

DEVICE AND METHOD FOR PERFORMING PARALLEL MESH SIMPLIFICATION

TECHNICAL FIELD

[0001] Various aspects of this disclosure relate to a device for performing mesh simplification. Various aspects of this disclosure relate to a method for performing mesh simplification. Various aspects of this disclosure relate to a non-transitory computer-readable medium storing computer executable code for performing mesh simplification. Various aspects of this disclosure relate to a computer executable code for performing mesh simplification.

BACKGROUND

[0002] Mesh simplification is a geometrical process that converts a complex three-dimensional (3D) mesh model into a simplified mesh while preserving its overall shape and features as much as possible. It can be used to generate resolution hierarchies for efficient geometric processing and level-of-detail (LOD) rendering in applications such as 3D games and virtual reality.

[0003] A method of mesh simplification method is incremental decimation, which removes one element, for example one vertex, one edge or one face at a time based on some criteria. The criteria are designed to determine how the next removal operation is chosen and also to preserve the quality of the simplified mesh. Most incremental decimation algorithms are greedy-based sequential approaches implemented on a central processing unit (CPU), which chooses one element with the minimum cost to decimate at each time. A greedy-based sequential approach follows the problem-solving heuristic of making the locally optimal choice at each stage. However, since only one element, for example one vertex, one edge or one face is removed at a time, this sequential scheme is time consuming and slow for processing meshes with a large number of vertices.

[0004] With the development of a graphics processing unit (GPU), GPU-based algorithms have been proposed to speed up mesh simplification by exploiting the parallelism provided by GPU to accelerate mesh simplification in which all vertices are sorted based on their QEM errors and a number of vertices with the lowest error

values are removed in parallel using a set of per-vertex boundaries to prevent mesh foldovers. One example is super-independent vertices which are vertices that are at least 3 rings away from each other. Each independent vertex could define an independent area, in which only one edge could be collapsed without affecting other areas. The super-independent vertices are computed with a parallel adaption of the greedy serial algorithm. The one ring triangles of each candidate edge are locked by labeling, and then the label of each of its adjacent triangles is checked thereafter. If the labels are consistent, this candidate edge is independent of others. Otherwise, the edge has conflicts with others. Another example uses a set of disjoint parts of a mesh which is constructed using k-d tree space partitioning, which enables the edge contraction operations along these regions to run in parallel. However, the main issue that parallelized algorithms have to deal with is to avoid the conflicts, which exist in the overlapped regions modified by the parallel incremental decimation operators. Moreover, the simplified mesh by parallelized algorithms may produce low quality meshes compared to a sequential greedy algorithm running on CPU. Therefore, a parallel method to avoid choosing wrong elements, which introduce large distortion, to decimate is desired.

SUMMARY

[0005] Various embodiments concern a device for performing mesh simplification, the device including: one or more processor(s); and a memory having instructions stored therein, the instructions, when executed by the one or more processor(s), cause the one or more processor(s) to: randomly group the plurality of component edge of an input mesh representing a three-dimensional model into a plurality of edge groups, with each group including a plurality of component edges; select a plurality of candidate edges from the plurality of component edges; select a set of independent edges from the plurality of candidate edges; and parallelly collapse the set of independent edges.

[0006] According to one embodiment, the selected candidate edges have an error less than a predetermined candidate threshold.

[0007] According to one embodiment, selecting a plurality of candidate edges from the plurality of component edges includes: determining the component edge with a minimal error for each edge group of the plurality of edge groups; and selecting the

component edge as the selected candidate edge if the minimal error is less than the predetermined candidate threshold.

[0008] According to one embodiment, each candidate edge is surrounded by one adjacent triangle ring.

[0009] According to one embodiment, the one or more processor(s) is configured to: (a) regard the candidate edge as a checking candidate edge; (b) check whether any candidate edge(s) exists within two adjacent triangle ring away from the checking candidate edge; (c) if there exists neighbouring candidate edge(s), compare an error of the checking candidate edge and an error of the neighbouring candidate edge(s); and (d) reject the checking candidate edge if the neighbouring candidate edge(s) has a smaller error.

[0010] According to one embodiment, the one or more processor(s) is configured to: perform steps (a) to (d) for each candidate edge in the plurality of candidate edges; and form the set of independent edges from non-rejected candidate edges.

[0011] According to one embodiment, each independent edge in the set of independent edges are at least two adjacent triangle rings away from each other.

[0012] Various embodiments concern a method for performing mesh simplification, the method including: randomly grouping the plurality of component edges of an input mesh representing a three-dimensional model into a plurality of edge groups with each edge group including a plurality of component edges; selecting a plurality of candidate edges from the plurality of component edges; selecting a set of independent edges from the plurality of candidate edges; and parallelly collapse the set of independent edges.

[0013] According to one embodiment, the selected candidate edges have an error less than a predetermined candidate threshold.

[0014] According to one embodiment, selecting a plurality of candidate edges from the plurality of component edges includes: randomly grouping the plurality of component edges into a plurality of edge groups; determining the component edge with a minimal error for each edge group of the plurality of edge groups; and selecting the component edge as the selected candidate edge if the minimal error is less than the predetermined candidate threshold.

[0015] According to one embodiment, each candidate edge is surrounded by one adjacent triangle ring.

[0016] According to one embodiment, the method includes: (a) regarding a candidate edge as a checking candidate edge; (b) checking whether exists any candidate edge(s) wherein it is within two adjacent triangle ring away from the checking candidate edge; (c) if there exists neighbouring candidate edge(s), comparing an error of the checking candidate edge and an error of the neighbouring candidate edge(s); and (d) rejecting the checking candidate edge if the neighbouring candidate edge(s) has a smaller error.

[0017] According to one embodiment, the method includes: performing steps (a) to (d) for each candidate edge in the plurality of candidate edges; and selecting the set of independent edges from non-rejected candidate edges.

[0018] According to one embodiment, each independent edge in the set of independent edges are at least two adjacent triangle rings away from each other.

[0019] According to one embodiment, a non-transitory computer-readable medium storing computer executable code including instructions for performing mesh simplification described above.

[0020] According to one embodiment, a computer executable code comprising instructions for performing mesh simplification described above.

[0021] It should be noted that embodiments described in context of the method for performing mesh simplification are analogously valid for the device for performing mesh simplification and vice versa.

BRIEF DESCRIPTION OF THE DRAWINGS

[0022] The present disclosure will be better understood with reference to the detailed description when considered in conjunction with the non-limiting examples and the accompanying drawings, in which:

- FIG. 1 shows an example of edge collapse according to various embodiments.
- FIG. 2 illustrates a method 200 for performing mesh simplification according to various embodiments.
- FIG. 3 illustrates a device for performing mesh simplification according to various embodiments.
- FIGS. 4A and 4B show edges of a mesh according to various embodiments.

- FIG. 5 illustrates an exemplary flowchart of a mesh simplification process according to various embodiments.
- FIG. 6 illustrates results of the mesh simplification process disclosed herein on a head model on GPU vs QSlim on CPU.
- FIG. 7 illustrates results of the mesh simplification process disclosed herein on a Kylin model on GPU vs QSlim on CPU.
- FIG. 8 illustrates results of the mesh simplification process disclosed herein on a Poisedon model on GPU vs QSlim on CPU.
- FIG. 9 illustrates results of the mesh simplification process disclosed herein (with and without threshold) on a fan-disk model on GPU vs QSlim on CPU.
- FIG. 10 illustrates results of the mesh simplification process disclosed herein (with and without threshold) on a chair model on GPU vs QSlim on CPU.

DETAILED DESCRIPTION

[0023] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and embodiments in which the disclosure may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the disclosure. Other embodiments may be utilized and structural, and logical changes may be made without departing from the scope of the disclosure. The various embodiments are not necessarily mutually exclusive, as some embodiments can be combined with one or more other embodiments to form new embodiments.

[0024] Embodiments described in the context of one of the devices or methods are analogously valid for the other devices or methods. Similarly, embodiments described in the context of a device are analogously valid for a method, and vice-versa.

[0025] Features that are described in the context of an embodiment may correspondingly be applicable to the same or similar features in the other embodiments. Features that are described in the context of an embodiment may correspondingly be applicable to the other embodiments, even if not explicitly described in these other embodiments. Furthermore, additions and/or combinations and/or alternatives as described for a feature in the context of an embodiment may correspondingly be applicable to the same or similar feature in the other embodiments.

[0026] In the context of various embodiments, the articles “a”, “an” and “the” as used with regard to a feature or element include a reference to one or more of the features or elements.

[0027] As used herein, the term “and/or” includes any and all combinations of one or more of the associated listed items.

[0028] FIG. 1 shows an example of edge collapse according to various embodiments.

[0029] A greedy-based sequential approach (e.g., QSlm) decimates the mesh by applying the edge collapse operator in a greedy strategy driven by quadric error metric (QEM). QEM is used to measure the error introduced by edge collapse. Given a vertex v and a plane $q = (a, b, c, d)$ where (a, b, c) is its unit normal vector and d is the distance of the origin to the plane, the squared distance from v to q can be expressed as $A_q(v) = (v, 1) (q^T q)(v, 1)^T = (v, 1) Q(q)(v, 1)^T$, where the quadratic $Q(q)$ is a 4 x 4 matrix associated with q . For a vertex v on the mesh, its quadratic Q_v is defined to be the sum of the matrices associated with its incident faces: $Q_v = \sum_{i \in N_v} Q_i$, where N_v is the set of v 's incident triangles.

[0030] As illustrated FIG. 1, an edge 102, $e = (v_s, v_t)$ is collapsed into a vertex v_c 104. The optimal position of v_c is determined by minimizing the error $\Delta(v_c) = [v_c^T (Q_{v_s} + Q_{v_t})] / 2$, and the minimal error is called the QEM error of the edge e . In QSlm, all the edges are sorted according to their QEM error $\Delta(v_c)$. In each iteration, the edge with the minimal error is chosen for collapse and the errors of its adjacent edges are then updated after the edge collapse. The algorithm will terminate once the target ratio is reached. Topologically, an edge collapse process includes merging v_s and v_t into v_c , deleting two faces f_r and f_l , and updating adjacent edges of v_c . The sequential scheme of iteratively finding the best edge for collapse in QSlm can be implemented on CPU using a heap data structure to store all the existing edges with the smallest error cost at top. Since only one edge is collapsed at a time and the edges in the heap are needed to be updated and maintained, this sequential scheme is slow for processing meshes with a large number of vertices.

[0031] Therefore, it is desirable to have an efficient method of processing meshes.

[0032] FIG. 2 illustrates a method 200 for performing mesh simplification according to various embodiments.

[0033] According to various embodiments, the method 200 for performing mesh simplification may be provided. In some embodiments, the method 200 may include a step 202 of using a processor to randomly group the plurality of component edges of an input mesh into a plurality of edge groups. The input mesh may represent a three-dimensional model. The plurality of edge groups may include a plurality of component edges.

[0034] In various embodiments, the method 200 may also include a step 204 of selecting a plurality of candidate edges from the plurality of component edges.

[0035] In various embodiments, the method 200 may also include a step 206 of selecting a set of independent edges from the plurality of candidate edges.

[0036] In various embodiments, the method 200 may also include a step 208 of parallelly collapsing the set of independent edges.

[0037] Steps 202 to 208 are shown in a specific order, however other suitable arrangements are possible. Steps may also be combined in some cases. Any suitable order of steps 202 to 208 may be used.

[0038] FIG. 3 illustrates a device 300 for performing mesh simplification according to various embodiments.

[0039] In an embodiment, the device 300 may include at least one of: a processor 302 for performing mesh simplification, a memory 304, an input/output module 306, an edge grouping module 308, a candidate edge selecting module 310, an independent edge selecting module 312, an edge collapsing module 314 and/or a combining module 316.

[0040] While such program modules are shown in block diagram format, it is to be understood that the functionality performed by a single program module or by multiple program modules.

[0041] In an embodiment, the input/output module 306, the edge grouping module 308, the candidate edge selecting module 310, the independent edge selecting module 312, the edge collapsing module 314 and/or the combining module 316 may be part of or may be done by the processor 302. However, in other embodiments, the input/output module 306, the edge grouping module 308, the candidate edge selecting module 310, the independent edge selecting module 312, the edge collapsing module 314 and/or the combining module 316 may be part of or done by another processor.

[0042] In an embodiment, the memory 304 may be a random-access memory (RAM). The memory may be used to store mesh data.

[0043] According to various embodiments, a computer program product may store the computer executable code including instructions for performing mesh simplification according to the various embodiments. The computer executable code may be a computer program. The computer program product may be a non-transitory computer-readable medium. The computer program product may be in or may be the device 300.

[0044] In an embodiment, the processor 302 may be configured to randomly group a plurality of component edges of an input mesh into a plurality of edge groups using the edge grouping module 308. The input mesh may represent a three-dimensional model. The plurality of edge groups may include a plurality of component edges.

[0045] In an embodiment, the processor 302 may be configured to select a plurality of candidate edges from the plurality of component edges by the candidate edge selecting module 310.

[0046] In an embodiment, the processor 302 may be configured to select a set of independent edges from the plurality of candidate edges by the independent edge selecting module 312.

[0047] In an embodiment, the processor 302 may be configured to parallelly collapse the set of independent edges using the edge collapsing module 314.

[0048] In an embodiment, the processor 302 may use the combining module 316 to combine a simplified component mesh with another simplified component mesh to form the final simplified mesh.

[0049] In an embodiment, the device 300 may receive the input via the input/output module 306. The input may be a manifold triangular mesh $M = (V, F)$, in which $V \in R^{3vn}$ stores the position of a list of vn vertices and $F \in R^{3fn}$ is a list of fn triangular faces. In an embodiment, each triangle face may be defined by a triple (v_1, v_2, v_3) of its three vertices' indices. In an embodiment, the target number $r\%$ that is the ratio of the vertex number of the simplified mesh to the original vn . may be specified. In an embodiment, the output may be a simplified triangular mesh $M' = (V', F')$ with fewer vertices ($\# \text{ of } V' \approx r\% * vn$) and fewer faces.

[0050] In an embodiment, the simplified mesh may reduce the complexity of the input mesh in terms of representation, while preserving the shape and features of the input mesh as much as possible.

[0051] In an embodiment, the method and device disclosed herein may implement parallel mesh simplification (e.g., on a GPU) which may take both speed and quality into account, resulting in more efficient and/or accelerated mesh simplification.

[0052] In various embodiments, a parallel algorithm to accelerate mesh simplification may be used. The algorithm may simultaneously performs multiple edge-collapses on a manifold triangular mesh driven by the classic quadratic error metric (QEM).

[0053] In various embodiments, the algorithm may include two techniques. The first technique may be an edge candidate generation and independent edge selection scheme. The first technique may result in the selection of a set of independent edges with relatively small QEM and without conflicts from existing edges for edge collapse.

[0054] In various embodiments, the second technique may be a threshold control and constraint checking mechanism for edge candidate generation which may ensure the quality of the simplified meshes.

[0055] In various embodiments, the method disclosed herein may be run entirely on GPU, minimizing the data transfer between CPU and GPU. In various embodiments, compared with the sequential implementation of QSlim and existing GPU-based mesh simplification methods, the method disclosed herein may select more independent edges to collapse at each round of edge collapses, which may substantially speeds up the simplification process. The method disclosed herein may reduce the runtime and improve the simplified mesh quality as well.

[0056] In various embodiments, the techniques disclosed herein may be based on edge collapse driven by quadratic error metric. However, by exploiting the parallelism provided by GPU, the techniques disclosed herein may utilize a set of independent edges for simultaneous collapse rather than greedily finding only one best edge to collapse at a time on CPU.

[0057] In various embodiments, the threshold control and constraint checking mechanism for candidate generation may ensure good quality of mesh simplification results. In various embodiments, the independent edge selection scheme may help to find as many edges without conflicts for further simultaneous edge collapse as possible, which may outperform previous known super-independent vertices-based methods.

[0058] In various embodiments, the input mesh may be simplified to have only lesser (e.g., 10%) number of vertices compared to the original number of vertices. In an embodiment, if the input mesh has 10% of the original number of vertices, roughly 90% edges may be collapsed.

[0059] In various embodiments, if the edge collapse is conducted one by one based on the edge QEM errors strictly, a lot of extra computation and memory consumption to find the edge with minimal error is required. The techniques disclosed herein may find a group of edge candidates with relatively small QEM errors. The order to collapse these candidates may not affect the final simplification result much. A set of independent edges may be selected without conflicts from these candidates to perform edge collapse in parallel.

[0060] In various embodiments, the techniques disclosed herein may result in an efficient and effective way to generate a group of edge candidates with relatively small QEM errors.

[0061] In an exemplary embodiment, if among every 8 possible edges the edge with minimum cost is chosen, the probability that the choice is wrong is $(\frac{5}{100})^8 \approx 4 * 10^{-11}$ for target ratio $r\% = 5\%$, $(\frac{50}{100})^8 \approx 0.001$ for $r\% = 50\%$, and $(\frac{90}{100})^8 \approx 0.3487$ for $r\% = 90\%$. Although the probability may be low when the mesh is largely simplified, the probability may increase dramatically when r increases. In an embodiment, collapsing a wrong candidate edge may affect the quality of the simplified result. Therefore, it will be necessary to avoid choosing the wrong edge. For example, if all possible edges in a group have large costs, none of the edges should be collapsed.

[0062] In various embodiments, neighbouring indexed vertices may be likely topologically close to each other and the QEM costs may be likely similar in a local region.

[0063] In various embodiments, the processor 302 may select a plurality of candidate edges from the plurality of component edges by randomly grouping the plurality of component edges into a plurality of edge groups.

[0064] In various embodiments, the processor 302 may determine the component edge with a minimal error for each edge group of the plurality of edge groups.

[0065] In various embodiments, the processor 302 may select the component edge as the selected candidate edge if the minimal error is less than the predetermined candidate threshold.

[0066] In an exemplary embodiment, groups may be constructed such that each group include randomly sampled edges (e.g., 8 edges). For each group, the edge with the minimal QEM error may be determined. The edge with the minimal QEM error may be checked to determine whether it satisfies a predetermined candidate threshold. If the edge does not pass the threshold check, the edge will be eliminated.

[0067] In various embodiments, the processor 302 may randomly shuffle the array of existing edges and may assign one thread per existing edge. In each thread, the QEM error for each edge may be calculated based on local features. After synchronizing, the QEM errors may be normalized into [0, 1].

[0068] In various embodiments, to avoid the wrong choice, the processor 302 may calculate a threshold τ and may filter out the candidate edge that may has an error larger than τ .

[0069] In an exemplary embodiment, assuming that the simplification ratio is $r\%$, the edge located in the tail $1-r\% \sim 100\%$ portion of the queue that is lined up in the ascending order of QEM may not be selected. In various embodiments, the edges may not be sorted but may use for example a binary search to find the value of threshold τ . In various embodiments, the number of edges with the error below τ will be counted, and the binary search may be stopped if the number reaches roughly $1-r\%$ of the total number of existing edges.

[0070] In various embodiments, the lower and upper bound for the binary search of the threshold may be set by:

$$\begin{cases} \varepsilon_{upper} = 1 - r\% - 10\% \\ \varepsilon_{lower} = 1 - r\% - 15\% \end{cases}, \quad \text{if } r\% < 30\%$$

$$\begin{cases} \varepsilon_{upper} = 1 - r\% \\ \varepsilon_{lower} = 1 - r\% - 5\% \end{cases}, \quad \text{if } 70\% < r\% < 90\%$$

$$\begin{cases} \varepsilon_{upper} = 1 - r\% + 10\% \\ \varepsilon_{lower} = 1 - r\% + 5\% \end{cases}, \quad \text{if } r\% > 90\%$$

[0071] In various embodiments, an edge may be suitable for edge collapse if the QEM error of the edge is less than the threshold τ . This threshold may ensure the quality of the simplified mesh.

In various embodiments, in each thread that is assigned to an existing edge, the topology constraints may be checked to avoid self-intersection and/or occurrence of vertices for example with valence less than 3, and/or shape constraints. This may avoid occurrence of silver triangles and flipped normal. In an exemplary embodiment, every 8 threads may be regarded as a group. The edge with the minimum error which passes the constraint checking may be chosen as the candidate edge for this group. If the QEM error of the candidate edge is larger than the estimated threshold, this candidate will be deleted. Therefore, the total number of candidates may be less than or equal to one-eighth of numbers of the existing edges.

[0072] FIGS. 4A and 4B show edges of a mesh according to various embodiments.

[0073] In various embodiments, each candidate edge may be surrounded by one adjacent triangle ring. In various embodiments, two edges may be independent to each other if there is no overlapping between their surroundings. In other words, the two independent edges may be at least 2 rings away from each other.

[0074] In various embodiments, the processor may be configured to do step (a) of regarding a candidate edge as a checking candidate edge. In various embodiments, the processor may be configured to do step (b) of checking whether exists any candidate edge(s) which may be within two adjacent triangle ring away from the checking candidate edge. In various embodiments, if there exists any neighboring candidate edge(s), the processor may be configured to do step (c) of comparing an error of the checking candidate edge and an error of the neighbouring candidate edge(s). In various embodiments, the processor may be configured to do step (d) of rejecting the checking candidate edge if the neighbouring candidate edge(s) has a smaller error.

[0075] In various embodiments, the processor may be configured to perform steps (a) to (d) for each candidate edge in the plurality of candidate edges. In various embodiments, the processor may be configured to select the set of independent edges from non-rejected candidate edges.

[0076] In various embodiments, each independent edge in the set of independent edges are at least two adjacent triangle rings away from each other.

[0077] As shown in FIG. 4A, a thread 402 may be assigned per candidate edge and may label the two end vertices 403A, 403B of each candidate edge. After synchronizing

the labeling of all threads, each thread will start checking the independency of its assigned

candidate edge. For each candidate edge, the processor may check if any of its one-ring vertices is labeled. For each labeled neighboring vertex, the processor may check if any of its one-ring neighboring edges is a candidate edge. If there exists a candidate edge, the processor may compare the QEM error of this neighboring candidate edge with the checking candidate edge. If the neighboring candidate edge has a smaller QEM error, the checking candidate is rejected. If the neighboring candidate edge has a larger QEM error, the checking candidate is kept. Otherwise, the neighboring candidate edge and the checking edge have the same QEM error, and then the edge with a smaller index will be selected. So, if there exists any candidate edge whose QEM error is smaller than that of the checking candidate edge within the 2-ring neighborhood, the checking process may return false and may discard this checking candidate. The remaining candidates may form the independent edges set.

[0078] In the example shown in FIG. 4B, e_1 and e_2 are independent to each other while e_2 and e_3 are conflicting, and e_1 and e_3 are conflicting as well. In various embodiments, independent edges may be collapsed simultaneously, so they may be collapsed in parallel on GPU.

[0079] In various embodiments, compared with previous super-independent vertices algorithms, the processor may choose independent edges with small QEM errors and may avoid over-rejection due to mutual locking of multiple edges. For example, given e_1 , e_2 , e_3 with the QEM errors in the ascending order as candidates in FIG. 4B, the processor may check e_1 , e_2 , e_3 in parallel. For e_1 , since e_3 is a candidate in its neighboring region but has a larger error, e_1 will be kept. Similarly, e_2 is kept. For e_3 , there are two candidates e_1 and e_2 that have conflicts. Once the processor detects that any of e_1 and e_2 has a smaller error than e_3 , e_3 will be dropped out. The processor may select e_1 and e_2 as an independent set. Even if all existing edges are initially taken as the candidates, the processor may still find a set of independent edges.

[0080] In various embodiments, in each round, the processor may select about 2.6% independent edges for parallel collapse among the existing edges. In general, one edge collapse operation will reduce the number of edges by 3. Based on this rate, the processor may simplify the original mesh to 50% in roughly 10 rounds, and may simplify the mesh to 25% and 5% in around 20 rounds and 40 rounds respectively.

[0081] In various embodiments, to collapse the selected independent edges, the processor may assign per independent edge per thread. The edge collapse operator may be done in parallel without conflicts. In each parallelized thread, the edge collapse operation for e as shown in FIG. 1 may include removing two faces f_l and f_r and/or removing vertex v_s (suppose that v_s has smaller index in the two end vertices of e) and/or removing e , edge $\overline{v_s v_r}$ and edge $\overline{v_s v_l}$. The processor may update (for example in the memory) topological information in related half edges as well.

[0082] In various embodiments, the processor may conduct post-processing. The post-processing may include filtering out the list of existing vertices V' and updating indices in existing faces F' .

[0083] In various embodiments, the processor may assign per vertex in V per thread. For the vertex still labeling true (still existing), it may be pushed into V' . An atomic add operator may be used to calculate the current index in V' and record the mapping of the vertex's corresponding indices in V and V' . Then, each thread may be assigned to each face in F . For the existing face, the indices of three vertices may be updated according to the recorded mapping and $f' = (v'_1, v'_2, v'_3)$ may be pushed into F' .

[0084] In various embodiments, the generated V' and F' of the simplified mesh may be transferred from GPU back to CPU and may be exported into a required format such as OBJ and/or PLY.

[0085] FIG. 5 illustrates an exemplary flowchart 500 of a mesh simplification process according to various embodiments.

[0086] In various embodiments, a mesh file 502 may be loaded in and parsed in step 504. The mesh file 502 may be parsed into an array of vertices V 506A and an array of faces F 506B on a CPU, for example in the RAM.

[0087] In various embodiments, once the file of the 3D mesh model 502 is loaded in and parsed into an array of vertices V and an array of faces F on CPU RAM, a global memory may be allocated on GPU and these data may be transferred from the CPU to the GPU.

[0088] In various embodiments, in step 508 the half-edge data structure may be created based on V and F in parallel on GPU. In step 510, each half-edge may include the following elements: halfedge (HE) index, previous HE index, next HE index, opposite HE index, start vertex index, target vertex index, and face index. In various

embodiments, halfedge data structure may help to organize the topological information and search for neighboring elements.

[0089] In various embodiments, the selection of candidate edges and independent edges runs entirely on GPU which may be an iterative process framed in a dashed box 512. For each round, in step 514 candidate edges will be sampled and identified from every 8 existing edges, which satisfy topological constraints and be within the threshold. Then in step 516, independent edges will be selected among the candidates with the help of the labeling and checking method. In step 518, the edge collapse operator will be applied to all independent edges in parallel thereafter. In step 520, the existing edges will be compressed for the next round until meeting the target simplification ratio.

[0090] In various embodiments, the simplified half-edge list 522 will then be post-processed in step 524 to generate simplified vertex list V' 526A and simplified face list F' 526B of the simplified mesh. V' and F' will be transferred from GPU back to CPU thereafter and the simplified mesh file 530 is exported in step 528 any required format.

[0091] FIG. 6 illustrates results of the mesh simplification process disclosed herein on a head model on GPU vs QSlim on CPU.

[0092] FIG. 7 illustrates results of the mesh simplification process disclosed herein on a Kylin model on GPU vs QSlim on CPU.

[0093] FIG. 8 illustrates results of the mesh simplification process disclosed herein on a Poisedon model on GPU vs QSlim on CPU.

[0094] FIG. 9 illustrates results of the mesh simplification process disclosed herein (with and without threshold) on a fan-disk model on GPU vs QSlim on CPU.

[0095] FIG. 10 illustrates results of the mesh simplification process disclosed herein (with and without threshold) on a chair model on GPU vs QSlim on CPU.

[0096] The parallel mesh simplification method is implemented on GPU with CUDA Toolkit 10.1. The experiments are performed and tested on a computer with CPU i7-7700 3.60 GHz and GPU GTX 1080Ti. The processing time and quality is evaluated when comparing results of the mesh simplification process disclosed herein with the with the QSlim method.

[0097] As shown in FIGS. 6-8, the original input model in FIGS. 6a, 7a, 8a and 6f, 7f, 8f is simplified to have 50%, 25%, 5% and 1% numbers of vertices using the mesh simplification process disclosed herein in FIGS. 6b-d, 7b-d, 8b-d and QSlim implemented by MeshLab in FIGS. 6g-j, 7g-j, 8g-j respectively. The quality of mesh simplification process disclosed herein is comparable with QSlim visually.

[0098] As shown in Table 1. the root mean square (RMS) of Hausdorff distance between the simplified mesh and the original mesh is calculated. It can be seen from the table that the parallel mesh simplification method sometimes could even generate a simplified mesh with higher quality than QSlim.

[0099] Table 1 also records the runtime. Compared with QSlim, the parallel mesh simplification method can run much faster. For many smooth models like Head and Kylin in FIGS. 6-7, the parallel mesh simplification method could simplify the original mesh to have 50%, 25%, 5% and 1% number of vertices in roughly 10 rounds, 20 rounds, 40 rounds and 60 rounds of edge collapses. The number of rounds increases for models with sharp features since more candidates will be rejected under the constraint of threshold. It can be seen from Table 1 that the parallel mesh simplification method could achieve 22.5 — 33.7 times speedup compared to QSlim for these models.

[00100] In general, compared with QSlim, the parallel mesh simplification method can run much faster and achieve comparable good mesh quality. For meshes with a large number of vertices such as Poseidon (more than 2 million vertices), the parallel mesh simplification method could simplify within 4 seconds.

Table 1. Statistics of experimental results (Our GPU based method vs CPU based QSlim)

Model	Ratio	Our GPU				CPU QSlim		
		Fig.#	Rounds	Time (ms)	Quality (RMS)	Fig.#	Time (ms)	Quality (RMS)
Head (#v 44241) FIG. 6(a)(f)	50%	FIG. 6(b)	9	32.94	0.042018	FIG. 6(g)	1110	0.04976
	25%	FIG. 6(c)	18	50.62	0.07924	FIG. 6(h)	1348	0.089771
	5%	FIG. 6(d)	39	71.09	0.330579	FIG. 6(i)	1557	0.35785
	1%	FIG. 6(e)	60	87.67	1.023095	FIG. 6(j)	1630	1.1173
Kylin (#v 655980) FIG. 7(a)(f)	50%	FIG. 7(b)	14	489.17	0.005207	FIG. 7(g)	15956	0.004737
	25%	FIG. 7(c)	25	696.46	0.010601	FIG. 7(h)	20263	0.009925
	5%	FIG. 7(d)	49	856.66	0.028478	FIG. 7(i)	24204	0.029014
	1%	FIG. 7(e)	74	919.41	0.077826	FIG. 7(j)	25356	0.078198
Poseidon (#v 2003932) FIG. 8(a)(f)	50%	FIG. 8(b)	16	2066.47	0.00404	FIG. 8(g)	51649	0.003483
	25%	FIG. 8(c)	28	2901.19	0.008758	FIG. 8(h)	68784	0.008
	5%	FIG. 8(d)	55	3451.83	0.033663	FIG. 8(i)	80666	0.03516
	1%	FIG. 8(e)	81	3588.09	0.126848	FIG. 8(j)	80863	0.128313

[00101] In various embodiments, introducing threshold control may be at the sacrifice of time. However, this threshold control may be necessary to preserve shape and ensure the quality. In FIG. 9, the original model is shown in the first column, the 50%, 25% and 5% simplified models using the parallel mesh simplification method with threshold are shown in the second column, the meshes generated by the parallel mesh simplification method without the control of threshold are displayed in the third column, and the results by QSlim on CPU are displayed in the fourth column. Without the threshold control, the candidates with the local minimum error in the group but still having high QEM errors such as the edges on the sharp boundaries are not detected. If these candidates survive after independency checking and thereafter are collapsed, large distortion will be introduced cumulatively. Similar experimental results are shown in FIG. 10. The parallel mesh simplification method with the threshold control could produce similar results as QSlim, but the parallel mesh simplification method without the threshold control fails to preserve the sharp feature of chair arms and legs. Therefore, the threshold control plays an important role in preserving sharp features and ensuring good quality. From Table 2, it can see that the parallel mesh simplification method may be 10-24 times faster compared to QSlim for those models with sharp features.

Table 2. Statistics of experimental results (Our GPU based method vs CPU based QSlim)

Model	Ratio	Our GPU with threshold				GPU without threshold				CPU QSlim		
		Fig.#	Rs	Time (ms)	Quality (RMS)	Fig.#	Rs	Time (ms)	Quality (RMS)	Fig.#	Time (ms)	Quality (RMS)
Fandisk (#v 6475) FIG. 9(a)	50%	FIG. 9(b)	26	32.84	0.000017	FIG. 9(c)	11	9.56	0.000971	FIG. 9(d)	326	0.000002
	25%	FIG. 9(e)	46	49.68	0.000208	FIG. 9(f)	21	14.29	0.00169	FIG. 9(g)	373	0.000086
	5%	FIG. 9(h)	87	85.36	0.004332	FIG. 9(i)	50	24.87	0.007341	FIG. 9(j)	440	0.001486
Chair (#v 164459) FIG. 10(a)	50%	-	16	151.45	0.000051	-	8	75.45	0.024802	-	3632	0.000069
	25%	-	28	208.23	0.000145	-	17	120.08	0.045378	-	5116	0.000191
	5%	FIG. 10(b)	55	272.97	0.000769	FIG. 10(c)	36	158.23	0.139939	FIG. 10(d)	5664	0.000963

[00102] The methods described herein may be performed and the various processing or computation units and the devices and computing entities described herein may be implemented by one or more circuits. In an embodiment, a "circuit" may be understood as any kind of a logic implementing entity, which may be hardware, software, firmware, or any combination thereof. Thus, in an embodiment, a "circuit" may be a hard-wired logic circuit or a programmable logic circuit such as a programmable processor, e.g. a microprocessor. A "circuit" may also be software being implemented or executed by a processor, e.g. any kind of computer program, e.g. a computer program using a virtual machine code. Any other kind of implementation of the respective functions which are described herein may also be understood as a "circuit" in accordance with an alternative embodiment.

[00103] While the disclosure has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the present disclosure as defined by the appended claims. The scope of the present disclosure is thus indicated by the appended claims and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced.

CLAIMS

1. A device for performing mesh simplification, the device comprising:
one or more processor(s); and
a memory having instructions stored therein, the instructions, when executed by the one or more processor(s), cause the one or more processor(s) to:
randomly group the plurality of component edges of an input mesh representing a three-dimensional model into a plurality of edge groups comprising a plurality of component edges;
select a plurality of candidate edges from the plurality of component edges;
select a set of independent edges from the plurality of candidate edges; and
parallelly collapse the set of independent edges.
2. The device of claim 1, wherein the selected candidate edges have an error less than a predetermined candidate threshold.
3. The device of claim 2, wherein selecting a plurality of candidate edges from the plurality of component edges comprises:
randomly grouping the plurality of component edges into a plurality of edge groups;
determining the component edge with a minimal error for each edge group of the plurality of edge groups; and
selecting the component edge as the selected candidate edge if the minimal error is less than the predetermined candidate threshold.
4. The device of any one of claims 1-3, wherein each candidate edge is surrounded by one adjacent triangle ring.
5. The device of any one of claims 1-4, wherein the one or more processor(s) is configured to:
(a) regard a candidate edge as a checking candidate edge;

(b) check whether exists any candidate edge(s) within two adjacent triangle ring away from the checking candidate edge;

(c) if there exists neighbouring candidate edge(s), compare an error of the checking candidate edge and an error of the neighbouring candidate edge(s); and

(d) reject the checking candidate edge if the neighbouring candidate edge(s) has a smaller error.

6. The device of claim 5, wherein the one or more processor(s) is configured to:
perform steps (a) to (d) for each candidate edge in the plurality of candidate edges; and
select the set of independent edges from non-rejected candidate edges.

7. The device of any one of claims 1-6, wherein each independent edge in the set of independent edges are at least two adjacent triangle rings away from each other.

8. A method for performing mesh simplification, the method comprising:
randomly grouping the plurality of component edges of an input mesh representing a three-dimensional model into a plurality of edge groups comprising a plurality of component edges;
selecting a plurality of candidate edges from the plurality of component edges;
selecting a set of independent edges from the plurality of candidate edges; and
parallelly collapsing the set of independent edges.

9. The method of claim 8, wherein the selected candidate edges have an error less than a predetermined candidate threshold.

10. The method of claim 9, wherein selecting a plurality of candidate edges from the plurality of component edges comprises:
randomly grouping the plurality of component edges into a plurality of edge groups;
determining the component edge with a minimal error for each edge group of the plurality of edge groups; and

selecting the component edge as the selected candidate edge if the minimal error is less than the predetermined candidate threshold.

11. The method of any one of claims 8-10, wherein each candidate edge is surrounded by one adjacent triangle ring.

12. The method of any one of claims 8-11, comprising:

- (a) regarding a candidate edge as a checking candidate edge;
- (b) checking whether exists any candidate edge(s) within two adjacent triangle ring away from the checking candidate edge;
- (c) if there existing neighbouring candidate edge(s), comparing an error of the checking candidate edge and an error of the neighbouring candidate edge(s); and
- (d) rejecting the checking candidate edge if the neighbouring candidate edge(s) has a smaller error.

13. The method of claim 12, comprising:

performing steps (a) to (d) for each candidate edge in the plurality of candidate edges; and
selecting the set of independent edges from non-rejected candidate edges.

14. The method of any one of claims 8-13, wherein each independent edge in the set of independent edges are at least two adjacent triangle rings away from each other.

15. A non-transitory computer-readable medium storing computer executable code comprising instructions for performing mesh simplification according to any one of claims 8 to 14.

16. A computer executable code comprising instructions for performing mesh simplification according to any one of claims 8 to 14.