2003

Compression and decompression of quadrilateral meshes.

Quanbin. Jing
University of Windsor

Compression and Decompression of Quadrilateral meshes

By

Quanbin Jing

A Thesis

Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of the Master of Science at the
University of Windsor

Windsor, Ontario, Canada

Your file Votre référence
ISBN: 0-612-84594-X
Our file Notre référence
ISBN: 0-612-84594-X

Abstract

3D Quad mesh plays an important role in various engineering fields. Due to improved design and model acquisition tools, as well as the need for higher accuracy, the number and complexity of these models are growing more rapidly than network bandwidth. Therefore, reducing the amount of transmission by compressing the 3D quad model is imperative. A mesh may be represented by its vertex data and its connectivity. Vertex data comprise coordinates of all the vertices and optionally the vertex colors and the associated normal vectors and textures. Connectivity captures the incidence relation between the quads of the mesh and their bounding vertices.

Traditionally, the quad mesh connectivity encoding process involves triangulation and triangle mesh compression; this may introduce additional cost. Quad mesh can be compressed and decompressed linearly without triangulation. We introduce it in terms of a simple data structure, which we call the OE Table. It represents the connectivity of any manifold quad mesh as two tables, V and OE. V[i] is an integer reference to a vertex. OE[i] is an integer reference to an edge. Spirale Reversi decompression of quad mesh will be described in detail. It is possible to combine vertex data compression techniques with the connectivity compression. A lower upper bound 2.67 bits/quad for coding quad mesh connectivity is presented.

Keyword: quadrilateral mesh, compression/decompression, connectivity encoding

Dedication

This thesis is dedicated to my wife, Lijun He, and my son, Michael Jing, for their love, understanding and support.

Acknowledgements

First of all, I would like to thank Dr. Asish Mukopadhyay for his dedicated instruction and advice on this thesis. My many valuable discussions with him and his excellent suggestions have not only significantly contributed to the quality of this thesis, but have also benefited my future endeavors. Dr. Mukopadhyay's excellent supervising ability in the research field impressed me very much. Without his supervision and support, this thesis would not have been possible.

My appreciation also goes to committee members, Dr. Boubakeur Boufama, Dr. Yash Paul Aneja, and Dr. Xiaobu Yuan for having the patience to read drafts and for their attentive, invaluable comments, suggestions and relevant feedback.

Special thanks to Mr. Glen McCluskey of Glen McCluskey & Associates LLC, for his prompt response to my questions about bits manipulation.

Table of Contents

List of figures

List of Tables

Chapter 1

Introduction

This chapter presents the motivation for this thesis. First, the concepts of data compression and geometry compression are introduced. Then, concepts of connectivity and vertex compression are illustrated. Following this, the objectives of this thesis are introduced. Finally, the layout for the rest of this thesis is described.

1.1 Concepts of Data Compression

Data compression is often referred to as coding, where coding is a general term encompassing any special representation of data that satisfies a given need. Information theory studies efficient coding and its consequences in the form of speed of transmission and probability of error [31].

Numerous techniques exist for compressing the binary data used by digital computers and communication devices. The binary system represents each alphanumeric character with a string of eight binary digits (bits), each of which is either a 0 or 1, and which together form a byte. A rudimentary data-compression system is based on key word encoding, whereby frequently occurring words such as "the" are converted into a two-byte token. More advanced techniques analyze, identify, and then replace commonly occurring text

patterns with single characters and symbols. These techniques may also represent characters and symbols with strings of fewer than eight bits, with the characters used most often represented by the least number of bits. A requirement for successfully decoding schemes in using strings of variable lengths is that the bits designating the ends of characters must be unambiguously identifiable. Huffman encoding is a widely used form of this technique. Run-length encoding is used for data containing repetitive characters; it stores the repeated string once and indicates the number of occurrences [50, 61, 94, 95, 97] .

The principal benefits of data compression are: larger storage capacity; more efficient transmission of information over the Internet; and encryption, or disguising the meaning of information. A trade-off between time and speed characterizes most advanced methods of data compression. Typically, the more time a compression program (instruction set) is allowed to analyze data, the greater will be the compression, although the rule is subject to diminishing marginal returns [99].

1.2 Geometry Compression

In the mesh compression literature, a distinction is often made between three things: *mesh geometry*, which includes vertex coordinates; *mesh properties* such as normals, colors, and texture coordinates that are attached to vertices and faces; and *mesh connectivity*, which describes the incidences between vertices, edges, and faces. The mesh connectivity information is also referred to as mesh topology. Geometry and property data can be

efficiently compressed with schemes that predict a position or a feature from previously decoded neighbors.

Computer graphics applications are using more and more complex geometric models, containing millions or billions of triangles. It is imperative that good compression schemes be found to compress the polygonal models in order to reduce the storage space since these geometry data do not yield to traditional data compression techniques.

In addition, with the rapid development of the Internet, more and more graphics applications are running across the Internet, such as distributed collaborated modeling and multi-user games. It is essential for such distributed applications to be able to transmit geometry efficiently, which in turn require good geometry compression schemes.

Furthermore, the speed gap between CPU and memory is becoming even larger. Interactive animation of geometry models requires a large number of triangles to be efficiently loaded into the on-chip cache across the bus. The current bottleneck is the memory bandwidth, necessitating good geometry compression algorithms to reduce the traffic to the slow memory [100].

Constraints on bandwidth, storage space, and rendering time often limit the use of 3D models and the geometric data sets in e-commence, entertainment, CAD/CAM, medical, and visualization applications. The goal of 3D graphics compression is to address this problem by developing more compact representations and more effective quality-of-

service algorithms for multidimensional datasets. Efforts to achieve this goal have used both general purpose compression techniques (such as entropy coding and wavelet compression) and graphics-specific methods based on solid modeling, differential geometry, graph theory, and image-based rendering.

1.3 Connectivity Compression

A mesh may be represented by its vertex data and by its connectivity. Vertex data comprises coordinates of the associated normal vectors and textures. In its simplest form, connectivity captures the incidence relation between the triangles (or other polygons) of the mesh and their bounding vertices. It can be represented by a polygon-vertex incidence table, which associates with each polygon the references to its bounding vertices. Since triangle meshes have been extensively studied and other polygon meshes can be easily triangulated, Chapter 2 focuses more on triangle mesh. For all meshes homeomorphic to a sphere, and in fact for most meshes in practice, the number of triangles is roughly twice the number of vertices [70]. Consequently, when pointers for integer indices are used as vertex references and when floating pointer coordinates are used to encode vertex locations, connectivity data consumes twice more storage than are needed to store vertex coordinates.

Therefore, many computer scientists are working in this area to determine techniques for hiding some or all of the connectivity cost in the vertex encoding, i.e. connectivity

compression. They have defined three different objectives: efficient rendering, progressive transmission, and maximum compression.

1.4 Vertex Compression

Vertex coordinates may be compressed through various forms of vector quantization. Most approaches use predictors to encode corrections instead of absolute vertex data. Both the encoder and the decoder use the same prediction formula. The encoder transmits the difference between the predicted and the correct vertex data.

1.5 Objectives of the Thesis

The following paragraphs give the motives of this study:

Direct Compression

Geometry compression of triangle meshes has been extensively studied. Quadrilateral meshes (in this thesis, the terms "quadrilateral mesh" and "quad mesh" are used synonymously) are usually triangulated first and then compressed using techniques of triangle mesh compression. This may introduce additional cost. We explore an approach to compressing quad meshes without prior triangulation.

Linear Time Compression

Since triangle mesh connectivity compression can be realized in linear time using corner-table data structure, we extend the corner table data structure to quad meshes, with which we can compress quad meshes connectivity in linear time.

Spirale Reversi Decompression

Although Spirale Reversi Decompression of quad mesh connectivity has been mentioned in Kronrod and Gotsman's paper [48], no detailed description has been provided. We also extend Isenburg and Snoeyink's Spirale Reversi Decompression of triangle meshes [] to quad meshes.

Upper Bound on Quad mesh Connectivity Compression

Kronrod and Gotsman's [48] direct compression of quad meshes yields 3.5 bits per vertex for connectivity data. However, this is not as good as the 2.67 bits per vertex result of King et al [], which first triangulates a quad mesh. We propose a new coding scheme to improve the upper bound of Kronrod and Gotsman's method.

1.6 Overview of the thesis

This thesis is organized into six chapters as follows.

Chapter 1 contains a brief introduction to data compression, geometry compression, connectivity compression, and vertex compression, outlines the objectives of the thesis, and provides an overview of the thesis.

Chapter 2 contains a thorough literature review of geometry compression. Both connectivity and vertex compression are discussed. Triangle mesh and quad mesh connectivity compression are highlighted.

Chapter 3 first gives a detailed description of the Opposite-edge table structure with which linear time compression algorithm of quad mesh connectivity information is presented. Then a detailed description of the Spirale Reversi Decompression of quad meshes is presented. Then examples are given to illustrate the compression and decompression algorithm. Finally, the time complexity is analyzed.

Chapter 4 gives an encoding technique that provides an improved upper bound on quad mesh connectivity compression.

Chapter 5 gives implementation details and experimental results.

Finally, Chapter 6 offers some concluding remarks and discusses future improvements for the coding schemes and other work.

Acknowledgement and Reference are presented thereafter. The Appendix section contains major source codes.

Chapter 2

Literature Review

There are two major research trends in the area of geometry compression: one is focused on mesh connectivity (connectivity compression) and the other is focused on vertex data (vertex compression). This chapter focuses on connectivity compression and consists of seven sections. Theoretical results on succinct representation of graphs are introduced briefly in section 2.1. Based on triangle meshes, the three directions of connectivity compression research (Efficient Rendering, Progressive transformation, and maximum compression) are reviewed in sections 2.2, 2.3 and 2.4 respectively. Then quadrilateral mesh connectivity compression is reviewed in section 2.5. Vertex compression for general meshes is reviewed in section 2.6. Then the whole chapter is summarized in section 2.7.

2.1 Succinct representation of a graph

The theoretical problem of succinctly encoding planar graphs has been studied extensively in the graph theoretic literature [10, 27, 39, 40, 59, 60, 91, 92, 93]. Most of these theoretical results may be applied to any polygon mesh.

One of the nicest proofs of Euler's relation for planar graphs (V vertices, F faces and E edges) partitions the edges into two spanning trees [79]. One tree, spanning the vertices,

has V-1 edges and the other, spanning the faces (i.e. dual graph), has F-1 edges, so E=(V-1)+(F-1). Turan [91] observed that this partition into two spanning trees could be used to encode planar graphs. He gave an encoding that uses 12 bits per vertex (**bpv**) for unlabeled planar graphs. Keleer & Westbrook's [40] worst-case bound of 3 bits/edge for any polygon mesh is equivalent to 6 bits/vertex for a quad mesh and a value between 6 to 9 bits/vertex for any tri/quad mesh. For simple triangle mesh, Keleer's method guarantees 4.6 bpv. Chuang [10] achieves guaranteed encodings of less than 2.5+2log3 bits/vertex or 5.67 bits/vertex for quad and tri/quad meshes. He & Kao [27] achieves encoding of less than log3(V+E) bits, or 4.75 bits/vertex for quad meshes.

2.2 Efficient rendering

The standard representation for uncompressed polygon meshes uses a list of vertex coordinates to store geometry and a list of vertex indices for each face to store mesh connectivity. For triangle meshes of v vertices, this requires approximately $6v\log_2 v$ bits for the mesh connectivity. Note that this representation does not directly store face adjacency, which must be recovered by sorting around vertices if the mesh is to be checked for cracks or turned into triangle strips.

Encodings for rendering use partial information about mesh connectivity to reduce the work in the graphics pipeline. In the standard representation, each triangle of the mesh must be rendered individually by sending its three vertices (for triangle or triangulated meshes) to the graphics hardware. On average, every mesh vertex is processed six times. Processing a vertex involves passing its coordinates from the memory to and through the graphics pipeline. Typically, this also includes normal, color, and texture information. The most common technique to reduce the number of times this data needs to be transmitted is to send long runs of adjacent triangles. These are called triangle strips.

2.2.1 Triangle Strips

A triangle (t) can be specified by its three vertices (say, v_k, v_{k+1}, v_{k+2}). This gives a cost of 3n for n triangles, which may have overlapping vertices and thus implies extra cost. A cheaper way to transmit the triangles of a mesh is to order them so that the consecutive

triangles are adjacent. That is, triangles t (v_k, v_{k+1}, v_{k+2}) and t' =(v_{k+1}, v_{k+2}, v_{k+3}) are consecutive. A sequence of n+2 (n is the number of vertices in the mesh) vertices that determine n triangles in this way is called a triangle strip.

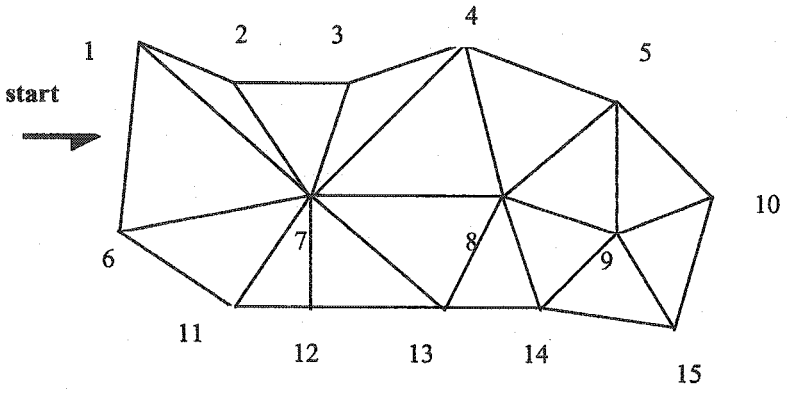
Brad Gramtham [] has a simple program to construct triangle strips from a given triangle mesh.Evans et al [18] proposed an algorithm to stripify any polygonal mesh model.

Triangle strips are widely supported by today's graphics hardware [18, 76, 96]. Two vertices from the previous triangle are reused for all but the first triangle of every strip. Two successive triangles in a triangle strip join at an edge. Therefore, from the second triangle on, the vertices of the previous triangle can be combined with only one new vertex to form the next triangle. As a triangle mesh has twice as many triangles as vertices, the maximal gain is that each vertex has to be transmitted only about two times. Depending on the quality of the strips, this can reduce the number of vertex repetitions by a factor of three.

2.2.2 Generalized triangle mesh

For some models, it is possible to get longer strips allowing a swap operation over the last two vertices in the sequence. A triangle strip with swap operations is known as a generalized triangle strip. A generalized triangle strip with a buffer that allows look back to the previous k vertices is called a generalized triangle mesh.

Such a data structure was first proposed by M. Deering based on his previous work [15, 16]. This approach is a compromise between a standard triangle strip and a general scheme for referencing any previously decoded vertex. He uses a 16-register cache to store temporarily 16 of the previous decoded vertices for subsequent use. He suggests using one bit per vertex to indicate whether a newly decoded vertex should be saved in the cache. Two bits per vertex are used to indicate how to form a triangle. One bit per triangle indicates whether the next vertex should be read from the input stream or retrieved from the cache. Four bits of address allow random access to a vertex in the stack-buffer every time an old vertex is used.



Generalized Triangle Strip: R6, O1, O7, O2, O3, M4, M8, O5, O9, O10, O15, M14, M8, O13, O7, O12, O11, M6
Generalized Triangle Mesh: R6p, O1, O7p, O2, O3, M4, M8p, O5, O9, O10, O15, M14, M-1, O13, O-2, O12, O11, M-3
Legend:
First Letter: R= Restart , O = Replace Oldest, M = Replace Middle;
Trailing "p" = push into mesh buffer, Number is vertex number, -number is mesh buffer reference where -1 is most recent pushed vertex

Figure 2-1. Generalized Triangle Mesh

Considering the geometry in Figure 2-1, while it can be represented by one triangle strip, many of the interior vertices appear twice or more in the strip. The generalized triangle mesh technique explicitly pushes old vertices into a queue and then referenced in the future. Deering's approach underlies Sun's Java 3D Geometry Compression API Specification [82].

Subsequently, Chow [] presented another algorithm to produce generalized triangle strips that improves on Deering's work. He modified the traversal of a generalized triangle mesh and removed the restriction of a 16–register cache [9].

Arkin et al. [2] examined the problem of testing whether a triangulation can be covered with a single triangle strip. For generalized triangle meshes this problem has been proved to be NP-complete, but for sequential strips there exists a simple linear time algorithm. But no results or algorithm were given to cover a mesh with several strips.

Bar-Yehuda and Gotsman [8] made an analysis of appropriate size of the buffer. They proved that a triangle mesh with n vertices can be optimally rendered, i.e. each vertex is transmitted once, if a buffer for $12.72\ n^{1/2}$ vertices is provided. They also show that this upper bound is tight and no algorithm can work with less than $1.649\ n^{1/2}$ buffered vertices. In their analysis, they neglect the time consumed by the referencing of buffered vertices, which make it hard to determine the suitability of the approach for connectivity compression. Again, the algorithms to compute the rendering sequences are not fast enough for on-line generation.

2.3 Progressive transmission

Encodings for progressive transmission use incremental refinements of mesh connectivity and geometry so that partial data already represents the entire mesh at a lower resolution. Hoppe's [28] Progressive Mesh Scheme encodes a mesh by collapsing edges one by one. Decoding starts with a small base mesh, and expands the collapsed edges in reverse order. The scheme is for storing and transmitting arbitrary triangle meshes and it is a lossless, continuous-resolution representation, and it addresses several practical problems in graphics: smooth geomorphing of level-of-detail approximations, progressive transmission, mesh compression, and selective refinement. Figure 2-2 is illustration of progressive mesh representation through edge collapse transformation.

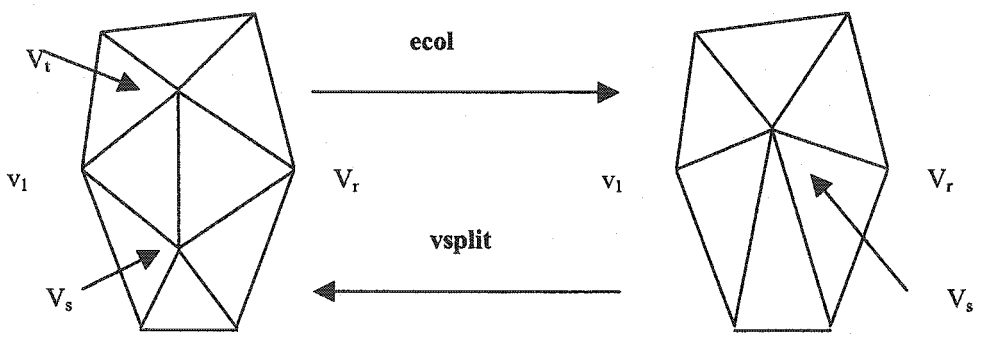


Figure 2-2: Illustration of the edge collapse transformation and its inverse, the vertex split.

While the first progressive schemes were not designed for compression and used a large number of bits per vertex, recent schemes [7, 11, 12, 63, 88] group the refinement operations into large batches and achieve bit-rates that come close to those of non-progressive methods. Even though more bits are used for the connectivity information, the progressive nature of the decoding allows more accurate geometry and property prediction.

Cohen-Or et al's scheme [11] is based on decimation of triangle meshes. For deletion of each vertex, the resulting hole is filled with a new triangulation. Hierarchies of the model are generated at various resolutions.

Khdakovsky et al [41] proposed a new progressive compression scheme for arbitrary topology, highly detailed, and densely sampled meshes arising from geometry scanning. They observe that meshes consist of three distinct components: geometry, parameter, and connectivity information. The latter two do not contribute to the reduction of errors in a compression setting. Using semi-regular meshes, parameter and connectivity information can be virtually eliminated. Coupled with semi-regular wavelet transforms, zerotree coding, and subdivision based reconstruction, they obtain error reduction by a factor four (12dB) compared to other progressive coding schemes. This method using wavelet compression can represent 3-D objects 12 times more efficiently than MPEG4 compression.

G. Taubin et al. [86] introduces a new adaptive refinement scheme for string and transmitting manifold triangular meshes in progressive and highly compressed form. It

achieves high compression rate-a forest split operation doubling the number n of triangles of a mesh requires a maximum of approximately 3.5v bits to represent the connectivity changes. The paper also shows how a surface simplification algorithm based on edge collapses can be modified to convert single resolution triangular meshes to the PFS format.

In P. Alliez and M. Desbrun's algorithm [1] a fine-grained fully progressive encoding is achieved for lossless transmission of 3d polygonal models.

For the special case of terrains based on Delaunay triangulations, Snoeyink and van Kreveld [77, 78] used ideas from Kirkpartrick's point location scheme [46] to encode all topology information in a permutation of the vertices, from which the mesh is progressively reconstructed. Denny and Sohler's work [17] extended this scheme to arbitrary planar triangulations. Although the cost of storing the topology is zero, the unstructured order in which the vertices are received and the absence of adjacency information during their decompression prohibits predictive geometry encoding. This makes these schemes overall more expensive. Moreover, it is not clear that it is possible to extend this idea to general surface meshes.

Hoppe[29], Rossignac and Borrel[71], Staadt et al.[81], Gumhold and Klei[22] studied multi-resolution compression and reconstruction. Garland and Heckbert [19] studied surface simplification through quadric error metric. Hadwiger presented mesh simplification and multi-resolution data structure in [26]. Hoppe et al. also presented their

mesh optimization result in [30]. Lengynel [51] and Rockwood [65] have presented some results on compression of time-dependent geometry.

2.4 Maximum Compression

Most schemes for maximum mesh compression encode mesh connectivity through a compact and often interwoven representation of two dual spanning trees. Neither the triangle nor vertex tree is sufficient by itself to capture the connectivity information. Typically, such compression schemes [23, 35, 54, 70, 88, 90] use a pair of spanning trees obtained by traversing the vertices and the triangles of the mesh with a deterministic strategy (e.g., breadth or depth first search). The geometry data and the property data of the mesh are usually compressed using predictive encoding based on local neighborhood information [88, 90]. As far as bit allocation is concerned, King et al. present optimal bit allocation for compressed 3D models [43].

2.4.1 Topological surgery

The idea of this method is to cut the mesh in a peeling-orange-skin style. It looks for edges of the model where to cut. The result of cutting through these edges is a triangulated simply connected polygon (in the case of a 2-manifold mesh). The complete structure of the mesh is stored in two spanning trees, a vertex tree and a triangle tree. This algorithm can encode edge information in 4 bits per vertex on the average with run-length encoding technique.

This approach was first proposed by Taubin and Rossignac [88] based on research results of [84, 85, 89]. Their scheme explicitly encodes both vertex spanning tree and face spanning tree. The topological surgery approach is proposed to encode VRML files [87].

Gueziec et al, [24] extends the earlier efforts of Topological Surgery (TS) to non-manifold models. This is done by developing an algorithm to decompose a non-manifold model into multiple manifold surfaces, which are then compressed using TS algorithm.

2.4.2 Edgebreaker

The Edgebreaker algorithm encodes each triangle at a time according to classification of its adjacent vertices and edges. This classification leads to one of five possible operations (C, L, E, R, S) codes, as illustrated in Figure 2-3. A sequence of labels (operation names) suffices to reconstruct the graph is found. For any mesh of T triangles that is homeomorphic to a sphere, the sequence may be trivially encoded using less than 2T bits. The decoding algorithm recreating the triangles in the same order they have been visited by the encoding algorithm has the worst case tome complexity of $O(n^2)$ [70].

At each step, Edgebreaker identifies the unique triangle X, that is part of boundary (B) and is incident upon gate g (see Figure 3). Let v be the only vertex of X that does not bound g. Edgebreaker analyze the relation that v has with respect to boundary and g, distinguishing 5 cases labeled C, L E, R, and S. The selection of appropriate case may be performed by the following sequence of tests:

IF v∉B THEN case C

ELSE IF v follows g

THEN IF v precedes g THEN case E

ELSE case R

ELSE IF v precedes g THEN case L

ELSE case S

Through the introduction of a novel decompression technique in Edgebreaker, which in one pass automatically extracts the offsets of S operations from the compression history, Rossignac has eliminated the need to encode these offsets explicitly and thus have achieved a linear connectivity cost for meshes with a constant of handles and holes. This result improves the approaches [23, 90], which exhibit an asymptotic O(vlogv) connectivity cost, even for simple meshes without holes or handles. On a more theoretical aspect, the author also improves on all previous work on coding planar triangulated graphs [39, 40, 60, 91, 92] by providing a linear code with the lowest constant: a guaranteed 2 bits per triangle or less.

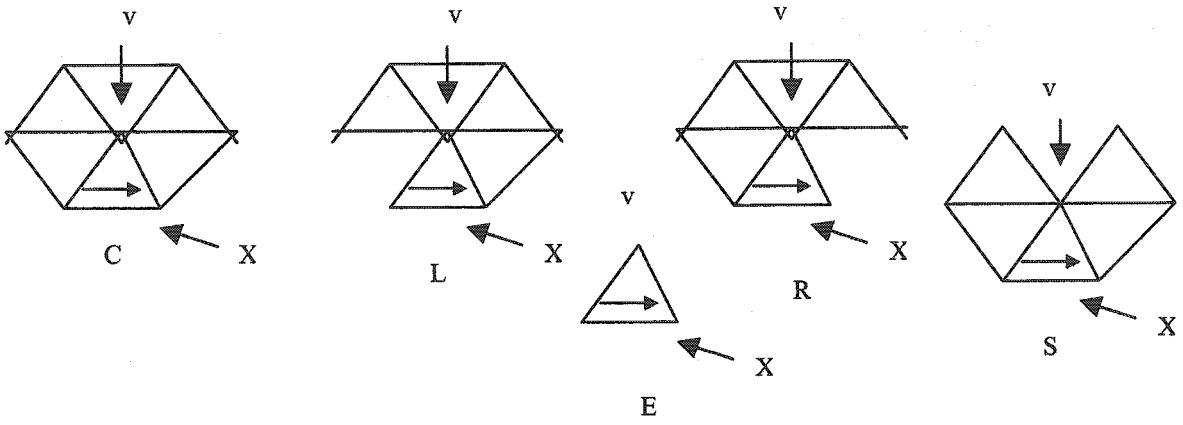


Figure 2-3 Edgebreaker process: Triangle X is formed by the gate edge g (represented by arrow) and vertex v. The location of v with respect to the boundary B determines the operation type: C (v is not on B), L (v immediately precede g), R (v immediately follows g), E (v precedes and follows g), S (v is elsewhere on B).

The edgebreaker scheme gives the best guaranteed bit-rates for triangle meshes connectivity. The S and E operations replaces the split operation used by the "cut border machine" [23], thereby eliminating the need for explicitly encoding the associated offset value. Improvements on the original paper give linear decoding time [74, 32, 33, 34, 36] and tighten the guaranteed bit-rate to 3.67 bpv [42]. This is currently the lowest worst-case bound and lies within13% of the theoretical lower limit by Tutte [92]. It may be easily combined with a variety of vertex data compression schemes based on vertex estimates that are derived from the incidence graph and from the location or previously decoded vertices.

It has been extended to manifold meshes with handles and holes [70], to triangulated boundaries of non-manifold solids [72], and to meshes that contain only quadrilaterals or a

combination of simply-connected faces with an arbitrary number of sides.[44]. It was also optimized for meshes with nearly regular connectivity [45].

Rossignac [73] introduce a new formulation, which leads to a simple implementation of Edgebreaker for compressing 3D triangle meshes. They describe it in terms of a simple data structure, the Corner Table, which represents the connectivity of any manifold triangle mesh as two table of integer. It compresses vertex locations using a parallelogram predictor.

Gumhold's approach is somewhat similar to Edgebreaker. His encoding algorithm has seven building operations. These operations fix the order in which the triangle mesh has to be traversed and how the mesh can be reconstructed [23].

Isenburg and Snoeyink presented a simple linear time algorithm for decoding triangle meshes in a single traversal [36].

2.4.3 Breadth-first search based connectivity compression

The connectivity of an orientable manifold triangle mesh is encoded in a lossless manner using three operation codes: add, split and merge. Assuming a fixed order to traverse the adjacent edges of each vertex, this set of operation codes reconstruct the topology of the encoded mesh. This algorithm can encode edge information in less than 1.5 bits per vertex on the average [90].

2.5 Quadrilateral-mesh compression

Although triangle meshes are most extensively studied and most frequently used, sometimes we do need to use polygonal meshes. For example, modeling structures, modeling cloth deformations and simulating torsion and crashes, we have to use quadrilateral meshes. Muller [58] summarized the various quadrilateral meshes generation methods in CAD. Bern and Espstein[3] present quadrilateral meshing method through circle Packing. Other polygon meshes are used much less in industry and they are still of important academic value.

Several authors have reported extensions to their schemes, which are used to compress connectivity of purely triangle meshes, in order to handle polygonal input. A naïve approach arbitrarily triangulates the polygon mesh and then uses one bit per edge to distinguish the original edges from those added during the triangulation process. Marking every edge can be avoided by triangulating the polygons systematically.

For the Topological Surgery method the extension to polygonal meshes first cuts the mesh along a vertex spanning tree and then triangulates the dual polygon spanning tree. Only the edges interior to the resulting triangle spanning tree need to be marked [87].

Similarly, King et al. describe how to let the Edgebreaker method guide the triangulation process. For simple polygon meshes without vertices of degree two they give an encoding that guarantees 5 bpv. [44] In fact, King et al. are the first to prove that quadrangular

meshes can be compressed more efficiently than their triangulated counterparts by avoiding the triangulation step. They give compact codings for pure quadrangles and for meshes containing mostly quadrangles and a few triangles.

The essence of King's approach is to split each quad into triangles according to a rule ensuring that two triangles created from each quad are adjacent in Edgebreaker's traversal sequence (a triangle spanning tree). As shown in Figure 4, each quad is split so that the second half of the quad becomes the right neighbor of the first half. Since the second half has not been previously visited, the first half can not be an R or E. This split rule is easy to implement, and it allows the original quads to be recovered by simply joining adjacent pairs of triangles during decompression. It leads to efficient encodings because it makes some combinations of the Edgebreaker labels impossible.

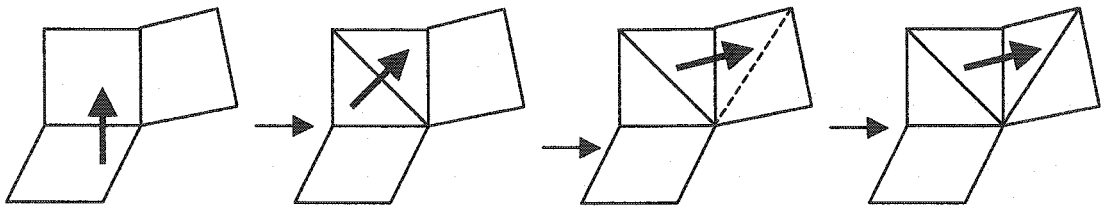


Figure 2-4 Quad split pattern

Li et al. [52, 53, 54] using dual graph approach traverse the edges of the dual and records a stream of symbols and integer values, which is compressed with a carefully designed context based entropy code. Decoding uses the recorded information to re-play this traversal, thereby reconstructing the mesh connectivity. Their use of split offsets resembles that of the "cut-border machine" [23].

Bajaj, Pascucci and Zhuang [5, 6, 7] propose an algorithm to split the mesh into triangle layers through a breath-first traversal. Their topological layering structure is inspired by the layering scheme that is used to construct vertex spanning trees for manifold meshes in TS. The topological layering structure based on vertex neighborhood is used to encode the connectivity information of arbitrary triangular meshes as well as to index and establish local neighborhood and the second order of predict or corrector geometry encoding scheme. The input meshes are partitioned into two basic layers: vertex layers and triangle layers. The 0^{th} vertex layer is a randomly chosen vertex (could be a chain of vertices) of the mesh. The kth vertex layer(k>0) includes a vertex V if V is not included in any previous vertex layer and there exists an edge E=(V,V*) where V* is included in the (k-1)th vertex layer. The kth triangle layer(k>=0) includes a triangle T if T has one vertex in kth vertex layer and T is not included in any previous triangle layer. Then the authors extend the layering scheme to encompass polygonal faces while maintaining its locality properties: the vertices of each polygon are included either in a single vertex layer or in two consecutive vertex layers. But the author did not analyze the worst case bits per vertex or show the experimental bits per vertex.

Inspired by Edgebreak[70], Isenburg and Snoeyink propose an edge-based compression scheme that encodes the connectivity of 2-manifold polygon meshes and extends to capture structural information as well. When this approach is applied to quadrilateral mesh, it guarantees 4v-8 bits encoding [38]. It encodes meshes directly in their polygonal representation and extends to capture face groupings in a natural way. Avoiding the triangulation step the author reduces the storage costs for topological polygon models that have group structures and property data. The benefits of this approach compared with methods that compress pre-triangulated polygon meshes are: (i) the original connectivity is preserved; (ii) Properties associated with faces and corners need not to be replicated; (iii) Subsequent stripification algorithms can generate better triangle strips.

In Issenburg and Snoeink's approach, [38] the connectivity of polygon mesh is encoded as a sequence of labels F_n, R, L, S, E, H_n, and $M_{i,k,l}$. The total number of labels equal the number of mesh edges. The sequence of labels represents an interwoven description of a polygon spanning tree and its complementary vertex spanning tree. For every face of n sides there is a label F_n and for every hole of size n there is a label H_n. Together they label the edges of the polygon spanning tree that need to be "fixed" together to re-create the handle. The remaining labels R, L, S, and E label the edges of the corresponding vertex spanning tree and describe how to "fix' faces and holes together. Subsequently an entropy coder compresses the label sequence into a bit-stream.

For polygon meshes with associated properties, Bossen [4], Issenburg and Snoeyink[37], Kobbelt and Seidel[47] has presented some results.

The Edgebreaker scheme assigns a code (C,L,R,E,S) to a triangle determined by its interaction with the rest of the mesh. Kronrod and Gotsman's [48] scheme generalizes this observation to meshes with polygonal elements. A polygonal element can interact with the rest of the mesh in a finite number of ways and the coding scheme uses as many code labels. In particular, for a quad mesh we need 13 different code labels.

The following two figures illustrate the interaction between a quad and boundary.

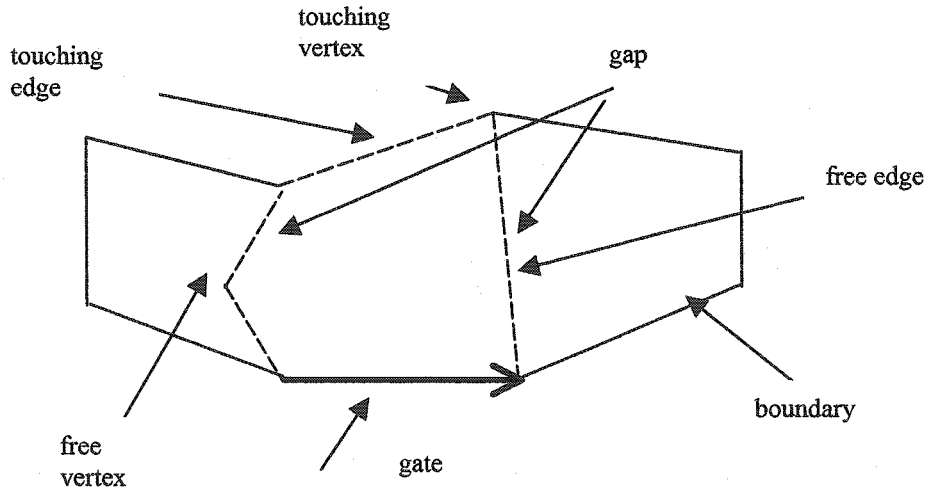
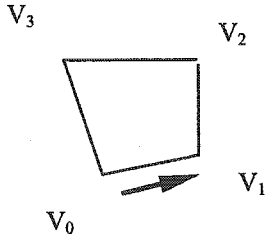
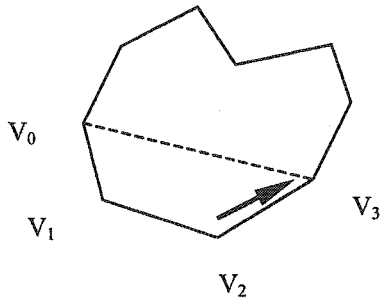


Figure 2-5. Terminology of a quad interacting with a mesh boundary

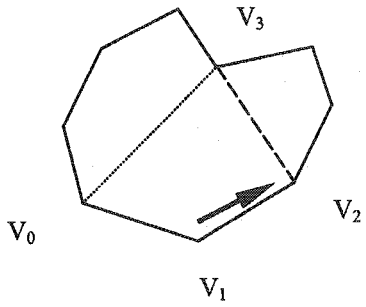
Note: Gate is an oriented edge on the boundary. Gap is an edge between two boundary vertices which are not adjacent on the boundary.



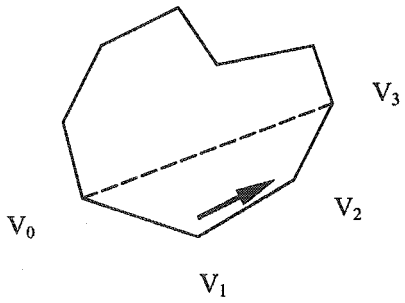
(1) Q1



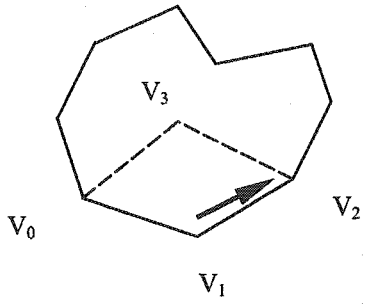
(2) Q2



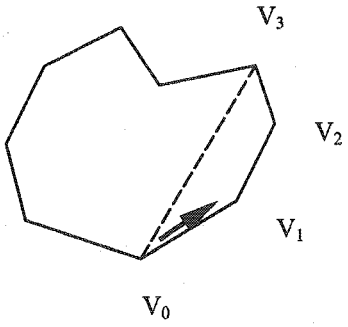
(3)Q3



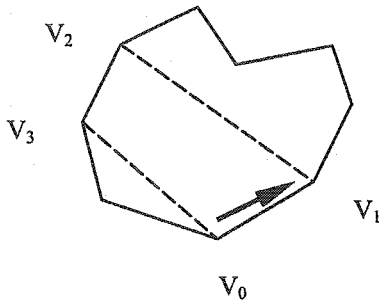
(4) Q4



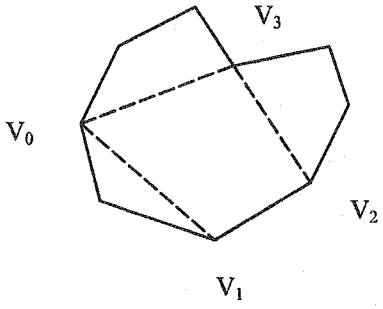
(5) Q5



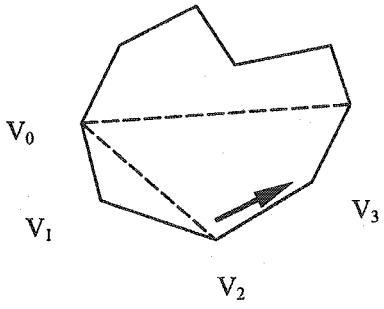
(6) Q6



(7) Q7



(8) Q8



(9) Q9

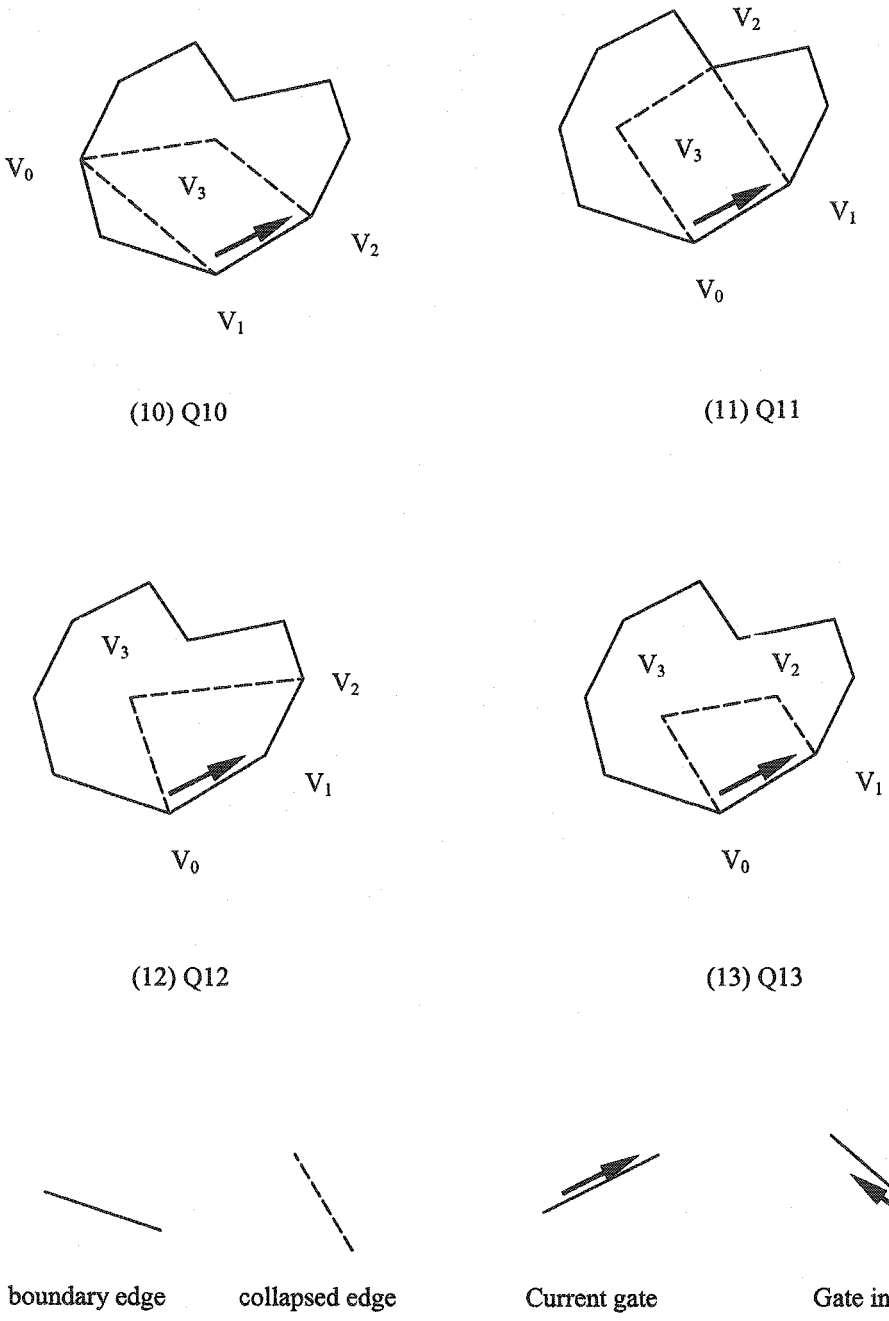


Figure 2-6. All possible interactions between a quad and a boundary. The thick edge is the gate. Case Q3, Q7, Q9, Q10, Q11 (having 2 gaps) generate a new boundary, and case Q8(having 3 gaps) generate two new boundaries.

2.6 Vertex Compression

Current 32-bits IEEE floating point number has excessive precision for most models. In practice, it is sufficient to represent each coordinates using 16 or 8 bits with a negligible loss of accuracy. In Deering's work [14], each coordinates of vertex is quantized to 16 bits. Then the result is converted into a sequence of difference vectors (x,y,z), which a integer differences between the previous traversed vertex and the next vertex. The difference vectors were encoded using Huffman code. This algorithm achieves less than 16 bits to encode each coordinate.

In the quantization process, first we need to select a quantization level such as n bits. Then we need to scan the vertex data to find the minimum and maximum values of each coordinate and save these values to a file (I call this file, bounding). Then, each coordinate of a vertex will be represented by an integer number between 0 and 2^n depending on how this coordinate partition between the minimum and maximum value. If it is the minimum value, it will be represented by 0 and if it is the maximum vale, it will be represented by 2^n.

Chow [9] improved Deering's algorithm above so that the quantization level is adaptively chosen depending on the local minuteness of the geometry.

Taubin and Rossignac[15] proposed an algorithm to encode the vertex positions based on the following linear predictor

$$P = \sum_{i=1}^{k} l_i v_i$$

where the v_i are previously traversed k vertices .

The difference between the real value and predicted value (ε, the correction vector) will be encoded. It achieves a compression of 12 bits per vertex on average.

Touma and Gotsman[90] proposed the more precise vertex prediction below,

$$P = v_i + v_{t+1} - v_{i+2} + \varepsilon$$

The modified ε preserves the curvature characteristics of the model. This scheme achieves a compression ratio of about 9 bits per vertex.

Lee and Ko's [101] compression algorithm also uses three previous traversed vertices to construct a local coordinate system, and utilizes the clustering behavior of the transformed result to achieve a compression of about 6.7 bits/vertex on average.

2.7 Summary

This chapter discusses the background required for this thesis. Theoretical results of representing a graph are introduced. Then, three different research directions, namely efficient rendering, progressive transformation, and maximum compression are discussed with a focus on connectivity compression of triangle meshes. Then, research on quadrilateral meshes is reviewed. Finally, vertex compression is discussed.

Chapter 3

Linear time compression and decompression of Quadrilateral mesh

Efficient rendering of a mesh is a very important subarea of research within the broad research area of computational geometry. Although linear time compression and decompression algorithms are available for triangular meshes, there is no linear time compression and decompression algorithm available for quadrilateral meshes without prior triangulation. This chapter introduces a new data structure through which efficient compression of quadrilateral meshes is possible. Then, the compression algorithm based on OE data structure is discussed in section 3.2. Next, the extension of the efficient decompression algorithm for triangle meshes to quadrilateral meshes is discussed in section 3.3. Examples are given for meshes with boundary and without boundary. Finally, the time complexity of the compression and decompression process is analyzed, and this chapter is summarized.

3.1 Definition of Opposite Edge(OE) data structure

In this chapter we propose a simple data structure, called the opposite-edge table for representing the connectivity of a quad mesh. The connectivity compression/decompression algorithm of this paper can be easily combined with the vertex compression algorithms proposed in [14,15, 90, 101] to give a complete compression/decompression scheme for quad meshes.

Each quad in a quad mesh is represented by four integer references for the four vertices and four integer references for opposite-edges. The opposite-edge of an edge A1 in quad B refers to the edge that is next to next to A2 in quad C, where A2 is a common edge of quads B and C. If A2 is a boundary edge, we arbitrarily assign -1 in the opposite-edge field for A1. In other words, given an edge of a quad P, its opposite-edge is an edge that belongs to an adjacent quad Q and is the edge of Q that is opposite to the edge that it shares with P. Vertices, edges, and opposite edges are identified using positive integers.

Figure 3-1 illustrates the opposite-edge data structure. When the shared edge of quads Q_i and Q_j is represented by directed edges e_i and e_j, the opposite-edge of e_l, according to the definition above, is the edge opposite to e_j in Q_j; similarly, the opposite-edge of e_j is the edge opposite to e_i in Q_i. In terms of this nomenclature, in Figure 3-1 the opposite-edge of e_3 is e_5 and the opposite-edge of e_5 is e_5.

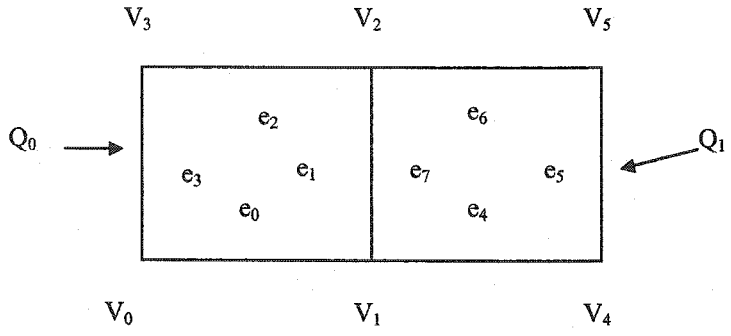


Figure 3-1. A mesh of two connected quad

For boundary edges, we arbitrarily assign -1 in their opposite-edge fields. Each quad of the opposite-edge table has the following structural representation:

v0 v1 v2 v3 oe0 oe1 oe2 oe3,

where the vi's are vertex references and the oei's are the opposite-edge references.

Therefore, the opposite-edge data structure of the mesh shown in Figure3-1 is as below:

$V_0\ V_1\ V_2\ V_3\ -1\ -1\ -1\ e_5$

$V_1\ V_4\ V_5\ V_2\ -1\ e_3\ -1\ -1$

(Note: The vertices and edges of a quad are enumerated in anti-clockwise order)

The coordinates of the mesh vertices are stored in an array G, each of whose entry has the following structure:

x1 y1 z1

which are respectively the coordinates of a 3D point, encoding the location of a vertex.

The boundary information will be saved in a file as a linked list of vertices in anti-clockwise order around the boundary called the BList. The BList of a mesh without boundary is empty.

3.2 OE Compression algorithm

The encoding algorithm given below is based on Kronrod and Gotsman's scheme [48]. The input for compression will be two files: a file for the vertex & opposite-edge tables, and a file containing the references of boundary vertices listed in anti-clockwise order. If the mesh is closed, then the file of boundary list is empty.

The output of the compression algorithm will be three files: a file of interaction types (Q1-Q13), a file called InnerVertex (which records the vertex references of the quad mesh during the compression process that are not on the boundary), and a file called BList. Actually, the InnerVertex file represents a permutation of all the inner vertices of the mesh that are important for reconstructing the mesh.

For a mesh without boundary, we need to choose any quad in the mesh to cut open and the four edges of the quad will act as a boundary. Since we cut the quad open, we are not going to assign any operation code for this quad. Therefore, the number of interaction types will be one less than the number of quad. The BList will contain these four vertices after compression.

Since our connectivity compression algorithm is independent of vertex data, we can apply any vertex compression algorithm on the vertex data.

The following is the pseudo code for OE compression:

```
Read boundary file to initialize BList;                    //step1
Initialize all quads by reading Vertex & Opposite-Edge Table; //step 2
If BList is empty, we add four vertices of any quad in the mesh
into it and mark this quad as visited;                     //step 3
Initialize an empty stack of gates, GateStack, and push onto it some
edge of the BList;                                         //step4
While(!GateStack.empty())                                  //step5
{
  ActiveGate=GateStack.pop();
```

Current processing Quad will be Quad[ActiveGate.OppositeEdge.eid/4];
Write the interaction type with BList onto output stream, Operation;
For every gap on Current processing Quad in clockwise order from ActiveGate, push the edge of the gap furthest from ActiveGate on GateStack;
Write the vertex location of every new vertex in each operation into output file InnerVertex;
Mark Current processing Quad as visited and update BList;
}

Program 1 OE compression process

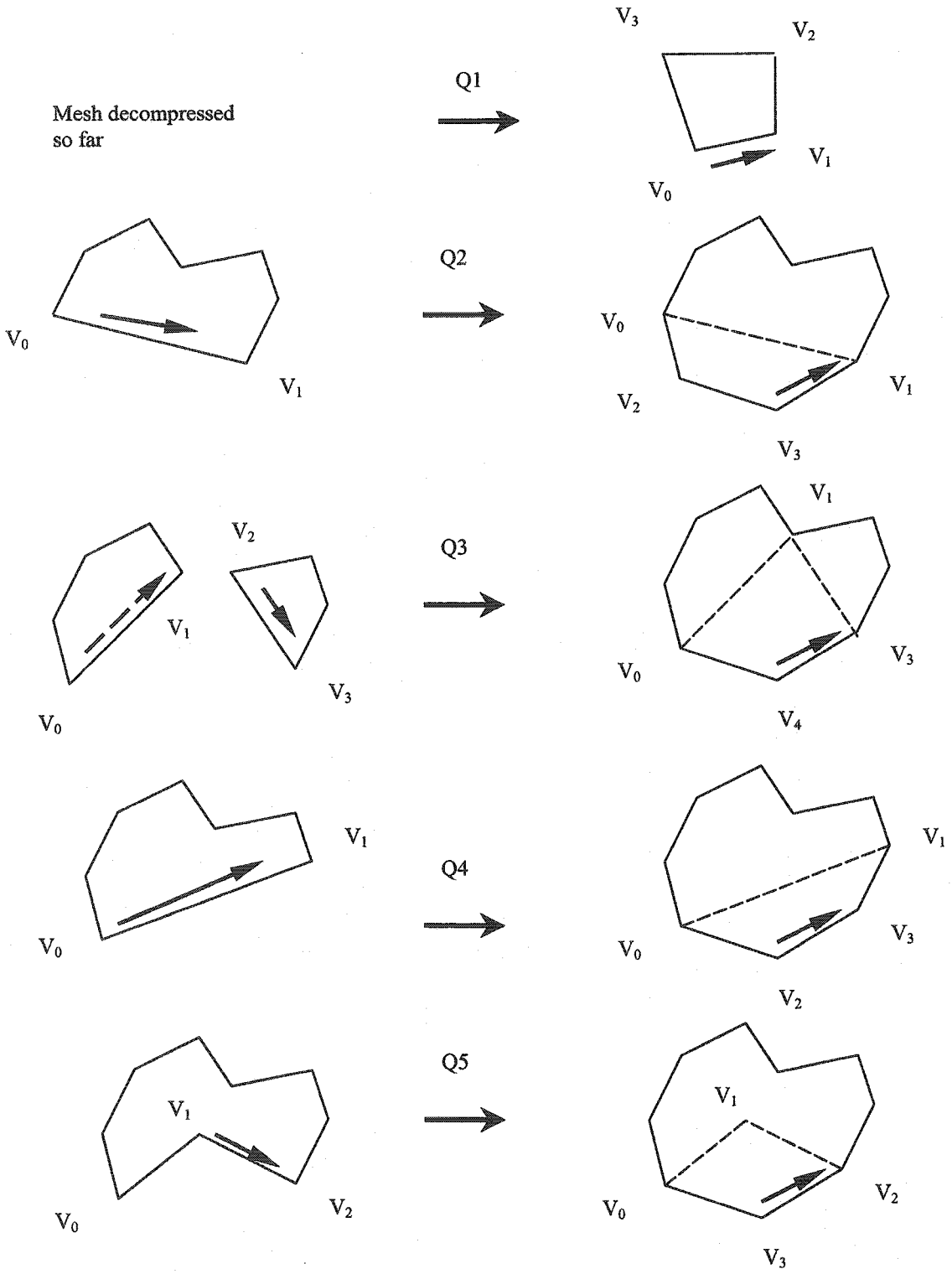
After the compression process, we get the ASKII code for the compression operation corresponding to each quad. We need to encode each operation to convert the ASKII code to binary code to finalize the compression process. The code optimization will be discussed in details in Chapter 4.

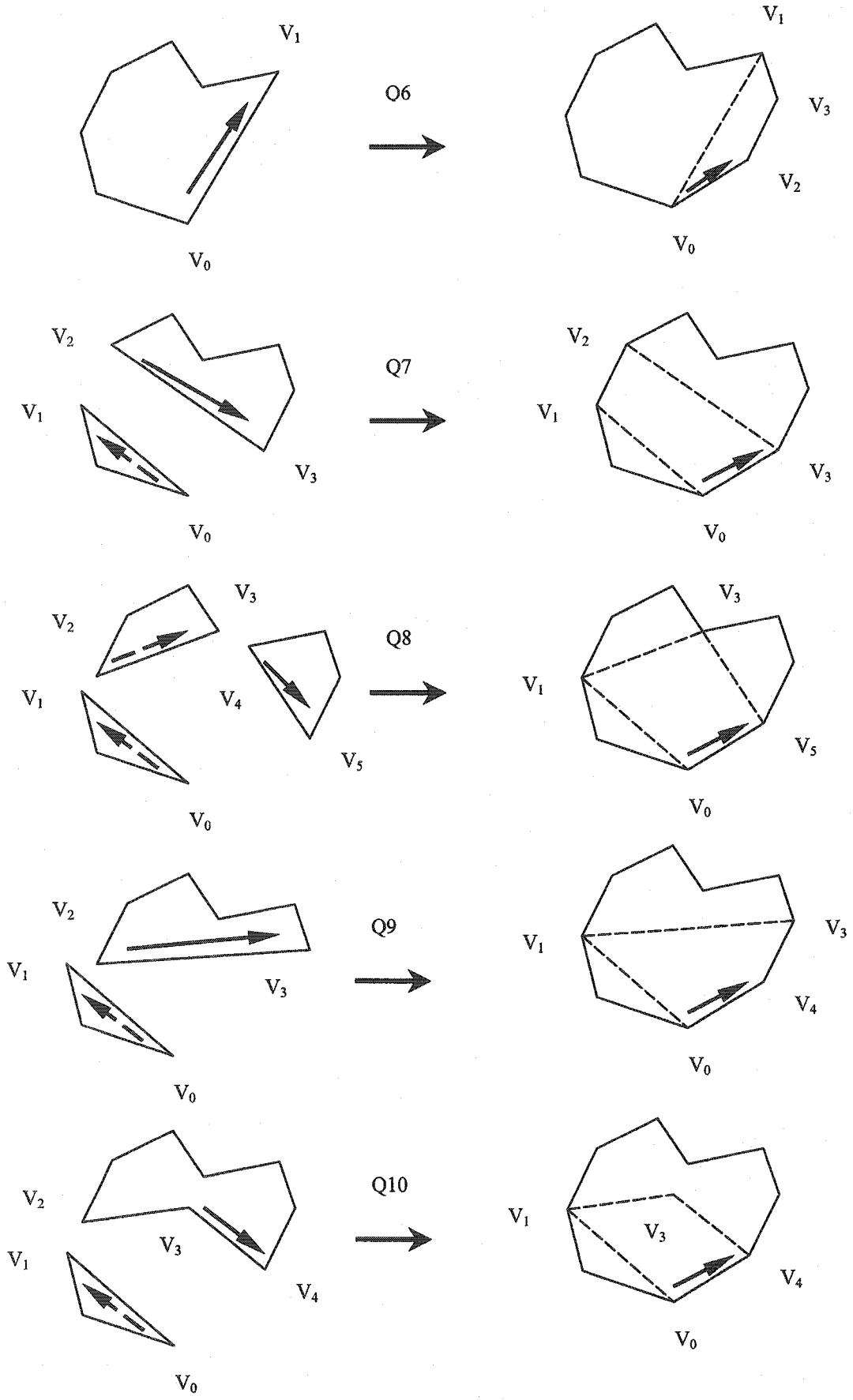
Once we have the coding scheme and binary coding stream, we can easily extract ASKII operations from the coding stream.

3.3 Spirale Reversi Decompression

The inputs for decompression will be three files: a file of interaction types (Q1-Q13), a file of InnerVertex (which records the inner vertex references of the quad mesh during the compression process), and a file of vertex data. The output of the decompression will be a file of opposite edge table, and a file of boundary vertex references.

Following the Spirale Reversi decoding scheme of Isenburg and Snoeyink[36], the boundary edges are directed counterclockwise; the inside of the boundary is to the right of a gate. The outside of the boundary is to the left of the gate. The compression process is viewed as growing the inside until there was no unencoded quad is left outside. The decompression process is viewed as growing the outside until there is no undecoded quad left inside. Figure 3-2 illustrates the decompression scheme.


Mesh decompressed so far
Q1
Q2
Q3
Q4
Q5
V0
V1
V2
V3
V4


V1
V0
Q6
V1
V3
V2
V0
V2
V1
V3
V0
Q7
V2
V1
V3
V0
V3
V2
V1
V4
V5
V0
Q8
V3
V1
V5
V0
V2
V1
V3
V0
Q9
V1
V3
V4
V0
V2
V1
V3
V4
V0
Q10
V1
V3
V4
V0

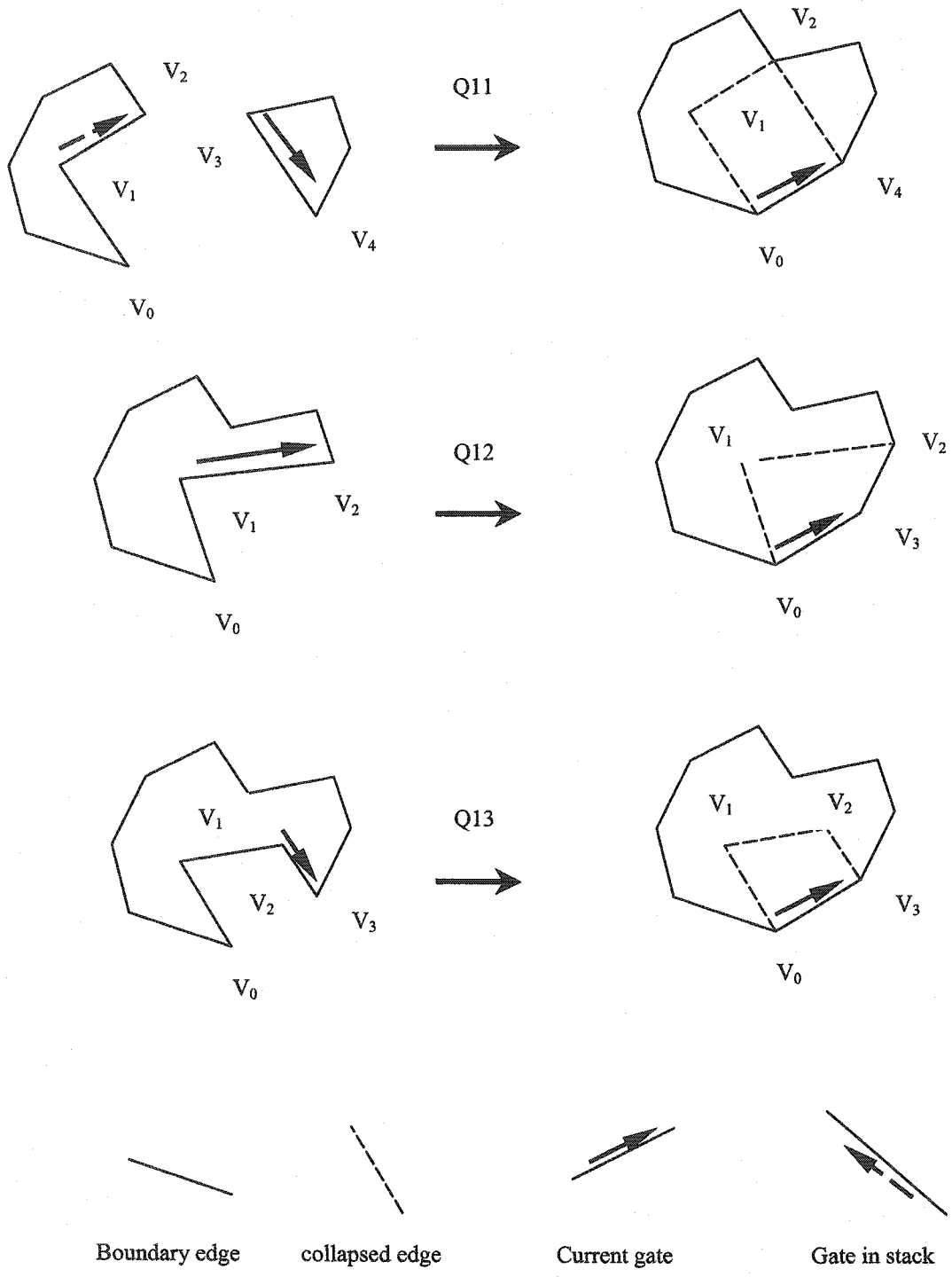


Figure 3-2. Spirale Reversi decompression operations

The detailed description of the Spirale Reversi decompression for each quad operation namely, the creation of new vertices, the deletion of old vertices, and the updates of the boundary and the gate are as follows:

- *The Q1 operation creates four new vertices. If there is already a boundary list before the Q1 operation, it will be pushed onto a stack, BListStack; and the current active gate will be pushed onto the gate stack. The four new vertices will comprise the new boundary list, BList. Edge(V_0,V_1) will be the active gate.*

- *The Q2 operation creates two new vertices. The two new vertices will be inserted between V_0 and V_1 to form the updated boundary, where edge(V_0,V_1) is the current active gate. Edge(V_3,V_1) will be the new active gate.*

- *The Q3 operation creates one new vertex. The end vertex of the gate popped from gate stack and the start vertex of current gate will be merged since they are the same vertex. The boundary list popped from BListStack and the current boundary list will be merged through the new vertex to form the updated boundary, i.e. V_4 will connect V_0 and V_3. Edge(V_4,V_3) will be the new active gate.*

- *The Q4 operation creates two new vertices. The two new vertices will be inserted between V_0 and V_1 to form the updated boundary, where edge(V_0,V_1) is the current active gate. Edge(V_2,V_3) will be the active gate.*

- *The Q5 operation creates one new vertex. The new vertex, V_3, will be inserted between the vertex previous to the start vertex of the active gate, V_0, and the end vertex of the active gate, V_2. The start vertex of the active gate, V_1, will be pushed onto the inner vertex stack. Edge(V_3, V_2) will be the updated active gate.*

- *The Q6 operation creates two new vertices. The two new vertices will be inserted between V_0 and V_1 to form the updated boundary, where edge(V_0,V_1) is the current active gate. Edge(V_0,V_2) will be the updated active gate.*

- *The Q7 operation creates no new vertex. The end vertex of the gate popped from gate stack and the start vertex of current gate will form a new edge; the start vertex of the gate popped from gate stack and the end vertex of current gate will form another new edge. The boundary list popped from the BListStack and the current boundary list will be connected through these two new edges and the edges of the two gates are collapsed to form the updated boundary. Edge(V_0,V_3) will be the new active gate.*

- *The Q8 operation creates no new vertex. The end vertex (V_3) of the gate popped from gate stack and the start vertex (V_4) of current gate will be merged since they are the same vertex. The end vertex(V_1) of the gate popped* ***(again)*** *from gate stack and the start vertex(V_2) of previously popped gate will be merged since they are the same vertex. The two boundary lists popped from BListStack and the current boundary list will be merged through the new*

edge(V_0,V_5) and the two common vertices, and collapsing of the three gates to form the updated boundary. Edge(V_0,V_5) will be the new active gate.

- *The Q9 operation creates one new vertex. The end vertex of the gate popped from gate stack and the start vertex of current gate will be merged since they are the same vertex. The boundary list popped from BListStack and the current boundary list will be merged through the new vertex to form the updated boundary, i.e. V_4 will connect V_0 and V_3. Edge(V_0,V_4) will be the new active gate.*

- *The Q10 operation creates no new vertex. The end vertex (V_1) of the gate popped from gate stack and the previous vertex (V_2) of the start vertex of current gate in the current BList will be merged since they are the same vertex. The boundary list popped from BListStack and the current boundary list will be merged through V_1 and the edge(V_0,V_4) to form the updated boundary. The Edge(V_0,V_1), Edge(V_2,V_3), and Edge(V_3,V_4) will be collapsed. The start vertex of the active gate, V_2, will be pushed onto inner vertex stack. Edge(V_0,V_4) will be the new active gate.*

- *The Q11 operation creates no new vertex. The end vertex (V_2) of the gate popped from gate stack and the start vertex (V_3) of current gate in the current BList will be merged since they are the same vertex. The boundary list popped from BListStack and the current boundary list will be merged through V_2 and the edge(V_0,V_4) to form the updated boundary. The Edge(V_0,V_1), Edge(V_1,V_2), and Edge(V_3,V_4) will be collapsed. The start vertex of the gate*

popped from gate stack, V_1, will be pushed onto inner vertex stack. Edge(V_0,V_4) will be the new active gate.

- *The Q12 operation creates one new vertex. The new vertex, V_3, will be inserted between the previous vertex of the start vertex of the active gate, V_0, and the end vertex of the active gate, V_2. The start vertex of the active gate, V_1, will be pushed onto inner vertex stack. Edge(V_0, V_3) will be the updated active gate.*

- *The Q13 operation creates no new vertex. The previous vertex to previous vertex of the start vertex of the active gate, V_0 , will connect directly with the end vertex of the active gate, V_3, to form the updated boundary list. The Edge(V_0,V_1), Edge(V_1,V_2), and Edge(V_2,V_3) will be collapsed. Vertices V_1 and V_2 will be pushed onto the inner vertex stack in sequence. Edge(V_0, V_3) will be the updated active gate.*

The decompression algorithm builds the opposite-edge table and also the table of vertex locations. It is possible to decode the compressed information using schema of Wrap and Zip[74] in linear time. For simplicity, we have used the Spirale Reversi scheme of Isenburg and Snoeyink. The mesh of Quads will be reconstructed in an order opposite to which they were encoded. The following is the pseudo code:

```
Push every interaction type in to a stack, operationStack; //step1
Create a tempBList with no element;                        //step2
```

```
While( !operationStack.empty())                    //step3
{
   currentOperation= operationStack.pop();
   For this quad, assign new vertices  temporary IDs (an integer number);
   Assign Opposite-Edge of each edge as -1;
   Update Opposite-Edge according to interaction type;
   Push vertex which will be remove from tempBlist into InnerVertexStack;
   Update tempBList;
}
Assign each vertex popped from InnerVertxStack the vertex location information;
//step4
Assign each vertex in final tempBList the vertex location  information.
//step5
```

Program 2 Spirale Reversi decompression process

Since we use the Spirale Reversi algorithm to reconstruct the mesh, we have to order the edges of each quad from the beginning. The last quad in the compression process will be the first quad in decompression process. Therefore, the opposite edge references will be different from the original opposite-edge table. But this will not affect the topology of the mesh since all vertex references of each quad remains the same in the compression/decompression process.

3.4 Example go-through

3.4.1 A mesh with boundary

Figure 3-3 shows the compression and decompression process of a quadrilateral mesh with boundary.

For the compression process, Figure 3-3(a) is the input mesh., an arbitrary edge of the boundary is selected as a gate, and the quad it incident on has interaction type Q13 with the boundary. Thus, Q13 is written to the code output.

In Figure 3-3(a), there is only one gap available, the edge in current quad adjacent to the gate in the anti-clockwise order will be selected as new gate as shown in Figure 3-3(b). The boundary will be updated by deleting the old gate. The interaction type of the new quad, which the gate is incident on, is Q6. This is written to the code output.

In Figure 3-3(b), there is only one gap available, the edge in the current quad that is previous to the gate in clockwise order will be selected as new gate as shown in Figure 3-3(c). The boundary will be updated by deleting the other three edges of the current quad. The interaction type of the new quad, which the gate is incident on, is Q6. This is written to the code output.

In Figure 3-3(c), there is only one gap available, the edge in the current quad that is previous ne to the gate in the clockwise order will be selected as new gate as shown in Figure 3-3(d). The

boundary will be updated by deleting the other three edges of the current quad. The interaction type of the new quad, which the gate is incident on, is Q5. This is written to the code output.

In Figure 3-3(d), there is only one gap available, the edge in current quad next to the gate in anti-clockwise order will be selected as new gate as shown in Figure 3-3(e). The boundary will be updated by deleting the other two edges (not on the gap) of the current quad. The interaction type of the new quad, which the gate is incident on, is Q12. This is written to the code output.

In Figure 3-3(e), there is only one gap available, the edge in the current quad next to next to the gate in the anti-clockwise order will be selected as new gate as shown in Figure 3-3(f). The boundary will be updated by deleting the other two edges (not on the gap) of the current quad. The interaction type of the new quad, which the gate is incident on, is Q12. This is written to the code output.

In Figure 3-3(f), there is only one gap available, the edge in the current quad next to next to the gate in anti-clockwise order will be selected as new gate as shown in Figure 3-3(g). The boundary will be updated by deleting the other two edges (not on the gap) of the current quad. The interaction type of the new quad, which the gate is incident on, is Q6. This is written to the code output.

In Figure 3-3(g), there is only one gap available, the edge in the current quad previous to the gate in anti-clockwise order will be selected as new gate as shown in Figure 3-3(h). The

boundary will be updated by deleting the other three edges of the current quad. The interaction type of the new quad, which the gate is incident on, is Q6. This is written to the code output.

In Figure 3-3(h), there is only one gap available, the edge in the current quad next previous to the gate in anti-clockwise order will be selected as the new gate as shown in Figure 3-3(i). The boundary will be updated by deleting the other three edges of the current quad. The interaction type of the new quad, which the gate is incident on, is Q12. This is written to the code output.

In Figure 3-3(i), there is only one gap available, the edge in current quad next to next to the gate in anti-clockwise order will be selected as the new gate as shown in Figure 3-3(j). The boundary will be updated by deleting the other two edges (not on the gap) of the current quad. The interaction type of the new quad, which the gate is incident on, is Q12. This is written to the code output.

In Figure 3-3(j), there is only one gap available, the edge in the current quad next to next to the gate in the anti-clockwise order will be selected as the new gate as shown in Figure 3-3(k). The boundary will be updated by deleting the other two edges (not on the gap) of the current quad. The interaction type of the new quad, which the gate is incident on, is Q9. This is written to the code output.

In Figure 3-3(k), there two gaps available, the edge in the current quad next to next to the gate in anti-clockwise order will be selected as the new gate as shown in Figure 3-3(l); the edge in the current quad previous to the gate in anti-clockwise order will be pushed onto the stack for

gates . The boundary will be updated by deleting the other two edges of the current quad, which will result in a split into two boundaries. The boundary, which the new gate is incident on, will be kept as current boundary while the other boundary will be pushed onto the stack for boundary lists. The interaction type of the new quad, which the gate is incident on, is Q1. This is written to the code output.

In Figure 3-3(l), after the Q1 operation there is no gap or boundary left. We need to pop boundaries and gates from their respective stacks. The popped gate and boundary are shown in Figure 3-3(m). In Figure 3-3(m), there is only one gap available, the edge in current quad previous to the gate in anti-clockwise order will be selected as new gate as shown in Figure 3-3(n). The boundary will be updated by deleting the other three edges of the current quad. The interaction type of the new quad, which the gate is incident on, is Q1. This is written to the code output.

In Figure 3-3(n), after the Q1 operation there is no gap or boundary left. We need to pop a boundary and a gate from their respective stacks. But at this time both the gate stack and the boundary stack are empty, implying that we have compressed all the quads in the mesh.The compression process therefore terminates.

The final compression code is: Q13-Q6-Q6-Q5-Q12-Q12-Q6-Q6-Q12-Q12-Q9-Q1-Q6-Q1.

In the decompression process, we start to reconstruct the mesh in the reverse order of the compression process.

At the beginning, we do not have any boundary or gate. After reading Q1, we need to generate a new boundary list for this new quad and assign any one of its edges as a gate, as shown in Figure 3-3(n).

After reading Q6, we need to create two new vertices to form a new quad with the current gate; then we update the boundary list and assign the edge adjacent to the shared edge(anti-clock-wise) in the new quad as the new gate, as shown in Figure 3-3(m).

After reading Q1, we need to generate a new boundary list for this new quad and assign any one of its edges as the new gate, as shown in Figure 3-3(k).

After reading Q9, we need to create a new quad to combine the previous two boundaries together to form a new boundary. The old gate and the new gate have a common vertex. One new vertex is generated to form a new quad with the two gates. The edge previous to previous to the current gate in the boundary will be assigned as the new gate, as shown in Figure 3-3(j).

After reading Q12, we need to create a new vertex the form a new quad with the current gate and the edge previous to the gate in the boundary. The edge previous to previous to the current gate in the boundary will be assigned as the new gate, as shown in Figure 3-3(j).

After reading Q12, we need to create a new vertex the form a new quad with the current gate and the edge previous to the gate in the boundary. The edge previous to previous to the current gate in the boundary will be assigned as the new gate, as shown in Figure 3-3(i).

After reading Q6, we need to create two new vertices to form a new quad with the current gate. The edge in the new quad next to (anti-clockwise) the current gate will be assigned as the new gate, as shown in Figure 3-3(h).

After reading Q6, we need to create two new vertices to form a new quad with the current gate. The edge next to (anti-clockwise) the current gate in the new quad will be assigned as the new gate, as shown in Figure 3-3(g).

After reading Q12, we need to create one new vertex to form a new quad with the current gate and the edge previous to the gate in the boundary. The edge previous to previous to (anti-clockwise) the current gate in the new quad will be assigned as the new gate, as shown in Figure 3-3(f).

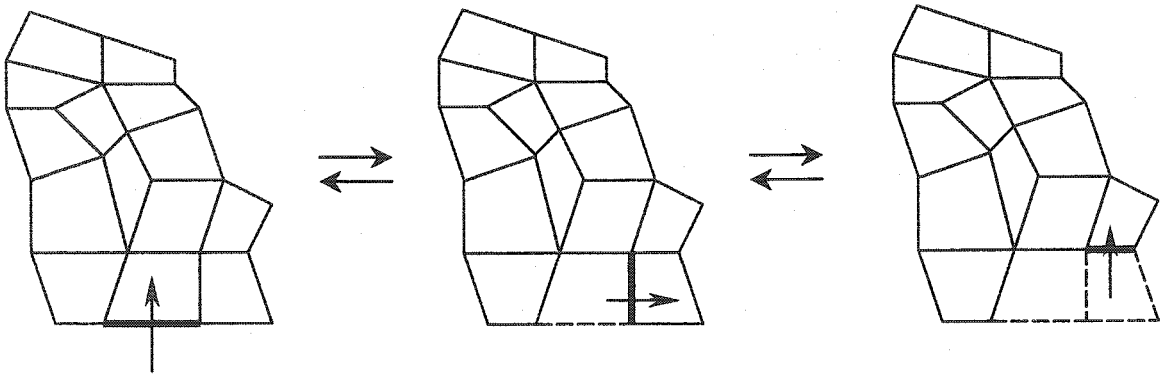
After reading Q12, we need to create one new vertex to form a new quad with the current gate and the edge previous to the gate in the boundary. The edge previous to previous to (anti-clockwise) the current gate in the new quad will be assigned as the new gate, as shown in Figure 3-3(e).

After reading Q5, we need to create one new vertex to form a new quad with the current gate and the edge previous to the gate in the boundary. The edge next to (anti-clockwise) the current gate in the new quad will be assigned as the new gate, as shown in Figure 3-3(d).

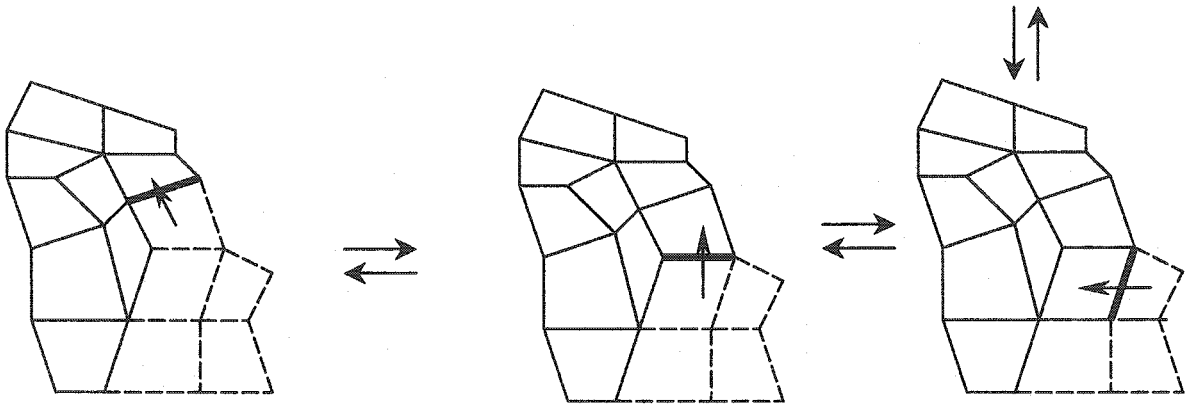
After reading Q6, we need to create two new vertices to form a new quad with the current gate. The edge in the new quad next to (anti-clockwise) the current gate will be assigned as the new gate, as shown in Figure 3-3(c).

After reading Q6, we need to create two new vertices to form a new quad with the current gate. The edge in the new quad next to (anti-clockwise) the current gate will be assigned as the new gate, as shown in Figure 3-3(b).

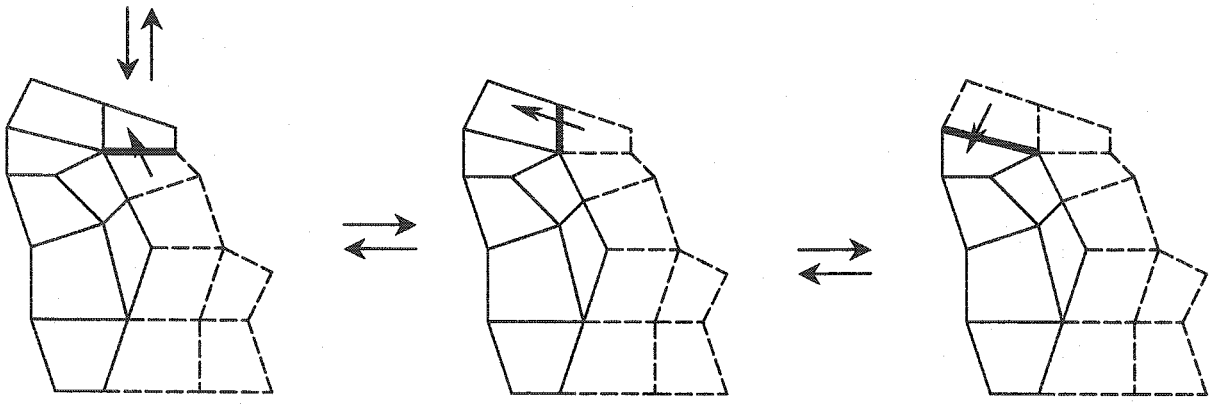
After reading Q13, we need to create no new vertex to form a new quad with the current gate. But we need the edge previous to previous to the gate, the edge previous to the gate, and the gate to form a new quad. The edge in the new quad previous to (anti-clockwise) the current gate will be assigned as the new gate, as shown in Figure 3-3(a). Since all quads are recovered, the compression process terminates.



(a) Q13 (b) Q6 (c) Q6



(f) Q12 (e) Q12 (d) Q5



(g) Q6 (h) Q6 (i) Q12

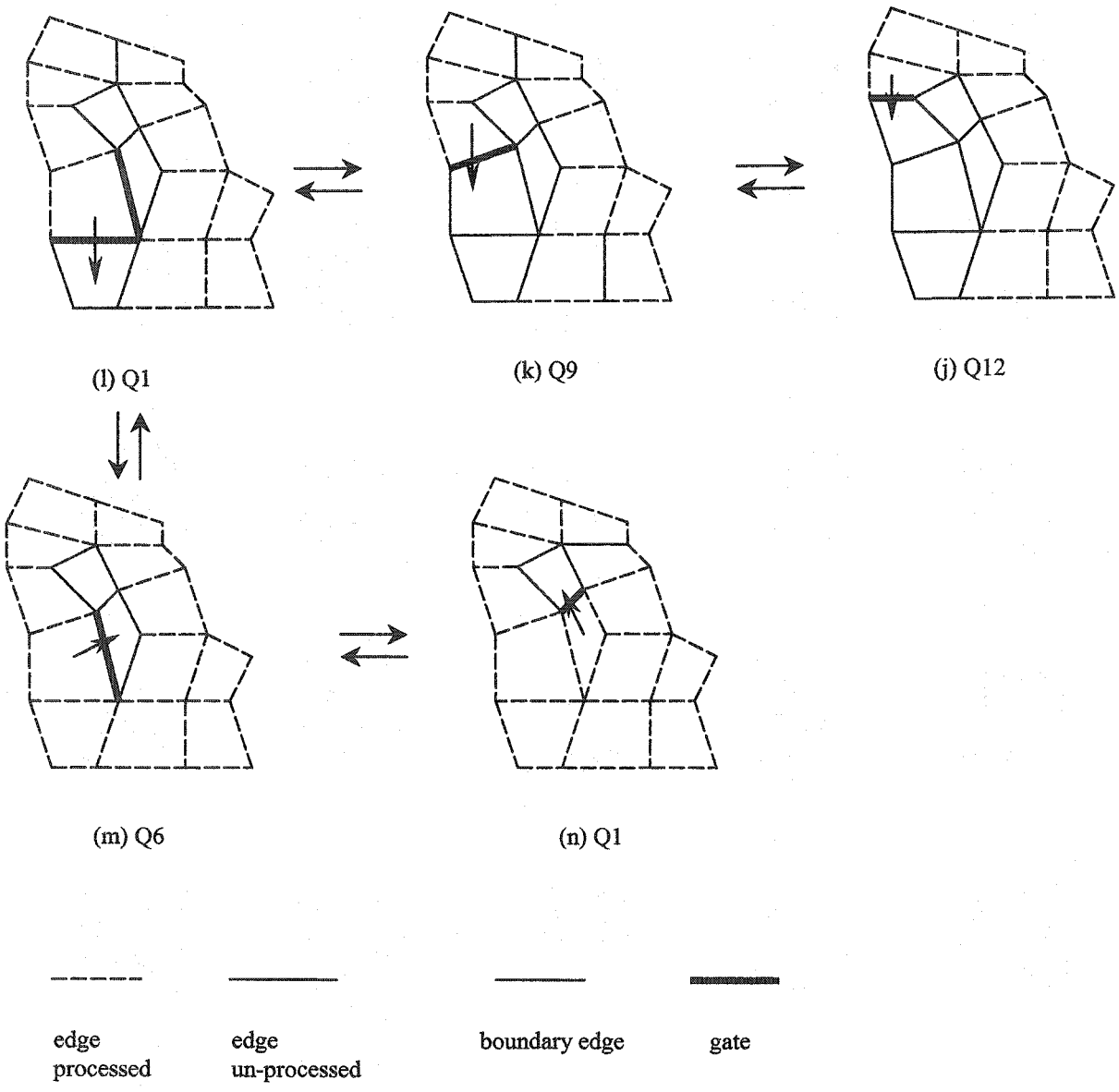


Figure 3-3. Compression/Decompress of a quad mesh with boundary

3.4.2 A mesh without boundary

For a mesh without boundary, we can easily create a boundary by cutting any one quad open, then the four edges of this quad will form the mesh boundary, as shown in Figure 3-4(b).

Then the compression and decompression process will be exactly the same as for a mesh with boundary, as shown in Figure 3-4(b-f). After the decompression process terminates, we need to glue back the quad we originally cut open, as shown in Figure3-4(a).

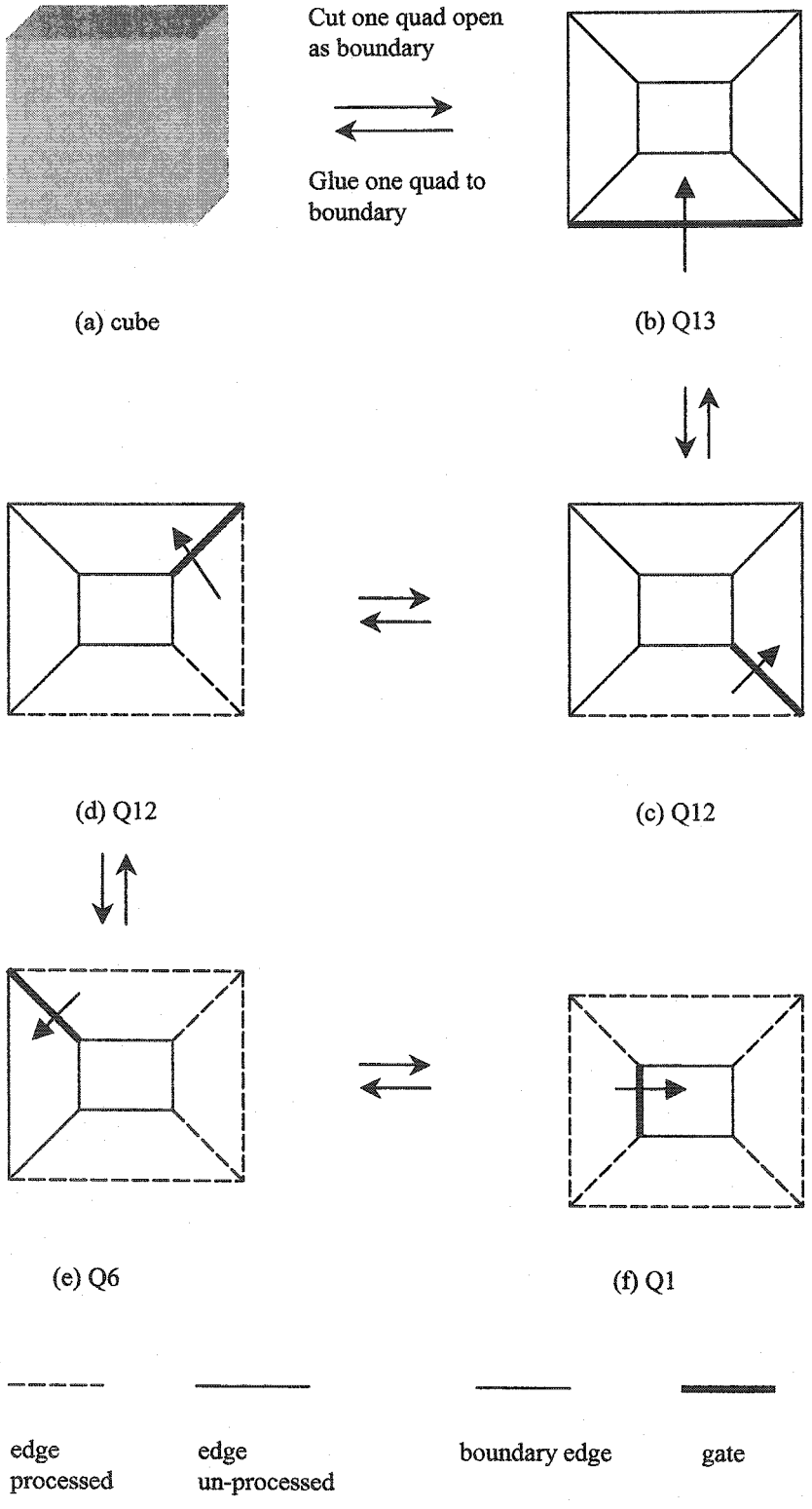


Figure 3-4. Compression/Decompress of a quad mesh without boundary

3.5 Time Complexity analysis

The compression process is shown in Program 1. Step 1 takes constant time *c1*. Step 2 takes *O(n)* time. Step 3 takes constant time c2. Step 4 takes constant time *c3*. Since each quad is processed exactly once, the processing time for each quad will be constant time, and accessing the next quad is constant time, step 5 takes *O(n)* time. Therefore, the overall time complexity of compression process will be *c1+O(n)+c2+C3+O(n)* which is *O(n)*.

The decompression process is shown in Program 2. Step 1 takes constant time *d1*. Step 2 takes constant *d2*. In step 3, each quad is processed exactly once; the processing time may vary for different operation but for each operation, the processing time will be constant; therefore, step 3 takes *O(n)* time. Step 4 taken *O(n)* time. Step 5 takes *O(n)* time. Therefore, the overall time complexity of decompression process will be *d1+d2+O(n)+O(n) +O(n)* which is *O(n)*.

In sum, both the compression and decompression processes have linear time complexities in terms of the mesh size.

3.6 Summary

Efficient rendering of a quad mesh is an integral part of quad mesh compression/decompression. The difficulty of traditional data structures in the compression process of quad mesh is that it is time consuming to find which quad is adjacent to a given quad. This problem is solved using opposite edge data structure in this thesis by storing four integer references for all the quads adjacent to the quad. The Spirale Reversi decompression algorithm for triangle meshes is successfully extended to quadrilateral meshes. Time complexity analysis shows that both the compression and decompression algorithms are linear in the mesh size.

Chapter 4

Coding schemes with lower upper bound

Using the compression algorithm discussed in Chapter 3, we can get one interaction type for each quad in a quadrilateral mesh in ASKII format. This chapter will present new coding schemes. Section 4.1 will prove that a coding scheme has 3.0 bpv upper bound. Section 4.2 will present 2.67 coding schemes.

4.1 Coding scheme of 3.0 bpv upper bound

ASKII format is not a succinct format since each character in the file will consume at least 8 bits (1 byte). We need to assign codes to the thirteen interaction types to convert ASKII file to binary file. For a manifold quad mesh, there are 13 possible interactions (see summarized table 4-1) between a quad and mesh boundary. Normally, thirteen types need $[\log_2 13]= 4$ bits to distinguish each other. This will guarantee 4.0 bpv upper bound.

Kronrod and Gotsman's algorithm[48], briefly called KG algorithm, traverse the quad mesh in clock-wise and achieves 3.5 bpv. If we modify the traverse order to counterclockwise, the upper bound of 3.5 bits per vertex still holds. With our new coding scheme, it is very easy to prove 3.0 bpv upper bound.

Table 4-1 Property of interaction types

Interaction type	Number of new edges introduced	Number of new vertices introduced
Q1	0	0
Q2	1	0
Q3	2	0
Q4	1	0
Q5	2	1
Q6	1	0
Q7	2	0
Q8	3	0
Q9	2	0
Q10	2	1
Q11	3	1
Q12	2	1
Q13	3	2

Based on careful observation of the interaction types during the compression process leads us to make the following claim:

Claim: A quad of type Q5, Q10, Q12, or Q13 is never followed by a quad of type Q1, Q2, Q3, Q4, Q5, if we traverse the mesh in anti-clockwise order.

Proof: When the quad–boundary interaction is of type Q1, Q2, Q3, Q4 and Q5, the quad that is removed has the following property: the previous edge of the gate in clockwise order belongs to the boundary. However, if we examine a quad-boundary interaction of the type Q5, Q10, Q12, or Q13 in counterclockwise order, the new gate is the right edge or the opposite edge of the gate of the removed quad, and the removal also generates at least one free vertex. (for interaction type Q13, two free vertices will be generated and the first free-vertex in anti-clockwise order will be part of the new gate). This means the left edge of the previous quad will be the previous edge of the new gate on the boundary and the

first free-vertex of previous quad must be part of the new gate. It is impossible for the second quad to share both the right edge and opposite edge of the removed quad. Therefore, the previous edge of the new gate will not be an edge of the second quad. Therefore, a quad-boundary interaction of type Q5, Q10, Q12, or Q13 is never followed by one of type Q1, Q2, Q3, Q4, or Q5.

With these constraints, the code for each interaction-type can be selected in an elegant manner such that the total cost to code the connectivity will not exceed 3 bits per vertex.

Table 4-2 Code table of Coding I:

Encoding	Current Quad	Next Quad	Code	Num. of bits
Quad started with Q6-13	Q6	Q1-5	1010	4
		Q6-13	1011	4
	Q7	Q1-5	111100	6
		Q6-13	111101	6
	Q8	Q1-5	111010	6
		Q6-13	111011	6
	Q9	Q1-5	111110	6
		Q6-13	111111	6
	Q10	Q6-13	11100	5
	Q11	Q1-5	1100	4
		Q6-13	1101	4
	Q12	Q6-13	100	3
	Q13	Q6-13	0	1
Quad started with Q1-5	Q1	Q1-5	00	2
		Q6-13	01	2
	Q2	Q1-5	1100	4
		Q6-13	1101	4
	Q3	Q1-5	1010	4
		Q6-13	1011	4
	Q4	Q1-5	1000	4
		Q6-13	1001	4
	Q5	Q6-13	111	3

Here is the proof for 3.0 bits per vertex upper bound.

Let E be the number of edges, V the number of vertices and Q the number of quads in a simple quad mesh (without boundary?). Since each edge is shared by exactly two quads,

$$E=2Q \qquad (1)$$

By Euler's formula we have

$$V-E+Q=2 \qquad (2)$$

Adding (1) and (2) and rearranging we get,

$$V=Q+2 \qquad (3)$$

For large mesh, Q>>2, therefore ignoring the additive term in (3) above, we can claim that

$$2V=E,\ V=Q. \qquad (4)$$

From (4) above and Table 4-1, it follows that

$$2|Q13|+|Q5|+|Q10\text{-}12|=Q \qquad (5)$$

as the left-hand side counts the number of vertices in the quad mesh, where $|Qi|$ denotes the number of quads of type Qi and $|Qi\text{-}j|=|Qi|+\ldots+|Qj|$, $j>i$

Again as V=Q, the number of quads which have two free-vertices must be equal to the number of quads which have no free vertices. Thus from Table 4-1 it follows that,

$$|Q13|= |Q1\text{-}4|+|Q6\text{-}9| \qquad (6)$$

Each branch in quad spanning tree ends in a Q1 (a leaf node), and each branch begins either at the root gate or at a Q3 or Q7-11. For each of Q3, Q7, Q9, Q10, and Q11, one more Q1 is associated. For one Q8, two more Q1's are associated. Therefore

$$|Q3|+|Q7\text{-}11|+|Q8|=|Q1|-1 \qquad (7)$$

Following the amortization analysis technique used in [2], we obtain the following groupings and amortized costs:

Step 1. Using the constraints of equation (6), viz. |Q13|= |Q1-4|+|Q6-9| we can group each of the interaction-types Q1-4 and Q6-9 with interaction-type Q13 as shown in the following table:

Table 4-3 Code bits analysis for Q1 to Q9

Code1	# bits of Code1	Code2	# bits of Code2	Average bits	<=3??
Q1	2	Q13	1	1.5	Yes
Q2	4	Q13	1	2.5	Yes
Q3	4	Q13	1	2.5	Yes
Q4	4	Q13	1	2.5	Yes
Q6	4	Q13	1	2.5	Yes
Q7	6	Q13	1	3.5	No
Q8	6	Q13	1	3.5	No
Q9	6	Q13	1	3.5	No

Thus the grouping of Q1, Q2, Q3, Q4 and Q6 with Q13 yield an average bit count of less than 3. However the grouping for the interaction-types Q7, Q8 and Q9 is still not satisfactory.

Step 2. We need to use the constraint of equation (7), viz. |Q3|+|Q7-11|+|Q8|=|Q1|-1

This equation implies that |Q7-11|+|Q8|<|Q1|-1, which in turn implies that each of the interaction-types from Q7-11 and Q8 can be associated with at most one Q1.

Since Q3 has been taken care of in step 1, we do not need to group it in step 2. Also because Q1 is grouped with Q13 already and one Q1 is associated with each one of Q7, Q9, Q10, Q11 and two Q1's are associated with a Q8, we need to associate a single interaction-type group (Q1, Q13) with each one of (Q7, Q13), (Q9, Q13), Q10, and Q11 and we need to associate two of (Q1, Q13) associated with one of (Q8, Q13). The grouping details are shown in Table 4-4:

Table 4-4 Code bits analysis for Q7 to Q11

Code1	# bits of Code1	Code2	# bits of Code2	Average bits	<=3??
(Q7, Q13)	7	(Q1,Q13)	3	2.5	Yes
(Q8, Q13)	7	(Q1,Q13, Q1, Q13)	6	2.17	Yes
(Q9, Q13)	7	(Q1,Q13)	3	2.5	Yes
Q10	5	(Q1,Q13)	3	2.67	Yes
Q11	4	(Q1,Q13)	3	2.33	Yes

Step 2 makes sure that Q7, Q8, Q9, Q10, Q11 can grouped to achieve no more than 3 bits per vertex coding.

Interaction-types Q5 and Q12 are still left, but these are already assigned 3 bits each.

We conclude that quad mesh connectivity can be encoded in no more than 3V bits. The code group is summarized in Table 4-5.

Table 4-5 Amortization analysis table of Coding I

Grouping	Total Cost	# of quads in group	Amortized Cost	#bits saved	#occurrences of this group
(Q2, Q13) or (Q3,Q13) or (Q4,Q13) or (Q6,Q13)	5	2	2.5	1	|Q2|+|Q3|+|Q4|+|Q6|
(Q7, Q13, Q1, Q13) or (Q9, Q1, Q13) or	10	4	2.5	2	|Q7|+|Q9|
(Q8, Q13,Q1, Q13, Q1, Q13)	13	6	13	5	|Q8|
(Q10,Q1,Q13)	8	3	2.67	1	|Q10|
(Q11, Q1,Q13)	7	3	2.33	2	|Q11|
remaining(Q1,Q13)	3	2	1.5	3	|Q3|+1
Q5	3	1	3	0	|Q5|
Q12	3	1	3	0	|Q12|

According to Table 4-5, the total cost of the encoding, T(I), equals 3Q-(Q2|+|Q3|+|Q4|+|Q6|)- 2(|Q7|+|Q9|)-5|Q8|-|Q10|-2|Q11|-3|Q3|-3.

From equations (6)-(7), we get

$$|Q13| = |Q1-4|+|Q6-9|-(|Q1|-1)+(|Q3|+|Q7-11|+|Q8|)$$

$$= |Q2| + 2|Q3| + |Q4| + |Q6| + 2|Q7| + 2|Q8| + 2|Q9| + |Q10| + |Q11| \quad (8)$$

Substitute (8) into the above equation, we get

$$T(I) = 3Q-|Q13|-2|Q3|-3|Q8|-|Q11|-3 \quad (9)$$

Since V=Q+2 for a quad mesh, the total cost is therefore guaranteed to be less than 3V bits.

In the next section, we show how to obtain a tighter upper bound.

4.2 Improving the upper bound to 2.67 bpv

Firstl, we modify the coding scheme of Table 4-2 by switching the code for Q11 followed by Q1-5 with the code for Q10. This gives us the coding scheme of Table 4-6 below.

Table 4-6 Code table of Coding II:

Encoding	Current Quad		Next Quad	Code	Num. of bits
Quad started with Q6-13	Q6	Q6	Q1-5	1010	4
		Q6	Q6-13	1011	4
	Q7	Q7	Q1-5	111100	6
		Q7	Q6-13	111101	6
	Q8	Q8	Q1-5	111010	6
		Q8	Q6-13	111011	6
	Q9	Q9	Q1-5	111110	6
		Q9	Q6-13	111111	6
	Q10	Q10	Q6-13	1100	4
	Q11	Q11	Q1-5	11100	5
		Q11	Q6-13	1101	4
	Q12	Q12	Q6-13	100	3
	Q13	Q13	Q6-13	0	1
Quad started with Q1-5	Q1	Q1	Q1-5	00	2
		Q1	Q6-13	01	2
	Q2	Q2	Q1-5	1100	4
		Q2	Q6-13	1101	4
	Q3	Q3	Q1-5	1010	4
		Q3	Q6-13	1011	4
	Q4	Q4	Q1-5	1000	4
		Q4	Q6-13	1001	4
	Q5	Q5	Q6-13	111	3

Table 4-7 Amortization analysis table of Coding II

Grouping	Total Cost	# of quads in group	Amortized Cost	#bits saved	#occurrences of this group
(Q2, Q13) or (Q3,Q13) or (Q4,Q13) or (Q6,Q13)	5	2	2.5	1	|Q2|+|Q3|+|Q4|+ |Q6|
(Q7, Q13, Q1, Q13) or (Q9, Q13, Q1, Q13)	10	4	2.5	2	|Q7|+|Q9|
(Q8, Q13,Q1, Q13, Q1, Q13)	13	6	2.17	5	|Q8|
(Q10,Q1,Q13)	7	3	2.67	2	|Q10|
(Q11 followed by Q1-5, Q1,Q13)	8	3	2.33	1	|Q11 followed by Q1-5|
(Q11, followed by 6-13, Q1,Q13)	7	3	2.33	2	|Q11 followed by Q6-13|
remaining(Q1,Q13)	3	2	1.5	3	|Q3|+1
Q5	3	1	3	0	|Q5|
Q12	3	1	3	0	|Q12|

Then, the total cost for coding II is:

$$T(II) = 3Q-|Q13|-2|Q3|-3|Q8|-|Q10|-|Q11 \text{ followed by } Q6\text{-}13|-3 \quad (10)$$

Secondly, we can modify the coding much more by switch the code for Q11 followed by Q1-5 with the code for Q10 (as shown in Table 4-6), we can get the following amortization table:

Table 4-8 Code table of Coding III:

Encoding	Current Quad		Next Quad	Code	Num. of bits
Quad started with Q6-13	Q6	Q6	Q1-5	1010	4
		Q6	Q6-13	1011	4
	Q7	Q7	Q1-5	10010	5
		Q7	Q6-13	10011	5
	Q8	Q8	Q1-5	111110	6
		Q8	Q6-13	111111	6
	Q9	Q9	Q1-5	11100	5
		Q9	Q6-13	11101	5
	Q10	Q10	Q6-13	1000	4
	Q11	Q11	Q1-5	1100	4
		Q11	Q6-13	1101	4
	Q12	Q12	Q6-13	01	2
	Q13	Q13	Q6-13	00	2
Quad started with Q1-5	Q1	Q1	Q1-5	000	3
		Q1	Q6-13	111	3
	Q2	Q2	Q1-5	1100	4
		Q2	Q6-13	1101	4
	Q3	Q3	Q1-5	001	3
		Q3	Q6-13	101	3
	Q4	Q4	Q1-5	1000	4
		Q4	Q6-13	1001	4
	Q5	Q5	Q6-13	01	2

Table 4-9 Amortization analysis table of Coding III

Grouping	Total Cost	# of quads in group	Amortized Cost	#bits saved	#occurrences of this group
(Q2, Q13) or (Q3,Q13) or (Q4,Q13) or (Q6,Q13)	6	2	3	0	|Q2|+|Q3|+|Q4|+|Q6|
(Q7, Q13, Q1, Q13) or (Q9, Q13, Q1, Q13)	12	4	3	0	|Q7|+|Q9|
(Q8, Q13,Q1, Q13, Q1, Q13)	18	6	3	0	|Q8|
(Q10,Q1,Q13)	9	3	3	0	|Q10|
(Q11, Q1,Q13)	9	3	3	0	|Q11|
remaining(Q1,Q13)	5	2	2.5	1	|Q3|+1
Q5	2	1	2	1	|Q5|
Q12	2	1	2	1	|Q12|

Then, the total cost for coding III is:

$$T(III) = 3Q-|Q3|-|Q5|-|Q12|-3 \qquad (11)$$

Adding up equations (8),(9) and (10), we get

$$T(I)+T(II)+T(III) = 3Q-|Q13|-2|Q3|-3|Q8|-|Q11|-3$$
$$+ \; 3Q-|Q13|-2|Q3|-3|Q8|-|Q10|-|Q11 \text{ followed by } Q6\text{-}13|-3$$
$$+ \; 3Q-|Q3|-|Q5|-|Q12|-3$$
$$= 9Q-(2|Q13|+|Q5|+|Q10|+|Q11|+|Q12|)-7|Q3|-6|Q8|-9$$

By virtue of equation (4), viz. $2|Q13|+|Q5|+|Q10\text{-}12|=Q$, we get

$$T(I)+T(II)+T(III) = 9Q-Q-7|Q3|-6|Q8|-|Q11 \text{ followed by } Q6\text{-}13|-9$$
$$= 8Q-7|Q3|-6|Q8|-|Q11 \text{ followed by } Q6\text{-}13|-9 \qquad (12)$$
$$< 8Q$$

Therefore minimum cost of the coding schemes I, II, III is no more than

$$1/3(T(I)+T(II)+T(III)) <= 8Q/3 = 2.67Q \qquad (13)$$

i.e., one of coding schemes I , II, or III is guaranteed cost less than 2.67Q (since Q=V-2≈V, the cost may represented as 2.67 V as well) bits in the worst case.

Equation (12) suggests that it may be possible to get a tighter bound than 2.67V bits.

4.3 Summary

First, this chapter gives an encoding scheme which has an upper bound of 3.0 bpv. Then after amortization analysis of three different coding schemes, a tighter upper bound 2.67 bpv is found.

Chapter 5

Implementation and experimental results

In this chapter, the implementation of compression/decompression of quad meshes is introduced in section 5.1. Then, the experimental results are presented in section 5.2.

5.1 Implementation

The overall geometry compression and decompression is implemented in windows 98 platform. All programs, except those used to convert from ASKII code to binary and vice versa, have been written in Java; since Java is platform-independent, these programs are portable to other operating systems. All programs converting between ASKII code and binary one are compiled using the Visual C++ compiler. The implementation aspect of this thesis is composed of two parts: one addresses connectivity compression while the other addresses vertex compression.

5.1.1 Connectivity compression/decompression

Figure 5.1 illustrates the connectivity compression/decompression process. In the compression process, program *OEcomp* takes *OEtable* and *Boundary list* as input and generates *operation* in ASKII format as output. The vertex data is partitioned into two parts, one is *InnerV* (inner vertices) and the other is *BoundaryV* (boundary vertices). The

operation is converted into *operation.bin* as binary format depending on the coding scheme selected with the help of program *Bitsmap*.

In the decompression process, program *BitsToOp* takes *operation.bin* and the coding scheme used as input and converts it to its ASKII counterpart, *operation*. Program *OEdecomp* takes *operation, InnerV, and BoundaryV* as input and generates a new *OEtable* and *boundary list*.

The inter-relationships of the various modules are shown schematically in Figure 5-1.

5.1.2 Vertex compression/decompression

Figure 5.2 illustrates the vertex compression/decompression process. In the compression process, program *vertexComp* takes *Vertex* as input, quantizes each coordinate into 8 bits and generates *Delta* (Delta difference of the coordinates), *bounding*, and *Huffman table* of the delta differences as output. Then these three files are used by the program DeltaToBits to convert Delta (ASKII format) to Delta.bin (binary format).

In the decompression process, the program *BitsToCode* takes *Delta.bin*, *bounding*, and *Huffman table* to extract Delta (in ASKII format). Once delta is available, program *vertexDEcomp* converts the delta differences to vertex coordinates.

The inter-relationships of the various modules are shown schematically in Figure 5-2.

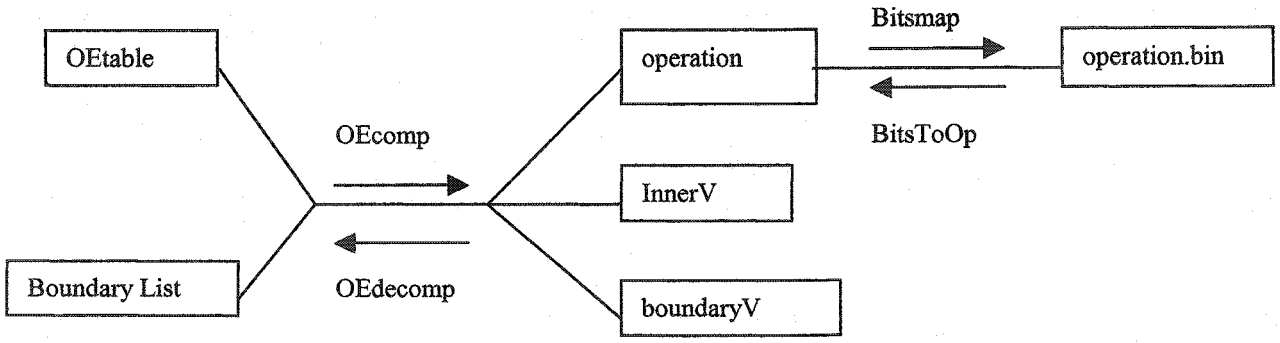


Figure 5-1 Connectivity Compression/decompression flow chart

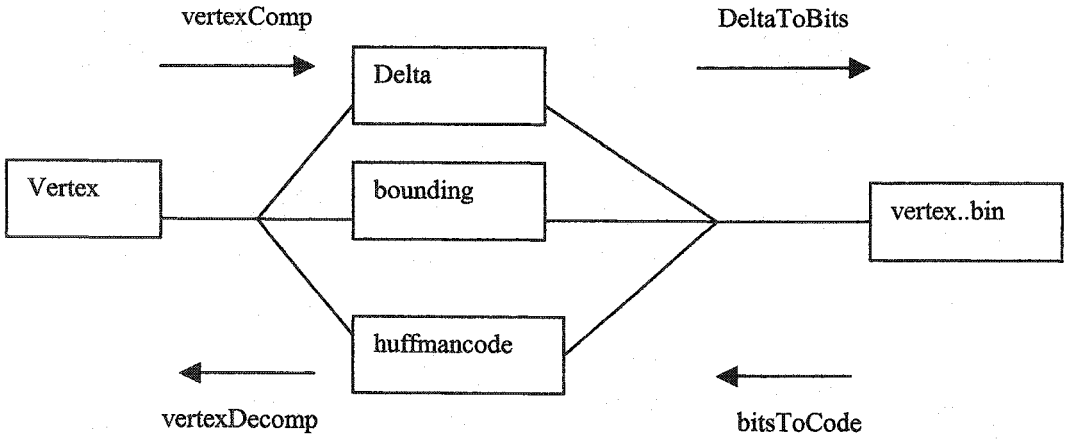


Figure 5-2 Vertex Compression/decompression flow chart

5.2 Experimental results

Sample pictures of meshes without boundary are shown as Figure 5-3 , 5-4, 5-5. Table 5-1 shows the experimental results of quad mesh compression and decompression:

Table 5-1. Compression/Decompression time and compression size

Num of Quad	Num Of Vertices	compression time (ms)	decompression time(ms)	Size Of Connectivity (bytes)	bpv
60	62	280	270	28	3.61
240	242	610	610	88	2.91
960	962	1320	1100	344	2.86
3840	3842	3510	4340	1364	2.84
15360	15362	12090	13350	5440	2.83
17414	17412	13409	14340	6172	2.83
61440	61442	44270	47790	21752	2.83

Note: 1.time is runtime on a Pentium 560, with other programming running;
2. ms is short form of millisecond;
3. bpv is short form of bits per vertex;
4. in the above examples, the mesh of 17414 quad is cow; all the other meshes are spheres.

If we draw a graph of compression/decompression time over the number of quad in the mesh (in logarithmic scale), shown as Figure 5-6, it clearly shows the linear relation.

Table 5-2 lists the performance of three coding schemes discussed in Chapter 4. From this table, we know for a large mesh, the cost of each coding scheme is smaller than 3.0 bpv. This agrees to the results of section 4.1. Also, cost of III is smaller than 2.67 bpv, which agrees with result of section 4.2. Coding III always has good performance probably because Q12 happens very frequently compared with other operations (See equation 10 in Chapter 4).

Table 5-2 Performance of difference coding schemes

Num of Quad	Num Of Vertices	Size Of Connectivity(bytes)			bpv		
		I	II	III	I	II	III
60	62	28	24	20	3.61	3.10	2.58
240	242	88	84	72	2.91	2.78	2.38
960	962	344	340	284	2.86	2.83	2.36
3840	3842	1364	1360	1128	2.84	2.83	2.35
15360	15362	5440	5436	4493	2.83	2.83	2.34
17412	17414	6172	6088	5156	2.83	2.80	2.37
61440	61442	21752	21748	17944	2.83	2.83	2.34

Table 5-3 Overall performance of geometry compression

File Size (bytes)	Num Of Quad	Num Of Vertices	connectivity (bytes)	vertex.bin (bytes)	Huffman table (bytes)	bounding	Overall (bpv)	**overall compression ratio**
3708	60	62	28	112	436	45	10.02	5.876
15,662	240	242	88	560	1136	27	7.48	8.613
66,888	960	962	344	2320	1706	27	4.57	15.196
288,538	3840	3842	1364	8420	1959	29	3.06	24.50
1,224,604	15360	15632	5440	28876	2035	31	2.33	33.66
5,197,754	61440	61442	21752	93840	2157	31	1.92	44.13

Note: 1. 8 bits quantization level is chosen for vertex compression;

2. Overall compression ratio is calculated using equation Cost(input files)/Cost(Output files, which include operation.bin, vertex.bin, Huffman table, bounding). Huffman table and bounding are in text format.

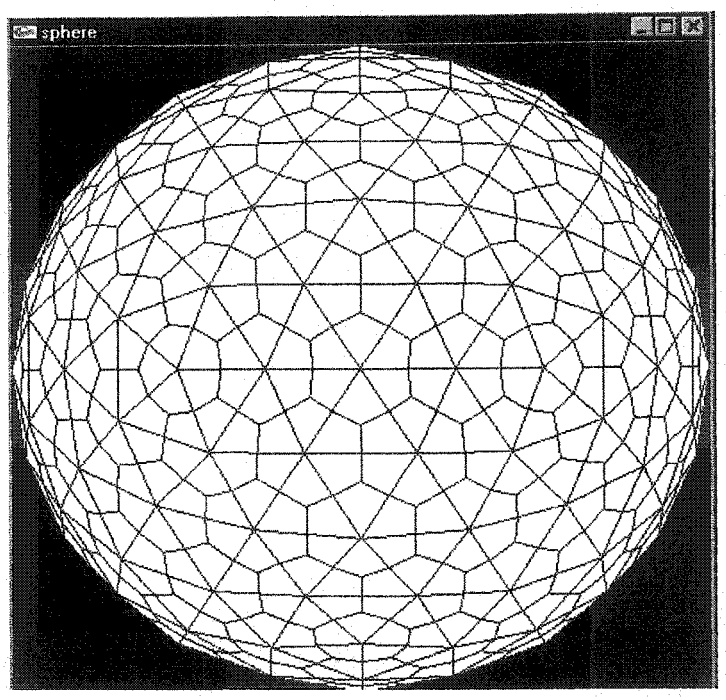


Figure 5-3 Sphere mesh with 240 quads and 242 vertices

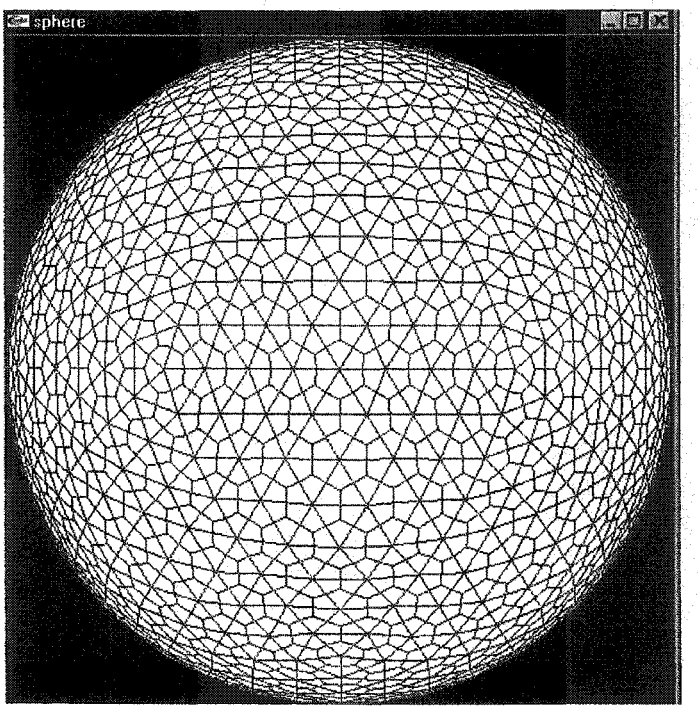


Figure 5-4 Sphere mesh with 960 quads and 962 vertices

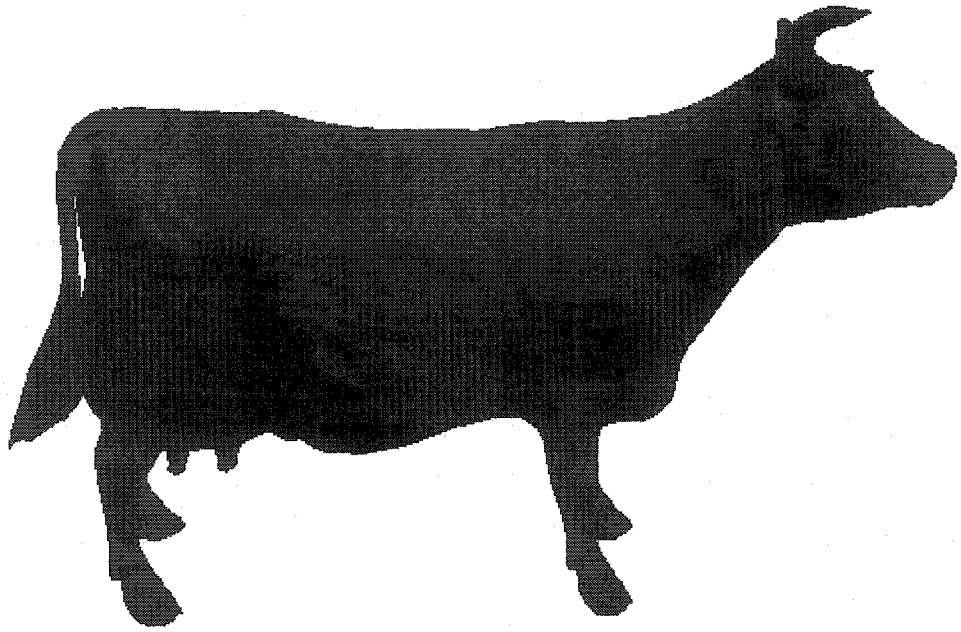


Figure 5-5 Cow mesh with 17412 quads and 17414 vertices

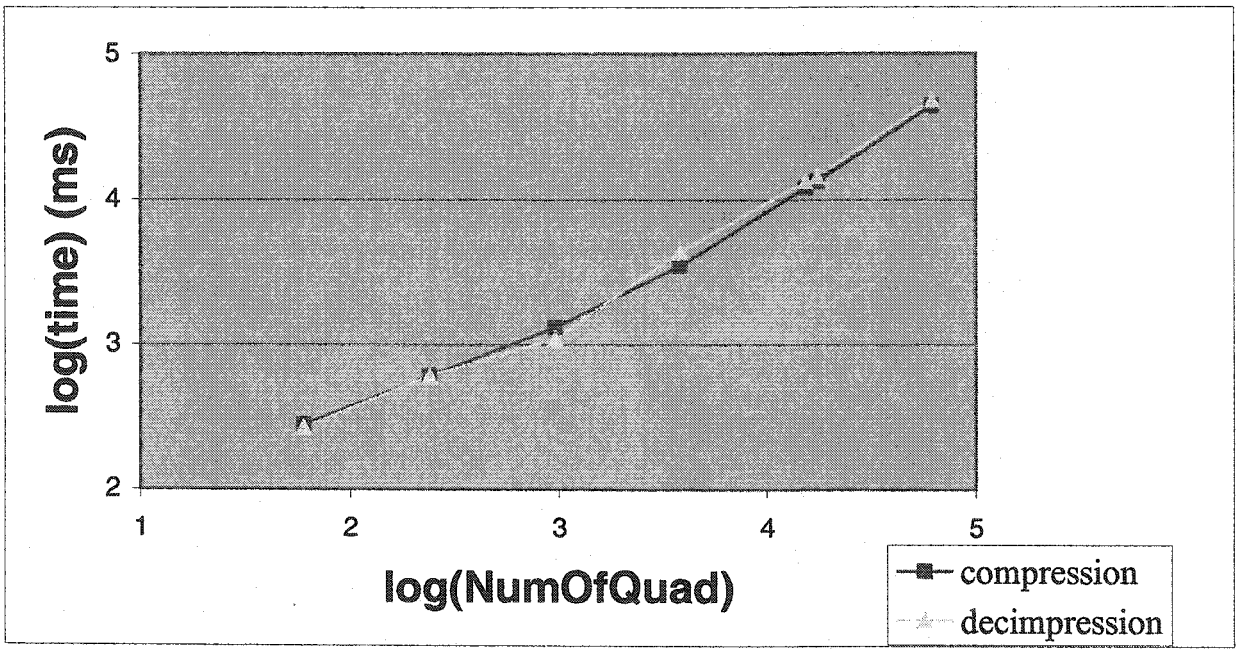


Figure 5-6 Compression/Decompression time over size of input (logarithmic)

Table 5-3 lists the overall performance of coding scheme I. From this table, we see the overall compression ratio increases with the increasing of input file size. This is because Huffman table is not increasing substantially with the increase of input size. But its size is relatively big compared with the binary files (operation.bin and vertex.bin) when the input file size is small.

5.3 Summary

This Chapter gives the implementation details of both quad mesh connectivity compression/decompression algorithms and vertex compression/decompression algorithms. Spheres and cow meshes are used for experiments. The experimental results illustrate the linear time compression/decompression processes and 2.67 bpv upper bound of the encoding schemes.

Chapter 6

Conclusions

This chapter gives a brief summary of the topics discussed in this thesis. Then, the major contributions of the thesis are presented. Finally, future areas of improvement for this thesis are discussed.

6.1 Summary

The field of geometry compression involves studies of how to succinctly represent a mesh. A mesh may be represented by its vertex data and its connectivity. Vertex data comprises coordinates of the associated normal vectors and textures. Connectivity captures the incidence relation between polygons of the mesh and their bounding vertices.

Vertex data may be compressed through various forms of vector quantization. Most approaches use predictors and encode corrections of predicted value. Both the encoder and decoder use the same prediction formula.

There are three different approaches of connectivity compression: efficient rendering, progressive transmission, and maximum compression.

Although triangle meshes are used most frequently and studied extensively, quadrilateral meshes are used a lot in scientific applications. Traditionally, the problem of connectivity compression of quadrilateral meshes is solved by triangulating the mesh first and then compressing it using triangle compression techniques. This strategy may introduce additional cost. Some researchers have attempted to compress polygon meshes without prior triangulation.

6.2 Major contributions

There are three major contributions achieved in this thesis, which have been illustrated in chapters 3,4 and 5. The following is a summary of these contributions:

Improved data structure for linear time encoding

In chapter 3, an improved data structure (Opposite Edge table) is proposed for compression and decompression of quad meshes in linear time. This data structure stores four opposite edge labels for each quad, and this information proves to be extremely useful to retrieve a quad which is adjacent to the current quad. This data structure is also easy to implement. While using traditional data structures, we need O(n) time to find adjacent quads of a quad, therefore we need $O(n^2)$ time to compress the quad mesh.

Extension of Spirale Reversi algorithm to quadrilateral mesh

Although Spirale Reversi decompression for non-triangle meshes has been mentioned by Kronrod and C. Gotsman [48], they never give a detailed explanation in their publication. This study, to the author's knowledge, presents the first detailed description of the Spirale Revesi decompression process of quadrilateral mesh.

Improved upper-bound coding

The best known upper-bound is 3.5 bpv for direct connectivity compression of quad meshes homeomorphic to a sphere. This has been proved by Kronrod and C. Gotsman [48]. Chapter 4 first presents a 3.0 bpv coding scheme for quad meshes, and then gives a new coding scheme which further improves the upper bound to 2.67 bits/quad.

Experimental results reported in Chapter 5 confirm these theoretical estimates very well.

6.3 Future work

The data structures and algorithms described in this thesis are capable of handling mesh homeomorphic to a sphere. A modification of the algorithms and data structure is needed to handle meshes with holes or handles.

Experimental results show that coding scheme III always has a very good performance for large meshes. This may be proven theoretically.

We conjecture that the upper bound on the coding scheme reported in Chapter 4 can be further improved to a value less than 2.67 bpv.

Reference

[1] P. Alliez and M. Desbrun, *Progressive Encoding for lossless Transmission of 3d Meshes*, SIGGRAPH 2001

[2] E.M. Arkin, M. Held, J.S.B. Mitchell, and S.S. Skiena, *Hamiltonian triangulations for fast rendering,* Lecture Notes in Computer Science, 855:36-57, 1994

[3] M. Bern, D. Epstein, *Quadrilateral meshing by circle Packing*, Proceedings of 6th International meshing Roundtable, Sandia National Laboratories, pp.7-20, October 1997

[4] F. Bossen, *On the art of Compressing three-dimensional Polygonal Meshes and their Associated Properties*, PhD Thesis, Ecole Polytechnique Federale de Lausanne, June 1999

[5] C. Bajaj, V. Pascucci and G. Zhuang, *Compression and Coding of Large CAD Models*, University of Texas Technical Report, 1998

[6] C. Bajaj, V. Pascucci and G. Zhuang, *Single Resolution Compression of Arbitrary Triangle Meshes with Properties,* Proceedings IEEE Data Compression Conference, March 29-31, 1999

[7] C. Bajaj, V. Pascucci and G. Zhuang, *Progressive compression and transmission of arbitrarily triangular meshes*, In IEEE Visualization `99, pp.307-316, 1999

[8] R. Bar-Yehuda and C. Gotsman, *Time/space tradeoffs for polygon mesh rendering,* ACM Transactions on Graphics, Vol.15, No.2, pp.141-152, April 1996

[9] M. Chow, *Optimized geometry compression for real-time rendering,* Proceedings on the IEEE Visualization '97.

[10] R.C. Chuang, A. Garg, X. He, M. Kao, and H. Lu, *Compact Encodings of Planar Graphics via Canonical Orderings and Multiple Parentheses*, Proceedings of 25th International Colloquium on Automata, Languages and Programming, pp.118-129, 1998

[11] D. Cohen-Or, D. Levin, and O. Remez, *Progressive compression of arbitrary triangular meshes*. In IEEE Visualization 99 Conference Proceedings, pp.67-72, 1999

[12] D. Cohen-Or, Y. Mann, and S. Fleishman, *Deep Compression for stream Texture Intensive Animations,* In Proceedings of Siggraph'99, 1999

[13] A. Crosnier and J. Rossignac, *Tribox-based simplification of three-dimensional objects*, Computer & Graphics, March 1999

[14] M. Deering, *Geometry Compression*, Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pp.13-20, August 1995.

[15] M. Deering and S. Nelson, *Leo: A System for cost Effective shaded 3D graphics*, Proceedings of SIGGRAPH '93, pp.101-108, August 1993

[16] M. Deering, S. Winner, B. Schediway, C. Duffy and N. Hunt, *The triangle Processor and Normal Vector Shader: A VLSI system for High Performance* Graphics, Proceedings of SIGGRAPH '88, pp. 21-30, August 1988

[17] M. Denny and C. Sohler, *Encoding a triangulation as a permutation of its point set.* In Proceedings of 9th Canadian Conference on Computational Geometry, pp. 13-20, 1997

[18] F.Evans, S. Skiena and A. Varshney. *Optimizing Triangle Stripes for Fast Rendering* IEEE Visualization 96 Proceedings, pp. 319-326, October 1996

[19] M. Garland and P. Heckbert, *Surface Simplification Using Quadric Error Metric*, SIGGRAPH 97

[20] M. Gross, L. Lippert and O. Staadt, *Compression Methods for Visualization*, Future Generation Computer Systems, Vol 15, No.1, pp.11-29, 1999

[21] S. Gumhold, S. Guthe and W. Straber, *Tetrahedral Mesh Compression with the Cut-Border Machine,* Proceedings of the conference on IEEE Visualization '99 pp..51-58, 1999

[22] S. Gumhold and Klei, *Compression of discrete multi-resolution models*, Technical Report WSI-98-1, Wilhelm-Schicjard-Institu fur Informatik, University of Tubingen, Germany, January 1998

[23] S. Gumhold and W. Strasser, *Real Time Compression of triangle mesh connectivity.* Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH '98, p133-140. 1998

[24] A. P. Gueziec, F. Bossen, and G. Taubin, and C.T. Silva, *Efficient Compression of Non-Manifold Polygonal Meshes*, IEEE Visualization'99, pp.73-80, San Francisco, October 1999

[25] I. Guskov, K. Vidime, W. Sweldens, and P. Schroder, *Normal Meshes*, SIGGRAPH 1999

[26] M. Hadwiger, *Geometry over Network Techniques: Mesh Simplification and Multiresolution Data Structures*, http://www.cg.tuwien.ac.at/studentwork/VisFoSe98/msh/

[27] X. He, M. Kao, and H. Lu, *Linear-Time Succinct Encodings of planar Graphs via Canonical Orderings*, SIAM Journal on Discrete Mathematics, Vol 12, No.3, pp.317-325, 1999

[28] H. Hoppe, *Progressive meshes*, SIGGRAPH 96, pp.99-108, 1996

[29] H. Hoppe, *View Dependent Refinement of Progressive Meshes,* Proceedings ACM SIGGRAPH '97, August 1997

[30] H. Hoppe, T. DeRose, T. Duchamp, J. McDonold, and W. Stuetzle, *Mesh optimization*, Proceedings SIGGRAPH '93, pp.19-26, August 1993

[31] F.M. Ingels, *Information and Coding Theory*, Intext, Scranton, Penn. 1971

[32] M. Isenburg, *Triangle Fixer: Edge-based Connectivity Compression*, European Workshop on Computational Geometry '2000, December 1999

[33] M. Isenburg, *Triangle Strip Compression,* Graphics Interface '2000, January, 2000

[34] M. Isenburg, S. Gumhold, and C. Gotsman, *Connectivity Shapes,* http://www.cs.unc.edu/~isenburg/connectivityshapes/

[35] M. Isenburg and J. Snoeyink, *Mesh Collapse Compression*, Proceedings of XII SIGGRAPH, pp.1-2, 1999

[36] M. Isenburg and J. Snoeyink, *Spirale Reversi: Reverse decoding of the Edgebreaker encoding*, 12th Canadian Conference on Computational Geometry, pp.247-253, August, 2000

[37] M. Isenburg and J. Snoeyink, *Compressing the Property Mapping of Polygon Meshes*

[38] M. Isenburg and J. Snoeyink, *Face Fixer: Compressing Polygon Meshes with Properties*, UNC Technical Report TR-00-04

[39] A. Itai and M. Rodeh, *Representations of Graphs*, Acta Informatica, No17, pp.215-219, 1982

[40] K. Keeler and J. Westbrook, *Short Encodings of Planar Graphics and Maps*, Discrete Applied Mathematics, No. 58, pp.239-252, 1995

[41] A. Khdakovsky, P. Schroder and W. Sweldens, *Progressive Geometry Compression*, SIGGRAPH 2000

[42] D. King and J. Rossignac, *Guaranteed 3.67V bit encoding of planar triangle graphs*, 11th Canadian Conference on Computational Geometry, pp.146-149, Vancouver, CA, August 15-18, 1999

[43] D. King and J. Rossignac, *Optimal Bit Allocation in Compressed 3D Models*, Journal of Computational Geometry, Theory and Applications, Vol. 14, No.1-3, pp.91-118, November 1999

[44] D. King, J. Rossignac, and A. Szmczak, *Connectivity Compression for Irregular Quadrilateral Meshes*, GVU Tech Report GVU-GIT-99-36

[45] D. King, J. Rossignac, and A. Szmczak, *An Edgebreaker-based efficient compression scheme for regular meshes*, 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick, August, 2000

[46] D.G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM Journal of Computing, 12(1): 28-35, 1983

[47] L. Kobbelt and H. Seidel, *Efficient Storage, Compression and Transmission of Complex polygonal 3D Models*, Computer Graphics International 1998, June 1998, Hannover, Germany

[48] B. Kronrod and C. Gotsman, *Efficient coding of non-triangular meshes*, In Proceedings of 16th European Workshop on Computational Geometry, pp24-26,2000

[49] D.T. Lee and F.P. Preparata, *Location of a point in a planar subdivision and its applications*, SIAM Journal on Computers, Vol. 6, pp.594-606, 1977

[50] D.A. Lelewer and D.S. Hirschberg, *Data Compression*, ACM Computing Surveys, Vol. 19, No. 3, September 1987

[51] J.E. Lengyel, *Compression of Time-Dependent Geometry*, proceedings of 1999 ACM Symposium on Interactive 3D Graphics, April, 1999, Atlanta, Georgia

[52] J. Li and C. J. Kuo, *Progressive Coding of 3D Graphic Models*, Proceedings of IEEE, pp.1052-1063, June 1998

[53] J. Li and C. J. Kuo, *A dual graph approach to 3D triangular mesh compression*, Contribution Document M3195, MPEG-4 San Jose Meeting, 1998

[54] J. Li and C. J. Kuo, *Compression of mesh connectivity by dual graph approach*, Contribution Document M3324, MPEG-4 Tokyo Meeting, 1998

[55] Y. Mann and D. Cohen-Or, *Selective Pixel Transmission for Navigation in Remote Environments*, Proceeding of Eurographics'97, Budapest, Hungary, September 1997

[56] W. Mark, L. McMillan and G. Bishop, *Post-rendering 3D warping, 1995 Symposium on Interactive 3D Graphics*, ACM Press, pp.7-16, April 1997

[57] W. Massey, *Algebraic Topology: An Introduction*, Harcourt, Brace&World Inc., 1967

[58] M. Muller-Hannemann, *Quadrilateral Mesh Generation in Computer-Aides Design*, PhD Thesis, Technischen Universitat Berlin, 1997

[59] J.I. Munro and V. Raman, *Succinct Representation of Balanced Parentheses, Static Trees and Planar Graphs*, Proceedings of 38th Annual IEEE Symposium on the Foundations of Computer Science, pp.118-126, 1997

[60] M. Naor, *Succinct representation of general unlabeled graphs*, Discrete Applied Mathematics, North-Holland, Vol.29, pp.303-307, 1990

[61] M.R. Nelson, LZW Data Compression, Dr. Dobb's Journal, October 1989

[62] R. Pajarola and J. Rossignac, *Squeeze: Fast and Progressive Decompression of Triangle Meshes*, Computer Graphics International Conference, Switzerland, June 2000

[63] R. Pajarola, J. Rossignac and A. Szymczak, *Implant Sprays: Compression of Progressive Tetrahedral Mesh Connectivity,* IEEE Visualization, October, 1999

[64] R. Pajarola and J. Rossignac, *Compressed Progressive Meshes*, IEEE Transactions on Visualization and Computer Graphics, Vol. 6, No. 1, pp.79-93, January-March 2000

[65] A. Rockwood, K. Heaton, and T. Davis, *Real-time Rendering of Trimmed Surfaces, Computer Graphics*, 23(30): 107-116, 1989

[66] R. Ronfard, and J, Rossignac, *Full-range approximation of triangulated polyhedra*, Proceeding of Eurographics'96, Computer Graphics Forum, pp.C-67, Vol. 15, No3, August 1996

[67] J. Rossignac, *Through the cracks of the solid modeling milestone*, From Object Modeling to Advanced Visual Communication, Eds. Coquillart, Strasser, Stucki, SpringerVerlag, pp. 1-75, 1995

[68] J. Rossignac, *The 3D Revolution: CAD Access for All!* , International Conference on Shape Modeling and Applications, Aizu-Wakamatsu, Japan, IEEE Computer Society Press, pp.64-70, March 3-6, 1997

[69] J. Rossignac, *Interactive exploration of distributed 3D datasets over Internet*, Computational Graphics International Congress '98, Hanover, pp.324-335, June 1998

[70] J. Rossignac, *Edgebeaker: Connectivity compression for triangle meshes*, IEEE Transactions on Visualization and Computer Graphics, Vol.5, No.1, pp.47-61, January-March 1999

[71] J. Rossignac and P. Borrel, *Multi-resolution 3D approximations for rendering complex scenes*, In B. Falcidieno and T. Kunii, editors, Geometric Modeling in Computer Graphics: Methods and Applications, pages 455-465, 1993

[72] J. Rossignca and D. Cardoze, *Matchmaker: Manifold Breps for non-manifold r-sets*, Technical Report: GIT-GVU-99-03, GVU center, Georgia Institute of Technology, October 1998.

[73] J. Rossignac, A. Safonova, and A. Szymczak, *3D Compression Made Simple: Edgebreaker on a Corner-Table*, Shape Modeling International Conference, Gemoa, Italy, May 2001

[74] J. Rossignac and A. Szymczak, *Wrap & Zip decompression of connectivity of triangle meshes compressed with Edgebreaker*, Journal of Computational Geometry, Theory and Applications, Vol. 14, Issue 1-3, pp.119-135, November 1999

[75] W.J. Schroeder, J. A. Zarge, and W.E. Lorensen, *Decimation of Triangle Meshes*, Siggraph'92 Proceeding, pp. 65-70

[76] Silicon Graphics Inc., *GL programming guide*, 1991

[77] J. Snoeyink and M. Kerveld, *Good orders for incremental (re)construction*, Proceeding of ACM Symposium on Computational Geometry, pp. 400-402, Nice, France, June 1997,

[78] J. Snoeyink and M. Kerveld, *Linear-time reconstruction of Delaunay triangulations with applications*, Proceedings of European Symposium of Algorithm, pp. 459-471, 1997

[79] D.M.Y. Sommerville, *An introduction to the geometry of N Dimensions*, Dutton Publications, New York, 1929

[80] O. Staadt, and M. Gross, *Progressive Tetrahedralizations*, IEEEE Visualization'98, pp.397-402, 1998

[81] O. Staadt, M. Gross, and R. Weber, *Multi-resolution Compression and Reconstruction*, Proceedings of IEEEE Visualization '97, pp. 337-346, 1997

[82] Sun Microsystems, Inc. Java 3D Guide. *Java 3D Geometry Compression API Specification*, http://www.sun.com:80/products/java-media/3D

[83] A. Szymczak and J. Rossignac, *Grow & Fold: Compression of Tetrahedral Meshes*, Proc. ACM Symposium on Solid Modeling, pp.54-64., June 1999

[84] G. Taubin, A signal processing approach to fair surface design, Computer Graphics (Proceedings of Siggraph), pp351-358, August, 1995,

[85] G. Taubin, Curve and surface smoothing without shrinkage, In Proceedings of the 5th International Conference on Computer Vision, pp.852-857, June, 1995

[86] G. Taubin, A. Gueziec, W. Horn and F. Lazarus, *Progressive Forest Split Compression*, Proceedings of Siggraph '98, pp. 123-132, 1998

[87] G. Taubin, W.P. Horn, F.Lazarus, and J. Rossignac, *VRML compression,* Proceedings of the IEEE, Vol. 96, No. 6, pp.1228-1243, June 1998

[88] Gabriel Taubin and Jarek Rossignac, *Geometry Compression Through Topological Surgery* ACM Transactions on graphics, Vol. 17, No. 2, pp.84-115, April 1998

[89] G. Taubin, G. Zhang, and G. Golub, *Optimal surface smoothing as filter design.* In Proceedings of the European Conference on Computer Vision, pp.283-292, April, Cambridge, UK, 1996

[90] Costa Touma and Craig Gotsman, *Triangle Mesh Compression,* Graphics Interface '98. Vancouver, Canada

[91] G. Turan, *On the Succinct Representation of Graphics*, Discrete Applied Mathematics, 8, pp.289-294, 1984

[92] W. Tutte, *A census of planar triangulations*, Canadian Journal of Mathematics, 14, pp.21-28, 1962

[93] W. Tutte, *The Enumerative Theory of Planar Graphs*, In Survey of Computational Theory, J.N. Srinivasan et al.(Eds.), North-Holland, 1973

[94] T. Welch, *A Technique for High-performance Data Compression*, Computer, Vol.17, No6, pp.8-19, June 1984

[95] I. H. Witten, R.M. Neal, and J.G. Cleary, *Arithmetic coding for data compression*, Communications of the ACM, 30(6):520-540, 1987

[96] M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide*, Addison Wesley, 1996

[97] J.Ziv and A. Lempel, *A Universal Algorithm for Sequential Data Compression*, IEEE Transactions on Information Theory, Vol. IT-23, No3, pp.337-343, May 1977

[98] *http: www.3dcompression.com*

[99] *http:www.britannica.com*

[100] *Http://www-graphics.stanford.edu/~sliang/CS448_win00/sliang-cs448b-contrib.html*

[101] E. Lee and H. Ko, *Vertex Data Compression For Triangle Meshes,* EUROGRAPHICS'2000, Volume 19,(2000), No.3

Vita Auctoris

Personal Information

Name: Quanbin Jing

Date of Birth: July 4th, 1970

Birthplace: Baishui county, Shaaxi, P.R.China

Education

B.Eng., Applied Chemistry

National University of Defence Technology, P.R.China, 1992

M.Eng., Chemical Engineering

National University of Singapore, Singapore, 2000

M.Sc., Computer Science

University of Windsor, Canada, 2002