Computer-Aided Design 45 (2013) 262–269

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

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

Zipper: A compact connectivity data structure for triangle meshes

Topraj Gurung [a,*], Mark Luffel [a], Peter Lindstrom [b], Jarek Rossignac [a]

[a] Georgia Institute of Technology, United States

[b] Lawrence Livermore National Laboratory, United States

ARTICLE INFO

Keywords:
Triangle meshes
Mesh connectivity
Hamiltonian cycle
Differential coding

ABSTRACT

We propose Zipper, a compact representation of incidence and adjacency for manifold triangle meshes with fixed connectivity. Zipper uses on average only 6 bits per triangle, can be constructed in linear space and time, and supports all standard random-access and mesh traversal operators in constant time. Similarly to the previously proposed LR (Laced Ring) approach, the Zipper construction reorders vertices and triangles along a nearly Hamiltonian cycle called the ring. The 4.4× storage reduction of Zipper over LR results from three contributions. (1) For most triangles, Zipper stores a 2-bit delta (plus three additional bits) rather than a full 32-bit reference. (2) Zipper modifies the ring to reduce the number of exceptional triangles. (3) Zipper encodes the remaining exceptional triangles using 2.5× less storage. In spite of these large savings in storage, we show that Zipper offers comparable performance to LR and other data structures in mesh processing applications. Zipper may also serve as a compact indexed format for rendering meshes, and hence is valuable even in applications that do not require adjacency information.

1. Introduction

Zipper, introduced here, is a compact representation for manifold triangle meshes with fixed connectivity. It is a randomly-accessible-and-traversable (RAT) mesh representation: it provides **constant-time** retrieval of any triangle, vertex, or corner (equivalently half-edge) in the mesh, and also supports constant-time access to the consecutive vertices around a triangle and to the consecutive incident triangles around a vertex.

Popular polygon representations such as the Half-Edge representation use 48 bpt (bits per triangle) to represent the geometry (using three 32-bit floats per coordinate) and 528 bpt to represent the connectivity [1], as discussed in Section 2.2. The most compact existing triangle RAT, BELR (bit-efficient LR) [2], uses 26 bpt on average for connectivity. It reorders most vertices and triangles along a ring (a nearly Hamiltonian cycle) and stores one reference for each normal triangle and up to 15 references for each exceptional triangle.

Zipper uses a similar reordering, but represents most vertex indices in normal triangles using differential coding, which reduces storage for most triangles to only a 2-bit delta and 3 additional bits, and uses 2.5× less storage for exceptional triangles. It stores connectivity using only 6 bpt (on average over tested models), and hence provides a 4.4× improvement over BELR.

This storage decrease is remarkable, because it improves by 4.4× on the best results achieved over the past 40 years by experts across many research areas. By comparison, these earlier efforts have seen a decrease in storage of 17× from 432 bpt [3] to 26 bpt [2]. Zipper uses only an equivalent of a fifth of a 32-bit reference per triangle to store enough information to access (in constant time) not only the three vertices of each triangle, but also its three neighboring triangles, as well as one incident triangle for each vertex.

Our Zipper implementation of the standard mesh traversal operators not only has constant-time complexity, but is very fast (7–40 ns per operator). When executed on a small mesh, the basic operators for mesh access and traversal in Zipper are faster than those in BELR, but about 1.8×–3.6× slower than their counterparts in the optimized standard LR. However, the impact of this overhead on the performance of an application is often negligible when compared to other processing costs, and is more than compensated by performance gains resulting from fewer page faults when processing large meshes that do not fit in memory.

Zipper is significantly more compact than the commonly used triangle strip/fan or indexed formats. Hence, it is also a good candidate for applications (such as rendering) that do not require adjacency information.

2. Prior art

2.1. Corner operators

Many mesh processing algorithms may be formulated using operators that access the next vertex around a triangle or the next

* Corresponding author.
E-mail address: topraj@gmail.com (T. Gurung).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2012.10.009

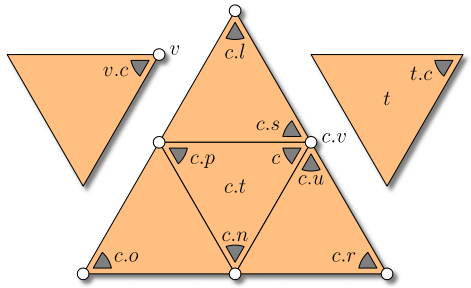

Fig. 1. The standard corner operators associate with corner c its vertex $c.v$, triangle $c.t$, and next $c.n$ and opposite $c.o$ corners. The other corner operators, previous $c.p$, swing $c.s$, unswing $c.u$, left $c.l$, and right $c.r$ are derived from $c.n$ and $c.o$. In addition, $v.c$ returns one corner of vertex v, and $t.c$ returns one corner of triangle t.

triangle around a vertex. In this paper we use **triangle corners** [4], which uniquely identify a triangle and one incident vertex, to mark a specific spot on the connectivity graph of the mesh and to serve as a mesh traversal primitive. Hence, our mesh traversal operators manipulate corners, their vertices and their triangles. Note that an equivalent set of operators may be defined in terms of half-edges (also called edge-uses or darts) [1,5], because one may trivially establish a mapping between corners and half-edges.

RAT mesh representations support the standard set of corner operators [6], defined below. The operators provide a simple mechanism to access all vertices, triangles, and corners in constant time from their IDs or from adjacent elements.

Given a corner c, the **standard corner operators** (see Fig. 1) are: the triangle $c.t$ of c, the vertex $c.v$ of c, the next corner $c.n$ in $c.t$, the opposite corner $c.o$ defined such that $c.n.v = c.o.n.n.v$ and $c.n.n.v = c.o.n.v$, a corner $t.c$ of triangle t, and a corner $v.c$ of vertex v. From these, we define a set of **derived corner operators**: the previous corner $c.p = c.n.n$ in $c.t$, the left $c.l = c.n.o$ and right $c.r = c.p.o$ neighboring corners of c, and the swing $c.s = c.l.n$ and unswing $c.u = c.r.p$ corners used to walk around $c.v$. In this paper we focus on efficiently encoding $c.v$ and $c.o$—the remaining operators can be inferred trivially (see [2]).

Early boundary graph representations of connectivity [1] use 32-bit memory pointers. Some of the more recent representations (for example [4]) assign consecutive positive integers to vertices, triangles, and corners, and use arrays to store sorted lists of references (32-bit integer indices), rather than pointers, which, for example, identify a triangle incident upon a vertex, the three vertices of a triangle, or the three opposite corners in adjacent triangles.

To simplify our algorithms and representation, as others have done in the past, we assume the mesh is manifold. Furthermore, our storage and performance statistics are only representative of meshes where the genus and the number of border edges are small relative to the number m of vertices. With these assumptions, there are roughly $n = 2m$ triangles and $3m$ edges. So, if a representation uses a total of nr references, we say that its storage cost is $32r$ bpt (bits per triangle) or r rpt (references per triangle).

To reduce connectivity storage, some representations reorder vertices, corners, or triangles. For example, ECT (the Extended Corner Table) [4,6] encodes connectivity in 6.5 rpt, by assigning the three corners of a triangle consecutive numbers that are consistent with the orientation of the triangle. This assumption makes it unnecessary to store several of these look-up tables, because their content may be computed in constant time when needed: $c.t = \lfloor c/3 \rfloor$, $c.n = 3c.t + (c + 1 \mod 3)$, $t.c = 3t$. Even with this improvement, connectivity accounts for up to 90% of the total storage when using 16-bit quantized coordinates to represent geometry.

2.2. General representations

Early representations use much storage, because they cater to more general (polygonal or higher-dimensional) meshes.

Brisson's cell-tuple structure [7] generalizes the quad-edge data structure of Guibas and Stolfi [8], which was restricted to 2-manifolds without boundaries, and the facet-edge data structure of Dobkin and Laszlo [9], which catered to subdivisions of 3-manifolds. It is restricted to subdivided manifolds with or without boundary. When applied to triangle meshes, the cell-tuple structure associates each triangle t with 6 groupings (n-tuples), each one corresponding to a choice of three entities (v, e, t): the triangle t, an edge e of t, and a vertex v of e. There are 6 groupings for each triangle because one has 3 choices for e and then 2 choices for v. With each grouping $g = (v, e, t)$, one stores a reference to triangle t and to vertex v, plus three references to adjacent groupings: $s_0(g)$ returns grouping (v', e, t), where v' is the other vertex of e; $s_1(g)$ returns grouping (v, e', t), where e' is the other edge of t that is incident upon v; and $s_2(g)$ returns grouping (v, e, t'), where t' is the other triangle incident upon e. To support the standard operators, one also stores, for each triangle and for each vertex, a reference to one of its groupings. Hence, the total storage cost for connectivity is 31.5 rpt: 6 tuples per triangle that store 5 references each (vertex, triangle, and 3 swaps), plus a tuple reference for each vertex and triangle.

In the cell-tuple structure, groupings g, $s_0(g)$, $s_2(g)$, and $s_0(s_2(g))$ refer to the same edge. The popular Winged-Edge representation [3] combines them into a single edge, with which it associates references to its two bounding vertices, to its two incident triangles, and to the previous and the next edge in each triangle. To be compatible with the RAT operators supported by other schemes, we also assume that it stores a reference to a winged-edge for each triangle and each vertex, so as to support the $v.c$, $t.c$, and $c.t$ operators in constant time. Adding these references pushes the extended winged-edge storage cost to 13.5 rpt.

The Half-Edge representation [1] associates with each half-edge a reference to the next, the previous and the opposite half-edge, together with a reference to a bounding vertex and incident face for a storage cost of 5 references per half-edge, or 15 rpt. Adding support for $t.c$ and $v.c$ for their half-edge counterpart yields a total cost of 16.5 rpt. The Surface-Mesh representation [10] uses half-edges, but reorders them so that opposite ones are consecutive, which eliminates one reference per half-edge. Surface-Mesh also does not store a reference to the previous half-edge in a triangle. Hence, its resulting storage cost is 10.5 rpt.

Star-vertices [11] stores for each vertex a radially sorted list of references to neighboring vertices. It also stores the reference to where that list starts. To make it compliant with our definition of RAT, we must add a reference from each triangle to one of its vertices or half-edges (equivalent to $t.c$) and a reference from each half-edge to its incident triangle. Hence the total cost includes an average of 6 references to neighboring vertices from each vertex (two per edge or equivalently 3 rpt), one reference per half-edge to an incident triangle (that amounts to 3 rpt), a reference per vertex to the start of the list (1 per vertex, or equivalently 0.5 rpt), and 1 rpt for $t.c$. Hence, the total storage for a RAT compatible star-vertices mesh is 7.5 rpt.

2.3. Representations for triangle meshes

Representations restricted to triangle meshes exploit the regularity of the connectivity (3 vertices per triangle and 3 neighbors per interior triangle) and reorder triangles, edges, and/or vertices.

The Directed Edge representation [12] is identical to the Corner Table [4], when considering a bijection between half-edges and