JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 21, 1277-1288 (2005)

# Short Paper

## A Simple and Efficient Triangle Strip Filtering Algorithm

BYUNG-UCK KIM, KYOUNG-WHA KIM, WOO-CHAN PARK
SUNG-BONG YANG AND TACK-DON HAN
*Department of Computer Science*
*Yonsei University*
*Seoul, 120-749 Korea*
*E-mail: kimbu@yonsei.ac.kr*

A triangle strip is one of the standard rendering primitives used to reduce the amount of data transmitted to the graphics pipeline. In order to exploit such efficient triangulation data in level-of-detail-based rendering applications, real-time updating of triangle strips is required, and it can be done by repeating vertices. However, the number of degenerate triangles increases in strips as the mesh degrades to a coarser level. Such triangles only add overhead to the rendering task. In this paper, we propose a simple and efficient triangle strip filtering algorithm to convert a strip into a more suitable one for display. The proposed algorithm eliminates the vertices of degenerate triangles from a strip and then reorders the remaining vertices in such a way that the two consecutive triangles share a common edge. In addition, the filtering process takes $O(n)$ time units since it scans vertices only once and does so sequentially from the first vertex to the last one in a strip. Experimental results show that the proposed filtering algorithm not only reduces the amount of data transmitted to the graphics pipeline by 22.2% on average, but also slightly reduces the filtering time by 7.1% on average, compared with the previous method.

***Keywords:*** 3D graphics, triangle mesh, triangle strip, level-of-detail, degenerate triangle

### 1. INTRODUCTION

A triangle mesh is one of standard representations for 3D meshes, where each face has exactly three vertices as its corner. At the rendering stage, each triangle face 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 transmitting its three coordinates and optional normal, color and texture information from the memory to and through the graphics pipeline. 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.

A common way to reduce such data is to represent a mesh as a set of triangle strips, where each of the 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

Received May 29, 2003; revised May 18, 2004; accepted May 31, 2004.
Communicated by C. C. Jay Kuo.

graphics pipeline by a factor of two or three, compared with sending triangles individually. Computing the optimal triangle strips for a given mesh is known to be *NP*-complete [1]; therefore, various heuristic algorithms for stripping a mesh have been developed [4, 10]. However, such *stripification* of a mesh can be applied only in off-line applications due to the high complexity of the computations involved. Thus run-time updating of triangle strips is required in order to exploit triangle strips in graphics systems in which the mesh topology changes frequently.

The skip strips data structure [2] has been used to efficiently compute and update triangle strips by repeating vertices in the presence of connectivity changes resulting from simplifications, such as edge collapses. These updated strips are referred to as *simplified* strips. However, the duplicated vertices may be repeated many times within simplified strips as a mesh degrades to a coarser level and the mesh size is reduced. Such vertices produce *degenerate triangles*, each of which has zero area. Obviously, sending such a redundant triangle and its associated 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 as degenerate triangles are drawn. Therefore, the elimination of redundant vertices from a strip and the use of well-defined strips at the rendering stage are of paramount importance for enhancing the graphics performance.

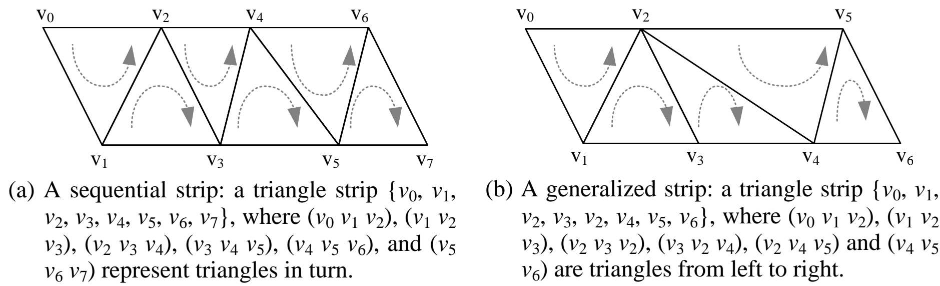
In this paper, we propose a simple and efficient triangle strip filtering algorithm, called the *degenerate triangle-free filter* (DTFF), which can reduce the number of degenerate triangles and their associated vertices. DTFF detects the pattern of vertices which forms a *valid* triangle. Here, a valid triangle consists of three consecutive distinct vertices. Such scanning allows us to obtain a strip without degenerate triangles. Then, the algorithm reorders the remaining vertices in such a way that two consecutive valid triangles share a common edge. As a result, we can establish well-defined triangle strips even at a coarser level. In addition, DTFF takes $O(n)$ time units since it scans vertices only once and does so sequentially from the first vertex to the last one in a strip. The experimental results show that DTFF not only reduces the amount of data transmitted to the graphics pipeline by 22.2% on average, but also slightly reduces the filtering time by 11.2% on average, compared with the previous method.

## 2. PREVIOUS WORK

In this section, triangle strips, simplified strips, and the previous triangle strip filtering approach are reviewed briefly.

### 2.1 Triangle Strips

The most common way to enhance the graphics performance is to reduce the amount of data to be handled, using an efficient triangulation approach, such as one that employs triangle strips. A strip encodes a sequence of triangles, where every two consecutive triangles share a common edge. The sequence $\{v_0, \ldots, v_{n-1}\}$ of $n$ vertices represents triangles $(v_i\ v_{i+1}\ v_{i+2})$ for $0 \le i \le n - 2$. In an ideal (*sequential*) strip, the direction of strip formation alternates between counterclockwise (ccw) and clockwise (cw) directions [8] as shown in Fig. 1 (a). The direction can be overridden by duplicating a vertex, called



(a) A sequential strip: a triangle strip $\{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, where $(v_0\ v_1\ v_2)$, $(v_1\ v_2\ v_3)$, $(v_2\ v_3\ v_4)$, $(v_3\ v_4\ v_5)$, $(v_4\ v_5\ v_6)$, and $(v_5\ v_6\ v_7)$ represent triangles in turn.

(b) A generalized strip: a triangle strip $\{v_0, v_1, v_2, v_3, v_2, v_4, v_5, v_6\}$, where $(v_0\ v_1\ v_2)$, $(v_1\ v_2\ v_3)$, $(v_2\ v_3\ v_2)$, $(v_3\ v_2\ v_4)$, $(v_2\ v_4\ v_5)$ and $(v_4\ v_5\ v_6)$ are triangles from left to right.

Fig. 1. Examples of two types of triangle strips.

a *swap* operation, in the sequence of vertices (for example, '$v_2$' 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 $(v_2\ v_3\ v_2)$ with cw orientation is inserted between two consecutive triangles $(v_1\ v_2\ v_3)$ and $(v_3\ v_2\ v_4)$ with the same ccw orientation. The generalized strips form the mesh API referred to in the remainder of this paper. To assure a consistent direction of strip formation, we assume that the first triangle in each strip starts to for in the ccw direction; this implies that a triangle for even $i$ forms in the ccw direction, and that a triangle for odd $i$ forms in the cw direction.

## 2.2 Simplified Strips

The trade-off between complexity and performance is an important issue in interactive graphics systems [7]. 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 the simplified version of an original model. An edge collapse is one of the operations commonly involved in simplification and can be implemented by repeating vertices within triangle strips [9]; that is, if '$p$' and '$q$' collapse to '$p$', then all the occurrences of '$q$' are replaced by 'p' in the sequence of vertices.

Fig. 2 depicts an example of such an edge collapse. As shown in this example, more and more identical vertices are generated as the strip becomes more simplified. Such repetition also increases the number of patterns of degenerate triangles. For instance, the sequence $\{c, d, c, d, c, f, c, f\}$ of eight vertices includes one valid triangle $(d\ c\ f)$ and five degenerate triangles; they are $(c\ d\ c)$, $(d\ c\ d)$, $(c\ d\ c)$, $(c\ f\ c)$, and $(f\ c\ f)$ as shown in Fig. 2 (e).

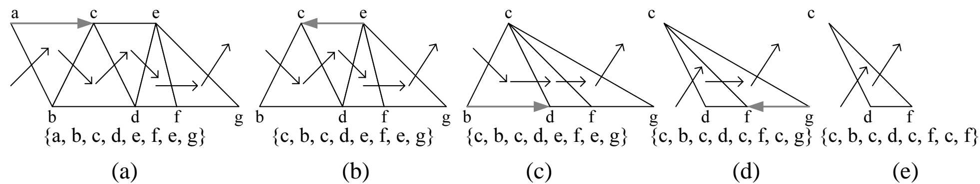



Fig. 2. Example of a simplified strip, where $q \rightarrow p$ denotes that $p$ and $q$ collapse to $p$.

### 2.3 The Simple Triangle Strip Scanner

The simple triangle strip scanner (STSS) [2] has been used to filter out simplified strips to obtain triangle strips, called *display strips*, which are more suitable for display. STSS is based on the fact that repeating vertices produces one of three patterns, (*a a b*), (*a b a*), on (*b a a*), since two candidate vertices for an edge collapse 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 accumulate in a strip as the mesh becomes more simplified. Hence, STSS acknowledges the patterns of vertices as regular expressions $(a\ a)^+$ and $(a\ b)^+$ while scanning from each strip and replaces them with (*a a*) and (*a b*), respectively. In Fig. 2 (e), the strip {*c*, *d*, *c*, *d*, *c*, *f*, *c*, *f*} will be filtered out to obtain {*c*, *d*, *c*, *f*} as a display strip. In this case, we send only four vertices to the graphics pipeline, while eight vertices are sent without 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 resulting from some redundant patterns, such as (*a a b b*), (*a a a b*), (*a b b a*), … etc. The possibility of numerous occurrences of such patterns exists at a coarser level.

## 3. THE PROPOSED FILTERING ALGORITHM

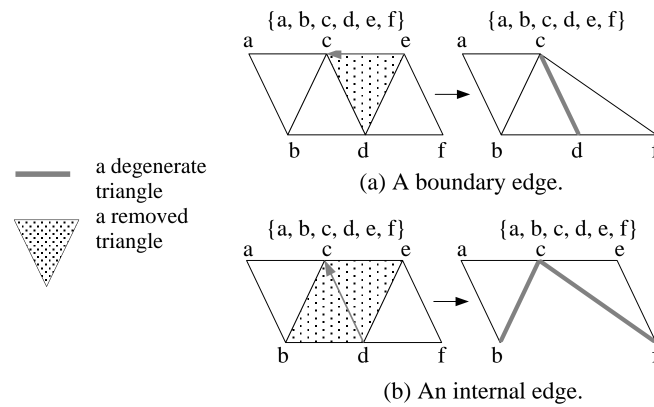
In this section, we describe an edge collapse transformation in triangle strips and the proposed filtering algorithm in detail. An example of triangle strips filtered with STSS and DTFF is also presented.

### 3.1 Edge Collapses in Triangle Strips

An edge collapse transformation collapses an edge {*p*, *q*} to a single vertex '*r*'. This causes the edge {*p*, *q*} as well as the triangles associated with that edge to be removed. For simplicity, we assume a half-edge collapse. Then, the vertex '*r*' is set to '*p*'. The same process is applied to a triangle strip, but instead of explicitly removing the edge and the triangles from the current mesh, vertex duplication is performed; that is, each occurrences of '*q*' are replaced by '*p*' in the current strip. As described in section 2.2, this induces a pattern of vertices which produces degenerate triangles and, thus, performs the removal of triangles.

The edges in triangle strips are of two types: *internal* edges and *boundary* edges. An internal edge is shared by the subsequent triangles except for the first triangle, and all other edges are regarded as boundary edges. An edge collapse transformation on a boundary edge removes one triangle but creates one degenerate triangle. For example, when edge {*c*, *e*} collapses as shown in Fig. 3 (a), the triangle (*c d e*) is removed and the degenerate triangle (*c d c*) is created. More importantly, the triangles spanning a degenerate triangle still share two vertices. This implies that the simplified strip resulted from an edge collapse on a boundary edge can be reproduced as a generalized strip.

However, collapsing an internal edge results in both removing two triangles and creating two degenerate triangles. As shown in Fig. 3 (b), triangles (*b c d*) and (*c d e*) are




(a) A boundary edge.

(b) An internal edge.

Fig. 3. Implicit removals of triangle(s).

removed from the current mesh, and two degenerate triangles, (*b c c*) and (*c c e*), are present in the simplified strip. This forces the simplified strip to be split into two strips since two consecutive triangles spanning degenerate triangles share only one vertex. Therefore, a triangle strip is apt to be fragmented due to frequent collapsing of an internal edge as the mesh becomes more simplified.

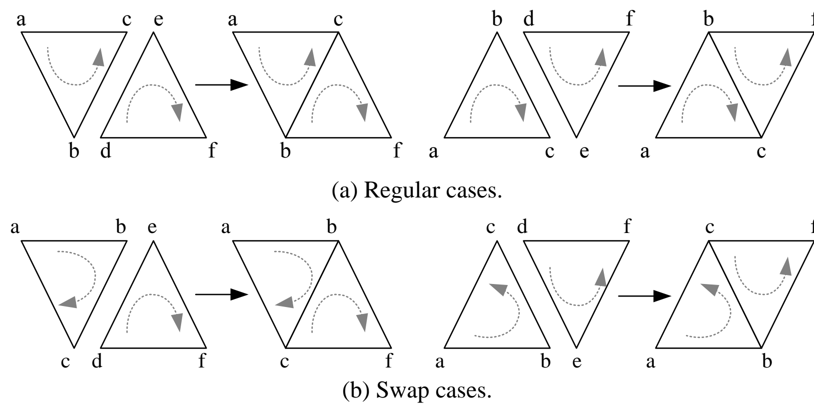
### 3.2 The Degenerate Triangle Free-Filter

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. Thus, removing them is quite important for enhancing the graphics performance since the amount of data sent to the graphics pipeline is reduced.

The proposed filtering algorithm is based on the observation that two consecutive patterns of a valid triangle scanned from a strip 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 will be split into two strips; the current strip will be terminated by one triangle, and the next strip will be started by the subsequent triangle.

Our approach consists of two phases: *degenerate triangle-free scanning* and *local strip reordering*. In the degenerate triangle-free scanning phase, a sequence of vertices is scanned from each strip, and it then detects only the patterns of a valid triangle. Again, the pattern of a valid triangle consists of three consecutive distinct vertices. This scanning step allows us to have a strip without the patterns of degenerate triangles. The degenerate triangle-free filter is named after this property. 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. Also assume that the previous pattern (*a b c*) has already been added to the current display strip, and that the current pattern (*d e f*) will now be tested to determine whether it can be added to the current display strip or not.

In the local strip reordering phase, they can be represented as a generalized strip, depending on the incidence relationship between two consecutive patterns of valid triangles, as described below.




(a) Regular cases.

(b) Swap cases.

Fig. 4. Examples to illustrate how DTFF works in the regular and swap cases.

**The regular case:** If two consecutive patterns of valid triangles share two vertices, ((*b* == *d*) *AND* (*c* == *e*)), then their orientation alternates between ccw and cw (or vice versa) as shown in Fig. 4 (a). In this case, '*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 {*b*, *c*}, and that this naturally leads to reuse of the previous two vertices, *b* and *c*, in the triangle (*a b c*).

**The swap case:** If two consecutive patterns of valid triangles share two vertices, ((*b* == *e*) *AND* (*c* == *d*)), then their orientation does not alternate as shown in Fig. 4 (b). In this case, '*b*' and '*f*' are added to the current display, and then {*a*, *b*, *c*, *b*, *f*} is resulted as a display strip. Note that this can save one vertex in the pattern of the valid triangle, (*d e f*), since '*b*' is repeated to implement a swap operation.

**The split case:** If two consecutive patterns of valid triangles are applicable neither to the regular case nor the swap case, then the strip is split into two strips; the current display strip is terminated with the triangle {… *a*, *b*, *c*} and the new display strip is started with the triangle {*d*, *e*, *f*, …}. Note that the new display strip may started with the sequence {*d*, *d*, *e*, *f*, …} if the direction of (*d e f*) is cw; that is, one degenerate triangle (*d d e*) is inserted for assuring the consistent direction of a strip formation where the direction of triangle ($v_i$, $v_{i+1}$, $v_{i+2}$) can be easily computed by ccw for even *i* and cw for odd *i* from the definition of triangle strips.

Fig. 5 shows the pseudo code of DTFF. As in STSS, DTFF takes $O(n)$ time units since it scans vertices only once and does so sequentially from the first vertex to the last one in a strip, where *n* is the number of vertices. In the procedure, *IsValidTriangle*() investigates three consecutive vertices as the pattern of a valid triangle or a degenerate triangle. If a degenerate triangle is detected then it is skipped. *IsFirstTriangle*() finds the first valid triangle in a simplified strip and the new display strip is started with *NewDisplayStrip*(). *ComputeTheRelation*() investigates the relationship between the previous valid triangle (*PrvValTri*) and the current valid triangle (*CurValTri*). Depending on the

```
Procedure: DTFF ( )

for (i = 0; i < strip->VTS-2; ++i) {    // strip->VTS: the number of vertices
// v[i], v[i + 1], v[i + 2]; three consecutive vertices in a strip.
    if (IsValidTriangle(v[i], v[i + 1], v[i + 2]) {
            if (IsFirstTriangle(v[i], v[i + 1], v[i + 2]))
                NewDisplayStrip(v[i], v[i +1], v[i +2]);    //start new display strip.
           else {
                RS = ComputeTheRelationShip(PrvValTri, CurValTr);
                if (RS == Regular)
                        AddToCurrentDisplayStrip(v[i + 2]);
                else if (RS == Swap)
                         AddToCurrentDisplayStrip(v[i], v[i + 2]);
                else    // Split
                      NewDisplayStrip(v[i], v[i + 1], v[i + 2]);
            }
            SetPreviousValidTriangle(v[i], v[i + 1], v[i + 2]);
    }    // just skip if the current pattern is not valid triangle
}
```

Fig. 5. Pseudo code of DTFF.

A sequence of vertices in a simplified strip: 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

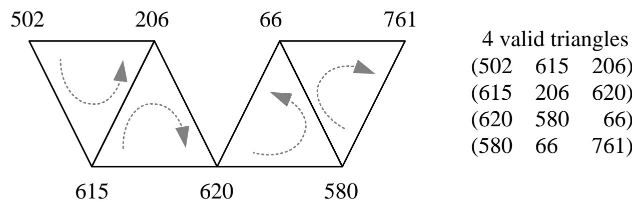



Fig. 6. An example of a sequence of vertices in a simplified strip. The sequence is extracted from one of the triangle strips in model 'Eight' at 10% LOD of the original model.

incidence relationship, DTFF adds one or two vertices to the current display strip with *AddToCurrentDisplayStrip*(), or splits (the new display strip is started). If the process of the current valid triangle is done then it is set to as the pattern of the previous valid triangle. The next three consecutive vertices are fetched and its filtering process is preceded.

Fig. 6 shows an example of the simplified version of a strip. From the sequence of 27 vertices in the example, STSS gets {615, 615, 502, 615, 206, 620, 620, 580, 66, 761, 258} as a display strip. DTFF filters a sequence of vertices to obtain {502, 615, 206, 620, – 1, 620, 580, 66, 761}, where '– 1' indicates the starting point 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 DTFF are 11 and 8, respectively, while 12 vertices are sent if each triangle is transmitted individually.

## 4. EXPERIMENTAL RESULTS

We have simulated our filtering algorithm on four different 3D models: 'Bishop', 'Eight', 'Femur', and 'Triceratops', which can be easily obtained from the public domain. Table 1 lists the characteristics of each original model at full resolution; the number of triangles (T), the number of vertices (V), the number of vertices in triangle strips (VTS), the number of triangle strips (TS), and the number of swaps (S).

**Table 1. The characteristics of the 3D models.**

| Models | T | V | VTS | TS | S |
|---|---|---|---|---|---|
| Bishop | 496 | 250 | 570 | 1 | 72 |
| Eight | 1536 | 766 | 1648 | 24 | 64 |
| Femur | 7798 | 3897 | 10254 | 237 | 1982 |
| Triceratops | 5660 | 2832 | 7372 | 144 | 1424 |

In order to convert the original mesh into a set of triangle strips, one of the popular stripification programs, STRIPE [3], was used. Then, the triangle strips were simplified with the half-edge collapse operation and the quadric error metric [5] until 5%-LOD of the original mesh was achieved. We decided to use the half-edge collapse operation as a simplification operator because it can reuse vertex information, such as the geometry of the previous LOD [6]. QEM was used when vertex collapse pairs were chosen for simplification. It is a simplification error metric that offers a combination of speed, robustness, and fidelity [7].

The various simplified models employed in our experiment are shown in Fig. 7. In addition, we extracted a sequence of vertices uniformly from skip strip nodes. In other words, we do not impose any view-dependent constraints. Such extraction does not affect the performance of a filtering algorithm but offers more generality even for a continuous LOD.

The main objective of the triangle strip filtering algorithm is to reduce the amount of data at the rendering stage. Thus, the efficiency of the proposed filtering algorithm can be measured in terms of the cost of rendering; this is done by counting the number of vertices that need to be sent to the graphics pipeline.

Fig. 8 shows the cost comparison between DTFF and STSS, respectively, at various LODs for the four 3D models. We also present the result of transmitting each triangle individually (TETI). As one would expect, the costs increase as the mesh moves gradually to the original mesh. DTFF reduces more significantly the number of vertices sent for rendering than STSS, especially as a model moves toward a coarser level. DTFF also shows good performance over TETI due to an efficient triangulation data. On the other hand, STSS provides well-defined triangle strips as much as DTFF at a finer level, but sends even more vertices than TETI at certain coarser levels. Table 2 shows DTFF saves more vertices than TETI and STSS on the average about and 44.4% and 22.2%, respectively.

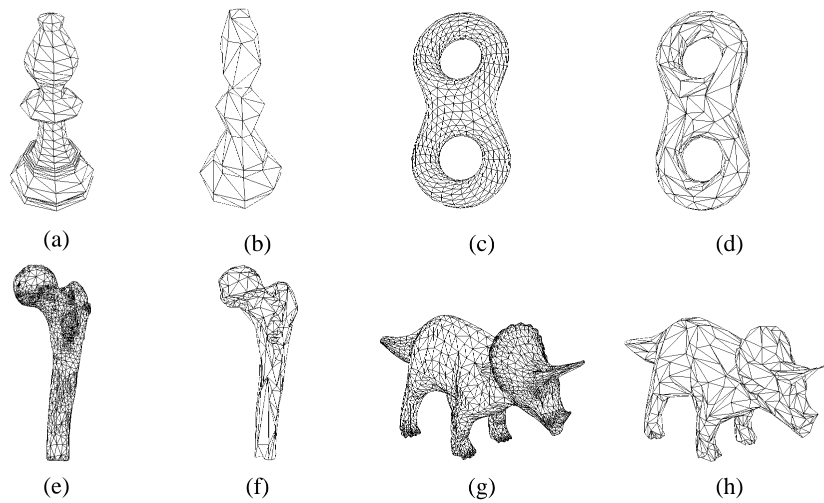



Fig. 7. Various simplified models: (a) the original Bishop, (b) 30%-LOD Bishop(148/247/32/35), (c) the original Eight, (d) 40%-LOD Eight(614/997/117/149), (e) the original Femur, (f) 10%-LOD Femur(778/2099/484/351), (g) the original Triceratops, (h) 20%-LOD Triceratops(1134/2808/591/492), where the numbers within parentheses are T, VTS, TS, and S from (a) to (h).

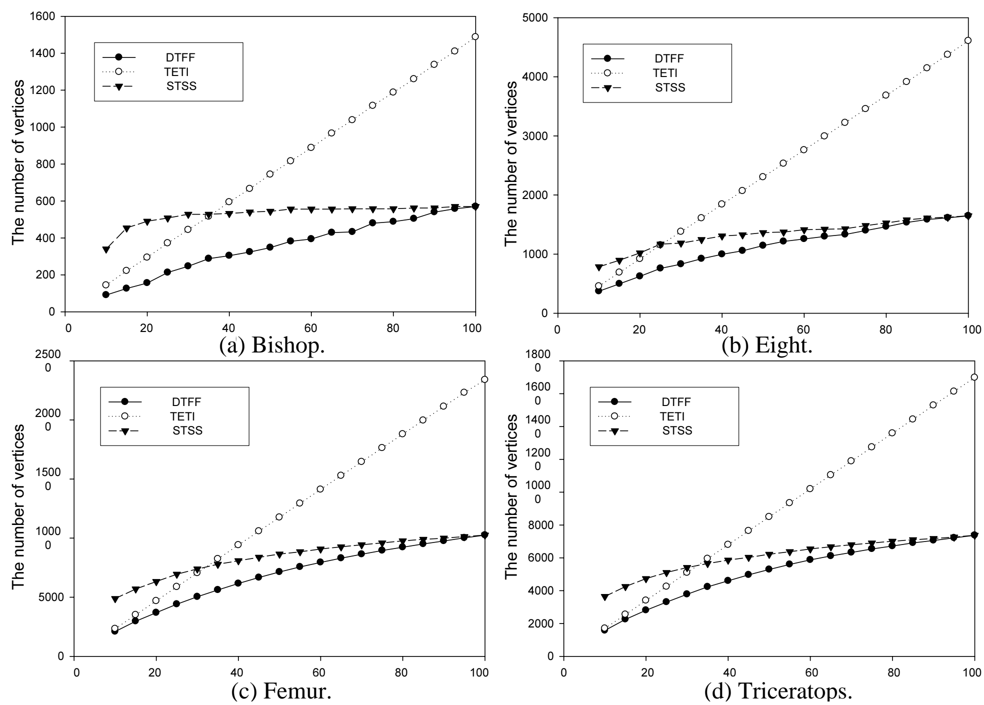



Fig. 8. The number of vertices sent to the graphics hardware for each 3D model at various LODs. The *X*-axis indicates the LODs at an interval of 5% from 10% to 100%.

**Table 2. The saving ratios with respect to the number of vertices to be sent for rendering, where DvS denotes for DTFF vs. STSS, DvT for TETI and SvT for STSS vs. TETI. The unit is percentage.**

| LODs | Bishop | | | Eight | | | Femur | | | Triceratops | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DvS | DvT | SvT | DvS | DvT | SvT | DvS | DvT | SvT | DvS | DvT | SvT |
| **10** | 73.2 | 36.8 | **-136.1** | 52.7 | 18.4 | **-72.4** | 57.0 | 10.1 | **-109.3** | 56.8 | 7.2 | **-114.6** |
| **15** | 72.2 | 43.2 | **-104.5** | 44.4 | 27.8 | **-29.9** | 47.8 | 15.2 | **-62.3** | 46.9 | 11.6 | **-66.6** |
| **20** | 68.2 | 46.9 | **-66.7** | 38.8 | 32.0 | **-11.1** | 41.6 | 21.1 | **-35.0** | 40.6 | 17.5 | **-39.0** |
| **25** | 58.1 | 42.7 | **-36.6** | 35.0 | 34.1 | **-1.4** | 36.5 | 25.0 | **-18.1** | 35.2 | 22.2 | **-20.1** |
| **30** | 53.2 | 44.4 | **-18.9** | 30.0 | 39.9 | 14.1 | 31.8 | 28.5 | **-4.7** | 30.0 | 25.8 | **-6.0** |
| **35** | 45.5 | 44.2 | **-2.3** | 26.0 | 42.7 | 22.5 | 27.8 | 31.8 | 5.5 | 25.1 | 28.8 | 4.9 |
| **40** | 42.9 | 48.8 | 10.4 | 23.7 | 45.9 | 29.1 | 23.8 | 34.5 | 14.1 | 21.5 | 32.4 | 13.8 |
| **50** | 36.0 | 53.2 | 26.9 | 16.0 | 50.3 | 40.9 | 17.4 | 39.3 | 26.4 | 14.9 | 37.7 | 26.9 |
| **60** | 29.1 | 55.6 | 37.4 | 11.0 | 54.5 | 48.8 | 12.4 | 43.7 | 35.7 | 10.2 | 42.4 | 35.8 |
| **70** | 22.4 | 58.3 | 46.2 | 6.7 | 58.6 | 55.7 | 8.4 | 47.5 | 42.7 | 6.8 | 46.7 | 42.8 |
| **80** | 12.5 | 58.9 | 53.0 | 4.1 | 60.3 | 58.6 | 5.5 | 50.9 | 48.0 | 4.1 | 50.6 | 48.4 |
| **90** | 3.9 | 59.6 | 58.0 | 1.5 | 61.7 | 61.2 | 2.4 | 53.8 | 52.7 | 1.6 | 53.8 | 53.1 |
| **100** | 0.0 | 61.7 | 61.7 | 0.0 | 64.2 | 64.2 | 0.0 | 56.2 | 56.2 | 0.0 | 56.6 | 56.6 |
| **AVG.** | 33.6 | 52.2 | 9.8 | 17.8 | 49.0 | 31.3 | 19.4 | 38.9 | 15.5 | 17.9 | 37.3 | 14.7 |

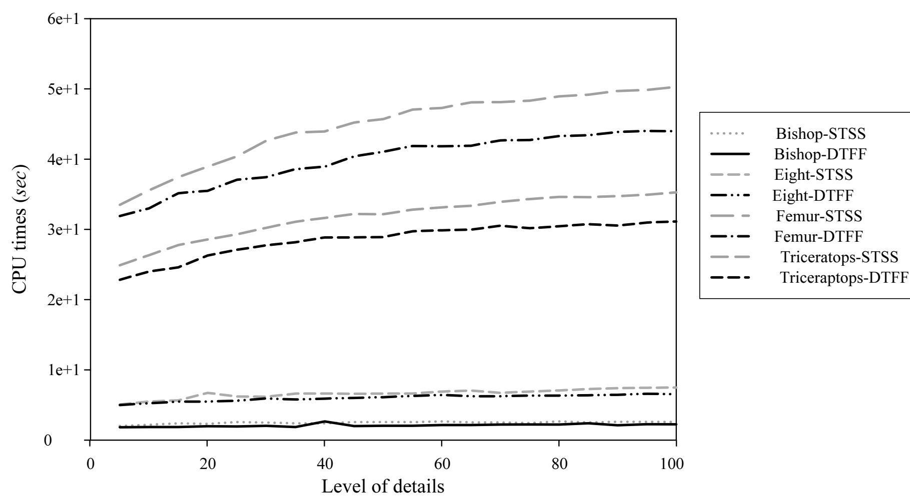



Fig. 9. Time performance of DTFF and STSS.

The processing time of a triangle strip filtering algorithm is another important factor that determines the performance of the filtering algorithm since displaying the first frame of each intermediate level of mesh should be delayed until the build-up of display strips is completed. Fig. 9 compares the time complexities of STSS and DTFF, based on the estimated CPU times on a 2.4 GHz Pentium-IV PC with 512M RAM, where both STSS

and DTFF are processed 10000 times per level. 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, DTFF slightly reduces the filtering time by 9.4% ~ 14.5% compared to STSS since three consecutive vertices for each vertex need to be fetched from a strip in DTFF, while four consecutive vertices are read in STSS when a strip is scanned.

## 5. CONCLUSIONS

In this paper, we have proposed a simple and effective triangle strip filtering algorithm that removes degenerate triangles and their associated vertices from simplified strips, and then reproduces them in forms more suitable for display. The key idea in DTFF is to share the common edges of the two consecutive valid triangles in each simplified strip, which naturally results in excluding degenerate triangles, except for those needed for swap operations. Hence, we can effectively reduce the number of vertices sent to the graphics pipeline. Sending a smaller number of vertices results in per-vertex bandwidth saving; thus, we can potentially achieve better rendering performance. DTFF not only produces better quality triangle strips compared to STSS, especially as a model degrades to a coarser level, but also slightly reduces the filtering time. DTFF is quite easy to implement and can be well integrated into any LOD-based application in which changes of the mesh connectivity are achieved through run-time updating of triangle strips performed by repeating vertices.

## ACKNOWLEDGMENTS

We would like to thank to F. Evans and M. Garland for allowing us to access their valuable implementations on mesh stripification and simplification, respectively, through their homepage, and the anonymous reviewers for their comments and suggestions to improve the original manuscript. This research was supported by the NRL-Fund from the Ministry of Science & Technology of Korea.

## REFERENCES

1. E. Arkin, M. Held, J. Mitchell, and S. Skiena, "Hamiltonian triangulations for fast rendering," *The Visual Computer*, Vol. 12, 1996, pp. 429-444.
2. J. El-Sana, F. Evans, S. Skiena, and E. Azanli, "Efficiently computing and updating triangle strips for real-time rendering," *The Journal Computer-Aided Design*, Vol. 32, 2000, pp. 753-772.
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," in *Proceedings of 7th Annual IEEE Visualization Conference*, 1996, pp. 319-326.
5. M. Garland and P. Heckbert, "Surface simplification using quadratic error metrics," in *Computer Graphics* (*SIGGRAPH '97 Proceedings*), Vol. 31, 1997, pp. 209-216.

6. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," in *Computer Graphics* (*SIGGRAPH '93 Proceedings*), Vol. 27, 1993, pp. 19-26.
7. D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level of Detail for 3D Graphics*, Morgan Kaufmann, 2003.
8. J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide*, 3rd Edition, Addison-Wesley, Reading, MA, 1999.
9. SIGGRAPH 2000 Course Note no. 41, Advanced Issues in Level of Detail, 2000.
10. X. Xiang, M. Held, and J. S. B. Mitchell, "Fast and effective stripification of polygonal surface models," in *Proceedings of ACM Symposium on Interactive 3D Graphics*, 1999, pp. 71-78.

**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 of Yonsei University and his research interests include computational geometry, geometric processing, mesh optimization, and polygon-based rendering system.

**Kyoung-Wha Kim** received his B.S. degree in Mathematics and M.S. degree in Computer Science from Yonsei University, Seoul, Korea, in 2002 and 2004. He is currently working at LG Electronics Inc.

**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.

**Tack-Don Han** serves as a faculty member at the Department of Computer Science, Yonsei University, Seoul, Korea, and his research interests includes high performance computer architecture, media system architecture, and wearable computing. He received the Ph.D. degree in Computer Engineering from the University of Massachusetts.