A Novel Progressive Lossy-to-Lossless Coding Method for Mesh Models of Images

by

Xiao Feng
B.Sc., Macau University of Science and Technology, 2011

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

A Novel Progressive Lossy-to-Lossless Coding Method for Mesh Models of Images

by

Xiao Feng
B.Sc., Macau University of Science and Technology, 2011

Supervisory Committee

Dr. Michael D. Adams, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Alexandra Branzan Albu, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Michael D. Adams, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Alexandra Branzan Albu, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

A novel progressive lossy-to-lossless coding method is proposed for mesh models of images whose underlying triangulations have arbitrary connectivity. For a triangulation T of a set P of points, our proposed method represents the connectivity of T as a sequence of edge flips that maps a uniquely-determined Delaunay triangulation (i.e., preferred-directions Delaunay triangulation) of P to T. The coding efficiency of our method is highest when the underlying triangulation connectivity is close to Delaunay, and slowly degrades as connectivity moves away from being Delaunay. Through experimental results, we show that our proposed coding method is able to significantly outperform a simple baseline coding scheme. Furthermore, our proposed method can outperform traditional connectivity coding methods for meshes that do not deviate too far from Delaunay connectivity. This result is of practical significance since, in many applications, mesh connectivity is often not so far from being Delaunay, due to the good approximation properties of Delaunay triangulations.

Contents

List of Tables

List of Figures

List of Acronyms

DT	Delaunay triangulation
PDDT	preferred-directions Delaunay triangulation
LOP	local optimization procedure
CGAL	Computational Geometry Algorithms Library
SPL	Signal Processing Library
SPLEL	Signal Processing Library Extensions Library
PSNR	peak signal-to-noise ratio
MSE	mean squared error
FIFO	first-in first-out
LIFO	last-in first-out
IT	image tree
NF	non-finalization
ACNF	arithmetic-coding-non-finalization
ACF1	arithmetic-coding-finalization-1
ACF2	arithmetic-coding-finalization-2

ACKNOWLEDGEMENTS

This thesis would have never been written without the support and help of numerous people. Taking this opportunity, I would like to express my thankfulness to certain individuals in particular:

To my marvelous supervisor Dr. Michael D. Adams Thank you for your guidance, encouragement, and extreme patience to me. The past three years has really been a life-changing era for me. Embracing with the amazing image geometry processing world, I thank you for your meticulous and earnest teaching in C++ which really is an asset of my career. Thank you for granting me permission to apply intern jobs in the software industry, and concerning me even during my work term. Without your mentoring in my research project, triangulation-connectivity coding, this thesis wouldn't have been written. It is my honor to be your padawan in programming, and what you have taught me would continue influencing me in the future.

To my course instructors I would like to express my sincere gratitude to Dr. Wu-Sheng Lu, Dr. T. Aaron Gulliver, Dr. Alexandra Branzan Albu, and Dr. Bruce Kapron. Thanks for offering me such interesting lessons during my first year of graduate studies.

To my dearest friends Yiyi Xu and Li Ji, thank you for always being honest to me. The time we spent together, has always constituted my most cherishing memory. Changhao Guo, Wen Shi, Yongyu Dai, Yue Yin and Season Ji, my lovely little independent companions, Winter solstice, Chinese new year eve, lantern festival, and Friday night badminton, the past three years has flashed before my eyes. Thank you for always being there for me in my low moments.

I also wish to express my gratitude to Brian Ma, Ali Mostafavian, Yucheng Wang, Philip Baback Alipour, Howard Lu and Bricklen Anderson, for their kindness and inspiration. Hiteshi Sharma, Shan Luo, Yi Chen, Mengyue Cai, Di Lu, Jason Du, Yanqiao Zhang, Binyan Zhao, Ping Cheng, Qifei Wang, Bingxian Mu, Xiaotao Liu, Jessie Zhou, Leyuan Pan, Zheng Xu, Le Liang, Dan Han, Xia Meng, Yue Fang, Yue Tang and all other friends, it is my life time treasure to become friend with you.

To our fantastic faculty staffs Moneca Bracken, Janice Closson, Dan Mai, Lynn Barrett, Amy Rowe and Erik Laxdal, thank you for your help and assistance during my graduate studies.

To my most lovely family Without your understanding and being supportive, I could never come here at the first place. Thank you for loving me as your most precious gift.

DEDICATION

To my dearest grandmother and parents, thank you for offering me endless love with patience!

Chapter 1

Introduction

1.1 Mesh Modelling and Mesh Coding of Images

In recent years, there has been a growing interest in image representations that exploit the geometric structure inherent in images. Traditionally, a commonly used approach for image representation is based on uniform sampling. Due to the fact that images are nonstationary in the real world, this approach is far from optimal. The sampling density would inevitably be too low in the rapidly changing regions or too high in those regions of slow variation. To that end, image representations based on nonuniform sampling have drawn increasing attention from researchers.

By using nonuniform sampling, the position of sample points can be made adaptive to image content, allowing more accurate image representations to be obtained with fewer sample points. Furthermore, the geometric structure inherent in images (i.e., image edges) can be better captured by image representations based on nonuniform sampling. In practice, nonuniform sampling has proven to be beneficial in various applications including: feature detection [10], pattern recognition [32], computer vision [36], restoration [9], interpolation [39], and image/video coding [2, 33, 26, 1, 42, 11, 22, 5].

To date, many approaches to nonuniform sampling have been proposed, such as: inverse distance weighted methods [7, 20], radial basis function methods [7, 20], Voronoi and natural neighbor methods [7], and finite-element methods which includes triangle meshes [7, 20]. Among these classes of approaches, triangle meshes for nonuniform sampling have become quite popular. Such representations are known as **mesh models**. With a triangle mesh model of an image, the image domain is partitioned into triangles using a triangulation of sample points, and then over each face of the triangulation, an approximating

function is constructed. Numerous approaches based on mesh models have been proposed, including triangle meshes based on Delaunay triangulations [3, 45, 4], constrained Delaunay triangulations [40], and data-dependent triangulations [15, 17, 16, 6, 28].

In order to be able to efficiently store and communicate (triangle) mesh models of images, we need effective coding methods for such data. To code a mesh, two types of information must be conveyed: 1) mesh geometry (i.e., vertices of mesh), and 2) mesh connectivity (i.e., how vertices in the underlying triangulation of a mesh are connected by edges).

1.2 Historical Perspective

As mentioned earlier, coding mesh models of images requires the compression of geometry and connectivity data. Over the years, a number of methods [13, 1] have been developed to code arbitrarily-sampled image data. Demaret and Iske proposed a scattered data coding scheme in [13], which codes data using an octree. The arbitrary sampled image data is viewed as a collection I of points in a 3-D volume. Each point $(x_i, y_i, z_i) \in I$ represents a sample point (x_i, y_i) and its corresponding sample value z_i. By using an octree data structure to represent I, this scheme provides a means to efficiently code the information in I. In [1], Adams proposed the so called **image-tree (IT) method**, which is based on a recursive quadtree partitioning of the image domain along with an iterative averaging process for sample data. The image dataset is represented by a tree-based data structure called an image tree. By efficiently coding the information in an image tree using the top-down traversal of the tree, this method is able to provide the progressive-coding functionality. In comparison with the scattered data coding scheme, the IT method achieves the functionality of progressive coding as well as more efficient lossless coding.

Over the last 20 years, numerous methods have been proposed to efficiently encode connectivity data [34, 23]. In the absence of any assumptions about mesh connectivity, the theoretical lower bound for connectivity coding established by Tutte [41] is $\log_2(256/27) \approx 3.245$ bits/vertex given a triangulation with a sufficiently large number of vertices. Based on the work in [34, 23], current connectivity coding methods can typically encode meshes using around 3.67 bits/vertex. In [34], Rossignac proposed the well-known edgebreaker algorithm to compress the connectivity of simple triangle meshes with a bound of 4 bits/vertex. The edgebreaker algorithm performs a series of steps to traverse the faces of the mesh in a depth-first order, nominally moving from a face to a neighboring one in each step. At each step, a label from the set $\{C, S, R, L, E\}$ is coded to depict the topological connec-

tion between the current triangle face in the mesh and the boundary of the remaining part of the mesh. In particular, each C operation is coded with a single bit and each one of the S, R, L and E operations is coded with three bits (e.g., $\{C \rightarrow 0,\ S \rightarrow 100,\ R \rightarrow 101,\ L \rightarrow 110,\ E \rightarrow 111\}$). By modifying the edgebreaker algorithm in [23], King and Rossignac improved the coding performance of the method to a guaranteed 3.67 bits/vertex. More specifically, three new substitute codes are proposed for encoding the labels in the string of operations that captures the connectivity of the mesh. By choosing the code with the minimum connectivity coding cost of the three substitute codes, this improved edgebreaker algorithm is able to guarantee a cost no worse than 3.67 bits/vertex.

In data compression [1, 5, 21, 38], entropy coding schemes are employed to exploit statistical redundancy and represent information more compactly. One of the most commonly used entropy coding methods is arithmetic coding [43]. In practice, a very popular kind of arithmetic coding is binary arithmetic coding [44] that takes only binary source alphabets. Other than arithmetic coding, many other entropy coding schemes have been developed over the years, such as universal coding [27], including Fibonacci coding [19] and Elias gamma coding [18]. Several of these entropy coding methods are of interest for the mesh-coding work presented in this thesis.

1.3 Overview and Contributions of This Thesis

In this thesis, we explore the coding of mesh models of images with arbitrary connectivity. In particular, our work has focused on the development of effective techniques for coding mesh connectivity.

One contribution of this thesis is the proposal of a new framework for coding mesh models of images with arbitrary connectivity, which extends the highly efficient IT method [1] by adding to it a means for coding mesh connectivity. As our proposed framework has several free parameters, we studied how different choices of those free parameters affect coding efficiency, leading us to recommend a particular set of choices. The other contribution of this thesis is that it proposes a new progressive mesh-coding method derived from our framework by employing the recommended set of choices. As we shall see, the proposed method is shown to outperform more traditional connectivity coding approaches for meshes whose connectivity is sufficiently close to Delaunay.

The remainder of this thesis is organized into three chapters as well as one appendix. The three chapters provide the core content of the thesis while the appendix provides supplemental information about software developed in our work.

In Chapter 2, some essential background information is introduced to facilitate the understanding of the work in this thesis. First, some of the basic notation and terminology used herein are presented. This is then followed by an introduction to various concepts from computational geometry, such as triangulations and preferred-directions Delaunay triangulations [14]. Thereafter, a well known triangulation connectivity optimization method, called the local optimization procedure (LOP) [25], is introduced. Following this, concepts related to mesh models of images are presented. Finally, several entropy coding methods are introduced that are relevant to the work in this thesis.

Chapter 3 presents a new framework for coding triangle meshes models of images with arbitrary connectivity. Our framework has several free parameters. Then, we study how different choices of these free parameters affect coding performance. This leads to a recommended set of choices for use in our framework. Following this, a new progressive mesh-coding method is proposed using this recommended set of choices. The performance of the proposed method is evaluated by comparison with a simple baseline coding scheme. Through experimental results, our proposed method is shown to outperform the baseline approach by a margin of up to 11.56 bits/vertex (for connectivity coding). Moreover, for meshes whose connectivity is sufficiently close to Delaunay, our proposed method is demonstrated to be likely to be able to outperform more traditional connectivity coding approaches. Furthermore, unlike many coding schemes, our proposed method has progressive coding functionality, which can be beneficial in many applications.

Finally, Chapter 4 summarizes the key results of the thesis and provides recommendations for future work.

As supplemental information, a description of the software developed in our research is provided in Appendix A. Some examples of how to use this software are included in this appendix.

Chapter 2

Preliminaries

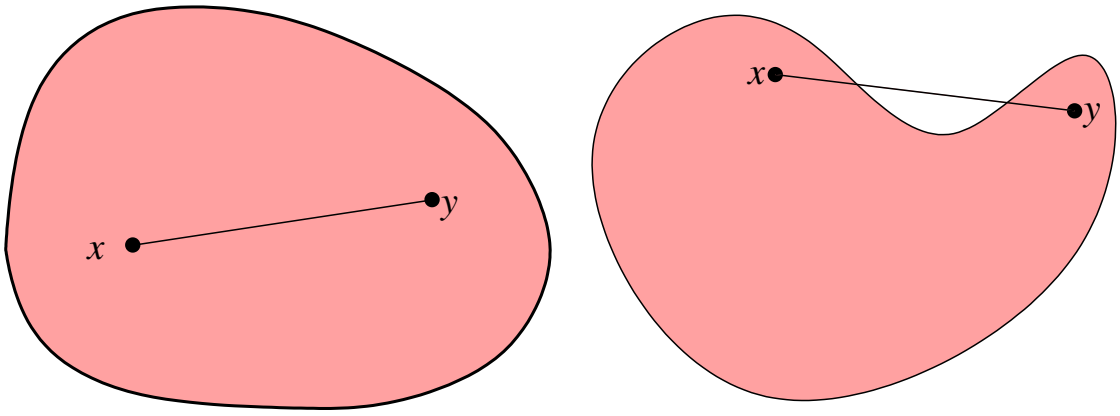
2.1 Overview

In this chapter, some fundamental background information is introduced to promote a better understanding of the work presented in this thesis. To begin, we introduce some of the basic notation and terminology used herein. Then, some fundamental concepts from computational geometry are provided. Following this, we present a description of the well-known connectivity optimization method for triangulations known as the Lawson local optimization procedure (LOP) [25]. Next, we discuss mesh modeling of images based on triangulations. Lastly, some basic background on entropy coding is presented.

2.2 Notation and Terminology

Before proceeding further, some basic notation and terminology are introduced. In this thesis, we denote the sets of integers and real numbers as $\mathbb{Z}$ and $\mathbb{R}$, respectively. For $a,b \in \mathbb{R}$, the expressions (a,b), $[a,b]$, $[a,b)$, and $(a,b]$ denote the sets $\{x \in \mathbb{R} : a < x < b\}$, $\{x \in \mathbb{R} : a \leq x \leq b\}$, $\{x \in \mathbb{R} : a \leq x < b\}$, and $\{x \in \mathbb{R} : a < x \leq b\}$, respectively. For $x \in \mathbb{R}$, $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the largest integer no greater than x, and the smallest integer no less than x, respectively. For a set S, the cardinality of S is denoted $|S|$. Similarly, the length of a (finite-length) sequence S is denoted $|S|$.

For two line segments $\overline{a_0a_1}$ and $\overline{b_0b_1}$, we say that $\overline{a_0a_1} < \overline{b_0b_1}$ in lexicographic order if and only if either 1) $a'_0 < b'_0$, or 2) $a'_0 = b'_0$ and $a'_1 < b'_1$, where $a'_0 = \min(a_0,a_1)$, $a'_1 = \max(a_0,a_1)$, $b'_0 = \min(b_0,b_1)$, and $b'_1 = \max(b_0,b_1)$, and $\min(a,b)$ and $\max(a,b)$ denote the least and greatest of the points a and b in xy-lexicographic order, respectively.



(a) (b)

Figure 2.1: Examples of a (a) convex set and (b) nonconvex set.

2.3 Computational Geometry

In what follows, we introduce some concepts from computational geometry. This includes concepts such as triangulations, Delaunay triangulations, preferred-directions Delaunay triangulations [14], and edge flips.

To begin, we present the definitions of a convex set and convex hull, which are needed in order to define the concept of a triangulation.

Definition 2.1 (Convex set)**.** *A set P of points in $\mathbb{R}^2$ is said to be **convex** if and only if for every pair of points $x, y \in P$, every point on the line segment $\overline{xy}$ is contained in P.*

The definition of a convex set is illustrated in Figure 2.1. In particular, the set P of points denoted by the shaded area in Figure 2.1(a) is convex since every point on the line segment formed by any pair of points x and y in P is also contained in P. On the contrary, in Figure 2.1(b), the set P of points denoted by the shaded region is not convex due to the fact that part of the line segment $\overline{xy}$ is not within the shaded region.

Definition 2.2 (Convex hull)**.** *The **convex hull** of a set P of points in $\mathbb{R}^2$ is the intersection of all convex sets containing P (i.e., it is the smallest convex set that contains P).*

To illustrate the above definition, we consider an example in Figure 2.2. For the set P of points shown in Figure 2.2(a), the convex hull of P is the shaded area presented in Figure 2.2(b).

With the definition of the convex hull introduced, now we are ready to present the definition of a triangulation [30, 8], which serves as an essential concept in this thesis.

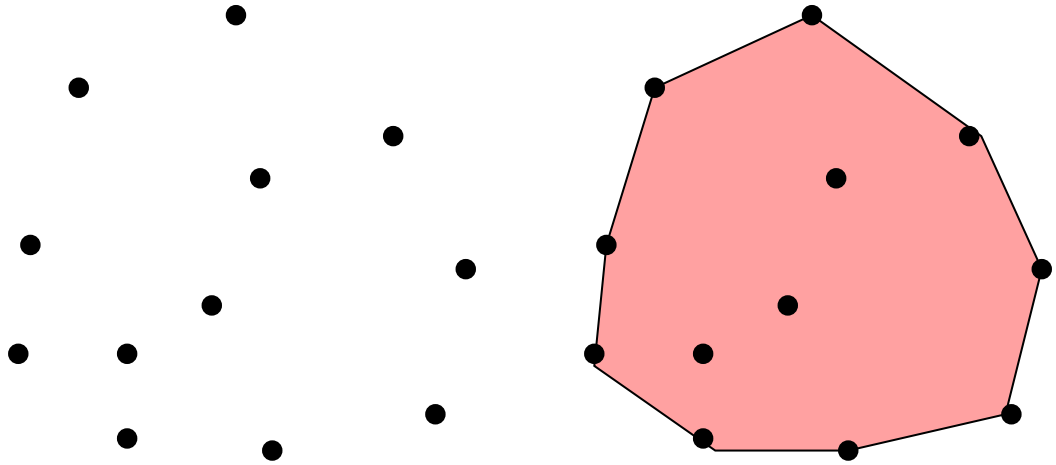


Figure 2.2: Convex hull examples. (a) A set P of points, and (b) the convex hull of P.

Definition 2.3 (Triangulation)**.** *A **triangulation** T of the set P of points in $\mathbb{R}^2$ is a set T of non-degenerate open triangles that satisfies the following conditions:*

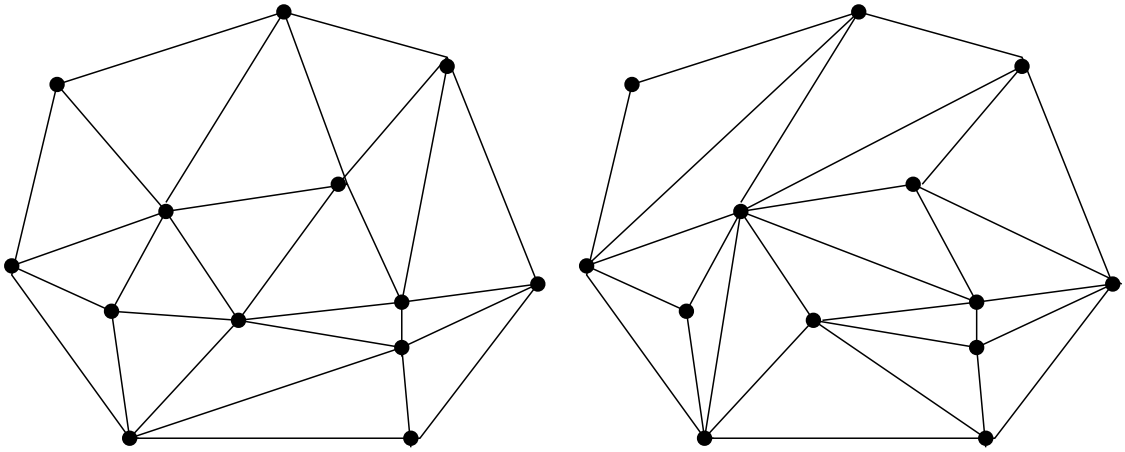
1. *the union of all triangles in T is the convex hull of P;*
2. *the set of the vertices in all triangles of T is P; and*
3. *the interiors of any two triangle faces in T do not intersect.*

In other words, a triangulation T of a set P of points can be viewed as a subdivision of the convex hull of P into a set of triangles such that any two triangles do not overlap with each other. As a matter of notation, the sets of all vertices and edges in a triangulation T are denoted as vertices(T) and edges(T), respectively.

Examples of valid triangulations are shown in Figure 2.3. In particular, two different triangulations of a set P of points are illustrated in Figures 2.3(a) and (b). As we can observe from these figures, although the same set of points have been used to construct the triangulations, the connectivities of these triangulations (i.e., how vertices are connected by edges) are quite different.

Over the years, various types of triangulations have been proposed. One commonly used type of triangulation is the Delaunay triangulation, which was introduced by Delaunay in 1934 [12]. Before presenting the definition of a Delaunay triangulation, we must first introduce the definition of a circumcircle, which is given below.

Definition 2.4 (Circumcircle of a triangle)**.** *The **circumcircle** of a triangle is defined as the unique circle passing through all three vertices of the triangle.*



(a) (b)

Figure 2.3: Examples of triangulations of a set P of points. (a) A triangulation of P, and (b) the other triangulation of P.

With the definition of the circumcircle of a triangle in mind, we can now define the Delaunay triangulation as follows:

Definition 2.5 (Delaunay triangulation (DT))**.** *A triangulation T of the set P of points is said to be **Delaunay** if no point in P is strictly in the interior of the circumcircle of any triangle face of T.*

An example of a Delaunay triangulation is shown in Figure 2.4 with the circumcircles of the faces in the triangulation displayed using dashed lines. As we can observe from the figure, it is clear that no vertices are strictly inside any circumcircles of the faces of the triangulation. Therefore, the triangulation is Delaunay.

The Delaunay triangulation of a set P of points is not necessarily unique. In particular, the Delaunay triangulation is only guaranteed to be unique if no four points in P are cocircular. In Figure 2.5, we present two Delaunay triangulations of the set P of points. As one can see, four cocircular points are presented in P and the Delaunay triangulation of P is not unique. In the case that a set of points is a subset of the integer lattice, many cocircular points could be present resulting in multiple Delaunay triangulations for the set of points. In fact, some methods have been proposed for uniquely choosing one of all possible Delaunay triangulations of a point set. One such method is the preferred-directions scheme [14]. The unique (Delaunay) triangulation produced by this scheme is known as the **preferred-directions Delaunay triangulation (PDDT)**.

As a matter of terminology, an edge e in a triangulation is said to be **flippable** if e has two incident faces (i.e., is not on the triangulation boundary) and the union of its two inci-

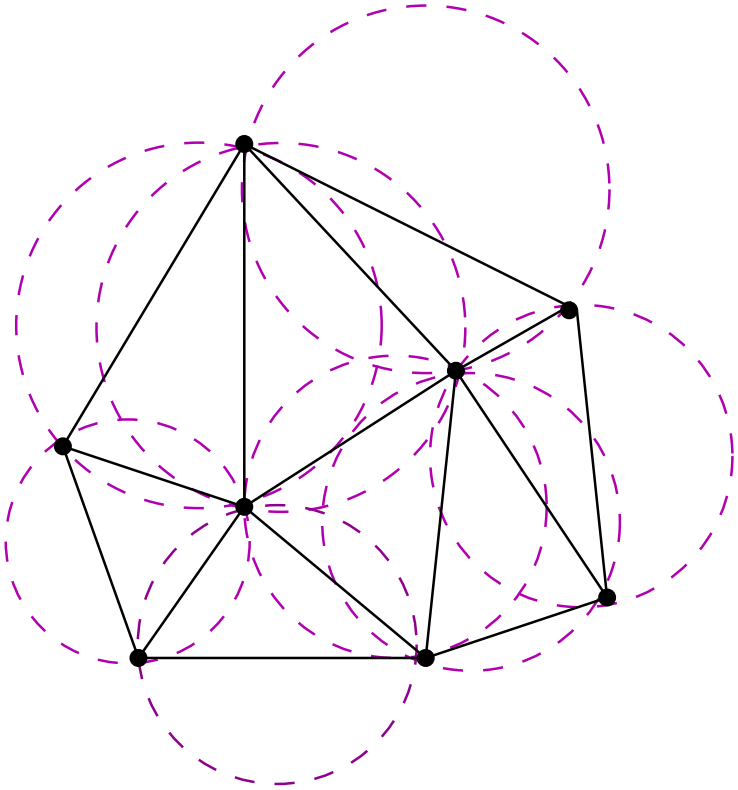


Figure 2.4: An example of a Delaunay triangulations where the circumcircle of each triangle in the Delaunay triangulation is specified.

dent faces is a strictly convex quadrilateral Q. To illustrate the definition of a flippable edge, we present two examples in Figures 2.6(a) and (b). In each figure, part of a triangulation associated with the quadrilateral $v_i v_j v_k v_\ell$ is shown. In Figure 2.6(a), we can clearly observe that the edge $\overline{v_k v_\ell}$ is flippable as $\overline{v_k v_\ell}$ is the diagonal of a strictly convex quadrilateral. On the other hand in Figure 2.6(b), the edge $\overline{v_k v_\ell}$ is not flippable as a non-convex quadrilateral is formed by two faces incident on $\overline{v_k v_\ell}$.

Next, we introduce the definition of an edge flip which is a fundamentally important operation for transforming triangulations. For any flippable edge e, an **edge flip** operation deletes e from the triangulation, and replaces it with the other diagonal of the convex quadrilateral formed by the two faces incident on e. An example of an edge flip is shown in Figure 2.7. Through applying the edge flip, one triangulation with flippable edge $\overline{v_i v_j}$ in Figure 2.7(a) is transformed into another triangulation with edge $\overline{v_k v_l}$ in Figure 2.7(b). As it turns out, every triangulation of a set of points can be transformed into every other triangulation (of the same set of points) by a finite sequence of edge flips [24, 31]. Consequently, the edge flip operation is quite important and forms the basis for many algorithms

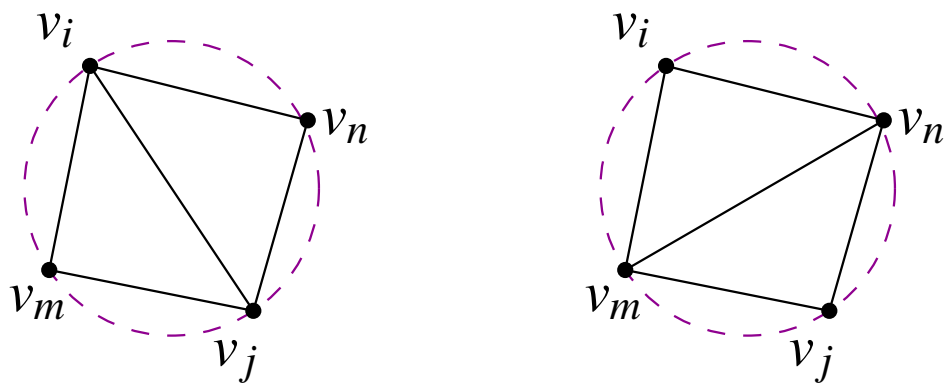


Figure 2.5: Examples of two different Delaunay triangulations of a set P of points. (a) A Delaunay triangulation of P, and (b) the other Delaunay triangulation of P.

involving triangulations.

2.4 Lawson Local Optimization Procedure (LOP)

Motivated by the fact that any two triangulations of the same set of points are reachable via a finite sequence of edge flips, Lawson proposed a scheme for optimizing the connectivity of triangulations based on edge flips, known as the Lawson local optimization procedure (LOP) [25]. With the LOP, one must define an optimality criterion for edges. An edge e being optimal means that the flipped counterpart of e is not preferred over e (i.e., the triangulation obtained by flipping e is not more desirable than the original triangulation with e). The LOP deems a triangulation optimal if every flippable edge in the triangulation is optimal. Essentially, the LOP is an algorithm that simply keeps applying edge flips to flippable edges that are not optimal until all flippable edges are optimal (i.e., the triangulation is optimal).

In more detail, the LOP works as follows. A priority queue, called the **suspect-edge queue**, is used to record all edges whose optimality is suspect (i.e., uncertain). Initially, all flippable edges in the triangulation are placed in the suspect-edge queue. Then, the following steps are performed until the suspect-edge queue is empty:

1. remove the edge e from the front of the suspect-edge queue;

2. test e for optimality;

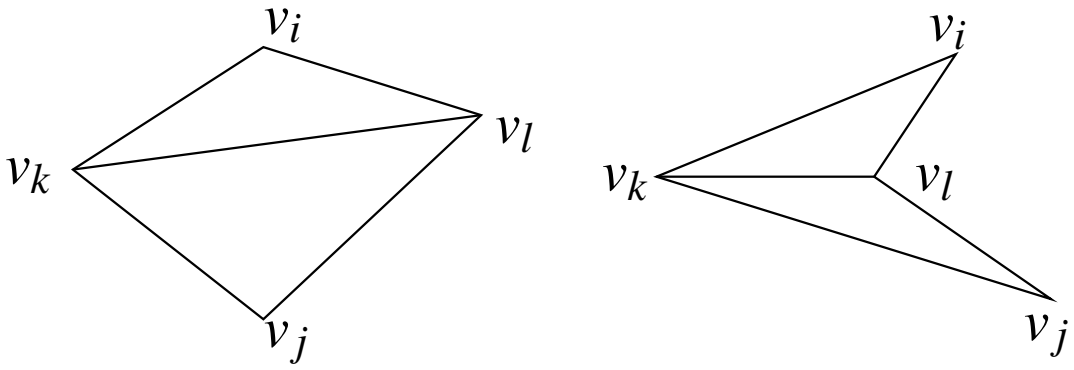


Figure 2.6: Examples of flippable and nonflippable edges in triangulations. A (a) flippable edge $\overline{v_k v_\ell}$ and (b) unflippable edge $\overline{v_k v_\ell}$ in the part of the triangulations associated with quadrilateral $v_i v_j v_k v_\ell$.

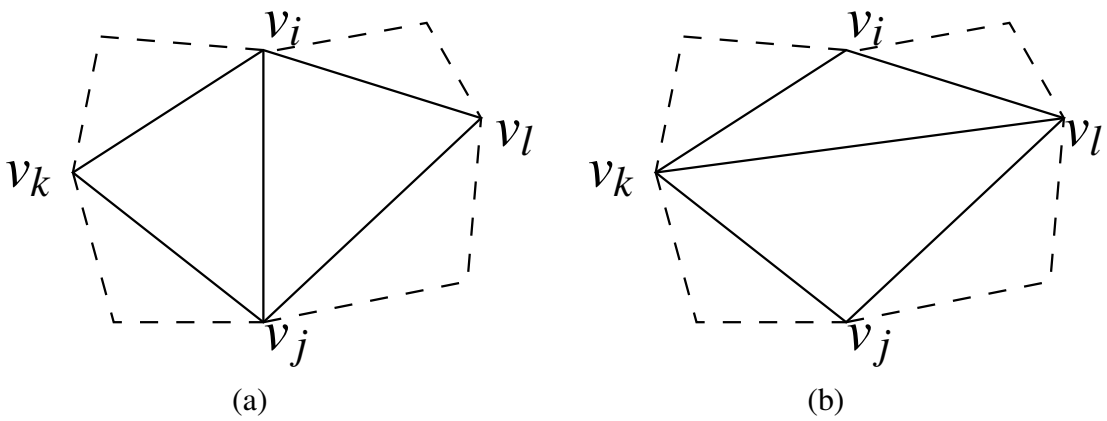


Figure 2.7: An edge flip. The part of the triangulation associated with quadrilateral $v_i v_j v_k v_\ell$ (a) before and (b) after applying an edge flip to e (which replaces e by e').

3. if e is not optimal, apply an edge flip to e and place any newly suspect edges (resulting from the edge flip) on the suspect-edge queue.

When the iteration terminates, the resulting triangulation is optimal. The LOP can be used to compute the PDDT of a point set by providing any valid triangulation as input to the LOP and specifying the PDDT criterion [14] (to be discussed in detail shortly) as the edge optimality criterion for the LOP. In this case, the LOP will yield the PDDT as output. Although the PDDT produced is unique, the particular sequence of edge flips performed by the LOP is not, and will depend on the specific priority function used for the suspect-edge queue.

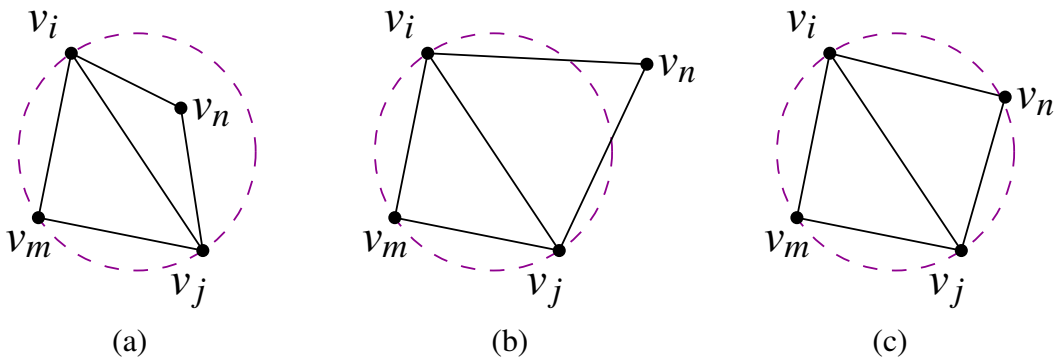


Figure 2.8: Examples of nonoptimal and optimal edges in triangulations according to the Delaunay edge optimality criterion. A (a) nonoptimal edge $\overline{v_i v_j}$, and (b) as well as (c) optimal edge $\overline{v_i v_j}$ in the part of the triangulation associated with quadrilateral $v_i v_j v_m v_n$.

2.4.1 Delaunay and PDDT Edge Optimality Criteria

In what follows, we introduce two edge optimality criteria, each of which can be used as an optimality criterion for the LOP. First, we consider the Delaunay edge optimality criterion.

Delaunay edge optimality criterion. The Delaunay edge optimality criterion is used in the LOP in order to obtain a triangulation that is Delaunay. With this criterion, the optimality of an edge $\overline{v_i v_j}$ in the part of the triangulation associated with quadrilateral $v_i v_j v_m v_n$ is defined as follows:

1. If the point v_n is strictly in the interior of the circumcircle of $\triangle v_i v_j v_m$, as Figure 2.8(a) shows, $\overline{v_i v_j}$ is not optimal (i.e., the diagonal $\overline{v_m v_n}$ is preferred over $\overline{v_i v_j}$).
2. If the point v_n lies strictly outside the circumcircle of $\triangle v_i v_j v_m$, as Figure 2.8(b) illustrates, $\overline{v_i v_j}$ is optimal (i.e., $\overline{v_i v_j}$ is preferred over $\overline{v_m v_n}$).
3. If the point v_n falls on the circumcircle of $\triangle v_i v_j v_m$, as illustrated in Figure 2.8(c), $\overline{v_i v_j}$ and $\overline{v_m v_n}$ are both deemed optimal (i.e., neither choice is preferred).

When the Delaunay edge optimality criterion is used with the LOP, the LOP is guaranteed to produce a triangulation that is Delaunay. In passing, we note that the Delaunay triangulation of a set P of points (as computed by the LOP) is not necessarily unique due to case 3 above.

PDDT edge optimality criterion. Sometimes a unique Delaunay triangulation may be desired. One method to obtain such triangulations is to use the PDDT edge optimality criterion [14] in the LOP. In general, this criterion augments the Delaunay edge optimality criterion by modifying case 3 (i.e., the case where the point v_n falls on the circumcircle of $\triangle v_i v_j v_m$). In particular, the PDDT edge optimality criterion modifies this case by utilizing two non-zero direction vectors d_1 and d_2, which are neither parallel nor orthogonal to each

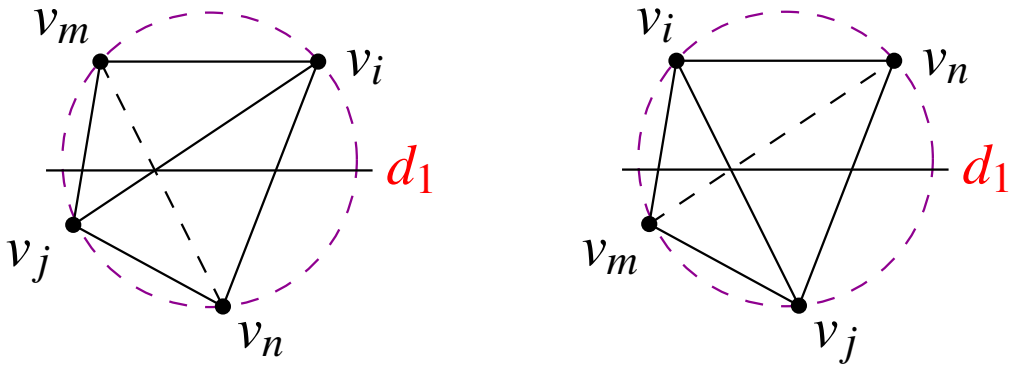


Figure 2.9: Examples of nonoptimal and optimal edges in triangulations according to the PDDT edge optimality criterion. An (a) optimal edge $\overline{v_i v_j}$ and (b) nonoptimal edge $\overline{v_i v_j}$ in the part of the triangulation associated with quadrilateral $v_i v_j v_m v_n$.

other, to determine a uniquely preferred edge direction. More specifically, case 3 is changed to the following where $\theta(e,d)$ denotes the angle that the edge e makes with the vector d:

If the point v_n falls on the circumcircle of $\triangle v_i v_j v_m$:

(a) if $\theta(\overline{v_i v_j}, d_1) < \theta(\overline{v_m v_n}, d_1)$, as shown in Figure 2.9(a), $\overline{v_i v_j}$ is optimal (i.e., $\overline{v_i v_j}$ is preferred over $\overline{v_i v_j}$);

(b) if $\theta(\overline{v_i v_j}, d_1) > \theta(\overline{v_m v_n}, d_1)$, as shown in Figure 2.9(b), $\overline{v_i v_j}$ is not optimal (i.e., $\overline{v_m v_n}$ is preferred over $\overline{v_i v_j}$);

(c) if $\theta(\overline{v_i v_j}, d_1) = \theta(\overline{v_m v_n}, d_1)$, $\overline{v_i v_j}$ is optimal if $\theta(\overline{v_i v_j}, d_2) < \theta(\overline{v_m v_n}, d_2)$ and not optimal if $\theta(\overline{v_i v_j}, d_2) > \theta(\overline{v_m v_n}, d_2)$.

(Note that we cannot have both $\theta(\overline{v_i v_j}, d_1) = \theta(\overline{v_m v_n}, d_1)$ and $\theta(\overline{v_i v_j}, d_2) = \theta(\overline{v_m v_n}, d_2)$ since d_1 and d_2 are neither parallel nor orthogonal.) In our work, we choose the direction vectors as $d_1 = (1,0)$ and $d_2 = (1,1)$.

2.5 Mesh Model of Images

As mentioned earlier, image representations based on triangle meshes are of great practical interest. Now, we introduce some background information related to (triangle) mesh models of images.

Consider an image function ϕ defined on $\Gamma = [0, W-1] \times [0, H-1]$ and sampled at points in $\Lambda = \{0,1,...,W-1\} \times \{0,1,...,H-1\}$ (i.e., a rectangular grid of width W and height H). In the context of our work, a mesh model of ϕ is characterized by:

1. a set $P = \{p_i\}$ of sample points (where $P \in \Lambda$);

2. a triangulation of P; and

3. a set $Z = \{z_i\}$ of function values for ϕ at each point in P (i.e., $z_i = \phi(p_i)$).

In passing, we note that the set P must include all of the extreme points on the convex-hull of Γ (i.e., the four corners of the image bounding box) so that the triangulation of P covers all of Γ.

The above mesh model is associated with a continuous piecewise-linear function $\hat{\phi}$ that approximates ϕ, where $\hat{\phi}$ is determined as follows. Over each face f in the triangulation T of a set P of sample points in the mesh model, $\hat{\phi}$ is defined as the unique linear function that interpolates ϕ at the three vertices of f. Therefore, the approximating function $\hat{\phi}$ is continuous and interpolates ϕ at each point in P.

Typically, subject to a constraint on maximum model size, the mesh model is chosen to minimize the **mean-squared error (MSE)** ε between the original image ϕ and the approximated image $\hat{\phi}$ as given by

$$\varepsilon = |\Lambda|^{-1} \sum_{p \in \Lambda} \left(\hat{\phi}(p) - \phi(p)\right)^2. \tag{2.1}$$

For convenience, MSE is normally expressed in terms of **peak-signal-to-noise ratio (PSNR)**, which is defined as

$$\text{PSNR} = 20 \log_{10} \left[(2^{\rho} - 1)/\sqrt{\varepsilon}\right], \tag{2.2}$$

where ρ is the sample precision in bits/sample. Essentially, the PSNR represents the MSE relative to the dynamic range of the data using a logarithmic scale, with a higher PSNR corresponding to a lower MSE.

Figure 2.10 illustrates the mesh modelling process for images. The image in Figure 2.10(a) can be viewed as a surface with the brightness of the image corresponding to the height of the surface above the plane as Figure 2.10(b) illustrates. In Figure 2.10(c), the image domain is partitioned by a triangulation T of a set of sample points. The corresponding (triangle) mesh model of the image is shown in Figure 2.10(d). Furthermore, a reconstructed (raster) image can be generated from the triangle mesh in Figure 2.10(d) to yield the result shown in Figure 2.10(e).

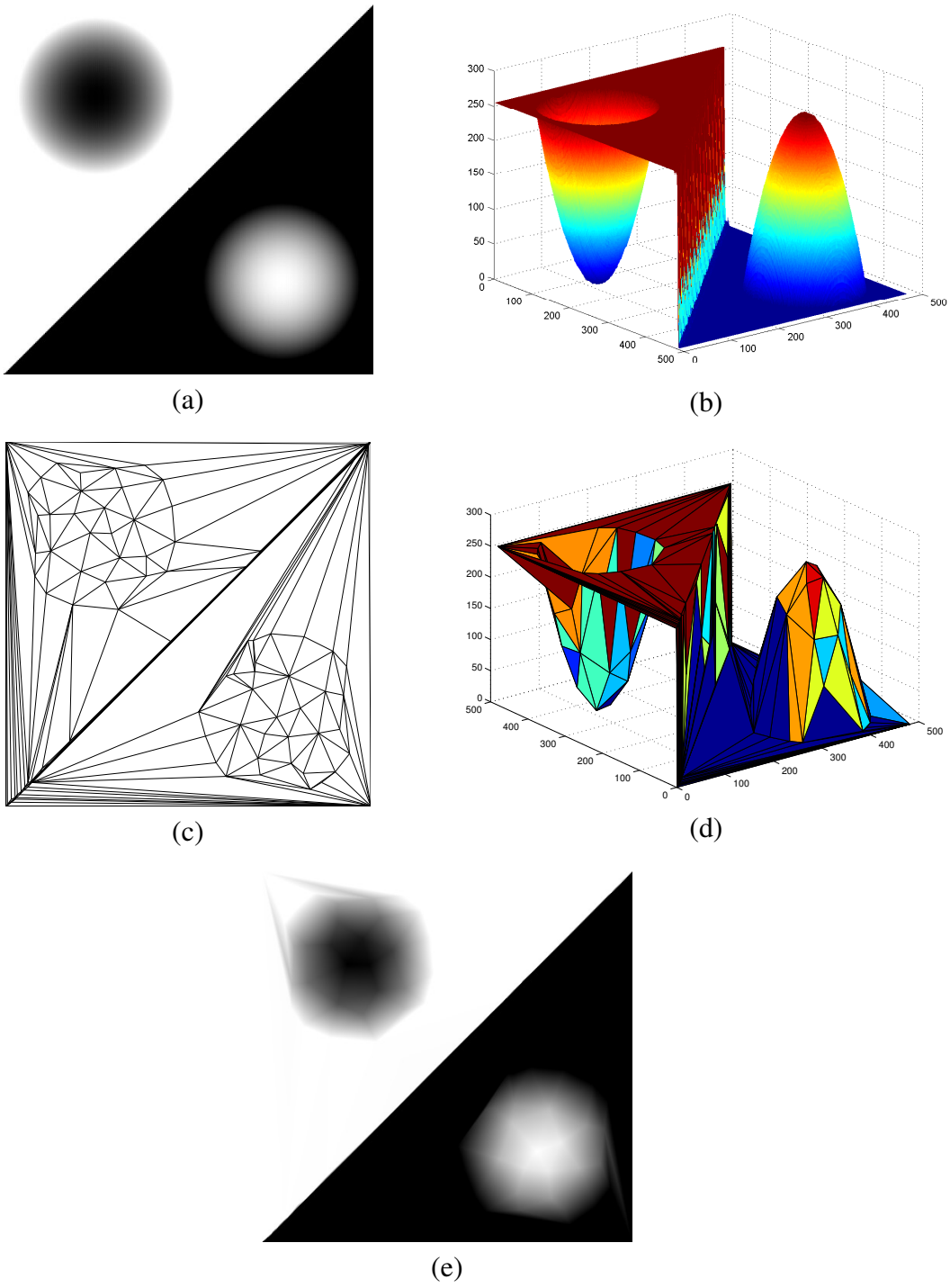


Figure 2.10: Mesh modelling of an image. (a) The original image, (b) the image modelled as surface (c) triangulation of the image domain, (d) resulting triangle mesh and (e) the reconstructed image

2.6 Entropy Coding

In addition to all the background discussed previously, we need to introduce several entropy coding schemes which are of great importance for our work in this thesis. In information theory, an entropy encoding scheme is a kind of lossless data compression that exploits the statistical redundancy of the source of the data so that the encoded data can be represented using fewer bits. Before presenting several specific entropy coding schemes, we first introduce the definition of the entropy which is important to such schemes.

Entropy is a measure of uncertainty. In particular, the **entropy** H of a discrete random variable X, with possible values $\{x_1, x_2, \ldots, x_n\}$ and probability function $p(x_i)$, is defined in [37] as

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i). \tag{2.3}$$

For the given variable X with n possible values, the entropy H attains a maximum of $\log_2(n)$ when each value is equiprobable. Furthermore, Shannon's source coding theorem [37, 29] states that a lower bound on the code rate (i.e., average number of bits per symbol) of the source message is given by the entropy of the source. Hence, lossless data compression schemes generally aim to achieve a rate as close as possible to the entropy of the source. In the sections that follow, we will present several entropy coding schemes relevant to our work in this thesis.

2.6.1 Arithmetic Coding

Among various entropy coding schemes proposed over the years, one of the most commonly utilized schemes is arithmetic coding [44]. Generally speaking, arithmetic coding represents the source message as an interval in $[0,1)$. As the source message becomes longer, a shorter interval is needed for the representation, leading to a growth in the number of bits needed to specify the interval. Each successive symbol coded reduces the size of the interval in accordance with the symbol's probability. Furthermore, when more likely symbols are coded, the interval range is reduced less significantly than when coding less likely symbols, leading to fewer bits being added to specify the range. Initially, the range for the source message is the entire interval $[0,1)$. When each symbol in the message is coded, the range is updated to a subinterval in its previous range.

To better illustrate the arithmetic coding process, we consider coding symbols from some alphabet and assume that both the arithmetic encoder and decoder know the probability distribution of the symbols to be coded as shown in Table 2.1. Suppose that the

Table 2.1: A probability distribution for the symbols $\{a, e, i, o, u, !\}$ associated with each symbol's interval in the initial source message range

Symbol	Probability	Interval
a	0.2	[0, 0.2)
e	0.3	[0.2, 0.5)
i	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
!	0.1	[0.9, 1)

source message to be encoded is "*eaii*!". We explain how arithmetic encoding works in what follows.

Before any symbols are encoded, the initial source message range is $[0,1)$. When encoding a symbol, the arithmetic encoder divides the range into several subintervals as Figure 2.11 shows, where each subinterval represents a fraction of the range that is proportional to the probability of the symbol (in Table 2.1). In each step, the new interval for the source message is updated to the interval of the symbol being encoded. For example, after seeing the first symbol e, the encoder updates the range to $[0.2, 0.5)$, which is the symbol e's interval in the initial range $[0,1)$. Then, a second symbol a further narrows the range to the first one-fifth of it. This produces $[0.2, 0.26)$, as the length of the previous range is 0.3 units, and the interval of a is the first one-fifth of the previous range. Proceeding in this way, we encode the message "*eaii*!" obtaining the final range $[0.23354, 0.2336)$ as shown in Figure 2.11. Since a value can be easily determined to be within a certain range, it is not necessary to encode both endpoints of the interval. Therefore, it is sufficient to encode a single number within the final interval, for instance, 0.23354.

Now we consider the case of decoding for the above example. The decoding process works as shown in Figure 2.12. To begin, we suppose the decoder is given the value, 0.23354 (in the final range determined by the encoder). Given 0.23354, the decoder can immediately deduce the first character of the message as e, as this value is within the interval of e in the initial range $[0,1)$. Then the range is updated to $[0.2, 0.5)$, and the second character decoded is a as its interval, [0.2, 0.26) entirely encloses 0.23354. Proceeding further, the decoder can identify the entire message "*eaii*!". In this coding example, the special symbol "!" is used to indicate the end of the message. When the decoder decodes this "!" symbol, the decoding process will stop.

In what follows, we introduce some additional terminology related to arithmetic coding.

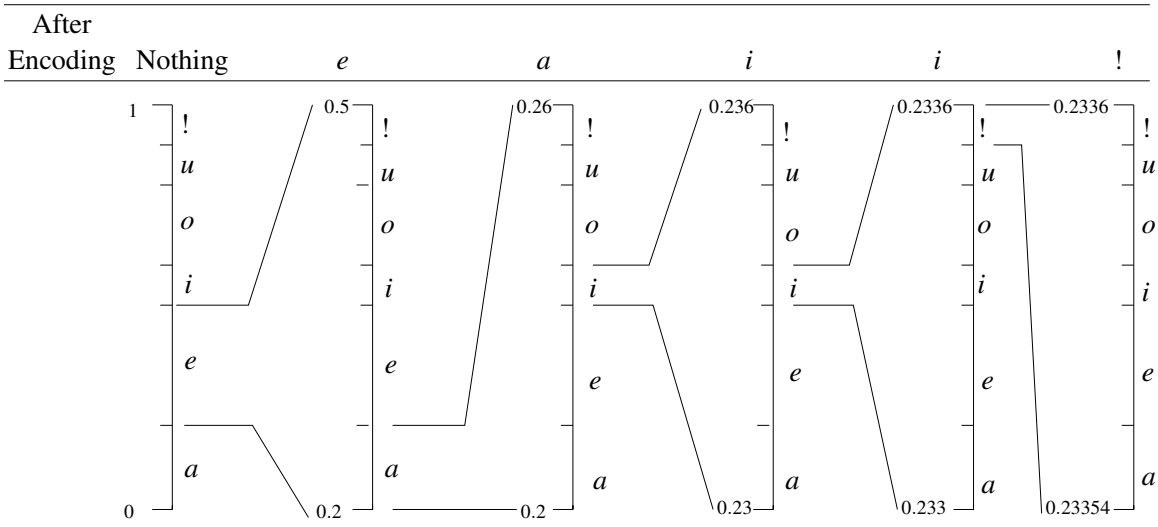


Figure 2.11: An example of arithmetic encoding showing the interval updated process.

First, an arithmetic coder is said to be binary if it only codes alphabets comprised of two symbols [35].If the need to code nonbinary symbol arises, a binarization scheme (such as the UI scheme in [1]) must be employed to convert each non-binary symbol to a sequence of binary symbols. When coding a symbol, a set of symbol probabilities must be specified. An arithmetic coder is said to be context-based if the set of probabilities selected is based on contextual information (available to both the encoder and decoder), rather than always using the same set of probabilities. Lastly, an arithmetic coder is said to be adaptive if it updates the probabilities of the symbols to be coded during the coding process.

2.6.2 Universal Coding

In data compression, a universal code for integers is a prefix code that maps the set of positive integers onto a set of binary codewords with the assumption that the probability distribution of the integers is monotonically decreasing for increasing integer values. Given any arbitrary source with nonzero entropy, the universal code can achieve an average codeword length that is within a constant factor of the theoretical lower bound (as determined by Shannon's source coding theorem [19]). As a matter of terminology, the process of using such codes for data compression is called universal coding. In the sections that follow, we introduce two kinds of universal coding schemes of relevance to our work, namely, Fibonacci coding [19] and Elias gamma coding [18].

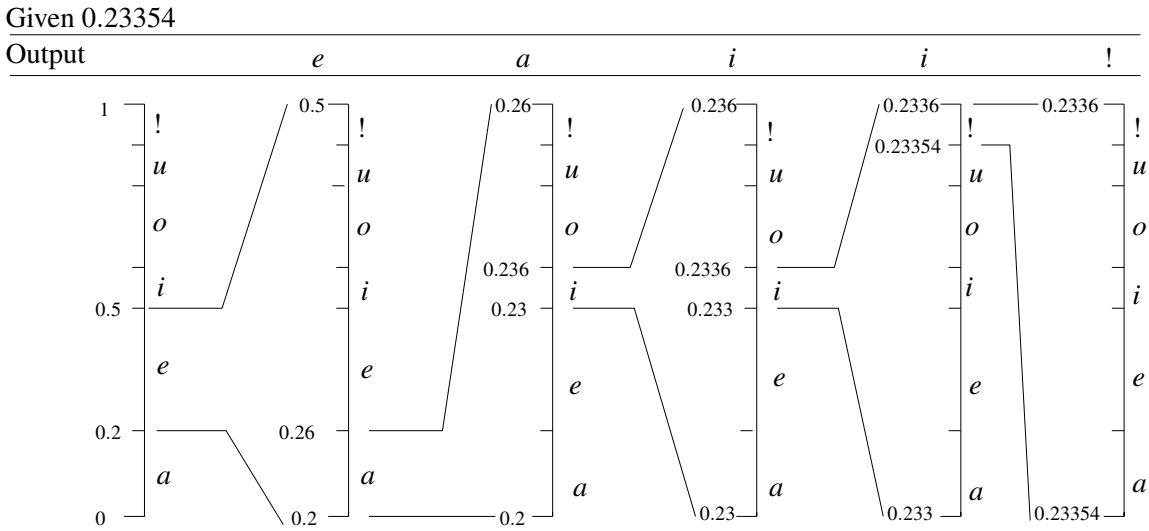


Figure 2.12: An example of arithmetic decoding showing the interval updated process.

2.6.2.1 Fibonacci Coding

Fibonacci codes, as described by Fraenkel and Klein [19], represent the set of positive integers based on Fibonacci numbers of order $m \geq 2$. In general, the j^{th} Fibonnaci number of order $m \geq 2$ is given by

$$F_j^{(m)} = \begin{cases} F_{j-1}^{(m)} + F_{j-2}^{(m)} + \cdots + F_{j-m}^{(m)} & \text{for } j > 1, \\ 1 & \text{for } j = 1, \\ 0 & \text{for } j \leq 0. \end{cases}$$

Particularly, the Fibonacci numbers of order $m = 2$ ($F_j \equiv F_j^{(2)}$) are the standard Fibonacci numbers $\{1,1,2,3,5,8,13,21,34,55,\ldots\}$. In what follows, we will describe the Fibonacci codes based on the standard Fibonacci numbers, as this is most relevant to the work in this thesis.

Any positive integer i can be represented as a binary string of the form

$$I = I_1 I_2 ... I_r, \text{ where } I_j \in \{0,1\}, \; i = \sum_{j=1}^{r} I_j F_{j+1}, \text{ and}$$

the sequence $\{I_j\}_{j=1}^{r}$ is chosen such that no two consecutive elements are both one and the rightmost element's index r satisfies $F_{r+1} \leq i$. We further append an extra one bit after the rightmost bit of I, and define the resulting binary string as the Fibonacci code of i, which

ends with "11" and contains no other instance of "11".

Next, we illustrate how Fibonacci coding works. Before coding begins, we suppose that both encoder and decoder know the probability for all possible symbols to be coded in the source message and an integer is uniquely associated with each symbol from most to least frequent, starting from one.

In order to encode a positive integer i, the following steps are performed by the encoder:

1) Find the largest Fibonacci number such that $F_{r+1} \leq i$. Set $I_r = 1$. Let $I_{j-1} = 0$ for $j \in \{2, ..., r\}$ in a binary string $I = I_1 I_2 ... I_r$. Set a remainder $r = (i - F_{r+1})$.
2) If r is zero, go to step 4; otherwise, find the largest Fibonacci number such that $F_{j+1} \leq r$. Update $I_j = 1$ in I. Let $r := r - F_{j+1}$.
3) If $r \neq 0$, go to step 2.
4) Output each bit of I from left to right, and then output a one bit.

As a numeric example, the positive integer 6 is encoded as follows. We start with finding F_5 as the largest Fibonacci number less than or equal to 6 and set the remainder $r = 6 - F_5$. Next, we obtain F_2 as the largest Fibonacci number less than or equal to r and update $r = r - F_2 = 0$. As r is zero now, we stop finding Fibonacci numbers. Following this, we output the binary string $I = 1001$ since $6 = 1 \times F_2 + 0 \times F_3 + 0 \times F_4 + 1 \times F_5$. Finally, we output a one bit which yields the Fibonacci codeword 10011.

To decode the Fibonacci code of an integer i, the following steps are performed by the decoder:

1) Read bits from the bit-stream until two consecutive ones are encountered.
2) Save those bits read in the previous step, except the last one, into a binary string $I = I_1 I_2 ... I_r$, where I_j with a smaller index j stores an earlier read bit (i.e., I_1 stores the first bit read in the previous step).
3) Obtain the decoded integer $i = \sum_{j=1}^{r} I_j F_{j+1}$.

As an example, the Fibonacci codeword 10011 is decoded as follows. We first read bits until two consecutive one bits are reached. Following this, the binary string $I = 1001$ is formed by the earlier read four bits. Finally, the decoded integer is obtained as $1 \times F_2 + 0 \times F_3 + 0 \times F_4 + 1 \times F_5 = 6$.

In Table 2.2, Fibonacci codes for a few small integers are given as additional examples to illustrate the mapping between positive integers and Fibonacci codewords. For instance, the number $32 = 3 + 8 + 21$ is represented by the Fibonacci code 00101011, since the codeword is obtained by appending an extra bit one after a binary string $I = I_1 I_2 ... I_r$, where $\sum_{j=1}^{r} I_j F_{j+1} = 0 \times F_2 + 0 \times F_3 + 1 \times F_4 + 0 \times F_5 + 1 \times F_6 + 0 \times F_7 + 1 \times F_8 = 32$.

Table 2.2: Examples of Fibonacci codes

Integer i	Sum of Fibonacci Numbers	Fibonacci Codeword
1	F_2	11
2	F_3	011
3	F_4	0011
4	F_2+F_4	1011
5	F_5	00011
6	F_2+F_5	10011
7	F_3+F_5	01011
8	F_6	000011
9	F_2+F_6	100011
16	F_4+F_7	0010011
17	$F_2+F_4+F_7$	1010011
32	$F_4+F_6+F_8$	00101011

2.6.2.2 Elias Gamma Coding

In 1975, Elias [18] proposed several universal coding schemes for representing a set of positive integers by a set of binary codewords. As one of the first universal codes proposed, the **Elias gamma code** was simple but not optimal. Generally speaking, the gamma code represents a positive integer i as $\lfloor \log_2 i \rfloor$ zero bits followed by a binary representation of i. In particular, the binary value of i is represented by as few bits as possible and therefore this representation always begins with a one bit.

In what follows, we illustrate how gamma coding works. Before any information is coded, we suppose that both encoder and decoder know the probability for all possible symbols to be coded in the source message and an integer is uniquely associated with each symbol from most to least frequent, starting from one.

In order to encode a positive integer i, the following steps are performed by the encoder:

1) Output a string of $\lfloor \log_2 i \rfloor$ zeros.
2) Output n as a $(\lfloor \log_2 i \rfloor + 1)$-bit integer.

As a numeric example, we encode a positive integer six as 00110. This is due to the fact that $\lfloor \log_2 6 \rfloor = 2$ and the binary representation of 6 with three bits is 110.

To decode the gamma code of an integer i, the following steps are performed by the decoder:

1) Read and count zeros from the bit-stream until the first bit one is encountered, then denote this count of zeros b.
2) Consider the bit of one received in the previous step as the first digit of the binary

Table 2.3: Examples of Gamma codes

Integer i	Binary Representation	Gamma Codeword
1 (2^0+0)	1	1
2 (2^1+0)	10	010
3 (2^1+1)	11	011
4 (2^1+2)	100	00100
5 (2^2+1)	101	00101
6 (2^2+2)	110	00110
7 (2^2+3)	111	00111
8 (2^3+0)	1000	0001000
9 (2^3+1)	1001	0001001
16 (2^4+0)	10000	000010000
17 (2^4+1)	10001	000010001
32 (2^5+0)	100000	00000100000

representation of i, read the following b digits as an integer r, and $i = 2^b + r$.

For example, the gamma codeword 00110 is decoded as follows. To begin, we reach the first one bit after reading two zero bits from the bit-stream. Therefore, the integer to be decoded has three bits. Then, the following two bits are received and converted into an integer $r = 1 \times 2^1 + 0 \times 2^0$. Finally, we obtain the decoded integer $i = 1 \times 2^2 + r$, which is 6.

In addition, examples of Elias gamma codewords for several small integers are given in Table 2.3 to better illustrate the mapping between positive integers and gamma codes. For instance, the integer $7 = 2^2 + 3$ is represented by the codeword 00111, since $\lfloor \log_2 7 \rfloor = 2$ and the binary representation of 7 using three bits is 111.

Chapter 3

Proposed Mesh-Coding Framework and Method

3.1 Overview

One highly effective approach for encoding mesh models of images is the IT method, proposed in [1]. The IT method, however, assumes the connectivity of the mesh to be Delaunay. In this chapter, a flexible mesh-coding framework, which adds connectivity coding to the IT method, and a new method derived from this framework for coding image mesh models with arbitrary connectivity are proposed. To begin, we present our new mesh-coding framework with several free parameters that must be chosen to yield a fully specified mesh-coding method. Next, we study how different choices of the free parameters affect coding efficiency, leading to the recommendation of a particular set of choices. Following this, we propose a specific mesh-coding method which employs the recommended choices in our framework. Lastly, the performance of the proposed method is evaluated by comparison with a simple coding scheme as well as more traditional coding approaches.

3.2 Proposed Mesh-Coding Framework

With the necessary background in place, we can now introduce our general framework for mesh coding. As mentioned earlier, our approach is based on the IT scheme [1]. The IT method, although highly effective for coding mesh geometry, has no means for coding mesh connectivity. Our proposed coding framework extends the IT coding scheme by adding to it a mechanism for coding connectivity. By providing the ability to code connectivity, our

framework can be used to encode mesh models with arbitrary connectivity, unlike the IT scheme.

Our approach to connectivity coding is based on the idea of expressing the connectivity of a triangulation relative to the connectivity of a uniquely determined reference triangulation via a sequence of edge flips. More specifically, for a triangulation T of a set P of points, our approach represents the connectivity of T as a sequence S of edge flips that transforms the PDDT of P to T. Since the PDDT of P is unique, P and S completely characterize the connectivity of T. Given P and S, we can always recover T by first computing the PDDT of P and then applying the sequence S of edge flips to this PDDT. If T is close to having PDDT connectivity, the sequence S will be very short and the connectivity coding cost (in bits) will be very small. As T deviates further from having PDDT connectivity, the sequence S will grow in length and the connectivity coding cost will increase. In the sections that follow, we describe the encoding and decoding processes in more detail.

3.2.1 Encoding

First, we consider the encoding process. As input, this process takes a mesh model, consisting of a set P of sample points, a triangulation T (of P), and the set Z of function values (at the sample points). Given such a model, the encoding process outputs a coded bit stream, using an algorithm comprised of the following steps:

1. *Geometry coding.* Encode the mesh geometry (i.e., P and Z) using the IT scheme as described in [1].

2. *Sequence generation.* Generate a sequence S of edge flips that transforms the PDDT of P to the triangulation T.

3. *Sequence optimization.* Optionally, optimize the edge-flip sequence S to facilitate more efficient coding.

4. *Sequence encoding.* Initialize the triangulation τ to the PDDT of P. Encode the edge-flip sequence S, updating the triangulation τ in the process.

In the sections that follow, we explain each of steps 2, 3 and 4 (from above) in more detail.

3.2.1.1 Sequence Generation (Step 2 of Encoding)

In step 2 of our encoding framework, an edge-flip sequence is generated. We now explain in more detail how this is accomplished. To begin, we assign a unique integer label to

each edge in the triangulation T by numbering edges starting from zero using the lexicographic order for line segments (as defined in Section 2.2). Next, we apply the LOP to the triangulation with the edge-optimality criterion chosen as the PDDT criterion, which will yield the PDDT of P. As the LOP is performed, each edge flip is recorded in the sequence $S' = \{s'_i\}_{i=0}^{|S'|-1}$, where s'_i is the label of the ith edge flipped. (Note that flipping an edge does not change its label.) After the LOP terminates (yielding the PDDT of P), each edge in the triangulation is assigned a new unique label using a similar process as above (i.e., by numbering edges starting from zero using the lexicographic order for line segments). Let ρ denote the function that maps the original edge labels to the new ones. The edge-flip sequence $S = \{s_i\}_{i=0}^{|S|-1}$ that maps the PDDT of P to T is then given by $s'_i = \rho(s_{|S|-1-i})$. In other words, S is obtained by reversing the sequence S' and relabelling the elements of the sequence so that they are labelled with respect to the PDDT of P. The particular sequence S obtained from the above process will depend on the specific priority scheme employed by the suspect-edge queue. In our work, the following three priority schemes were considered:

1. first-in first-out (FIFO),
2. last-in first-out (LIFO), and
3. lexicographic (i.e., edges are removed from the queue in lexicographic order).

As for which choice of priority scheme might be best, we shall consider this later in Section 3.4.

3.2.1.2 Sequence Encoding (Step 4 of Encoding)

The sequence encoding process in step 4 of our encoding framework employs a scheme that numbers a subset of edges in a triangulation relative to a particular edge. By utilizing this relative indexing approach, we can exploit the locality in the edge-flip sequence (i.e., the tendency of neighbouring elements in the sequence to be associated with edges that are close to one another in the triangulation). Therefore, before discussing the sequence encoding process further, we must first present this relative indexing scheme for edges. To begin, we first introduce some necessary terminology and notation. For an edge e in a triangulation, dirEdge(e) denotes the directed edge oriented from the smaller vertex to the larger vertex of e in xy-lexicographic order. For a directed (triangulation) edge h:

1) opp(h) denotes the directed edge with the opposite orientation (and same vertices) as h;

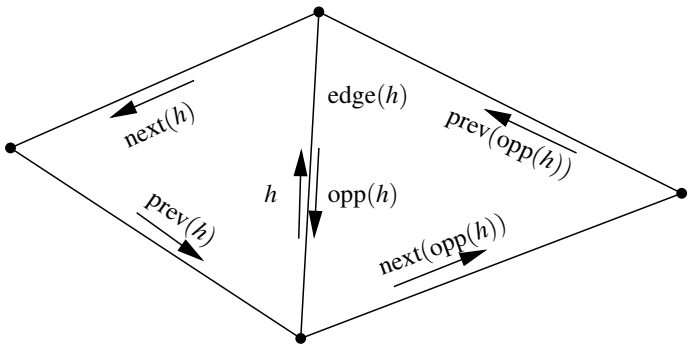


Figure 3.1: Illustration of various definitions related to directed edges. An edge e in a triangulation with two incident faces and the associated directed edges h and $\text{opp}(h)$.

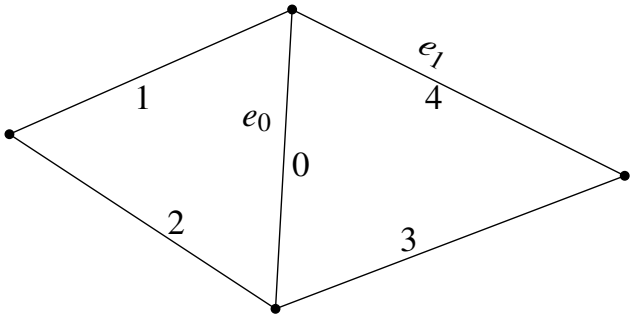


Figure 3.2: An example of numbering edges using the relative indexing scheme

2) $\text{next}(h)$ and $\text{prev}(h)$ denote the directed edges with the same left face as h that, respectively, follow and precede h in counterclockwise order around their common left face; and
3) $\text{edge}(h)$ denotes the (undirected) edge associated with h.

The preceding definitions are illustrated in Figure 3.1. With the above notation in place, we can now specify our relative indexing scheme for edges. Given a triangulation τ and a subset Θ of its edges and two distinct edges $e_1, e_0 \in \Theta$, the index of the edge e_1 relative to the edge e_0, denoted $\text{relIndex}(e_1, e_0, \tau, \Theta)$, is determined as specified in Algorithm 1. (Note that $\text{relIndex}(e_1, e_0, \tau, \Theta)$ is not necessarily equal to $\text{relIndex}(e_0, e_1, \tau, \Theta)$.)

To further demonstrate how our relative indexing approach works, we present a simple example in Figure 3.2 that aims to find the index of edge e_1 relative to edge e_0 in the set $\Theta = \text{edges}(\tau)$ (i.e., the set of all edges in the triangulation τ). From this figure, we can see that each edge in Θ has been assigned a unique index, starting from zero. In particular, the edge e_0 and edges in the left face of $\text{dirEdge}(e_0)$ are first numbered in a counterclockwise order as 0, 1 and 2. Then the edges in the left face of $\text{opp}(\text{dirEdge}(e_0))$ are numbered in a

Algorithm 1 Calculating relIndex(e_1, e_0, τ, Θ) (i.e., the index of the edge e_1 relative to the edge e_0 in the set Θ of edges in the triangulation τ.)

```
1: {q is FIFO queue of directed edges}
2: {h is directed edge and c is integer counter}
3: h := dirEdge(e_0)
4: mark all edges in τ as not visited
5: clear q
6: insert opp(h) and then h in q
7: c := 0
8: while q not empty do
9:    remove element from front of q and set h to removed element
10:   if edge(h) ∈ Θ then
11:      if edge(h) not visited then
12:         mark edge(h) as visited
13:         if edge(h) = e_1 then
14:            return c as index of edge e_1 relative to edge e_0; and stop
15:         endif
16:         c := c + 1
17:      endif
18:   endif
19:   if opp(e) has left face then
20:      if edge(next(opp(h))) not visited then
21:         insert next(opp(h)) in q
22:      endif
23:      if edge(prev(opp(h))) not visited then
24:         insert prev(opp(h)) in q
25:      endif
26:   endif
27: endwhile
28: abort with error indicating e_1 ∉ Θ
```

counterclockwise order as 3 and 4. Since e_1 is the edge with an index of 4, we deem the index of e_1 relative to e_0 in Θ as 4 (i.e., $\text{relIndex}(e_1, e_0, \tau, \Theta) = 4$).

Having explained the relative indexing scheme for edges, we can now describe the edge-flip sequence encoding process in more detail. In our work, we proposed five variants of the encoding process, each of which is based on the idea of coding each edge in the edge-flip sequence S (excluding the first) relative to the preceding edge in the sequence. For the purpose of presentation, we have partitioned these variants into two groups: the non-finalization (NF) group and finalization group. In what follows, we present the variants of our sequence encoding process, starting with the non-finalization group.

Non-finalization group. The first of the two groups of encoding approaches is the non-finalization group. This group contains three variants: 1) arithmetic-coding-non-finalization (ACNF), 2) Fibonacci, and 3) gamma. These variants share a common algorithmic framework, only differing in the particular entropy coding scheme used. For entropy coding, the ACNF, Fibonacci, and gamma variants employ context-based adaptive binary arithmetic coding [44], Fibonacci coding [19], and gamma coding [18], respectively.

Given a triangulation T with the set P of vertices, the edge-flip sequence $S = \{s_i\}_{i=0}^{|S|-1}$ that transforms the PDDT of P to T, and the variant v of the sequence encoding scheme to be used, the encoding process proceeds as follows:

1. Initialize the triangulation τ to the PDDT of P.
2. Encode $|S|$ as a 30-bit integer.
3. If $|S|$ is zero, stop.
4. Flip the edge s_0 in τ.
5. (a) If v is ACNF, encode s_0 as an m-bit integer, where $m = \lceil \log_2(|\text{edges}(\tau)|) \rceil$ (i.e., m is the number of bits needed for an integer representing edge labels).
   (b) If v is gamma, encode $(1 + s_0)$ using gamma coding.
   (c) If v is Fibonacci, encode $(1 + s_0)$ using Fibonacci coding.
6. If $|S| < 2$, stop.
7. For $i \in \{1, 2, \ldots, |S| - 1\}$:
   (a) Let $r_i = \text{relIndex}(s_i, s_{i-1}, \tau, \text{flippableEdges}(\tau)) - 1$, where $\text{flippableEdges}(\tau)$ denotes the set of all flippable edges in τ.
   (b) Flip the edge s_i in τ.

8. If v is ACNF, encode $n = \lceil \log_2 (1 + \max\{r_1, r_2, \ldots, r_{|S|-1}\}) \rceil$ as a 30-bit integer (i.e., n is the number of bits needed for an integer representing relative indexes).

9. If v is ACNF, initialize the arithmetic coding engine and start a new arithmetic codeword.

10. For $i \in \{1, 2, \ldots, |S| - 1\}$:
    (a) If v is ACNF, encode r_i using the UI$(n, 4)$ binarization scheme as described in [1] (where n and r_i are as calculated above).
    (b) If v is gamma, encode $(1 + r_i)$ using gamma coding.
    (c) If v is Fibonacci, encode $(1 + r_i)$ using Fibonacci coding.

11. If v is ACNF, terminate the arithmetic codeword.

Finalization group. Before presenting the variants in the finalization group, we first introduce the notion of a finalized edge. An edge is said to be **finalized** if it will not be flipped again at any later point in the coding of the edge-flip sequence $S = \{s_i\}_{i=0}^{|S|-1}$ (which transforms the PDDT of the set P of points to the triangulation T of P). To be more specific, we take an edge corresponding to the edge-flip sequence element s_i as an example. Before we code any elements in S, the edge corresponding to s_i is not finalized since this edge will be flipped later during the coding process. Once the ith element s_i is processed in the coding of S, however, the corresponding edge is deemed finalized if $s_i \neq s_j$ for $j \in \{i+1, i+2, ..., |S| - 1\}$.

With the preceding notion in mind, we can now introduce the second of the two groups of encoding approaches, that is, the finalization group. This group contains two variants: 1) arithmetic-coding-finalization-1 (ACF1), and 2) arithmetic-coding-finalization-2 (ACF2). In general, the ACF1 and ACF2 variants are based on the idea of coding whether each edge flip results in the flipped edge becoming finalized. In order to let the decoder know if the edge corresponding to s_i is finalized, once s_i is coded, both variants code a bit indicating if the edge corresponding to s_i is finalized (i.e., is not flipped again). Besides that, the ACF2 variant codes extra bits to let the decoder know which edges are finalized before any elements of the edge-flip sequence are coded (i.e., which edges are never flipped in the edge-flip sequence).

Given a triangulation T with the set P of vertices, the edge-flip sequence $S = \{s_i\}_{i=0}^{|S|-1}$ that transforms the PDDT of P to T, and the variant v of the sequence encoding scheme to be used, the encoding process proceeds as follows:

1. Initialize the triangulation τ to the PDDT of P.
2. Mark all edges in τ as not finalized. Then mark all border edges as finalized.
3. Encode $|S|$ (i.e., number of edge flips) as a 30-bit integer.
4. If $|S|$ is zero, stop.
5. Initialize the arithmetic coding engine and start a new arithmetic codeword.
6. If v is ACF2:
   (a) Mark all nonborder edges that are not in S as finalized.
   (b) For each nonborder edge in lexicographic order, encode a bit indicating if the edge is finalized, conditioned on the flippablility of the edge (i.e., one context for flippable edges and one context for unflippable edges).
7. Encode s_0 as an m-bit integer, where $m = \lceil \log_2(|\text{edges}(\tau)|) \rceil$, using arithmetic coding with the probability of a 1 symbol being fixed at 0.5.
8. Flip the edge s_0 in τ. If the edge corresponding to s_0 will not be flipped again (i.e., $s_0 \notin \{s_1, s_2, ..., s_{|S|-1}\}$), mark the corresponding edge as finalized.
9. Encode a bit indicating if the edge corresponding to s_0 is finalized, conditioned on f, where

$$f = \begin{cases} 0 & \text{if } \alpha = 0 \\ 1 & \text{if } \alpha \in \{1,2\} \\ 2 & \text{if } \alpha \in \{3,4\} \end{cases}$$

   and α is the number of edges already marked as finalized in the convex quadrilateral formed by two faces incident on the edge corresponding to s_0.
10. For $i \in \{1, 2, \ldots, |S|-1\}$:
    (a) Encode $r_i = \text{relIndex}(s_i, s_{i-1}, \tau, E) - 1$ using the UI$(n,4)$ binarization scheme described in [1], where E denotes the set of all edges in τ that are flippable and not marked as finalized, and $n = \lceil \log_2(|E|) \rceil$.
    (b) Flip the edge s_i in τ. If the edge corresponding to s_i will not be flipped again (i.e., $s_i \notin \{s_{i+1}, s_{i+2}, ..., s_{|S|-1}\}$), mark the corresponding edge as finalized.

(c) Encode a bit indicating if the edge corresponding to s_i is finalized, conditioned on f, where

$$f = \begin{cases} 0 & \text{if } \alpha = 0 \\ 1 & \text{if } \alpha \in \{1,2\} \\ 2 & \text{if } \alpha \in \{3,4\} \end{cases}$$

and α is the number of edges already marked as finalized in the convex quadrilateral formed by two faces incident on the edge corresponding to s_i.

11. Terminate the arithmetic codeword.

Additional comments. For all variants of the sequence encoding process, each edge flip in the edge-flip sequence S (excluding the first) is coded relative to the preceding edge flip in the sequence. Such an approach is effective since the edge-flip sequence tends to exhibit locality (i.e., neighbouring elements in the sequence tend to be associated with edges that are close to one another in the triangulation). As for which choice of sequence encoding scheme might be best, we shall consider this later in Section 3.4.

3.2.1.3 Sequence Optimization (Step 3 of Encoding)

As mentioned earlier, our coding scheme relies on the fact that the edge-flip sequence tends to exhibit some degree of locality. The purpose of the (optional) sequence-optimization step (i.e., step 3) in our encoding process is to attempt to improve the locality properties of the edge-flip sequence through optimization (prior to encoding). In what follows, we describe this optimization process in more detail.

Before proceeding further, we must first introduce some notation and terminology related to the optimization process. Let $\text{tri}_{T,S}(i)$ denote the triangulation obtained by applying the first i edge flips in the sequence S to the triangulation T and let $\text{tri}_{T,S}$ denote the triangulation obtained by applying all of the edge flips in the sequence S to the triangulation T. To illustrate the preceding notation, we present an example in Figure 3.3. In this figure, the initial triangulation T is $\text{tri}_{T,S}(0)$ and the edge-flip sequence is $S = \{s_0, s_1, s_2\}$. The triangulation $\text{tri}_{T,S}(1)$ is obtained by applying the first edge flip s_0 to $\text{tri}_{T,S}(0)$. Then, we transform from $\text{tri}_{T,S}(1)$ to $\text{tri}_{T,S}(2)$ by applying the edge flip s_1 to $\text{tri}_{T,S}(1)$. Finally, we obtain $\text{tri}_{T,S}(3)$ by flipping the edge s_2 in $\text{tri}_{T,S}(2)$.

Two edge-flip sequences S and S' are said to be **equivalent** if $\text{tri}_{T,S} = \text{tri}_{T,S'}$ (i.e., the application of each edge-flip sequence to the triangulation T produces the same final trian-

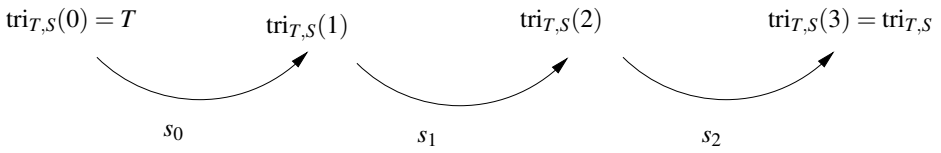


Figure 3.3: An example that transforms triangulations named by the notation introduced in Section 3.2.1.3 by applying an edge flip in the edge-flip sequence S to a triangulation.

gulation). Let $\mathrm{swap}(S,i,j)$ denote the new sequence formed by swapping the ith and jth elements in the sequence S. Let $\mathrm{erase}(S,i,j)$ denote the new sequence formed by removing elements $s_i, s_{i+1}, \ldots, s_j$ from the sequence S (where elements in S with index greater than j are shifted downwards by $j-i+1$ positions to form the new sequence). Two adjacent elements of an edge-flip sequence S with indices i and $i+1$ are said to be **swappable** if they correspond to edges that are not incident on the same face of the triangulation $\mathrm{tri}_{T,S}(i)$.

For a given edge-flip sequence, it is possible to find many (distinct) sequences that are equivalent (in the sense of "equivalent" as defined above). Some of these equivalent sequences, however, have better locality properties than others, and therefore lend themselves to more efficient coding. The optimization process attempts to produce an edge-flip sequence with better locality by applying a series of transformations to the sequence that yields an equivalent sequence. Two types of transformations are of interest. First, if the ith and $(i+1)$th elements in the sequence S are swappable (as defined above), swapping these elements will yield an equivalent sequence (i.e., $\mathrm{tri}_{T,S} = \mathrm{tri}_{T,\mathrm{swap}(S,i,i+1)}$). Second, if the ith and $(i+1)$th elements in S are equal, the deletion of these two elements will yield an equivalent sequence (i.e., $\mathrm{tri}_{T,S} = \mathrm{tri}_{T,\mathrm{erase}(S,i,i+1)}$).

With the above in mind, the optimization process works as follows. The optimization algorithm makes repeated passes over the elements of the sequence S, until a full pass completes without any change being made to S. Each pass performs the following for $i \in \{0,1,\ldots,|S|-2\}$:

1. If $s_i = s_{i+1}$, $S := \mathrm{erase}(S,i,i+1)$ (i.e., delete ith and $(i+1)$th elements from S).

2. Otherwise, if s_i and s_{i+1} are swappable and the binary decision function $d_S(i) \neq 0$, $S := \mathrm{swap}(S,i,i+1)$ (i.e., swap the ith and $(i+1)$th elements in S) and $i := i+1$. The binary decision function $d_S(i)$, which is used to determine if the ith and $(i+1)$th elements in S should be swapped, is a free parameter of our method and will be described in more detail shortly.

In our work, we considered three choices for the decision function d_S in step 2 above. To

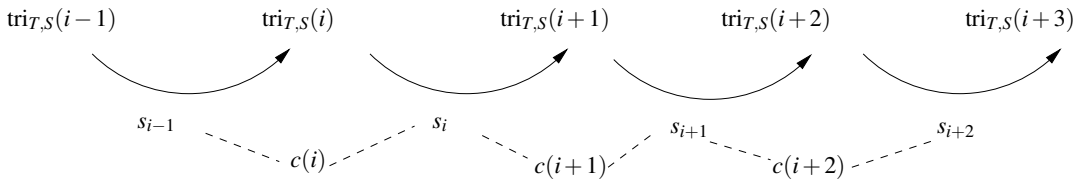


Figure 3.4: An example that shows the relationship between triangulations, edge flips and approximate coding cost.

assist in specifying these choices, we introduce some additional notation in what follows. For $x, y \in \mathbb{R}$, we define the binary-valued functions

$$\text{lt}(x,y) = \begin{cases} 1 & x < y \\ 0 & \text{otherwise} \end{cases}$$

and

$$\text{lte}(x,y) = \begin{cases} 1 & x \leq y \\ 0 & \text{otherwise} \end{cases}$$

(i.e., lt and lte are boolean-valued functions for testing the less-than and less-than-or-equal conditions). Let $c_S(i)$ denote the approximate cost of coding the ith edge flip in the sequence S, where

$$c_S(i) = \begin{cases} \text{relIndex}(s_i, s_{i-1}, \text{tri}_{T,S}(i), \text{flippableEdges}(\text{tri}_{T,S}(i))) & i \in \{1,2,\ldots,|S|-1\} \\ 0 & i \in \{0,|S|\} \end{cases}$$

and $\text{flippableEdges}(T)$ is the set of all flippable edges in the triangulation T. For convenience, let $c(i)$ and $c'(i)$ denote $c_S(i)$ and $c_{\text{swap}(S,i,i+1)}(i)$, respectively (i.e., $c(i)$ and $c'(i)$ represent the cost without and with the ith and $(i+1)$th edges swapped, respectively). The relationship between triangulations, edge flips and approximate coding cost is illustrated in Figure 3.4. In this figure, triangulations are transformed by flipping edges s_{i-1}, s_i, s_{i+1} and s_{i+2}. For $k \in \{i, i+1, i+2\}$, the approximate coding cost $c(k)$ is the index of s_k relative to s_{k-1} in $\text{flippableEdges}(\text{tri}_{T,S}(k))$. Thus, if s_i and s_{i+1} were swapped, $c(i), c(i+1)$, and $c(i+2)$ would be affected.

With the above notation in place, we can now introduce the three choices for the deci-

sion function d_S considered in our work. These three choices for d_S are as follows:

$$\text{rule 1:} \qquad d_S(i) = \text{lt}\left(\sum_{k=i}^{i+2} c'(k), \sum_{k=i}^{i+2} c(k)\right)$$

(i.e., elements are swapped if this would strictly reduce the overall sum of the relative indexes to be coded);

$$\text{rule 2:} \quad d_S(i) = \left[\prod_{k\in I} \text{lte}\left(c'(k), c(k)\right)\right]\left[\max_{k\in I} \text{lt}(c'(k), c(k))\right],$$

where $I = \{\max\{1,i\},\ldots,\min\{|S|-2, i+2\}\}$ (i.e., elements are swapped if this would not increase the cost of any of the three relative indexes to be coded that are affected by the swap and at least one cost would be strictly reduced); and

$$\text{rule 3:} \qquad d_S(i) = \begin{cases} \text{lt}\left(c'(i), c(i)\right) & i \geq 1 \\ 0 & i = 0 \end{cases}$$

(i.e., elements are swapped if this would strictly reduce the ith relative index to be coded). As for which choice of d_S might be best, we will explore this later in Section 3.4.

3.2.2 Decoding

Having introduced the encoding process, we now consider the decoding process. Given a coded bit stream as input, this process outputs the corresponding mesh model, characterized by a set P of sample points and their corresponding set Z of function values, and a triangulation T of P. The decoding process consists of the following steps:

1. *Geometry decoding.* Decode the mesh geometry using the IT scheme (as described in [1]), yielding P (and Z).

2. *Sequence decoding.* Initialize the triangulation τ to the PDDT of P. Label the edges in the triangulation in an identical manner as in the encoder (i.e., lexicographic order starting from zero). Decode the edge-flip sequence, updating τ in the process. After each edge flip is decoded, the edge flip is applied to the (current) triangulation τ. The final value of τ corresponds to the decoded triangulation T.

In step 2 of the decoding process above, the steps involved in the decoding of an edge-flip sequence simply mirror the corresponding steps in encoding (described earlier in Sec-

tion 3.2.1.2).

3.3 Test Data

Before proceeding further, a brief digression is necessary in order to introduce the test data used in our work. Herein, we employed a set of 50 mesh models of images that were produced by the state-of-the-art mesh-generation scheme proposed in [28]. Since the efficiency of our proposed coding approach depends on the extent to which mesh connectivity deviates from being Delaunay, we have grouped our meshes into two categories, A and B, based on the extent of this deviation. In order to quantify the extent to which the connectivity of a mesh deviates from being Delaunay, we used the length of the edge-flip sequence required to transform the mesh connectivity to PDDT connectivity, measured as a percentage of the total number of edges in the mesh. The 25 meshes in category A have connectivity relatively close to being Delaunay (i.e., the relative length of the edge-flip sequence is less than or equal to 15%), while the 25 meshes in category B have connectivity that deviates relatively more from being Delaunay (i.e., the relative length of the edge-flip sequence is greater than 15%). Herein, we present statistical results taken across all of our meshes as well as results for individual meshes. For consistency, when presenting results for individual meshes, we focus on the ten representative meshes listed in Table 3.1, where the meshes A1, A2, A3, A4 and A5 are from category A and the meshes B1, B2, B3, B4 and B5 are from category B.

3.4 Proposed Mesh-Coding Method and Its Development

Earlier in Section 3.2, we introduced our proposed framework for mesh coding. This framework has three free parameters, namely, the choice of priority scheme (used by the LOP), the choice of sequence coding method (used by sequence coding), and the choice of optimization strategy (used by sequence optimization). In the sections that follow, we study how different choices for these parameters affect coding efficiency. Since the difference amongst these choices is in the connectivity coding alone, we will focus our attention on evaluating the coded sizes for connectivity information. After an analysis of our experimental results, we recommend a particular set of choices to be used for the parameters in our framework, leading to the specific coding method proposed herein.

Table 3.1: Several of the mesh models used in our work

Name	Category	Vertex Count	Edge Count	Sequence Length†	Relative Sequence Length (%)†
A1	A	15728	47080	1070	2.3
A2	A	6881	20632	1421	6.9
A3	A	5242	15374	2298	14.9
A4	A	7864	23536	410	1.7
A5	A	7864	23585	1163	4.9
B1	B	7864	23536	7326	31.1
B2	B	7864	23514	6720	28.6
B3	B	2621	7797	3235	41.5
B4	B	2621	7797	2517	32.3
B5	B	2621	7801	2968	38.0

†Sequence generated with lexicographic priority scheme.

3.4.1 Choice of Sequence Coding Method

To begin, we consider how different choices of sequence coding method (in step 4 of the encoding process) affect coding efficiency. For each of the 50 meshes in our test set, we coded the mesh using each of the priority schemes described earlier in Section 3.2.1.1 (i.e., lexicographic, FIFO, and LIFO) and each of the sequence coding schemes described earlier in Section 3.2.1.2 (i.e., ACNF, Fibonacci, gamma, ACF1, and ACF2). In each case, the resulting bit rate was measured. The results obtained for each of lexicographic, FIFO and LIFO priority schemes are shown in Tables 3.2, 3.3 and 3.4, respectively. In each case in the tables, the best result is highlighted in bold font.

Lexicographic priority scheme. First, we consider the connectivity coding performance results obtained with the lexicographic priority scheme as given by Table 3.2. From the results for the individual meshes in Table 3.2(a), it is clear that the ACNF scheme consistently leads to the lowest bit rate in 9/10 of the test cases by a margin of up to 2.19 bits/vertex. As to the overall statistical results in Table 3.2(b), the ACNF scheme consistently beats the other four methods. In fact, a more detailed examination of the results shows that our ACNF scheme performs best in 43/50 (86%) of the test cases by a margin of up to 2.19 bits/vertex and second best in 7/50 (14%) of the test cases.

FIFO priority scheme. Next, we consider the connectivity coding performance results obtained with the FIFO priority scheme as given by Table 3.3. Examining the results for the individual meshes in Table 3.3(a), we can clearly see that the ACNF scheme beats the

Table 3.2: Comparison of the connectivity coding performance obtained with the lexicographic priority scheme and various sequence coding methods. (a) Individual results. (b) Overall results.

(a)

	Coded Size (bits/vertex)				
Dataset	Gamma	ACNF	Fibonacci	ACF1	ACF2
A1	1.24	**0.86**	0.98	0.87	0.90
A2	2.58	**1.98**	2.11	2.03	2.13
A3	4.45	2.74	3.78	**2.69**	3.35
A4	0.84	**0.61**	0.67	0.62	0.66
A5	2.73	**1.83**	2.14	1.86	1.93
B1	9.76	**7.70**	8.14	7.94	8.58
B2	9.17	**7.24**	7.62	7.52	7.96
B3	10.98	**9.03**	9.36	9.34	10.12
B4	9.56	**7.64**	8.01	7.98	8.39
B5	0.93	**8.74**	9.21	8.98	9.64

(b)

	Mean Coded Size (bits/vertex)				
Category	Gamma	ACNF	Fibonacci	ACF1	ACF2
A	1.65	**1.16**	1.33	1.17	1.25
B	9.78	**8.02**	8.32	8.30	8.94
Overall	5.71	**4.59**	4.83	4.73	5.09

Table 3.3: Comparison of the connectivity coding performance obtained with the FIFO priority scheme and various sequence coding schemes. (a) Individual results. (b) Overall results.

(a)

Dataset	Coded Size (bits/vertex)				
	Gamma	ACNF	Fibonacci	ACF1	ACF2
A1	1.67	**1.02**	1.29	1.03	1.03
A2	3.89	**2.53**	3.05	2.58	2.60
A3	5.58	3.95	4.54	**3.85**	4.10
A4	1.19	**0.73**	0.92	0.74	0.76
A5	2.89	**1.87**	2.26	1.90	1.95
B1	11.39	**8.42**	9.32	8.58	9.04
B2	10.79	**7.87**	8.78	8.09	8.35
B3	12.93	**9.93**	10.76	10.12	10.66
B4	10.65	**8.10**	8.80	8.39	8.62
B5	12.43	**9.45**	10.28	9.64	10.08

(b)

Category	Mean Coded Size (bits/vertex)				
	Gamma	ACNF	Fibonacci	ACF1	ACF2
A	2.17	**1.41**	1.70	1.41	1.45
B	11.40	**8.80**	9.48	9.02	9.46
Overall	6.78	**5.10**	5.59	5.22	5.45

other four methods in 9/10 of the test cases by a margin of up to 3 bits/vertex. With respect to the overall statistical results in Table 3.3(b), the ACNF scheme consistently leads to the lowest bit rate. In fact, a more detailed examination of the results shows that the ACNF scheme outperforms the other four methods in 45/50 (90%) of the test cases by a margin of up to 3.64 bits/vertex, and never losing by a margin of more than 0.1 bits/vertex in the remaining test cases.

LIFO priority scheme. Lastly, we consider the connectivity coding performance results obtained with the LIFO priority scheme as given by Table 3.4. From the results for the individual meshes in Table 3.4(a), it is clear that the ACNF scheme consistently leads to the lowest bit rate in 9/10 of the test cases by a margin of up to 2.97 bits/vertex. As to the overall statistical results in Table 3.4(b), the ACNF scheme consistently beats the other four methods. In fact, a more detailed examination of the results shows that our ACNF scheme performs best in 45/50 (90%) of the test cases by a margin of up to 3.40 bits/vertex and second best in 5/50 (10%) of the test cases.

Table 3.4: Comparison of the connectivity coding performance obtained with the LIFO priority scheme and various sequence coding schemes. (a) Individual results. (b) Overall results.

(a)

Dataset	Coded Size (bits/vertex)				
	Gamma	ACNF	Fibonacci	ACF1	ACF2
A1	1.67	**1.02**	1.29	1.03	1.04
A2	3.89	**2.54**	3.05	2.60	2.62
A3	5.57	4.01	4.54	**3.93**	4.25
A4	1.19	**0.74**	0.92	0.74	0.76
A5	2.89	**1.89**	2.27	1.92	1.96
B1	11.33	**8.36**	9.25	8.53	8.99
B2	10.38	**7.63**	8.49	7.93	8.15
B3	12.74	**9.81**	10.63	10.05	10.55
B4	10.35	**7.89**	8.56	8.14	8.42
B5	12.21	**9.29**	10.13	9.45	9.89

(b)

Category	Mean Coded Size (bits/vertex)				
	Gamma	ACNF	Fibonacci	ACF1	ACF2
A	2.16	**1.41**	1.70	1.42	1.46
B	11.08	**8.60**	9.24	8.82	9.24
Overall	6.62	**5.00**	5.47	5.12	5.35

Overall results. Finally, we consider all of the priority schemes together. Based on the above results, we deem the best choice for the sequence coding scheme in the proposed framework to be the ACNF scheme. In passing, we note that, although the ACF1 and ACF2 schemes often produced smaller relative indexes (compared to the ACNF scheme), the extra cost required for coding bits indicating if edges are finalized rendered the coding efficiency of the ACF1 and ACF2 schemes worse than that of the ACNF scheme.

3.4.2 Choice of Priority Scheme

Next, we consider how different choices of priority scheme used by the LOP (in step 2 of the encoding process) affect coding efficiency. For each of the 50 meshes in our test set, we coded the mesh using the ACNF sequence coding scheme (previously found to perform best) in conjunction with each of the three priority schemes (namely, LIFO, FIFO, and lexicographic) and measured the resulting bit rate. The results obtained are shown in Table 3.5. In each case, the best result is highlighted in bold font.

Table 3.5: Comparison of the connectivity coding performance obtained with the various priority schemes and the best sequence encoding method: ACNF scheme described in the non-finalization category of Section 3.2.1.2. (a) Individual results. (b) Overall results.

(a)

Dataset	Coded Size (bits/vertex)		
	LIFO	FIFO	Lex.†
A1	1.02	1.02	**0.86**
A2	2.54	2.53	**1.98**
A3	4.01	3.95	**2.74**
A4	0.74	0.73	**0.61**
A5	1.89	1.87	**1.83**
B1	8.36	8.42	**7.70**
B2	7.63	7.87	**7.24**
B3	9.81	9.93	**9.03**
B4	7.89	8.10	**7.64**
B5	9.29	9.45	**8.74**

†lexicographic

(b)

Category	Mean Coded Size (bits/vertex)		
	LIFO	FIFO	Lex.†
A	1.41	1.41	**1.16**
B	8.60	8.80	**8.02**
Overall	5.00	5.10	**4.59**

†lexicographic

Examining the results for individual meshes in Table 3.5(a) and the overall statistical results in Table 3.5(b), we can clearly see that the lexicographic priority scheme is most effective, consistently leading to the lowest bit rate in the individual cases as well as overall. In fact, a more detailed examination of the results shows that the lexicographic scheme performs best in all 50/50 of the test cases. A careful analysis shows that the superior performance of the lexicographic priority scheme is due to its ability to produce edge-flip sequences with better locality properties.

Based on the above results, we deem the best choice for priority scheme (used by the LOP) in the proposed framework to be lexicographic. Therefore, we recommend this choice to be used in our framework.

3.4.3 Choice of Optimization Strategy

Finally, we consider how different choices of optimization strategy in the edge-flip sequence optimization step (i.e., step 3) of the encoding process affect coding efficiency. For each of the 50 meshes in our test set, we coded the mesh with optimization using each of the three decision rules (namely, rules 1, 2, and 3, as introduced earlier) as well as without optimization, and measured the resulting bit rate in each case. The results obtained are given in Table 3.6, with results for ten individual meshes in Table 3.6(a) and overall statistical results for all meshes in Table 3.6(b). In these tables, the best result in each case is high-

Table 3.6: Comparison of the connectivity coding performance obtained with the various decision functions as well as the unoptimized approach selected. (a) Individual results. (b) Overall results.

(a)

Dataset	Coded Size (bits/vertex)			
	Optimized			
	Rule 1	Rule 2	Rule 3	Unoptimized
A1	0.82	0.84	**0.76**	0.86
A2	1.92	1.93	**1.78**	1.98
A3	2.70	2.64	**2.54**	2.74
A4	0.59	0.59	**0.55**	0.61
A5	1.73	1.79	**1.64**	1.83
B1	7.86	7.66	**7.44**	7.70
B2	7.43	7.20	**6.98**	7.24
B3	9.21	8.98	**8.67**	9.03
B4	7.71	7.58	**7.30**	7.64
B5	8.94	8.70	**8.35**	8.74

(b)

Category	Mean Coded Size (bits/vertex)			
	Optimized			
	Rule 1	Rule 2	Rule 3	Unoptimized
A	1.12	1.13	**1.06**	1.16
B	8.11	7.95	**7.64**	8.02
Overall	4.62	4.54	**4.35**	4.59

lighted in bold font. We note that the above results were obtained with the lexicographic priority scheme and the ACNF sequence coding variant. Similar results were obtained with other priority schemes and sequence coding variants, however.

Examining the results of Tables 3.6(a) and (b), we can clearly observe that rule 3 consistently performs best, leading to the lowest bit rate in all of the individual cases as well as overall. As it turns out, rule 3 outperforms the other two rules as well as the approach without optimization in all 50/50 of the test cases. A more careful analysis of the results shows that rule 3 tends to perform more swap operations, allowing locality to be improved more than with rules 1 and 2.

Based on the above results, we deem the best choice for our sequence optimization strategy in the proposed framework to be rule 3. In passing, we note that a closer examination of the above results also shows the approach without optimization sometimes

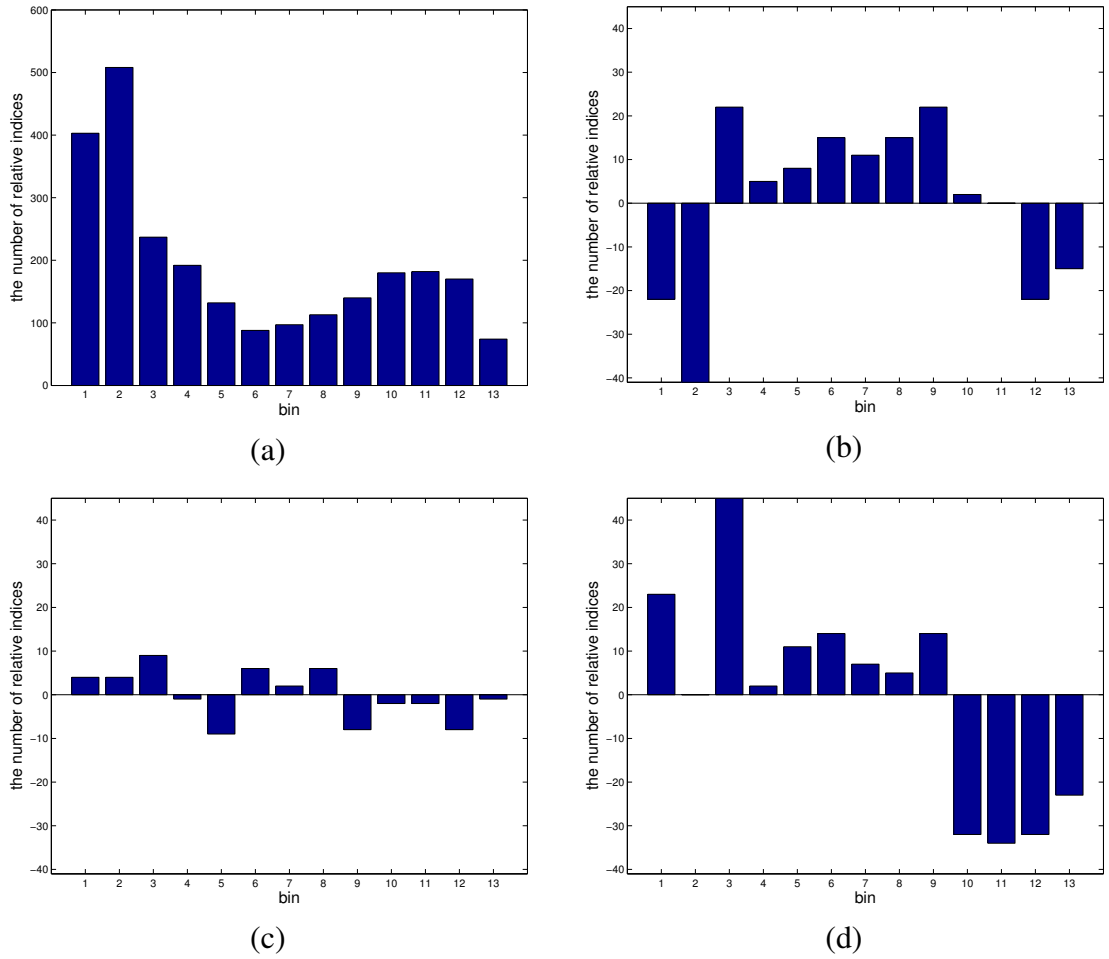


Figure 3.5: Statistical evaluation of relative indexes produced by coding B4 mesh. (a) Histogram of relative indexes generated by the approach without optimization where the integer range of each bin i is $[2^{i-1}, 2^i)$, and each bin measures the number of relative indexes in the range of the bin. (b) Histogram that shows the increase in the number of relative indexes in each bin produced by decision rule 1 relative to the approach without optimization. (c) Histogram that shows the increase in the number of relative indexes in each bin produced by decision rule 2 relative to the approach without optimization. (d) Histogram that shows the increase in the number of relative indexes in each bin produced by decision rule 3 relative to the approach without optimization.

performs better than optimization with rule 1. This is due to the fact that although rule 1 generally reduces the number of very large relative indexes, the number of very small relative indexes also tends to decrease (compared to the approach without optimization). As a consequence, rule 1 can sometimes result in worse coding performance than the approach without optimization.

Lastly, we provide some additional insight into why the various optimization decision rules have the coding performance that they do. In order to do this, we show some additional results for the test case of coding the mesh B4 from earlier in Table 3.6(a). The additional results obtained are shown in Figure 3.5. In particular, Figure 3.5(a) shows the histogram of relative indexes obtained by the approach without optimization, where the integer range of each bin i is $[2^{i-1}, 2^i)$ and each bin measures the number of relative indexes in the range of the bin. In each of Figures 3.5(b), (c) and (d), we provide a histogram that shows the increase in the number of relative indexes in each bin produced by the decision rule relative to the approach without optimization. As we can see in Figure 3.5(b), although rule 1 reduced the number of very large relative indexes (i.e., a behavior that can lead to better coding performance), the number of very small relative indexes are also reduced (compared to the approach without optimization) which can make the coding performance worse. As a result, rule 1 performs worse than the approach without optimization. On the other hand in Figures 3.5(c) and (d), rules 2 and 3 produce more very small as well as less very large relative indexes (compared to the approach without optimization), which can lead to better coding performance than the approach without optimization. Moreover, by comparing the histograms of relative indexes in Figures 3.5(c) and (d), it is clear that rule 3 produces much more very small relative indexes and much less very large relative indexes compared to rule 2. As a consequence, rule 3 performs best among all optimization strategies. Furthermore, we observed that the behaviors we saw in this example (i.e., coding the mesh B4) are typical for other datasets as well. This explains why: rule 1 can sometimes perform worse than the approach without optimization; rules 2 and 3 outperform the approach without optimization; and rule 3 performs best overall.

3.4.4 Proposed Method

As demonstrated by the above experimental results, our proposed mesh-coding framework is most effective when the priority scheme, the sequence coding scheme, and the optimization strategy are chosen as lexicographic, ACNF, and rule 3, respectively. Therefore, we recommend these particular choices for the free parameters in our coding framework. In

the remainder of this thesis, we will refer to our framework with the preceding choices for the free parameters as our proposed method for mesh coding.

3.5 Evaluation of Proposed Mesh-Coding Method

Having introduced our proposed connectivity-coding scheme, we now evaluate its coding performance by comparing it to a straightforward baseline connectivity coding scheme. In particular, the baseline scheme directly encodes edge labels of the edge-flip sequence as n-bit integers where n is fixed and chosen as small as possible (i.e., data is packed).

Proposed vs. baseline. To begin our evaluation, we first compare the coding performance of the proposed method and the method with connectivity coding replaced by the baseline scheme. For each of the 50 mesh models in our test set, we encoded the model using each of the two methods and measured the number of bits required to code the connectivity information as well as the complete mesh. The results obtained are shown in Table 3.7, with results for ten specific meshes in Table 3.7(a) and overall statistical results for all 50 meshes in Table 3.7(b). Since the difference between the two methods is in the connectivity coding alone, we will focus our attention on comparing the connectivity coding numbers from these tables in what follows.

Examining the individual results in Table 3.7(a), we can see that our proposed method outperforms the baseline method in all 10/10 of the test cases by a margin of at least 0.24 bits/vertex (for connectivity coding). The overall results in Table 3.7(b) are consistent with the individual results, with our proposed method significantly outperforming the baseline scheme for meshes from both categories A and B. In fact, a more detailed analysis of the results shows that the proposed method beats the baseline scheme in all 50/50 of the test cases, by a margin of up to 11.56 bits/vertex (for connectivity coding).

Proposed vs. traditional methods. Furthermore, we note that in the case of all 25 meshes in category A, our proposed method consistently requires less than the 3.67 bits/vertex for connectivity coding often cited for more traditional connectivity coding methods [34, 23]. Also, it is easy to see that our proposed method can outperform more traditional methods (such as the edgebreaker algorithm) for meshes with underlying triangulations having PDDT connectivity. For such meshes, our connectivity-coding method has zero cost, while traditional connectivity-coding methods have some nontrivial cost. As the mesh connectivity moves further away from PDDT connectivity, the coding cost of our method increases. Nevertheless, our proposed method will continue to outperform traditional methods as long as the mesh connectivity is not too far from PDDT (or Delaunay)

Table 3.7: Coding performance comparison of the proposed method and the method with connnectivity coding replaced by the baseline scheme. (a) Individual results. (b) Overall results.

(a)

Dataset	Connectivity Size (bits/vertex)		Total Size (bits/vertex)	
	Proposed	Baseline	Proposed	Baseline
A1	**0.76**	1.09	15.07	15.40
A2	**1.78**	3.11	14.86	16.19
A3	**2.54**	6.15	11.38	14.99
A4	**0.55**	0.79	14.95	15.19
A5	**1.64**	2.23	15.62	16.21
B1	**7.44**	13.98	21.84	28.38
B2	**6.98**	12.83	22.59	28.44
B3	**8.67**	16.08	24.28	31.69
B4	**7.30**	12.51	22.94	28.15
B5	**8.35**	14.75	24.33	30.73

(b)

Category	Mean Connectivity Size (bits/vertex)		Mean Total Size (bits/vertex)	
	Proposed	Baseline	Proposed	Baseline
A	**1.06**	1.72	14.87	15.54
B	**7.64**	13.92	24.83	31.11
Overall	**4.35**	7.82	19.85	23.32

connectivity. This situation is of significant practical interest since, in many applications, meshes are often close to having Delaunay or PDDT connectivity. Admittedly, when the mesh connectivity is far from being Delaunay, the traditional methods are likely to be better. Determining the crossover point, at which traditional methods will start to outperform our proposed method, would require a detailed experimental comparison. Due to time constraints, however, it is not feasible in this research to implement one of traditional methods (like edgebreaker) for comparison purposes. For the reasons discussed above, it is clear that our proposed method is superior to traditional methods in many practical situations where near-Delaunay meshes are involved. Moreover, unlike many traditional connectivity coding methods [34, 23], our proposed method is progressive, which can be beneficial in many applications.

Computational complexity. Next, we examine the computational complexity of the proposed method as measured by execution time. For the ten mesh models listed in Ta-

Table 3.8: Computational complexity of the proposed method for meshes in Table 3.1.

Dataset	Encoding Time (s) Spent On		Decoding Time (s)
	step 1, 2, 4 of encoding	Optimization	
A1	5.38	77.91	**3.39**
A2	2.10	28.93	**1.31**
A3	2.65	83.87	**1.74**
A4	1.67	12.83	**1.02**
A5	5.22	74.41	**3.49**
B1	11.10	165.19	**8.03**
B2	8.91	150.08	**6.61**
B3	1.34	15.48	**1.05**
B4	1.41	14.29	**1.01**
B5	1.65	19.11	**1.19**

ble 3.1, we measured the time required for encoding and decoding of each model. The results are presented in Table 3.8. Since the time required for the optimization step during encoding is very significant, the optimization time is accounted for separately in the table. In passing, we note that the above timing results were collected on relatively modest hardware, namely, a computer with a 2.13 GHz Intel Core2 CPU and 4 GB of RAM.

Examining the results in Table 3.8, we can clearly observe that decoding requires less than 9 seconds in all cases and the time spent on encoding excluding the optimization step is comparable to the decoding time. From these results, however, we can also see that the total time for the encoding process (including optimization) is very much greater than the time required by decoding, due to the large amount of time needed for optimization. Fortunately, the optimization step in our framework is optional. So, in applications where computational resources are more constrained, the optimization can be disabled to save computation at a small cost to coding efficiency (as demonstrated by the earlier results in Table 3.6).

Progressive performance. Lastly, as indicated earlier, our proposed coding method is progressive. To illustrate the progressive capability of our coding scheme, we provide a brief example. For four of the meshes from Table 3.1 (namely, A1, B3, A4 and B5), we measured the image reconstruction quality obtained from the decoded mesh model (in terms of PSNR) as a function of bit rate. The results obtained are shown in Figures 3.6(a), (b), (c) and (d), with the right side of each graph corresponding to lossless decoding of the original mesh model. As we can see from the figures, an incrementally better quality image reconstruction (i.e., with higher PSNR) is decoded as the bit rate increases. Such

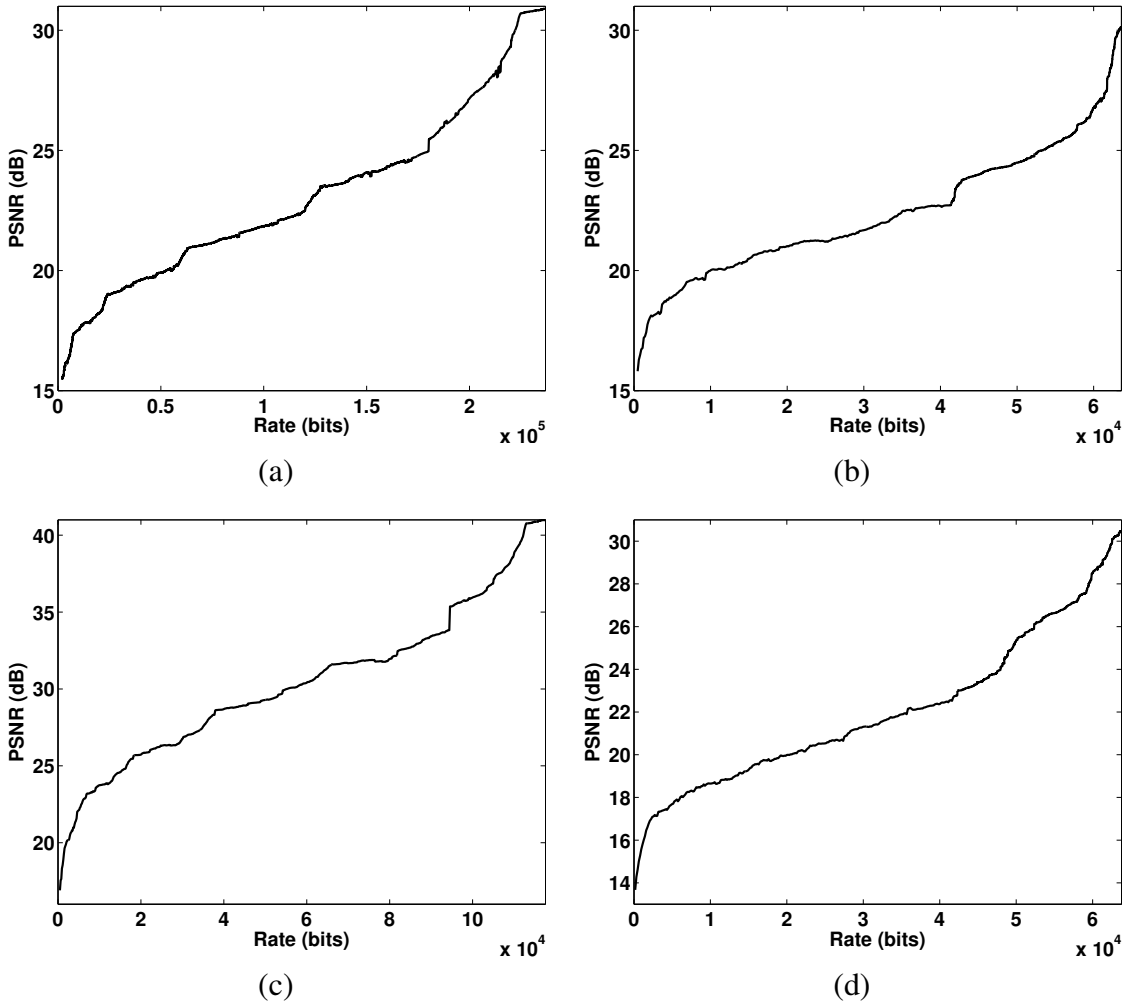


Figure 3.6: Progressive coding results for the (a) A1, (b) B3, (c) A4 and (d) B5 meshes.

functionality is beneficial in many applications.

Chapter 4

Conclusions and Further Research

4.1 Conclusions

In this thesis, a novel progressive mesh-coding framework was proposed that extends the highly efficient IT method by adding to it a means for coding mesh connectivity. As the proposed framework has several free parameters, we studied how different choices of those free parameters affect coding efficiency, leading us to recommend a particular set of choices. Following this, we proposed a new progressive mesh-coding method derived from our framework by employing the recommended set of choices for the free parameters (i.e., lexicographic priority scheme, ACNF sequence coding variant, and optimization using rule 3). Our method is such that its coding efficiency is highest for meshes that have connectivity close to being Delaunay, with performance degrading slowly as connectivity moves further away from Delaunay connectivity.

Through experimental results, our method was shown to be significantly better than a simple baseline coding scheme. Furthermore, we showed that it is likely that our proposed method can outperform traditional connectivity coding methods for meshes that do not deviate too far from Delaunay connectivity (i.e., the relative length of the edge-flip sequence that transforms the mesh connectivity to the PDDT connectivity is less than about 15%). Since Delaunay meshes have many desirable properties for approximation, it is quite common to encounter meshes that are close to being Delaunay in practice. Therefore, the excellent performance of our method for such meshes is of great practical value. In addition, our coding method yields very substantially more compact representations than a simple naive coding scheme like the baseline approach described earlier. Also, our coding method is progressive, which can be beneficial in many applications. For the above

reasons, our proposed coding method can benefit applications that must efficiently store or transmit mesh models of images.

4.2 Future Research

Although this thesis made some significant contributions to the coding of mesh models of images, additional work in this area would still be beneficial. In what follows, some potential areas for future research are discussed.

As mentioned before, in the sequence optimization step of our framework (as described in Section 3.2.1.3), our optimization algorithm aims to optimize the locality of the edge-flip sequence generated by the LOP algorithm to whatever extent is possible. As a result of this optimization, the computational cost of encoding in our framework is quite high. Although we can easily disable optimization in order to code meshes with less computational cost, the coding efficiency will suffer somewhat. Hence, further investigations into finding new optimization algorithms that have lower computational cost would be worthwhile. Also, in the optimization step, only three decision functions were considered. It would be worthwhile to consider other decision functions so that the locality of the edge-flip sequence can be increased. Thus the coding performance would have the potential to be enhanced.

As has been noted, we improved the locality of the edge-flip sequence in the optimization step. It might also be worth of trying to improve the locality of the sequence initially produced by the LOP in the sequence generation step. Our work only considered three priority schemes (namely, FIFO, LIFO and lexicographic) to be used by the LOP. It would be worthwhile to evaluate other priority schemes such that either the length of edge-flip sequence can be reduced or the edge-flip sequence itself can have better locality (i.e., more neighbouring elements in the sequence are associated with edges that are close to one another in the triangulation). In this way, the coding performance can potentially be improved.

Appendix A

Software User Manual

A.1 Introduction

As part of the work described in this thesis, the author developed software that implements the proposed mesh-coding framework with its many options. This implementation was written in C++ and consists of more than 8000 lines of code, which includes some fairly complex data structures and algorithms. The software utilizes several libraries, such as the Boost library, the Computational Geometry Algorithms Library (CGAL), the Signal Processing Library (SPL), and the Signal Processing Library Extensions Library (SPLEL).

Generally speaking, our software consists of two programs, namely, `compression` and `decompression`, each of which provides functionalities in our proposed mesh-coding framework. The `compression` program performs mesh encoding. Given a mesh model, priority scheme, sequence coding variant and optimization strategy as the basic inputs, this program produces encoded mesh connectivity as well as uncompressed mesh geometry information. The `decompression` program performs mesh decoding. Given the coded connectivity and uncoded geometry of an mesh as basic inputs, this program yields a decoded mesh.

To better illustrate how our software fits in the proposed mesh-coding framework, we present the overall structure of our mesh-coding framework in Figure A.1. From this figure, we can clearly see that `compression` and `decompression` programs provide the functionality of coding mesh connectivity. Note that the geometry coders in the framework are provided by the software implementing the IT method (as described in [1]). Thus, our framework is able to code mesh models of images with arbitrary connectivity.

In the sections that follow, we will introduce our software in more detail, including

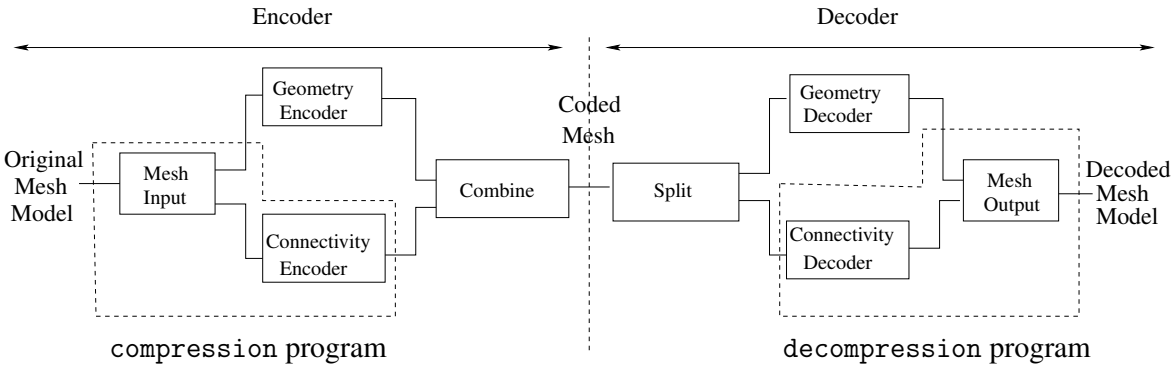


Figure A.1: The overall structure of the mesh-coding framework

information such as: how to build the software, a detailed functional description of the software, examples of using the software, and a description of the source code.

A.2 Building the Software

In order to use our software, it must first be built. The build process is based on the well known software build utility called `make`, which is available on most operating systems. The `make` command builds the executable programs and libraries automatically from source code with the help of makefiles, which specify how the target programs are derived from the source code. As mentioned in Appendix A.1, our software requires some libraries be installed prior to building. To be able to build our software, the user must already have installed:

- CGAL (version 4.2 or later),
- Boost (version 1.49 or later),
- SPL (version 1.1.13 or later), and
- SPLEL (version 1.1.24 or later).

Assuming the necessary libraries are installed, the build process proceeds as follows. To build the software, the current working directory needs to be set as the top level of the software file distribution. To ensure any superfluous files are removed before compiling, execute the command:

```
make clean
```

Then, to compile and link the programs, execute the following command:

```
make
```

This should produce all the executable programs associated with the software.

A.3 Detailed Functional Description of Software

In this section, we present the functional description of our software (`compression` and `decompression`), starting with an introduction to the `compression` program.

A.3.1 The `compression` Program

Synopsis

```
./compression -i $input_mesh -e $encode_method_name \
-c $out_connect -v $out_points [options]
```

Description

The `compression` program compresses the connectivity for a given mesh depending on the priority scheme, sequence coding variant, and optimization strategy. In general, the program is able to read a mesh from an input file in OFF format, encode the mesh connectivity, and write the compressed connectivity data as well as uncompressed geometry information into files.

Options

The `compression` program accepts the options listed below.

`-i` `$input_mesh`
: Specifies the file from which to read the input mesh in OFF format.

`-e` `$encode_method_name`
: Sets the sequence coding variant used for compressing connectivity. The available choices are as listed in Table A.1.

`-c` `$output_connectivity`
: Specifies the file in which to store the encoded connectivity information of the input mesh.

`-v` `$output_vertices`
Specifies the file in which to store the uncompressed geometry information of the input mesh.

`-p` `$priority_method_name`
Sets the priority scheme used by LOP (described in Section 3.2.1.1). The available choices are listed in Table A.2.

`-t` `$output_step`
Specifies the file in which to store the first edge label in the edge-flip sequence and the relative indexes calculated at the sequence encoding process (described in Section 3.2.1.2).

`-s` `$output_stats`
Specifies the file in which to store the statistical evaluation relating to mesh connectivity coding.

`-f` `$output_flip_edge_points`
Specifies the file in which to store each edges' end points in the edge-flip sequence generated by LOP.

`-n` `$output_encoding_time`
Specifies the file in which to store the time consumption of the program.

`-d` `$debug_level`
Sets debug level for compression with choice of 0, 1, 2, 3. The default value is 0 indicating no debugging.

`-g` `$stop_encoding_counter`
Specifies a number to indicate only coding the first `$stop_encoding_counter` edges in the edge-flip sequence. If the value is zero, the entire edge-flip sequence is coded. The default value `$stop_encoding_counter` is 0.

`-o` `$out_encoder_mesh`
Specifies the file in which to store a mesh model in OFF format after coding the edge-flip sequence with a `$stop_encoding_counter` specified.

`-a` `$swapping_method`
Sets the optimization strategy to re-arrange the order of edge flips in the edge-flip sequence produced by LOP. The available choices are listed in Table A.3.

Table A.1: Options for the sequence coding method

Choice	Description
`simple`	Baseline coding scheme (default) introduced in Section 3.5
`gamma`	Gamma sequence coding variant introduced in Section 3.2.1.2
`fibonacci`	Fibonacci sequence coding variant introduced in Section 3.2.1.2
`basic_arithmetic`	Arithmetic-coding-non-finalization sequence coding variant introduced in Section 3.2.1.2
`with_initial_complex_arithmetic`	Arithmetic-coding-finalization-2 sequence coding variant introduced in Section 3.2.1.2
`without_initial_complex_arithmetic`	Arithmetic-coding-finalization-1 sequence coding variant introduced in Section 3.2.1.2

Table A.2: Options for the priority scheme

Choice	Description
`lex`	Lexicographic priority scheme (default) introduced in Section 3.2.1.1
`fifo`	First-in-first-out priority scheme introduced in Section 3.2.1.1
`lifo`	Last-in-first-out priority scheme introduced in Section 3.2.1.1

A.3.2 The `decompression` Program

Synopsis

```
./decompression -c $encoded_connectivity -v $input_vertices \
-o $out_mesh [options]
```

Description

The `decompression` program decompresses the mesh model from its coded connectivity and uncoded geometry information. In general, the program is able to read the encoded connectivity and uncompressed geometry information of a mesh from input files, decode the mesh connectivity, and write the decoded mesh model into a file. Moreover, given an original image as the additional input, this program can produce a reconstructed image

Table A.3: Options for the optimization strategy

Choice	Description
`null`	Approach without optimization on the edge-flips sequence generated by LOP (default)
`prev_next_adjacent_sum`	Decision function rule 1 introduced in Section 3.2.1.3
`prev_next_adjacent_less`	Decision function rule 2 introduced in Section 3.2.1.3
`prev_none_adjacent`	Decision function rule 3 introduced in Section 3.2.1.3

based on the decoded mesh and generate a file storing the PSNR values for reconstructed images obtained from each intermediate triangulation during decoding the edge-flip sequence.

Options

The `decompression` program accepts the options listed below.

`-c` `$encoded_connectivity`
Specifies the file from which to read the encoded connectivity of a mesh.

`-v` `$input_vertices`
Specifies the file from which to read the uncompressed geometry information of a mesh.

`-o` `$output_mesh`
Specifies the file in which to store the decoded mesh model in OFF format.

`-n` `$output_decoding_time`
Specifies the file in which to store the time consumption of the program.

`-d` `$debug_level`
Sets debug level for decompression with choice of 0, 1, 2, 3. The default value is 0 indicating no debugging.

`-w` `$input_original_image_file`
Specifies the file from which to read a PNM image being considered as the original

image when calculating PSNR.

`-z $out_psnr_file`
Specifies the file in which to store the PSNR of reconstructed images obtained for each intermediate triangulation during the decoding of the edge-flip sequence.

`-x $out_delaunay_image_file`
Specifies the file in which to store the reconstructed PNM image obtained for the triangulation before the decoding of the edge-flip sequence.

`-y $out_ddt_image_file`
Specifies the file in which to store the reconstructed PNM image obtained for the triangulation after the decoding of the edge-flip sequence.

`-g $stop_decoding_counter`
Specifies a number to indicate only coding the first `$stop_decoding_counter` edges in the edge-flip sequence. If the value is zero, the entire edge-flip sequence is coded. The default value `$stop_decoding_counter` is 0.

A.4 Examples of Using the Software

In this section, examples are given showing how to use the software.

Example A
Suppose that we want to encode a mesh model stored in the file `B3.off` with the following requirements:

- the priority scheme is chosen as FIFO;
- the sequence coding variant is selected as Fibonacci;
- the optimization strategy used is no optimization; and
- the encoded connectivity and uncompressed geometry information are saved in the flies `encodeConn.dat` and `outPoints.dat`, respectively.

This can be accomplished by running the `compression` program as follows:

```
./compression -i B3.off -p fifo -e fibonacci -a null \
-c encodeConn.dat -v outPoints.dat
```

To decode the mesh, the following command can be used:

```
./decompression -c encodeConn.dat -v outPoints.dat \
-o decodedB3.off
```

This results in the decoded mesh stored in the file `decodedB3.off`.

Example B

Suppose that we want to encode the mesh stored in the file `B3.off` again with the following requirements:

- the priority scheme is chosen as lexicographic;
- the sequence coding variant is selected as arithmetic-coding-non-finalization;
- the optimization strategy used is no optimization; and
- the encoded connectivity and uncompressed geometry information are saved in the flies `encodeConn.dat` and `outPoints.dat`, respectively.

This can be accomplished by running the `compression` program as follows:

```
./compression -i B3.off -p lex -e basic_arithmetic \
-a null -c encodeConn.dat -v outPoints.dat
```

To decode the mesh, the following command can be used:

```
./decompression -c encodeConn.dat -v outPoints.dat \
-o decodedB3.off
```

This results in the decoded mesh stored in the file `decodedB3.off`.

Example C

Suppose that we want to encode a mesh model stored in the file `A1.off` with the following requirements:

- the priority scheme is chosen as lexicographic;
- the sequence coding variant is selected as arithmetic-coding-non-finalization;
- the optimization strategy used is rule 3; and
- the encoded connectivity and uncompressed geometry information are saved in the flies `encodeConn.dat` and `outPoints.dat`, respectively.

This can be accomplished by running the `compression` program as follows:

```
./compression -i A1.off -p lex -e basic_arithmetic \
-a prev_none_adjacent -c encodeConn.dat -v outPoints.dat
```

To decode the mesh and output a reconstructed image based the decoded mesh, the following command can be used:

```
./decompression -c encodeConn.dat -v outPoints.dat \
-o decodedA1.off -y recImage.pnm -w oriImage.pnm
```

This results in the decoded mesh and reconstructed image stored in the files `decodedA1.off` and `recImage.pnm`, respectively.

Example D

Suppose that we want to know the time consumption for coding the connectivity of the mesh model stored in the file `A1.off` with the following requirements:

- the priority scheme is chosen as lexicographic;
- the sequence coding variant is selected as arithmetic-coding-non-finalization;
- the optimization strategy used is rule 3; and
- the encoded connectivity and uncompressed geometry information are saved in the flies `encodeConn.dat` and `outPoints.dat`, respectively.

This can be accomplished by running the `compression` program as follows:

```
./compression -i A1.off -p lex -e basic_arithmetic \
-a prev_none_adjacent \
-c encodeConn.dat -v outPoints.dat -n encodeTime.dat
```

To decode the mesh and output the decoding time spent on mesh connectivity, the following command can be used:

```
./decompression -c encodeConn.dat -v outPoints.dat \
-o decodedA1.off -n decodeTime.dat
```

This results in the decoded mesh and decoding time being stored in the files `decodedA1.off` and `decodeTime.dat`, respectively.

Example E

Suppose that we want to code a mesh stored in the file `B4.off` at an intermediate rate with the following requirements:

- the priority scheme is chosen as lexicographic;
- the sequence coding variant is selected as arithmetic-coding-non-finalization;
- the optimization strategy used is rule 3;
- the number `stop_counter` indicating only encoding the first `stop_counter` edges in the edge-flip sequence is set to 100; and
- the intermediate encoded connectivity and entire geometry information are saved in the flies `encodeConn.dat` and `outPoints.dat`, respectively.

This can be accomplished by running the `compression` program as follows:

```
./compression -i B4.off -p lex -e basic_arithmetic \
-a prev_none_adjacent \
-c encodeConn.dat -v outPoints.dat -g 100
```

To decode the mesh with the option of only decoding the first 50 edges in the edge-flip sequence, the following command can be used:

```
./decompression -c encodeConn.dat -v outPoints.dat \
-o decodedB4.off -g 50
```

This results in the intermediate decoded mesh being stored in the file `decodedB4.off`.

Example F

Suppose that we want to obtain the histogram of relative indexes generated by the proposed mesh-coding method for the mesh stored in the file `B4.off` with the following requirements:

- the priority scheme is chosen as lexicographic;
- the sequence coding variant is selected as arithmetic-coding-non-finalization;
- the optimization strategy used is rule 3; and
- the encoded connectivity and uncompressed geometry information are saved in the flies `encodeConn.dat` and `outPoints.dat`, respectively.

This can be accomplished by running the `compression` program as follows:

```
./compression -i B4.off -p lex -e basic_arithmetic \
-a prev_none_adjacent \
-c encodeConn.dat -v outPoints.dat -t sequence.dat
```

Given `sequence.dat`, we use an algorithm `show_histogram` (source code is briefly introduced in Section A.5) to collect the frequency of numbers at various number ranges. Finally, those frequencies would become inputs of the Matlab bar function to produce the histogram of relative indexes.

A.5 Description of the Source Code

The source code for the software consists of a number of files that provides numerous functionalities. In the list below, each file is introduced briefly.

`coder.hpp`

- contains definitions of classes that includes connectivity coding methods, priority schemes, and several decision function rules

`config.hpp`

- contains some macro definitions

`compression.cpp`

- contains the main function for encoding

`decoder.cpp`

- contains definitions of functions in a `Decoder` class that is used by the main function for decoding

`decoder.hpp`

- contains declarations of functions and data variables in the `Decoder` class

`decompression.cpp`

- contains the main function for decoding

`encoder.cpp`

- contains definitions of functions in an `Encoder` class that are used by the main function for encoding

`encoder.hpp`

- contains declarations of functions and variables in the `Encoder` class

`entropy_coding.cpp`

- contains functions that encode and decode numbers using various entropy coding

`entropy_coding.hpp`

- contains declarations of functions for `entropy_coding.cpp`

`histogram.hpp`

- contains declarations of a `Histogram` class

`RationalNumber.cpp`

- contains definitions of functions in the `RationalNumber` class that represent a number by its numerator and denominator, compute addition, reduction, multiplication, and division for numerator and denominator separately to avoid round-off errors

`RationalNumber.hpp`

- contains declarations of functions and data variables in `RationalNumber.cpp`

`read_write_pnm_image.cpp`

- contains definitions of functions that generate the PNM image from a mesh model of an image, read an image from file to a two-dimensional array class defined in the SPL library, and calculate the peak signal-to-noise ratio (PSNR) value

`read_write_pnm_image.hpp`

- contains declarations of functions in `read_write_pnm_image.cpp`

`show_histogram.cpp`

- contains definitions of functions in histogram class declared in `histogram.hpp`

`triangulation_utility.cpp`

- contains definitions of classes and functions, such as FIFO, LIFO and lexicographic priority schemes, edge labeling, and Lawson local optimization procedure

`triangulation_utility.hpp`

- contains declarations of classes and functions used in the `Encoder` and `Decoder` classes, and definitions of template functions, such as: Lawson optimization, triangulation construction, sort edges, label edges in one triangulation

Bibliography

[1] M. D. Adams. An efficient progressive coding method for arbitrarily-sampled image data. *IEEE Signal Processing Letters*, 15:629–632, 2008.

[2] M. D. Adams. Progressive lossy-to-lossless coding of arbitrarily-sampled image data using the modified scattered data coding method. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Taipei, Taiwan, 2009.

[3] M. D. Adams. A flexible content-adaptive mesh-generation strategy for image representation. *IEEE Trans. on Image Processing*, 20(9):2414–2427, 2011.

[4] M. D. Adams. A highly-effective incremental/decremental delaunay mesh-generation strategy for image representation. *Signal Processing*, 93(4):749–764, 2013.

[5] M. D. Adams. An improved progressive lossy-to-lossless coding method for arbitrarily-sampled image data. In *Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 79–83, Victoria, BC, Canada, August 2013.

[6] L. Alboul, G. Kloosterman, C. Traas, and R. V. Damme. Best data-dependent triangulations. *Journal of Computational and Applied Mathematics*, 119(12):1–12, 2000.

[7] I. Amidror. Scattered data interpolation methods for electronic imaging systems: A survey. *Journal of Electronic Imaging*, 11(2):157–176, 2002.

[8] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

[9] J. G. Brankov, Y. Yang, and N. P. Galatsanos. Image restoration using content-adaptive mesh modeling. In *Proc. of 2003 IEEE International Conference on signal processing*, volume 2, pages 997–1000, 2003.

[10] S. A. Coleman, B. W. Scotney, and M. G. Herron. Image feature detection on content-based meshes. In *Proc. of IEEE International Conference on Image Processing*, volume 1, pages 844–847, 2002.

[11] F. Davoine, M. Antonini, J.-M. Chassery, and M. Barlaud. Fractal image compression based on delaunay triangulation and vector quantization. *IEEE Transactions on Image Processing*, 5(2):338–346, February 1996.

[12] B. N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, 7(6):793–800, 1934.

[13] L. Demaret and A. Iske. Scattered data coding in digital image compression. In *Curve and Surface Fitting: Saint-Malo 2002*, pages 107–117. Nashboro Press, 2003.

[14] C. Dyken and M. S. Floater. Preferred directions for resolving the non-uniqueness of delaunay triangulation. *Computational Geometry*, 34:96–101, May 2006.

[15] N. Dyn, D. Levin, and S. Rippa. Data Dependent Triangulations for Piecewise Linear Interpolation. *IMA Journal of Numerical Analysis*, 10:137–154, 1990.

[16] N. Dyn, D. Levin, and S. Rippa. Boundary correction for piecewise linear interpolation defined over data-dependent triangulations. *Journal of Computational and Applied Mathematics*, 39(2):179–192, 1992.

[17] N. Dyn and S. Rippa. Data-dependent triangulations for scattered data interpolation and finite element approximation. *Applied Numerical Mathematics*, 12(13):89–105, 1993.

[18] P. Elias. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on*, 21(2):194–203, Mar 1975.

[19] A. S. Fraenkel and S. T. Klein. Robust universal complete codes for transmission and compression. *Discrete Applied Mathematics*, 64:31–55, 1996.

[20] R. Franke and G. Nielson. Scattered data interpolation and applications: A tutorial and survey. In *Geometric Modeling*, Computer Graphics Systems and Applications, pages 131–160. Springer Berlin Heidelberg, 1991.

[21] P. G. Howard and J. S. Vitter. New methods for lossless image compression using arithmetic coding. In *Data Compression Conference, 1991. DCC '91.*, pages 257–266, Apr 1991.

[22] K.-L. Hung and C.-C. Chang. New irregular sampling coding method for transmitting images progressively. *IEE Proceedings Vision, Image and Signal Processing*, 150(1):44–50, 2003.

[23] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Canadian Conference on Computational Geometry*, Vancouver, BC, Canada, 1999. Available online http://www.cccg.ca/proceedings/1999/c39.pdf.

[24] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1972.

[25] C. L. Lawson. Software for c1 surface interpolation. In J. R. Rice, editor, *Mathematical Software III*, pages 161–194. 1977.

[26] P. Lechat, H. Sanson, and L. Labelle. Image approximation by minimization of a geometric distance applied to a 3D finite elements based model. In *Proc. of IEEE International Conference on Signal Processing*, volume 2, pages 724–727, 1997.

[27] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19:261–296, 1987.

[28] P. Li and M. D. Adams. A tuned mesh-generation strategy for image representation based on data-dependent triangulation. *IEEE Transactions on Image Processing*, 22:2004–2018, May 2013.

[29] D. J. C. MacKay. Information theory, inference, and learning algorithms. chapter The Source Coding Theorem, page 81. Cambridge: Cambridge University Press, 2003.

[30] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

[31] E. Osherovich and A. M. Bruckstein. All trangulations are reachable via sequences of edge-flips: an elementary proof. *Computer Aided Geometric Design*, 25:157–161, March 2008.

[32] M. Petrou, R. Piroddi, and A. Talebpour. Texture recognition from sparsely and irregularly sampled data. *Computer Vision and Image Understanding*, 102(1):95–104, 2006.

[33] G. Ramponi and S. Carrato. An adaptive irregular sampling algorithm and its application to image coding. *Image Vision Computing*, 19:451–460, 2001.

[34] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5:47–61, 1999.

[35] A. Said. *Introducing to Arithmetic Coding - Theory and Practice*. HPL-2004-76. Imaging Systems Laboratory, HP Laboratories Palo Alto, April 2004.

[36] M. Sarkis and K. Diepold. A fast solution to the approximation of 3d scattered point data from stereo images using triangular meshes. In *Proc. of 7th IEEE-RAS International Conference on Humanoid Robots*, pages 235–241, Pittsburgh, PA, USA, November 2007.

[37] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.

[38] S. Srikanth and S. Meher. Compression efficiency for combining different embedded image compression techniques with huffman encoding. In *Communications and Signal Processing (ICCSP), 2013 International Conference on*, pages 816–820, April 2013.

[39] D. Su and P. Willis. Image interpolation by pixel-level data-dependent triangulation. *Computer Graphics Forum*, 23(2):189–201, 2004.

[40] X. Tu and M. D. Adams. Image representation with explicit discontinuities using triangle meshes. In *Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 97–101, Victoria, BC, Canada, 2011.

[41] W. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.

[42] Y. Wang, O. Lee, and A. Vetro. Use of two-dimensional deformable mesh structures for video coding, part ii-the analysis problem and a region-based coder employing an active mesh representation. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(6):636–646, 1996.

[43] T. Wiegand and H. Schwarz. Source coding: Part I of fundamentals of source and video coding. *Foundations and Trends in Signal Processing*, 4(12):1–222, 2011.

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

[45] Y.-Y. Yang, M. N. Wernick, and J. G. Brankov. A fast approach for accurate content-adaptive mesh generation. *IEEE Trans. on Image Processing*, 12(8):866–881, August 2003.