a triangle mesh (V,F) (with V is a set of vertices, F of triangular faces, each defined as a triplet of indices to V), we want to automatically convert it into a sphere-mesh (S,T). To do so we leverage the technique presented in [TGB13], which we briefly summarize here for completeness only. For further detail, the reader is referred to the original article.

In our work, the triangle mesh is assumed to be very coarse, for triangular mesh standards (typically, in order of hundreds or thousands of elements), as the approximation introduced by the proxy representation can be expected to be substantially greater than the one introduced by the loss of polygonal resolution.

3.1. Existing algorithm

The basic idea of [TGB13] is to consider the initial mesh as a starting sphere-mesh, with one sphere in S for each vertex in V, and one triangular element in T for each triangle in F. Then, in an iterative local coarsening approach, similar to what is done in a well-studied class of polygonal-mesh simplification, pairs of connected spheres are collapsed into one, until a sufficiently coarse structure is reached.

The main difference with the traditional case of polygonal meshes is that collapses are allowed, and expected, to break the 2-manifoldness of the structure, potentially producing segments and fusing surfaces facing opposite directions into a single sphere-mesh element (for example, the side surface of a cylinder can be collapsed into a single segment, and the surface a pizza-box shaped mesh can be collapsed into a single pair of triangles).

This is done by disregarding the traditional conditions guaranteeing the preservation of the manifoldness of the meshing [DEGN99],

and even by allowing the collapses of pairs of spheres that are not initially connected by a segment or a triangle in T.

After [TGB13], only spheres with centers at a distance closer than a maximal distance value d are considered for collapse. The value d is defined as a parameter ε times the axis-aligned bounding-box diagonal of the input mesh (see Table 1 for the values used for ε).

The effect in T of the collapse of spheres s_i and s_j in S to simply remove (if it exists) the segment in T that connected these two spheres, and reducing any triangle in T that connected these two spheres into a segment; followed by the removal of this segment if it is already a side of any triangle in T, and the removal of any duplicate elements in T.

Just like in the case of triangular meshes, the order of the collapses is crucial, and so is the determination of the sphere (both center and radius) that results from the fusion of two spheres. Both matters are addressed by resorting to quadrics, after the successful idea by [MP97] for triangular mesh simplification. To summarize: each element s_i in S is associated with an error quadric q_i that computes, for each possible choice of center and radius, the sum of the squared distances from all the original planes that should be represented in the sphere-mesh by s_i. When two spheres are collapsed, their quadrics are summed to form the quadric q_i of the merged sphere s_i. That sphere is assigned to a center and a radius that are the minimizers of q_i. Potential collapses are associated the the quadric they would produce if executed, and prioritized in the inverse order of the minimal values of these quadrics. Concerning the familiar polygonal case of [MP97], the difference is that the quadric is a degree-2 function not only of the position but also of the radius of the sphere.

As an implementation note, in our code, the identification of the best potential operation is realized with a priority queue that contains all potential operations. To make this efficient, we use a Fibonacci-Queue [BenAR], which is based on the Fibonacci Heap [HELC09].

3.2. Analysis of the limitations

Despite its relative success, an analysis of the heuristic sketched above (and described more in-depth in [TGB13]) reveals that it suffers from a few intrinsic shortcomings.

First, it considers the original oriented points of the mesh to be approximated by only the spheres, disregarding the fact that much of the surface of the sphere-mesh is not that of any sphere in S, but rather of their interpolation (as defined by the connectivity T). This is disregarded by the construction heuristics, which just strives to make all of the initial surface approximated by the spheres.

Another problem is that the connectivity T is not explicitly optimized, and is simply the result of the initial mesh connectivity after the collapses. This means that only a small subset of the potential connectivities are explored. There is no guarantee that this space of solutions contains the best solution or even a good one.

A final problem is that the algorithm is driven solely by the effort to keep each original vertex of the mesh on the surface of a sphere. In reality, being on the surface of a sphere does not necessarily imply being on the surface of the sphere-mesh: that is the case only if no other sphere (including interpolated ones) engulfs that point in its interior. To counter this problem, [TGB13] proposes to identify, as a pre-process, a maximal radius for each part of the surface, computed so that a locally tangent sphere with that radius will not exceed the boundary of the mesh; during collapses, this value limits the radii of any sphere originated by that vertex. This ameliorates the problem, but, as they observe, it fails to provide any guarantee. First, because the sphere potentially engulfing other surface samples will not be centered at the location where this limit is imposed. Second, because interpolated spheres (which can also cause the problem) are, again, disregarded.

These problems are intrinsic to this approach (which, to the best of our knowledge, is the only existing one for the automatic creation of sphere-meshes), and it is not clear how they can be addressed. This obstacle motivates the present proposal for a *semi*-automatic approach. The observation is that, despite these limitations, the automatic heuristic provides an excellent starting point for manual sphere-mesh tweaking.

4. Manual Tweaking Tool

Our sphere-mesh tweaking tool displays a sphere-mesh on screen (controlling the view with a trackball), and lets a user manipulate directly its shape and connectivity with several kinds of operations. Each operation is issued by first selecting one or multiple target spheres in S (by pointing and clicking) and then activating it either with the press of a button or with a mouse stroke.

The operations available to the user include (refer to Figure 2):

- translate the selected sphere or change its radius;
- collapse two selected spheres into one, thus the elimination edges in T or the reducing triangles T into a segment;
- duplicate the selected sphere into a new one, connected to the old one by a new segment;
- create a new sphere connected with a new triangle to a pair of selected spheres;
- delete a sphere and all the adjacent elements in T.

Collapse operations trigger the fusion of two spheres into one, performed as in the automatic approach [TGB13]. In particular, the operation includes the initialization of the sphere center and radius computed by minimizing the associated quadric, and the removal of elements in T, as described in Section 3.1.

Translation operations are performed on a plane parallel to the screen and are controlled with a fine-grained mouse stroke (optionally, it is possible to translate the sphere orthogonally to that plane instead, using a key combination). Duplication operations trigger a translation of the newly created sphere.

Duplicate and creation operations trigger automatically a translate operation of the created sphere, which is initialized (center, radius, and associated quadric) as the selected sphere or as the interpolation of the two selected spheres.

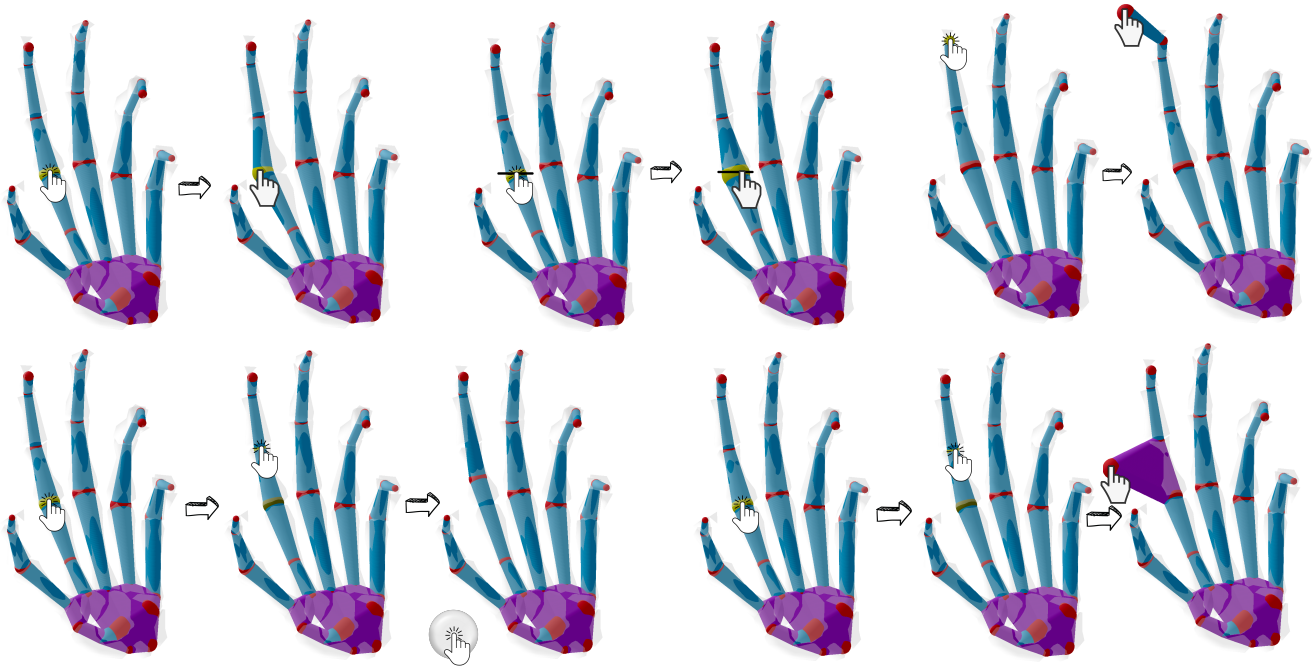

Figure 2: *An illustration of the main operations to edit a sphere-mesh. Above, from left: translation of one sphere (after its selection), scaling of a sphere (after its selection, with a horizontal drag), and generation of a new segment starting from an existing selected one. Below, from left: selection and collapse of two spheres of the sphere-mesh (by the press of a button); generation of a new triangle starting from a pair of selected spheres.*

4.1. Real-Time sphere-mesh rendering

During the entire interaction (including during mouse strokes), the sphere-mesh is displayed in conjunction with the semitransparent target mesh M, and their accurate intersection serves as a way to visualize their similarity (Figure 3, right).

A natural way to render a sphere-mesh would be to polygonize and rasterize its surface, which is known to always consist of the union of subsets of spheres, planes, and cone surfaces (including cylinders as special cases). We sidestep the need to explicitly identify and polygonize this surface with a more direct approach, where we simply render a collection of overlapping balls, including all S and a dense sampling of all the interpolated spheres, following directly the definition of the sphere-mesh. We rely on depth tests to produce the boundary of the union of the rendered balls.

While brute-force, this simple approach is made efficient by rendering each sphere as an impostor, following previous sphere-based rendering approaches [TCM06]. In short, a single screen-oriented quad is displayed for each sphere; fragments outside the inscribed 2D circle are discarded, and other fragments are lifted to have a pixel depth determined by their screen position, as the height on the front-facing half-sphere. This is fast and produces the correct intersections, via the depth-test, provided that an orthogonal projection is employed. Lighting is based on the normals computed, in screen space, for the half-sphere.

All figures in this paper are obtained in this way unless otherwise specified.

This approach has the advantage of requiring no preprocessing, making it a good fit for our scenario where the rendered sphere-mesh is modified on the fly. Also, because it produces a consistent depth buffer, it can be used to display the correct intersection with any polygonized surfaces (see Figure 3, right).

Structure visualization Optionally, the *structure* of the sphere-mesh can be displayed (rather than its boundary), by rendering only the spheres in S, and their connections in T as thick lines and trian-

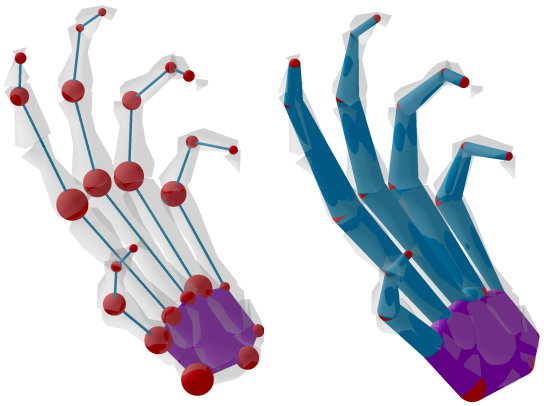

Figure 3: *A sphere-mesh displayed as either its connectivity or its boundary, rendered together with the intersecting, semitransparent input mesh (light gray).*

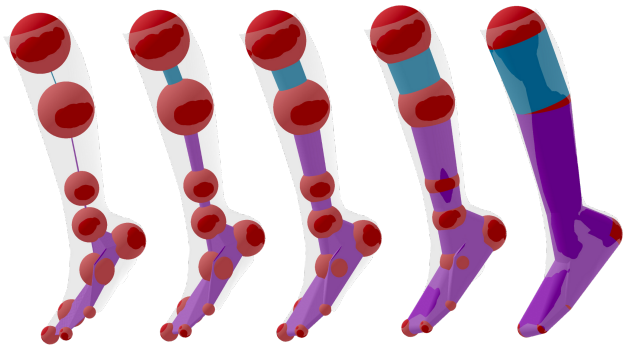

Figure 4: *Smooth transition between the visualization of the sphere-mesh boundary and the visualization of its structure. See also Video 01 in the additional materials.*

gles. This mode can be useful for selecting spheres and manipulating the structure (see Figures 1, 3, and 4, leftmost images).

We do so using the same algorithm, but simply replacing the radii of all the interpolated spheres (but not the ones in S) to a small constant c. This results in displaying segments and triangles in T as shaded small tubes or thin triangular membranes, easing their spatial interpretation (compared to simply using rasterized lines and flat triangles).

Because this is the only difference between the two rendering modes, we offer a continuous way to switch between the two modes by interpolating the radii of intermediate spheres from their original values to the c (see Figure 4 and attached Video 01).

5. Assessment of sphere-mesh accuracy

To quantify the geometric similarity between a given sphere-mesh $A = (S,T)$ and the target mesh $M = (V,F)$, we use the mutual Hausdoroff Distance between the boundary δA of A and M.

As a practical approach, we first approximate δA with a sufficiently dense *point-cloud* $\delta A'$, by sampling the boundary of A, and then measure the reciprocal Hausdorff Distance between $\delta A'$ and M using existing off-the-shelf tools to compare point-clouds with polygonal-meshes. In our experiment, we used the MeshLab [PMM*08].

Observe that δA is not trivially defined, as A is the union of several overlapping and implicitly defined primitives. We use the following algorithm.

Sampling $\delta A'$ We create the point-could $\delta A'$ with a simple heuristic, where we populate an initially empty point-cloud, one point at a time, by electing a random point $\mathbf{p}$ on any primitive (segment or triangle) in T, identifying the interpolated sphere centered in $\mathbf{p}$, selecting a random point $\mathbf{q}$ on that sphere, and then projecting that point out of all other primitives in the sphere-mesh (using the algorithm in [AMA16]), until that point is either on or out of any such primitive (thus reaching a position in δA). Figure 7 shows examples of rendering of the obtained point-clouds.

6. Experimental results

We tested our method on a small collection of target meshes. All experiments were performed on commodity hardware (8-Core Intel Core i9, 2.3 GHz, 16 Gb DDR4 RAM, AMD Radeon Pro 5500, 8 GB VRAM).

The input meshes have been prepared by coarsening standard tri-meshes from various sources with a standard polygonal simplification tool; we used the open-source implementation [PMM*08] of [MP97].

First, we fed input meshes to the automatic phase (described in Section 3), and tweaked the resulting sphere-meshes using our proposed interactive tool; (described in Section 4);then, we analyzed each produced sphere-mesh in terms of similarity with the input mesh (as described in Section 5).

Renderings of the inputs and results (after either phase) are shown in Figure 6; the corresponding data is reported in Table 1, including resolutions, computation and interaction times, and distance measurements. We provide both the input and the output files in the additional materials.

Discussion of results. The processing times of the automatic phase are under two seconds on commodity hardware, which makes this tool usable in most contexts (e.g. content creation during the development of a video game).

The interactive tool is only prototypal, yet a few minutes were sufficient in each instance to produce the intended results, suggesting that a more engineered GUI would be adopted by generic users.

In each test, the automatically produced sphere-meshes proved to be quite faithful in terms of geometric adherence to input meshes, for example well within the thresholds of typical collision proxies. The expectation that the number of primitives is excessive is confirmed by the comparison with the sphere-meshes obtained with the subsequent manual editing. The manual phase reduces significantly the primitive counts, without a significant effect on the geometric precision (and occasionally an improvement).

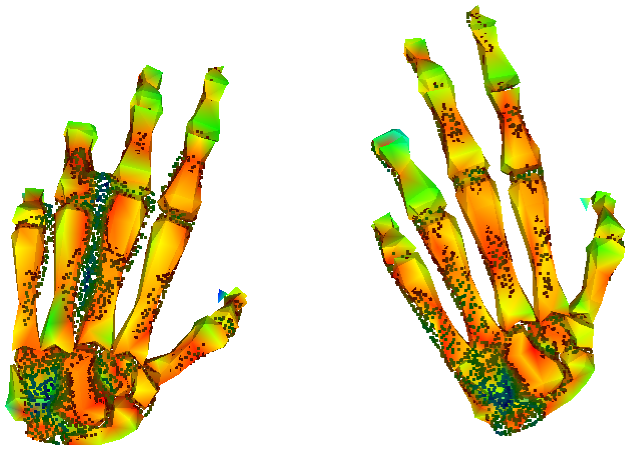

Figure 5: *Left: per point distance between the input mesh and the final sphere-mesh, and vice versa (the numerical values maximized over the surface are reported in the last two columns of Table 1).*

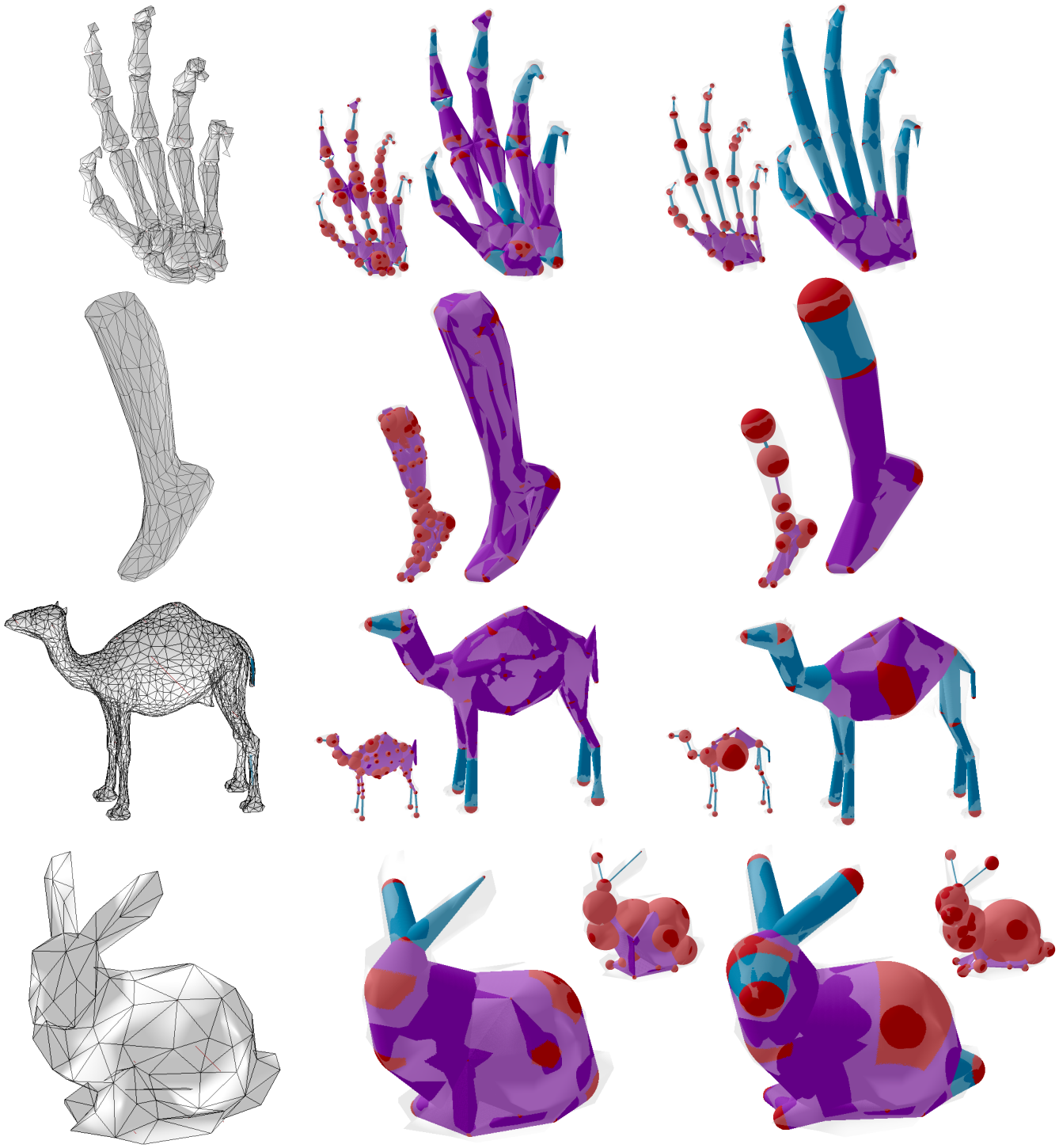

Figure 6: *Examples of sphere-meshes obtained from input tri-meshes (left) with our fully automatic method (middle), followed by manual tweaking (right). The smaller images show the connectivity of the sphere-meshes. In the sphere-meshes renderings, the original mesh is shown as an intersecting, lightly transparent surface. Refer to Table 1 for the associated resolutions and measures.*

Table 1: *Statistics on our results. Reported times are computation times for the automatic method and total interactive session times for the tweaking method. For the tweaking method, we report the used parameter ε (see Section 3.1). "Distance" reports the Hausdoroff distances expressed as percentages of the axis-aligned bounding-box diagonal.*

Input tri-mesh (A)		Construction			Output sphere-mesh (B)			Distance	
name	**faces**	**phase**	**ε**	**time**	**segments**	**triangles**	**spheres**	**A→B**	**B→A**
Foot	510	automatic	0.040	30 msec	130	126	64	0.87%	0.88%
		tweaking	-	∼ 3 min	17	17	14	0.85%	0.87%
Camel	4040	automatic	0.040	1988 msec	533	176	66	0.87%	1.11%
		tweaking	-	∼ 4 min	26	7	23	1.08%	1.41%
Bunny	260	automatic	0.095	28 msec	66	42	24	1.88%	1.24%
		tweaking	-	∼ 6 min	21	20	18	1.05%	1.80%
Hand	1600	automatic	0.035	296 msec	332	144	74	0.61%	1.20%
		tweaking	-	∼ 7 min	44	20	38	0.63%	1.27%

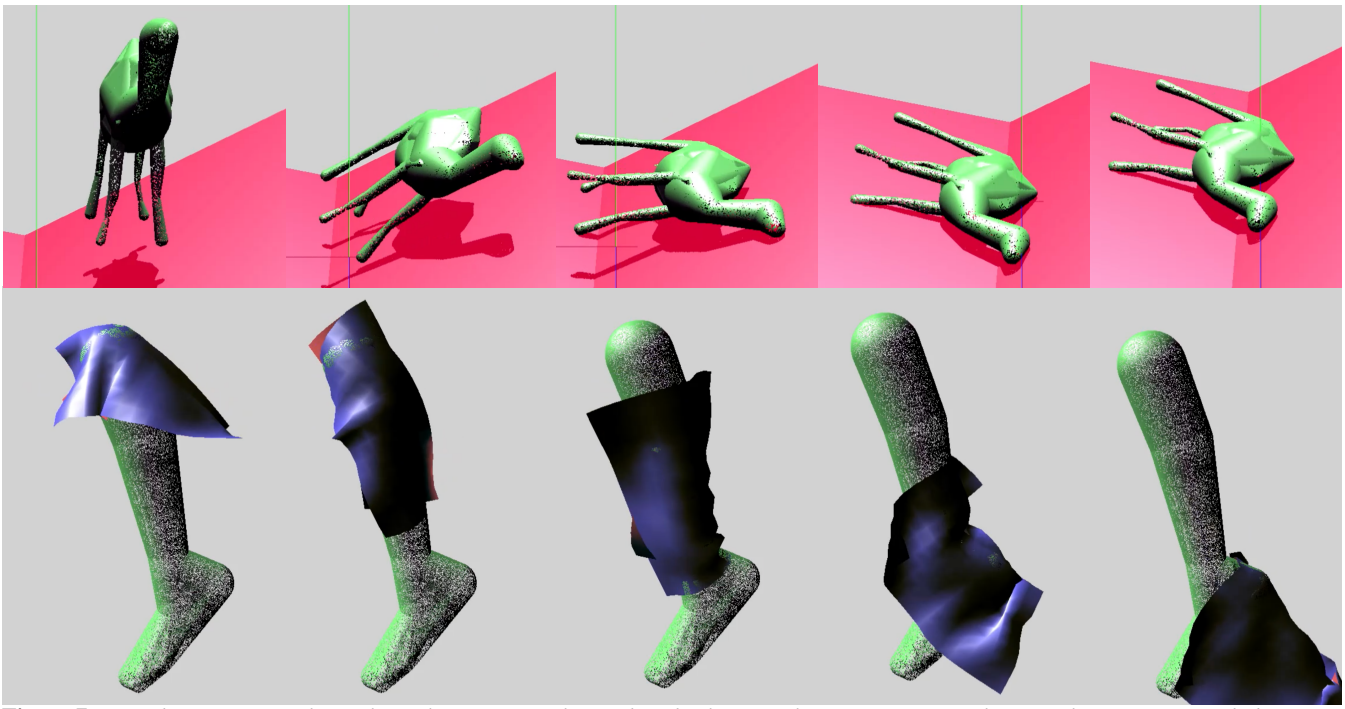

Figure 7: *Two demonstrative physical simulations using the produced sphere-meshes proxies. Top: sphere-mesh as a static rigid object in a cloth simulation. Bottom: sphere-mesh as a dynamic rigid object. See Videos 02 and 03 in the additional materials.*

7. Conclusions

In conclusion, our semi-automatic methods are capable of constructing sphere-meshes that are, at the same, sufficiently coarse and sufficiently accurate, to be conveniently employed in several potential downstream applications.

Field testing of results. This substantiates these claims, we cannot, unfortunately, experiment on existing software (such as game engines, or physical simulators), because, to our knowledge, none exists that natively supports this class of geometric proxy. We reiterate that we attribute the lack of support for sphere-meshes not only to their recent introduction but also to the current difficulty of construction, which we address in this paper. To test the output sphere-meshes, we employ them in two examples of simple Real-Time physical simulation scenarios, which we implement to this end (see Figure 7 and attached Videos 02 and 03): in one, the produced sphere-mesh is used as a dynamic rigid body. On the other, it is used as a static object in a simple cloth simulation. The demonstrative simulations themselves are implemented as standard Position Based Dynamics with Verlet integration, with positional constraints preventing intersections. For the rendering of the sphere-meshes in the simulations, we reused the dense point-cloud sampling the sphere-mesh surface (see Section 5), as it is more efficient than our preview rendering when the sphere-mesh is constant.

We think that, in addition to the potential direct practical usability, our semi-automatic tool has value in establishing the potential expressiveness of sphere-meshes as a representation for geometric proxies, fostering future fully automatic construction algorithms.

Future work. We recognize that our work is far from conclusive. Both the automatic parts and the manual editor can be subject to substantial ameliorations, both algorithmic and engineering (including the GUI). Proper user studies are necessary to assess the usability of the interactive editor, with our current experiments (with ourselves as users) serving only as a first indication of their usability.

Additional materials and a reference implementation are available at the Project Page: https://tarini.di.unimi.it/spheremesh/.

References

[AMA16] ANASTASIA, TKACH, MARK, PAULY, and ANDREA, TAGLIASACCHI. "Sphere-Meshes for Real-Time Hand Modeling and Tracking". *ACM Trans. Graph.* 35.6 (Dec. 2016). ISSN: 0730-0301. DOI: 10.1145/2980179.2980226. URL: https://doi.org/10.1145/2980179.2980226 1, 5.

[AMD07] A., SUD, M., FOSKEY, and D., MANOCHA. "Homotopy-preserving medial axis simplification". *International Journal of Computational Geometry & Applications* 17.05 (2007), 423–451 2.

[ATW*17] ANGLES, BAPTISTE, TARINI, MARCO, WYVILL, BRIAN, et al. "Sketch-Based Implicit Blending". *ACM Trans. Graph.* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130825. URL: https://doi.org/10.1145/3130800.3130825 2.

[BenAR] BENAZERA, EMMANUEL. *Fibonacci Heap Implementation in Python*. https://github.com/beniz/fiboheap. GitHub repository. YEAR 3.

[BJM10] B., MIKLOS, J., GIESEN, and M., PAULY. "Discrete scale axis representations for 3D geometry". *ACM Transactions on Graphics (TOG)* 29.4 (2010), 101 2.

[BMUS15] BARBIERI, SIMONE, MELONI, PIETRO, USAI, FRANCESCO, and SCATENI, RICCARDO. "Skeleton Lab: an Interactive Tool to Create, Edit, and Repair Curve-Skeletons". *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. Ed. by GIACHETTI, ANDREA, BIASOTTI, SILVIA, and TARINI, MARCO. The Eurographics Association, 2015. ISBN: 978-3-905674-97-2. DOI: 10.2312/stag.20151299 2.

[CPW*23] CHEN, ZHEN, PAN, ZHERONG, WU, KUI, et al. "Robust Low-Poly Meshing for General 3D Models". *ACM Trans. Graph.* 42.4 (July 2023). ISSN: 0730-0301. DOI: 10.1145/3592396. URL: https://doi.org/10.1145/3592396 2.

[DEGN99] DEY, TAMAL, EDELSBRUNNER, HERBERT, GUHA, SUMANTA, and NEKHAYEV, DMITRY. "Topology preserving edge contraction". *Publications de l'Institut Mathématique* 66 (1999) 2.

[FA05] F., CHAZAL and A., LIEUTIER. "The λ-medial axis". *Graphical Models* 67.4 (2005), 304–331 2.

[H67] H., BLUM. "A Transformation for Extracting New Descriptors of Shape". *Models for the Perception of Speech and Visual Form*. Ed. by WATHEN-DUNN, W. Cambridge: MIT Press, 1967, 362–380 2.

[HELC09] H., CORMEN THOMAS, E., LEISERSON CHARLES, L., RIVEST RONALD, and CLIFFORD, STEIN. "Fibonacci Heaps". *Introduction to Algorithms*. 3rd. Cambridge: MIT Press, 2009, 505–530 3.

[JÉTE16] JEAN-MARC, THIERY, ÉMILIE, GUY, TAMY, BOUBEKEUR, and ELMAR, EISEMANN. "Animated Mesh Approximation With Sphere-Meshes". *ACM Trans. Graph.* 35.3 (May 2016). ISSN: 0730-0301. DOI: 10.1145/2898350. URL: https://doi.org/10.1145/2898350 1.

[Lue01] LUEBKE, DAVID P. "A developer's survey of polygonal simplification algorithms". *IEEE Computer Graphics and Applications* 21.3 (2001), 24–35 2.

[MP97] M., GARLAND and P., HECKBERT. "Surface simplification using quadric error metrics". *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997, 209–216 3, 5.

[NSR01] N., AMENTA, S., CHOI, and R., KOLLURI. "The power crust". *Proceedings of the sixth ACM symposium on Solid modeling and applications*. ACM, 2001, 249–266 2.

[PMM*08] PAOLO, CIGNONI, MARCO, CALLIERI, MASSIMILIANO, CORSINI, et al. "MeshLab: an Open-Source Mesh Processing Tool." Vol. 1. Jan. 2008, 129–136. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136 5.

[ST14] SCHNEIDER, ROSÁLIA G and TUYTELAARS, TINNE. "Sketch classification and classification-driven analysis using fisher vectors". *ACM Transactions on graphics (TOG)* 33.6 (2014), 1–9 2.

[TCM06] TARINI, MARCO, CIGNONI, PAOLO, and MONTANI, CLAUDIO. "Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization". *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), 1237–1244. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.115. URL: https://doi.org/10.1109/TVCG.2006.115 4.

[TGB13] THIERY, JEAN-MARC, GUY, ÉMILIE, and BOUBEKEUR, TAMY. "Sphere-Meshes: Shape Approximation using Spherical Quadric Error Metrics". *ACM Transactions on Graphics* 32.178 (Nov. 2013), 1–12. DOI: 10.1145/2508363.2508384 1–3.

[TW04] T., DEY and W., ZHAO. "Approximate medial axis as a voronoi subcomplex". *Computer-Aided Design* 36.2 (2004), 195–202 2.