Computer-Aided Design 78 (2016) 168–178

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

eBits: Compact stream of mesh refinements for remote visualization[☆]

Mukul Sati[a,*], Peter Lindstrom[b], Jarek Rossignac[a]

[a] *School of Interactive Computing, Georgia Institute of Technology, United States*

[b] *Lawrence Livermore National Laboratory, United States*

ARTICLE INFO

Keywords:
Triangle mesh compression
Remote visualization
Level of detail
Selective transmission
Local refinement
Triangle collapse

ABSTRACT

We focus on applications where a remote client needs to visualize or process a complex, manifold triangle mesh, M, but only in a relatively small, user controlled, ***Region of Interest*** (RoI) at a time. The client first downloads a coarse ***base mesh***, pre-computed on the server via a series of simplification passes on M, one per ***Level of Detail*** (LoD), each pass identifying an independent set of triangles, collapsing them, and, for each collapse, storing, in a ***Vertex Expansion Record*** (VER), the information needed to reverse the collapse. On each client initiated RoI modification request, the server pushes to the client a selected subset of these VERs, which, when decoded and applied to refine the mesh locally, ensure that the portion in the RoI is always at full resolution. The eBits approach proposed here offers state of the art compression ratios (using less than 2.5 bits per new full resolution RoI triangle when the RoI has more than 2000 vertices to transmit the connectivity for the selective refinements) and fine-grain control (allowing the user to adjust the RoI by small increments). The effectiveness of eBits results from several novel ideas and novel variations of previous solutions. We represent the VERs using persistent labels so that they can be applied in different orders within a given LoD. The server maintains a shadow copy of the client's mesh. To avoid sending IDs identifying which vertices should be expanded, we either transmit, for each new vertex, a compact encoding of its ***death tag*** – the LoD at which it will be expanded if it lies in the RoI – or transmit vertex masks for the RoI and its neighboring vertices. We also propose a three-step simplification that reduces the overall transmission cost by increasing both the ***simplification effectiveness*** and the regularity of the valences in the resulting meshes.

1. Introduction

The triangle-count in large meshes, which may represent iso-surfaces generated during scientific simulations on a high performance computing cluster or detailed scans of human anatomy, precludes their transmission at full resolution from the server to a remote client for visualization.

The solution of (1) transmitting user-controlled camera motions to the server, (2) generating the corresponding images on the server, and (3) streaming them to the client is impractical when high image resolution and low latency are desired and does not support local processing of the mesh on the client (e.g., calculating surface normals, ridges, or other properties).

Our ***eBits*** approach strives to minimize the transmission cost involved in remote visualization and processing. It focuses on applications where the client needs, at any moment, to have local access to the mesh at full resolution, but to only a small portion of a mesh at a time. We refer to that portion as the ***Region of Interest (RoI)***. The server streams the geometry and connectivity information needed by the client to maintain the RoI at full resolution each time it is moved by the user.

Hence, the eBits representation provides read and annotation access of the full resolution mesh to the client, but operations that alter or refine the mesh locally must be carried out on a private copy that the client must maintain.

Our contribution builds upon previously proposed strategies where, at initialization, the server transmits a coarse (highly simplified) ***base mesh*** and then transmits compressed encodings of local refinements that allow the client to restore to full resolution the current RoI (Fig. 2), which may be changed arbitrarily by the user.

To obtain the base mesh, we perform n simplification steps, each one producing the next ***level of detail (LoD)***. Each step per-

[☆] This paper has been recommended for acceptance by Scott Schaefer and Charlie C.L. Wang.

[*] Corresponding author.

E-mail addresses: mukul@gatech.edu (M. Sati), pl@llnl.gov (P. Lindstrom), jarek@cc.gatech.edu (J. Rossignac).

http://dx.doi.org/10.1016/j.cad.2016.05.016
0010-4485/© 2016 Elsevier Ltd. All rights reserved.

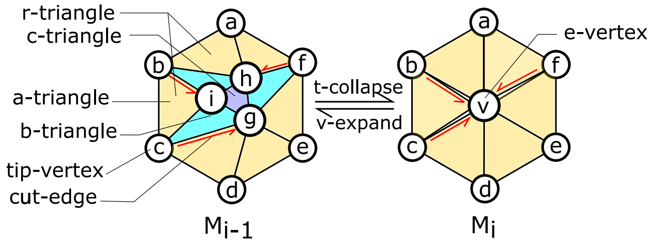

Fig. 1. A t-collapse operation and its inverse v-expand operation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

forms a constrained set of triangle-collapse (***t-collapse***) operations, each collapsing the three vertices of a ***c-triangle*** (collapsible triangle) into a single vertex v, which becomes an ***e-vertex*** (expandable vertex) in the less refined LoD (Fig. 1). A t-collapse removes a ***cluster*** of triangles formed by the c-triangle (blue) and its 3 edge-adjacent ***b-triangles*** (boundary triangles, cyan). ***r-triangles*** (remaining triangles, yellow) in a LoD are those that remain in the coarser LoD. A ***cut-edge*** is a clockwise oriented edge of an r-triangle that is adjacent to a b-triangle and that points to a vertex of the c-triangle in the mesh before the t-collapse and to v after the t-collapse. Hence, there are three cut-edges (red arrows) per c-triangle. The r-triangles that contain a cut-edge are also called ***a-triangles***.

The inverse ***v-expand*** (vertex expand) operation is fully characterized by identifying v and the three cut-edges amongst the oriented half-edges that point to v.

Using t-collapse and v-expand operations as building blocks, we present several novel contributions, which allow us to transmit the missing connectivity information using only about 2.3 bits per full resolution triangle added in the modified RoI.

The transmission cost of the connectivity grows with the number n of ***simplification passes*** and with the average degree (i.e., valence) of the e-vertices. We minimize the combined effect of both factors by using triangle collapses, rather than edge collapses, as simplification primitives and by using a novel algorithm for optimizing the selection of ***independent sets*** of c-triangles at each pass. Due to these contributions, we observe an average reduction of the vertex count by about 41% at each pass. Consequently, $n = 7$ simplification passes suffice to produce a base mesh with only about 2.5% of the original triangle count. Furthermore, we obtain an average degree of 7.3 for the resulting e-vertices, while a naive simplification approach yields a significantly higher degree average (see Section 4).

The information needed to reverse a triangle collapse is stored in a ***Vertex Expansion Record (VER)***. There are several challenges associated with the use of such VERs to define the local expansion of the mesh on the client:

1. The client needs to identify the e-vertices. To address this challenge, we propose two solutions of comparable effectiveness: (1) transmit a bit mask and (2) transmit a ***death tag*** (LoD number at which the vertex will be expanded) for each vertex created by the v-expansion (Section 4).
2. To reduce latency, we do not want the client to have to request each VER. Hence, the server pushes selected VERs to the client. But how does the client know which VER should be applied to which e-vertex? To address this challenge, an eBits server maintains a shadow copy of the clients mesh and sends the VERs in the order of increasing vertex IDs in the client's representation of the mesh.
3. The VERs are created during a one-time simplification process on the server. They are indexed by a vertex ID that identifies the corresponding e-vertex v. Each VER identifies three ***cut-edges*** of v. Unfortunately, the IDs of v and of the cut-edges in the LOD produced by that simplification pass cannot be used on the ***working mesh*** (the shadow copy of the client mesh maintained by the server) because the two meshes are typically different: the working mesh has a full resolution RoI surrounded by rings of decreasing LoDs. To address this problem, we convert these initial vertex and edge IDs into their ***universal ID*** counterparts, which are initialized to the natural IDs of the base mesh and maintained for all vertices of the working mesh by both the server and the client. On the server, the VERs are indexed by their universal vertex IDs using a hash map.
4. We wish to reduce the number of bits used to encode the IDs of the 3 cut-edges and of the death tags of the 3 vertices created by the v-expansion defined by a VER. Since we know the LoD at which a vertex is created, we know the range of valid values for its death tag. Each value has a different probability.

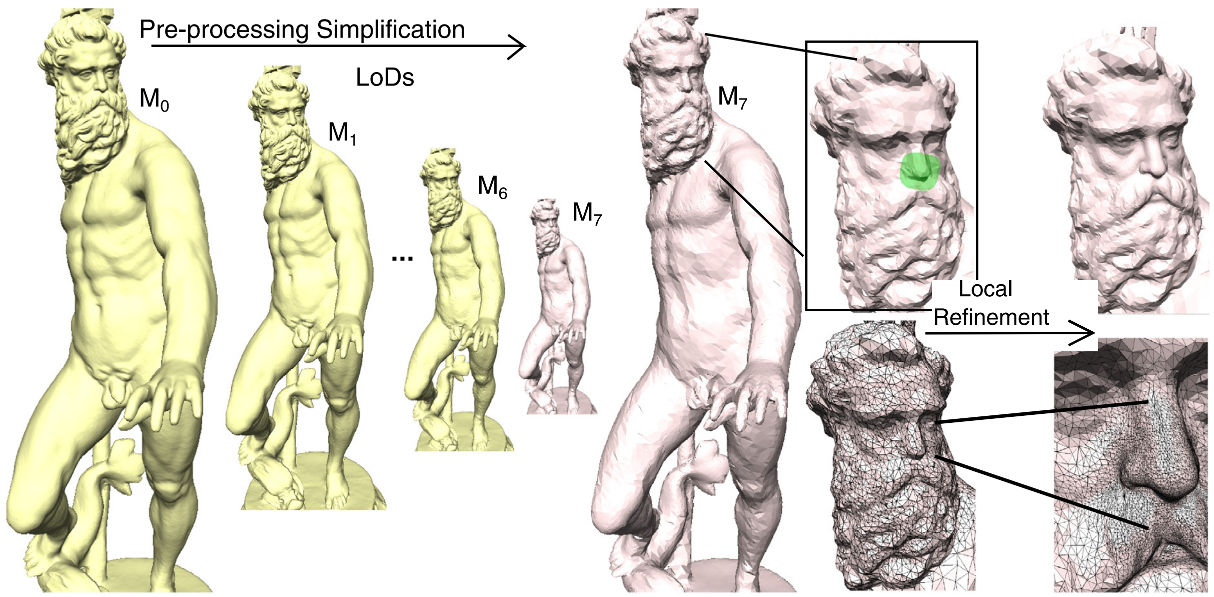

Fig. 2. An overview of eBits: The original mesh M_0, is simplified to obtain a sequence of LoDs. The coarsest mesh (base mesh), M_7, is transmitted to the client, who can request to view a particular region of M_0 in full detail (green RoI in the boxed zoomed in view). The server computes and sends to the client the information needed to refine the mesh locally (right). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

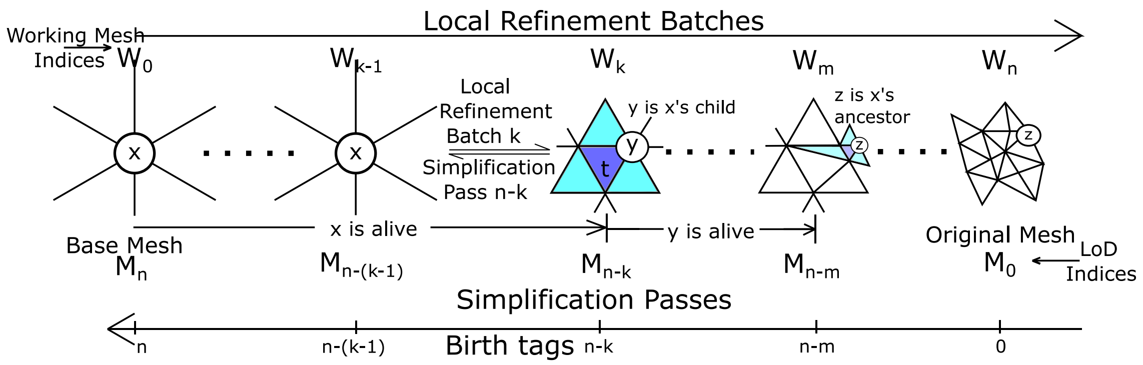

Fig. 3. During preprocessing on the server, an e-vertex x of the base mesh M_n is created in LoD $M_{n-(k-1)}$ during simplification pass $n-k$, as a result of the collapse of a c-triangle, t (blue), of LoD M_{n-k}. The three vertices of t, including vertex y, disappear (i.e., die) during this t-collapse. We say that y is a "child" of x, because it will be "born" (created) when x "dies" and is expanded (split) on the client mesh during refinement pass k. Refinement pass k reproduces the exact connectivity of the portion of the LoD M_{n-k} inside (a slightly expanded version of) the RoI. A subsequent refinement pass may further expand y and create new vertices (such as z, which is an ancestor of x). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The measured entropy of this information is about 1.2 bits. Similarly, we exploit the fact that the distribution of cut-edges around an e-vertex is biased. On average, the entropy of the cut-edge information is about 7.09 bits per vertex expansion.

In our experiments, when transmitting the entire mesh, eBits achieves a connectivity compression ratio approaching 2.08 bpt (bits per triangle). It is remarkable that eBits approaches state-of-the-art single-rate compression (e.g., 1.84 bpt guaranteed by Edgebreaker [1]), yet offers fine-grain control that allows the user to grow and slide the RoI by small (one edge) increments.

The remainder of the paper starts with an overview of eBits and discusses its relation with prior art, then provides implementation details, reports experimental results, and concludes with a detailed comparison with prior art.

2. Overview

We assume that the original triangle mesh, M_0, is an orientable manifold without boundary and that it is represented using a data-structure where vertices and triangles are associated with consecutive positive integer IDs.

During a one-time pre-process on the server, M_0 is simplified using a series of n simplification passes yielding increasingly coarse LoDs: $\{M_0, M_1, \ldots, M_n\}$. Pass i carefully selects a set of independent collapsible triangles (***c-triangles***) in M_{i-1} and collapses them using ***t-collapse*** operations. Each t-collapse collapses 4 triangles and 3 vertices and creates an expandable "parent" vertex (***e-vertex***) v in M_i.

The server archives ***Vertex Expansion Records (VERs)***, which encode the information needed to undo each t-collapse and to refine M_i to M_{i-1}: namely, the ID of v in M_i, the IDs, in M_i, of 3 ***a-triangles*** that define the cut-edges (see Fig. 1), geometry information for the new vertices being added, and the death tag of each child u of v that will be created by the expansion of v. The ***death tag*** indicates the LoD at which u will have to be expanded (if it is in or near the RoI).

The server also archives a compressed version of the ***base mesh*** (coarsest LOD), M_n, and the death tag of each one of its vertices. The compressed base mesh and these death tags are downloaded by the client before the interactive remote inspection of M_0 starts. M_n is used as the initial version of the ***working mesh***, W. Both the server and the client maintain synchronously their own local copy of the evolving W.

As the user defines, and subsequently freely moves the RoI, both the client and server update W identically to ensure that W is expanded correctly, representing at full resolution the RoI portion of W that is selected by the user.

After each user-guided change of the RoI the information used by the client (and synchronously by the server) to refine the working mesh is delivered to the client in n batches, called ***Batch Expansion Records (BERs)*** (Fig. 3). Let W_0 denote the state of the working mesh W before the refinement process starts and let W_k denote the result of applying BER_k.

Below, we first discuss what happens on the client's side after each change of RoI, then discuss the pre-processing done once on the server for each mesh, and finally, what is done on the server to support user interaction, after each change of RoI.

Client interactions and actions after a change of RoI:

The client and server may agree on a particular protocol for communicating changes of the RoI.

We propose three operations: Pick, Grow, and Slide. For a ***pick***, the client selects a vertex v of the working mesh W. If v is not at full resolution, the server transmits VERs to refine the mesh locally until W contains a full resolution vertex f that is a descendant of v (and is close to v). A ***grow*** preserves f, but increases the size of the RoI (either by one edge if size is defined using graph distance or by some increment if size is defined by Euclidean distance). A ***slide*** moves f by one edge of M_0 to a neighboring full resolution vertex.

The client receives a series of BERs, each containing a batch of VERs. For each BER, the client identifies the set B of vertices of W that lie in the RoI and that, according to their death tag, should be expanded by BER k, which contains the VER for each vertex of B in the order of their IDs in W.

However, as in [2], some vertices of B must be ***balanced***, before they can be expanded: we must bring their neighboring vertices to LoD at least $k-1$, so that the VER information can be interpreted unambiguously. Hence, in batch k, the VER of a vertex v is preceded by the VERs of neighboring vertices that must be expanded first to balance v. The client identifies these dependencies recursively and receives the corresponding VERs in post-order traversal of the dependency tree.

The VER contains: (1) information needed to identify the 3 cut-edges of W_{k-1} from the set of edges incident upon v (Fig. 1), (2) an encoding of the 2 vectors that allow the client to compute the vertex coordinates of the 3 children of v in W_k (during t-collapse, we place the parent vertex at the centroid of the child vertices) and (3) the death tags for the three children of v, indicating the LoDs when they are to be expanded.

After receiving batch k, the client executes its v-expansions to produce a new version, W_k, of W. Due to balancing, W_k contains full resolution vertices in the RoI and "rings" of vertices around the RoI at successively lower resolutions (Fig. 4).

The client and server may adopt a common policy to simplify the mesh by reversing the expansion outside of the RoI for clusters

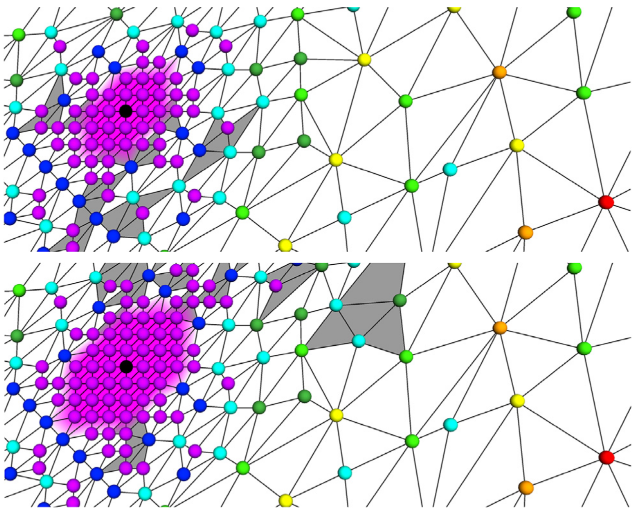

Fig. 4. Working mesh after 2 (top) and 3 (bottom) grow operations. The vertices are colored by death tag (red for base mesh, orange, yellow, light green, dark green, cyan, blue, magenta for full resolution). The triangles are shaded magenta around RoI vertices. Newly inserted clusters are shaded gray. Note that while a grow operation adds several full resolution vertices, the density of newly added clusters decreases with distance from the RoI. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

that are not required for balancing. We do not discuss the details of such cleaning because it may be implemented using simple bookkeeping and t-collapse operations. We do not include cleaning costs in our timing results.

One-time server side pre-processing:

During each simplification pass, the server carefully selects a particular set of c-triangles striving to meet three objectives:

1. The order in which the VERs in BER k are sent is unknown in advance and depends on the current state of W. To ensure that the encoding of the IDs of cut-edges can be interpreted unambiguously, we require that the star (incident triangles) of a balanced e-vertex be always identical. This requirement constrains valid selections of c-triangles to form an ***independent set***, in which no two c-triangles share a vertex.
2. Each VER requires encoding the choice of 3 edges amongst the $d = \deg(v)$ edges incident upon a vertex v. The entropy of that information increases with d. Hence, during simplification, we avoid creating high degree e-vertices.
3. To reduce the cost of transmitting death tags, we wish to reduce the total number n of batches, so that the differences between the death tag of a vertex and the current batch number k are small integers that can be encoded using fewer bits. An eager approach to identify the largest number of independent c-triangles in M_k may not only increase the irregularity of the degrees of the vertices of M_{k+1}, but may also reduce the number of independent c-triangles in M_{k+1}, and hence increase the number of batches (n).

We have explored a variety of approaches for reducing both n and the degree irregularity and report (Section 4) a three-phase solution, ValPack, which is highly effective, as shown by our experimental results.

Server actions after a change of RoI:

The server receives the same changes of RoI as the client does and mimics the client actions, described above, for maintaining its shadow copy of W.

Additionally, the server is responsible for identifying the e-vertices of each batch, for retrieving the suitable set of VERs, for translating the information stored in them so that they can be properly interpreted by the client (in terms of vertex IDs in W), and for sending these translated VERs in the proper order.

3. Prior art

Randomly accessed clusters: To provide a compressed transmission format for random access, some approaches divide the mesh into clusters, compress the connectivity of each cluster using single-rate compression ([3] yields about 1 bpt using entropy encoding) and encode how clusters are stitched. For large clusters, (for example about a million triangles, as used in [4]), the approach yields excellent results. For example, the Progressive Forest Split (PFS) [5] identifies clusters of triangles, removes the interior edges of the clusters, encodes the resulting polygonal mesh using Topological Surgery compression [6] and transmits it as a base mesh first. Then, it transmits the internal connectivity of each cluster using Topological Surgery compression to encode its cut-edges. PFS encodes connectivity using about 5 bpt. Choe et al. [7] use Lloyd's algorithm to form clusters of 1000–2000 triangles. They encode the connectivity between clusters using polygon compression [8], the geometry of the edges between clusters using parabolic prediction, and the connectivity inside each cluster using Angle Analyzer [9]. Yoon and Lindstrom [10] form clusters of a few thousand triangles directly from a streaming mesh [11]. Each triangle and vertex is assigned to a single cluster. Their format uses about 4 bpt. Such approaches support random access, but only at a per cluster granularity.

Compressed custom expansion: Edgebreaker can be used [3] to compress manifold meshes with boundaries. A mesh M of v vertices with a single bounding loop of e edges can be encoded using $2(v+e)$ bits or using $2v+e$ bits plus an integer used to encode the value of e. Consider a simply connected portion P of the original mesh that has already been downloaded and reconstructed by the client. We want to download the connectivity of another adjacent simply connected portion M of the original mesh that shares a series of one or more consecutive edges with P. We could do so by encoding the connectivity of M as explained above, plus two integers identifying the portion of the boundary of P that is shared with M. Using such a scheme, the cost of transmitting the connectivity for extending the mesh is capped by 3 bits per triangle, plus 2 integers per run (connected component) of shared edges between P and M. Hence, this scheme may incur a cost significantly higher than 3 bits per triangle, especially when the portion of the new RoI that needs to be transmitted is small and has multiple edge-connected components.

Progressive meshes: Progressive mesh transmission allows clients to receive first a base mesh (simplest LOD), and then, if desired a series of incremental refinements. Hoppe [12] encodes a set of vertex-splits, each undoing an edge-collapse simplification step. The information needed for each vertex-split identifies the vertex to be split and two of its incident edges. This approach selects a sequence of edge-collapses (ordered by inverse impact on geometric fidelity) so as to achieve the best approximation to the mesh at each refinement step. The order of the vertex splits is fixed to be the reverse order of the corresponding edge collapses. To relax this restriction and allow local refinements, both [13,14] proposed constraints for valid edge-collapses so as to construct a tree hierarchy over the vertex-splits. These schemes offer fine granularity, but have a high transmission cost because, for each vertex-split, the client must receive the ID of the next vertex to be split in the current mesh. A number of compressed progressive schemes aim to amortize this cost by creating batches of edge-collapses, leading to the creation of a number of discrete LoDs for the mesh. The Compressed Progressive Meshes (CPM) [15] performs refinements in batches on the entire mesh. CPM uses a bit-mask to identify the split-vertices in each batch (this avoids having to encode the ID of each split-vertex) and uses a compact encoding (less than 5 bits per split). Thus, CPM encodes the connectivity using about 3.6 bpt. A variation of the CPM solution that offers a fast decompression by using a simplified geometry prediction function was proposed in [16]. To obtain the set of vertex-splits for each batch, Alliez and Desbrun [17] perform a conquest of the mesh, decimating vertices at the center of a 1-ring patch and deterministically re-triangulating the resulting hole, reporting average connectivity compression rates of 1.86 bpt across tested meshes. These approaches, however, lose the granularity of refinement offered by [12]. Further, these approaches also sacrifice selective refinement—if a region is to be viewed at a desired LoD, the records of vertex-split batches for the entire mesh have to be transmitted up to that LoD.

Geometry driven progressive meshes: Hierarchical space partition schemes have also been used to create selective progressive meshes. For example, Gandoin and Devillers [18] store the vertices in a kD-tree and encode how the vertex count is distributed at each split. Using edge-flips and a simplification heuristic that collapses the longest edge first (if possible), Valette et al. [19] achieve compression rates of about 2 bpt. Their approach thus utilizes geometric information to obviate the need for identifying the vertex to be split. While successful in providing the granularity of access of non-batched progressive meshes and competitive compression ratios, their approach does not allow for selective refinement and, thus, for random access to specific portions of the mesh.

Progressive and random accessible meshes: A few attempts have been made at providing a progressive and randomly accessible format. The approach in [20] utilizes the truly selective refinement scheme for progressive meshes [21] to allow for random access to blocks of vertices. Their encoding of vertex splits allows for refinement of just the desired Region of Interest, but requires 5.5 bpt to encode the connectivity. In contrast, our approach refines a slightly larger area than the RoI, providing a gradual change in the refinement level while transitioning from the RoI to the unrefined portion of the mesh.

The eBits approach proposed here builds on many of these previously proposed ideas by sending first a crude base mesh and then selective refinements [12], by using a bit-mask to identify vertices to be expanded [15] and, by using heuristics to maximize the effectiveness of the simplification passes [8].

Our approach most closely resembles POMAR [22], a view-dependent, random-accessible, compressed progressive scheme. POMAR uses edge-collapses to build LoDs, partitions the mesh into clusters, transmits the base mesh first, and then transmits refinement in batches that each increase the LoD for a selected set of clusters while ensuring balancing constraints (i.e., that LoDs of adjacent clusters do not differ by more than one). The eBits approach proposed here improves on POMAR by providing a more compact encoding of the refinements and a finer granularity for selecting which portion should be refined to full resolution. Specifically, using eBits instead of POMAR reduces (1) the number of bits that need to be transmitted per vertex to define the connectivity change for each refinement operation and (2) the number of *balancing vertices* which are not in the RoI, but must be transmitted to support the desired refinements. For example, when the original mesh is fully recovered, POMAR transmits between 4 and 6 bpt (depending on the size of the clusters), while eBits transmits only about 2 bpt.

4. Implementation details

In this section, we explain non-trivial implementation details and discuss alternatives that we have explored.

Data structure and mesh operations:

The working mesh is maintained both on the client and the server. We represent it using a simple extension of the Corner Table [23], which associates with triangle t three corners $3t$, $3t+1$, and $3t+2$ and which, for corner c, stores two integers: $V[c]$, which is the ID (denoted $c.v$) of the vertex at that corner, and $S[c]$ which is the ID (denoted $c.s$) of the next corner around $c.v$. Other corner operators may be derived trivially: the ID $c.t$ of the triangle containing c is $c/3$. Corners $c.n$ and $c.p$, which are the next and previous corners from c around triangle $c.t$ can be computed as $3c.t + ((c+1) \mod 3)$ and $3c.t + ((c+2) \mod 3)$. The term "around" is defined as clockwise around the outward normal at the vertex or triangle. The simplicity of this data structure and of its operations make it trivial to traverse the mesh and to update the connectivity when performing the t-collapse and v-expand operations. For example, the tip of each red arrow in Fig. 1 corresponds to a corner. The three corners identifying the cut-edges may be used as arguments of a v-expand function, which appends two new vertices and four new triangles to the Corner Table and updates the entries in the $V[]$ and $S[]$ tables of the corners associated with these triangles and their neighbors. We also store, as vertex annotations, on the server, the birth tags of triangles, the death tags of vertices and the universal vertex and triangles labels. The client maintains triangle birth tags, which it uses for identifying candidate r-triangles. Both the server and client also maintain an ***age*** tag for each vertex (see below).

Encoding cut-edges:

To expand a vertex v, the client needs to identify three ***cut-edges*** out of the d edges incident upon v in W, where $d = \deg(v)$ is the degree of v. To reduce transmission cost, the client and server use a tacit agreement. The edges are numbered $\{0, 1, \ldots, d-1\}$ clockwise around v, starting with the edge that joins v to the neighbor with the lowest ID. Hence, we need to encode three different integers $\{e_1, e_2, e_3\}$ in $[0, d-1]$. We use offsets $\{o_1 = e_1, o_2 = e_2 - e_1, o_3 = e_3 - e_2\}$. We have explored five encoding options listed below in order of decreasing average cost, specified in ***bpe (bits per vertex expansion)***. We used the 4th approach to produce the statistics in Section 5. We report here the associated cost or entropy of each alternative:

1. Send a mask of d bits: 7.27 bpe.
2. Compress that bit-stream using entropy coding: 7.09 bpe.
3. Eliminate trailing zeros from approach 1: 6.33 bpe.
4. Encode one of the $\lceil \log_2 \binom{\deg(v)}{3} \rceil$ choices: 5.78 bpe.
5. Use arithmetic coding on the bit strings $o_1 o_2 o_3$: 5.31 bpe.

Universal labels:

The client receives VERs ordered in accordance with the IDs of the current working mesh. The server, however, has no such luxury and must know where to look up this information. Universal labels for vertices and triangles with IDs t and v are denoted as $\bar{\mathbf{t}}$ and $\bar{\mathbf{v}}$, and defined as follows. The ***universal vertex label (UVL)*** of vertex v in the base mesh M_n is the same as its vertex ID: $\bar{\mathbf{v}} = v$. If v is not expanded in M_{n-1}, its UVL in M_{n-1} is $3\bar{\mathbf{v}}$. If it is expanded, its three children vertices are given UVLs $3\bar{\mathbf{v}}$, $3\bar{\mathbf{v}}+1$, and $3\bar{\mathbf{v}}+2$. The scheme extends to higher LoDs: a vertex with UVL $\bar{\mathbf{v}}$ in M_k will either correspond to a vertex with UVL $3\bar{\mathbf{v}}$ in M_{k-1} or will be expanded and its three children in M_{k-1} will have UVLs $3\bar{\mathbf{v}}$, $3\bar{\mathbf{v}}+1$, and $3\bar{\mathbf{v}}+2$. Such a scheme ensures that the tuple $\langle \mathit{UVL}, \mathit{birth\ tag} \rangle$ unambiguously identifies a single vertex across all the LoDs and all possible states of W and may be used by the server to index and query for VERs.

A similar scheme is used to define and track the ***universal triangle labels (UTLs)*** for the triangles. However, as a triangle that is once born persists in all subsequent refined meshes, we use a numbering that persists across the LoDs as well, to ease book-keeping. A triangle with UTL $\bar{\mathbf{t}}$ in M_k will be associated with UTL $4\bar{\mathbf{t}}$ in M_{k-1}. The four triangles created by the expansion of a vertex with UVL $\bar{\mathbf{v}}$ in M_k will be associated with UTLs $n_t + 4\bar{\mathbf{v}}$, $n_t + 4\bar{\mathbf{v}} + 1$,

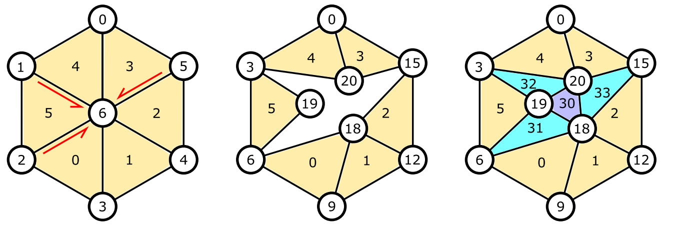

Fig. 5. UVLs and UTLs assigned during two consecutive passes: The base mesh M_1 (left) has six triangles and a single e-vertex v with UVL 6. The refined mesh M_0 (right) results from the v-expansion of v in M_1. The three a-triangles for v are those containing the cut-edges. The a-triangle with the smallest UTL (0 in this case), is selected as reference for assigning labels 18, 19 and 20 to the added vertices and labels 30, 31, 32 and 33 to the added triangles.

$n_t + 4\bar{\mathbf{v}} + 2$, and $n_t + 4\bar{\mathbf{v}} + 3$ in M_{k-1}, where n_t is the triangle count in M_k. Adding n_t ensures that each triangle in W has a different UTL.

We also specify precisely how these UVLs and UTLs are assigned for vertices and triangles resulting from a v-expansion. Triangle t is an ***a-triangle*** for an e-vertex v if there is a corner c such that $c.t = t, c.v = v$, and the edge e from vertex $c.p.v$ to v is a cut-edge (Fig. 1). v has 3 such a-triangles. Let t_0 be the a-triangle that has the lowest UTL, and e_0 be the corresponding cut-edge. We call t_0 and e_0 the ***first cut-triangle*** and the ***first cut-edge*** respectively. Let c be the corner for which $c.t = t_0$ and $c.v = v$. We use c to define the assignment of UVLs and UTLs to the new vertices and triangles created by the v-expansion of v (see Fig. 5). For example, the new vertex inserted at the corner c of t_0 will be labeled $3\bar{\mathbf{v}}$. Similarly, the triangle inserted in the cut-edge e_0 will have UTL $n_t + 4\bar{\mathbf{v}}$.

We also use a simple convention to select consistently the correct cyclic assignment of consecutive integer IDs to the corners of each new triangle created by a v-expansion—the first corner, c, of each new triangle is not incident upon a new vertex (i.e., $c.v$ is a tip-vertex).

In the discussion so far, the server orders VERs by the vertex IDs of the working mesh. As the client and server working meshes are synchronized, the client does not need to maintain UVLs or UTLs. These labels are only used by the server to access the information required to construct VERs and to encode the VERs. The information required to construct VERs is indexed by UVLs. Because UVLs are not contiguous integers, to reduce memory usage, we use a hash-map with the UVL serving as the key, for accessing the VER information. In collaborative environments where all users view the same RoI, a single working mesh W suffices. To support multiple independent clients, the server needs to maintain a different copy of W per client. However, the VERs are identical for all client and thus need not be replicated. Additionally, the UVLs and UTLs allow the server to offload much of its processing to the client and thus handle very large client counts, in the following manner—as previously mentioned, the working mesh data-structure is augmented and each vertex and each triangle of the working mesh W is associated with its UVL or UTL. This association is maintained as the candidate vertex sets are identified and some of their vertices expanded. Such book-keeping is not necessary on the client, since an ***active server*** "translates" the UVLs and UTLs into integer IDs in the Corner Table representation of the working mesh on the client. However, doing this book-keeping as well allows the client to interpret the content of the VERs without the help of an active server. In such a ***thin server*** architecture, the server becomes a passive database. The slightly more expensive, but computationally cheaper encoding scheme of sending a 0/1 bit for each e-vertex edge, minus the trailing 0's, may also be employed to further reduce server computation.

Birth and death tags: Birth and death tags are computed on the server during the simplification process. A t-collapse in pass k collapses three child-vertices into their parent-vertex, v. We say that v dies during the reverse expansion process at LoD k. Hence, we set the death tag of v to k and the birth tags of its three children to k.

Age tags: The working mesh W contains vertices at different LoDs. For each change of RoI, the server and client identify the set U of vertices in the RoI that are not at full resolution and the set I of vertices that must be expanded first to make these balanced, and so on, recursively. VERs for both U and I are transmitted by the server. To help identify these, the client maintains, for each vertex v of W, an ***age tag***. When v is born, its age is initialized to its birth tag. Then, it is incremented at each pass to keep track the current age of v.

Vertex mask: We have also implemented and tested the following alternative to let the client know which vertices need to be expanded. Instead of sending death tags for the vertices of the base mesh and for the 3 child-vertices created by each vertex expansion, the server can send a bit mask for a set of well defined candidate vertices of W in the RoI and for rings of edge-adjacent vertices. The rings are used to ensure balancing constraints and the masks are sent in order of decreasing LoDs.

For each age a (all our experiments use 7 LoDs and thus, $a \in [1, 7]$), the client interprets the bit-stream it is receiving as BER_a in the following manner: For each vertex v in the RoI that has age a, the client receives a bit for the vertex mask. If that bit is 0, the client knows it does not need to expand v at age a, and hence it increments its age. Otherwise, the client knows that it will receive a VER for the vertex, and marks it as an e-vertex. However, the client may need to first expand some of the neighbors of v to ensure that v is balanced. Hence, for each neighbor u of v, if the age of u is $a - 1$ (note that it can never be lower), the client receives a vertex mask bit for u and either increment u's age (if the bit is 0), or iterates recursively the above steps for $\langle u, a - 1\rangle$ (if the bit is 1). At the end of the recursive process, the client has identified the e-vertices. For each BER_a batch, the client receives the VERs in order of increasing $\langle age, vertex\ ID\rangle$ tuple.

To compare the effective costs of both solutions (sending death tags or sending vertex masks) we use the ***effective entropy*** (***EE***), which we define as the entropy of the corresponding bit stream multiplied by the total number of vertices created during the refinement and divided by the total number of such vertices that have been created at the highest LoD in the RoI.

Our experiments indicate that transmitting the entire mesh has an EE of about 1.18 bits when sending the death tags and an EE of 1.25 bits when sending the vertex masks. When the RoI is small, the grow operation has lower transmission cost per RoI vertex when using bit-masks than death tags. For this reason, and because they are trivial to decode, we report our performance results using our bit mask implementation.

Encoding of the geometry and other vertex attributes:

The 3 cut-edges identify the 3 new vertices in the refined working mesh. For example, the first two new vertices bound the b-triangle that collapsed to the first cut-edge. When we collapse a c-triangle t, we place the new vertex v at its centroid G and encode, in the VER, two vectors from G to the first and second vertices of t. This information suffices to recover, during the expansion of v, the locations of its three child vertices. This solution uses fewer coordinates than encoding the locations of the three child-vertices. Furthermore, these two vectors are typically shorter than vectors between child vertices, and hence, may yield more compact encodings when quantized.

Additionally, we may use the local connectivity and geometry information in a particular RoI to predict the location of each new vertex and transmit a compressed encoding of the correction. Prior approaches for extrapolating prediction achieve, depending on the

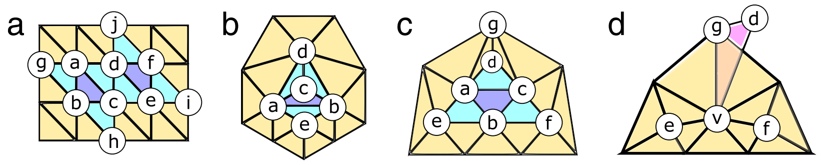

Fig. 6. Invalid configurations of c-triangles: (a) Two clusters share an edge. (b) Vertex c of the c-triangle (blue) has degree 3. (c) A b-triangle bounded by a vertex with degree 3 (vertex d) produces an ear as shown in (d). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

smoothness of the mesh, about 5 bits per coordinate for geometry that is quantized to 12 bits per coordinate [24]. Vertex attributes may also be compressed using existing techniques such as [25], which uses prediction and quantization to encode normals using about 6 bits per vector. Hence, reducing the cost of transmitting connectivity has a non-negligible impact on the total cost, unless several attributes are needed.

Constraints on sets of c-triangles:

On the server, each simplification pass identifies a set of c-triangles. The ***cluster*** of a c-triangle t comprises t and its three edge-adjacent b-triangles. The ***tip*** (or tip-vertex) of a b-triangle of a cluster is the vertex of that triangle that does not bound the c-triangle of the cluster (Fig. 1).

To avoid creating non-manifold topologies and to ensure that we can unambiguously identify the three cut-edges of an e-vertex v, regardless of which other e-vertices of the same age (belonging to the same LoD) have already been expanded, we impose several validity constraints: (a) No two c-triangles share a vertex; (b) No two b-triangles share an edge; and (c) The three vertices of a b-triangle do not have a common neighbor. Fig. 6 shows examples of invalid configurations that violate one or more of these constraints.

Constraints (a) and (b) ensure that clusters do not share edges, and also imply that each cluster has a border of exactly 6 edges (is a Hajós graph). Furthermore, condition (b) ensures that no c-triangle has a vertex of degree 3 (see Fig. 6(b)). Constraint (c) ensures that the mesh remains manifold. Violating it may produce an “ear” where two triangles of opposite orientations share the same 3 vertices (see Fig. 6(c) and (d)). While our eBits scheme can be modified to support such pseudo-manifold configurations, for simplicity and fair comparison with other schemes, we forbid this configuration in the experiments reported here. A corollary of constraint (c) is that a b-triangle cannot have a degree 3 vertex. Edge contractions that preserve topology have been studied in [26]. Criteria for valid collapses are discussed in [27] and used in [28] to avoid ambiguities in the specification of vertex-splits. Because a degree 3 vertex prevents t-collapses in subsequent LoDs, we wish to reduce the probability of creating degree 3 vertices and hence impose an additional constraint: (d) a tip-vertex cannot have a ***residual degree*** of 4 or less. The residual degree of a vertex, v, in LoD M_k is the degree that its parent has in M_{k+1}.

Algorithm for selecting c-triangles:

The ***simplification ratio*** of simplification pass k is defined as the ratio of the vertex count of M_k to the vertex count of M_{k-1}, and hence is 1 or less. The choice of c-triangles affects the degree distribution of the mesh. In general, simplification increases the entropy of degrees. This observation guides our strategy for selecting sets of c-triangles.

The degree $\deg(v)$ of e-vertex v may be computed from the degrees of its three children g, h, and i (see Fig. 1):

Property 1 : $\deg(v) = \deg(g) + \deg(h) + \deg(i) - 9$

Proof: This property holds (for manifold meshes without boundary) because v inherits all the corners of g, h, and i except for the 9 corners that are contained in the triangles of the cluster and incident upon the three children of v.

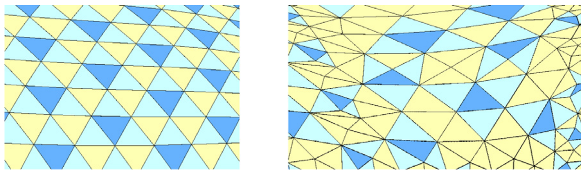

Fig. 7. The clusters (blue and cyan) produced by the spiraling algorithm for a regular mesh region (left) results in a simplification ratio of 0.33. Note how, in these clusters, all 3 tip-vertices are incident on a c-triangle (all the 3 c-triangle vertices are shared with other clusters). Thus, the e-vertices produced on collapsing these clusters are of degree 6. The clusters produced by the spiraling algorithm for an irregular mesh region (right), are less densely packed. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

For good compression performance, we want low and consistent simplification ratios across LoDs. We start by briefly describing two approaches that we initially considered and explain their shortcomings. Then, we present our final ValPack (valence driven packed) traversal algorithm.

Naive eager traversal: The first scheme that we have considered is an eager scheme that tests triangles one by one in order of their IDs in M_k and selects as c-triangles those that pass the validity test. Unfortunately, this simple strategy leads to deteriorating simplification performance over consecutive LoDs (Fig. 10) because it tends to create a high ratio of degree 3 and 4 vertices, which increase the simplification ratio of subsequent passes. Furthermore, the average degree of e-vertices created by the naive approach is high (9.15 for the subdivided horse mesh), which implies a greater cost for encoding the VERs.

Spiraling eager traversal: A better approach spirals out from a random starting corner, eagerly marking any valid c-triangle. It may be implemented trivially (without using a stack or recursion) by using a simple variation of the RingExpander algorithm of [29]. We also carry out a naive traversal after the spiral to ensure maximality of the c-triangle set. This approach works extremely well on regular portion of a mesh, where most vertices have degree 6 (Table 1). In such regions, all vertices of a c-triangle are shared with nearby clusters and, thus, such a region remains regular after the t-collapses (Fig. 7). For example, for a four times subdivided small horse mesh, the simplification ratio is 0.44 for the first pass using this strategy. In irregular regions, however, the spiraling algorithm does not fare that well (see Fig. 7). The spiraling algorithm is dependent on the starting corner. We perform several spiraling traversals from randomly selected initial corners and retain the best one. Furthermore, even for highly regular meshes, this approach produces successive simplified LoDs in which degree 4 vertices become increasingly dominant. For example, in the subdivided horse, M_0 has 99% regular (degree 6) vertices and <1% degree 4 vertices. After two simplification passes, M_2 has only 32% degree 6 vertices, but 43% degree 4 vertices (see Fig. 8).

ValPack: Let the ***degree of a triangle*** denote the sum of the degrees of its 3 vertices. Note that, given the constraints discussed above, the degree of a c-triangle is at least 12. Recall that the t-collapse of a c-triangle with degree v produces a vertex with degree $v - 9$. Hence, collapsing a degree 12 triangle yields a degree 3 vertex.

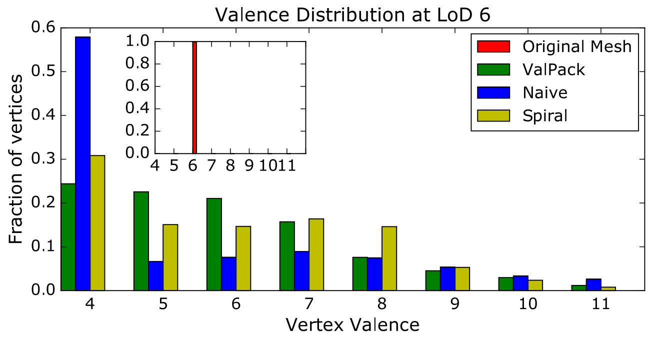

Fig. 8. Histogram of degrees in the subdivided horse mesh (M_0, inset) and after the 6th simplification pass using the naive, spiral and ValPack approaches.

Table 1
The importance of both degree driven selection and regular mesh packing in ValPack: For two meshes with very different regularity, the cumulative product of the simplification ratios across all 7 LoDs obtained: (a) by using just degree driven selection (followed by a naive traversal to obtain a maximal valid c-triangle set), (b) by the spiraling eager traversal and (c) by using ValPack. ValPack performs consistently irrespective of the mesh regularity.

Mesh	Degree driven	Spiral	ValPack
Subdiv Horse	0.063	0.025	0.020
Buddha	0.039	0.074	0.038

Note that collapsing a triangle of degree 15 that has three degree 5 vertices, yields a vertex with degree 6, effectively correcting 3 low degree vertices. So, we could consider giving priority to triangles of low degree when selecting c-triangles. Also, a t-collapse of a triangle reduces the degree of each of its tip vertices by one. Thus, if we wish to prevent the creation of low-degree vertices, we could consider giving priority to clusters where the minimum degree across the 3 tip vertices is high. Such degree driven heuristics perform well for many meshes. However, they tend to produce sub-optimal distributions of clusters that are not densely packed in regular regions, hence increasing the simplification ratio of the simplification pass (Table 1).

Based on these observations, we propose a 3-step solution for our Degree Driven Packed traversal (ValPack):

1. *Degree driven selection*: Our strategy is to prioritize c-triangles that have the lowest degree. First, we use eager selection and an ID ordered traversal to identify candidate c-triangles, placing triangles that have degree between 13 and 17 (both inclusive) in a degree ordered min priority queue. We then visit each triangle in the queue sequentially, check if it is a valid c-triangle, and if so, mark its cluster (mark it as a c-triangle, and its adjacent triangles as b-triangles).
2. *Regular mesh packing*: We wish to ensure that we produce a tight packing of clusters in the regular regions (where all vertices have degree 6 and hence all triangles have degree 18). Note that these have not been affected by step 1. To this effect, we perform the spiraling order traversal of the mesh as described above, spiraling out from each of the c-triangles produced at the end of the priority selection step. We use the eager spiraling traversal to produce a tightly packed configuration of additional c-triangles in regular mesh regions.
3. *Hole filling*: Finally, we augment the set of c-triangles marked by the above steps with other c-triangles discovered by a final eager traversal.

An additional benefit of ValPack apart from the consistently good simplification performance (as seen in Table 1) is that the lower degree of the collapsed triangles, and hence of the resulting e-vertices (due to the degree driven selection step), allows us to encode the v-expand operations with fewer bits than approaches

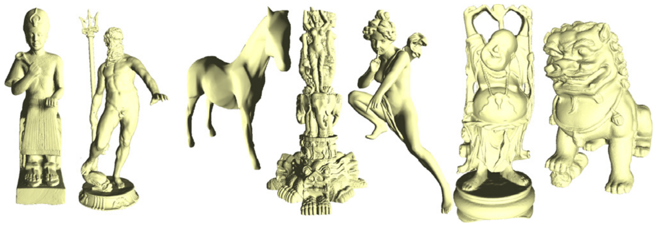

Fig. 9. The meshes used in our experiments.

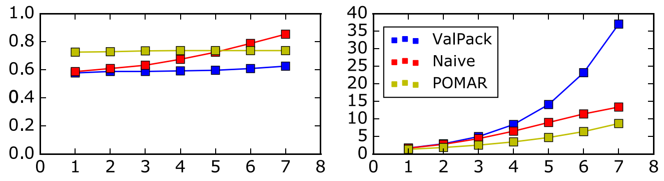

Fig. 10. (a) Ratio of vertex counts in M_i to vertex counts in M_{i-1} for different LoDs i. (b) Ratio of vertex counts in M_0 to vertex counts in M_i. Both ratios are for the Ramses model.

that are not degree driven. We obtain an average e-vertex degree of 7.3 bits for eBits for our test meshes.

Base mesh compression:

The connectivity of the base mesh is compressed using EdgeBreaker [3]. Although EdgeBreaker combined with a Huffman coding [30] of consecutive pairs of symbols yields about 1 bpt, it is less effective for highly irregular meshes, such as our base meshes. Hence, we use the standard EdgeBreaker format [1], which guarantees 2.0 bpt. Other compression schemes could be used for transmitting the base mesh, but most are less effective than EdgeBreaker for highly irregular meshes. When the base mesh has about 3% of the triangles of the original mesh, the effective cost (amortized per triangle of the original mesh) of transmitting the base mesh connectivity to the client is approximately 0.06 bpt of the original mesh.

5. Results

Simplification effectiveness:

As shown in Fig. 10(a), our naive traversal algorithm for selecting c-triangles yields a rapidly increasing simplification ratio. POMAR [22] maintains a steady simplification ratio of about 0.73. Our ValPack solution produces a lower, although slowly increasing simplification ratio of about 0.60. (POMAR prevents edge-collapses that produce flips of a triangle normal. Therefore, for fairness, we also disabled triangle-collapses that do so in ValPack for this comparison.) This improvement of the simplification performance is essential because it reduces the number of simplification passes needed to produce a relatively coarse base mesh and hence reduces the cost of encoding the death tags or the number of vertex masks that would need to be sent. Fig. 10(b) captures the cumulative effect of i consecutive simplification passes and shows that in 7 passes ValPack produces a base mesh 4 times smaller than the one produced by POMAR.

When using ValPack, 7 simplification passes produce a base mesh with about 2.5% of the original triangle count. This ratio is slightly higher for the Buddha mesh, which has highly irregular connectivity [29]. The simplification ratios for the LoDs of different meshes are tabulated in Table 2. See Fig. 9 for a visualization of these meshes.

Transmission costs:

We present our transmission cost results for streaming the entire mesh, and also for servicing the three client interaction requests of pick, grow and slide as defined in Section 2.

Table 2
The simplification ratios R_i obtained using ValPack for successive simplification passes and their cumulative product $\prod = \prod_{i=1}^{7} R_i$. Rightmost column: $C =$ the cost (in bpt) for transmitting the connectivity of the entire mesh using the eBits format.

Mesh (vertex count)	R_1	R_2	R_3	R_4	R_5	R_6	R_7	$\prod$	C
Big Horse (48 485)	0.53	0.59	0.59	0.59	0.59	0.59	0.60	0.023	2.15
Subdiv Horse (64 002)	0.44	0.57	0.60	0.59	0.60	0.61	0.61	0.020	1.93
Angel (237 018)	0.56	0.59	0.59	0.59	0.60	0.61	0.62	0.026	2.19
Buddha (543 652)	0.59	0.59	0.60	0.61	0.63	0.66	0.71	0.038	2.20
Dragon (655 980)	0.47	0.57	0.58	0.59	0.60	0.61	0.64	0.020	1.93
Ramses (826 266)	0.58	0.59	0.59	0.59	0.60	0.61	0.62	0.025	2.16
Neptune (2 003 932)	0.52	0.58	0.59	0.59	0.59	0.60	0.61	0.022	1.94
Thai (4 999 996)	0.57	0.59	0.59	0.59	0.59	0.60	0.62	0.025	2.15
Average	0.53	0.58	0.59	0.59	0.60	0.61	0.63	0.025	2.08

Connectivity cost for reconstructing M_0:

The connectivity transmission cost for expanding the entire mesh (when it is completely contained in the RoI) is shown in Table 2. The cost includes the transmission of the base mesh, but not the cost of transmitting geometry. The cost was computed using an enumerative encoding of the three cut-edges of a vertex v (thus using $\lceil \log_2(\binom{\deg(v)}{3}) \rceil$ bits) and using vertex masks. Note that the numbers are competitive with single rate mesh compression algorithms (e.g., Edgebreaker [3,1] guarantees 1.84 bpt).

Connectivity cost for servicing client interactions:

We call the ratio of the total number of bits transmitted to the number of full resolution vertices created the ***Effective Transmission Cost*** (ETC) per vertex, and divide this number by 2 to approximate the ETC per triangle. The number of bits required for servicing different client interactions (see Section 2) are specified as ETC per triangle.

Picking: Picking is a parameter-less operation. We measure the ETC for a number of pick operations across all meshes, with the working mesh initialized to the base mesh for each of our experimental runs. The average pick cost is 154 bits, with a high variance. The pick operation translates a base mesh vertex v to a descendant f in M_0 by drilling down the family tree of v. f is initialized to v and is replaced by one of its children recursively. In this process, if the resulting f has a birth tag different from 0, just one full resolution vertex is created (this is f itself). In other cases, v will be expanded to create 3 full resolution vertices. This variation affects the ETC scores and makes them vary by multiples of 3. The pick cost is correlated with the regularity of the region (in M_0) being picked, due to our good packing in such regions. Thus, the average pick cost for the highly regular subdivided horse mesh (136 bits) is less than the average pick cost of the irregular Buddha mesh (182 bits).

Growing and sliding: The practical benefits of our format are highlighted when random access to nearby regions of the mesh is desired, such as in terrain flybys. In such scenarios, the ETC is amortized over neighboring vertices. When we enforce balancing constraints, we create a smooth transition of LoDs from the most refined to the least refined vertices (Fig. 4). Thus, the BERs servicing a RoI request also contain information that can be leveraged for servicing requests for neighboring RoIs.

Both grow and slide (given a desired focus) are dependent on the current size of the RoI (which is measured in terms of edge-distance from the focus vertex in M_0). Shrinking the RoI has no cost associated with it—it may only result in some t-collapses being performed on W on both the client and server, depending on the cleaning policy. In our proposed interactions, to grow a RoI to size r at a newly picked location, the client requests r grow operations to get a desired size of the RoI. Following this, we envision that the client would make small changes to r. We thus report the ETCs for growing to a particular RoI size r at a newly picked to location

Table 3
For different RoI sizes (measured in terms of edge distance r at full resolution), we show $V(r)$: the average number of vertices in the RoI and the ETC (bpt) for three operations: *Grow to r* (iterative growing the RoI from size 0 to size r), *Grow* (from $r-1$ to r), and *Slide* by one edge.

r	1	5	10	20	30	40	50
$V(r)$	1	88	395	1793	4455	8426	13 655
Grow to r	–	5.00	3.53	2.87	2.56	2.45	2.38
Grow	11.46	3.74	2.96	2.50	2.29	2.27	2.20
Slide	19.31	4.20	2.77	2.36	2.31	2.28	2.16

(this is a cumulation of grow(i), $i \in \{0, \ldots, r-1\}$), and the cost of growing a RoI of size r in Table 3. The numbers follow a similar trend across meshes so we report the across all test meshes. The difference between the subdivided horse and the Buddha meshes is less than 0.3 bpt across all values of $r > 1$ for both types of grow operations.

For sliding, we observe that our ETC costs for directional slide operations (i.e., sliding along edges of M_0 that are oriented similarly in space), are mostly independent of the number of edges we slide along. Thus, we present ETC costs for sliding along one edge of M_0 (while ensuring an r-ring of edges around the new focus is at full resolution). Our results are presented in Table 3. The ETC for sliding drops quickly with r, and is relatively similar across all meshes. We thus report the average across the meshes. When the sliding motion is not directional, but random, the ETC costs reported could decrease further, due to greater amortization of the ETC if the cleaning strategy is LRU.

Performance: Our tests are carried out on a machine with 32 GB RAM, a 2.7 GHz processor (8 cores, but a, single threaded implementation) and 4 SSDs in RAID-0. The simplification process is linear in the number of vertices and takes around 93 s for 7 LoDs of the Neptune mesh to be generated (this includes the I/O time to write the VERs to disk). The average time required for collating and applying VERs to perform a grow operation, for different values of RoI size r for the some meshes are plotted in Fig. 11. While our implementation of the grow algorithm is linear in the RoI size, our non-optimized, single-threaded implementation can be improved upon. The set operations performed by grow are parallelizable. Additionally, our C++ code also spends a non-trivial fraction of its execution time for memory allocation. Thus, using a custom memory allocator could also offer significant performance gains.

6. Comparison with prior art

As discussed in Section 3, using single rate compression, such as Edgebreaker, for encoding the connectivity of the portion of the mesh recently conquered by the RoI is likely to exceed 3 bpt.

The most relevant prior art is the recently proposed POMAR [22], a random accessible, progressive and loss-less manifold triangle mesh compression algorithm. It provides a compressed file, which can be used as a database of records needed by the

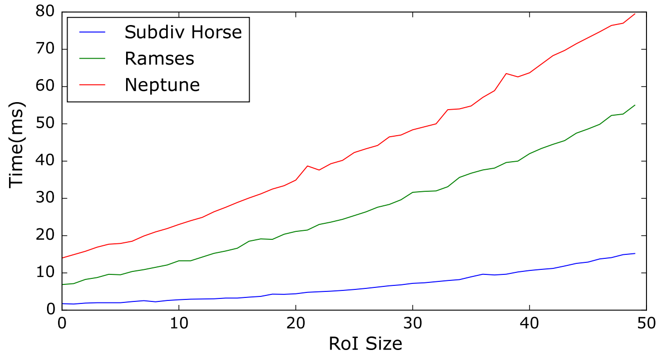

Fig. 11. The time taken (ms) for grow operations (growing the RoI by one ring), as a function of the RoI size.

client for refining the mesh after each change of RoI. Similarly to eBits, POMAR produces a base mesh through a series of simplification batches, yielding a sequence of simplified LoDs, each obtained from the previous one by a series of *half-edge collapses*. During refinement, the first *n* refinement batches increase the resolution of all the vertices (i.e., the refinement is global, not adaptive). These *n* refinements yield the *base clustered mesh*. A cluster is defined as the set of vertices that collapse to the same vertex in the base clustered mesh during simplification. At each LoD, POMAR allows a sequence of multiple half-edge collapses onto a single vertex v. The collapse of a half-edge *e* affects the configuration of a set of faces called its patch. POMAR constrains half-edge collapses so that once half-edge *e* has been collapsed, no half-edge collapse whose patch intersects with *e*'s patch may be collapsed.

We highlight four advantages of eBits over POMAR:

Better simplification: Using t-collapses, eBits allows the collapse of pairs of edges that are incident upon the same triangle and also allows for the vertices of one patch to be collapsed into a vertex of another patch. Thus, each eBits simplification pass removes a higher fraction of vertices, which, in turn, reduces the total number of LoDs required to obtain the same number of base mesh vertices. Fig. 10 compares the number of vertices at successive LoDs for POMAR and eBits. As a direct result of the improved simplification, eBits has improved retrieval ratios (see below). The reduction in LoDs is also leveraged by eBits to reduce the encoding cost. These two ideas are detailed below.

Improved retrieval ratios: Both POMAR and eBits fetch additional vertices than those requested by the client to ensure balancing constraints are met. Due to better simplification ratios, eBits produces a smaller number of LoDs than POMAR. Thus, it requires fewer "balancing" vertices to be sent in response to an expansion request. For example, we see a quantitative measure of the difference between POMAR and eBits when the client requests for a single base mesh vertex to be fully expanded so as to produce all of its full resolution descendants. We define the ***retrieval ratio*** as he ratio of the number of these full resolution descendants to the total number of vertices received (which includes balancing vertices and ancestors). For comparison, we use a mesh (Ramses model) simplified to the same number of vertices with eBits and POMAR, eBits has a retrieval ratio that is an order of magnitude higher: 0.212 for eBits vs. 0.0268 for POMAR. Note that in this comparison, we have disabled the global LoDs in POMAR, because these would further decrease its retrieval ratio.

Granularity of access: Apart from the improved simplification ratios, the improvement in the retrieval ratio is also explained by the fact the eBits encodes each triangle collapse independently, indexable by the unique labels, as opposed to POMAR, which groups edge collapses into clusters. The ability to index individual vertex expansion bits provides eBits with finer granularity of access, allowing it to transmit a much smaller set of VERs.

Encoding compactness: Due to a smaller number of LoDs and finer granularity of access, the vertex mask variants of eBits sends fewer vertex mask bits than what would have been sent by POMAR. Also, POMAR does not maintain a working mesh on the server and hence cannot take advantage of the compact encoding proposed in eBits. Instead, for each vertex split, POMAR encodes the information needed to identify two out of the $\deg(v)$ edges incident upon v. POMAR guesses which of the incident edges is the reference edge and uses entropy encoding to transmit a correction of that guess, plus two offsets relative to the reference edge that identify the two split edges.

The reported transmission cost for POMAR depends on the cluster size and is about 3.90 bpt for clusters having around 200 vertices. eBits, while using a very simple encoding, achieves a transmission cost of 2.08 bpt which is a significant improvement. Even with the more stringent ETC measure, eBits offers effective transmission costs that quickly approach the asymptotic transmission cost mentioned above.

Conclusion

We have presented a compact format, called eBits, for transmitting the local connectivity of the subset of a triangle mesh that lies inside the Region of Interest (RoI) as the RoI is moved in a user-controlled manner. When the RoI includes the entire mesh, eBits transmission cost is only 2.08 bpt, including the cost of transmitting the base mesh. When the RoI is smaller the, Effective Transmission Cost approaches 2.08 bpt—for example, if the RoI has 2000 vertices, the ETC is about 2.5 bpt.

There are numerous avenues for future exploration. As it stands, eBits provides a block level random accessible compressed triangle mesh format for remote visualization. By transmitting mesh connectivity and geometry instead of server rendered images, eBits allows for client side geometry processing. While multiple clients can be trivially served by maintaining a single copy of the VER table on the server, saving client side edits of local geometry and connectivity on the server is an interesting topic of future research.

Isosurfaces from structured grids are often ordered along a spatial axis. An easy option for random access of such meshes is to transmit the entire slices stabbed by the RoI. eBits sends only portions of these slices inside the RoI (plus balancing rings). However, eBits does not exploit the spatial ordering itself, which could be used to devise efficient out of core simplification algorithms for the server side pre-processing.

Finally, the current implementation of eBits does not support non-manifold meshes or meshes with boundaries. Meshes with boundaries can be supported by extending the encoding of t-collapses to also account for border c-triangles or by preventing the collapse of border triangles. A watertight non-manifold mesh (when each edge has an even number of incident faces) could be converted to a manifold representation using MatchMaker [31]. Handling more general non-manifold meshes requires further research.

Acknowledgments

This work was performed in part under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This material is based upon work supported by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

References

[1] King D, Rossignac J. Guaranteed 3.67 v bit encoding of planar triangle graphs. In: Proceedings of the 11th Canadian conference on computational geometry, CCCG'99, 1999.

[2] Weiss K, De Floriani L. Bisection-based triangulations of nested hypercubic meshes. In: Proceedings of the 19th international meshing roundtable, 2010, pp. 315–333.
[3] Rossignac J. Edgebreaker: Connectivity compression for triangle meshes. IEEE Trans Vis Comput Graphics 1999;5(1):47–61.
[4] Ho J, Lee KC, Kriegman D. Compressing large polygonal models. In: Proceesings of visualization, 2001, VIS'01, 2001, pp. 357–573.
[5] Taubin G, Guéziec A, Horn W, Lazarus F. Progressive forest split compression. In: Proceedings of SIGGRAPH 98, 1998, pp. 123–132.
[6] Taubin G, Rossignac J. Geometric compression through topological surgery. ACM Trans Graph 1998;17(2):84–115.
[7] Choe S, Kim J, Lee H, Lee S. Random accessible mesh compression using mesh chartification. IEEE Trans Vis Comput Graphics 2009;15(1):160–73.
[8] Khodakovsky A, Alliez P, Desbrun M, Schröder P. Near-optimal connectivity encoding of 2-manifold polygon meshes. Graph Models 2002;64(3):147–68.
[9] Lee H, Alliez P, Desbrun M. Angle-Analyzer: A triangle-quad mesh codec. Comput Graph Forum 2002;21(3):383–92.
[10] Yoon SE, Lindstrom P. Random-accessible compressed triangle meshes. IEEE Trans Vis Comput Graphics 2007;13(6):1536–43.
[11] Isenburg M, Lindstrom P. Streaming meshes. In: Proceedings of visualization, 2005, VIS'05, 2005, pp. 231–238.
[12] Hoppe H. Progressive meshes. In: Proceedings of SIGGRAPH 96, 1996, pp. 99–108.
[13] Xia JC, El-Sana J, Varshney A. Adaptive real-time level-of-detail based rendering for polygonal models. IEEE Trans Vis Comput Graphics 1997;3(2):171–83.
[14] Hoppe H. View-dependent refinement of progressive meshes. In: Proceesings of SIGGRAPH 97, 1997, pp. 189–198.
[15] Pajarola R, Rossignac J. Compressed progressive meshes. IEEE Trans Vis Comput Graphics 2000;6(1):79–93.
[16] Pajarola R, Rossignac J. Squeeze: Fast and progressive decompression of triangle meshes. In: Proceedings of computer graphics international, 2000. IEEE; 2000. p. 173–82.
[17] Alliez P, Desbrun M. Progressive compression for lossless transmission of triangle meshes. In: Proceedings of SIGGRAPH 01, 2001, pp. 195–202.
[18] Gandoin P-M, Devillers O. Progressive lossless compression of arbitrary simplicial complexes. ACM Trans Graph 2002;21(3):372–9.
[19] Valette S, Chaine R, Prost R. Progressive lossless mesh compression via incremental parametric refinement. Comput Graph Forum 2009;28(5):1301–10.
[20] Kim J, Choe S, Lee S. Multiresolution random accessible mesh compression. Comput Graph Forum 2006;25(3):323–31.
[21] Kim J, Lee S. Truly selective refinement of progressive meshes. In: Proceedings of graphics interface 2001, GI'01, 2001, pp. 101–110.
[22] Maglo A, Grimstead I, Hudelot C. POMAR: Compression of progressive oriented meshes accessible randomly. Comput Graph 2013;37(6):743–52.
[23] Rossignac J, Safonova A, Szymczak A. Edgebreaker on a corner table: A simple technique for representing and compressing triangulated surfaces. In: Hierarchical and geometrical methods in scientific visualization. 2003. p. 41–50.
[24] Ibarria L, Rossignac J. Dynapack: space–time compression of the 3D animations of triangle meshes with fixed connectivity. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation, 2003, pp. 126–135.
[25] Ahn JH, Kim CS, Ho YS. Predictive compression of geometry, color and normal data of 3-d mesh models. IEEE Trans Circuits Syst Video Technol 2006;16(2):291–9.
[26] Dey TK, Edelsbrunner H, Guha S, Nekhayev DV. Topology preserving edge contraction. Publ Inst Math (Beograd) (NS) 1999;66(80):23–45.
[27] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Mesh optimization. In: Proceedings of SIGGRAPH 93, 1993, pp. 19–26.
[28] Diaz-Gutierrez P, Gopi M, Pajarola R. Hierarchyless simplification, stripification and compression of triangulated two-manifolds. Comput Graph Forum 2005;24(3):457–67.
[29] Gurung T, Luffel M, Lindstrom P, Rossignac J. LR: Compact connectivity representation for triangle meshes. ACM Trans Graph 2011;30(4).
[30] Salomon D, Motta G. Handbook of data compression. Springer; 2010.
[31] Rossignac J, Cardoze D. Matchmaker: Manifold BReps for non-manifold r-sets. In: Proceedings of the 5th ACM symposium on solid modeling and applications, 1999, pp. 31–41.