Enabling Single Machine Topological Simplification with Distributed Preprocessing of Large Datasets

Nathan Blanken
blanken1@umd.edu
University of Maryland
USA

Abstract

This paper presents a method for topologically simplifying large triangle meshes using Apache Spark for parallel computation of Forman gradient and critical net. Actual simplification is performed on a single machine with an edge-collapse algorithm working against low-persistence arcs. The goal is to reduce topological data size with preservation of critical structure and lowered memory and compute requirements. Four datasets were used to evaluate the approach under threshold-based and percentage-based simplification. 24 experiments were done measuring time, memory usage, and Critical Net preservation. This paper demonstrates that a hybrid approach, distributed pre-processing with a single machine simplification, is able to do topological simplification to a certain extent, and outlines the requirements for further research in fully distributed simplification algorithms.

ACM Reference Format:
Nathan Blanken. 2025. Enabling Single Machine Topological Simplification with Distributed Preprocessing of Large Datasets. In . ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

1.1 Purpose

Topological simplification is a well-studied approach to decreasing the size of a mesh while avoiding removal or merging of significant topology features (e.g., ridges and valleys). This technique is important for reducing the complexity of topological structures while maintaining essential topographic features and fixing data issues such as filling in holes and filtering noise. This paper will not discuss the background of topological simplification but rather directs the reader to a number of papers that deeply analyze this topic. [1, 2]

When exploring the topological simplification, there is a desire to calculate a simple topology as quickly as possible. However, as datasets grow in size and complexity, maintaining computational efficiency becomes increasingly difficult. While researchers are constantly exploring ways to find more efficient algorithms, the only other area that can improve computing speeds is increasing computing power. Adding more computing power on a single machine can only offer marginal gains beyond a certain point. To achieve substantial reductions in processing time, it is necessary to leverage multiple machines to make a significant difference on computation time.

Distributing computations across multiple machines offers several key advantages, including a faster computation, increased tolerance to faults, and an ability to handle much larger data sets than a single machine can handle. If the goal of the simplification is to compute the result as quickly as possible, the limit is simply the number of machines that are brought into the cluster. By utilizing parallelization and load balancing, computations can be completed exponentially quicker than it being done on a single machine. Similarly, if the computation is done on a single machine and there is an issue with the calculation, there is a high risk of the entire computation failing. When working on multiple machines there is a lower risk in this case as one machine failing does not have a large impact on other machines in the cluster. Furthermore, by leveraging multiple machines, it is possible to process large datasets within timeframes comparable to those required for smaller datasets on a single machine, which can be quite helpful for large-scale projects.

1.2 Motivation

We investigate topological simplification for large triangulated terrains, aiming to enhance the efficiency of terrain analysis while preserving topologically significant features. To improve scalability, the simplification process is not performed entirely on a single machine. Instead, we employ an Apache Spark-based framework [**10747680**] to first compute the discrete gradient and extract the critical net of large triangulated terrains in a distributed manner. Subsequently, topological simplification is applied to the extracted critical nets on a single machine, significantly accelerating the process and improving overall scalability. This advancement brings us closer to efficient iceberg analysis by enabling faster computation of surface area and volume—key metrics for tracking iceberg dynamics and assessing their potential impact on global warming.

2 Background

To understand topological simplification, there are a number of concepts that need to be understood before simplifying terrain. Some key topics include Triangulated Irregular Networks, critical nets, and Discrete Morse Theory.

2.1 Triangulated Irregular Networks

Triangulated Irregular Networks (TINs) are a commonly used representation of 3D surfaces in a number of real-world applications such

XXX'25, April 2025, XXX, XXX

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

as terrain modeling, computational geometry, and geospatial analytics. Unlike other models that are raster-based, TINs are vector-based which make them quite helpful in displaying important terrain features. This is important because since they are irregular, they are able to show features that regular grid representations could not show. A disadvantage of TINs however is that if not stored efficiently, they can require significant amounts of storage to maintain the precise level of detail they hold.

There are a number of connectivity relations that need to be understood for TINs such as boundary, co-boundary, and adjacency relations. These boundaries are made of simplexes and facets. A k-simplexes is a convex hull of $k + 1$ independent points. As an example, a 0-simplex is a vertex whereas a 1-simplex is an edge and a 2-simplex is a triangle. Facets on the other hand is based on the vertices of a simplex. An h-facet is made up of $h + 1$ vertices of the simplex. For example, the vertices of a triangle are 0-facets and the edges of a triangle are 1-facets. When looking at these particular connectivity relations, they each play their own role in TINs. Boundary relations are defined as the relationship between a simplex and its facets. Co-boundary relations show the connection between a particular simplex within all of the simplexes and a facet. Finally, adjacency relations are the set of simplexes that share a facet. [2]

2.2 Discrete Morse Theory

As stated by Stephen Smale, the Morse-Smale complex is a topological data structure that shows the gradient of a scalar area. [13, 12] The Discrete Morse Theory states that a Morse-Smale Complex could be created by essentially applying a combinatorial adaptation of Morse Theory. By simplifying the classical Morse Theory, Discrete Morse Theory continues to maintain topological properties by assigning values to cells according to specific rules. An important aspect of Discrete Morse Theory is the idea of Forman gradients which show discrete gradient vector fields that are connected to a Morse function. Forman gradients are defined as the set of simplex pairs where a k-simplex of the triangle mesh is matched to a (k-1)-simplex or a (k+1)-simplex. It is also important to note that every simplex is only connected to one pair. [2] The pairs of these simplexes can be viewed as an arrow denoting the direction of the gradient. Simplexes that are not part of a pair are defined as a *critical* simplex. These unpaired simplexes are shown in the form of critical triangles (maxima), critical edges (saddles), and critical vertices (minima). These critical points and the gradient simplex pairs create the critical net.

Critical nets are a crucial part of the entire simplification process as all of the data and important characteristics of the topology are held within the critical net. Parts of the critical nets include three types of critical points. These critical points are *minima*, *saddles*, and *maxima*.

(1) *Minima* - A point where all surrounding terrain have a higher elevation.
(2) *Saddle* - A point where some surrounding terrain have a higher elevation and some surrounding terrain have a lower elevation. Typically, two opposite cardinal directions have increasing elevation and the other two opposite cardinal directions have decreasing elevation.
(3) *Maxima* - A point where all surrounding terrain have a lower elevation.

In a critical net, *maxima only directly connect to saddles* and *minima only directly connect to saddles*. Maxima *never* connect directly to minima. Connecting all of these critical points creates the critical net.
Looking at the critical net, there are natural *"regions"* that form between the critical edges of the critical net. These regions can be categorized into two different kinds of *"manifolds"* depending on the type of critical points particular regions are connected to. A set of regions could be called an *"ascending manifold"* if they surround a minima. On the flip side a set of regions could be called a *"descending manifold"* if they surround a maxima.

2.3 Distributed Systems

A distributed system is a network of connected computers that work together to solve a problem. Unlike typical single-machine systems, distributed systems split up workloads between multiple computers which allow for faster processing, higher ability to manage faults (should they occur), and an increased ability to scale. The basis, and many applications of, distributed systems can be found in the paper by L. Kleinrock. [5]
Apache Spark is a tool used for running processes on a cluster of distributed systems. According to the Apache Spark website, the tool can be used for a number of different applications that fall under the categories of data science, machine learning, analytics, and storage/infrastructure. Some features of Apache Spark that help it stand up are it's ability to do real-time data streaming, get quick analytics on SQL queries, and train machine learning models on a single machine which can scale to many fault-tolerant clusters. This tool was chosen because of its familiarity to researchers and previous work was done using this tool.

Within Apache Spark, Resilient Data Structures (RDDs) are an important data structure that show collections of immutable objects that are processed in a parallel fashion. RDDs allow users to create data transformations such as map, filter, and count. As stated in the name, this data structure is resilient because of its fault tolerance since it can reconstruct lost data through lineage. This data structure is not well optimized so it is important to manage it well.

This management can be done through another Apache Spark tool called DataFrames. DataFrames are similar to RDDs except they are stored more efficiently, have named columns, and an execution engine to optimize queries. Separately, DataFrames can handle SQL queries which allows users to easily process large data tasks. A key part of DataFrames too is that they easily distribute across nodes in a cluster which is important for the work done in this project.

Building off of DataFrames are GraphFrames. By taking graph algorithms and combining them with SQL optimizations, they find key patterns and allow new kinds of queries to be run. Users can use GraphFrames to find relationships in graph data through algorithms like shortest path and PageRank. At a high level, GraphFrames are represented as two DataFrames: one for the nodes of the graph (vertices) and another for the connections between the nodes (edges). Because of the flexibility of GraphFrames, they are an ideal choice for graph analytics in large datasets.

3 Related Work

There is a lot of previous work on topological simplification. The process of topological simplification itself can be done in a number of different ways. This paper chooses to do a version of edge collapse whereas other papers choose vertex clustering or vertex decimation.

Vertex clustering looks to provide a 3D approximation that accurately represents the original object but only has a fraction of the original faces, edges, and vertices. Because vertices are combined, faces are lost and a level of error is introduced. This is limited however by the user's control. According to the paper written by Rossignac and Borrel, this technique creates different "levels of detail" because of the number of 3D approximations that are made. [10]

The process of vertex decimation essentially looks at each vertex in a triangle mesh. At a high level, each vertex in the mesh is examined one at a time. While looking at a particular vertex, if the vertex meets particular criteria then it is deleted. Examples of the criteria are vertices that are more than a particular distance from another or have a particular angle. From there, if the vertex is removed then the remaining hold is filed using a local triangulation technique.[11]

When simplifying a mesh, choosing to do an edge collapse as the means of simplification is quiet simple as a concept however there are a number of conditions that must be satisfied in order for it to be a legal collapse. First, definitions of a boundary vertex and boundary edges must be set. A boundary edge is an edge that is a subset of only one face. A boundary vertex exists for a given vertex if that vertex is part of a boundary edge. Looking at the conditions for edge collapse, the first condition is that all adjacent vertices make a face with the edge in question and that face must be contained within the simplicial complex. The second condition is that both vertices in the edge must be boundary vertices. The third and final condition is that the simplicial complex must contain more than 4 vertices if neither vertex in the edge are boundary vertices or more than 3 vertices if either vertex in the edge are boundary vertices. Assuming all of these conditions are met, the edge collapse is a legal operation within the simplicial complex. This simple process of combining the two vertices (one edge) into one vertex (collapsed edge) is the basis for the topological simplification described in the following section. [4]

Another paper that focuses on topological simplification but is a bit outside the scope of what is covered in this paper is Gyulassy et al. [3] Gyulassy's team chose to focus on topological simplification for 3D scalar functions. The concepts required for this type of simplification are very similar to the concepts mentioned in this paper. In the paper by Gyulassy et al., it presents a new concept of saddle-saddle cancellation that simultaneously removes to saddles from the topology. This saddle-saddle technique is not used in our paper however it is something that could be researched further to determine its effectiveness for 2D triangular mesh. In this paper, all edge collapses are done on on maximum-saddle or minimum-saddle pairs.

4 Method

4.1 Computing Forman gradient and Critical Net using Apache Spark

In order to get to a polished and simplified topology, researchers must first start with a large raw dataset that is progressively simplified and reformatted. The steps taken to ultimately simplify a critical net are described in the section below however, we must first explain how we got there.

To improve the scalability of our methods of topological simplification on a single machine, in our work, the input is not the raw mesh triangulated from point clouds. Instead, we perform topological simplification over the critical net of huge triangle meshes. The critical nets are extracted through a distributed framework, Apache Spark, for mesh processing [8]. In this section, we explain how we compute the critical net of a triangle mesh by using the proposed Spark framework.

In the context of discrete Morse theory, the critical net is identified by tracing the gradient paths that link critical simplices—namely, critical vertices (local minima), critical edges (saddles), and critical triangles (maxima). To construct the Forman gradient, an established method grounded in homotopy expansion [9] was utilized and adapted for parallel execution using Apache Spark. The core mechanism of the algorithm involves imposing a total ordering I on the vertices of a triangle mesh Σ. This ordering serves as the basis for partitioning Σ into manageable sub-components, facilitating parallel gradient computation.

The total order is derived by arranging the vertices of the mesh according to the elevation. The order is extended from vertices to higher-dimensional simplices—edges and triangles—by defining $I(\sigma) := \max_{v \in \sigma} I(v)$, where σ represents a simplex and v is a vertex on its boundary. Based on this ordering, the complex is symbolically decomposed into units known as lower stars.

The *lower star* of a vertex v encompasses all simplices in the star of v that share the same index $I(v)$. Within each lower star, simplices are paired through the homotopy expansion process [9]. This pairing is performed incrementally by simplex dimension. Specifically, a k-simplex σ is paired with a $(k+1)$-simplex τ if σ has no unmatched faces and τ contains exactly one unmatched face, which must be σ. For an in-depth explanation of the methodology, the reader is referred to the foundational work by Robins et al. [9].

Once the Forman gradient is computed, we extract the critical net from the calculated gradient. The critical net forms a graph whose nodes correspond to critical simplices, and whose edges (or arcs) represent gradient paths linking critical simplices of successive indices. More precisely, a V_1-path associated with a critical edge e is formed by following a sequence of gradient pairs that connect e to nearby critical vertices. A V_2-path of a critical edge e is constructed by aggregating gradient-connected edges that link e to adjacent critical triangles.

Within the Spark framework, the resulting critical net is stored in two separate data files: the vertex-edge file and the edge-triangle file. The vertex-edge file encodes all critical simplex pairings between critical vertices and critical edges (i.e., minima to saddles), while the edge-triangle file captures the pairings between critical edges and critical triangles (i.e., saddles to maxima).

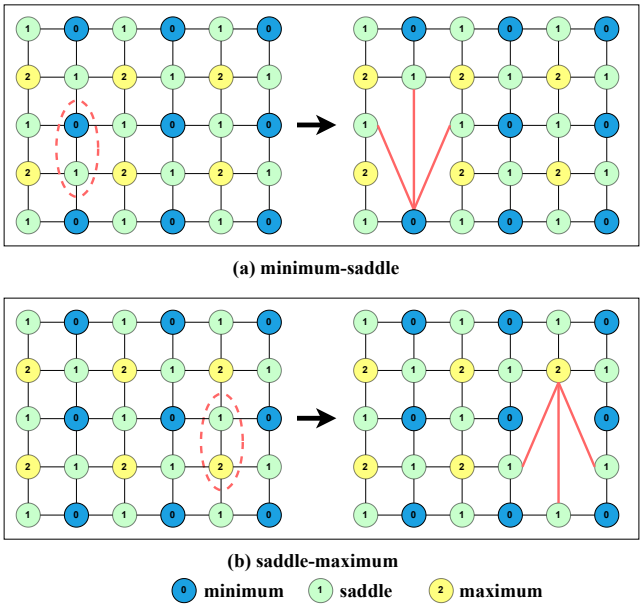

Figure 1: a) minimum-saddle depicts the removal of a critical edge to critical vertex arc and then the resulting critical connections that are created in the process. b) saddle-maximum depicts the removal of a critical triangle to critical edge arc and then the resulting critical connections that are created in the process.

4.2 Topological Simplification on a single machine

The data used to compute the single machine topological simplification were given as two files, one containing all of the connections between critical vertices and critical edges and the other file containing all of the connections between critical edges and critical triangles. These connections are known as *critical arcs*. The critical arcs in both files were inevitably combined into one file to more easily compute the total amount of data. It is important to note the difference in how a critical arc containing a critical vertex and critical edge (also known as a "minimum-saddle") is simplified versus a critical arc containing a critical edge and critical triangle (also known as a "saddle-maximum") is simplified. See Figure 1[7] for a visual of the specific simplification for each minimum-saddle and saddle-maximum.

To actually simplify the topology, there are several common operators such as vertex clustering, vertex decimation, and edge collapse. A more detailed version of each operator is described in section 3, related work. Vertex clustering is the process of removing points that are packed tightly together or points that are on top of one another so that there is only one remaining point in the nearby area. An example of this would be having multiple points that all map to one particular pixel in an image. It would be impossible to visualize all of the points that are on that one pixel so all but one of them are deleted. [6] Vertex decimation is the process of examining each vertex in a triangle mesh and determining if it is a candidate for removal based on a set of specific criteria. Once a point is removed, the hole that is now in the mesh is filled using local triangulation techniques. This continues until a certain level of simplification is achieved. [11] Edge collapse is the process of taking an edge and bringing both of its vertices together to make a single vertex. A certain set of conditions must be met for this to be a legal edge collapse. Those specific conditions are mentioned and proven in Hoppe et al. [4]

For this paper, we have chosen to focus on edge collapse as the simplification operator of choice. Edge collapse was chosen due to its adaptability for unique and varying topologies. At a high level, the simplification in this paper is done in a manner that focuses on collapsing arcs that have the lowest persistence value in the critical net. This simplification continues until either the persistence threshold is met or a certain number of arcs have been collapsed depending on the experiment being run. More information about experiments can be found in the next section.

In the following section of this paper, we present and discuss the pseudocode for the simplification algorithm step-by-step. It should be noted that the inputs and outputs are clearly specified to maintain reproducibility and transparency. The major steps of the pseudocode are as follows. First, the arc with the lowest persistence is popped from the priority queue. Once the current edge's persistence is found, the vertices adjacent to each vertex in the chosen edge are determined. After that, the edge in question is removed from the triangle mesh. Following that step, the critical arcs in the critical net are updated. The update process for this step focuses on matching minima with saddles and maxima with saddles. Minima and maxima cannot be directly connected so minima and maxima must be connected with saddles. Once each critical simplex pair has been matched, the persistence values for each of those critical arcs are determined and all of that information is added to the priority queue. Finally, all references of the simplified nodes and arcs are removed from the original lists. This entire simplification process is continued for each subsequent arc with the lowest persistence value until the threshold has been reached.

5 Designing the Implementation for Topological Simplification

When developing code for this project, a program was written for topological simplification carried out on a single machine. This program reads in the critical net, the *vertex-edge* file (mentioned above 4.1.2), and the *edge-triangle* file (mentioned above 4.1.2). The persistence of each critical edge is determined by calculating the difference in height for each element within each critical edge. This calculation can vary according to the elements within the critical edge. If the element is a *minimum*, it will simply be a single vertex. If the element is a *saddle*, there will be two vertices and the maximum elevation of the two vertices will be selected. If the element is a *maximum*, there will be three vertices and the maximum elevation of the three vertices will be selected. After the persistence is determined for all critical edges, the topological simplification algorithm is run as shown in *Algorithm 1*. Ultimately, these results are output into the final simplified form.

When looking specifically at single-machine simplification, there are a number of major steps that take place. Beginning with the vertex-edge and edge-triangle files, a priority queue is initialized

Algorithm 1 Single Machine Simplification

Input
A: Array of critical arcs; *Q*: Priority queue of critical arcs and their respective persistence values; *t*: Simplification threshold
Output
A: List of critical arcs

```
removedNodes = set()
connectedArcs = {arc : True for arc in A}
adjacency = defaultdict(set)
for arc in connectedArcs do
  for point in arc do
    adjacency[point].add(arc)
for count of t do
  persistence, currentArc = Q.pop()
  criticalPoint1 = currentArc[0]
  criticalPoint2 = currentArc[1]
  if criticalPoint1 in removedNodes or criticalPoint2 in removedNodes then
    continue
  removedNodes.update([criticalPoint1, criticalPoint2])
  secondConnections = []
  relatedArcs = set()
  relatedArcs.update(adjacency[criticalPoint1])
  relatedArcs.update(adjacency[criticalPoint2])
  for arc in relatedArcs do
    if arc not in connectedArcs or arc == currentArc then
      continue
    a0, a1 = arc
    if a0 in (criticalPoint1, criticalPoint2) or a1 in (criticalPoint1, criticalPoint2) then
      target = a1 if a0 in (criticalPoint1, criticalPoint2) else a0
      secondConnections.append(target)
      del connectedArcs[arc]
      adjacency[a0].discard(arc)
      adjacency[a1].discard(arc)
  newArcs = []
  if len(secondConnections) > 1 then
    for each ele in secondConnections do
      if len(ele) == 1 or len(ele) == 3 then
        for each secondEle in secondConnections do
          if ele != secondEle and len(secondEle) == 2 then
            persistence = abs(max(ele) - max(secondEle))
            newArcs.append((persistence, (ele, secondEle)))
  for each pair in newArcs do
    Q.append(pair)
    for pt in pair do
      adjacency[pt].add(pair)
return A
```

to manage all critical simplex pairs, ordered by their persistence values. The algorithm proceeds by repeatedly removing the arc with the lowest persistence from the queue. Then, the current lowest persistence arc is popped from this priority queue. For each selected arc, the algorithm identifies all critical simplices directly adjacent to it. Once these neighboring simplices are located, the arc is removed from the critical net. Subsequently, the set of critical simplex pairs is updated to reflect this change, and the priority queue is reordered to maintain the correct prioritization based on updated persistence values. This iterative process continues until a specified threshold is reached. The threshold can either be a particular persistence value between a critical simplex pair or a certain percentage of arcs that are removed. This variable is tested in the experimental section below.

6 Experimental Evaluation

6.1 Experimental Datasets

This section dives into the specific datasets that were used for this paper's evaluation. Four large datasets were selected that initially exceeded the memory capacity and/or computational limits of a single machine which is the motivating factor for switching certain computations to distributed systems as mentioned in earlier sections. The critical nets of each of these large datasets were computed in a distributed system.

6.2 Evaluation Metrics

When running our experiments to simplify the critical nets, there are a number of metrics that were important to determine the effectiveness of the topological simplification algorithm. These metrics are defined below:

(1) Number of Removed Critical Nodes - The total number of critical vertices, edges, and triangles eliminated during the simplification process.
(2) Number of Removed Critical Arcs - The number of critical simplex pairs removed, specifically those linking critical vertices to critical edges and critical edges to critical triangles.
(3) Node Connectivity Preservation - Measured as the percent of unique total critical vertices, critical edges, and critical triangles that exist after topological simplification.
(4) Critical Arc Preservation - Measured as the percent of original arcs that still exist in the simplified critical net. Calculated as the number of critical arcs before simplification minus the number of critical arcs removed then divided by the original number of critical arcs in the critical net.
(5) Max Memory Usage (MB) - The max amount of memory used for a given experiment measured in Megabytes.
(6) Time for Simplification - Time required to run the sequential topological simplification on a single machine measured.

6.3 Experimental Results

All four datasets underwent six experiments each resulting in a total of 24 experiments. The experiments were split between percentage-based and threshold-based tests. Tables 2 through 7 show the simplification experiments on the datasets. Each of these experiments depict the results for the number of removed critical arcs, number of removed nodes, number of total critical arcs after simplification, number of new arcs, number of critical vertices after simplification, number of critical edges after simplification, number of critical triangles after simplification, the node connectivity preservation, arc connectivity preservation, max memory usage (MB) during the computation, and time to compute the simplification (in seconds

and minutes). For the node connectivity preservation computation, this is based on the sum of the total critical vertices, critical edges, and critical triangles that exist after simplification compared to the sum of total critical vertices, critical edges, and critical triangles that existed before simplification. The preservation of arc connectivity is a percentage of the original arcs that remained after the topological simplification process.

Looking specifically at Tables 2 through 4, these tables show percentage-based simplification experiments. Table 2 shows the results of a 1% critical arc simplification, Table 3 shows 10%, and Table 4 shows 25%. In each of these tables, a set number of critical arcs was removed that represents its respective percentage of total critical arcs present in the dataset before topological simplification. It is important to note that despite these experiments only removing a set percentage of critical arcs in the dataset, the *arc connectivity preservation* metric does not perfectly match the experiment's respective removal percentage because when one arc is removed, multiple new arcs can be created.

When examining tables 5 through 7, these tables display threshold-based simplification experiments. Table 5 are the results of removing any persistence values ≤ 10, table 6 shows the removal of persistence values ≤ 50, and table 7 depicts the removal of persistence values ≤ 100. For these threshold-based simplifications, a slight modification of the algorithm was needed to account for a continual simplification until all critical arcs under a certain persistence value were removed.

For each experiment, it should be noted that a relatively small percentage of critical vertices, critical edges, and critical triangles are removed from the overall critical net. This outcome arises because the simplification threshold is defined based on the percentage of critical arcs targeted for removal, not critical nodes. During each application of the edge collapse operator, significantly more critical arcs are eliminated compared to critical nodes. Figure 1 shows for each critical arc that is being removed, it originally has six adjacent critical arcs that connect to other critical vertices, critical edges, and critical triangles. When the critical arc in question is removed and new critical arcs are created to fill the void, there are now three new critical arcs. Including the critical arc that was removed, there were originally seven total arcs before simplification and after simplification there were only three. This example results in a net loss of four critical arcs and only two critical nodes (one critical vertex or critical triangle and one critical saddle). For each critical arc simplification, a similar result is seen with a larger number of critical arcs being removed than critical nodes.

Another observation found in this research is that for most datasets, there is a lower percentage of critical vertices after simplification compared to the percentage of critical triangles after simplification. We believe this is due to the datasets containing more critical edge to critical vertex (minimum-saddle) pairs than critical triangle to critical edge (saddle-maximum) pairs. Because of this imbalance of minimum-saddle to saddle-maximum arcs, it can be seen in Tables 2 through 7 that as more arcs are removed in the simplification process, they tended to be minimum-saddle arcs.

Despite these critical net datasets being smaller than the raw data, they were still quite large as seen by the size of the files in *table 1*. These large files required a significant amount of time and a powerful machine to compute the results. In order to process these files, a virtual machine was setup on Google Cloud Platform (GCP) with 768 GB of memory and 8 virtual N2D CPUs. At this size, this machine cost $6.55 per hour to run on Google's platform. This large amount of memory and CPUs were required to store the supplemental data structures needed to compute the simplification. Even with a large amount of memory and a computationally powerful CPUs, the simplification process still took a considerable amount of time to complete each experiment. Averaging data from all 24 experiments, the average time to complete topological simplification was 1,688 seconds (28.13 minutes) with an average memory usage of 131,300 MB.

7 Concluding remarks

To review, this paper presents the process of sequential topological simplification and highlights the need to perform topological simplification on a set of distributed systems. As described in the methods section of this paper, the Forman gradient and critical net are computed from the original dataset using Apache Spark. Subsequently, the critical net is simplified on a single machine using the algorithm described in *Algorithm 1*. The results of the single machine topological simplification process as mentioned above showed that these computations as computationally intense for a single machine as seen by the significant amount of memory and time needed. This paper's contribution to this domain is proven in the results of the 24 experiments that show how infeasible it is to host these computations on a single machine because of the resources needed and the associated cost to access a machine of the caliber needed for these calculations.

There were a number of limitations that were faced in this process of completing this research project. The largest limitation that was faced was getting access to a single machine powerful enough to run the simplification experiments. Luckily Google Cloud Platform (GCP) offered a comprehensive solution that allowed us to complete the experiments. Another limitation of this process was that it was difficult to get small enough datasets for the experiments. Although these datasets are multiple gigabytes in size, they are quite small compared to other datasets available. If a significantly larger dataset needed to be simplified, the process would likely take multiple hours or days if done on a single machine. When looking at future work, researchers could focus on creating a topological simplification algorithm that can be supported in a distributed set of systems. By creating an algorithm that can successfully spread out the topological simplification problem onto multiple machines, these calculations can be completed in a significantly shorter amount of time.

8 References

[1] Riccardo Fellegara et al. "Efficient computation and simplification of discrete morse decompositions on triangulated terrains". In: *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2014, pp. 223–232.

[2] Riccardo Fellegara et al. "Terrain trees: a framework for representing, analyzing and visualizing triangulated terrains". In: *GeoInformatica* 27.3 (2023), pp. 525–564.

[3] A. Gyulassy et al. "A topological approach to simplification of three-dimensional scalar functions". In: *IEEE Transactions on Visualization and Computer Graphics* 12.4 (2006), pp. 474–484. DOI: 10.1109/TVCG.2006.57.

[4] Hugues Hoppe et al. "Mesh optimization". In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 1993, pp. 19–26.

[5] Leonard Kleinrock. "Distributed systems". In: *Communications of the ACM* 28.11 (1985), pp. 1200–1213.

[6] Kok-Lim Low and Tiow-Seng Tan. "Model simplification using vertex-clustering". In: *Proceedings of the 1997 symposium on Interactive 3D graphics*. 1997, 75–ff.

[7] Yuehui Qian. "Efficient Representation and Analysis of Large-Scale Meshes Using Apache Spark". Doctoral dissertation proposal. University of Maryland, 2025.

[8] Yuehui Qian et al. "Efficient representation and analysis for a large tetrahedral mesh using Apache Spark". In: *2024 IEEE Topological Data Analysis and Visualization (TopoInVis)*. IEEE. 2024, pp. 1–11. DOI: 10.1109/TopoInVis64104.2024.00005.

[9] Vanessa Robins, Peter John Wood, and Adrian P Sheppard. "Theory and algorithms for constructing discrete Morse complexes from grayscale digital images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.8 (2011), pp. 1646–1658.

[10] Jarek Rossignac and Paul Borrel. "Multi-resolution 3D approximations for rendering complex scenes". In: *Modeling in computer graphics: methods and applications*. Springer, 1993, pp. 455–465.

[11] William J Schroeder, Jonathan A Zarge, and William E Lorensen. "Decimation of triangle meshes". In: *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. 1992, pp. 65–70.

[12] Stephen Smale. "Generalized Poincaré's conjecture in dimensions greater than four". In: *Topological library: Part 1: Cobordisms and their applications*. World Scientific, 2007, pp. 251–268.

[13] Stephen Smale. "On gradient dynamical systems". In: *Annals of Mathematics* 74.1 (1961), pp. 199–206.

Acknowledgments

I would like to thank Dr. Leila De Floriani and Yuehui Qian for their continued support throughout the duration of this research project. Through the rollercoaster that is research, both Dr. De Floriani and Yuehui guided me in the right direction and gave me advice when I needed it. Thank you both for all of your help!

9 Appendix

	Canyon	Dragons	Idaho	Big Creek
Number of Vertices in Original Dataset	48,988,726	91,034,766	113,060,804	151,339,499
Number of Triangles in Original Dataset	97,977,344	182,069,460	226,121,529	302,678,963
Number of Critical Vertices	6,196,532	6,239,171	9,092,496	16,294,855
Number of Critical Edges	11,785,078	12,697,682	16,919,056	32,065,780
Number of Critical Triangles	5,588,547	6,457,516	7,826,561	15,770,926
Number of Total Critical Nodes	23,570,157	25,394,369	33,838,113	64,131,561
Number of Critical Arcs	46,536,448	50,383,368	67,151,291	126,879,162
Size of Critical Net File	2.49 GB	2.67 GB	3.7 GB	7.17 GB

Table 1: Basic information about the size of each dataset and its respective critical net before any topological simplification. The critical net is only encapsulated by the critical nodes and critical arcs. The critical nodes are the sum of critical vertices, critical edges, and critical triangles, while critical arcs refer to the critical simplex pairs connecting critical vertices and critical edges, and those connecting critical edges and critical triangles. It is important to note that the first two rows in the dataset show how large the original datasets are and how much smaller they become when looking solely at the critical net.

	Canyon	Dragon	Idaho	BigCreek
Number of critical arcs before	46,536,448	50,383,368	67,151,291	126,879,162
Number of removed critical arcs	465,364	503,833	671,512	1,268,791
Number of removed nodes	592,274	770,790	1,034,267	1,817,385
Percent of critical vertices after	94.71%	96.33%	96.46%	95.15%
Percent of critical edges after	97.86%	97.27%	97.24%	97.52%
Percent of critical triangles after	99.70%	99.67%	99.66%	99.65%
Node Connectivity Preservation	97.47%	97.65%	97.59%	97.45%
Arc Connectivity Preservation	99.00%	99.00%	99.00%	99.00%
Max Memory Usage (MB)	39,007	41,618	57,074	103,434
Time for simplification (sec)	349	373	592	896
Time for simplification (min)	6	6	10	15

Table 2: Results of 1% critical arc simplification for each dataset.

	Canyon	Dragon	Idaho	BigCreek
Number of critical arcs before	46,536,448	50,383,368	67,151,291	126,879,162
Number of removed critical arcs	4,653,644	5,038,336	6,715,129	12,687,916
Number of removed nodes	2,875,553	3,216,931	2,141,106	5,810,339
Percent of critical vertices after	93.11%	94.47%	96.01%	94.38%
Percent of critical edges after	90.75%	90.16%	94.41%	92.39%
Percent of critical triangles after	97.28%	97.27%	99.06%	98.28%
Node Connectivity Preservation	92.92%	93.03%	95.92%	94.35%
Arc Connectivity Preservation	90.00%	90.00%	90.00%	90.00%
Max Memory Usage (MB)	88,585	92,488	113,939	211,281
Time for simplification (sec)	1,137	1,237	1,691	2,364
Time for simplification (min)	19	21	28	39

Table 3: Results of 10% critical arc simplification for each dataset.

	Canyon	Dragon	Idaho	BigCreek
Number of critical arcs before	46,536,448	50,383,368	67,151,291	126,879,162
Number of removed critical arcs	11,634,112	12,595,842	16,787,822	31,719,790
Number of removed nodes	3,222,310	3,705,792	2,190,094	5,862,902
Percent of critical vertices after	92.44%	93.74%	95.96%	94.37%
Percent of critical edges after	90.07%	89.16%	94.31%	92.33%
Percent of critical triangles after	96.55%	96.52%	99.01%	98.22%
Node Connectivity Preservation	92.23%	92.16%	95.84%	94.30%
Arc Connectivity Preservation	75.00%	75.00%	75.00%	75.00%
Max Memory Usage (MB)	129,456	153,322	169,555	461,338
Time for simplification (sec)	1,843	1,823	2,541	8,060
Time for simplification (min)	31	30	42	134

Table 4: Results of 25% critical arc simplification for each dataset.

	Canyon	Dragon	Idaho	BigCreek
Number of critical arcs before	46,536,448	50,383,368	67,151,291	126,879,162
Number of removed critical arcs	11,892,473	5,293,541	1,502,640	4,532,254
Number of removed nodes	3,229,046	3,299,535	2,069,505	5,581,517
Percent of critical vertices after	92.43%	94.38%	96.08%	94.50%
Percent of critical edges after	90.06%	89.94%	94.56%	92.63%
Percent of critical triangles after	96.55%	97.06%	99.12%	98.37%
Node Connectivity Preservation	92.22%	92.84%	96.02%	94.52%
Arc Connectivity Preservation	74.44%	89.49%	97.76%	96.43%
Max Memory Usage (MB)	127,789	93,647	71,978	154,047
Time for simplification (sec)	1,691	1,089	794	1,655
Time for simplification (min)	28	18	13	28

Table 5: Persistence values removed that are ≤ 10 for each dataset.

	Canyon	Dragon	Idaho	BigCreek
Number of critical arcs before	46,536,448	50,383,368	67,151,291	126,879,162
Number of removed critical arcs	24,849,203	7,542,754	1,564,987	4,624,351
Number of removed nodes	3,371,938	3,496,664	2,088,461	5,607,235
Percent of critical vertices after	92.06%	94.08%	96.07%	94.49%
Percent of critical edges after	89.83%	89.60%	94.52%	92.60%
Percent of critical triangles after	96.31%	96.80%	99.10%	98.36%
Node Connectivity Preservation	91.95%	92.53%	96.00%	94.50%
Arc Connectivity Preservation	46.60%	85.03%	97.67%	96.36%
Max Memory Usage (MB)	173,644	110,406	72,910	154,893
Time for simplification (sec)	2,347	1,305	653	1,515
Time for simplification (min)	39	22	11	25

Table 6: Persistence values removed that are ≤ 50 for each dataset.

	Canyon	Dragon	Idaho	BigCreek
Number of critical arcs before	46,536,448	50,383,368	67,151,291	126,879,162
Number of removed critical arcs	28,967,408	8,316,202	1,703,202	4,787,232
Number of removed nodes	3,399,684	3,552,267	2,112,346	5,655,707
Percent of critical vertices after	91.99%	93.67%	96.05%	94.47%
Percent of critical edges after	89.79%	89.49%	94.47%	92.55%
Percent of critical triangles after	96.27%	96.72%	99.08%	98.34%
Node Connectivity Preservation	91.90%	92.36%	95.96%	94.46%
Arc Connectivity Preservation	37.75%	83.49%	97.46%	96.23%
Max Memory Usage (MB)	183,851	116,217	74,295	156,429
Time for simplification (sec)	2,832	1,472	729	1,523
Time for simplification (min)	47	25	12	25

Table 7: Persistence values removed that are ≤ 100 for each dataset.