Memory Efficient Adjacent Triangle Connectivity of a Face Using Triangle Strips

Hidekuni Annaka
annaka@src.ricoh.co.jp

Tsukasa Matsuoka
matsuoka@src.ricoh.co.jp

Akihiro Miyazawa
miyazawa@src.ricoh.co.jp

Multimedia Lab, Software R&D Group
Ricoh Company, Ltd.

ABSTRACT

We often need to refer to adjacent elements (e.g., vertices, edges and faces) in triangle meshes for rendering, mesh simplification and other processes. It is, however, sometimes impossible to prepare the enormous memory needed to represent element connectivity in gigantic triangle meshes. We proposed a memory efficient scheme for referring to adjacent faces around a vertex in non-manifold triangle meshes [AM04]. But the scheme has a redundancy in case of two-manifold triangle meshes. This paper proposes new schemes for referring to adjacent faces around a face in two-manifold triangle meshes. First, as our previous scheme, we introduce the constraints to allow random access to a triangle in a sequence of triangle strips. Then, for each face, we construct a list of references to adjacent strips as a representation of triangle connectivity. Experimental results show that, compared with conventional indexed triangle set based methods, our schemes reduce total strage for a triangle mesh and adjacent triangle connectivity by less than 50%.

Keywords: strip, mesh, memory efficient,adjacent connectivity

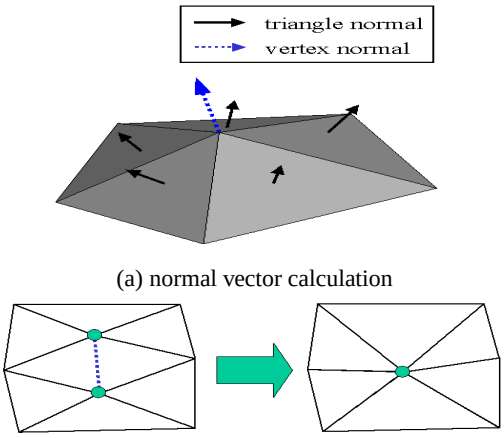
1 INTRODUCTION

One of the most popular representations for 3D models is a triangle mesh. It has often been used not only for rendering but also for various processes in many scenes. In such scenes, adjacent triangle connectivity is frequently used: to calculate vertex normals, to simplify meshes to reduce storage, and to detect collisions for virtual reality systems.

The rendering of triangle meshes requires not only vertex coordinates but also vertex normals. To reduce memory, without storing vertex normals, adjacent triangles around a vertex must be acquired to successively calculate its normal from three triangle vertices (figure 1(a)). Adjacent connectivity of vertices and triangle faces is also required to simplify triangle meshes in the popularly used *vertex contract* operation [Gar99], which unifies two vertices of triangle meshes. When faced with gigantic triangle meshes, however, it is impossible to prepare the enormous memory needed to represent adjacent connectivity. Those gigantic meshes are generated from the latest 3D shape input devices, for example laser range scanners, and we must process triangles of those models numbering in the hundreds of millions.

Therefore, we devised a method based on a triangle strip reference to represent adjacent faces of a vertex [AM04]. Required total storage for a triangle mesh and adjacent triangle connectivity is half that of conventional representation based on an **ITS** (indexed triangle set mesh).

WSCG 2005 SHORT papers proceedings
ISBN 80-903100-9-5, January 31-February 4, 2005
Plzen, Czech Republic.



(a) normal vector calculation

(b) mesh simplification

Figure 1: Usage of adjacent relations

But most of triangle meshes are 2-d manifolds where adjacent connectivity can be represented using adjacent faces of a face. This is why, in this paper we propose a more memory efficient scheme for referring to adjacent faces of a face. In this new scheme, required total storage is about 75% of our previous scheme. Our representation is characterized by the following:

- **CTS (Constrained Triangle Strips)**
  In accessing arbitrary triangles of multiple strips in a strip buffer, we use constrained index triangle strips. We insert one or two delimiters between strips so that the strip size is always an even number. Three vertices orders of triangles at even indices in the strip are counterclockwise and three vertices orders of triangles at odd indices are clockwise in the strip.

- **reference to adjacent faces of a face using CTS**
  We use strip indices to suppress storage size for adjacent face connectivity of a face. A triangle has three adjacent triangles, and three triangle indices are required to store adjacent face references. In triangle strip representation, two adjacent triangles are typically in the same strip and one adjacent face is in a different strip (figure 4). Therefore, one strip index is usually required to store adjacent face references. This configuration is three times more efficient than ITS based representation [Gar99].

Here are some details on how we suppress storage size: We use 63.6%–63.7% the storage of ITS based representation by not using an indexed triangle set. Instead, we use a constrained index triangle strip for triangle connectivity. Further, we use 46.6%–46.7% the storage of ITS based representation by using strip indices for adjacent faces connectivity of a face rather than triangle indices.

The triangle strip is supported by OpenGL APIs and provides fast rendering on popular hardware accelerators. Our method is especially efficient both for adjacent triangle connectivity of vertices and for rendering meshes.

The remainder of this paper is organized as follows: Section 1.1 summarizes related work. Section 2 describes indexed triangle sets and indexed triangle strips for triangle mesh representations. Section 3 introduces our constraints against indexed triangle strips to suppress adjacent triangle connectivity of a vertex. Section 4 describe adjacent faces connectivity of a face by using constrained index triangle strips. Section 5 presents the results of experiments using the proposed method and Section 6 discusses conclusions.

1.1 Related work

Some proposed data structures for adjacent connectivity between elements of 3D models can represent various genus models. The winged-edge [Bau72] is a representative structure that can represent two-manifold models. 3D models often consist of non-manifold surfaces in actual scenes. The radial-edge [Wei88] is a representative structure that can represent non-manifold models.

Those structures are popular representations. The following triangle mesh representation [NDW93] can represent non-manifold models with no explicit adjacent connectivity between elements but vertex connectivity of each triangle.

- triangle set
- triangle strip
- triangle fan

The above representations are suitable for sequential accessing and the rendering of triangles because vertex coordinates are stored directly in the coordinate table. This being the case, common vertex coordinates are stored separately in the coordinate table, so the table cannot be used for referencing neighborhoods.

The following representations are derived from the above representations where each triangle consists of three indices to a vertex coordinate table to share coordinates with other triangles.

- indexed triangle set
- indexed triangle strip
- indexed triangle fan

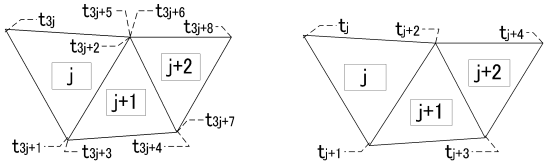
Because of the absence of explicit adjacent connectivity, it takes a long time to refer to adjacent elements of these triangle meshes, for example, to adjacent vertices of a vertex and adjacent triangles of a vertex. It is faster to refer to adjacent elements with explicit adjacent connectivity than without. Hoppe uses adjacent triangles of a triangle in the Progressive Mesh [Hop97] techniques for two-manifold meshes. Garland et al. use adjacent triangles of a vertex in the triangle mesh simplification [Gar99] techniques for non-manifold meshes. In the Hoppe techniques, adjacent connectivity consists of adjacent triangles around a triangle and is equal to the size of triangle connectivity. With Garland techniques, adjacent connectivity is equal to the size of the triangle connectivity. We proposed a scheme for referring to adjacent triangles around a vertex in non-manifold triangle meshes [AM04]. But the scheme may have a storage redundancy in case of two-manifold meshes. Because we can refer to all faces referring adjacent faces around a face in case of two-manifold mesh. This paper proposes a memory efficient method to acquire adjacent triangles around a face for two-manifold meshes.

2 TRIANGLE MESH REPRESENTATIONS

A triangle mesh M is represented widely as a tuple consisting of vertex coordinates table V and vertex indices table T. Coordinates table $V = (v_0, v_1, v_2, ...v_m)$ lists vertex coordinates, each element of which corresponds to a vertex coordinate $v_i = [x_i, y_i, z_i]$. Indices table $T = (t_0, t_1, t_2, ...t_n)$ lists vertex indices that compose triangles. Each element of T corresponds to an index t_j to V. We classify meshes into two representations according to interpretation of T: an indexed triangle set representation and an indexed triangle strip representation.

2.1 Indexed triangle sets

Figure 2(a) shows an indexed triangle set representation where a triplet of consecutive indices, such as $[t_{3j}, t_{3j+1}, t_{3j+2}]$, $[t_{3j+3}, t_{3j+4}, t_{3j+5}]$, and $[t_{3j+6}, t_{3j+7}, t_{3j+8}]$, denotes three vertex indices of a triangle. For each triplet $[t_{3j}, t_{3j+1}, t_{3j+2}]$, we refer to j as a *triangle index*, which is one third of the subscript of the first vertex index. The orientations of a triangle are decided by the order of the three indices that compose the triangle. Generally speaking, the side on which the order is counterclockwise is the front side. It is easy to refer to an arbitrary triangle by triangle index j in the case of an indexed triangle set, because three vertex indices $[t_{3j}, t_{3j+1}, t_{3j+2}]$ are uniquely derived from the triangle index j.



(a)triangle set (b)triangle strip

Figure 2: Triangle mesh representations

2.2 Indexed triangle strips

Figure 2(b) shows an indexed triangle strip representation where a triplet of consecutive indices, such as $[t_j, t_{j+1}, t_{j+2}]$, $[t_{j+2}, t_{j+1}, t_{j+3}]$, and $[t_{j+2}, t_{j+3}, t_{j+4}]$, denotes three vertex indices of a triangle. Note two indices in each triplet are also used in the previous and next triplets. For each triplet $[t_j, t_{j+1}, t_{j+2}]$, we refer to j as a *triangle index*, which is the subscript of the first vertex index and starts from 0. The orientations of a triangle are decided by the order of three indices [NDW93] in this representation also. Unlike the indexed triangle set, however, the order flips alternately depending on whether the triangle index is even or odd.

The indices table of an indexed triangle strip mesh is about half to one third the size that of an indexed triangle set mesh. This is because the indices are shared in an indexed triangle strip representation. In addition, we can render indexed triangle strip meshes faster than indexed triangle set meshes because the number of vertices processed by the rendering libraries (e.g., OpenGL) is smaller.

Indices table T usually represents a sequence of indexed triangle strips. It is difficult, however, to refer to an arbitrary triangle by triangle index j in the case of a sequence of indexed triangle strips. Specifically, if the indices table stores multiple strips to represent a model, we can't determine the orientation of a triangle by the triangle index. This is because the parity (being odd or even) of triangle index j, which holds true in one strip by definition, doesn't match the parity defined by the number counted from the beginning of T, which consists of a sequence of multiple strips of different lengths, where each strip ends with a delimiter -1.

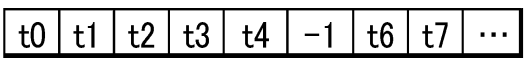
3 REVIEW OF CONSTRAINED TRIANGLE STRIPS

This section reviews a method that enables random access to a triangle in a sequence of triangle strips [AM04]. Access is accomplished by imposing a constraint on the triangle strips described in the previous section.

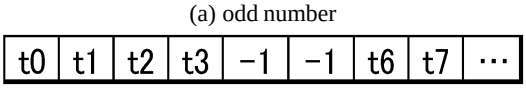
In this method, we insert delimiters depending on the size of strip indices as follows:

- If the size of the strip is an odd number, we insert one delimiter (Figure 3(a)).
- If the size of the strip is an even number, we insert two successive delimiters (Figure 3(b)).

As a result, the three vertices compose a triangle in counterclockwise order if the triangle index is even, or in clockwise order if the index is odd. This enables us to find the orientation of any triangle in a sequence of strips.



(a) odd number



(b) even number

Figure 3: Constrained triangle strips

4 ADJACENT STRIPS OF A FACE

This section describes a new scheme for referring to adjacent triangles around a face in two-manifold triangle meshes, by using constrained index triangle strips.

In this section, let T be an index table that represents the constrained triangle strips. We define a data structure that represents *adjacent strips of faces* $A = (a_0, a_1, \ldots, a_n)$ whose size is the same as that of T. Each element of A is an index to T. As shown in Figure 4, for a given triangle j, an element a_{j+1} of A generally represents a triangle that is adjacent to j and belongs to a strip different from j's. If j is a starting or ending triangle of a strip, a_j or a_{j+2} represents another adjacent triangle of j.

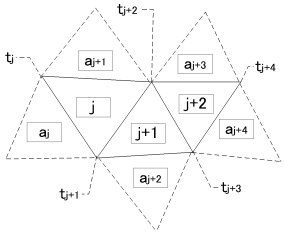


Figure 4: Adjacent strips of faces

For a given triangle j, which is composed of three vertices $[t_j, t_{j+1}, t_{j+2}]$, we can refer to its three adjacent triangles as follows:

1. If $t_{j-1} < 0$, the first adjacent triangle is a_j, which is composed of $[t_{a_j}, t_{a_j+1}, t_{a_j+2}]$. Otherwise, the first one is $j-1$, i.e., $[t_{j-1}, t_j, t_{j+1}]$.
2. The second adjacent triangle is always a_{i+1}, which is composed of $[t_{a_{j+1}}, t_{a_{j+1}+1}, t_{a_{j+1}+2}]$.
3. If $t_{j+3} < 0$, the third adjacent triangle is a_{j+2}, which is composed of $[t_{a_{j+2}}, t_{a_{j+2}+1}, t_{a_{j+2}+2}]$. Otherwise, the third one is $j+1$, i.e., $[t_{j+1}, t_{j+2}, t_{j+3}]$.

Figure 5 shows an example of referring adjacent strips of faces in Figure 4.

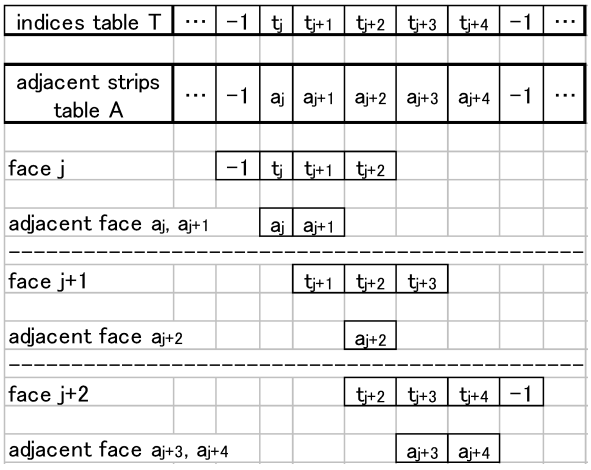


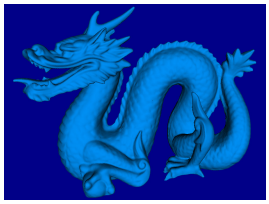
Figure 5: Example of referring adjacent strips of faces

5 RESULTS

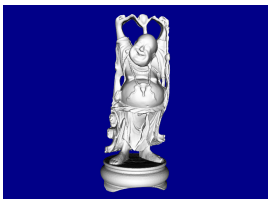
We implemented the schemes described in the previous sections and compared performance results with those of ITS schemes on a personal computer with a Pentium 4, 1.8 GHz CPU, 512MB of main memory and an NVIDIA GeForce2 MX 400 32MB VGA card. Performance was measured for the following items:

- total storage size ($|T+J|$ or $|T+A|$) : the total storage size of the vertex indices table ($|T|$) and the adjacent triangle connectivity of vertices ($|J|$) or triangles ($|A|$)[MB]
- access time (t_a) : the time to refer to adjacent triangles around all i) vertices or ii) triangles[ms]. The number of references is the same in both cases i) and ii). This is because the number in the case i) is eventually equal to the sum of the number of vertices (three) of each triangle and the number in the case ii) is equal to the sum of the number of edges (three) of each triangle.
- rendering time (t_r) : the time to render triangle meshes [ms]

We used the following triangle meshes, (a) Dragon and (b) Happy Buddha [1], in experiments and used the stripification algorithm [ESV96] for the meshes.



(a) Dragon
vertices : 437645
triangles : 871414



(b) Happy Buddha
vertices : 543652
triangles : 1087717

Figure 6: Experimental models

Table 1 lists experimental results of the following four methods.

- M_{tv} : ITS based method for referring to adjacent triangle around a vertex [Gar99].
- M_{sv} : Our previous method [AM04].
- M_{tt} : ITS based method for referring to adjacent triangle around a face [Hop97].
- M_{st} : Our proposed method.

The results show that M_{st} decreases total storage size by 46.6%–46.7% compared to M_{tt}, and by 75.0% – 75.1% compared to M_{sv}.

Regarding the access speed, the results show that M_{st} increases access time (t_a) by 113%–116%, compared to M_{tt}, and decreases by 82.7% – 82.9% compared to M_{sv}.

Considering the rendering time as well, the results show that M_{st} decreases total time ($t_a + t_r$) by 77.7%–78.0%, compared to M_{tt}, and by 87.6% – 87.7% compared to M_{sv}.

mesh	method	$\|T\|$	$\|A\|$ or $\|J\|$	$\|T+A\|$ or $\|T+J\|$	t_a	t_r	t_a+t_r
(a)	M_{tv}	19.9	26.6	46.6	210	345	555
	M_{sv}	9.3	15.5	24.8	411	161	572
	M_{tt}	19.9	19.9	39.9	300	345	645
	M_{st}	9.3	9.3	18.6	340	161	501
(b)	M_{tv}	24.9	33.2	58.1	260	441	701
	M_{sv}	11.6	19.3	30.9	520	202	722
	M_{tt}	24.9	24.9	49.8	371	441	812
	M_{st}	11.6	11.6	23.2	431	202	633

Table 1: Experimental results of referring to adjacent triangles around faces or vertices

6 CONCLUSION

Our proposed method achieves positive results for storage size and total time compared to the ITS based methods and our previous method. Regarding the access time, our proposed method is better than our previous method, although our proposed method is at a disadvantage compared to the ITS based methods. Our proposed method, however, decreases the total time with respect to the ITS based methods, because it exploits OpenGL triangle strips

Future work will focus on reducing access time.

References

[AM04]Hidekuni Annaka and Tsukasa Matsuoka. Memory efficient adjacent triangle connectivity of a vertex using triangle strips. In *Computer Graphics International 2004 (CGI2004) Conference Proceedings*, June 2004.

[Bau72]Bruce Guenther Baumgart. Winged edge polyhedron representation. Technical report, Stanford Artificial Intelligence Laboratory, October 1972. CS-320.

[ESV96]Francine Evans, Steven Skiena, and Amitabh Varshney. Optimizing triangle strips for fast rendering. In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 319–326, 1996.

[Gar99]Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, School of Computer Science Carnegie Mellon University, 1999. CMU-CS-99-105.

[Hop97]Hugues Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH'97 Conference Proceedings*, pages 189–198, August 1997.

[NDW93]Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, August 1993.

[Wei88]Kevin Weiler. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In *Geometric Modelling for CAD Applications*, pages 3–36, North Holland, 1988.

[1] Models are courtesy of Stanford 3D Scanning Repository.