High-Performance Graphics (2023)
J. Bikker and C. Gribble (Editors)

Spherical Parametric Measurement for Continuous and Balanced Mesh Segmentation

Huadong Zhang , Lizhou Cao, and Chao Peng †

School of Interactive Games and Media, Golisano College of Computing and Information Sciences, Rochester Institute of Technology, USA

Abstract

Mesh segmentation is an important process for building the discrete mesh structure used on the GPU to accelerate geometry processing applications. In this paper, we introduce a novel mesh segmentation method that creates balanced sub-meshes for high-performance geometry processing. The method ensures topological continuity within sub-meshes (segments) and evenly distributes the number of triangles across all sub-meshes. A new cohesion algorithm computes the chord distances between triangles in the spherical domain and re-groups the triangles into the sub-meshes based on a distance-based measurement condition. A new refinement algorithm between the neighboring sub-meshes is conducted to resolve the non-manifold issue and improve the boundary smoothness. Both algorithms are executed in a parallel fashion. In advancing the state-of-the-art, our approach achieves exactly balanced triangle counts and mitigates the non-manifold issue significantly. The algorithms require the input meshes to have a closed-manifold genus of zero, which is a constraint that is commonly associated with the concept of sphere-based parameterization. We evaluated the effectiveness of our approach in supporting two geometry processing applications. The results show that the performance is enhanced by leveraging the structure of the balanced sub-meshes from our approach.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ *Mesh models; Shape analysis; Shape modeling;*

1. Introduction

Surface mesh segmentation is a crucial problem in computer graphics applications, a highlighted in many existing studies [Sha08, CGF09, RMG18, LLS*05, CL18, PC06, CSAD04, KYD*18, MPO21]. While segmentation criteria vary depending on the application, determining which sub-mesh a triangle belongs to significantly impacts the resulting sub-meshes. Popular segmentation methods, like region growing and hierarchical clustering, maintain topological continuity but do not balance the number of triangles among the sub-meshes. This can lead to underutilization of GPU computing resources or imbalanced workloads among parallel threads, and make fast re-configuring of the number of sub-meshes problematic due to costly sequential algorithmic steps.

This work focuses on mesh segmentation for high-performance geometry processing applications that require a block-based storage scheme for meshes. Examples of such applications include mesh compression and GPU-based parallel LOD. In these cases, it is preferable to have geometry primitives represented in discrete, balanced mesh structures for optimal performance and balanced workloads. For instance, GPU architecture comprises an array of streaming multiprocessors, with each streaming multiprocessor containing identical execution cores that can manage blocks of threads to access the same shared memory. However, the bulk of vertices and triangles must be accessed from global memory. Until recently, RXMesh [MPO21] was capable of segmenting the mesh and creating a GPU-friendly mesh structure. This method mandates the largest sub-mesh to be smaller than the size of the GPU's shared memory so that all sub-meshes can fit into the distributed shared memory. However, the triangle counts are not balanced, resulting in memory underutilization and imbalanced workloads among the blocks of threads. The state-of-the-art balanced segmentation method based on the k-way graph partitioning, METIS [KK97], can generate sub-meshes with balanced triangle counts. However, the resulting sub-meshes often can not have the 2-manifold topology.

The objective of this work is to improve the efficiency of high-performance parallel computing by supporting balanced, block-based mesh segmentation. To achieve this goal, we propose a new approach, as shown in Figure 1, that advances the state-of-the-art in satisfying all three requirements below:

- *Achieving exact balancing of triangle counts among sub-meshes.* This is crucial because memory consumption is bounded by the size of the sub-mesh that has the largest amount of triangles. If the triangle counts are unbalanced, it can result in underutilized or

† {hz2208, lc1248, cxpigm}@rit.edu

DOI: 10.2312/hpg.20231140

fragmented memory on the streaming multiprocessors processing the sub-meshes with fewer triangles. This can impact the performance of applications that rely on balanced mesh structures.

- *Maintaining the 2-manifold topology of each sub-mesh.* This is important because each sub-mesh should be a continuous surface patch with 2-manifold triangle connection, allowing the preservation of local topological space. However, in traditional mesh segmentation methods, sub-meshes may lose their 2-manifold topology after segmentation. This issue typically manifests as a "bowtie" non-manifold shape, where two triangles share a vertex rather than an edge. Such non-manifold shapes can complicate the boundary of the sub-mesh, resulting in more external edges, and can also break the surface continuity.
- *Supporting fast segmentation of meshes.* This is important because the number of sub-meshes should be customizable according to hardware specifications or specific application demands. However, resegmenting a mesh to a different number of sub-meshes can be time-consuming, especially when done sequentially. To ensure that mesh segmentation is efficient and fast, parallel algorithms are needed. Our approach addresses this issue by utilizing parallel algorithms that can quickly segment the mesh into the desired number of sub-meshes.

2. Related Work

To provide the context for our work, we conducted a review of several methods for procedural mesh segmentation.

One such method is the region growing approach, which starts with k seed triangles and grows them into sub-meshes [KT96]. Other methods, such as those proposed by Lavoué et al. [LDB05] and Yamauchi et al. [YGZS05], conduct curvature analysis to determine the sub-mesh regions. Another approach, introduced by Bergamasco et al. [BAT12], is a semi-supervised growing method that starts from a set of user-selected seeds. More recently, Mahmoud et al. [MPO21] presented RXMesh, a new GPU-friendly data structure that represents surface meshes into patches using a seed-based growing method. While region growing methods guarantee the continuous topology of the sub-meshes, but they do not balance the triangle counts.

Another approach for mesh segmentation is hierarchical clustering (e.g., [XLXG11, AFS06, WLAT14]), which starts by treating each triangle as an individual sub-mesh and then merges clusters iteratively. The study conducted by Sander et al. [SSGH01] measured the merging cost using compactness and planarity parameters, where the squared perimeter of the sub-mesh is used to measure the compactness, and the mean-squared distance between the sub-mesh and the best-fitting plane is used to measure the planarity. In contrast, Garland et al. [GWH01] measured the planarity using the distance between pairs of triangles and orientation norms, and applied the quadric error metric for efficient computation. They calculated the compactness using the ratio between the square of the perimeter and the area. Hierarchical clustering methods maintain the continuity inside sub-meshes but do not allow customization of the number of sub-meshes and do not balance the triangle counts among sub-meshes.

Iterative clustering methods have similarities with the $k - means$ clustering algorithm in that they start with k sub-meshes and iteratively add triangles to them [Llo82, HSD00]. The approach introduced by Shlafman et al. [STK02] measured the geodesic distance when deciding which triangles to add to each sub-mesh. However, computing the geodesic distance on a 3D mesh surface can be computationally expensive [Sha08]. While iterative clustering methods can maintain the continuity of the sub-meshes, they do not guarantee balanced triangle counts among the sub-meshes. Another approach proposed by Zhao et al. [ZTT12] aimed at decomposition for GPU-accelerated mesh compression. They used the geodesic distance measurement from the work of Peyré et al. [PC06] but had to sacrifice the balance of triangle counts among the sub-meshes to improve the shapes and boundaries of the sub-meshes.

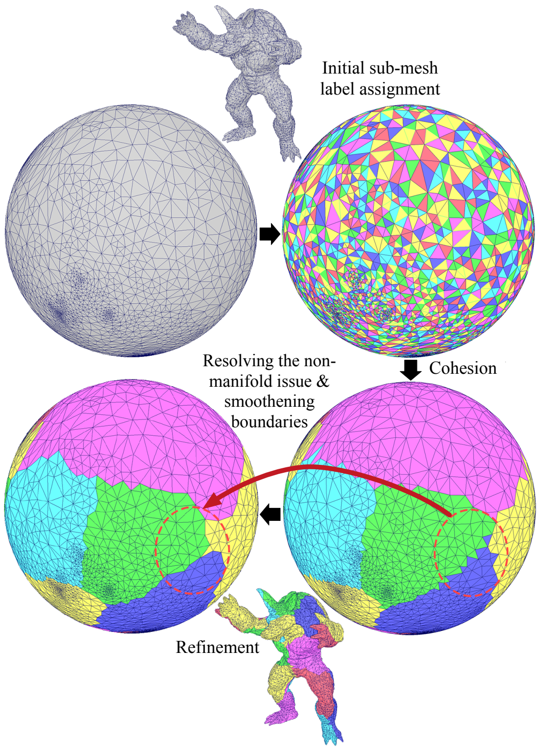

Figure 1: *An overview of our approach for segmenting a surface mesh into continuous and balanced sub-meshes. Initially, each triangle is randomly assigned a sub-mesh label, with a focus on achieving balanced triangle counts, rather than surface continuity. Then, we use a parallel cohesion algorithm that brings nearby triangles together into a common sub-mesh. After the cohesion step, we use a parallel refinement algorithm to resolve the non-manifold issue and improve the smoothness of the sub-mesh boundaries, while ensuring that the sub-meshes maintain the same number of triangles during the processing.*

Achieving an exact balance of sub-mesh sizes is a challenging task, and the previous work has acknowledged that it is an NP problem [BMS*16, MPO21]. In the segmentation methods like region growing, the degree of imbalance is influenced by the randomness of seed vertices and distances computed to the seeds, and these influences can not be adjusted as needed during iterative clustering, making it difficult to control the sizes of the sub-meshes. METIS [KK97], a state-of-the-art method for balanced mesh segmentation, uses the multilevel bisection method and Kernighan-Lin algorithm [KL70] to refine the segmentation result. However, due to the trade-off for achieving high performance with the multilevel method, the sub-meshes often have a serious non-manifold issue. Another approach proposed by Rahimian et al. [RPG*13] aimed at addressing the balance issue. They proposed a graph partitioning method named JA-BE-JA that can exactly balance the node counts and support customizable patch counts. Their method randomly swaps a node with one of the neighbor nodes or from a random set of the nodes to the graph patches in balance at the initial stage to guarantee the node counts are exactly balanced. However, when applying JA-BE-JA for mesh segmentation, it has a severe discontinuity issue because the node swapping uses randomness.

Our approach provides a new solution for generating an exactly balanced mesh segmentation result while maintaining the topological continuity of the sub-meshes. The use of a convex spherical representation allows for fast computation of distance measurements, reducing the overall time complexity of the algorithms. The parallel processing of creating and updating sub-meshes makes the algorithms more efficient.

3. Cohesion Algorithm

The cohesion algorithm is designed to ensure that the sub-meshes created from randomly assigned labels are contiguous. The input of the algorithm is the spherically parameterized representation of a 3D mesh, with vertices presented in Cartesian coordinates on the sphere. Parameterizing the mesh onto a sphere requires the shape of the mesh to have a closed-manifold genus of zero, which is a constraint that is common to the general concept of sphere-based parameterization.

The algorithm uses a label-swapping approach between any two triangles. This swapping process is carried out in parallel, and the pairs of triangles are evaluated based on chord distances between the centroids of the triangles and the centroids of the sub-meshes. Initially, all sub-meshes are assigned the same number of triangles, and the label-swapping operation preserves the triangle count. The next two subsections describe the key components of the algorithm. The first subsection explains the label-swapping method, and the second subsection describes a parallel processing technique for triangle pairs that eliminates any occurrence of swapping and accessing conflicts.

3.1. Swapping Labels

In order to ensure that the labels of the same sub-mesh are as compact as possible, The cohesion algorithm aims to minimize the geodesic distance between the triangles of the same sub-mesh. However, computing the geodesic distance in the mesh surface domain can be prohibitively expensive. To address this issue, an efficient estimation of the geodesic distance can be carried out in the spherical domain. This approach allows the algorithm to estimate the geodesic distance between triangles of the same sub-mesh in a computationally efficient manner, enabling the algorithm to improve the compactness of the sub-mesh labels.

Let's consider two triangles in the spherical domain, t_a and t_b, where $t_a \in sm_{l_a}$ and $t_b \in sm_{l_b}$, and l_a is the sub-mesh (sm) label of t_a, and l_b is the sub-mesh label of t_b ($a \neq b$). If $l_a \neq l_b$, then t_a and t_b potentially need to swap their sub-mesh labels. Algorithm 1 describes the method for swapping the labels, which involves a distance comparison condition to determine whether or not t_a and t_b should swap the labels (lines 5-9). The distance d_i represents the chord distance ($|\cdot|$) between the centroid of a triangle ($t_i.ctr$) and the centroid of a sub-mesh ($sm_i.ctr$). In this context, the centroid of the triangle is projected onto the sphere surface, while the centroid of a sub-mesh is the mean position of the centroids of the triangles in this sub-mesh, also projected onto the sphere surface.

Algorithm 1

```
1:  procedure CohesiveSwap(two triangles: $t_a$, $t_b$)
2:      $l_a \leftarrow t_a$'s sub-mesh label;
3:      $l_b \leftarrow t_b$'s sub-mesh label;
4:      if $l_a \neq l_b$ then
5:          $d_1 \leftarrow |t_a.ctr - sm_{l_a}.ctr|$;
6:          $d_2 \leftarrow |t_b.ctr - sm_{l_b}.ctr|$;
7:          $d_3 \leftarrow |t_a.ctr - sm_{l_b}.ctr|$;
8:          $d_4 \leftarrow |t_b.ctr - sm_{l_a}.ctr|$;
9:          if $d_1 + d_2 > d_3 + d_4$ then
10:             $t_a$'s sub-mesh label $\leftarrow l_b$;
11:             $t_b$'s sub-mesh label $\leftarrow l_a$;
12:         end if
13:     end if
14: end procedure
```

Calculating the geodesic distance involves a significant time complexity of $O(n^2)$ as it requires finding the shortest geodesic path between the centroids in the origin mesh surface domain. Although previous studies [SC20, HYYZ17, SSK*05] have made efforts to reduce the computation time of geodesic distance, its time complexity remains high. Our approach utilizes the chord distance in the spherical domain, which, although not as precise as the geodesic distance, provides a more efficient alternative. The chord distance allows for quick and correct distance comparisons and facilitates the determination of whether the labels of two triangles need to be swapped, all accomplished in constant time $O(1)$. The shorter the chord distance, the more suitable the triangle is labeled with this sub-mesh. To this end, we sum the distances from the centroids of the triangles to the centroids of their current sub-meshes ($d_1 + d_2$ in Algorithm 1). If this sum is greater than the sum of distances after swapping ($d_3 + d_4$), swapping the labels can result in a more closely cohered group.

In certain extreme scenarios, the cohesion algorithm may encounter situations where no progress can be made. One such example is when all centroids of sub-meshes coincide at the same position. Although this situation did not arise in our experiment with test models, it is worth noting as it has a potential to cause stagnation. In such cases, a possible solution could involve introducing a one-time random offset to shift the centroids of sub-meshes. By dispersing the centroids to different positions, the algorithm would regain the ability to make progress and overcome the stagnation.

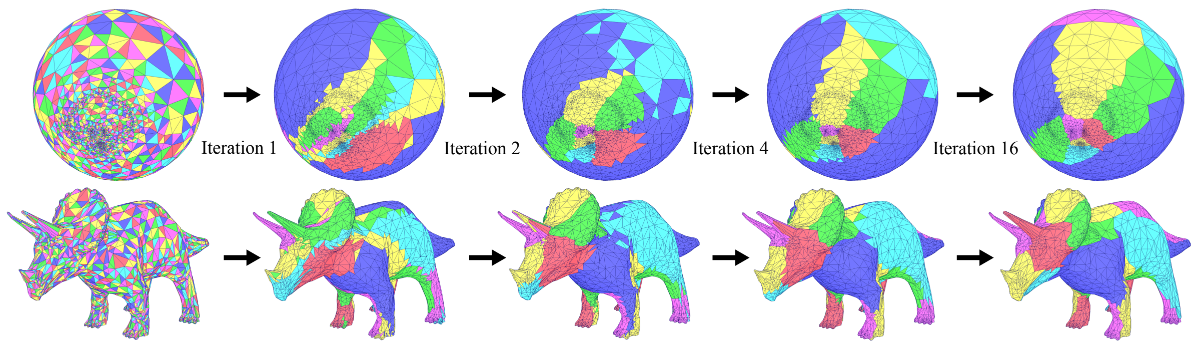

Figure 2: *Intermediate label-swapping results of different iterations during the cohesion process with Triceratops model. The number of sub-meshes is 20. The leftmost image shows the random initialization result. The rest images show the results after 1, 2, 4, and 16 iterations, respectively. The process stops after 16 iterations (reaching the minimal segmentation energy of the mesh).*

3.2. Parallelization with Round Robin Scheduling

All unique pairs of triangles need to be evaluated for swapping labels until no two triangles can swap labels any more. This iterative process can be compared to cohesion, as it aims to adhere nearby triangles together through the minimization of the distance energy $E_{mesh} = E_{sm_0} + E_{sm_1} + \cdots + E_{sm_{m-1}}$. Here, E_{sm_i} represents the energy of a sub-mesh, which is calculated as the sum of the distances between the centroids of each triangle in the sub-mesh and the centroid of the sub-mesh itself. Each swapping operation reduces the energy. When a triangle does not connect directly with another triangle of the same sub-mesh, there must be a path connecting them through one or more triangles that belong to other sub-meshes. The distance comparison condition can detect whether swapping the labels between this triangle and a triangle on the path will decrease the energy, thus ensuring the overall energy is minimized when the iteration stops. Figure 2 shows the intermediate results of several iterations.

Algorithm 2 Parallel cohesion

```
1:  procedure COHESION(all triangles: $T = \{t_0, \cdots, t_i, \cdots, t_{n-1}\}$)
2:      define the triangle index array $U = \{u_0, \cdots, u_i, \cdots, u_{n-1}\}$;
3:      for $i = 0$ to $n-1$ in parallel do ▷ initialize the index array
4:          $u_i \leftarrow i$;
5:      end for
6:      for $i = 0$ to $n-1$ rounds do
7:          for $j = 0$ to $\frac{n}{2} - 1$ in parallel do
8:              CohesiveSwap($t_{u_j}, t_{u_{n-1-j}}$);        ▷ Algorithm 1
9:          end for
10:         for $j = 1$ to $n-1$ in parallel do   ▷ Cyclic rotation of the indices
11:             $u_j \leftarrow u_j - 1$;
12:             if $u_j < 1$ then
13:                 $u_j \leftarrow n-1$
14:             end if
15:         end for
16:     end for
17:     Update all sub-mesh centroids in parallel with prefix sum;
18: end procedure
```

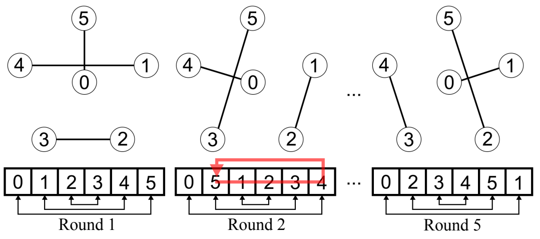

Figure 3: *Example of a schedule constructed by circle method. In the first round, element 0 is compared with element 5, element 1 is compared with element 4, etc. For each round, except for element 0, all elements will be shifted in the forward direction.*

We propose a parallel scheduling strategy aimed at reducing the execution time. It is a fine-grained parallelization to process triangle pairs. In our parallel algorithm design, a triangle is involved in only one label swapping evaluation to avoid potential access conflicts arising from multiple requests to swap the label. To achieve this, we use the theory of round robin tournaments [HM66]. Essentially, each triangle can only be paired with one other triangle in each tournament round. For instance, if the triangle pair (t_0, t_1) is evaluated for label swapping, t_0 and t_1 can not be in other pairs in the same round. As a result, the triangle pairs in the tournament round can be evaluated concurrently.

The parallel cohesion algorithm for one iteration is shown in Algorithm 2. The triangle index array U is initialized to the default triangle order in lines 3-5, and it is used to track the unique pairs of triangles for each tournament round of the swapping evaluation. Specifically, the set of triangle pairs $\{(t_{u_j}, t_{u_{n-1-j}})\}$ for $j \in [0, \frac{n}{2} - 1]$ can be evaluated in parallel without access conflict in lines 7-9. In the first round, this set pairs the first and last triangles, the second and the second-last, and so on towards the two middle triangles. After that, the circle method (lines 10-15) is employed to update the index array so that each triangle can be paired with a different one in the next round. The circle method shifts the index values in the forward direction by one array element in line 11. This cyclic rotation starts with the second element of the index array (u_1) and the value of the first element u_0 remains unchanged. Figure 3 provides an example of the circle method. For n triangles ($n \ mod \ 2 = 0$), there will be $n-1$ tournament rounds with $\frac{n}{2}$ paralellizable triangle pairs each.

At the end of each cohesion iteration, the centroids of the sub-meshes are updated with sub-mesh-level parallelization (line 17). The centroid of sub-mesh s is the mean of the centroids of the triangles that belong to sub-mesh s. We used the pre-fixed sum method [HSO07] to calculate the mean of the centroids of the triangles.

4. Refinement

After the cohesion step, a common issue is the presence of "bowtie" non-manifold configurations, in which a single triangle becomes a boundary triangle and is no longer editable in primitive-level applications. Moreover, the resulting boundary edges may not be smooth, which can pose challenges in local domains. For instance, the additional "ribbon" triangles [MPO21] introduced at the boundaries may become bloated.

Our refinement approach aims to address the bowtie issue and smooth out the boundaries of the segments by minimizing the number of boundary edges through refining the label assignment obtained from the cohesion process. The refinement process also involves an iterative label swapping process, similar to the cohesion process, to ensure that the triangle counts of the sub-meshes are not altered.

Algorithm 3

```
 1: procedure REFININGSWAP(two triangles: $t_a$, $t_b$)
 2:     $l_a \leftarrow t_a$'s sub-mesh label;
 3:     $l_b \leftarrow t_b$'s sub-mesh label;
 4:     $e_1 \leftarrow cost(t_a, sm_{l_a})$;                ▷ Equation 1
 5:     $e_2 \leftarrow cost(t_b, sm_{l_b})$;                ▷ Equation 1
 6:     $e_3 \leftarrow cost(t_a, sm_{l_b})$;                ▷ Equation 1
 7:     $e_4 \leftarrow cost(t_b, sm_{l_a})$;                ▷ Equation 1
 8:     if $e_1 + e_2 > e_3 + e_4$ then
 9:         $t_a$'s sub-mesh label $\leftarrow l_b$;
10:         $t_b$'s sub-mesh label $\leftarrow l_a$;
11:     end if
12: end procedure
```

Given two labeled triangles, t_a and t_b, Algorithm 3 determines whether or not swapping their labels, l_a and l_b, can refine the boundaries of the sub-meshes, in terms of alleviating the non-manifold triangle cases and reducing the number of boundary edges. The algorithm takes into account the neighboring edges of the corresponding sub-meshes of the two input triangles.

To formulate the refinement cost, let us first define the function $\varphi(t_i, sm_j)$ as shown in Algorithm 4, which returns the number of the boundary edges of the sub-mesh sm_j that the triangle t_i has. A measurement takes into account both $\varphi(t_i, sm_j)$ and the number of boundary edges of the neighboring triangles. The refinement cost is calculated as the sum of squares for these two parts of this measurement, as expressed in Equation 1.

$$cost(t_i, sm_j) = \varphi(t_i, sm_j)^2 + \sum_{(t_k \in t_i.neighbors) \wedge (t_k \in sm_j)} \varphi(t_k, sm_j)^2 \quad (1)$$

Algorithm 4

```
 1: procedure φ(triangle: $t_i$, sub-mesh: $sm_j$)
 2:     $\{t_0, \cdots, t_k, \cdots, t_{q-1}\} \leftarrow t_i$'s neighboring triangles; ▷ $q = 3$
    in a 2-manifold mesh
 3:     $count \leftarrow 0$;
 4:     for $k = 0$ to $q-1$ do
 5:         $l_k \leftarrow t_k$'s sub-mesh label;
 6:         if $j \neq l_k$ then
 7:             $count += 1$;
 8:         end if
 9:     end for
10:     return $count$;
11: end procedure
```

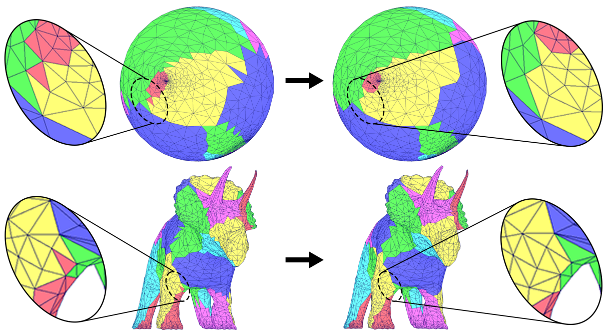

Figure 4: *The refinement example with the Triceratops model. The non-manifold issue (the red triangle connecting to the red sub-mesh through a single vertex) can be removed, and the boundaries of the sub-meshes become smoother after the refinement.*

In Algorithm 3, the number of the boundary edges associated with the two input triangles with the current sub-mesh labels ($e1$ and $e2$) and the number of the boundary edges after the refining swapping ($e3$ and $e4$) are computed (lines 4-7). If the boundary edges become less after the swapping, the labels of the two triangles are swapped (lines 8-10).

For each iteration of the refinement, Algorithm 3 must be applied to all pairs of triangles. The parallelization of the refinement process is the same as the cohesion, including the round-robin scheduling and the circle method to update the index array to pair the triangles in each round (Section 3.2), which would require $O(n)$ time. However, during the refinement process, the centroids of the sub-meshes are not used or updated. Figure 4 shows an example of the sub-meshes before and after refinement. The refinement process minimizes the total number of boundary edges so as to smooth the boundaries and mitigate the non-manifold issue in all the sub-meshes.

5. Implementation and Evaluation

We have implemented a progressive spherical parameterization method that allows us to preprocess meshes into spherical representations. As illustrated in Figure 5, the method starts by simplifying the mesh into a base shape, which is a tetrahedron, using the half-edge collapsing process from Peng and Timalsena's work [PT16].

Table 1: *The segmentation results of 21 test models. Each model was segmented into 200 sub-meshes.* ***CV*** *is the coefficient of variation of all the sub-meshes. and* ***MTCD*** *stands for the max triangle count difference between any two sub-meshes.* ***# of Bowtie*** *is the total number of the "bowtie" non-manifold shapes, and* ***# of BE*** *is the total number of boundary edges of the sub-meshes.*

		Our Approach				**Region Growing**				**METIS**			
Test Models	**# of Triangles**	**CV**	**MTCD**	**# of Bowtie**	**# of BE**	**CV**	**MTCD**	**# of Bowtie**	**# of BE**	**CV**	**MTCD**	**# of Bowtie**	**# of BE**
Elf	68,384	0.00	1	0	5,667	0.62	1,509	0	9,737	0.01	10	0	5,695
Monster	48,904	0.00	1	0	4,735	0.61	885	0	8,049	0.01	6	0	4,835
Armadillo	9,996	0.00	1	0	2,033	0.39	169	0	2,985	0.02	3	45	2,149
Bunny	3,996	0.01	1	0	1,303	0.34	41	0	1,695	0.02	2	143	1,447
Cow	5,804	0.00	1	0	1,580	0.34	57	0	2,112	0.02	2	19	1,631
David	10,390	0.00	1	0	1,963	0.19	60	0	3,011	0.02	3	34	2,163
Gargoyle	6,322	0.00	1	0	1,538	0.28	60	0	2,192	0.03	3	14	1,702
Horse	17,332	0.01	1	0	2,657	0.27	105	0	4,198	0.02	5	0	2,903
Human	20,096	0.00	1	0	3,041	0.35	182	0	4,749	0.01	6	0	3,182
Man Head	20,658	0.00	1	0	2,805	0.29	171	0	4,838	0.02	6	13	3,205
Triceratops	4,996	0.01	1	0	1,459	0.44	55	0	1,947	0.01	2	62	1,607
Dolphin	3,730	0.03	1	0	1,264	0.47	62	0	1,634	0.03	1	67	1,381
Dog	3,200	0.00	0	0	1,199	0.30	26	0	1,458	0.05	2	27	1,229
Car	6,550	0.01	1	0	1,652	0.29	54	0	2,274	0.03	3	48	1,824
Merman	86,206	0.00	1	0	6,494	0.99	3,882	0	11,046	0.01	21	6	6,210
Volleyball	7,680	0.01	1	0	1,853	0.39	79	0	2,512	0.02	2	27	1,947
Troll	4,958	0.02	1	0	1,460	0.37	54	0	1,992	0.02	3	111	1,579
Exterminator	6,346	0.01	1	0	1,677	0.33	67	0	2,350	0.03	3	3	1,671
Sculpture 1	156,240	0.00	1	0	9,060	1.52	2,834	0	24,954	0.001	4	0	10,357
Sculpture 2	319,200	0.00	0	0	13,030	1.24	4,215	0	46,588	0.001	6	44	15,176
Rock	558,976	0.00	1	0	15,162	0.50	7,301	0	37,727	0.001	10	0	16,700

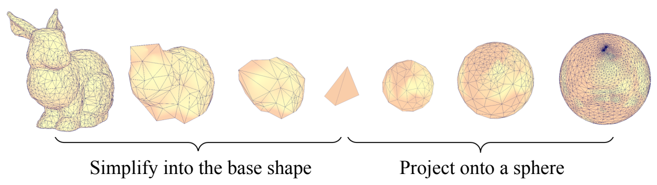

Figure 5: *An example of spherical parameterization in the preprocessing step. The Bunny model is simplified into a tetrahedron by collapsing edges progressively. The tetrahedron is projected onto the surface of a sphere, and vertices are inserted back one by one and projected onto the sphere.*

This process collapses one edge at a time, with the selection criterion designed to remove high curvature regions as early as possible. Our implementation evaluates all edges using this criterion in an edge-level parallel fashion. After collapsing the edges, we projected the tetrahedron onto a sphere and inserted the vertices back in the reverse order of collapsing. To ensure that each vertex is inserted to an optimal location on the sphere, we minimized the distortion measurement function introduced by Hu et al. [HFL17]. This allows us to find the least-distorted location inside the 1-ring region of the adjacent vertices for projection. To inquire the relationship between triangles directly, we built a dual graph of the triangular mesh [Del99], which encodes the topological structure of the mesh.

The cohesion and refinement algorithms have been implemented using C++ and CUDA version 11.6 on a machine equipped with an Intel(R) Core(TM) i9-10980XE CPU and an RTX 3090 GPU. We evaluated our approach using 21 different models, which are listed in Table 1. The segmentation results for these test models are presented in Figure 6 and Figure 7.

The spherical parameterization preprocess has been executed on the same machine used to evaluate our cohesion and refinement algorithms. The time complexity of our parameterization is $O(n\log n)$, where n represents the number of triangles in the mesh. For instance, the parameterization time is 21.27 seconds for the Human model, which consists of 20,096 triangles. This time is 71.26 seconds for the Monster model, composed of 48,904 triangles. In existing state-of-the-art spherical parameterization methods, the models used for evaluation are usually limited to around 350 thousand triangles [HFL17, CQW*19]. Advancing the existing methods, our implementation was able to parameterize the Rock model, composed of 558,976 triangles, but it was not successful in handling models with more than one million triangles due to the accumulation of area distortions, limited parametric space, and floating-point numerical errors.

The dual graph generation is fast, benefiting from the implementation with a triangle-level parallelization. For instance, the Human model takes 9.27 milliseconds, while the Monster model takes 22.61 milliseconds for dual graph generation.

To compare the segmentation results of our approach, we conducted experiments with two other methods: a region growing implementation based on the existing algorithms [LDB05, YGZS05], and the METIS software package. The region growing implementation selects triangles randomly as the seeds, with the number of

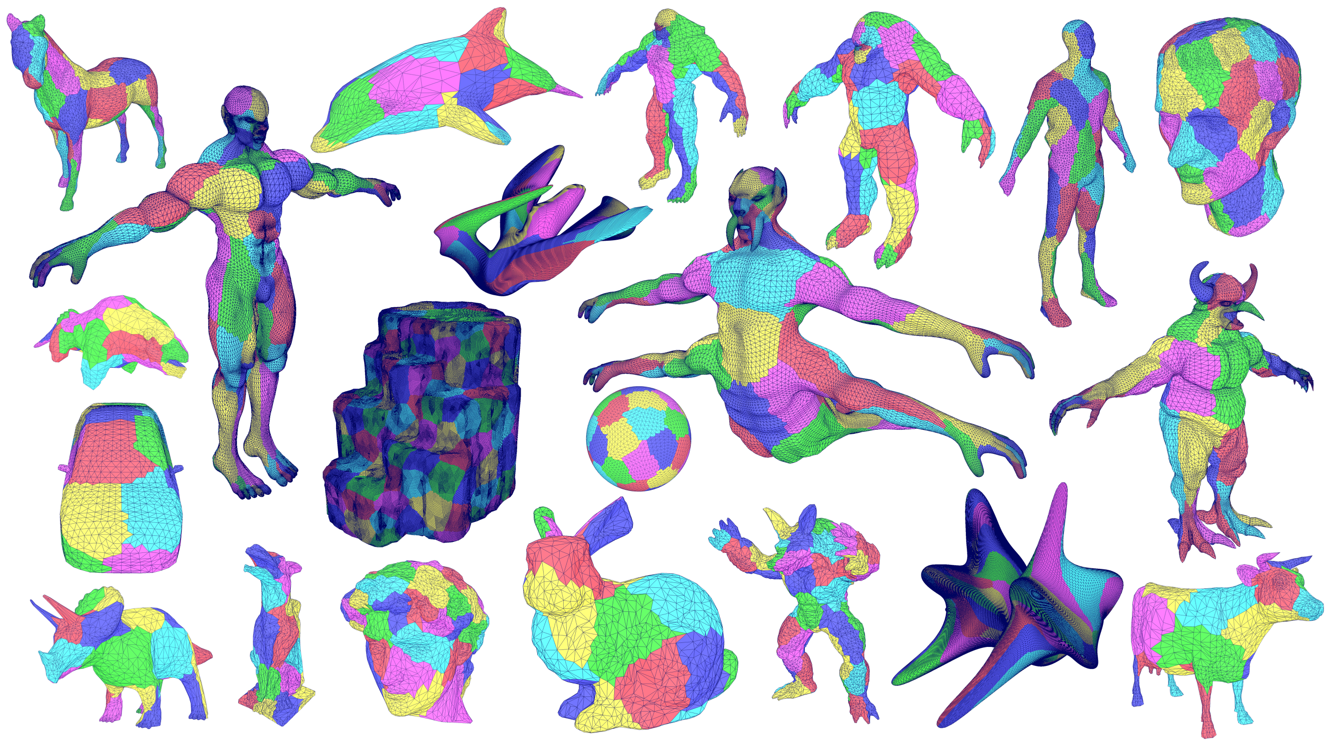

Figure 6: *The segmentation results of the test models produced by our approach.*

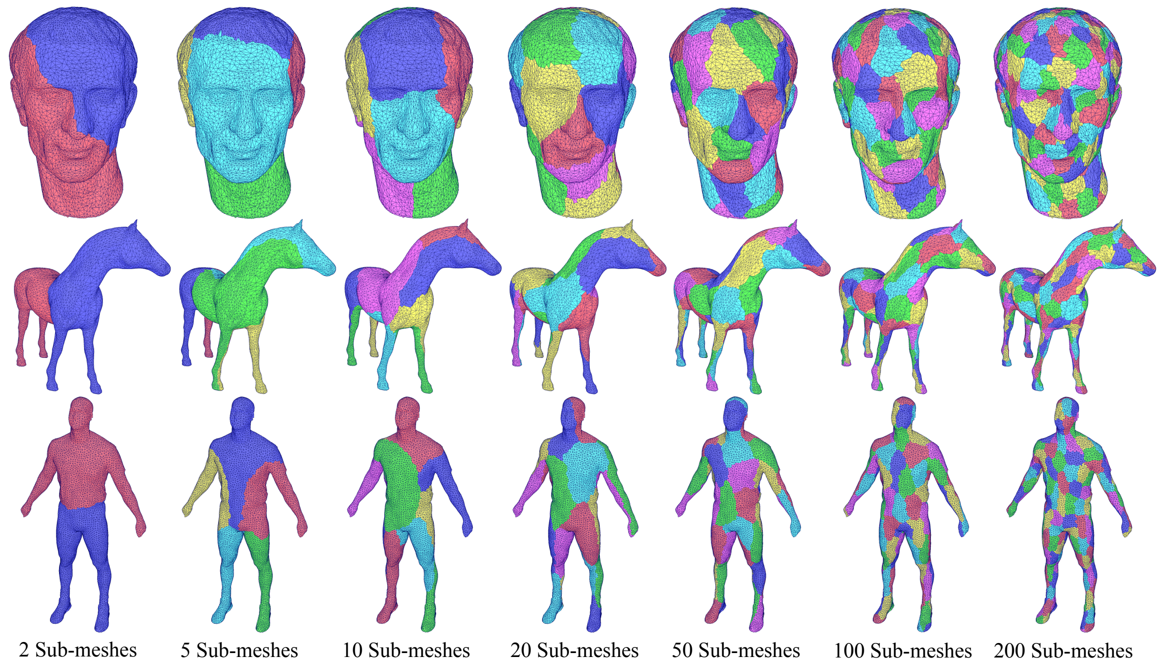

Figure 7: *The segmentation results of Man Head, Horse, and Human models with 2-200 sub-meshes produced by our approach.*

seeds equals to the number of the sub-meshes. During the growing iteration, unassigned triangles in the neighborhood of a region are evaluated based on the distance to the center of the region. The triangle with the lowest distance that does not cause a non-manifold issue is added to the corresponding sub-mesh. For the METIS software package, we enabled the option "-contig" to produce contiguous mesh segmentation results. While the "-ufactor" option is available in METIS to define the maximum permissible load imbalance among sub-meshes, we found that it was unable to achieve precisely balanced mesh segmentation results, even when we set "-ufactor" to the minimum allowed value of 1.001. Modifying "-ufactor" to a lower value may also increase the edge-cut in the mesh segmentation results. Therefore, we relied on the default settings provided by METIS and did not specify the "-ufactor" option.

5.1. Triangle Count Balancing among Sub-Meshes

In order to evaluate the quality of triangle count balancing, we used two metrics: the coefficient of variation (CV) [Bro98] and the maximum triangle count difference (MTCD) among all the sub-meshes. The CV is defined as the ratio of the standard deviation to the mean of the triangle count, and in our experiment, it was used to normalize the comparison on the test models since they have different geometric complexity. The MTCD is defined to express the largest difference in the triangle count between any two sub-meshes.

Table 1, shows that our approach has achieved exactly balanced results (MTCD = 0 or 1). The reason of MTCD = 1 on some of the test models is that the total number of triangles can not be evenly divided by the number of sub-meshes. Therefore, some sub-meshes must have one extra triangle compared to other sub-meshes. On the other hand, the region growing method results in high rates of CV (ranging from 0.19 to 1.52) and MTCD (ranging from 26 to 7,301). While METIS performs better than the region growing method in terms of triangle count balancing, it can not achieve exact balancing, resulting in a varying rate of MTCD (ranging from 1 to 21). This is because the METIS algorithm only balances the load at each partition, resulting in a locally optimized configuration, and the global balancing among all sub-meshes is not guaranteed. In contrast, our approach mandates a balanced sub-mesh label assignment initially, and the subsequent cohesion and refinement processes ensure that such balanced triangle counts are not altered.

5.2. Topological Continuity and Boundary Smoothness

We counted the number of "bowtie" non-manifold shapes and the number of boundary edges in all sub-meshes, listed in Table 1.

In various graph partitioning and mesh segmentation techniques, the edge-cut is frequently used as a metric to evaluate the quality of partition or segmentation, as discussed in the previous studies [KK97, RPG*13]. The edge-cut represents the number of edges that traverse distinct patches, reflecting the communication volume between these patches. In our triangle-based mesh segmentation approach, the edges that separate various sub-meshes do not intersect the sub-meshes. Instead, they define the boundaries between the sub-meshes. Therefore, we used an alternative term, number of boundary edges (# of BE), to describe this metric. It is important to note that the number of boundary edges corresponds to the edge-cut in the mesh's dual graph. Thus, we employ the number of boundary edges as a metric to evaluate the quality of our mesh segmentation results. A smaller number of boundary edges indicates better roundness and less communication volume between sub-meshes.

The roundness of the sub-meshes produced by our approach and the METIS is similar, with an average boundary edge ratio (# of BE / total number of edges) of 13.1% and 14.0%, respectively. However, METIS produces "bowtie" shapes, as shown in Figure 8. There is no clear trend indicating that the non-manifold issue from METIS becomes more severe with more complex models or an increasing the number of sub-meshes. For example, the Bunny model, which has just 3,996 triangles, has 143 "bowtie" shapes on the 200 sub-meshes, which is the highest among all the test models. In contrast, the Sculpture 2 model with 319,200 has only 44 "bowtie" shapes. The Merman model has 74 "bowtie" shapes on the 20 sub-meshes, but this decreases to 6 on the 200 sub-meshes. This is because the Kernighan-Lin algorithm, which is at the core of METIS, only finds a local optimum and causes an edge-level discontinuity between partitions. The region growing method results in a large number of boundary edges, and although it maintains continuous topology (no "bowtie"), the sub-meshes have unbalanced triangle counts and poor roundness (average boundary edge ratio = 19.0%).

5.3. Iteration Analysis

We analyzed the behavior of our approach in terms of the number of boundary edges and the execution time over the iterations. As shown in Figure 9, for the test case of the Human model (200 sub-meshes), the cohesion process converged after 27 iterations, which has reached the condition of no more label swapping, and the refinement process converged after 5 iterations, which has reached the condition that the number of boundary edges stopped decreasing. The total number of boundary edges and the current iteration's execution time decrease rapidly at the beginning. For instance, the number of boundary edges went down from 29,670 to 3,888 in the first 6 iterations of the cohesion process. Then, the decrease rate slows down and eventually reached stability. In total, the cohesion process took 4.05s and the refinement process took 1.84s to complete the label swapping.

6. Applications

The resulting sub-meshes from our approach store vertices and triangles in arrays, which are suitable to conduct remeshing-related run-time geometry processing applications. We applied the sub-meshes in two such applications: parallel continuous level-of-detail (LOD) selection and multi-threaded mesh compression decoding.

6.1. Level-of-Detail

We utilized the GPU's shared memory and implemented a parallel level-of-detail selection scheme using CUDA version 11.6 for all the 200 sub-meshes of each model. Each sub-mesh was simplified individually by one CUDA block based on a set of common feature-preserving edge-collapsing criteria [FKY*10, PC12, LSW*22].

To assess the capability of the sub-meshes to be simplified, we

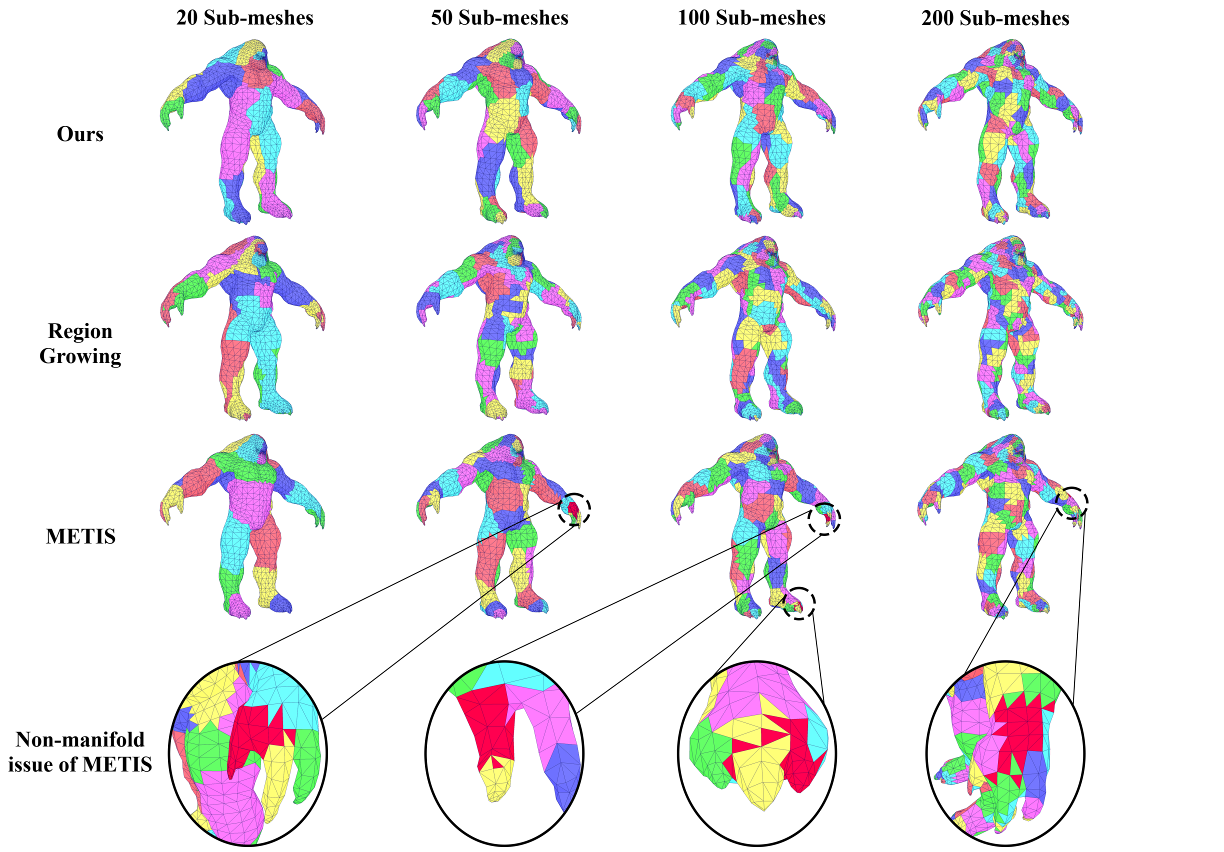

Figure 8: *The comparison of our approach, region growing, and METIS with 20, 50, 100, 200 sub-meshes. The first row is the results from our method. The second row is the results generated by region growing method. The third row shows the results created by METIS. In the last row, examples of the non-manifold issue in the results of METIS are highlighted.*

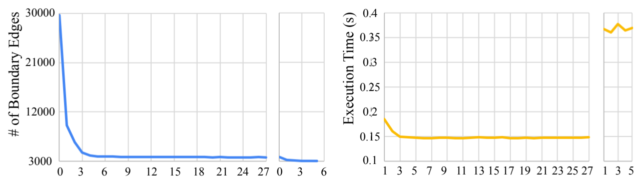

Figure 9: *The number of boundary edges (left chart) and the execution time for each step (right chart) changing over the iterations of the cohesion process (left side of x-axis) and refinement process (right side of x-axis) with the Human model. 0 in the x-axis means the initial # of boundary edges before the cohesion and refinement process.*

evaluated their non-collapsibility ratio, which we defined as the percentage of non-collapsible edges in all the sub-meshes over the total edges of the mesh. A lower ratio implies a better capability for simplification. However, since the boundary edges of the sub-meshes must be preserved to maintain topological continuity crossing neighbor sub-meshes, the edge-collapsing criteria can only collapse edges containing non-boundary vertices. Figure 10 (top) shows the results of our evaluation. We found that the sub-meshes generated by the

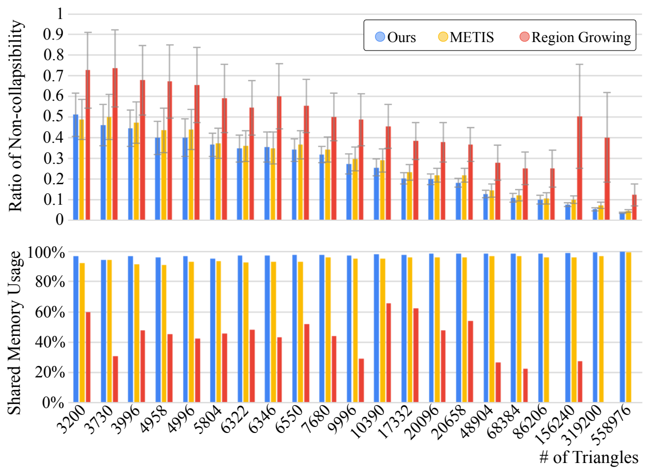

Figure 10: *The charts showing the ratio of non-collapsibility (top) and the shared memory usage (bottom). Note that, there is no red bar for the Merman (86,206), Sculpture 2 (319,200), and Rock (558,976) models, because the largest sub-mesh of these three models generated by the region growing method can not fit into the shared memory.*

region growing method had the highest ratio of non-collapsibility in all test models. This is mainly due to the highly irregular shapes of the sub-meshes looped with complex boundaries that the region growing method produces. Our approach achieved an overall lower ratio of non-collapsibility than METIS, except for two test models with small numbers of triangles (3200 and 6346). For all other test models, our approach gained clearly lower ratios. Specifically, the ratio from our approach was 11.3% lower than METIS on average, and the standard deviation was 38.2% smaller than METIS. The smaller standard deviation indicates more balanced capability among the sub-meshes for simplification. The results of METIS were affected by the non-manifold issue, which compromised the visual quality of the simplified sub-meshes, as shown in Figure 11. In contrast, the results of our approach did not have the non-manifold issue.

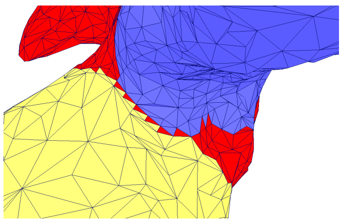
Figure 11: *The "bowtie" shapes (red) resulting from METIS prevent edges from collapsing.*

We also evaluated the usage of shared memory and the execution time of the LOD selection. All the CUDA blocks must be allocated with the same shared memory size. If the sizes of sub-meshes are not balanced, the shared memory of a CUDA block is underutilized to a greater extent when the sub-mesh size is smaller. Figure 10 (bottom) shows that our approach resulted in almost full utilization of the shared memory (only a small influence due to the difference in the number of vertices between sub-meshes). METIS achieved shared memory utilization that is comparable to ours, with an average difference of only 2.7%. This is due to the consideration of load balancing during the partitions in METIS. Figure 12 (left) shows the execution time on the test models. For the test models with 48k or more triangles, on average, the region growing method and METIS were 29.1% and 6.8% slower than our approach, on average. For smaller models, the cost of launching the CUDA kernel was the major cost, and the computing time was small so as not to make the difference significant. For the sub-meshes of the Merman, Sculpture 2, and Rock models generated by the region growing method, the desired amount of memory was larger than the maximum amount of shared memory that can be allocated on the GPU, so their execution time evaluation was not included in the comparison.

6.2. Mesh Compression

Mesh compression is a widely used application that can benefit from mesh segmentation, as noted in previous studies [CL18, KG00, ZTT12]. To test this, we utilized Google's Draco [Goo23] mesh compression library to encode and decode the meshes that were segmented into 20 sub-meshes. The Powershell ForEach-Object Parallel feature was used to conduct parallel execution. Figure 12 (right) shows the execution time of the decoding process. The difference in decoding execution time is more noticeable when testing with larger models. For models with 68k or more triangles, our approach resulted in an average decoding performance that was 7% faster than the region growing method. The decoding performance with results

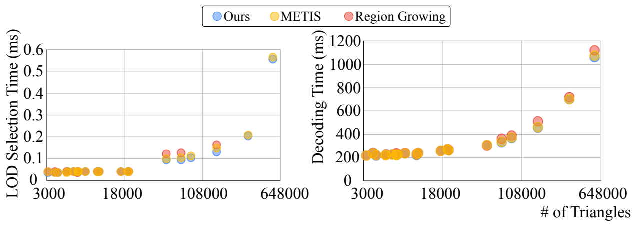

Figure 12: *The execution time of the LOD selection (left) and the mesh compression decoding (right).*

from METIS was similar to using the results from our approach because both methods tended to balance the triangle counts among the sub-meshes.

7. Conclusion

This work has presented a new approach to mesh segmentation that surpasses the capabilities of existing methods such as region growing and METIS, in terms of maintaining the topological continuity, balancing triangle counts, and improving sub-mesh shape regularities. To demonstrate the effectiveness of our approach, we implemented two parallel geometry processing applications on both CPU and GPU platforms. These applications were designed to showcase the enhanced performance that our segmentation technique provides. Through these experiments, we observed significant improvements in both the quality of the sub-meshes and the execution time required to produce them.

One limitation of our approach is that it is currently restricted to genus-zero meshes due to the nature of spherical parameterization. Although genus-zero meshes are widely used in various applications, it would be desirable to extend our approach to higher genus meshes in the future. This could potentially be achieved by introducing seams or open boundaries to reduce the genus level, thereby enabling the mapping of higher genus meshes onto spheres for continuous and balanced mesh segmentation using our approach.

The refinement algorithm effectively resolves the "bowtie" issue for the sub-meshes of the test models in our experiment. However, if the original model is composed of highly irregular geometric primitives, it is possible that several adjacent sub-meshes grow into long and narrow shapes, which can have a low possibility of causing the refinement process to yield a local optimum. This scenario would require further investigation in the future.

Acknowledgements

We sincerely thank the anonymous reviewers for their valuable feedback and suggestions. We are also grateful for the support provided by the RIT MAGIC Center.

References

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3 (2006), 181–193. 2

[BAT12] BERGAMASCO F., ALBARELLI A., TORSELLO A.: A graph-based technique for semi-supervised segmentation of 3D surfaces. *Pattern Recognition Letters 33*, 15 (2012), 2057–2064. 2

[BMS*16] BULUÇ A., MEYERHENKE H., SAFRO I., SANDERS P., SCHULZ C.: *Recent advances in graph partitioning*. Springer, 2016. 2

[Bro98] BROWN C. E.: Coefficient of variation. In *Applied Multivariate Statistics in Geohydrology and Related Sciences*. Springer, 1998, pp. 155–157. 8

[CGF09] CHEN X., GOLOVINSKIY A., FUNKHOUSER T.: A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (tog) 28*, 3 (2009), 1–12. 1

[CL18] CHEN H.-K., LI M.-W.: A novel mesh saliency approximation for polygonal mesh segmentation. *Multimedia Tools and Applications 77*, 13 (2018), 17223–17246. 1, 10

[CQW*19] CUI L., QI X., WEN C., LEI N., LI X., ZHANG M., GU X.: Spherical optimal transportation. *Computer-Aided Design 115* (2019), 181–193. 6

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *ACM SIGGRAPH 2004 Papers*. 2004, pp. 905–914. 1

[Del99] DELINGETTE H.: General object reconstruction based on simplex meshes. *International Journal of Computer Vision 32*, 2 (1999), 111–146. 6

[FKY*10] FENG W.-W., KIM B.-U., YU Y., PENG L., HART J.: Feature-preserving triangular geometry images for level-of-detail representation of static and skinned meshes. *ACM Trans. Graph. 29*, 2 (apr 2010). URL: https://doi.org/10.1145/1731047.1731049, doi:10.1145/1731047.1731049. 8

[Goo23] GOOGLE: Google/draco: a library for compressing and decompressing 3D geometric meshes and point clouds that is intended to improve the storage and transmission of 3D graphics., 2023. URL: https://github.com/google/draco. 10

[GWH01] GARLAND M., WILLMOTT A., HECKBERT P. S.: Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D Graphics* (2001), pp. 49–58. 2

[HFL17] HU X., FU X.-M., LIU L.: Advanced hierarchical spherical parameterizations. *IEEE Transactions on Visualization and Computer Graphics 24*, 6 (2017), 1930–1941. 6

[HM66] HARARY F., MOSER L.: The theory of round robin tournaments. *The American Mathematical Monthly 73*, 3 (1966), 231–246. 4

[HSD00] HART P. E., STORK D. G., DUDA R. O.: *Pattern classification*. Wiley Hoboken, 2000. 2

[HSO07] HARRIS M., SENGUPTA S., OWENS J. D.: Parallel prefix sum (scan) with CUDA. *GPU Gems 3*, 39 (2007), 851–876. 5

[HYYZ17] HAN X., YU H., YU Y., ZHANG J.: A fast propagation scheme for approximate geodesic paths. *Graphical Models 91* (2017), 22–29. 3

[KG00] KARNI Z., GOTSMAN C.: Spectral compression of mesh geometry. In *Proceedings of the 27th annual conference on Computer Graphics and Interactive Techniques* (2000), pp. 279–286. 10

[KK97] KARYPIS G., KUMAR V.: METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1, 2, 8

[KL70] KERNIGHAN B. W., LIN S.: An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal 49*, 2 (1970), 291–307. 3

[KT96] KALVIN A. D., TAYLOR R. H.: Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications 16*, 3 (1996), 64–77. 2

[KYD*18] KHAN D., YAN D.-M., DING F., ZHUANG Y., ZHANG X.: Surface remeshing with robust user-guided segmentation. *Computational Visual Media 4*, 2 (2018), 113–122. 1

[LDB05] LAVOUÉ G., DUPONT F., BASKURT A.: A new CAD mesh segmentation method, based on curvature tensor analysis. *Computer-Aided Design 37*, 10 (2005), 975–987. 2, 6

[Llo82] LLOYD S.: Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. 2

[LLS*05] LEE Y., LEE S., SHAMIR A., COHEN-OR D., SEIDEL H.-P.: Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design 22*, 5 (2005), 444–465. 1

[LSW*22] LIANG Y., SONG Q., WANG R., HUO Y., BAO H.: Automatic mesh and shader level of detail. *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–12. doi:10.1109/TVCG.2022.3188775. 8

[MPO21] MAHMOUD A. H., PORUMBESCU S. D., OWENS J. D.: RXMesh: a GPU mesh data structure. *ACM Transactions on Graphics (TOG) 40*, 4 (2021), 1–16. 1, 2, 5

[PC06] PEYRÉ G., COHEN L. D.: Geodesic remeshing using front propagation. *International Journal of Computer Vision 69*, 1 (2006), 145–156. 1, 2

[PC12] PENG C., CAO Y.: A GPU-based approach for massive model rendering with frame-to-frame coherence. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 393–402. 8

[PT16] PENG C., TIMALSENA S.: Fast mapping and morphing for genus-zero meshes with cross spherical parameterization. *Computers & Graphics 59* (2016), 107–118. 5

[RMG18] RODRIGUES R. S., MORGADO J. F., GOMES A. J.: Part-based mesh segmentation: a survey. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 235–274. 1

[RPG*13] RAHIMIAN F., PAYBERAH A. H., GIRDZIJAUSKAS S., JELASITY M., HARIDI S.: Ja-be-ja: a distributed algorithm for balanced graph partitioning. In *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems* (2013), IEEE, pp. 51–60. 3, 8

[SC20] SHARP N., CRANE K.: You can find geodesic paths in triangle meshes by just flipping edges. *ACM Transactions on Graphics (TOG) 39*, 6 (2020), 1–15. 3

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1539–1556. 1, 2

[SSGH01] SANDER P. V., SNYDER J., GORTLER S. J., HOPPE H.: Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer Graphics and Interactive Techniques* (2001), pp. 409–416. 2

[SSK*05] SURAZHSKY V., SURAZHSKY T., KIRSANOV D., GORTLER S. J., HOPPE H.: Fast exact and approximate geodesics on meshes. *ACM Transactions on Graphics (TOG) 24*, 3 (2005), 553–560. 3

[STK02] SHLAFMAN S., TAL A., KATZ S.: Metamorphosis of polyhedral surfaces using decomposition. In *Computer Graphics Forum* (2002), vol. 21, Wiley Online Library, pp. 219–228. 2

[WLAT14] WANG H., LU T., AU O. K.-C., TAI C.-L.: Spectral 3D mesh segmentation with a novel single segmentation field. *Graphical Models 76*, 5 (2014), 440–456. 2

[XLXG11] XIAO D., LIN H., XIAN C., GAO S.: CAD mesh model segmentation by clustering. *Computers & Graphics 35*, 3 (2011), 685–691. 2

[YGZS05] YAMAUCHI H., GUMHOLD S., ZAYER R., SEIDEL H.-P.: Mesh segmentation driven by gaussian curvature. *The Visual Computer 21*, 8 (2005), 659–668. 2, 6

[ZTT12] ZHAO J., TANG M., TONG R.: Mesh segmentation for parallel decompression on GPU. In *International Conference on Computational Visual Media* (2012), Springer, pp. 83–90. 2, 10