Variational Shape Approximation

HARIHARA GOWTHAM JETTI, Institut Polytechnique de Paris, France
JACKSON SUNNY, Ecole Polytechnique, France

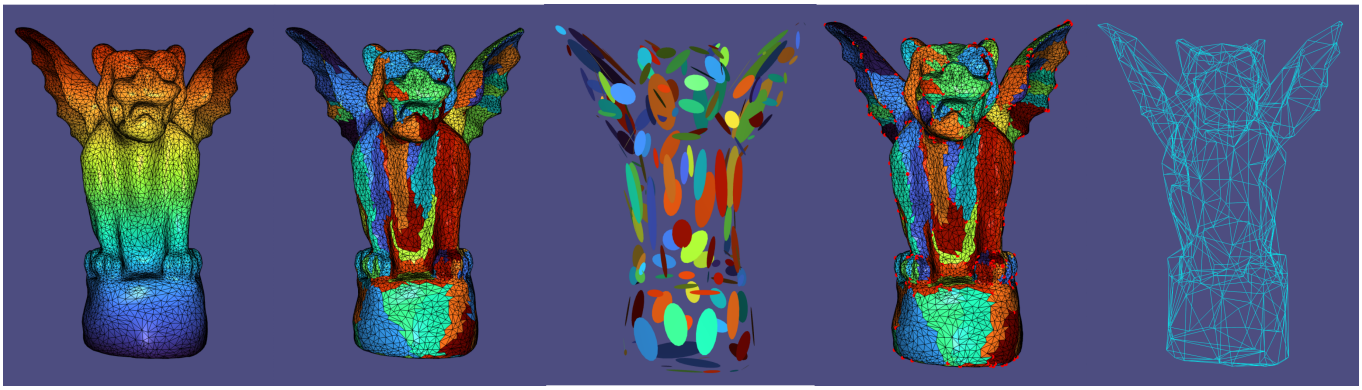

Fig. 1. Left to Right: Input mesh, Partitioned mesh, Approximated proxies, Computed anchor vertices, Final simplified mesh

1 INTRODUCTION

It is very expensive to represent, analyse, and render very densely sampled triangle meshes, as it requires a lot of computational power. Finding a compressed version of the original mesh, which is precise and geometrically similar to the original surface, is a core research problem. The authors propose a simple and efficient variational shape approximation approach. it is an error-driven optimization of a partition and its proxies. They provide a concise geometric representation in the form of a polygonal mesh. The main goal of this project is to approximate a given 3D shape with several geometric proxies while preserving its shape, topology and highlights minimizing the distortion error through repeated clustering of the face into best-fitting regions. This is very useful for compressing the shape without losing too many details. This method solely depends on the initial geometry of the 3d shape. And as the proposed method is iterative, it cannot compete with the greedy methods in terms of computational time.

2 RELATED WORKS

[1] gives a method for concise, faithful approximation of complex 3D datasets is key to reducing the computational cost of graphics applications. Using the concept of geometric proxies, it drives the distortion error down through repeated clustering of faces into best-fitting regions. Also, a new metric based on normal deviation and demonstrating its superior behaviour at capturing anisotropy has been introduced. Though this is the paper that we are implementing as part of this project, after its publication in 2004, a lot of new works have been built on it. In this section, we will quickly go through a few of the very relevant and recent works in this category.

[2] presented an approach on how to obtain a simplified mesh from the output of the VSA procedure. This simplification is either based on a simple plane intersection or based on a variational optimization problem. The main result of [3] is a polynomial-time approximation algorithm that computes a piecewise-linear surface of size O(Ko log Ko), where Ko is the complexity of an optimal surface satisfying the constraints of the problem. The technique developed in this paper is more general and applies to several other problems that deal with the partitioning of points subject to certain geometric constraints. [4] proposes a novel polygonal remeshing technique that exploits a key aspect of surfaces: the intrinsic anisotropy of natural or man-made geometry. As a result, this technique generates polygon meshes mainly composed of quads in anisotropic regions and triangles in spherical regions. This approach provides the flexibility to produce meshes ranging from isotropic to anisotropic, from coarse to dense, and from uniform to curvature-adapted. Modellers manually craft low-poly meshes for given 3D building models to achieve the ideal balance between the small element count and the visual similarity.[5] proposes a novel and simple algorithm to automate this process by converting high-poly 3D building models into both simple and visually preserving low-poly meshes. [6] presents a new method for simplifying 3D point clouds for digital twin city models. Such data usually contains much redundant information, noise and outliers. This implies that most subsequent processing tasks are costly both in terms of processing times and hardware infrastructure. The core idea of this paper is that most of the objects present in such scenes can be approximated as a combination of simple geometric primitives. [7] proposes a family of tractable variational approximations that are more accurate and faster to calibrate for this case. It combines a parsimonious approximation for the parameter posterior with the exact conditional posterior of the latent variables.[8] present a variational method for extracting general quadric surfaces from a 3D mesh surface. The method mentioned in [9] performs an automatic trade-off between representation complexity and approximation error without relying on a user-supplied predetermined number of shape proxies.

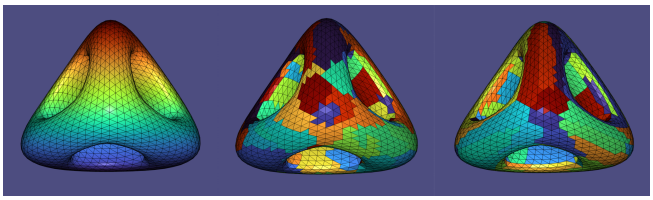

Fig. 2. An input mesh, Regions computed using L^2, Regions computed using $L^{2,1}$

3 TECHNICAL DETAILS

Defining an appropriate error metric plays a significant role in shape approximation. In general, the L_2 or Hausdorff metrics are often used to compare two triangulated surfaces. In this method, the authors introduced a new metric measuring the geometric relevance of the Proxies for a given surface. Which will allow us to score a partition in terms of how well it approximates a surface.

L^2 metric: It measures the integral of the squared error between the region Ri and its orthogonal projection of the proxy P i to a plane. Finding the L^2-optimal proxy for a given region is very simple as it just computes a barycenter and a covariance matrix. As the authors mentioned, this metric tries to match the geometry of the given shape by approximating the geometric position of the object in space. Unfortunately, it fails in preserving normals which accounts for lighting/shading effects, and this error may also fail in a hyperbolic case. Moreover, there is no unique optimal shape and alignment. Since we are targeting a variational approach, this non-unique optimality is not desired. Thus, the minimization algorithm can randomly jump around in the null space resulting in undesired oscillations. This error is defined as:

$$L^2(R_i, P_i) = \iint_{x \in R_i} ||x - \pi_i(x)||^2 dx$$

where π_i is the orthogonal projection on P_i.

$L^{2,1}$ metric: While L^2 is an extension of the classic Hausdorff metrics, to overcome these shortcomings, the authors introduced a new metric called $L^{2,1}$, as it is based on an L^2 measure of the normal field. This is the main innovation of this paper. The error captures the anisotropy of the surface in a better way and enables an easier implementation of the optimal proxy. This error can also be computed very quickly as it is just the average of the normals associated with that region. And they proved that there is a unique optimal shape and alignment for almost all the surface types like parabolic, elliptic, or hyperbolic. And $L^{2,1}$ error consistently provides equal or better visual results, as shown. This error is defined as:

$$L^{2,1}(R_i, P_i) = \iint_{x \in R_i} ||n(x) - n_i||^2 dx$$

where n_i is normal of the triangle, n(x) is the normal of the proxy.

3.1 Total distortion error

The total distortion error of a surface is given by the sum of either the L^2 or $L^{2,1}$ error metrics of the proxies made with their respective region. Although the authors cannot guarantee the global convergence for all models in the variational approach, very good and visually pleasing results are observed within a few iterations as the proxies start settling down or oscillate around extremely similar distortion errors. The sum of the L^2 or $L^{2,1}$ errors of all the regions is defined as the Total distortion error. It can be defined as follows:

$$E(R, P) = \sum_{i=1...k} E(R_i, P_i)$$

Algorithm 1 Geometry Partitioning and Proxy Fitting

```
procedure GEOMETRY_PARTITIONING(mesh , seed triangles, adjacency graph)
    Create a priority queue PQ (using the error it makes with a region as the priority)
    Associate a region ID with each seed triangle
    for each seed triangle do
        Insert its adjacent triangles into PQ along with the region ID
    while PQ is not empty do
        Pop the PQ and assign it to the corresponding region if not already assigned
        Push its adjacent triangles to PQ
    Replace each region with a proxy
    while Convergence of total distortion error do
        Pick a triangle from each region that makes the maximum error with its proxy
        Recursively call GEOMETRY_PARTITIONING with these newly picked triangles as seed triangles
```

4 IMPLEMENTATION

The variational shape approximation method has the following essential steps. In summary, the input 3D surface is divided into k- clusters, followed by computing the best-fitting proxies by the convergence of total distortion error. These newly updated proxies are then used to find the anchor vertices and edges to triangulate the surface and are used to generate the final abstract mesh by removing unnecessary edges. Here is our step-by-step implementation

4.1 Geometry Partitioning & Proxy fitting

Initially, we performed a k-proxy clustering to divide the whole mesh into k- regions. We randomly pick a set of k- triangles (seed triangles) which act as starting points of the regions respectively, and add their neighbours to a priority queue. While the priority queue is not empty, we pick the triangle that has the least error with the region it belongs to. We check whether the triangle is already assigned to any region. If not already assigned, we assign the triangle to that region and add its adjacent triangles along with the region to the priority queue. While performing this loop, we also add a constraint that a triangle cannot enter the queue more than 3 times. When the priority queue is empty, we get all the initial regions. Then, for each region, we compute a best-fitting proxy (plane or ellipse), i.e. the mean of all the barycenters of the triangles in that region and a best-fit normal of that region. Now, we pick one triangle from all the regions with a maximum error with its proxy. We follow the above procedure by taking this set of triangles as seed triangles until global convergence of the total distortion error is achieved.

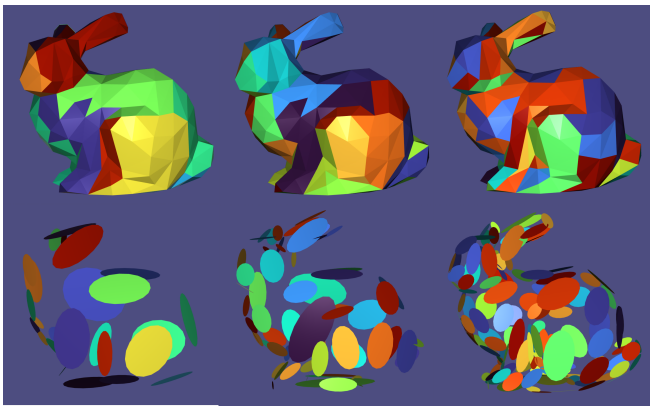

Fig. 3. Geometric partitions (first row) and corresponding proxies (second row)

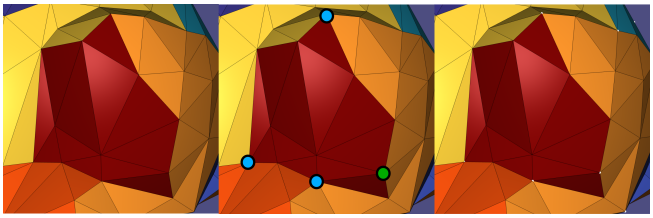

Fig. 4. A region, expected anchor vertices (blue - on junctions, green - additional anchor vertex inserted based on the threshold), anchor vertices computed by the algorithm

4.2 Remeshing

. After dividing our input region into k regions, we have to create a mesh from these regions to visualise these regions as a single mesh. First, we compute the anchor vertices and edges to achieve this.

4.2.1 Extracting anchor vertices and edges. .

From the nearly optimal partitioning of the whole surface, the initial step is to mark the set of points whose location is at the intersection of three or more regions. We then mark a few other points on each boundary of the region as anchor vertices. We decide that a point on the boundary of two regions be one of the anchor vertices when the maximum orthogonal distance between its two adjacent anchor vertices is multiplied by the angle between the normals of the two adjacent regions less than a threshold of 0.4. We repeat this step until each boundary vertex is either an anchor or the criteria is satisfied. A set of edges, called anchor edges, are then added between each pair of adjacent anchor vertices.

4.2.2 Triangulation & Polygonization. . Now, with the anchor vertices and edges already defined, we have a polygonal mesh. But in order to truly call it a mesh, they need to triangulate it because when the number of proxies is fairly small, the polygons have no guarantee of being almost flat or convex.

We have implemented a Dijkstra-based approach on all the points belonging to the region. We create a weighted undirected graph between all the vertices in the region with edge lengths as the weights.

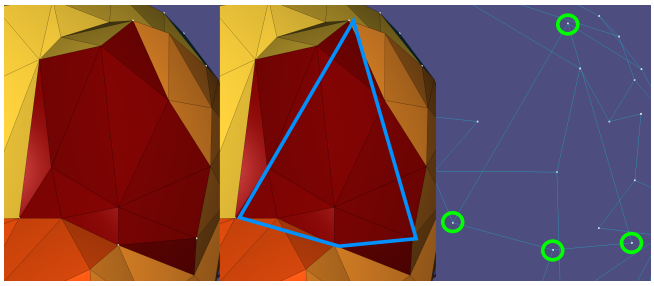

Fig. 5. A region with anchor points (in white colour), Expected polygon connecting the anchor vertices (drawn in blue), Polygon generated by the algorithm (vertices are highlighted)

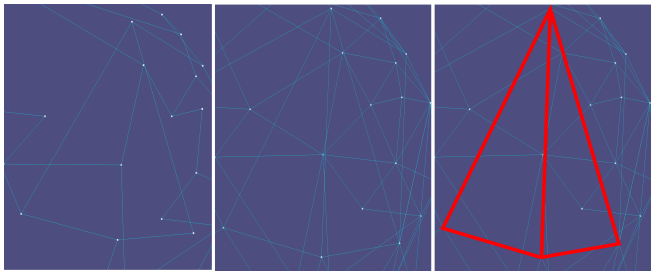

Fig. 6. Initial polygon, Polygon after triangulation, The triangles of the region highlighted in red colour

With anchor vertices as the source, and all the other vertices, including anchor vertices in the region, as targets. For each vertex, we assign an anchor vertex which has the shortest path between them. Now, we check the vertices of the triangle with three different anchor vertices assigned to them and connect those anchor vertices. We repeat this process for all the regions to get the final edges displayed. The overall procedure is shown in Algorithm 3, which is called for each region.

We could not complete the polygonization because of the time constraints, stopping us from visualising the final result as polygons. So, instead, we rendered the final output using edges, which is a visually similar result to that after polygonization.

Algorithm 2 Meshing (triangulation and polygonization)

```
1: procedure TRIANGULATION(vertices, anchor vertices, associated triangles)
2:     for each vertex i do
3:         if anchor a is shortest to i then
4:             Assign the label of i to a
5:     for each face f do
6:         if all vertices of f has different labels then
7:             Create a triangle with these three anchor vertices.
```

Additional implementation: Since it took us some time to understand what was happening with the triangulation, we also implemented a new algorithm for converting each 3D polygon (made by connecting anchor vertices) into triangles. The method we invented is similar to the ear-clipping algorithm on 2D polygons but

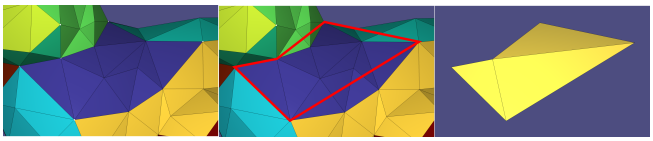

Fig. 7. Result of our new algorithm for triangulating a 3D polygon represented by a surface patch. The computed regions and anchor points, Polygon connecting anchor vertices drawn in red colour, Triangles computed from the polygons

with an extra constraint to ensure that the ears are made to lay on/near the surface. The method is as follows:

A Dijkstra-based approach to computing the triangles. We create a weighted undirected graph for each region with the connections of all anchor vertices corresponding to that region with weights equivalent to edge lengths. Then, We find the shortest (minimum) distance between i-1 and i+1 vertices and create a triangle from i-1, i, i+1 anchor vertices and pop out i from the list of anchor vertices from the region. We repeat the same process until we are left with 3 vertices. Then, we form the last triangle using these three vertices. Though this seems theoretically correct, a few triangles are missing. So, the result is visually not pleasing.

Algorithm 3 Meshing (triangulation and polygonization)

```
1:  procedure TRIANGULATION(anchor vertices, associated triangles)
2:      Create a list of anchor vertices in order
3:      Compute the shortest distances between all anchor points
4:      while the list has more than three vertices do
5:          Find the ear (i-1)i(i+1) such that the distance between (i-1) and (i
6:          if distance of anchor i − 1 and anchor i + 1 is shortest then
7:              draw an edge from i − 1 to i and i to i + 1
8:              remove the anchor i
9:      for each face f do
10:         if all vertices of f has different labels then
11:             Create a triangle with these three anchor vertices.
```

5 RESULTS

5.1 Observations

We could clearly see the difference in the computation time for each model with different error metrics. The difference in results with the L^2 and $L^{2,1}$ metrics is very noticeable. For example, the character in Figure 8 illustrates the effect of the error metric on the approximation. L^2 is on the left, and $L^{2,1}$ is on the right. They behave similarly on near-spherical regions such as the top of the head. But the mouth region is noticeably very different in each case.

5.2 Limitations and failure cases

Though our code is able to generate partitions for no-water-tight (open) meshes, it fails to compute anchor vertices. Thus, the final output cannot be generated on non-water-tight models. We have used only closed (water-tight) models to compute the results.

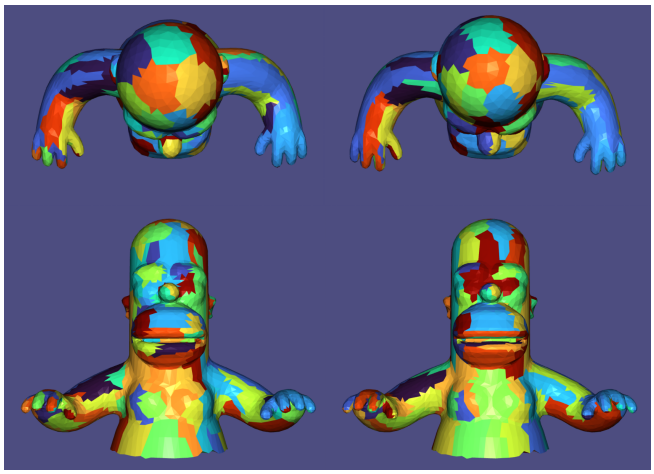

Fig. 8. A noticeable difference between the L^2 and $L^{2,1}$ metrics

6 CONCLUSION AND FUTURE DIRECTIONS

In this project, we implemented an approximation technique that converts a densely sampled triangular mesh into a simplified mesh. To achieve this, the authors proposed a new error metric $L^{2,1}$ based on the L^2 measure of the normal fields. This allows for capturing more minute details than the old L^2 error metric. As an extension, we can implement our code to work on open meshes as well.

REFERENCES

[1] Cohen-Steiner, David, Pierre Alliez, and Mathieu Desbrun. "Variational shape approximation." ACM SIGGRAPH 2004 Papers. 2004. 905-914

[2] Skrodzki, Martin, Eric Zimmermann, and Konrad Polthier. "Variational shape approximation of point set surfaces." Computer Aided Geometric Design 80. 2020

[3] Agarwal, Pankaj K., and Subhash Suri. "Surface approximation and geometric partitions." SIAM Journal on Computing 27.4 (1998)

[4] Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., and Desbrun, M. "Anisotropic polygonal remeshing." In ACM SIGGRAPH 2003 Papers

[5] Gao, Xifeng, Kui Wu, and Zherong Pan. "Low-poly mesh generation for building models." ACM SIGGRAPH 2022 Conference Proceedings

[6] Guinard, Stephane A., Sylvie Daniel, and Thierry Badard. "3D point clouds simplification based on geometric primitives and graph-structured optimization." 2022 26th International Conference on Pattern Recognition (ICPR). IEEE, 2022

[7] Rubén Loaiza-Maya, Michael Stanley Smith, David J.Nott, Peter J. Danaher, "Fast and accurate variational inference for models with many latent variables", ELSEVIER 2022

[8] Yan, Dong-Ming, Yang Liu, and Wenping Wang. "Quadric surface extraction by variational shape approximation." International conference on geometric modeling and processing. Springer, Berlin, Heidelberg, 2006

[9] Li, Bao, et al. "Variational surface approximation and model selection." Computer Graphics Forum. Vol. 28. No. 7. Oxford, UK: Blackwell Publishing Ltd, 2009

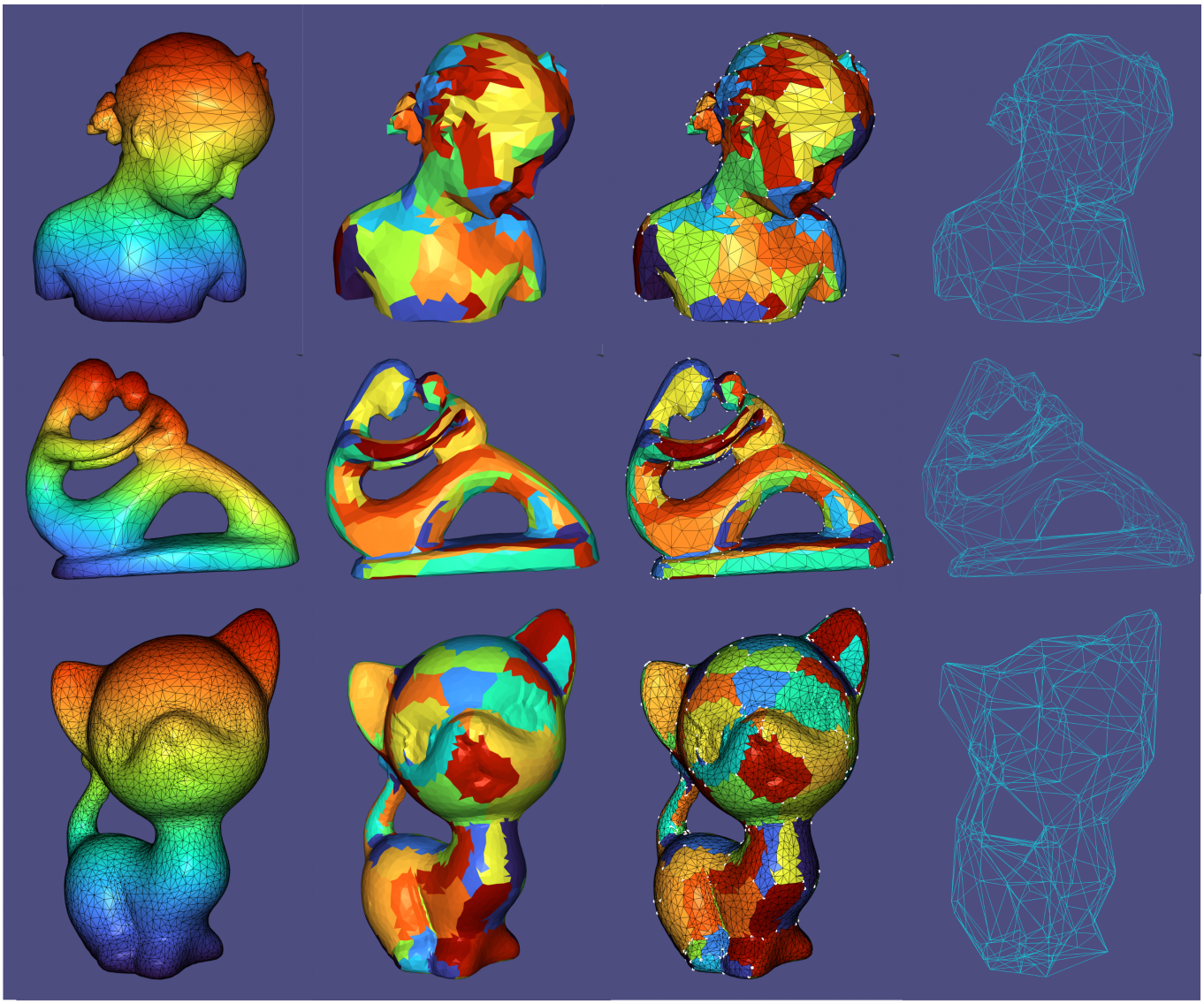

Fig. 9. More results: Input mesh, Partitions, Anchor vertices, Final simplified representation