PAPER

A Rendering-Efficient Progressive Transmission of 3D Meshes

Byung-Uck KIM[†a)], ***Student Member***, **Woo-Chan PARK**[††], **Sung-Bong YANG**[†], ***and*** **Francis NEELAMKAVIL**[†††], ***Nonmembers***

SUMMARY We present a novel mesh representation scheme exploiting the characteristics of triangle strips, called *progressives strips*, which gives rendering-efficient triangulation data at the rendering stage in a graphics system such as progressive transmission where the mesh topology changes continuously. Progressive strips consist of a set of triangle strips simplified to the base mesh and a set of refinement steps required to recover incrementally the original mesh at full resolution. We also propose an improved triangle strip filtering algorithm and the encoding of strip-edge collapses in order to reduce efficiently the redundant triangles and the amount of refinement information, both of which may increase as the mesh degrades. Our approach increases the overall graphics performance by reducing the amount of data sent to the graphics pipeline.

key words: *3D graphics, progressive transmission, triangle strips, triangle mesh*

1. Introduction

With increasing demand for on-line access of 3D models, the search for efficient methods for transmission of meshes and their interactive visualization have become important issues in computer graphics. Progressive transmission is a very useful framework when the network bandwidth between the server and the client is inadequate or the rendering capacity at the client site is not sufficient for displaying the model at full resolution. In progressive transmission, a simplified version of the mesh, called the *base mesh*, is sent first, and then the details are added incrementally through a series of refinement steps. Progressive mesh is a popular mesh representation scheme for progressive transmission of triangle meshes [7].

For interactive visualization, not only the speed at which a triangle mesh can be transmitted from the server to the client but also the speed at which it can be displayed at the client site is important [10]. The speed of the rendering process on a triangle mesh is bounded by the rate at which the triangulation data is sent to the graphics pipeline [4]. A common way to reduce such data is to represent a mesh as a set of triangle strips where each of subsequent triangles except the first triangle in a strip reuses two vertices from the previous triangle. In this way, triangle strips can reduce the amount of data sent to the graphics pipeline by a factor of two or three, compared with sending triangles individually [18]. Computing the optimal triangle strips for a given mesh is known to be $\mathcal{NP}$-complete [1]; therefore, various heuristic algorithms for stripping meshes have been developed [4], [19]. However, such *stripification* of a mesh can be applied only to off-line applications due to the high complexity of the computations involved.

In order to exploit triangle strips in a graphics system where the mesh topology changes frequently, procedures for the recalculation [8] and repairing [15] stripification per change have been formulated using triangle adjacency information such as adjacent face lists or the dual graph of the mesh, respectively. However, maintaining such information introduces unnecessary overheads and it also fails to exploit the correlation between the connectivity and the stripification of a mesh.

Since triangle strips themselves deliver a compact representation of the connectivity, they can be directly updated in the presence of changes in the connectivity resulting from simplification with edge collapses, as in Skip Strips [2]. However, the client may experience difficulties in maintaining the local copy of Skip Strips, which are built on top of vertex hierarchy that represents the set of all possible edge collapses. Moreover, Skip Strips approach defines only the process for simplification of triangle strips at full resolution and does not provide the inverse of such a process explicitly.

We present a novel mesh representation scheme exploiting the characteristics of triangle strips, called *progressive strips*, which gives rendering-efficient triangulation data at the rendering stage in a graphics system such as progressive transmission where the mesh topology changes continuously. Progressive strips consist of a set of triangle strips simplified to the base mesh and a set of refinement steps required to incrementally recover the original mesh at full resolution. In progressive strips representation, changes in the connectivity resulting from edge collapses are carried out by repeating vertices within triangle strips [2], [16]. This is a *strip-edge collapse* in this paper. We also reformulate the refinement data in order to compute the inverse of a strip-edge collapse, referred to as a *strip-vertex split*.

Strip-edge collapses and strip-vertex splits incorporate the stripification of a mesh and the associated changes in the mesh topology by directly updating triangle strips; thus it allows us to fully exploit the correlation between the connectivity and the stripification. However, identical vertices begin to accumulate within triangle strips as the mesh de-

Manuscript received April 22, 2004.

[†]The authors are with the Department of Computer Science, Yonsei University, Seoul, Korea.

[††]The author is with the Department of Internet Information, Sejong University, Seoul, Korea.

[†††]The author is with the Department of Computer Science, Trinity College Dublin, Ireland.

a) E-mail: kimbu@yonsei.ac.kr

grades to the base mesh. Such vertices may cause some redundancy in terms of degenerate triangles with zero-areas and add overheads to the rendering process; furthermore, the amount of refinement data could introduce additional overhead to the network bandwidth.

To resolve these problems, we propose an improved triangle strip filtering algorithm and the encoding of a strip-edge collapse. The proposed filtering algorithm removes the patterns of degenerate triangles from the sequence of vertices of a strip, and then reorders the remaining vertices in such a way that two consecutive patterns of triangles share a common edge. The encoding of strip-edge collapses alleviates the accumulation of identical vertices; this is done by defining four offset difference symbols, each of which is the difference between positions at which the two vertices that are collapsed are placed in the sequence of vertices of a strip.

The proposed mesh representation scheme enhances the graphics performance as the amount of data sent to the graphics pipeline is substantially reduced. Experimental results show that progressive strips together with progressive transmission of our datasets can reduce the amount of data sent to the graphics pipeline by about 40.5%∼45.6%, compared with the conventional progressive mesh where the mesh is rendered as triangles individually. These results indicate that the proposed new filtering algorithm improves the filtering performance by about 19.4%∼22.4% over the previously reported approach. In addition, the proposed encoding method alleviates the accumulation of identical vertices by a factor of two thereby reducing the amount of refinement information by about 22.5%∼27.9% compared to progressive strips with non-encoding of strip-edge collapses.

The rest of this paper is organized as follows. After presenting the background to this paper in Sect. 2, we describe our progressive strips and encoding strip-edge collapse in Sect. 3. The triangle strip-filtering algorithm is proposed in Sect. 4, experimental results are given in Sect. 5 and the conclusions are drawn in Sect. 6.

2. Backgrounds

2.1 Representation of Meshes

3D meshes consist of two main components: *geometry* and *connectivity*. Geometry describes the coordinates of the mesh vertices in the 3D space, and connectivity describes how these vertices are connected in faces to form the mesh surfaces. A triangle mesh is one of the standard representations for 3D meshes in which each face has exactly three vertices as its corners. In some graphics API [12], [13], the *indexed representation* of a triangle mesh is the standard interface where a mesh is organized as an array of vertices and an array of faces where each face refers to its vertices through indices [9]; each index is assumed to occupy 2 bytes.

Figure 1 shows an example of the indexed triangles and the indexed triangle strips interface. In the indexed triangles, each triangle can be rendered individually by sending its three vertices to the graphics pipeline. Here, every mesh vertex is processed about six times and this involves passing its three coordinates (3 words) and optional normal (3 words), color (1 word), and texture information (2 words) from the memory to and through the graphics pipeline [10].

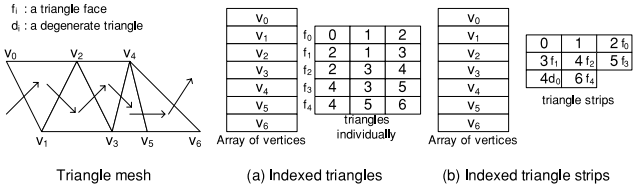


Fig. 1 Triangle mesh interface.

The most common way to enhance the graphics performance is to reduce the amount of data to be handled, using an efficient triangulation such as triangle strips. A strip encodes a sequence of triangles where every two consecutive triangles share a common edge. The sequence $(v_0, \cdots, v_{n-1})$ of n vertices represents triangles $\{v_i, v_{i+1}, v_{i+2}\}$ for $0 \le i \le n-2$. In the indexed triangle strips interface, the sequence of vertices can be simply represented as a list of indices such as (0, 1, 2, 3, 4, 5, 4, 6) as shown in Fig. 1 (b). In an ideal (*sequential*) strip, the direction of strip formation alternates between counterclockwise (ccw) and clockwise (cw) directions [13]. The direction can be overridden by duplicating a vertex, called a *swap* operation, in the sequence of vertices (for example, v_4 in Fig. 1 (b)). Such a strip is referred to as a *generalized* strip. A swap operation implies the insertion of a degenerate triangle between two consecutive triangles with the same direction as that of strip formation. In the above example, the degenerate triangle $\{v_4, v_5, v_4\}$ with cw orientation is inserted between two consecutive triangles $\{v_3, v_4, v_5\}$ and $\{v_5, v_4, v_6\}$ with the same ccw orientation. The indexed triangle strips form the mesh API referred to in the remainder of this paper and we assume that each strip is a generalized strip, which allows swap operations.

2.2 Simplified Triangle Strips

The trade-off between complexity and performance is an important issue for interactive graphics systems [11]. Simplification techniques can control the level-of-detail (LOD) of a mesh by reducing the number of triangles. Thus the rendering process can be accelerated by using a simplified version of the original model. An edge collapse is one of the popular operations for simplification and can be implemented by repeating vertices within triangle strips [16]; that is, if v_p and v_q are collapsed to v_p then all the occurrences of v_q are replaced by v_p in the sequence of vertices of a strip.

Figure 2 depicts an example of a strip-edge collapse where $v_q \rightarrow v_p$ denotes that v_p and v_q are collapsed to v_p. As shown in this example, more and more identical vertices are generated as the strip gets more simplified. Such a repetition also increases the patterns of degenerate triangles. For in-

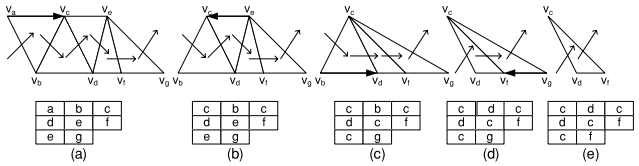


Fig. 2 Edge collapses on a strip.

stance, the sequence (c, d, c, d, c, f, c, f) of eight vertices includes one *valid* triangle $\{v_d, v_f, v_e\}$ and five degenerate triangles; they are $\{v_c, v_d, v_c\}$, $\{v_d, v_d, v_c\}$, $\{v_c, v_d, v_c\}$, $\{v_c, v_f, v_c\}$ and $\{v_f, v_f, v_c\}$ as shown in Fig. 2 (e). In this paper, we use the term *valid triangle* to mean a triangle with three consecutive distinct vertices, as opposed to a degenerate triangle.

Obviously, sending such redundant vertices to the graphics pipeline not only increases the data traffic between the memory and the graphics processor but also adds overhead to the rendering process in drawing degenerate triangles. Therefore, the elimination of redundant vertices from a strip and the use of well-defined strips for rendering, are of paramount importance in enhancing the graphics performance.

2.3 Simple Triangle Strip Scanner

The simple triangle strip scanner (STSS) [2] has been designed to filter out simplified strips to well-defined ones, called *display strips*, which are more suitable for displaying. It is based on the fact that repeating vertices produce one of three patterns, (a a b), (a b a) and (b a a), since two candidate vertices for strip-edge collapses are placed in such a way that one vertex is positioned within one or two places either in front of or behind the other vertex. Moreover, such patterns form next to each other and accumulates in a strip as the mesh gets more simplified. Hence, STSS acknowledges the patterns of vertices as some regular expressions, $(\text{a a})^+$ and $(\text{a b})^+$, while scanning from each strip and replaces them with (a a) and (a b), respectively.

In Fig. 2 (e), a strip (c, d, c, d, c, f, c, f) of vertices will be filtered out to (c, d, c, f) as a display strip. In this case we send only four vertices to the graphics pipeline, while eight vertices will be sent in the simplified strip with non-filtering.

STSS works well at a finer level because there are only a few duplicate vertices and thus only a few redundant patterns. However, STSS can not generate well-defined display strips at a coarser level because there is no way of removing degenerate triangles that are induced from some redundant patterns such as (a a b b), (a a a b), (a b b c), ... etc. The possibility of numerous occurrences of such patterns exists at a coarser level.

3. The Framework of Progressive Strips

3.1 Overview

The progressive strips representation-based transmission of a 3D mesh consists of three stages: the preliminary stage, the simplification stage, and the reconstruction stage.

In the preliminary stage, the original mesh at full resolution is converted into well-defined set of triangle strips by a dedicated stripification algorithm off-line. In the simplification stage, triangle strips are simplified to those of the base mesh through a series of strip-edge collapses. Simultaneously, a set of refinement information is collected. Optionally each strip-edge collapse can be encoded to reduce the amount of refinement information. In the reconstruction stage, the base mesh is reconstructed to obtain the original mesh through a series of computations using the refinement information. Here the connectivity of each intermediate mesh is recovered as triangle strips; so, for rendering a mesh, each strip is decoded (if it was encoded at the simplification stage), passed through a triangle strip filter, and then sent to the graphics pipeline.

3.2 Progressive Strips

In progressive transmission, edge collapses and vertex splits are the fundamental operations to subtract and add the details from or to a mesh [7]. Each edge collapse removes one vertex, two triangles and three edges from the mesh if the edges are not the boundary edges. Conversely, each vertex split adds one vertex, two triangles and three edges to the mesh.

Figure 3 shows an example of an edge collapse and a vertex split for transitions between two meshes M^i and M^{i-1}, where M^i is the mesh at resolution i. Let v_p^i be the vertex with an index p in mesh M^i, and $ecol_i(\{v_p^i, v_q^i\})$ denotes an edge collapse of vertices v_p^i and v_q^i to v_p^i. Here, $ecol_i(\{v_4^i, v_2^i\})$ indicates that v_4^i and v_2^i are collapsed to v_2^i and v_4^i is deleted from the array of vertices. The mesh geometry will be remapped in such a way that all the indices less than the index of a removed vertex are decremented by one as shown in Fig. 3 (b). In addition, two triangles $\{v_2^i, v_3^i, v_4^i\}$ and $\{v_2^i, v_4^i, v_8^i\}$ are removed explicitly from the mesh as shown in Fig. 3 (c). The vertex v_2^{i-1} is split into two vertices v_2^i and v_4^i during transition from M^{i-1} to M^i using the inverse of $ecol_i(\{v_4^i, v_2^i\})$. The geometric information of v_4^i is inserted in such a way that all indices greater than or equal to its index are increment by one. Two triangles $\{v_2^i, v_3^i, v_4^i\}$ and $\{v_2^i, v_4^i, v_8^i\}$ are also added to the mesh.

In general, if v_p^i and v_q^i are collapsed to v_p^i and if the two triangles incident to that edge are $\{v_p^i, v_r^i, v_q^i\}$ and $\{v_p^i, v_q^i, v_s^i\}$, then the refinement information for a vertex split is given by

$$r_i = \{v_q^i, (p^i, q^i, r^i, s^i)\}, \tag{1}$$

which includes the geometry of v_q^i and the connectivity of two triangles $\{v_p^i, v_r^i, v_q^i\}$ and $\{v_p^i, v_q^i, v_s^i\}$.

Progressive meshes are useful and have been applied successfully for progressive transmission, mesh compression, selective refinement and mesh editing; however, the increase in the amount of the data that should be sent to the graphics pipeline is proportional to the *mesh complexity* since the triangles are rendered individually at the rendering

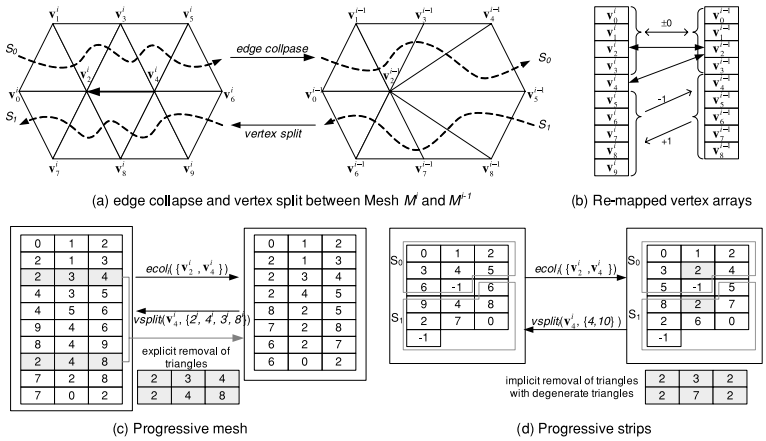


Fig. 3 Edge collapse and vertex split.

stage. The number of triangles in a mesh gives a measure of the mesh complexity. The increase in the amount of data can be reduced by exploiting the properties of the triangle strips at the rendering stage.

In progressive strips, an edge collapse is done by repeating vertices within triangle strips, and this results in degenerate triangles. During transition from M^i to M^{i-1} by $ecol_i(\{v_4^i, v_2^i\})$, all the occurrences of v_4^i are updated by v_2^i. Moreover, such a strip-edge collapse facilitates the implicit removal of triangles by creating degenerate triangles. For instance, two degenerate triangles $\{v_2^{i-1}, v_3^{i-1}, v_2^{i-1}\}$ and $\{v_2^{i-1}, v_7^{i-1}, v_2^{i-1}\}$ are generated in M^{i-1} instead of removing two triangles $\{v_2^i, v_3^i, v_4^i\}$ and $\{v_2^i, v_4^i, v_8^i\}$ explicitly from M^i.

Also, since the list of indices for each strip can be stored in the contiguous memory space, called an *index map*, the point at which updating of vertices occurs can be found by computing the offset which indicates the location with respect to the starting point of the index map; for example, when two v_4^i's are updated by v_2^i, their offsets are 4 and 10 in the index map, respectively; here we have assumed that each strip is restarted using a special vertex index '-1'. Such an interface has been implemented in the Microsoft Direc3D by the `DrawIndexedPrimitive()` function call. So, the refinement information can be represented as $r_i = \{v_4^i, (4, 10)\}$.

In general, if v_p^i and v_q^i are collapsed to v_p^i and if the offsets of v_q^i in the index map of M^i can be represented as $\sum^u o(v_q^i) = (o_1, \cdots, o_u)$, where $o(v)$ is the offset of the vertex v, then the refinement information for a strip-vertex split can be reformulated as follows

$$r_i = \left\{ v_q^i, \sum^u o(v_q^i) \right\}, \tag{2}$$

where u indicates the number of vertices to be updated by v_q^i during transition from M^{i-1} to M^i. It includes at least 12 bytes of the geometric information and $2 \times u$ bytes of the connectivity; thus, it requires $(12+2 \cdot u)$ bytes for computing each refinement.

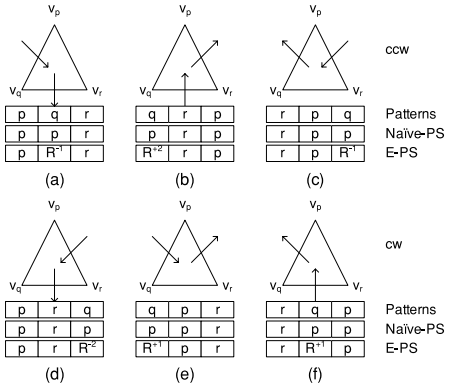


Fig. 4 The encoding of $ecol(\{v_p^i, v_q^i\})$.

3.3 Encoding Strip-Edge Collapse

A triangle $\{v_p, v_q, v_r\}$ is represented as one of six patterns in a strip, depending on the combination of the direction of a strip formation and the orientation of a triangle as shown in Fig. 4. Here the offset difference between two vertices of $ecol(\{v_p, v_q\})$ can be parameterized as $R^d = o(v_p) - o(v_q)$ with $d = -2, -1, +1, +2$. From this observation, we can encode some strip-edge collapses with four offset difference symbols $(R^{-2}, R^{-1}, R^{+1}, R^{+2})$.

Figure 4 illustrates $ecol(\{v_p, v_q\})$ without or with four offset difference symbols; these strips are called *Naïve-PS* and *E-PS*, respectively. The main objective of such an encoding is to reduce the accumulation of identical vertices as the mesh moves to a coarser level since the amount of refinement information is dependent on the number of vertices to be updated for a strip-vertex split as shown in Eq. (2).

Figure 5 illustrates strip-edge collapses showing the effectiveness of our encoding scheme which reduces the amount of refinement information. Here, transition between M^{i+1} and M^i by $ecol_{i+1}(\{v_2^{i+1}, v_4^{i+1}\})$, two v_4^{i+1}'s with the offsets of 4 and 10 are updated by v_2^i or encoded by offset difference symbols R^{-2} and R^{+2} as shown in Fig. 5 (e) and (h), respectively. The strip-vertex split can be achieved from the

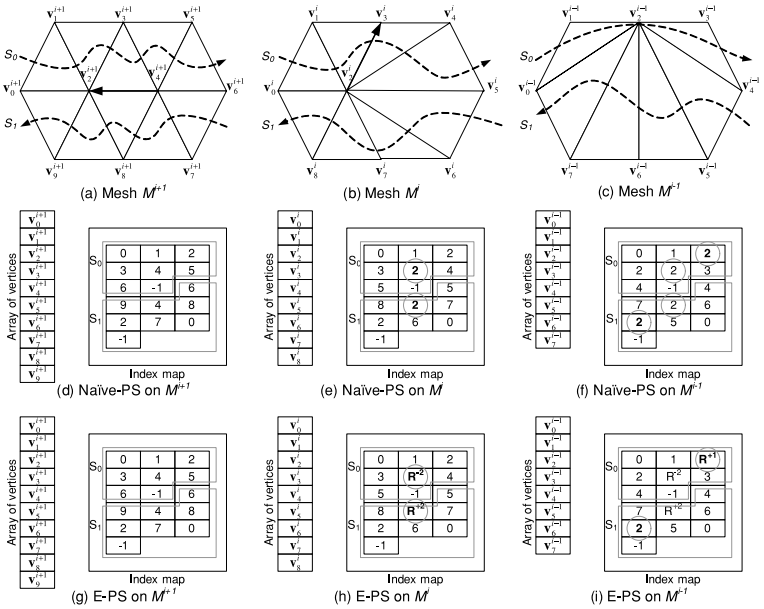


Fig. 5 The effectiveness of an encoding of a strip-edge collapse.

refinement information with $r_{i+1} = \{v_4^{i+1}, (4, 10)\}$ for both Naïve-PS and E-PS. Note that four v_2^i's are accumulated in Naïve-PS, while only two v_2^i's in E-PS. Such an accumulation will affect the size of data required in the refinement step if v_2^i is collapsed and then removed in the next transition.

During transition from M^i to M^{i-1} by $ecol_i(\{v_3^i, v_2^i\})$, two v_2^i's are updated by v_2^{i-1} or encoded by the offset difference symbol R^{+1} as shown in Fig. 5 (i), while four v_2^i's are replaced by v_2^{i-1} in Fig. 5 (f). Therefore, each vertex split from M^{i-1} to M^i is parameterized by $r_i = \{v_2^i, (2, 4, 10, 12)\}$ and $r_i = \{v_2^i, (2, 12)\}$ for Naïve-PS and E-PS, respectively. In this case, the number of vertices to be updated using the refinement information can be reduced by a factor of two in our encoding algorithm.

3.4 Encoding Strip-Edge Collapse – Exceptions

There are two exceptions in encoding a strip-edge collapse; the offset difference of two vertices that are collapsed cannot be represented by any of R^d. One of the exceptions occurs when a strip-edge collapse is performed on the common edge of a strip, which is shared by two consecutive triangles; for example, the edge $\{v_2^i, v_3^i\}$ in Fig. 5 (b) is the common edge shared by two consecutive triangles $\{v_1^i, v_3^i, v_2^i\}$ and $\{v_2^i, v_3^i, v_4^i\}$. This type of strip-edge collapse generates a *disjoint* vertex; a disjoint vertex splits one strip into two strips. More importantly, it is introduced newly to the other strip; thus, it cannot be represented with the offset difference symbols. For $ecol_i(\{v_3^i, v_2^i\})$, v_3^i (v_2^{i-1} after re-mapped) is a disjoint vertex. Then it replaces v_2^i in the strip s_1 and the strip s_0 is split on v_2^{i-1} as shown in Fig. 5 (c).

The other exception occurs on a strip with a spiral pattern as shown in Fig. 6. A spiral pattern may have a *closed*

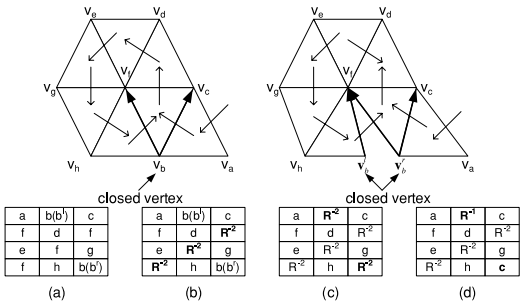


Fig. 6 Exception cases in the strip with a spiral pattern.

vertex which is duplicated twice within a strip except for a swap operation. A closed vertex, for instance v_b, can be treated as two separate vertices, v_b^l and v_b^r. Note that all swap operations can be encoded with symbol R^{-2} as shown in Fig. 6 (b). After that, if v_b and v_f are collapsed to v_f, then the closed vertex (conceptually, two vertices v_b^l and v_b^r) can be encoded by the offset difference symbols R^{-2} and R^{+2}, respectively, as shown in Fig. 6 (c) However, if v_b and v_c are collapsed to v_c then only v_b^r can be encoded by R^{-1}, while v_b^l is replaced by v_c since there is no edge between v_b^l and v_c as shown in Fig. 6 (d).

4. Triangle Strip Filtering Algorithm

The triangle strips for each intermediate level of a mesh may include numerous degenerate triangles unnecessary for rendering due to the implicit removal of triangles in the simplification stage. So removing them is quite important to enhance the graphics performance since the amount of data sent to the graphics pipeline is reduced.

The proposed filtering algorithm is based on the ob-

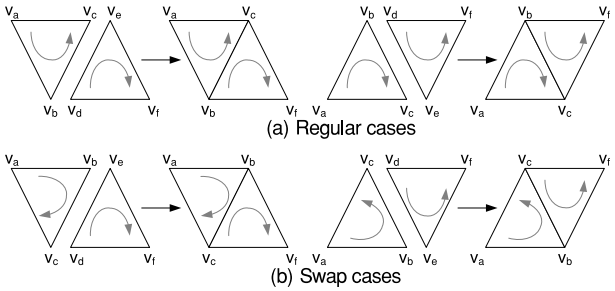


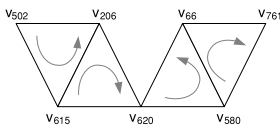
Fig. 7 Local strip reordering.

servation that two consecutive patterns of a valid triangle scanned from strips share one or two common vertices. Moreover if two consecutive patterns of a valid triangle share two vertices, we can convert them into a generalized strip. Otherwise, the strip is split into two strips; the current strip is terminated by one triangle and the next strip is started by the subsequent triangle.

The proposed filtering algorithm consists of two phases: the *degenerate triangle-free scanning* and the *local strip reordering*. In the degenerate triangle-free scanning phase, a sequence of vertices is scanned from each strip and then detects only the patterns of a valid triangle. Here, the pattern of a valid triangle consists of three consecutive distinct vertices. Such scanning allows us to have a strip without the patterns of degenerate triangles. The proposed filtering algorithm is called, *degenerate triangle-free scanner* (DTFS). Let two consecutive patterns of a valid triangle be (a b c) and (d e f) with $a \neq b \neq c$ and $d \neq e \neq f$, respectively, and we assume that the previous pattern (a b c) has already been added to the current display strip and the current pattern ($d\ e\ f$) is now tested to determine whether it should be added to the current display strip or not. In the local strip reordering phase, they could be transformed as a generalized strip, depending on the incidence relationship between two consecutive patterns of valid triangles as follows:

- **The regular case**: if two consecutive patterns of valid triangles share two vertices, $((v_b == v_d)\ AND\ (v_c == v_e))$, then their orientation alternates between cw/ccw as shown in Fig. 7 (a). In this case, the v_f is added to the current display strip and the resultant strip is organized as (a, b, c, f). Note that this can save two vertices in the pattern of the valid triangle, (d e f), since they share the common edge $\{v_b, v_c\}$ and this naturally leads to reuse of the previous two vertices v_b and v_c in the triangle $\{v_a, v_b, v_c\}$.
- **The swap case**: if two consecutive patterns of valid triangles share two vertices, $((v_b == v_e)\ AND\ (v_c == v_d))$, then their orientation does not alternate as shown in Fig. 7 (b). In this case, v_b and v_f are added to the current display strip and the resultant strip is transformed as (a, b, c, b, f). Note that this can save one vertex in the pattern of the valid triangle, (d e f), since the latter v_b is repeated for implementing a swap operation.
- **The split case**: if two consecutive patterns of valid triangles are applicable to neither the regular case nor the swap case, then the strip is split into two strips; the current display strip is terminated with the triangle ($\cdots$ a, b, c) and the new display strip is started with the triangle (d, e, f $\cdots$).

A list of indices: (615, 615, 502, 615, 502, 615, 206, 620
206, 620, 620, 620, 620, 620 620, 580, 66, 580, 66, 580, 66,
761, 761, 761, 761, 258, 761)



Four patterns
of a valid triangle
(502 615 206)
(615 206 620)
(620 580 66)
(580 66 761)

Fig. 8 An example of a simplified strip.

Figure 8 shows an example of the simplified version of a strip. From the example, STSS gets (615, 615, 502, 615, 206, 620, 620, 580, 66, 761, 258) as a display strip. DTFS filters a list of indices into (502, 615, 206, 620, −1, 620, 580, 66, 761) where '−1' indicates the starting of a new triangle strip and is not sent for rendering. In this example, the numbers of vertices to be sent for rendering with STSS and DTFS are 11 and 8, respectively, while 12 vertices are sent if each triangle is transmitted individually. Note that our filtering algorithm removes 19 redundant vertices from the non-filtered strip.

5. Experimental Results

We have simulated our progressive strips on four different 3D models: they are 'Cow', 'Skull', 'Dragon', and 'Bunny' which can be easily obtained from the public domain and they have a wide range of mesh complexity. Table 1 gives the characteristics of each model; the number of triangles (T) and the number of vertices (V) for each of the base mesh (M^0) and the original mesh (M^n), and the number of edge collapses (*ecols*) computed for the base mesh. In our experiments, the base mesh complexity was set at 5% of triangles in the original mesh.

In order to convert the original mesh into a set of triangle strips, one of the popular dedicated stripification programs STRIPE [3] was used. The triangle strips were simplified to those of the base mesh. For simplifying the mesh, the half-edge collapse and the quadric error metric (QEM) [5] were used. We decided to use the half-edge collapse as a simplification operator because it can reuse vertex information such as geometry of one of the two vertices that are the end-points of an edge [6]. This has an added advantage of saving storage when models are refined. QEM was used when vertex collapse pairs were chosen for simplification, which is known as a simplification error metric that offers a combination of speed, robustness, and fidelity [11].

The main objective of progressive strips is to reduce the amount of data sent to the graphics pipeline by exploiting the properties of triangle strips at the rendering stage. So the efficiency of progressive strips representation of a mesh can be measured easily in terms of the cost for rendering;

Table 1 The mesh characteristic.

Dataset	Cow	Skull	Dragon	Bunny
M^n (V/T)	2903/5804	10952/22104	25418/50761	35947/69451
M^o (V/T)	129/254	495/1104	1271/2538	1756/3471
ecols	2722	10412	24129	33077

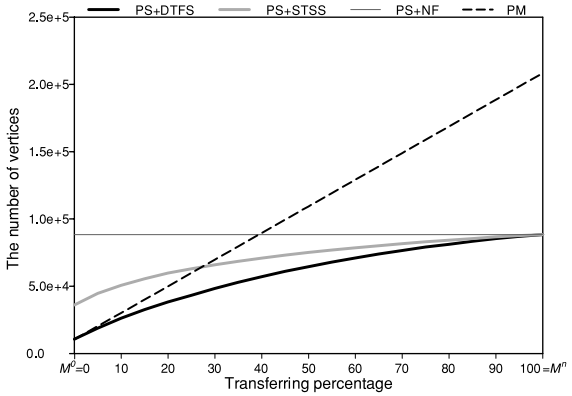


Fig. 9 The costs for rendering in the 'Bunny' model.

this is done by counting the number of vertices that must be sent to the graphics pipeline.

Figure 9 shows the cost comparison between the progressive mesh (PM) and the progressive strips (PS) for the 'Bunny' model, where progressive strips are implemented without a triangle strip filtering algorithm (PS+NF), and with the simple triangle strip scanner (PS+STSS) and the degenerate triangle-free scanner (PS+DTFS). As one would expect, except for PS+NF the costs increase as the mesh transforms gradually to the original mesh. The cost of PS+NF is constant with respect to the original mesh since a strip-edge collapse does not remove triangles explicitly but creates degenerate triangles due to the implicit removal of triangles. The cost of PM increases linearly with respect to the mesh complexity since one or two triangles are added to the mesh depending on whether a vertex split occurs on the boundary edge or not, and three vertices per triangle are sent for rendering.

In a triangle strip filtering algorithm incorporated in the progressive strips, most of the degenerate triangles are removed at a finer level. So, the cost is reduced by up to 57.6% compared with that of PM at the original mesh level. However, the performance of PS+DTFS and PS+STSS become closer to that of PM as the mesh degrades. PS+STSS sends more vertices than PM does at some coarser level; for instance, the percentage of transfer is less than 25%. This shows that STSS does not provide well-defined strips at a coarser level because of its poor filtering performance.

DTFS gives better performance compared to STSS over the entire transmission, especially at coarser levels. However, overridden vertices for assuring the consistent orientation of the first triangle in each strip may increase the cost slightly; for instance, the cost of sending 10,622 and 10,413 vertices to the graphics pipeline for PS+DTFS and PM, respectively, at the base mesh level. The worst case

Table 2 The experiments of progressive strips.

Dataset	Cow	Skull	Dragon	Bunny
Total amount of data traffic				
PM	2.6 Gbits	37.9 Gbits	201.3 Gbits	377.1 Gbits
PS+NF	2.3 Gbits	35.1 Gbits	180.7 Gbits	305.0 Gbits
PS+STSS	1.9 Gbits	29.0 Gbits	153.1 Gbits	260.6 Gbits
PS+DTFS	1.5 Gbits	22.3 Gbits	119.7 Gbits	205.0 Gbits
Filtering time on the average				
STSS	1.91 μs	8.19 μs	18.1 μs	23.37 μs
DTFF	1.77 μs	7.63 μs	16.81 μs	21.78 μs
The number of vertices to be updated on the average				
Naïve-PS	6.6	7	6.7	5.9
E-PS	3	3.1	3.1	3

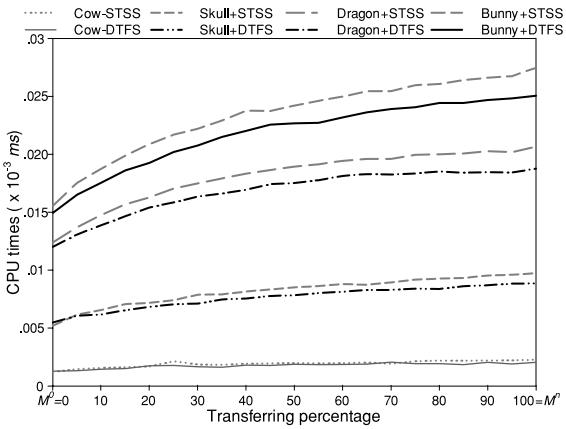


Fig. 10 Filtering time for STSS and DTFS.

value of DTFS is $4T$ and this occurs when each triangle in a mesh is itself an isolated strip and starts with cw (assuming that a strip starts with ccw).

The total amount of data traffic between the memory and the graphics processor is given in Table 2 where each intermediate mesh from the base mesh to the original mesh was rendered only once. It is clear that, PS+DTFS reduces the data traffic by 40.5%~46.5% on the average compared with PM. Similarly, DTFS improves the triangle strip filtering performance 16.5%~18.3% compared with STSS.

The processing time of a triangle strip filtering algorithm is an important factor that determines the performance of the entire graphic performance since displaying the first frame of each intermediate mesh should be delayed until the build-up of display strips is completed. Each of DTFS and STSS takes $O(n)$ time units since they process the scanning of vertices sequentially from the first vertex to the last one in a strip, where n is the number of vertices. Figure 10 compares the time complexities of STSS and DTFS based on the estimated CPU times on a 2.4 GHz Pentium-IV PC. The results show that the computation time for filtering increases as the number of redundant patterns that can be skipped decreases at a finer level. In practical implementation, DTFS improves slightly the filtering time by (6.8%~7.7%) against STSS, as shown in Table 2, since three consecutive vertices for each vertex need to be read from a strip in DTFS, while four consecutive vertices are read in STSS during the scan-

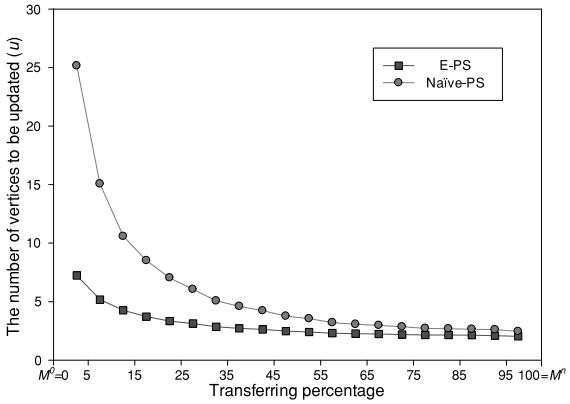


Fig. 11 The cost for recovering progressive strips in the 'Bunny' model.

ning of a strip.

To demonstrate the effectiveness of an encoding of strip-edge collapses, two kinds of progressive strips with (E-PS) and without (Naïve-PS) encoding strip-edge collapse were implemented. Figure 11 shows the number of vertices to be updated (u in Eq. (2)) in the 'Bunny' model, where u is averaged over each interval of 5% of the entire transmission. As one would expect, u increase as the mesh degrades to the base mesh since identical vertices get accumulated due to strip-edge collapses. The proposed encoding algorithm can reduce u by 49.9%∼55.2% on the average as shown in Table 2. This reduces total amount of refinement information by 25.8% on an average as per Eq. (2).

6. Conclusion

The main contribution of this paper is the development of an algorithm to generate rendering-efficient triangulation data at the rendering stage in progressive transmission of 3D meshes, where the mesh topology changes continuously. The proposed progressive strips representation of a mesh fully exploits the existing correlation between the connectivity and the stripification of a mesh since the connectivity for each intermediate mesh is directly updated on triangle strips through a set of refinement information. Thus it is possible to increase the graphics performance by reducing the cost for rendering.

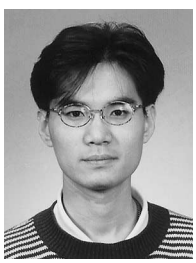
Our approach is useful for interactive visualization, where a client renders frequently or continuously the intermediate version of a mesh before receiving the full bit stream. Moreover, the efficient use of triangle strips reduces the cost for storing the local copy of a mesh at full resolution at the client site. Therefore The proposed scheme can have important applications in mobile graphics system, where the local resources such as the memory and the rendering capacity are somewhat restricted relative to the conventional client-server model.

Acknowledgement

We would like to thank to the anonymous reviewers for their comments and suggestions to improve the original manuscript. This paper was supported by IITA funds from the Ministry of Information and Communication of Korea.

References

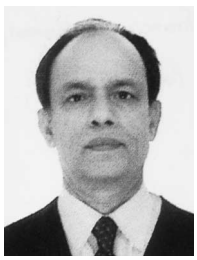
[1] E. Arkin, M. Held, J.S.B. Mitchell, and S. Skiena, "Hamiltonian triangulations for fast rendering," The Visual Computer (International Journal of Computer Graphics), vol.12, no.9, pp.429–444, 1996.
[2] J. El-Sana, F. Evans, S. Skiena, and E. Azanli, "Efficiently computing and updating triangle strips for real-time rendering," Computer-Aided Design, vol.32, no.13, pp.753–772, 2000.
[3] F. Evans, E. Azanli, S. Skiena, and A. Varshney, Stripe Version 2.0, http://www.cs.sunysb.edu/∼stripe
[4] F. Evans, S. Skiena, and A. Varshney, "Optimizing triangle strips for fast rendering," Proc. IEE Visualization, pp.319–326, San Francisco, California, United States, Oct. 1996.
[5] M. Garland and P. Heckbert, "Surface simplification using quadratic error metrics," Computer Graphics (SIGGRAPH), vol.31, pp.209–216, 1997.
[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," Computer Graphics (SIGGRAPH), vol.27, pp.19–26, 1993.
[7] H. Hoppe, "Progressive meshes," Computer Graphics (SIGGRAPH), vol.30, pp.99–108, 1996.
[8] H. Hoppe, "View-dependent refinement of progressive meshes," Computer Graphics (SIGGRAPH), vol.31, pp.189–198, 1997.
[9] H. Hoppe, "Optimization of mesh locality for transparent vertex caching," Computer Graphics (SIGGRAPH), vol.33, pp.269–276, 1999.
[10] M. Isenburg, "Triangle strip compression," Computer Graphics Forum, vol.20, no.2, pp.197–204, 2001.
[11] D. Luebke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson, and R. Huebner, Level of Detail for 3D Graphics, Morgan Kaufmann, 2003.
[12] http://www.microsoft.com/directx
[13] J. Neider, T. Davis, and M. Woo, OpenGL Programming Guide, Addison-Wesley, Reading, MA, 1993.
[14] W. Pugh, "Skip lists: A probabilistics alternative to balanced trees," Commun. ACM, vol.33, no.6, pp.668–678, 1990.
[15] A.J. Stewart, "Tunneling for triangle strips in continuous level-of-detail meshes," Graphics Interface, pp.91–100, Jun. 2001.
[16] SIGGRAPH'2000 Course Note, Advanced Issues in Level of Detail, Aug. 2000.
[17] J. Xia, J. El-Sana, and A. Varshney, "Adaptive real-time level-of-detail-based rendering for polygonal models," IEEE Trans. Vis. Comput. Graphics, vol.3, no.2, pp.171–183, June 1997.
[18] T.A. Möller and E. Haines, Real-Time Rendering, Second ed., pp.454–462, AK Peters, 2000.
[19] X. Xiang, M. Held, and J.S.B. Mitchell, "Fast and effective stripification of polygonal surface models," ACM Symposium on Interactive 3D Graphics, pp.71–78, April 1999.



Byung-Uck Kim received the B.S. and M.S. degree in Computer Science from Yonsei University, Seoul, Korea, in 1996 and 1998. He is currently a Ph.D. candidate at the Department of Computer Science, Yonsei University, Seoul, Korea, and his research interests include computational geometry, geometric processing, mesh optimization, and polygon-based rendering system.

Woo-Chan Park serves as a faculty member at the Department of Internet Engineering, Sejong University, Seoul, Korea, and his research interests includes 3D graphics accelerator architecture, micro-architecture, and computer arithmetic. He received the Ph.D. degree in Computer Science from Yonsei University.

Sung-Bong Yang serves as a faculty member at the Department of Computer Science, Yonsei University, Seoul, Korea, and his research interests includes mobile system, 3D computer graphics, and electronics commerce. He received the Ph.D. degree in Computer Science from the University of Oklahoma.



Francis Neelamkavil is a Professor of Computer Science at Trinity College Dublin, Ireland, and was responsible for the development of teaching and research in Computer Modelling & Simulation, Computer Graphics, Computer-Aided Design and Human-Computer Interaction there. He was an IITA Visiting Professor of Computer Science at Yonsei University, Seoul, during the academic year 2003-04, and participated in the International Joint Research Programme on Interactive 3D Data Visualization.