An Efficient Edge-Based Compression Algorithm for 3D Models with Holes and Handles*

BIN-SHYAN JONG, WEN-HAO YANG+ AND SHAWN SONG**
Department of Information and Computer Engineering
"Department of Electronic Engineering
Chung Yuan Christian University
Chungli, 320 Taiwan
"Department of Electronic Engineering
Chin Min Institute of Technology
Miaoli, 351 Taiwan
"R&D Division Engineer
VIA Technologies, Inc.

Recently numerous studies have attempted to develop efficient algorithms for compressing/decompressing geometric data. Almost all of these algorithms are either multiple pass traversals or operate in reverse order. Multiple pass traversals take a long time to execute. Operating in reverse order should work only off-line since its decompression order follows the reverse order of the compression. These factors restrict numerous applications. To conquer these restrictions, this study proposes an edge-based single-resolution compression scheme for handling triangular mesh connectivity. The proposed algorithm encodes and decodes 3D models straightforwardly via single pass traversal in a sequential order.

Most algorithms use the split operation to separate the 3D model into two components; however the displacement is recorded or an extra operator is needed for identifying the branch. This study proposes using the J operator to skip to the next edge of the active boundary; the method need not split overhead. Meanwhile, this study proposes the cut operation to compress/decompress the triangular mesh with holes and handles. The experimental results demonstrate that the proposed algorithm achieves better compression ratio and faster execution time than the conventional algorithm. The proposed algorithm is combined with a graphics engine, and the hardware structures and vertices replacement strategies are also presented.

Keywords: geometry compression, triangular mesh connectivity, single-resolution, graphics engine, hardware structures

1. INTRODUCTION

Recently, the continuing development of information technology has ensured increasing adoption of diversified computer applications. Moreover, the feasibility of virtual information has been extensively studied. Upgrading the hardware can perfect the virtual reality and computer graphics technologies based on three dimensional (3D) en-
environments. Virtual reality and computer graphics recently have received extensive attention, and consequently their applications have mushroomed. Given limitations of Internet bandwidth, a major current issue is the size of 3D models in applications involving virtual reality and computer graphics. Such models must exhibit extremely high quality to ensure user visual enjoyment, and thus generally require very large file sizes, creating problems in transmission on the Internet.

3D models usually consist of numerous types of information: including vertices, edges, faces, normals, texture coordinates, colors, and so on. Geometry compression was proposed for reducing the size of 3D models [5]. In applications, triangular meshes are one of the most popular data structures for representing 3D models. When a 3D model is composed of lots of triangles, the shared vertex of the adjacent triangles is stored recurrently. This storage requirement causes file size wastage because of the need to save the same information in duplicate. Moreover, the repeated transmission required in drawing also wastes time and space. Compression is the key to solving this problem. The algorithm used in the previous scientific literature transformed triangular meshes into various patterns such as string, fan, annularity, aster, dendrite, spiral or zigzag [13, 14, 32, 33] to suit the compression requirements.

The main research on geometry compression algorithms can be divided into three types: single-rate connectivity compression [1, 6, 7, 10, 14, 16, 19, 21], the progressive rendering approach [2, 14, 16, 19, 23], and the compression scheme for dynamic geometry models [18, 27, 28]. This study established an improved single-rate connectivity compression. Currently, methods of single-rate connectivity compression can be divided into two types: vertex-based [1, 19, 33] and edge-based coders [24, 26, 29]. Touma and Gotsman [33] proposed the first approach by using the vertex-based type to compress triangular meshes. Their investigation encoded the mesh connectivity as a list of vertex degrees in counter-clockwise order and generated valence codes. When all these vertices are coquetted, each edge generates an “add” code, and three additional codes, dummy, merge and split, to encode boundaries, handles and conquest incidents, respectively. Furthermore, Alliez [1] proposed a valence-driven method, which uses part of the adaptive conquest technique to minimize the split codes and decrease the offset range. Finally, Alliez encoded the sequences of the codes using adaptive arithmetic coding [12].

For the edge-based approach, Edgebreaker, developed by Rossignac [24], is guaranteed to encode any unlabeled triangulated planar graph. Edgebreaker compression stores the model as a CLERS string, and adds a special context-based coding to compress the CLERS sequence. Edgebreaker compression/decompression provides for the worst-case analysis of information-theoretic lower bound, and can deal with problems involving linear time and space complexity [20, 22]. The improved compression rates can approach 1.622 bits per vertex for cases involving regular meshes. The other advantage of regularity and rigidity is its suitability for sufficiently large and regular meshes [30]. The Spirale Reversi algorithm [17] for decompressing avoids the need for two passes, but suffers the drawback of the decompression process being unable to start until all compression processes are completed. The Handlebody theory for surface is investigated to solve the simply connected manifold triangular mesh with handles [25]. The assistant data structure Corner-Table is used to recognize handles, but extensive memory space is used because the dimension of the data structure is three times the number of triangles.

Another edge-based approach was presented by Floriani [7], “VERTEX,” “SKIP,”
“LEFT,” and “RIGHT” operators are used for compressing/decompressing the triangular mesh. This approach resembles the Edgebreaker approach and is suitable for extendable shellable triangular mesh. Moreover, the approach has been applied to triangulated irregular networks for applications related to geographical information systems. However, surfaces with a non-null genus are not extendable shellable [3], and the algorithm failed for triangular meshes with holes and handles (through holes).

The valence-driven method needs to identify better pivot vertex duplicately, meaning the compression/decompression processes is complicated. The major disadvantage of the Edgebreaker spirale reversi method is that since the decompression works only for a single pass; its applications only work off-line. This study proposes an improved edge-based compression algorithm for reducing the complicated compression process, and avoiding decompression in reverse order. The proposed algorithm can both encode and decode 3D models straightforwardly via sequential single pass traversal, and does not require a split operator for splitting the model. The proposed algorithm is simple and easy to implement. Furthermore, the proposed algorithm can be applied to on-line applications. Besides shorter execution time, the proposed method also offers a higher compression ratio.

The remainder of this paper is organized as follows. Section 1 briefly surveys compression algorithms. Section 2 then introduces the Edgebreaker encoding/decoding method, and also provides some improved algorithms based on the Edgebreaker method. Next, section 3 presents the proposed improved algorithm, and describes the solutions for the triangular mesh with holes and handles. Subsequently, section 4 combines the proposed algorithm with the graphics engine, while also presenting the hardware structures and vertices replacement strategies. Finally, sections 5 and 6 give implementation results and some concluding remarks.

2. ORIGINAL EDGEBREAKER ENCODING METHOD

This section briefly summarizes the Edgebreaker technique. A series of studies on the Edgebreaker technique are described. The Edgebreaker method uses CLERS operators to compress every triangle and generate a binary string different from the encoding code. The model of the hole and handle can be handled by adding another scheme. The Edgebreaker approach can achieve an excellent compression rate [24].

2.1 Original Edgebreaker Encoding Method

Rossignac [24] employs five operators C, R, L, S, and E, as illustrated in Fig. 1, to encode and decode triangular meshes. The encoding process begins from an arbitrary triangle which is selected as the initial active boundary. From the boundary, the active gate is directed counter-clockwise around the triangle. The encoded triangles are declared to be inside the boundary and the remaining triangles are considered to be outside. The process continues until all triangles have been encoded, and each triangle includes one operator except the initial and final triangles.

From Fig. 1, a thick side symbolizes the active gate and a thick dot represents the third vertex. Using the active gate and the third vertex can compress a triangle.
Fig. 1. Five operators of the Rossignac algorithm.

Table 1. Encoding rules of the Rossignac algorithm.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>101</td>
</tr>
<tr>
<td>L</td>
<td>110</td>
</tr>
<tr>
<td>S</td>
<td>100</td>
</tr>
<tr>
<td>E</td>
<td>111</td>
</tr>
<tr>
<td>CS</td>
<td>010</td>
</tr>
<tr>
<td>CR</td>
<td>011</td>
</tr>
</tbody>
</table>

The C operator indicates that the third vertex is a new vertex with the code 0. Furthermore, the R operator demonstrates that the third vertex is located at the active boundary, and is the next vertex of the active gate. The R operator is encoded as code 101. Additionally, the L operator states that the third vertex is located at the active boundary, and is the previous vertex of the active gate. The code of the L operator is 110. Furthermore, the S operator shows that the third vertex is located on the active boundary, is neither the previous nor the next vertex of the active gate, and is coded 100. The S operator splits the object into two components. Finally, the E operator indicates that the third vertex is located at the active boundary, is the next and previous vertex of the active gate and has the code 111. This encoding ensures that each triangle needs 2.0 bites. Since L or E shall not be behind C, S is encoded as 10 behind C and R is encoded as 11 behind C. The compression ratio then becomes such that each triangle takes up 1.5 to 2 bites, with the average being 1.7 bits. Table 1 lists the encoding rules.

Gumhold [10] proposed similar operators to the Edgebreaker technique for compressing and decompressing 3D models. Four operators are used: *, ->, <- and \( \infty \). The geometric characteristics of operators *, -> and <- are the same as those of operators C, R, L. \( \infty \) splits the object into two components, and \( i \) records the displacement of the third vertex. Owing to the “E” operator, the Rossignac method can achieve a better compression ratio than Gumhold approach, eliminating the need to store the offset using displacement “i” as in S.
2.2 Decoding the Five Operators

The decoding process receives a binary encoding of a \textit{CLERS} string and decodes it into a series of triangles. The decoding process includes two pass traversals of the \textit{CLERS} string. In the first pass, preprocessing calculates offset values for all of the \textit{S} operations. In the second pass, the triangles are generated in a sequence within the Edgebreaker decoding process.

The preprocessing is designed to calculate an offset value for every \textit{S} operation. To decode an \textit{S} operator and create a triangle, the third vertex of the active triangle is located on the active boundary other than the previous or next one. The offset value is the distance between the active gate and this vertex along the active boundary. The preprocessing process continues until the number of encountered \textit{E} operators exceeds the number of encountered \textit{S} operators.

Subsequently, the second pass of decoding involves producing the desired triangular meshes. An initial triangle is obtained, the active gate is identified, and the \textit{CLERS} string is processed. The \textit{C} operator reads a new vertex. Meanwhile, the \textit{R} operator reads the right vertex from the active boundary. Furthermore, the \textit{L} operator reads the left vertex from the active boundary. Additionally, the \textit{S} operator obtains the third vertex from the pre-computed offset value. The \textit{E} operator represents the last triangle of each split.

Calculating the cost of the third vertex displacement achieves a better compression ratio. However, the length of the third vertex displacement does not show up until all vertices are accessed twice, meaning conclusions cannot be drawn the first time.

Another solution for decoding the \textit{CLERS} string is the Wrap&Zip [26] method. This method requires just one pass traversal of the \textit{CLERS} string. Edgebreaker decoding and the Wrap&Zip method differ in terms of how it is decided which vertex is the third vertex of the triangle. The look-ahead technique of the Wrap&Zip method is used to identify the third vertex. This method is postponed for the operations \textit{R}, \textit{L}, \textit{S}, and \textit{E}. A newly created vertex is only used for the \textit{C} operation, which is the wrapping part. During the zipping part the unlabeled vertices are eventually identified with a previously created vertex. Although the Wrap&Zip decoding [26] improves the original Edgebreaker decoding [24], the zipping procedure may deal with multiple pass traversals of the triangular meshes with handles.

Isenburg and Snoeyink [17] described a Spirale Reversi decoding method. This method only requires one reverse traversal of the \textit{CLERS} string. The \textit{CLERS} sequence is processed in reverse order at decoding process. Depending on the operation the active boundary is shrunk (operation \textit{C}), expanded (operations \textit{R} and \textit{L}), merged with a stack boundary (operation \textit{S}), or created new (operation \textit{E}). This method eliminated the need for the look-ahead procedure, but all of the data needs to be delivered before starting the reverse order decoding.

3. PROPOSED EDGE-BASED COMPRESSION ALGORITHM

This section describes a reformed compression method for triangular mesh and expatiates on the encoding/decoding processes. The subject method has numerous advantages. The encoding/decoding performs a single pass traversal and constructs triangles in...
straightforward order. The proposed algorithm need not split the model, preventing the stack structure from storing offset information that is suitable for limited storage devices such as PDAs. For communication purposes, the process achieves on receipt of one or partial compressed codes, and the decoding algorithm can recreate triangles and then immediately draw the partial mesh. The proposed method avoids the waiting time for transmission after compression of the whole meshes.

From past compression researches, one of the sides of the active boundary is the active gate, and this gate must correspond with one and only one un-compressed triangle. One active gate and the third vertex can form this un-compressed triangle to compressed triangle. This investigation can compress multiple triangles once in certain condition; it can achieve more effective compression than the Edgebreaker algorithm.

3.1 Operators and Coding

With all sorts of conditions of active gates and third vertices, this study adopted five operators, QCRLJ, as illustrated in Fig. 2, and then used them to encode and decode triangular mesh. The proposed algorithm adopts the CRL operators of Rossignac, and two new operators were proposed, Q and J. For explanatory purposes, these operators are described as follows.

![Fig. 2. Five operators of the proposed algorithm.](image-url)
From Fig. 2, the thin arrows symbolize the active boundary, a thick arrow symbolizes the active gate, and a thick hollow dot symbolizes the third vertex.

1. **C**: The third vertex is a new vertex, in which the active gate is subsequently removed from the active boundary. Meanwhile, the other two sides of the compressed triangle are added to the active boundary, and the right side is chosen as the new active gate. One triangle can be compressed at a time.

2. **R**: The third vertex is located at the active boundary, and is the next vertex of the active gate. One triangle can be compressed at a time. Subsequently, the active gate and the next side of the active gate are removed from the active boundary, after which the left side of the compressed triangle is added to the active boundary and designated as the new active gate.

3. **L**: The third vertex is located at the active boundary, and represents the previous vertex of the active gate. One triangle can be compressed at a time. The active gate and the previous side of the active gate are removed from the active boundary. Subsequently, the right side of the compressed triangle is added to the active boundary and allowed to serve as the new active gate.

4. **Q**: The third vertex is a new vertex and its consecutive triangle is **R**. These two triangles, which comprise a quadrilateral, are then shifted from the un-compressed area into the compressed area. The active gate then is removed and the other two sides of the quadrilateral that are not on the active boundary are moved to the active boundary, then the right side is allowed to serve as the new active gate. The geometric characteristics demonstrate that the **Q** operator represents two triangles which are coded **CR**. Different from the further context-based encoding for **CR** codes conducted by Rossignac, this approach only needs to read **Q** at the decompression process, and treats it as two triangles. However, using the context-based coder requires transforming the code to **CR**, and then acknowledges these two triangles.

5. **J**: The third vertex is located on the active boundary and is not the previous or next vertex of the active gate. This operator does not compress any triangle and the next side of active boundary is allowed to serve as the new active gate. The active gate skips to next edge of the active boundary. Since the third vertex that corresponds with the active gate comprises one triangle, and this triangle divided the un-compressed area into two, numerous indications for the third vertex are stumped up under this condition. Thus, this triangle is not compressed and is eventually compressed by “**R**” or “**L**”.

Since the new active gates continuously spin towards the right, “**R**” rises after several “**C**.” In the experimental survey, 75% of “**C**” is attached to “**R**.” Restated, “**C**” compresses 50% of the triangular meshes; therefore “**CR**” compresses 75% of the triangular meshes. If “**Q**” is used to replace “**CR**” during compressing, the **Q** operators must save extensive coding costs.

This study also noted that not all of the operators could be arranged randomly. Some operators clash with each other. There is, in fact, three circumstances can never occur: **CR**, **CL**, and **JL**. Since after **CR** is transformed into **Q**, **CR** does not exist. Additionally, taking Fig. 3 for instance, when triangle 123 was compressed by the active gate 12 with
operator $C$, $\angle 132$ must be smaller than $180^\circ$ and the active gate becomes side 32. Since the interior angles of a triangle can never exceed $180^\circ$, side 23 can never compress the new triangle with vertex 1 and $CL$ does not exist [35]. As for $JL$, Fig. 4 is taken as an example, when side 12 accesses “$J$,” the active gate becomes side 23; then the triangle 123 is compressed by “$L$”. Nonetheless, “$R$” can compress triangle 123 when the active gate is still at the side 12. Therefore $JL$ does not exist.

The probabilities of the five operators from highest to lowest are $Q$, $C$, $R$, $L$, and $J$ orderly. This study encodes $Q$ as 1, $C$ as 01, $R$ as 001, $L$ as 0001, and $J$ as 0000, according to Huffman coding. Since no $R$ or $L$ is attached to $C$, the $J$ behind $C$ is represented by 00. Additionally, $L$ cannot be attached to $J$, so the $J$ behind $J$ is represented as 1, $R$ behind $J$ is encoded as 01, $Q$ behind $J$ is encoded as 001, and $C$ behind $J$ is encoded as 000. Because only $Q$ and $C$ are able to read a new vertex, the running out of all vertices is known to indicate that there are no $Q$ and $C$. In those cases, 1 is allowed to denote $R$, while 01 represents $L$, and 00 indicates $J$. Since $L$ is not attached to $J$, the 0 stands for $J$ behind $J$. Table 2 lists the encoding rules of the proposed algorithm.

### 3.2 Compression Process

The compression process begins from reading three vertices of the original triangle. The three sides naturally fashion an active boundary, which separates the compressed
and un-compressed areas from start to finish. Following the compression, the active boundary compresses the triangle from the un-compressed area into the compressed area step-by-step. During the compression, the size of the active boundary first appears to be increasing, and then appear to be decreasing. When accessing “Q” or “J,” the size of active boundary remains; as accessing “C,” the size of active boundary is plus one; while accessing “R” or “L,” the size of active boundary minus one. The compression of the 3D triangular mesh ends when the un-compressed area contains no triangle.

Figs. 5 and 6 illustrate similar triangular meshes. This study illustrates the difference between the Rossignac algorithm and the proposed algorithm. Fig. 5 illustrates the Rossignac compression process [24], the thick side represents the active boundary, and the active gate is indicated by a blue arrow on the active boundary; the cut parts represent $S$ operators that split the object into two components. In the first step (Fig. 5 (b)), triangle C1 is set to operator C, and the active gate moves to the edge between C1 and C2. In the second step C2 is set as the operator C (Fig. 5 (c)), and the active edge is located between C2 and R3. Meanwhile, in the third step (Fig. 5 (d)), the third vertex is the next vertex of the active gate, and is coded R, while the active gate is located between R3 and C4.

Fig. 5. The Rossignac compression process.
The algorithm of Rossignac gives further details [24]. The algorithm contains 31 operators as \texttt{CCRCRSERCS CRRCRRRERC RCRCRRRLLR L} (the last \texttt{E} operator can be ignored) with binary codes 0011011100 1111010100 1110101110 110111101 0110111011 0110111011 01011110 and 67 bits. Fig. 6 illustrates the proposed compression course, where the dotted lines represent \texttt{J} operators. A total of 27 operators are calculated as \texttt{CQQJRLRCJQ QRRLLLRQQQ RRLLRLR} with binary codes 0111000001 0001001010 0110010010 0010001000 1001111110 1011011, and a total of 57 bits using the proposed algorithm.

3.3 Reduced Repeat Records Operator \texttt{J}

Supposed that the 3D triangular mesh is long and the starting triangle is located in the middle of the mesh, then the triangles near the center will be compressed sooner in the compression process. With the expansion of the active boundary, the un-compressed area is structured as a long and narrow band. The condition then repeats several “\texttt{J}” (as shown in Fig. 7) and takes up more space. Taking Fig. 7 for example, the dotted lines represent \texttt{J} operators and this figure outlines the circumstances when “\texttt{J}” is encoded repeatedly.

In fact, it is fascinating that if one side has been skipped by “\texttt{J},” the triangle, which it corresponds with, definitely does not use it as an active gate. Accordingly during the encoding/decoding, if we mark those sides have been escaped by “\texttt{J},” then the repetition of encoding “\texttt{J}” is reduced.

3.4 Mesh with Holes and Handles

This section encodes/decodes triangular meshes with holes and handles. The proposed method only needs minimal overheads for storing the hole and handle information during the compression. The work related to holes and handles initially only needs to be judged and dealt with when outputting vertex data and connectivity information. Therefore the triangular mesh with holes and handles can be decoded directly based on the hole and handle information.

Considering the triangular mesh with holes, the valence-driven algorithm [1] provides a dummy vertex for each hole. And then connects each boundary edge of the hole to the dummy vertex to generate a close triangular mesh. The dummy vertex is marked, and the triangular mesh is encoded. During the decoding, the dummy vertex can be removed.
Accordingly these triangles with the dummy vertex as their vertex can be given up. This approach using the dummy vertex spent more operators for dummy triangles.

This study adopts the “cut” procedure to encode triangular mesh with holes. As illustrated in Fig. 8 (a), the hole can be recognized using the Euler formula for manifold theory [25] at the preprocess time, where $V_1V_2V_3$ consists of the hole. During the encoding, the active gate and the vertex $V_1$ create a compressed triangle, and then the “cut” procedure separates the mesh as shown in Fig. 8 (b). The cut vertex and the hole information are outputted. In other words, vertices $V_1V_2V_3$ are recorded to the hole file to provide the boundary of the hole. The hole is marked, and the triangular mesh is encoded using the proposed algorithm described previously. During the decoding, the cut vertex and the hole information from the hole file guide the correct production of the original mesh.

Fig. 7. An example of repeat condition for the $J$ operator.
In case of the multiple adjacent holes, these connected points can be checked in the preprocessing time. If just one point is connected, two adjacent holes can be separated and become one joined hole (as shown in Fig. 9 (a)). Even the condition of the connected point more than one, the adjacent holes become one hole naturally (as shown in Fig. 9 (b)). Consequently, these holes are unique after the preprocess time and then these holes can be handled in the encoding/decoding processes without extra overhead.

To solve the triangular mesh with handles, Rossingnac investigated the Handle-body theory for surfaces [25]. The Corner-Table data structure stores the corners, vertices, and triangles of triangular mesh; this data structure uses the concept of a corner that represents the association of a triangle with one of its incident vertices. When the S operator is encountered the left branch triangle has been visited during their right branch traversal, the handle is recognized. An extra handle file is used to identify handles of the triangular mesh, after which the Zip routine is processed during decompression. The drawback of Rossingnac algorithm is that the dimension of the data structure is three times the number of triangles.

For considering the triangular mesh with handles, this study adopts the “cut” procedure for encoding/decoding the triangular mesh. Figs. 10 (a) and (b) display a smooth...
Fig. 10. (a) Smooth torus.  
Fig. 10. (b) Triangulated torus.

Fig. 10. (c) The flat of triangular rectangle.

Fig. 10. (d) The case of boundary loop.

Fig. 10. (e) Cut operation.

Fig. 10. (f) After compress Loop 1 part.

Fig. 10. (g) Join two parts.
torus and a triangulated torus respectively. And Fig. 10 (c) illustrates the flat of triangular rectangle for this triangulated torus. Identifying the vertices indicated by the numbers in Fig. 10 (d), the four corner vertices are marked as 10 which there is the same vertex. Similar cases exist for vertices 4, 2, 7, 11, and 12.

During the encoding, beginning with the gray triangle, the compression operators are $\text{CCCCQCCJ} \text{CJIJJJJJJ}$. Under this condition, the case of the boundary loop is encountered, and the J operator traverses 11, 12, 6, 5, 8, 7, 9, 2, 4, 10, 11 around the active boundary, as shown in Fig. 10 (d). Moreover, no further operator can be used to compress any un-compressed triangle; the condition is the Mobius strip characteristic for 3D viewing. This characteristic shows that the 3D object contains one handle hole. In this figure, the active boundary 12, 6, 5, 8, 7 and 11, 10, 4, 2, 9 (indicated by the arrows on the opposite sides) structured the un-compressed area as the long and narrow band.

This study uses the “cut” procedure for cutting the handle hole. Fig. 10 (e) presents the cut edge. The algorithm searches the narrow band for two consecutive edges on both sides of it. In this example, the edges 65, 58 and 92, 24 satisfy the searching condition; the third vertex of edge 65 is 4, the third vertex of edge 24 is 5; the third vertex of edge 58 is 9, and the third vertex of edge 92 is 5. The “cut” procedure cuts edge 52. Following the “cut” procedure, there are two active boundaries: loop 1 and loop 2. Loop 1 contains 10, 11, 12, 6, 5, 2, 4, 10 and loop 2 contains 7, 9, 2, 5, 8, 7. Notably, the “cut” procedure may not separate the object into two components; so in this example, another candidate cut edge is 64.

After the “cut” procedure, one active boundary is chosen for further compression. Fig. 10 (f) shows the encoding process for loop 1. While producing $\text{RLRLJJJ}$ operators, the J operator traverses the active boundary again, and no further operator can be used to compress any triangle. The case of the boundary loop occurs for 10, 11, 12, 10. Under this condition, the “cut” procedure is used again. Fig. 10 (g) shows the triangular mesh for loop 1 and loop 2, where two consecutive edges are found in both sides of the active boundary. In this example, the cut edge is 7, 10. Following the “cut” procedure, two active boundaries are joined to one active boundary, and the triangular mesh is encoded using the proposed algorithm described previously.

The encoding process for triangular mesh with handle is as follows:

1. To compress the triangular mesh using $\text{CQRLJ}$ operators. As the J operator traverses the active boundary, the boundary loop is encountered, and the handle hole is found.
2. Identifying two consecutive edges in both sides of the narrow band, where the boundary of the narrow band is the partial edges of the active boundary.
3. Cutting the object between two consecutive edges and separating the boundary loop into two active boundaries. Handle file is used to identify the cut edge.
4. To choose one active boundary for further compression by using $\text{CQRLJ}$ operators until the boundary loop is encountered again.
5. Identifying two consecutive edges in both sides of the active boundary, and then cutting the object again. Two active boundaries are joined to one active boundary, and the cut edge is recorded to the handle file.
6. Repeat steps (1) to (6) until the entire triangular mesh is compressed.

The decoding process for the triangular mesh with handle is described below:
(1) Reading CQRLJ operators to decompress triangular mesh until the J operator traversals around the active boundary, after which the boundary loop is encountered.
(2) Reading the cut edge from the handle file.
(3) Cutting the boundary loop into two active boundaries.
(4) Reading CQRLJ operators to decompress the triangular mesh until the boundary loop are encountered.
(5) Reading the cut edge from the handle file, then cutting and joining two active boundaries to produce a single active boundary.
(6) Repeat steps (1) to (6) until no CQRLJ operators are read.

The proposed algorithm can encode/decode the triangular mesh with multiple handles. For each handle hole, two “cut” procedures are conducted.

4. COMBINING PROPOSED ALGORITHM TO GRAPHICES ENGINE

Two parts are considered for combining the proposed algorithm with the graphics engine. First, how should the three vertices in the limited triangle register is replaced? The replace work can be exchanged as fewer vertices as possible under current algorithm restrictions. Second, do any alternatives exist when vertices exceed the capacity of the mesh buffer?

First, this study needs focus on the order in which the three vertices were put, and hopefully should be able to access one vertex to draw a new triangle. The traditional algorithm requires the storage of one more spinning byte to provide guidance during drafting. However, this study hopes to gain a precise direction while accessing any vertex.

Second, which vertices to remove when vertices exceed the mesh buffer capacity must be determined? If vertices that might soon be useful are removed, then the repetition of such vertices will increase, which is not desirable.

4.1 Vertex Replacement Strategy for Triangle Register

Vertices on the active boundary were all accessed previously, accessing R or L resembles using the previous or next vertex of the active gate as the third vertex. Consequently, whole vertices of the active boundary that are stored at the mesh buffer can be imaged. Fig. 11 shows the active boundary structure maps to the mesh buffer. The next vertex of the active gate is located at the bottom of mesh buffer, and the previous vertex of the active gate is located at the top of the mesh buffer. Moreover, this study uses the commands “TPOP” and “BPOP” to access the mesh buffer respectively, and “PUSH” command to store the vertex in the top of mesh buffer. Consider Fig. 11 again, the triangle register contains three vertices which compress one triangle. In this figure, the side V1V2 is the active gate and while reading the third vertex, the triangle is encoded/decoded.

Regarding how to exchange the three vertices between the triangle register and mesh buffer, the two vertices of the next active gate must be moved into the triangle register and the third vertex must be read. The vertex replacement strategy of operators Q, C, R, L, J, is described below.
Fig. 11. The active boundary structure maps to the mesh buffer boundary.

Fig. 12. The replacement strategy of the triangle registers.
In Fig. 12, the replacement strategy of triangle register is presented, where $V_1$ and $V_2$ are vertices of the active gate.

1. Use the new vertex as the third vertex to produce a triangle while accessing "C," PUSH $V_1$ onto mesh buffer, move the third vertex to $V_1$, then constructs the next active gate with $V_2$.

2. Using the BPOP produces a vertex as the third vertex to form a triangle while accessing "R," move the third vertex to $V_3$, then construct the next active gate with $V_1$.

3. Using the TPOP obtains a vertex as the third vertex to form a triangle while accessing "L," move the third vertex to $V_1$, then construct the next active gate with $V_2$.

4. To PUSH $V_1$ onto mesh buffer, then move $V_2$ to $V_1$ and use BPOP to produce a vertex to $V_2$ while accessing “J,” the first two vertices are the vertices of the new active gate.

5. When accessing Q, by using cases C then R.

### 4.2 Vertex Replacement Strategy for Mesh Buffer

Vertices on the active boundary were all accessed previously and will be needed in the future. Consequently, these vertices will be included in the mesh buffer. However, if the vertices exceed the capacity of mesh buffer, some vertices must be removed.

Three sorts of vertices exist. As shown in Fig. 13, for the first type of vertex both connected sides are marked “J.” Meanwhile, for the second type one of the sides is marked “J.” Furthermore, for the third type neither of the connected sides is marked “J.”

While vertices exceed the capacity of the mesh buffer, the removal priority is first type, followed by second type and finally third type. Because the vertices of the first type are frequently skipped without drawing triangles, they are forsaken first. Besides, since sides marked with “J” can stretch and expand, the another side has more possibility to be marked as “J.” So it is discarded secondly.

![Three type vertex](image)

Fig. 13. Three type vertex.

Generally, when there are multiple vertices in each type, the further removal priority of each type is defined as:

1. Z: the vertex has been read and will not be read again.
2. Y: the vertex has been read and will be read again.
3. X: the new vertex.
These three vertices basically are processed in the order $Z$, $Y$, $X$. Because the $Z$ sort can release the space from the mesh buffer, if the mesh buffer is full and the process needs to access the new vertex, then one movement can be spared in emptying the mesh buffer. Decreasing the frequency of emptying the mesh buffer can repeatedly reduce the frequency of uploading the vertices and can also reduce the frequency of not finding in the mesh buffer. Thus, $Z$ sort is removed first. Moreover, the $X$ sort has lower priority because it takes up more space in mesh buffer. If the mesh buffer is full and the $X$ sort is discarded, then reloading can be performed.

Table 3. Compression results (in bit per triangle).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>#T</th>
<th>JR b/t</th>
<th>Our b/t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kidney</td>
<td>344</td>
<td>1.70</td>
<td>1.54</td>
</tr>
<tr>
<td>Note2</td>
<td>392</td>
<td>1.66</td>
<td>1.23</td>
</tr>
<tr>
<td>Mushroom 2</td>
<td>448</td>
<td>1.59</td>
<td>0.67</td>
</tr>
<tr>
<td>Stomach</td>
<td>534</td>
<td>1.64</td>
<td>1.55</td>
</tr>
<tr>
<td>Head</td>
<td>568</td>
<td>1.81</td>
<td>1.76</td>
</tr>
<tr>
<td>Opt-king</td>
<td>624</td>
<td>1.58</td>
<td>0.86</td>
</tr>
<tr>
<td>Winegobl</td>
<td>1000</td>
<td>1.56</td>
<td>0.71</td>
</tr>
<tr>
<td>Goldpan 2</td>
<td>1148</td>
<td>1.59</td>
<td>0.82</td>
</tr>
<tr>
<td>Taiwan</td>
<td>1354</td>
<td>1.79</td>
<td>1.84</td>
</tr>
<tr>
<td>Surfbd</td>
<td>1388</td>
<td>1.68</td>
<td>1.76</td>
</tr>
<tr>
<td>Screwfrw</td>
<td>1792</td>
<td>1.77</td>
<td>2.23</td>
</tr>
<tr>
<td>Cin-chr</td>
<td>2000</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>Opt-pump</td>
<td>2268</td>
<td>1.54</td>
<td>0.66</td>
</tr>
<tr>
<td>Blanket</td>
<td>2594</td>
<td>1.55</td>
<td>0.75</td>
</tr>
<tr>
<td>P51mustg</td>
<td>2992</td>
<td>1.68</td>
<td>1.65</td>
</tr>
<tr>
<td>Tennis_shoe</td>
<td>3634</td>
<td>1.66</td>
<td>1.21</td>
</tr>
<tr>
<td>83_camaro</td>
<td>3640</td>
<td>1.63</td>
<td>1.10</td>
</tr>
<tr>
<td>Skyscrpr</td>
<td>3692</td>
<td>1.79</td>
<td>1.68</td>
</tr>
<tr>
<td>V006</td>
<td>3732</td>
<td>1.64</td>
<td>1.12</td>
</tr>
<tr>
<td>Footbone</td>
<td>4204</td>
<td>1.64</td>
<td>1.05</td>
</tr>
<tr>
<td>Shark</td>
<td>5116</td>
<td>1.64</td>
<td>1.43</td>
</tr>
<tr>
<td>Triceratops</td>
<td>5660</td>
<td>1.69</td>
<td>1.61</td>
</tr>
<tr>
<td>Cow</td>
<td>5804</td>
<td>1.65</td>
<td>1.51</td>
</tr>
<tr>
<td>Chair16</td>
<td>6128</td>
<td>1.60</td>
<td>0.97</td>
</tr>
<tr>
<td>Octopus</td>
<td>8406</td>
<td>1.76</td>
<td>1.97</td>
</tr>
<tr>
<td>Easter4</td>
<td>8956</td>
<td>1.63</td>
<td>1.05</td>
</tr>
<tr>
<td>Handhires</td>
<td>9558</td>
<td>1.71</td>
<td>1.60</td>
</tr>
<tr>
<td>Chicken</td>
<td>20928</td>
<td>1.63</td>
<td>1.14</td>
</tr>
<tr>
<td>Lollypop</td>
<td>31340</td>
<td>1.52</td>
<td>0.58</td>
</tr>
<tr>
<td>Horse</td>
<td>39098</td>
<td>1.70</td>
<td>1.63</td>
</tr>
<tr>
<td>V156</td>
<td>49172</td>
<td>1.60</td>
<td>0.76</td>
</tr>
<tr>
<td>Head01</td>
<td>75616</td>
<td>1.63</td>
<td>0.98</td>
</tr>
<tr>
<td>Manta</td>
<td>110000</td>
<td>1.51</td>
<td>0.53</td>
</tr>
<tr>
<td>Average1</td>
<td>12726</td>
<td>1.65</td>
<td>0.95</td>
</tr>
<tr>
<td>Average2</td>
<td>4625</td>
<td>1.62</td>
<td>0.98</td>
</tr>
</tbody>
</table>

$#T$: the number of triangles
Average1: test models in this table
Average2: total 260 test models

Table 4. Transmission ratio (in read times per vertex).

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Vertex</th>
<th>Read</th>
<th>Transmission/Vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kidney</td>
<td>176</td>
<td>176</td>
<td>1</td>
</tr>
<tr>
<td>Note2</td>
<td>202</td>
<td>202</td>
<td>1</td>
</tr>
<tr>
<td>Mushroom 2</td>
<td>226</td>
<td>226</td>
<td>1</td>
</tr>
<tr>
<td>Stomach</td>
<td>269</td>
<td>269</td>
<td>1</td>
</tr>
<tr>
<td>Head</td>
<td>288</td>
<td>288</td>
<td>1</td>
</tr>
<tr>
<td>Opt-king</td>
<td>314</td>
<td>314</td>
<td>1</td>
</tr>
<tr>
<td>Winegobl</td>
<td>502</td>
<td>502</td>
<td>1</td>
</tr>
<tr>
<td>Goldpan 2</td>
<td>576</td>
<td>576</td>
<td>1</td>
</tr>
<tr>
<td>Taiwan</td>
<td>679</td>
<td>679</td>
<td>1</td>
</tr>
<tr>
<td>Surfbd</td>
<td>696</td>
<td>696</td>
<td>1</td>
</tr>
<tr>
<td>Screwdrw</td>
<td>904</td>
<td>904</td>
<td>1</td>
</tr>
<tr>
<td>Cin-chr</td>
<td>1028</td>
<td>1028</td>
<td>1</td>
</tr>
<tr>
<td>Opt-pump</td>
<td>1136</td>
<td>1136</td>
<td>1</td>
</tr>
<tr>
<td>Blanket</td>
<td>1299</td>
<td>1299</td>
<td>1</td>
</tr>
<tr>
<td>P51mustg</td>
<td>1508</td>
<td>1508</td>
<td>1</td>
</tr>
<tr>
<td>Tennis_shoe</td>
<td>1841</td>
<td>1841</td>
<td>1</td>
</tr>
<tr>
<td>83_camaro</td>
<td>1830</td>
<td>1830</td>
<td>1</td>
</tr>
<tr>
<td>Skyscrpr</td>
<td>2022</td>
<td>2022</td>
<td>1</td>
</tr>
<tr>
<td>V006</td>
<td>1888</td>
<td>1888</td>
<td>1</td>
</tr>
<tr>
<td>Footbone</td>
<td>2154</td>
<td>2154</td>
<td>1</td>
</tr>
<tr>
<td>Shark</td>
<td>2560</td>
<td>2618</td>
<td>1.03</td>
</tr>
<tr>
<td>Triceratops</td>
<td>2832</td>
<td>2871</td>
<td>1.01</td>
</tr>
<tr>
<td>Cow</td>
<td>2904</td>
<td>2976</td>
<td>1.02</td>
</tr>
<tr>
<td>Chair16</td>
<td>3082</td>
<td>3082</td>
<td>1</td>
</tr>
<tr>
<td>Octopus</td>
<td>4242</td>
<td>4655</td>
<td>1.1</td>
</tr>
<tr>
<td>Easter4</td>
<td>4508</td>
<td>4508</td>
<td>1</td>
</tr>
<tr>
<td>Handhires</td>
<td>4781</td>
<td>6021</td>
<td>1.26</td>
</tr>
<tr>
<td>V037</td>
<td>5212</td>
<td>5212</td>
<td>1</td>
</tr>
<tr>
<td>V007</td>
<td>9648</td>
<td>9648</td>
<td>1</td>
</tr>
<tr>
<td>Chicken</td>
<td>10468</td>
<td>10731</td>
<td>1.03</td>
</tr>
<tr>
<td>Lollypop</td>
<td>15676</td>
<td>15676</td>
<td>1</td>
</tr>
<tr>
<td>Horse</td>
<td>19851</td>
<td>22994</td>
<td>1.16</td>
</tr>
<tr>
<td>V156</td>
<td>25022</td>
<td>25022</td>
<td>1</td>
</tr>
<tr>
<td>Head01</td>
<td>37822</td>
<td>37822</td>
<td>1</td>
</tr>
<tr>
<td>Manta</td>
<td>55510</td>
<td>55510</td>
<td>1</td>
</tr>
<tr>
<td>Average1</td>
<td>6390</td>
<td>6540</td>
<td>1.02</td>
</tr>
<tr>
<td>Average2</td>
<td>2350</td>
<td>2376</td>
<td>1.01</td>
</tr>
</tbody>
</table>
5. IMPLEMENTATION AND RESULT

This section compares the proposed algorithm with that of Rossignac by using the Huffman encoder to compress the operators. A total of 260 test models employ the Wavefront file format, and the compression ratio is 0.98 bits per triangle. Table 3 lists the experimental results of the Rossignac algorithm and the proposed improved algorithm. Moreover, Fig. 11 illustrates the partial 3D meshes used for the present measurements. Furthermore, this study obtains statistics on the efficacy of the mesh buffer mentioned. Based on a mesh buffer capacity of 256, if the length of the active boundary is less than 256 reloading the vertices are unnecessary. Table 4 lists the transmission ratio of the proposed algorithm.

Clearly, the proposed algorithm lowers the compression ratio from 1.62 bits per triangle to 0.98 bits per triangle, while the transmission ratio is to 1.01 times per vertex. Even though no Q operator is used (conduce to CR operators), the compression ratio remains 1.27 bits per triangle. Besides, another compression result using the operators “QCRLSE” is 1.05 bits per triangle. Employing the “Q” operator performs better than the other operators. The “J” operator even though spends more cost than that of operator “SE,” the proposed algorithm can get faster run time and make sure real time in the encoding/decoding processes.

The compression time of the JR algorithm and the proposed algorithm is almost the same (JR is 72.88 sec and the proposed algorithm is 71.18 sec for all 260 test models). But adding the decompression time, the proposed algorithm spends 142.36 sec and JR spends 218.64 sec. According to JR algorithm, the decompression process must two pass or reverse decompression. Consequently, the total compression/decompression time will much more than the proposed algorithm.

The spread condition of the proposed compression ratio: 5% has best result (near 0.5 bits per triangle) and 3% has worst result (more than 2 bits per triangle) respectively. The best result of the compression ratio such as “manta” is 0.5 bits per triangle and the mean is these operators almost “Q.” The worst result of the compression ratio is more than 2.0 bits per triangle such as “screwfrw.” This kind of 3D model with worst result has a long and narrow band, and more “J” operators are required.

It goes without saying; the initial position will affect the compression ratio. In our experiences, there are little differences of compression ratios for most 3D models by selecting different initial active boundary triangle. And the compression ratio will get bigger influence for a few particularly 3D models. Although the initial active boundary is different for same 3D model, the output operators “QCRLJ” just have little change. The reason is that the topology of 3D model have already fixed. For example, the compression ratios of the “screwdrv” model are 2.23, 2.2, and 2.27 bits per triangle with various initial positions which are shown in Fig. 14 and indicated with three different yellow points.

Compared to previous research, this study has reduced access from multiple pass traversals and replaced it with direct access to a single pass traversal. Moreover, for a limited mesh buffer of 256 units, the proposed algorithm displays low reloading rate.
6. CONCLUSIONS AND FUTURE WORKS

This study presents a novel loss-less, single-resolution connectivity scheme for 3D triangular mesh with holes and handles. The proposed algorithm belongs to loss-less connectivity compression. Therefore, the connectivity information of 3D models will not be lost after the compression/decompression. Moreover, the proposed algorithm applied an edge-based compression strategy; the study requires neither splitting model topology nor multiple passes to obtain a fruitful compression rate. Additionally, the proposed algorithm does not need to store offsets and not require additional actions to calculate offsets. Moreover, the proposed algorithm does not need inverse decoding. Furthermore, this study proposed graphics engine architecture to compress/decompress triangular mesh, and also investigated the vertex replacement strategy for triangle registers and mesh buffer. The proposed method achieves the real goal, namely that the meshes are immediately decompressed whenever the receiver receives partial operators. The model testing demonstrates that an excellent compression rate can be obtained. Furthermore, the average compression ratio with the proposed method is better than that with the Edgebreaker method.

In the testing of certain models, some models are observed to have more than 2.0 bits per triangle (screwfrw model as shown in Table 3 and Fig. 14). Because the ratio of
branch components to the whole model is too big, more $J$ operators are required consequently. Future research must consider how to reduce $J$ operators and try to find a suitable initial position which is special fitting the proposed algorithm. The experiments performed here only use the Huffman encoder to compress $CQRLJ$ string and to achieve increased compression rate. Furthermore experiments will attempt to use an adaptive arithmetic encoder. Both the vertex- and edge-based algorithms share split operations that raise some bottlenecks for spending increased overheads for $S$ and $E$, or split and offset. To avoid $S$ and $E$ operators, this study proposed operator $J$, but in some cases, the number of $J$ is unpredictable. This study adds some remarks after operator $J$ to get the third vertex on the active boundary rapidly, and reduce overheads.

REFERENCES


**Bin-Shyan Jong** (鍾斌賢) is an Associate Professor in the Department of Information and Computer Engineering, Chung Yuan Christian University. He received the B.S. degree in Computer Science from Chung Yuan Christian University in 1978 and M.S./Ph.D. degrees from Institute of Computer Science in 1983 1989. During 1980-1985, he was a project manager at CAPITAL Computer Co., Taipei, Taiwan. His research interests include computer graphics, virtual reality, and computer aided education.

**Wen-Hao Yang** (楊文灝) received the B.S. degree in Electronic Engineering from St John’s and St Mary’s Institute of Technology in 1989. He received the M.S. degree in Electrical Engineering from Chung Hua University, Hsinchu, Taiwan. He is currently pursuing the Ph.D. degree in the Department of Electronic Engineering, Chung Yuan Christian University. He joined the Department of Electronic Engineering, Chin Min Institute of Technology, Miaoli, Taiwan, in 1996. He was the chair of the Department of Computer Center in 1998-2000. His research interests include computer graphics, image retrieval, algorithms, and Virtual Reality.

**Shawn Song** (宋廉祥) he received his B.S. and M.S. degrees in the Department of Information and Computer Engineering from Chung Yuan Christian University, Taiwan, R.O.C., in 1998 and 2000, respectively. Now he is a R&D division engineer at the VIA technologies Inc. His work is about graphics card.