LR: Compact connectivity representation for triangle meshes

T. Gurung, M. Luffel, P. Lindstrom, J. Rossignac

January 31, 2011

ACM Transactions on Graphics
Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.
Figure 1: The ring (black loop) delineates two corridors of triangles. Normal $T_1$ triangles (cream/orange) have one ring edge, dead-end $T_2$ triangles (blue) have two ring edges, and $T_0$ triangles (green) comprising bifurcations have no ring edges. Adjacent $T_0$ (gray/red) and $T_2$ triangles (left) are represented internally as inexpensive $T_1$ triangles (right), thereby significantly reducing storage. Our LR representation supports random access to connectivity, storing on average only 1.08 references or 26.2 bits per triangle.

Abstract

We propose LR (Laced Ring)—a simple data structure for representing the connectivity of manifold triangle meshes. LR provides the option to store on average either 1.08 references per triangle or 26.2 bits per triangle. Its construction, from an input mesh that supports constant-time adjacency queries, has linear space and time complexity, and involves ordering most vertices along a nearly-Hamiltonian cycle. LR is best suited for applications that process meshes with fixed connectivity, as any changes to the connectivity require the data structure to be rebuilt. We provide an implementation of the set of standard random-access, constant-time operators for traversing a mesh, and show that LR often saves both space and traversal time over competing representations.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations

Keywords: triangle meshes, mesh connectivity, Hamiltonian cycle

1 Introduction

Compact triangle mesh representations that support random access are of increasing importance, given the rising complexity of meshes handled by applications and the proliferation of mobile and multi-core architectures. Compact representations help to reduce 1) the frequency of page faults, 2) the cost of swapping mesh portions between processors, and 3) the amount of memory required for storing a complete scene on a GPU or game console.

Our contributions are best explained as a storage-saving modification of the Corner Table (CT) [Rossignac 2001], which for each triangle stores 3 integer references to its vertices in the $V$ table and 3 references to opposite corners in adjacent triangles in the $O$ table. In contrast, the LR (Laced Ring) representation proposed here for manifold triangle meshes with fixed connectivity can be used to reduce storage for the connectivity information to either about 1.08 rpt (references per triangle) or to only about 26.2 bpt (bits per triangle), based on averaging the storage costs for our benchmark models. In a CT representation with 32-bit references and 16-bit vertex coordinates, the connectivity accounts for 90% of the total storage cost. LR does not require any particular compression of the vertex geometry, but we assume that memory-constrained applications will favor 16-bit coordinates. Under these conditions, using LR instead of CT results in a 75% reduction in total storage.

In spite of its compactness, LR supports the full set of standard random-access operators, including all those supported by CT, plus the vertex-to-incident-triangle (star) reference. These operators provide random access from an element (vertex, edge, or triangle) to adjacent elements, and permit visiting the vertices of a triangle and the triangles or edges incident upon a vertex in the cyclic order defined by the orientation of the mesh. We provide the details of a practical and efficient implementation of these operators, which each have constant-time complexity.

This significant progress over prior art builds on the following novel contributions.
Ring-based ordering: We build a nearly-Hamiltonian cycle of primal mesh edges that we call the ring. It divides the mesh into two parts (Fig. 1) that form triangle strip corridors with bifurcations. To reduce storage, we classify triangles by the number of edges they have on the ring (bifurcation $T_0$, normal $T_1$, dead-end $T_2$). We store the ring vertices and the $T_1$ and $T_2$ triangles in the order in which they are visited by the ring. The isolated vertices not part of the ring are stored last. The $T_0$ triangles are stored using the standard CT data structure.

Omitted $V$ entries for $T_1$ and $T_2$ triangles: Most triangles are of type $T_1$ or $T_2$. Two of their vertex references ($V$ entries of the CT) are defined implicitly and need not be stored. Thus, we store two references, $L[v]$ and $R[v]$, per ring vertex and assume that triangle $2v$ has vertices $(v, L[v], v+n)$ and triangle $2v+1$ has vertices $(v, R[v], v)$, where $v+n = (v+1) \mod m_r$ is the next vertex after $v$ on the ring, and where $m_r$ denotes the number of ring vertices. Although this data structure has two entries for each $T_2$ triangle, the cost of this redundancy is amortized, because typically there are far fewer $T_2$ than $T_1$ triangles.

Omitted $O$ entries for cheap $T_1$ and $T_2$ triangles: We do not store $O$ table entries for the “cheap” $T_1$ and $T_2$ triangles that are not adjacent to a $T_0$, because we can access the opposite corners directly from neighboring ring vertices in constant time.

RING-EXPANDER construction of the ring: We propose a simple (linear time and space) greedy approach for computing a ring that, in all tested cases, either produces a Hamiltonian cycle or leaves a small proportion of isolated vertices. Our RING-EXPANDER algorithm tends to minimize the number of $T_0$ and $T_2$ triangles.

Wart skipping: To further reduce storage, we conceptually modify the ring to exclude warts—$T_2$ triangles adjacent to $T_0$ triangles—which allows the expensive $T_0$ triangles adjacent to warts to be represented as cheap $T_1$ triangles (Fig. 1).

Short offsets: When differences $|L[v] - v|$ and $|R[v] - v|$ are sufficiently small, we choose to store $L[v]$ and $R[v]$ as short 2-byte relative offsets. We provide a compact data structure for accommodating exceptions, when the offset is too large.

HYBRID-RING-EXPANDER for increased locality: For large meshes, LR provides the option of either minimizing the number of references or the number of bits stored. For the latter option, we provide a modified RING-EXPANDER that attempts to reduce the average magnitude of offsets.

2 Terminology and Notation

We assume a mesh with $m$ vertices and $n$ triangles. The vertices, which are numbered between 0 and $m-1$, are stored in a geometry table (array of points). The connectivity captures: (1) triangle/vertex incidence, (2) its reverse (star), (3) triangle/triangle and vertex/vertex adjacency (access to neighbors), and (4) ordering of vertices around triangles and of triangles around vertices.

Numerous data structures have been proposed for connectivity so as to support constant-time operators for traversing the mesh from one element (triangle or vertex) to adjacent ones in an orderly manner [Guibas and Stolfi 1985; Brisson 1989; Rossignac 1994].

When conventional data structures are used, the storage cost of the connectivity exceeds the storage cost of the geometry. Indeed, the triangle/vertex incidence alone amounts to $3 \cdot m$ and thus requires twice the storage of geometry, because in a manifold mesh with relatively low genus the number of triangles is roughly $2m$. Popular data structures use several additional references per triangle to encode adjacency, order, and other connectivity relationships.

RING-EXPANDER construction of the ring: We propose a simple (linear time and space) greedy approach for computing a ring that, in all tested cases, either produces a Hamiltonian cycle or leaves a small proportion of isolated vertices. Our RING-EXPANDER algorithm tends to minimize the number of $T_0$ and $T_2$ triangles.

Wart skipping: To further reduce storage, we conceptually modify the ring to exclude warts—$T_2$ triangles adjacent to $T_0$ triangles—which allows the expensive $T_0$ triangles adjacent to warts to be represented as cheap $T_1$ triangles (Fig. 1).

Short offsets: When differences $|L[v] - v|$ and $|R[v] - v|$ are sufficiently small, we choose to store $L[v]$ and $R[v]$ as short 2-byte relative offsets. We provide a compact data structure for accommodating exceptions, when the offset is too large.

HYBRID-RING-EXPANDER for increased locality: For large meshes, LR provides the option of either minimizing the number of references or the number of bits stored. For the latter option, we provide a modified RING-EXPANDER that attempts to reduce the average magnitude of offsets.

2.1 Corner Table

We present our work in terms of corners [Rossignac 2001], which each associate a triangle with a bounding vertex. To facilitate comparison with prior art that manipulates half-edges (also called edge-uses or directed-edges) [Mantyla 1988; Campagna et al. 1998], we observe that each half-edge $h$ corresponds to a unique facing corner $c$ and that the next and opposite half-edge operators, $h.n$ and $h.o$, map to equivalent corner operators $c.n$ and $c.o$.

Fig. 2 shows the standard corner operators. Although corner and vertex references are stored in the LR data structure as integers, we use an object-oriented notation that interprets the operator based on the type of the operand. For example, if $c$ is a corner, $c.n.v.n.c$ means: start with $c$, go to the next corner $c.n$ around the triangle $c.t$, obtain the reference $c.n.v$ to its vertex, go to the next vertex $c.n.v.n$ around the ring, and retrieve a corner $c.n.v.n.n.c$ of that vertex.

3 Prior Art

A customization of the popular Winged-Edge (WE) data structure [Baumgart 1972] to triangles stores connectivity using 3 references for each half-edge $h.c$, to its starting vertex $h.v$, to the opposite half-edge $h.o$, and to the next half-edge $h.n$ around the same triangle, resulting in a total of 9 rpt.

As suggested by Campagna et al. [1998], the Corner Table (CT) data structure [Rossignac 2001] sorts the half-edge entries so that the three entries of a triangle are consecutive and listed in clockwise order (with respect to the outward pointing normal). This makes storing $c.n$ unnecessary, since it can be derived trivially using modular arithmetic. Hence, CT stores only two references per corner $c$: its vertex $c.v$ and its opposite corner $c.o$; see Fig. 2. CT uses two arrays $V$ and $O$ of integers so that $c.v = V[c]$ and $c.o = O[c]$.

WE and CT do not store any vertex-to-triangle references. One may add these by storing, for each vertex $v$, a reference $v.c = C[v]$ to one of its corners in the $C$ table. Since there are three corners (and half-edges) per triangle and about twice as many triangles as vertices, with this addition, WE stores $9.5$ rpt, while CT stores $6.5$ rpt. The mesh connectivity is completely captured in the $V$ array of per-triangle vertex references and may be compressed to less than two bits per triangle [Rossignac 1999; Khodakovskoy et al. 2002]. However, compressed formats produced through sequential encoding cannot be used for random access mesh traversal, and $V$ alone does not provide constant-time access to neighboring elements. Some formats support local decompression [Yoon and Lindstrom 2007; Courbet and Hudelot 2009], but restrict the access pattern to a hierarchical or contiguous traversal and execute a complex decompression code each time a new portion of the mesh is accessed.

Castelli-Aleardi et al. [2006b] prove that a succinct representation of the connectivity of planar triangulations of $n$ triangles can be
obtained by forming clusters of $O(\log n)$ triangles and by using the connectivity of a cluster to index a catalog of pre-computed look-up tables from which results of connectivity operators may be extracted in constant time. They form groups of $O(\log n)$ clusters to reduce the cost of storing inter-cluster connectivity information. Although this theoretical formulation has not been fully implemented, a less succinct version restricted to simple catalogs has been explored [Castelli Aleardi et al. 2006a; Mebarki 2008].

The Star-Vertices of Kallmann and Thalmann [2001] use 3 rpt to store for each vertex $v$ a circularly ordered list of references to adjacent vertices $w$, each augmented with an index $i$ that identifies the position of the reference to $v$ in the list of $w$. Hence they store one such $(w, i)$ pair per half-edge. Because $i$ typically fits in a few bits, $w$ and $i$ can be packed into a single reference. An additional per-vertex reference locates the beginning of each vertex’s list.

Blandford et al. [2005] use a representation similar to the Star-Vertices, but reduce storage by ordering vertices according to a k-d tree, which allows them to take advantage of a variable-length encoding of relative vertex indices. They report storage costs of about 5 bytes per triangle. Their representation supports vertex adjacency efficiently, but does not allow linear indexing of triangles and corners (e.g. triangles exist only as vertex tuples), and hence does not support constant-time evaluation of the standard corner operators.

Snoeyink and Speckmann [1999] orient all edges and partition them into three disjoint vertex-spanning trees so that each vertex (except the vertices of a seed triangle) has exactly one outgoing edge in each tree. For each vertex $v$, they store six references to vertices $w$ so that $(v, w, u)$ is a triangle of the mesh and $(v, u)$ an outgoing edge from $v$. Their Tripod data structure uses 3 rpt.

Gurug et al. [2011] start with the Corner Table, but avoid storing the vertex-to-corner $v.c$ reference by matching each vertex with a corner of one or two incident triangles. They order the triangles so that the reference to the triangle or quad (triangle pair) matched with vertex $v$ may be trivially recovered from the index of $v$. Furthermore, they eliminate the need to store the $V$ table, and recover $c.v$ by walking around the unknown vertex using the $c.s$ swing operator until they reach the triangle or quad matched with $c.v$. Finally, they avoid storing pointers between corners within a quad. For efficiency, instead of storing the partial content of the $O$ table, they store the equivalent swap references in an $S$ table using 4 entries per matched quad or triangle. Their SQuad representation stores slightly more than 2 rpt. In contrast to SQuad, LR stores vertex references from which swing corners are inferred.

4 The LR Representation

In this and the following section, we outline the LR (Laced Ring) approach, describe its representation, and discuss its construction and use. We focus here on a simple representation aimed at minimizing the number of references per triangle. A variation aimed at minimizing the number of bits per triangle is discussed in Section 6.

4.1 Topological Domain

We assume that the triangle mesh is a connected manifold without borders. Meshes with borders can be converted to closed manifolds by adding a dummy vertex $v$ and a fan of dummy triangles around $v$ that are joined with the border edges. We discuss the implementation of borders further in Section 4.6. Non-manifold meshes that represent the boundary of a solid may be converted to pseudo-manifolds while minimizing vertex replication [Rossignac and Cardoze 1999], and as such can be represented compactly using our LR data structure.

Figure 3: Ring-expander traversal. The corners are numbered in the order in which they are visited, starting with the seed $s$.

4.2 The Ring

We first select and orient a manifold loop of mesh edges that visits most—and ideally all—vertices. We call it the ring and its edges the ring edges. The remaining edges are called transversal. Assume that the mesh has $m$ vertices, out of which $m_r$ vertices are on the ring. We want to minimize the number of isolated vertices $m_i = m - m_r$ that are not on the ring.

The perfect solution, i.e., a Hamiltonian cycle of edges, has been studied in graph theory. Unfortunately, previous studies of Hamiltonian cycles for triangle meshes are either focused on the dual graph, where the nodes represent triangles [Arkin et al. 1996; Gopi and Eppstein 2004], or are restricted to specific topologies and regular valence triangulations [Upadhyay 2010].

To construct the ring, we use the following greedy Ring-expander algorithm. We begin by marking each triangle $t$ and vertex $v$ as unvisited by setting the flags $t.m$ and $v.m$ to false, respectively. We then pick a random seed corner $s$, from which we perform an invasion that visits most vertices and about half of the triangles. We ensure that the visited region is edge-connected, has no interior vertices (surrounded by all-visited triangles), and is bounded by a single manifold loop of edges (i.e., the ring). The Ring-expander code, using corner operators, is simple:

```c
4.2 The Ring

We first select and orient a manifold loop of mesh edges that visits most—and ideally all—vertices. We call it the ring and its edges the ring edges. The remaining edges are called transversal. Assume that the mesh has $m$ vertices, out of which $m_r$ vertices are on the ring. We want to minimize the number of isolated vertices $m_i = m - m_r$ that are not on the ring.

The perfect solution, i.e., a Hamiltonian cycle of edges, has been studied in graph theory. Unfortunately, previous studies of Hamiltonian cycles for triangle meshes are either focused on the dual graph, where the nodes represent triangles [Arkin et al. 1996; Gopi and Eppstein 2004], or are restricted to specific topologies and regular valence triangulations [Upadhyay 2010].

To construct the ring, we use the following greedy Ring-expander algorithm. We begin by marking each triangle $t$ and vertex $v$ as unvisited by setting the flags $t.m$ and $v.m$ to false, respectively. We then pick a random seed corner $s$, from which we perform an invasion that visits most vertices and about half of the triangles. We ensure that the visited region is edge-connected, has no interior vertices (surrounded by all-visited triangles), and is bounded by a single manifold loop of edges (i.e., the ring). The Ring-expander code, using corner operators, is simple:

```n
```
In an attempt to minimize the number of isolated vertices $m_i$, we run Ring-expander several times with random seed corners and retain the seed leading to the smallest $m_i$, which usually is negligible with respect to $m$ and sometimes is zero. The first run yields a ratio $m_i/m$ of 0.005% averaged over our test models. Additional runs often reduce this ratio.

### 4.3 Ring-based Classification of Triangles

To simplify exposition, we distinguish several kinds of triangles (see Fig. 4). $T_0$ triangles (bifurcations) have no ring edges; $T_1$ triangles (the most common kind) have exactly one ring edge each; $T_2$ triangles (dead-ends of the “corridors”) have two edges on the ring. $T_1$ and $T_2$ are “expensive,” irregular $T_1$ and $T_2$ triangles that share an edge with at least one $T_0$ triangle. Finally, we call a $T_2$ triangle that is adjacent to a $T_0$ triangle a wait. Such pairs of triangles are denoted $T_0^a$ and $T_2^a$.

A triangle incident upon an isolated vertex must be $T_0$. Clearly a $T_2$ triangle cannot have an isolated vertex, since all of its three vertices are on the ring. Furthermore, a $T_1$ triangle has two consecutive ring vertices, $v$ and $v.n$. If its third vertex $v.w$ was isolated, then our construction algorithm would have included $w$ in the ring between $v$ and $v.n$, turning the triangle into a $T_2$.

### 4.4 Representing Incidence

We identify the ring vertices by integers between 0 and $m.v - 1$ assigned in order of appearance along the ring (starting from an arbitrary vertex). Hence, the references $v.p$ and $v.n$ to the vertices that respectively precede and follow $v$ on the ring may be computed as $v.p = (v + m.v - 1) \mod m.v$ and $v.n = (v + 1) \mod m.v$. Vertices with indices between $m.v$ and $m - 1$ are isolated vertices.

Each edge $e = (v, v.n)$ of the ring is associated with a starting vertex $v$ and with two incident triangles: $v.t_L$ on the “left” and $v.t_R$ on the “right.” We order the triangles so that $v.t_L = 2v$ and $v.t_R = 2v + 1$. Triangle $v.t_L$ has vertices $(v, v.l, v.n)$, where $v.l$ is stored in the $L$ table as $L[v]$. Similarly, triangle $v.t_R$ has vertices $(v.n, v.r, v)$, where $v.r$ is stored in the $R$ table as $R[v]$; see Fig. 5. $T_0$ triangles, which have no ring edges, are not stored in the $LR$ table. Rather, they are represented in the regular Corner Table arrays $V$ and $O$, and are assigned indices $2m.v$ and above.

We call the corners of the $v.t_L$ and $v.t_R$ triangles incident upon a ring edge the ring corners. We label them $v.0$, $v.1$, $v.2$, $v.4$, $v.5$, and $v.6$, as shown in Fig. 5, and assign to corner $v.i$ the integer index $8v + i$. Thus the offset $i$ of a corner $c$ is determined by the three least significant bits of $c$. By shifting the base of this scheme by eight rather than six for each vertex, we are not using corner IDs $8v + 3$ and $8v + 7$. This irregular assignment of indices speeds up some of the corner operators by allowing bit shifts and masks to be used in place of division and modulo. Although corners $8v + 3$ and $8v + 7$ do not exist, no storage is wasted on these unused indices, since we do not allocate space to each corner (except in the VO table; see below). Not using consecutive corner numbers limits the size of the mesh that can be stored, but using 32-bit references to opposite corners eliminates this concern for all practical purposes.

Figure 4: Triangles are classified based on their number of ring edges and whether they are adjacent to a $T_0$ triangle.

Note that there are two possible representations for the corners of a $T_2$ triangle: one for each of its two ring edges. For many traversal operations this is not a problem, but when unique corner references are desired, our convention is to associate the $T_2$ triangle with its second ring edge. We say that the other three corner references are redundant. We can easily detect that a reference is redundant and convert it to the corresponding canonical reference. For example, given a corner $c = v.p.2$ (see Fig. 5, top right), we detect that $c$ is redundant because $v.p.l = v.n$, and compute the canonical corner reference as $v.0$. Mappings of other corners in this and in the symmetric configuration are handled similarly.

We can obtain a reference $v.c$ to a corner of a ring vertex $v$ as $v.c = v.0 + 8v$ and visit the triangles incident on $v$ using the $c.s$ operator. A reference to one corner of each isolated vertex is stored explicitly in an auxiliary array $C$.

If all the vertices were on the ring and if all the triangles were incident upon at least one edge of the ring, this representation would suffice to support all the standard corner operators, and would store only two references per vertex, or 1 rpt (since there are roughly twice as many triangles as vertices).

### 4.5 Representing Adjacency

Triangle adjacency is provided by the opposite corner operator $c.o$. Within a quad formed by triangles $v.t_1$ and $v.t_2$, $v.1$ and $v.5$ are opposites. Hence $v.1.o$ and $v.5.o$ can be determined directly. In a $T_2$ triangle, $v.2.o$ and $v.4.o$ may be obtained by first remapping $v.2$ and $v.4$ to their redundant counterparts $v.p.1$ and $v.p.5$, and then computing their in-quad opposites (see Figs. 5 and 6).

Opposites that do not lie in a $T_0$ can be obtained for $T_1$ and $T_2$ corners $v.0$, $v.2$, $v.4$, and $v.6$ by visiting nearby vertices on the ring. That is, when crossing a transversal edge via $c.o$, one or both of the other edges in the adjacent $T_1$ or $T_2$ triangle must be ring edges. For instance, if $v.n.d = v.l$, then $v.0.d = v.n.2$. Otherwise, $v.p.l = v.n$ and $v.0.o = v.p.0$; see Fig. 6. When $c.o$ lies in a $T_2$ triangle, we must also remap the corner in case it is redundant.

Figure 5: Left: Left and right triangles $v.t_L$ and $v.t_R$ are defined for each ring edge $(v, v.n)$. Their corners are labeled $(v.0, v.1, v.2)$ and $(v.4, v.5, v.6)$. Right: Redundant (top) and canonical (bottom) representation of a $T_2$ triangle.
If \( c.o \) lies in a \( T_0 \), on the other hand, we cannot reach it via the ring, and we store in \( L \) or \( R \) a bit signaling that \( c.t \) is an expensive \( T_1 \) or \( T_2 \) triangle. In this case, rather than storing \( v.l \), we store in \( L[v] \) an index into a condensed corner table \( \text{VO}^* \) (and similarly for \( v.r \)). \( \text{VO}^* \) holds triplets \((v.l, v.0.o, v.2.o)\) and \((v.r, v.4.o, v.6.o)\).

Finally, for corners \( c \) in \( T_0 \) triangles, we consult the \( O \) table, which holds opposites for all three corners of such triangles.

### 4.6 Meshes with Borders

As discussed previously, LR can handle meshes with borders by introducing triangles that join border edges to a single dummy vertex \( v \). If there are several border loops, this addition creates non-manifold edges. We ensure that \( v \) is not a part of the ring by initially marking it as visited, which guarantees that we never invade any of the dummy triangles incident on \( v \). Any reference to \( v \) in the LR table is replaced with a special null index.

### 4.7 Implementation of Operators

We summarize here the implementation of the standard operators.

- \( \text{c.v} \): If \( c \geq 8m_r \), then \( c.t \) is a \( T_0 \) triangle and \( c.v = V[i] \), where \( i = c - \lfloor c/4 \rfloor - 8m_r \). (This subtraction of \( \lfloor c/4 \rfloor \) restores the base to six to avoid unused corners in the \( \text{VO} \) table.) Otherwise, we compute \( v = \lfloor c/8 \rfloor \) and use the relative corner offset \( c \mod 8 \) to select among \( v, v.n, v.e.l \), and \( v.r \) (see Fig. 5).

- \( \text{c.o} \): If \( c \geq 8m_r \), then \( c.t \) is a \( T_0 \) triangle and \( c.o = O[i] \), where \( i = c - \lfloor c/4 \rfloor - 8m_r \). Otherwise, we let \( v = \lfloor c/8 \rfloor \) and distinguish several cases (Fig. 6). In the first case, \( v.1.o = 8v + 5 \) and \( v.5.o = 8v + 1 \). In the next three cases, we infer the opposite from the \( L \) and \( R \) tables and ring vertices. For example, if \( v.l = v.n.l \), then \( v.0.o = 8v.n + 2 \). When \( c \) is in a \( T \) triangle, we look up \( c.o \) using \( v.e.l \) or \( v.r \) as an index into the \( \text{VO}^* \) table. The other cases can be derived by symmetry.

- \( \text{c.v} \): If \( v \geq m_v \), then \( v \) is isolated and \( v.e = C[v - m_v] \). Otherwise, if \( v.l = v.n.n \) (redundant \( T_2 \) triangle), then \( v.e = 8v.n + 1 \) (and similarly for \( v.r \)); otherwise \( v.e = 8v \).

- \( \text{c.t} \): The triangle \( c.t \) of corner \( c \) is defined as \( \lfloor c/4 \rfloor \).

- \( \text{t.c} \): The first corner \( t.c \) of triangle \( t \) is defined as \( 4t \).

- \( \text{c.n} \): The next operator is defined as \( c.n = c - 2 \) if \( c \mod 4 = 2 \); otherwise \( c.n = c + 1 \).

- \( \text{c.p}, \text{c.s}, \text{c.l}, \text{c.r} \) are derived from the operators discussed above.

### 5 Wart Skipping

The number of \( T_0 \) triangles is typically small compared to the number of \( T_1 \) triangles. However, the connectivity information associated with a \( T_0 \) triangle requires more storage, both for itself and for its adjacent triangles. Hence, it is important to reduce the number of \( T_0 \) triangles. To do so, we identify warts: \( T_2 \) triangles that are adjacent to \( T_0 \) triangles. (When more than one \( T_2 \) triangle is adjacent to a \( T_0 \), only one of them is considered a wart.) Because each \( T_2 \) triangle is duplicated, we may reclaim the storage for the redundant copy and use it to represent the \( T_0 \). That is, for a \( T_2 \) triangle \((v.n, v, v.p)\) adjacent to a \( T_0 \) triangle \((v.p, u, v.n)\), we store \( u \) rather than \( v.n \) in \( L[v.p] \) (see Fig. 7). We also store a bit in the entry for the \( T_0 \) to indicate that it has been paired with a wart, and use \( T_0^w \) to denote such triangles. Warts are denoted \( T_2^w \).

To correctly process \( T_0^w \) and \( T_2^w \) triangles, we conceptually modify the ring by skipping over the wart and its tip vertex \( v \) when accessing the \( T_0^w \), which in effect makes \((v.p, v, v.n)\) a ring edge and turns the \( T_0^w \) triangle into a regular \( T_1 \) (Fig. 7). This reclassification of the \( T_0 \) also affects any incident \( T_1 \) or \( T_2 \) triangles, which unless they are adjacent to another \( T_0 \) now become regular (cheap) triangles.

The impact of wart skipping on the corner operators is small: For \( c.v \) and \( c.o \), we let \( v.p.n = v.n \) and \( v.p.p = v.p \) whenever accessing a \( T_0^w \) triangle. Opposites of wart tip corners are also redefined as \( v.0.o = v.p.1 \) and \( v.6.o = v.p.5 \), and conversely for \( T_0^w \) tip corners. Aside from this change, the corner operators for warts stay the same.

To appreciate the benefit of wart skipping, note that we make actual use of the redundant reference for the \( T_0 \) triangle, reduce the 6-reference cost for the \( T_0 \) triangle to a single entry, and reduce the 4-reference cost of all adjacent \( T_1 \) and \( T_2 \) triangles to a single entry, for a gain of as many as 15 references per skipped wart. In practice, because \( T_0 \) and \( T_2 \) triangles often come in pairs, wart skipping usually reduces the number of \( T_0 \) triangles by more than half.

### 6 Storage Efficient LR Representation

The LR representation discussed so far has been optimized to reduce the number of integer references per triangle. Its binary storage efficiency can be improved by carefully considering how these references are encoded. In particular, by changing the traversal of \#RING-EXPANDER to produce a ring with greater locality of reference, we allow short relative indices to be used even for very large meshes, though possibly using a larger \#NUMBER of references. This space-optimized representation is discussed below.
6.1 Relative Indexing

The LR table, as presented above, stores 32-bit integer references to vertices. In practice the index difference, or offset, between a ring vertex \( v \) and its left and right neighbors \( v.l \) and \( v.r \) is often small enough to fit in 16 bits, even when the mesh has far more than \( 2^{16} \) vertices. We exploit this and store the offsets \( v.l - v \) and \( v.r - v \) (modulo the number of ring vertices \( m \)) more compactly than the absolute indices. For large meshes, however, the depth-first traversal of RING-EXPANDER often results in very long triangle strip corridors between bifurcations (Fig. 8, left). In general, more bifurcations, and thus shorter corridors, lead to smaller offsets.

A breadth-first strategy for RING-EXPANDER (Fig. 8, right) generates shorter offsets, but favors bifurcations—i.e. expensive \( T_0 \) triangles—over long corridors—i.e. cheap \( T_1 \) triangles. Hence, we propose a compromise (Fig. 8, center): A hybrid breadth- and depth-first traversal that balances the number of bifurcations and the magnitudes of offsets. It modifies RING-EXPANDER to interrupt the depth-first traversal every \( k \) steps and resets the traversal using breadth-first backtracking. Setting, \( k = 1 \) results in a pure breadth-first traversal, while \( k = \infty \) yields a depth-first traversal. Intermediate values of \( k \) may be used to tune the number of bifurcations and distribution of offsets.

Our HYBRID-RING-EXPANDER algorithm records backtracking corners in a double-ended queue \( d \) that is initially empty:

```c
while (true) {
    c.n.v.m = c.p.v.m = true; // mark vertices as visited
    if (!c.v.m) { // has c.v been visited?
        c.v.m = c.t.m = true; // invoke c.t
        d.push_back(c.l); // push left and right...
        d.push_back(c.r); // ... neighbors onto deque
        n++; // increment triangle count
    }
    if (d.empty()) break;
    if (n \& k == 0) c = d.pop_front(); // breadth-first
    else c = d.pop_back(); // depth-first
}
```

Though slightly more complex than RING-EXPANDER, this hybrid method still achieves a throughput of 25 million triangles/second.

Rings generated with an optimal value of \( k \) tend to have offsets that can be stored as 15-bit signed integers. When this is the case for both of a pair of \( v.t_L \) and \( v.t_R \) triangles, we store the offsets as 16-bit entries in the LR table. LR entries that require more bits are handled using one level of indirection into the VO* table, which is indexed by combining bits from the \( L \) and the \( R \) entries into a 26-bit reference \( a \). VO* stores the corresponding \( v.l \) and \( v.r \) indices in consecutive locations \( VO^*[a] \) and \( VO^*[a+1] \) using 32 bits each. We use \( T^0 \) to identify triangles that require this extra level of specification of \( v.l \) or \( v.r \) using long references. As in standard LR, \( T^0 \) denotes irregular triangles adjacent to a \( T_0 \) that also require long indexing into VO*, for which one vertex and two opposite corners are stored.

One may think of the VO* table as a full corner table, but with references that are already known removed (see Fig. 9). Our implementation discards the unused VO* entries and packs this table into a single linear array of integer references. Because we always arrive at a sequence of entries in this table knowing the type of each triangle in a pair—\( T^2/T^1 \), \( T^1/T^2 \), \( T^1/T^0 \), or \( T^0/T^1 \)—there is no ambiguity what the next 2, 4, or 6 integer entries represent. In particular, the first two references of a tuple always store \( v.l \) and \( v.r \).

6.2 Storage Format

For each LR entry we store two bits, \( L_w \) and \( R_w \), identifying one of four configurations: a pair of \( T_1 \) triangles, a \( T_0^w \) on the left or on the right, or a pair of \( T^1 \) or \( T^2 \) triangles that require long indexing. Note that \( T_0^w \) triangles can appear on the left or right, but not both simultaneously, as the triangle paired with the \( T_0^w \) triangle is adjacent to a \( T_0^i \) triangle and has at least one ring edge. Thus, the two bits stored in the LR table indicate whether to skip warts on the left and on the right, with the unused double-wart combination signaling the need for a long index (see Fig. 9).

As discussed above, when necessary, we combine the LR entries into a 26-bit index \( a \) into the VO* table. With two additional bits out of the 32 already used, the remaining four bits are used to encode left and right wart skips (since a triangle may require a long index into the VO* table and a wart skip) and whether \( v.t_L \) and/or \( v.t_R \) is adjacent to a \( T_0 \), i.e. if it is an irregular \( T^i \).

The VO table stores first a list of all \( T_0 \) triangles as six references per triangle. Any subsequent vertices and opposite corners that cannot be represented directly in the LR table are stored as variable-length records, in no particular order, in the VO* table. The index \( a \) and the combination of \( L_w \) and \( R_w \) bits, which distinguish \( T \) from \( T^1 \) triangles, are sufficient to determine the record type.
7 LR for Efficient Rendering

Although our LR representation was primarily designed to support efficient mesh connectivity queries, a leaner version that does not store corner pointers may find utility in any number of applications that require only a representation of a static indexed triangle list. Three such applications include efficient offline or in-memory storage, transmission, and high-throughput rendering.

At its heart, the LR table encodes a coherent sequence of vertices and triangles. As such, it makes for a lean indexed mesh representation with exactly one vertex reference per triangle; the ring vertices are consecutive and need not be specified. (The few T0 triangles may be represented verbatim.) This simple representation suggests the possibility of using LR as an efficient rendering primitive.

The LR representation directly rivals the common triangle strip. Each branch in one of the interlocking LR triangle trees can be thought of as a generalized triangle strip. Whereas a typical mesh can be represented as a collection of (non-generalized) triangle strips using about 1.35 references per triangle, LR requires only one reference per triangle, and avoids the overhead associated with strip "swaps" or "restarts." In addition to being more space efficient, our LR mesh is easier and faster to construct than triangle strips, which require greedy or more complex construction procedures. As an example, our RING-EXPANDER method is two orders of magnitude faster than Evans et al.’s [1996] Stripe software.

Efficient rendering is possible by “decompressing” the LR representation on the fly, e.g., using a geometry shader, into indexed triangles that reference a vertex buffer object (VBO). This allows storing a compressed index list on the GPU that omits the implicit consecutive ring vertex indices. We may maximize locality of reference for the GPU post-transform-and-lighting vertex cache by using a breadth-first LR construction. As in the standard LR representation, wart skips can be encoded using a dedicated bit that is set only for T0 triangles. Duplicates of T0 triangles and non-existent triangles across mesh borders can be eliminated by specifying either v or v:n as the tip vertex, which both result in degenerate triangles.

8 Results

We report in Table 1 statistics for several benchmark models. Let the mesh have m vertices, n triangles, and a ring with m vertices, leaving m = m - m0 vertices isolated. Let n1 be the number of T0 triangles remaining after wart skipping, and let n1 be the number of T1 and T2 triangles. In the standard LR representation, we store a total of 2m0 references in the LR table, m0 references in the VO table, 3m1 references in the VO* table, and m1 references in the C table. Hence, the storage cost for the standard LR representation is 2m0 + 6m0 + 3m1 + m1, which is 1.49 rpt, where f is about (6m0 + 3m1 - m0) / (2m0), assuming n ≈ 2m. For the bit-efficient LR, we store 1-bit references in the L and R tables, while the other tables hold 32-bit references. Hence, the total storage cost in bits is 32(m0 + 6m0 + 3m1 + n1 + m1), where n1 denotes the number of T1 and T2 triangles.

Table 1 reports for the standard LR representation the number of triangles n and isolated vertices m0, the percentage of valence-6 vertices and T0, T0, and T2 triangles, as well as the resulting rpt. The median rpt for LR is 1.08 rpt, which is about half the storage cost for SQuad. As in SQuad, the storage cost is influenced by the regularity of the mesh, and increases with fewer valence-6 vertices.

For the bit-efficient representation, we also report the period k between breadth-first restarts, the percentage of T2 triangles, and the resulting number of bits per triangle (bpt). The median is 26.2 bpt.

We ran the construction several times and chose the k that resulted in the lowest storage cost. Given that we can construct LR meshes from corner tables at a rate of two million triangles per second, we can afford to explore several k values and seed corners per mesh.

Fig. 10 shows the access time for processing the 55 million triangle David mesh using CT [Rossignac 2001], SQuad [Gurung et al. 2011], and the standard LR using various memory configurations. (Our unoptimized implementation of the bit-efficient LR is on average five times slower than standard LR.) Note that the operating system reserves at least 400 MB for system purposes and, hence, we are left with the remaining memory. Using sequential loops over corners and vertices, we report timings for c.v, c.o (for LR and CT) or c.s (for SQuad), and for operations that compute the valence and normal of a vertex. As in [Gurung et al. 2011], we also report the cost per triangle traversed when following the "contour" where the mesh intersects a plane, which involves a non-sequential, data-dependent traversal.

Although the sequential c.v and c.o table lookups in CT are faster than the computations needed by LR when the mesh fits in main memory, this performance difference is not observed when considering higher-level tasks that require some level of random access, e.g., to access neighboring vertices. In this case LR is generally faster than both CT and SQuad due to its smaller memory footprint and improved cache utilization. Furthermore, because the storage needed by LR for most meshes is about half the storage needed by SQuad, and about 1/6 the storage of CT, LR is significantly faster than CT and SQuad when processing large models that do not fit in main memory, because the reduced storage leads to fewer page faults.

Finally, we evaluated the number of references per triangle needed for a renderable indexed mesh representation of LR (i.e. with corner references removed). Using a depth-first traversal with wart skips, the mean storage required is 1.03 rpt, which increases to 1.10 rpt when wart skips are disallowed. This compares favorably with Stripe, which requires 1.35 rpt on average.
8.1 Limitations

Because LR stores vertex references in a contiguous memory array, and because inserting an array entry would require updating many of the indices in this array, incremental connectivity changes cannot be performed efficiently. We recommend LR for use in applications where mesh connectivity remains fixed, or where constructing a new LR with the desired connectivity is acceptable.

Our construction algorithm assumes as input a mesh data structure such as the corner table (CT) that provides constant-time adjacency queries. A CT can be constructed from a set of indexed triangles in linear time, but the memory overhead of keeping both the CT and LR in memory at the same time may be unacceptable.

9 Conclusion

We have described a simple and efficient implementation of the LR data structure for representing the connectivity of manifold triangle meshes. It supports a comprehensive set of constant-time, random-access operators for traversing the mesh and offers roughly the same performance as the best previously reported solution, SQuad. Yet LR requires only about half of the storage needed by SQuad, namely about 1.08 references per triangle, or, with the bit-efficient variation, only about 26.2 bits per triangle. Hence, LR requires about 6 times less storage than the corner table and 9 times less than the winged-edge representation.

References


