Image Representation with Explicit Discontinuities Using Triangle Meshes

by

Xi Tu
B.Sc., Shanghai Jiao Tong University, 2008

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

MASTER OF APPLIED SCIENCE

in the Department of Electrical Engineering

Image Representation with Explicit Discontinuities Using Triangle Meshes

by

Xi Tu
B.Sc., Shanghai Jiao Tong University, 2008

Supervisory Committee

---

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

---

Dr. Wu-Sheng Lu, Departmental Member
(Department of Electrical and Computer Engineering)

**Supervisory Committee**

---

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

---

Dr. Wu-Sheng Lu, Departmental Member
(Department of Electrical and Computer Engineering)

## ABSTRACT

Triangle meshes can provide an effective geometric representation of images. Although many mesh generation methods have been proposed to date, many of them do not explicitly take image discontinuities into consideration. In this thesis, a new mesh model for images, which explicitly represents discontinuities (i.e., image edges), is proposed along with two corresponding mesh-generation methods that determine the mesh-model parameters for a given input image. The mesh model is based on constrained Delaunay triangulations (DTs), where the constrained edges correspond to image edges.

One of the proposed methods is named explicitly-represented discontinuities-with error diffusion (ERDED), and is fast and easy to implement. In the ERDED method, the error diffusion (ED) scheme is employed to select a subset of sample points that are not on the constrained edges. The other proposed method is called ERDGPI. In the ERDGPI method, a constrained DT is first constructed with a set of prespecified constrained edges. Then, the greedy point insertion (GPI) scheme is employed to insert one point into the constrained DT in each iteration until a certain number of points is reached.

The ERDED and ERDGPI methods involve several parameters which must be provided as input. These parameters can affect the quality of the resulting image approximations, and are discussed in detail. We also evaluate the performance of our proposed ERDED and ERDGPI methods by comparing them with the highly effective ED and GPI schemes. Our proposed methods are demonstrated to be capable of producing image approximations of higher quality both in terms of PSNR and subjective quality than those generated by other schemes. For example, the reconstructed images produced by the proposed ERDED method are often about 3.77 dB higher in PSNR than those produced by the ED scheme, and

our proposed ERDGPI scheme produces image approximations of about 1.08 dB higher PSNR than those generated by the GPI approach.

# Contents

# List of Tables

# List of Figures

# List of Acronyms

**ED** Error Diffusion

**DT** Delaunay Triangulation

**GPI** Greedy Point Insertion

**ERD** Explicit Represented Discontinuities

**GPR** Greedy Point Removal

**PSNR** Peak Signal-to-Noise Ratio

**HVS** Human Visual System

**CGAL** Computational Geometry Algorithms Library

**PSLG** Planar Straight Line Graph

**1-D** One Dimensional

**2-D** Two Dimensional

**MSE** Mean Squared Error

**MMSODD** Maximum Magnitude Second Order Directional Derivative

# ACKNOWLEDGMENTS

This thesis would never have been done without the help and support from numerous people. I would like to take this opportunity to express my thankfulness to a few people in particular. I would like to thank:

**First and foremost, my supervisor Dr. Michael Adams.** Thank you for your mentoring, support, encouragement and patience. Thank you for leading me into the interesting and promising realm of image processing and computational geometry. Thank you for providing me an excellent environment to learn and practice C++ programming and Linux system. Thank you for your tireless and meticulous when helping me on the course study, on the difficulties I encountered in research, and even on the improvement of my technical writing skills. Your patient guidance in my project, image representation with explicit discontinuities using triangle meshes, is the pivotal step to the completion of this thesis. You are not only a good supervisor but also a great friend. Your energetic and aggressive attitude toward research, your hard-working, your high sense of responsibility and the care for your students, are the most important things I benefit from you, that will be helpful in my future career.

**Dr. Wu-Sheng Lu, Dr. Andreas Antoniou, Dr. Panajotis Agathoklis and Dr. Alexandra Branzan Albu.** I was very lucky to have the opportunities to take your courses. Thank you for your patient and kindness guidance in my course work and course project. Your plentiful professional knowledge, high sense of responsibility and kindness to your students are the most important that I benefit.

**Dan Li, Ping Li, Yuzhe Yao, Jie Yan, Ioana Sevcenco, Chamira Edussooriya, Moyuan Chen, Yang Song, Chenyuan Wang, Zhuang Zhuang Tian, Lebing Liu, Congzhi Liu and all other friends.** Thank you for your friendship and kindly help during my master studies. The friendship with you made all the difference during these two years. Dan Li, thank you for your accompany with me, and thank you for the inspiration and power you gave me. It is the most precious treasure of my life.

**Vicky Smith, Moneca Bracken, Dan Mai, Lynn Barrett and Janice Closson.** You help me with all the important, but numerous and trivial stuffs during my graduate study. Thank you for your kindness and care.

**NSERC and University of Victoria.** Thank you for providing me financial support in the form of a research grant and a university fellowship, respectively. The support

definitely helped me resolve the financial burden pertaining to my studies.

*I believe I know the only cure, which is to make one's centre of life inside of one's self, not selfishly or excludingly, but with a kind of unassailable serenity-to decorate one's inner house so richly that one is content there, glad to welcome any one who wants to come and stay, but happy all the same in the hours when one is inevitably alone.*

Edith Wharton

# Chapter 1

# Introduction

## 1.1 Mesh Modelling and Mesh Generation

In the last several years, there has been a growing interest in geometric representations for images, that is, the modelling of images using geometric primitives. This is largely due to the fact that, geometric representations are content-adaptive and capable of capturing geometric structure in images. In contrast, when regular sampling (i.e., lattice-based sampling) is employed, the sampling density is too low in regions where the signal is changing rapidly, and too high in regions where the signal is varying slowly or not at all. Geometric representations allow the modelling of large areas of pixels with basic geometric primitives. For example, a large region can be represented by a few polygons instead of by hundreds of pixels. Various image representations that attempt to exploit the geometric structure in images have been studied and developed, including ridgelets [1], curvelets [2], framelets [3], edgelets [4], contourlets [5] [6], wedgelets [7], bandlets [8], and normal meshes [9] [10].

Among the numerous geometric representations, those based on polygonal meshes (e.g., triangle or quadrilateral meshes) [11] [12] [13], especially those based on triangle meshes [10] [14], have received considerable attention and proven to be particularly effective. Such representations for images are known as **mesh models**. Mesh models have many advantages. They often have greater compactness; they allow for some operations on images to be performed more easily; and they are able to facilitate methods that yield higher quality approximations. There are numerous applications that can benefit from image representations based on mesh models, such as computer vision [15], pattern recognition [16], image restoration [17], and image/video compression [18–23].

If a mesh is to be used to represent an image, however, we typically need a way to

choose a good subset of sample points from the original image to form a mesh approximation. This is the so called the **mesh generation** problem. Due to the necessity for solutions to this problem, mesh generation methods are of fundamental importance.

## 1.2 Historical Perspective

Due to the numerous advantages of mesh modelling, a great many of mesh generation methods and frameworks have been developed over the years. Most of them can be categorized into two classes, the methods that determine all the sample points in one step and the mesh-refinement based methods. Yang, Wernick, and Brankov proposed a scheme named error diffusion (ED) [13] which determines all the sample points in one step. It is fast and easy to implement. The ED method uses Floyd-Steinberg error diffusion [24] to generate a set of sample points, distributed such that the local density of sample points is proportional to the largest magnitude second-order directional derivative of the image.

Different from the methods that determine all the sample points in one step, the mesh-refinement based schemes begin with an initial model of the image (such as a coarse, regular mesh), then refine the model iteratively until either a certain error (e.g., absolute error or squared error) between the current mesh approximation and the original image is reached or a certain number of sample points is obtained [25–28]. For example, the greedy point insertion (GPI) scheme of Adams [29] inspired by the work of Garland and Heckbert [10] is a mesh-refinement based algorithm. It first chooses a triangulation of the image bounding box to form an initial approximation, and then repeatedly inserts the sample point corresponding to the largest absolute error into the triangulation. The greedy point removal (GPR) scheme of Demaret and Iske [30], in contrast, first constructs a DT of all of the sample points of the image, and then repeatedly removes the point that yields the smallest increase in the squared error of the mesh approximation. Various types of triangle meshes can be used as the model in mesh-refinement based methods, such as those based on Delaunay triangulations (DTs) [31] and data-dependent triangulations [32] [33]. The DT minimizes the maximum interior angle of all triangles in the triangulation, consequently avoiding sliver (i.e., long-thin) triangles to whatever extent is possible. The data-dependent triangulation uses the values of the image function at the sample points in addition to their positions [32, 33] to determine triangulation connectivity in order to reduce approximation errors. Data-dependent triangulations can achieve lower approximation error than DTs, but tend to be associated with methods of higher complexity.

The methods that determine all sample points in one step are typically faster than the

mesh-refinement based methods. The quality of image approximations produced by the methods that determine all sample points in one step, however, are typically lower than those generated by the mesh-refinment based schemes. To obtain a high quality mesh in an efficient manner, some methods, that combine the advantages of both kinds of the methods, have been developed. One of these methods is proposed by Adams, called GPR from subset with ED (GPRFS-ED) [14]. It first uses the ED scheme to select a certain number of sample points, and then employs the GPR method to remove unnecessary points until the desired mesh size is reached. It is faster than the GPR scheme and can yield image approximations of higher quality than those generated by the ED scheme.

To evaluate the performance of the mesh-generation methods, peak signal-to-noise ratio (PSNR) is usually employed to measure the objective quality of the resulting image approximations. Besides objective measures (e.g., PSNR), subjective measures also play an important role in evaluating the quality of a mesh approximation. Human eyes are often drawn more to high frequencies than low frequencies. This is equivalent to saying that, human eyes are quite sensitive to image discontinuities (i.e., image edges). Therefore, mesh generation methods that can explicitly represent image discontinuities [34] [35] are of great interest. Among the numerous possibilities, the constrained DT is a good choice for modelling image discontinuities, since image edges can be handled as constrained edges in a constrained DT.

## 1.3 Overview and Contribution of the Thesis

In this thesis, we first introduce a mesh model that explicitly represents image discontinuities (i.e., image edges). Then, to select the model parameters for a given input image, we propose two mesh generation methods, called explicitly-represented discontinuities with ED (ERDED) and explicitly-represented discontinuities with GPI (ERDGPI), which employ the ED and GPI schemes respectively to select a subset of the sample points. The proposed methods are shown to be capable of producing mesh approximations of higher quality than previously-proposed highly-effective mesh generation schemes in terms of both objective and subjective measures.

The remainder of this thesis is organized as follows. Chapter 2 introduces the background necessary to understand the work presented in this thesis. Some of the notation and terminology used herein are presented, followed by background information pertaining to computational geometry (e.g., triangulations and polyline simplification) and image processing (e.g., edge detection, anti-aliasing and super-sampling). At last, we introduce

some fundamental concepts related to mesh modelling, including mesh generation, scan conversion and evaluation of image approximations.

In Chapter 3, we first introduce two highly effective mesh generation methods, namely the ED scheme of Yang, Wernick, and Brankov, and the GPI scheme of Adams. After that, we mention that the discontinuity (i.e., image edge) is an important feature of an image and easy to be captured by human eyes. Although many state-of-the-art mesh generation methods have been proposed to date, many of them do not take image discontinuities into consideration explicitly. We then propose a new mesh model, based on the constrained DT, that explicitly represents image discontinuities. After that, two mesh-generation approaches, that select mesh model parameters for a given image, are proposed and analyzed. One of the proposed methods is named ERDED, which employs the ED scheme to select a subset of sample points that are not on the constrained edges. The other proposed method is called ERDGPI, where the GPI scheme is employed to select a subset of the sample points. The ERDED and ERDGPI methods require the specification of several parameters. We also propose an automated scheme which chooses these parameters effectively. Through experimental results, we evaluate the performance of our proposed ERDED and ERDGPI methods. PSNR is used to measure the quality of the reconstructed images in our work. By comparison, we find that the image approximations produced by our proposed ERDED method are often about 3.77 dB higher in PSNR than those produced by the ED scheme, and our proposed ERDGPI method can generate reconstructed images of about 1.08 dB higher PSNR than those produced by the GPI scheme. Since our proposed methods explicitly model image edges, the resulting image approximations not only have higher PSNR, but also have higher subjective quality.

Chapter 4 concludes the thesis by summarizing the results presented in this thesis and suggesting some related topics for future research.

# Chapter 2

# Preliminaries

## 2.1 Overview

To facilitate a better understanding of the work presented in this thesis, some fundamental concepts related to this work are introduced in this chapter. We begin with an introduction to some of the notation and terminology used herein. Then, we present some basic concepts from computational geometry and image processing. At last, we introduce some rudimentary concepts related to mesh modelling, including mesh generation, scan conversion and mesh evaluation.

## 2.2 Notation and Terminology

Before proceeding further, a brief digression is in order concerning the notation and terminology used herein. The sets of integers and real numbers are denoted as $\mathbb{Z}$ and $\mathbb{R}$, respectively. The notation $(a,b)$, $[a,b)$, $(a,b]$, and $[a,b]$ denote the open interval $\{x \in \mathbb{R} : \ a < x < b\}$, the half-closed half-open interval $\{x \in \mathbb{R} : \ a \leq x < b\}$, the half-open half-closed interval $\{x \in \mathbb{R} : \ a < x \leq b\}$, and the closed interval $\{x \in \mathbb{R} : \ a \leq x \leq b\}$, respectively.

For $\alpha \in \mathbb{R}$, the notation $\lfloor \alpha \rfloor$ and $\lceil \alpha \rceil$ denote the largest integer no more than $\alpha$ (i.e., the floor function) and the smallest integer no less than $\alpha$ (i.e., the ceiling function), respectively. For $m,n \in \mathbb{Z}$, we define the **mod** function as $\text{mod}(m,n) = m - n \left\lfloor \frac{m}{n} \right\rfloor$. The cardinality of a set $S$ is denoted $|S|$.

Matrices and vectors are denoted by uppercase and lowercase boldface letters, respectively. The Euclidean norm of $\mathbf{v} = [v_0, v_1, \cdots, v_{n-1}]^T$ is denoted by $||\mathbf{v}||$, which is defined

as

$$||\mathbf{v}|| = \sqrt{v_0^2 + v_1^2 + \cdots + v_{n-1}^2}. \tag{2.1}$$

For matrices, their dimensions are specified by the subscripts. More specifically, $\mathbf{M}_n$ denotes an $n \times n$ matrix, and an $m \times n$ matrix is denoted as $\mathbf{M}_{m\times n}$.

For a function $f$ defined on $\mathbb{R}^2$, its gradient, denoted $\bigtriangledown f$, is defined as

$$\bigtriangledown f(x,y) = \begin{bmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} \end{bmatrix}^T. \tag{2.2}$$

The Laplacian of $f$, denoted $\triangle f$, is defined as

$$\triangle f(x,y) = \bigtriangledown^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{2.3}$$

For two complex-valued functions $f$ and $g$ defined on $\mathbb{R}$, their convolution, denoted as $f * g$, is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau. \tag{2.4}$$

Furthermore, for sequences $f$ and $g$ defined on $\mathbb{Z}$, their convolution is given by

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n-m]. \tag{2.5}$$

One of the extensively used numeric measurements, known as **peak signal-to-noise ratio** (PSNR), is the ratio between the maximum possible magnitude of a signal and the magnitude of corrupting noise. Since many signals have a very wide dynamic range, PSNR is usually expressed in terms of the logarithmic decibel (dB) scale.

To help formally define the PSNR, the **mean squared error** (MSE) is introduced. For an image $\phi$ of size $m \times n$ and its approximation $\hat{\phi}$, the MSE is computed as

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [\phi(i,j) - \hat{\phi}(i,j)]^2. \tag{2.6}$$

The PSNR is then defined by

$$\text{PSNR} = 20\log_{10}\left(\frac{2^p - 1}{\sqrt{\text{MSE}}}\right), \tag{2.7}$$

where $p$ is the number of bits per sample in image $\phi$.

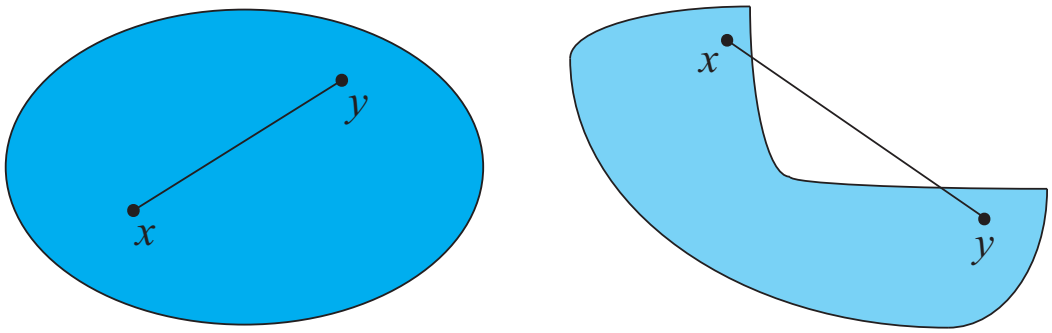
## 2.3 Computational Geometry

The process of mesh modelling involves numerous geometric algorithms, including triangulation and polyline simplification. In what follows, background related to triangulations are presented first, followed by the introduction of the Douglas-Peucker polyline simplification algorithm and a C++ open source library for computational geometry, known as the Computational Geometry Algorithms Library (CGAL).

### 2.3.1 Triangulations

Triangulation is one of the fundamental concepts extensively used in computational geometry. Particularly, it is an important concept in geometric image representations. In what follows, we first introduce the concepts of convex set and convex hull, followed by the formal definition of a triangulation [36] [37].

**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 $p,q \in P$, the straight line segment $\overline{pq}$ that joins $p$ and $q$ is also contained in $P$.*

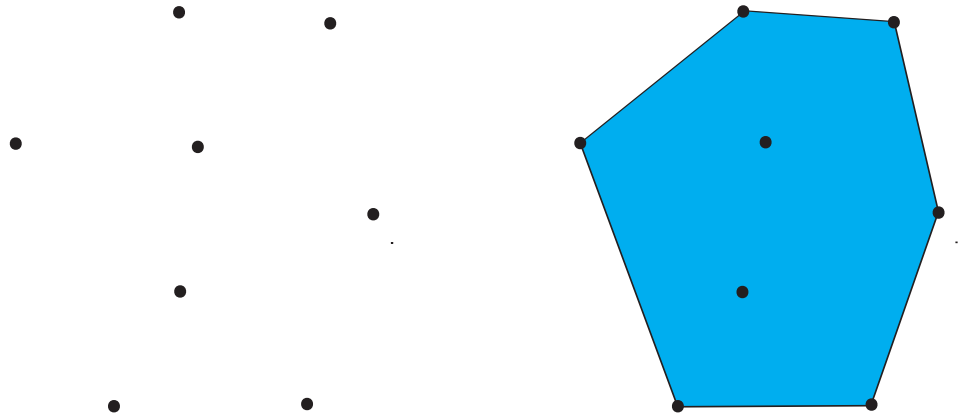



(a) (b)

Figure 2.1: Example of sets that are (a) convex and (b) not convex.

The definition of a convex set is illustrated in Figure 2.1. We can see from Figure 2.1(a) that, for every pair of points $x$, $y$ in the set, the line segment $\overline{xy}$ is also in the set. Therefore, the set in Figure 2.1(a) is convex. In the set shown in Figure 2.1(b), there exists a pair of points $x$ and $y$ that, the line segment $\overline{xy}$ that joins $x$ and $y$ is not in the set shown in Figure 2.1(b). Therefore, the set in Figure 2.1(b) is not convex.

**Definition 2.2** (Convex hull)**.** *The convex hull of a set $P$ of points is the intersection of all convex sets that contain $P$ (i.e., it is the smallest convex set containing $P$).*



(a) (b)

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

The definition of the convex hull is illustrated in Figure 2.2. Figure 2.2(a) shows a set $P$ of points in $\mathbb{R}^2$, and Figure 2.2(b) depicts the convex hull of $P$. The notion of convex hull is helpful for defining a triangulation.

**Definition 2.3** (Triangulation)**.** *A triangulation of a set $P$ of points in $\mathbb{R}^2$ is a subdivision of the convex hull of $P$ into a set $T$ of triangles such that the interiors of any two triangles in $T$ never intersect, and the set of points that are the vertices of $T$ coincides with $P$.*

For a given set $S$ of points, there are typically very many triangulations of $S$. For the set $P$ of points in Figure 2.3(a), two triangulations of $P$ among the numerous possibilities are illustrated in Figure 2.3(b) and Figure 2.3(c). We can see that, the edges of the triangulation in Figure 2.3(b) and those of the triangulation in Figure 2.3(c) are different. Triangulations are a key ingredient for mesh representations of images. An approximation of an image can be easily formed when the image domain is partitioned into triangles, since a large region in the image domain might be represented by a few triangles instead of by hundreds of pixels.

Various types of triangulations have been proposed over the years. One important and widely used type of triangulation, which has a number of useful properties, is the Delaunay triangulation (DT) [31]. To help define a DT, it is necessary to introduce the concept of a circumcircle. In geometry, the circumcircle of a triangle is the unique circle that passes through all three vertices of the triangle. With this in mind, the definition of a DT is then as follows.

**Definition 2.4** (Delaunay triangulation (DT))**.** *A triangulation $T$ is said to be Delaunay if each triangle in $T$ is such that the interior of its circumcircle contains no vertices of $T$.*

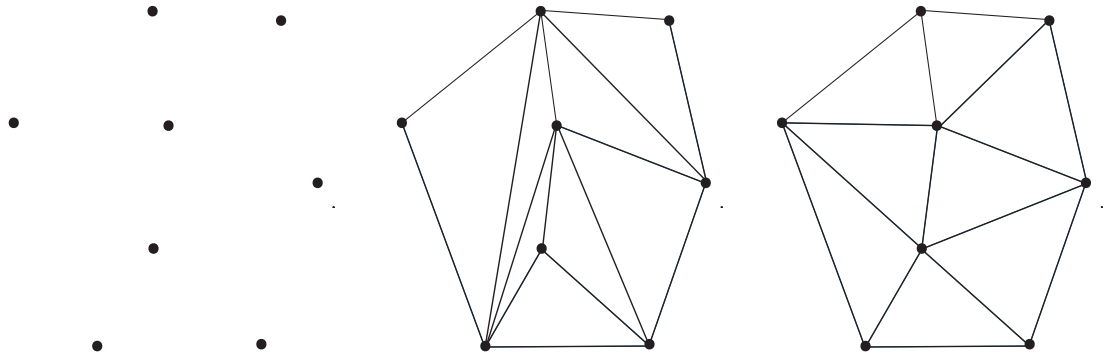



Figure 2.3: Triangulation example. (a) A a set $P$ of points, (b) a triangulation of $P$, and (c) another triangulation of $P$.

An example of a DT is illustrated in Figure 2.4. The circumcircles of the triangles in the triangulation are also shown by dashed lines in the figure. As shown in the figure, in the triangulation, each circumcircle contains no vertices of the triangulation in its interior. Therefore, the triangulation in Figure 2.4 is a DT. A DT maximizes the minimum interior angle of all triangles in the triangulation [36]. Consequently, it tends to avoid sliver (i.e., long-thin) triangles. The DT of a set of points is not guaranteed to be unique. More specifically, the DT of a set of points is guaranteed to be unique if no four points in the set are co-circular. In the case that a set of points is a subset of a rectangular array of points (e.g. raster image), there will typically be many co-circular points and therefore multiple DTs. Some methods have been proposed for choosing a unique DT from the numerous possibilities in those situations, including the symbolic perturbation method discussed in [36,38–40] and the preferred directions approach presented in [41].

In some applications, we want certain prescribed edges to appear in a triangulation. The prescribed edges (i.e., line segments) that are imposed during the triangulation process are called **constrained edges**. A triangulation with constrained edges is called a **constrained triangulation** [42]. To help understand the concept of constrained triangulation, we first introduce the notion of a planar straight line graph (PSLG), which is defined as below.

**Definition 2.5** (Planar straight line graph (PSLG))**.** *A planar straight line graph is a set $P$ of points in $\mathbb{R}^2$ and a set $E$ of line segments denoted $(P,E)$, such that: each line segment of $E$ must have its endpoints in $P$, and any two line segments of $E$ must either be disjoint or intersect at most at a common endpoint.*

An example of a PSLG is shown in Figure 2.5. The PSLG in the figure consists of a set of eight points and a set of two constrained edges. A constrained triangulation can be

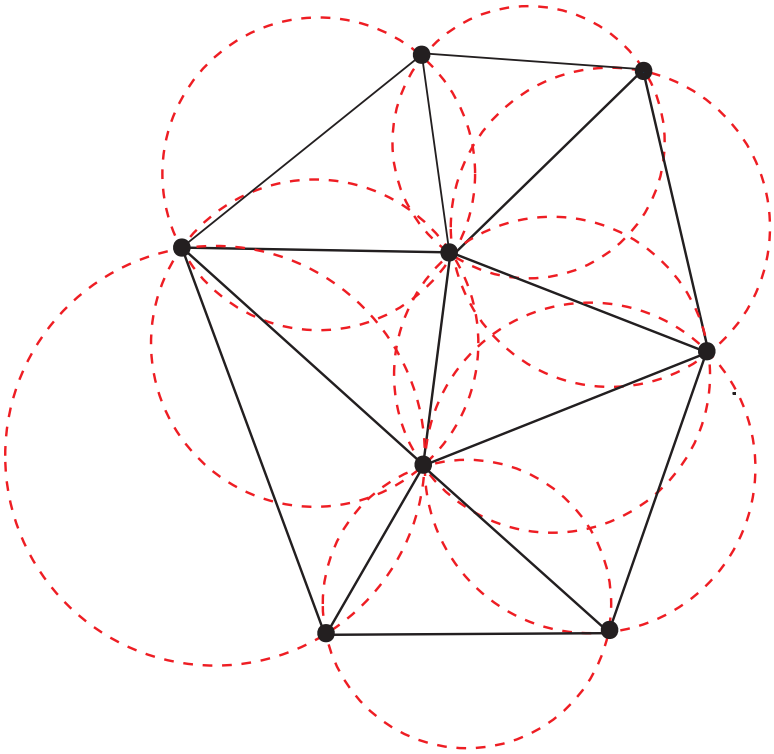


Figure 2.4: Example of a DT. The circumcircle of each triangle contains no vertices of the DT.

considered as a triangulation of the point set $P$ of a PSLG, where the line-segment set $E$ of the PSLG correspond to constrained edges.

Having introduced the concepts of DT and constrained triangulation, we now present a triangulation known as constrained DT [42]. The constrained DT combines the features of a constrained triangulation and a DT. Therefore, constrained DTs are extensively used in many applications.

To help define a constrained DT, the notion of **visibility** must first be introduced. In a triangulation, two points $p$ and $q$ are visible if and only if the line segment $\overline{pq}$ does not intersect any constrained edge. Furthermore, a point $p$ is said to be visible from the interior of a triangle $\triangle t$ if and only if, for every point $q$ inside $\triangle t$, the line segment $\overline{pq}$ does not intersect any constrained edge. Figure 2.6(a) gives an example of a point $p$ that is visible from the interior of a triangle $\triangle t$. We can see from the figure that, for any point $q$ in $\triangle t$, there is no line segment $\overline{pq}$ intersecting a constrained edge. Thus, $p$ is visible from the interior of $\triangle t$. An example of a point $p$ that is not visible from the interior of a triangle $\triangle t$ is illustrated in Figure 2.6(b). In this figure, the line segment $\overline{pq}$ intersects with the constrained edge $\overline{ab}$ at point $o$. Thus, $p$ is not visible from the interior of $\triangle t$. Having

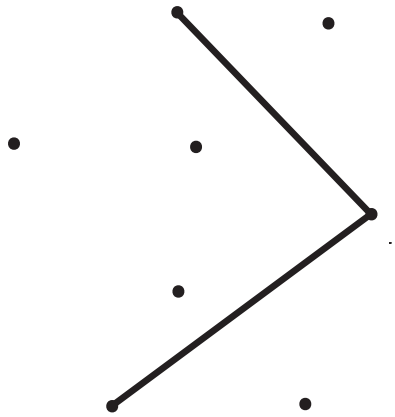
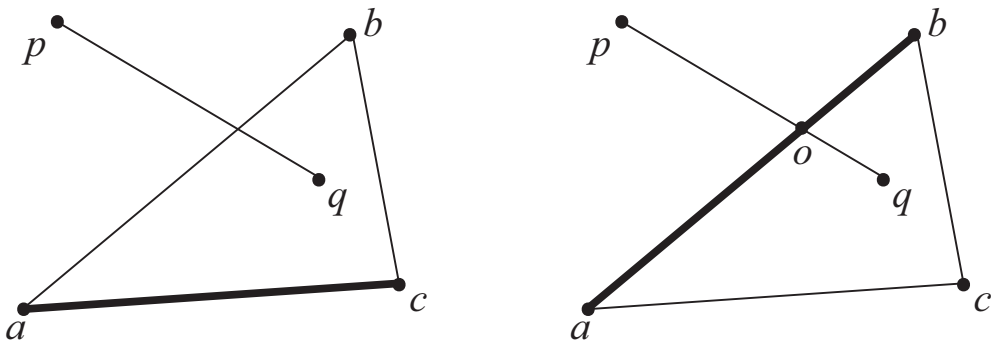


Figure 2.5: Example of a PSLG with two constrained line segments.

introduced the concept of visibility, we now define a constrained DT as follows.

**Definition 2.6** (Constrained DT)**.** *Given a PSLG* $(P,E)$*, a triangulation of P is said to be constrained Delaunay if each triangle* $T$ *in the triangulation is such that the interior of* $T$ *does not intersect any constrained edge in* $E$*; and no vertex inside the circumcircle of* $T$ *is visible from the interior of* $T$*.*

An example of a constrained DT is shown in Figure 2.7. Figure 2.7(a) and (b) show the given PSLG and its corresponding constrained DT, respectively. The thick line segments $\overline{ae}$ and $\overline{eg}$ in Figure 2.7(b) are the constrained edges of the constrained DT, which correspond to the set $E$ of line segments in the PSLG illustrated in Figure 2.7(a). In Figure 2.7(b), the circumcircle of each triangle in the triangulation is drawn with a dash line as well. As shown in the figure, the circumcircles of triangles $\triangle abe$, $\triangle ade$, $\triangle efg$ and $\triangle egh$ have vertices of the triangulation in their interiors. Point $d$ is inside the circumcircle of triangle




(a) (b)

Figure 2.6: Examples of visibility. (a) $p$ is visible from the interior of triangle $\triangle t$, and (b) $p$ is not visible from the interior of triangle $\triangle t$.

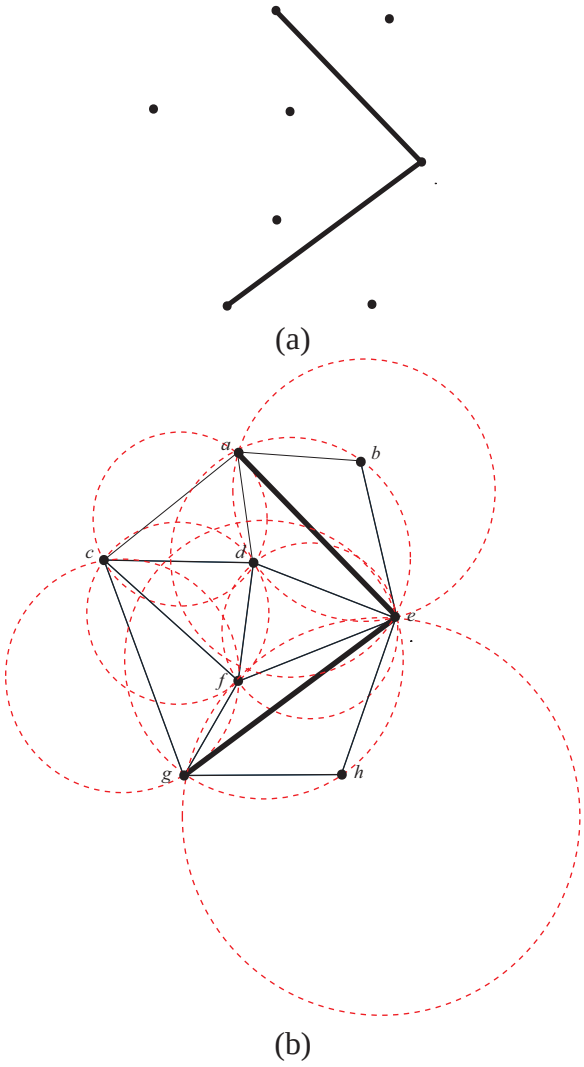



Figure 2.7: Example of a constrained DT. (a) Given PSLG, and its corresponding (b) constrained DT.

$\triangle abe$. Since the line segments connecting point $d$ and any point inside the triangle intersects the constrained segment $\overline{ae}$, $d$ is not visible from the interior of the triangle $\triangle abe$. Similarly, point $b$, which is inside the circumcircle of triangle $\triangle ade$, is not visible from the interior of triangle $\triangle ade$; point $h$, which is inside the circumcircle of triangle $\triangle efg$, is not visible from the interior of triangle $\triangle ade$; and points $d$ and $f$, which are inside the circumcircle of triangle $\triangle egh$, are not visible from the interior of triangle $\triangle egh$. Therefore, the triangulation in Figure 2.7(b) is a constrained DT. Similar to a DT, a constrained DT also tends to avoid sliver triangles, except those formed by satisfying the edge constraints.

### 2.3.2 Polyline and Polyline Simplification

In geometry, a **polyline** is a connected series of line segments with no self-intersection. It is used to approximate a curve in many applications. An example of a polyline that consists of eight points is shown in Figure 2.8. Often a polyline has too many points for an application. That is, the points on a polyline are too close together. For the sake of efficiency, it is better to find a simplified polyline with fewer points to approximate the original polyline. To achieve this, a **polyline simplification** method, which reduces the number of points in a polyline to produce a simplified polyline that approximates the original within a specified tolerance, is needed.

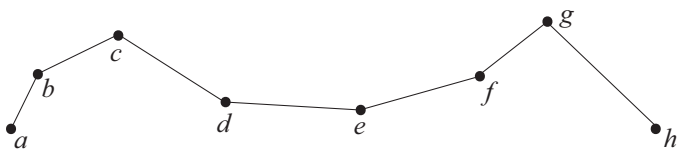



Figure 2.8: Example of a polyline

Many polyline simplification methods have been proposed to date. One of the classical algorithms is the Douglas-Peucker algorithm [43], which is extensively used in computer graphics and geographic information systems. Given a polyline, the Douglas-Peucker algorithm produces a simplified polyline consisting of a subset of the points that defined the original polyline, by discarding points based on a specified tolerance ε. The Douglas-Peucker algorithm automatically marks the first and the last points of the polyline to be kept. It starts with a line segment connecting the first and last points of the polyline, and then finds a point $p$ that is furthest from the line segment with a distance $d_l$. If $d_l$ is larger than ε, the point $p$ is marked to be kept and the algorithm continues. If $d_l$ is smaller than ε, the algorithm stops and all the unmarked points between the first and last points are discarded.

The process of Douglas-Peucker algorithm is illustrated in Figure 2.9, where the tolerance ε is specified in the top-left corner. The original polyline $abcdefgh$ is shown in Figure 2.8. In Figure 2.9(a), firstly, we mark the first point $a$ and the last point $h$ of the polyline to be kept. We then calculate the distance between each point from $b$ to $g$ and the line segment $\overline{ah}$, and find the point $g$ with the largest distance $d_l$. Since $d_l$ is larger than ε, $g$ is marked to be kept. Till now, three points, namely $a$, $g$ and $h$, are marked to be kept. In Figure 2.9(b), similarly, we find a point $c$ between $a$ and $g$ that is furthest from $\overline{ag}$ with a distance $d_l$ greater than ε, while no point is found between $g$ and $h$. Therefore, $c$ is marked

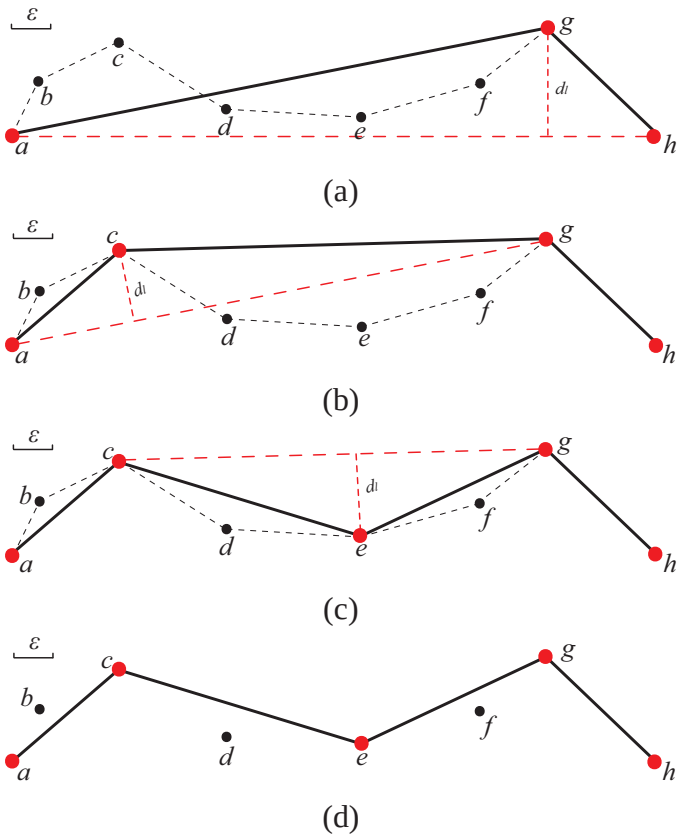



Figure 2.9: Douglas-Peucker polyline simplification algorithm. (a)-(c) the procedures of polyline simplification, and (d) the simplified polyline.

to be kept and the algorithm continues. In Figure 2.9(c), the point $e$ is found to be the furthest point from $\overline{cg}$ with a distance $d_l$ greater than ε. Thus, $e$ is marked to be kept. Finally, a simplified polyline consisting of all the points that have been marked as kept (i.e., $a$, $c$, $e$, $g$, and $h$), is generated as in Figure 2.9(d), . We can see from the figure that, the simplified polyline $acegh$ is a reasonably good approximation of the original one $abcdefgh$.

### 2.3.3 Computational Geometry Algorithms Library (CGAL)

Several libraries are utilized in the software developed for the research described herein. One such library is the Computational Geometry Algorithms Library (CGAL). CGAL is an open-source project that aims to provide easy access to efficient and reliable solutions and methods in computational geometry in the form of a C++ library. It provides many data structures and algorithms for geometric computation, such as convex hull, triangulations, DTs and constrained DTs. Besides those, interfaces to third party software, such as Qt,

Geomview, and the Boost Graph Library, are also provided. Furthermore, CGAL makes extensive use of templates, which facilitates the development of efficient and flexible code.

## 2.4 Image Processing

Before presenting our work, some image processing concepts and algorithms need to be introduced. These concepts and algorithms are employed in our proposed mesh-generation methods, and include binomial filters, bilinear interpolation, Canny edge detection, anti-aliasing, and super-sampling. In what follows, the details of these concepts and algorithms are presented.

### 2.4.1 Binomial Filter

Binomial filters are simple and efficient low-pass filters. They have reasonably-good frequency responses that form a compact approximation of the discrete Gaussian [44]. Compared to Gaussian filters [45], an appealing feature of binomial filters is that they do not require multiplications, which has potential benefits in terms of computational complexity. Therefore, binomial filters are widely used in many applications instead of Gaussian filters, especially in hardware implementations.

The transfer function $H_n$ of the $n$th order one dimensional (1-D) binomial filter $\mathbf{b}_n^{(1D)}$ with zero-phase and unity DC gain is

$$H_n(z) = z^{(n-1)/2}\left(\frac{1}{2}+\frac{1}{2}z^{-1}\right)^{n-1}, \tag{2.8}$$

where $n$ is odd. Alternatively, the coefficients of binomial filters $\mathbf{b}_n^{(1D)}$ correspond to the normalized $n^{th}$ row (starting from 0) of Pascal' s triangle. For example, the nonzero coefficients of the impulse response of $\mathbf{b}_2^{(1D)}$ are $\left[\frac{1}{4}\ \frac{1}{2}\ \frac{1}{4}\right]$.

A $n$th order two dimensional (2-D) binomial filter, denoted as $\mathbf{b}_n$, can be generated as the tensor product of 1-D filters $\mathbf{b}_n^{(1D)}$. For example, a second order 2-D binomial filter $\mathbf{b}_2$ is calculated by tensor product from $\mathbf{b}_2^{(1D)}$. The nonzero coefficients of the impulse response of the 2-D binomial filter $\mathbf{b}_2$ are $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$.

### 2.4.2 Bilinear Interpolation

In signal and image processing, **interpolation** is defined as a method of calculating the values of new data points which are in the range of a discrete set of known data points. Interpolation is extensively used in many applications, including image scaling and signal upsampling, as one of the basic resampling techniques.

In 1-D, one of the fundamental interpolation methods is linear interpolation. Though not precise, linear interpolation is quick and easy to implement, which makes it popular in applications. Linear interpolation takes two data points of a function $f$, $P_1 = x_1$ and $P_2 = x_2$ illustrated in Figure 2.10, to calculate the interpolant of $f$ at the point $Q = x$, which is given by

$$f(Q) = f(P_1) + (f(P_2) - f(P_1))\frac{x - x_1}{x_2 - x_1}. \tag{2.9}$$

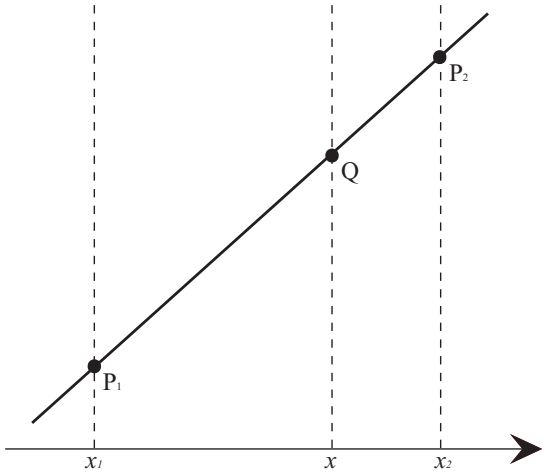



Figure 2.10: Linear interpolation.

Bilinear interpolation [46] is an extension of linear interpolation to two dimensions, which is extensively used to interpolate a function $f$ defined on $\mathbb{R}^2$ at a point inside a rectangular grid, as shown in Figure 2.11(a). The key idea of bilinear interpolation is to perform linear interpolation first in one direction, and then again in the other direction. To interpolate the function value at a point $p$, bilinear interpolation takes a weighted average of the function values of the 4 points, which are the closest $2 \times 2$ neighborhood surrounding $p$, to estimate the function value at $p$. The weight on each of the function values at the 4 points is based on the distance from each of the 4 points to $p$.

Suppose that we want to find the value of an unknown function $f$ at the point $Q = (x, y)$, as shown in Figure 2.11(a), and we know the value of $f$ at the four points $P_{11} = (x_1, y_1)$,

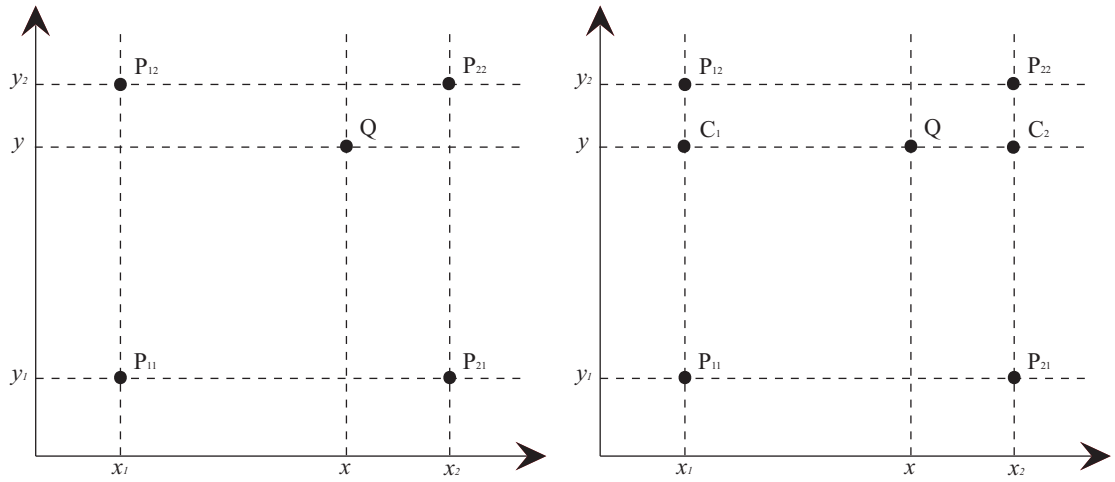



Figure 2.11: Bilinear interpolation. (a) We need to interpolate a 2-D function $f$ at an unknown point $Q = (x,y)$ within a rectangular grid. (b) We first perform linear interpolation in $y$ direction to calculate $f(C_1)$ and $f(C_2)$, then again in $x$ direction, to calculate f(Q).

$P_{12} = (x_1,y_2)$, $P_{21} = (x_2,y_1)$, and $P_{22} = (x_2,y_2)$. We first apply linear interpolation in y-direction, as shown in Figure 2.11(b), which calculates the estimated values of the function at two points, $C_1 = (x_1,y)$ and $C_2 = (x_2,y)$, as

$$f(C_1) = \frac{y_2 - y}{y_2 - y_1} f(P_{11}) + \frac{y - y_1}{y_2 - y_1} f(P_{12}),$$
$$f(C_2) = \frac{y_2 - y}{y_2 - y_1} f(P_{21}) + \frac{y - y_1}{y_2 - y_1} f(P_{22}).$$

With the approximated values of the points $C_1$ and $C_2$, we continue to do linear interpolation in x-direction to calculate the value of $f$ at point $Q = (x,y)$ as

$$f(Q) = \frac{x_2 - x}{x_2 - x_1} f(C_1) + \frac{x - x_1}{x_2 - x_1} f(C_2).$$

Thus, the desired approximation of $f$ at point $Q = (x,y)$ is obtained by

$$\begin{aligned} f(Q) = & \frac{f(P_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y) \\ & + \frac{f(P_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y) \\ & + \frac{f(P_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1) \\ & + \frac{f(P_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1). \end{aligned} \tag{2.10}$$

### 2.4.3 Canny Edge Detection

**Edges** are one of the most fundamental features of an image, and contain useful information for a number of applications. An edge is a collection of pixels which have a significant change in brightness compared with the pixels around them or, more formally, represent discontinuities. **Edge detection** is a process which aims at identifying edges in a digital image to capture important events and changes in properties [47] [48] [49]. Typically, the process of edge detection produces an **edge map**, which is a binary image containing only edge information. In the binary edge map, the points with a nonzero value are edge points, while those with a zero value are nonedge points. Edge detection is extensively used in many applications, including image processing [50], pattern recognition [51] and computer vision [52]. Figure 2.12 gives an example of edge detection. The original image is shown in Figure 2.12(a), and its corresponding edge map produced by edge detection is illustrated in Figure 2.12(b).

Suppose we have an image function $\phi$. Edges of $\phi$ can usually be determined from either, (1) the local maxima of $|\bigtriangledown\phi|$, or (2) the zero-crossings of $\bigtriangleup\phi$. Since derivative operations are sensitive to noise, a smoothing stage, typically binomial or Gaussian smoothing, is employed to reduce the noise. Often the edges in the edge map generated are very thick, as shown in Figure 2.13. Therefore, an edge thinning algorithm [53] is usually applied at the final stage to obtain a desirable edge map,

One of the most extensively used edge detectors, known as the Canny edge detector, was proposed by Canny in 1986 [49]. It is still a popular edge detector today. In many applications, it tends to perform quite well. Canny edge detection consists of four stages: (1) noise reduction, (2) gradient calculation, (3) local non-maxima suppression, and (4) hysteresis thresholding. In what follows, the detail of each stage is presented.

**NOISE REDUCTION.** Most types of real world imagery contain noise from a va-

(a) (b)

Figure 2.12: Example of edge detection. (a) The original image, and (b) the edge map produced by the edge detector.

riety of sources. Since the derivatives of images are sensitive to noise, we need to find a means to reduce the noise. A 2-D binomial filter $\mathbf{b}_n$ which approximates Gaussian filter is a good choice. Let $\phi$ denote the image function. The convolution of raw image function $\phi$ with a 2-D binomial filter $\mathbf{b}_n$, will generate a image function $f$ with reduced noise,

$$f = \phi * \mathbf{b}_n.$$

**GRADIENT CALCULATION.** After obtaining the image function $f$ with reduced noise, we need to compute the magnitude of its gradient. The first order partial derivatives of the image $g_x = \frac{\partial}{\partial x} f$ and $g_y = \frac{\partial}{\partial y} f$ can be obtained by the convolution of the edge detection operator (e.g., Roberts, Prewitt, Sobel) with $f$. Given such estimates of first order partial derivatives, the magnitude and direction of the gradient are computed respectively as

$$g = \sqrt{g_x^2 + g_y^2}, \text{ and} \tag{2.11}$$

$$\theta = \text{atan2}(g_y, g_x). \tag{2.12}$$

**NON-MAXIMA SUPPRESSION.** Given an estimate of the image gradient, for every pixel of the image, a test is carried out to determine whether the gradient magnitude at each pixel position assumes a local maximum in the gradient direction. That is, for every pixel $a$, linear interpolation is employed to interpolate the gradient magnitudes of the two points in the gradient direction of $a$ on rectangular grid. If the gradient magnitude of the

Figure 2.13: Thick edge example

pixel $a$ is greater than the gradient magnitudes of both of the interpolated points, the pixel $a$ is considered as a local maximum and marked as an edge point. This process is illustrated in Figure 2.14. As shown in the figure, to test the pixel $a$, we first find two points $p$ and $q$ which are the intersection of the line in the gradient direction of $a$ with the rectangular grid. The gradient magnitudes $g(p)$ and $g(q)$ are then interpolated based on the gradient magnitudes $g(a_{-1,-1})$, $g(a_{-1,0})$, $g(a_{1,0})$, and $g(a_{1,1})$. The pixel $a$ is marked as an edge point only when $g(a)$ is larger than both $g(p)$ and $g(q)$. At this stage, an edge map $\mathbf{M}_e$, which consists of a set of edge points, is obtained.

**HYSTERESIS THRESHOLDING.** In most cases, pixels with larger gradient magnitudes are more likely to correspond to edges. It is difficult, however, to determine whether a pixel with given gradient magnitude corresponds to an edge point or not with only one prespecified threshold. Therefore, the Canny edge detector uses hysteresis thresholding, which requires two thresholds, a high threshold $\tau_h$ and a low threshold $\tau_l$. We begin with generating an edge map $\mathbf{M}_h$ which contains the edge points we can be fairly certain are authentic. To generate $\mathbf{M}_h$, we first compare the gradient magnitude $g_p$ of each point $p$ in $\mathbf{M}_e$ to $\tau_h$. If $g_p$ is larger than $\tau_h$, we mark $p$ as an edge point and add it to $\mathbf{M}_h$. To further detect faint edges, we need to add true edge points to $\mathbf{M}_h$ from $\mathbf{M}_e$. We first find the starting points in $\mathbf{M}_e$, which are the points in $\mathbf{M}_e$ and adjacents to the points in $\mathbf{M}_h$ but not in $\mathbf{M}_h$. The edges are then traced through the edge map $\mathbf{M}_e$ from the starting points, and the points with gradient magnitude greater than $\tau_l$ are marked as edge points and added to $\mathbf{M}_h$.

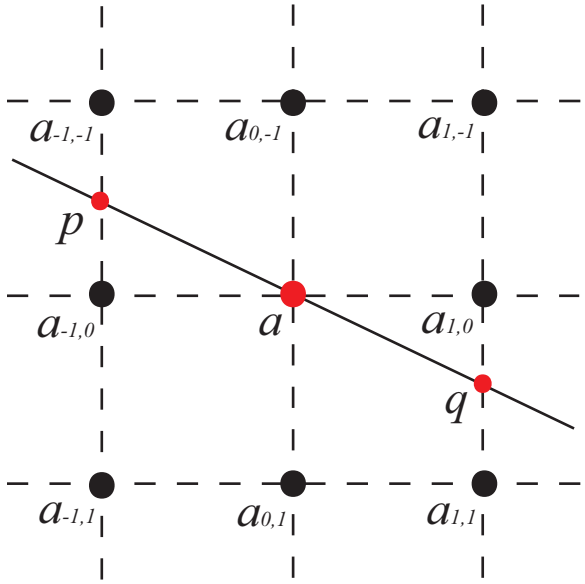



Figure 2.14: Local nonmaxima suppression. To test a point $a$, the gradient magnitudes of two points $p$, $q$ in the gradient direction of $a$ are interpolated.

To demonstrate the efficiency of hysteresis thresholding, we consider a modified version of Canny edge detection, which use only one threshold. The edge map produced with hysteresis thresholding is illustrated in Figure 2.15(a), while the edge map generated with only one threshold specified is shown in Figure 2.15(b). We can see from the figures that the edges obtained with hysteresis thresholding are much better than those obtained with only one threshold.

Once the above process is complete, a final edge map, which contains the information of the desired edges, is generated. Figure 2.12(b) mentioned earlier is an example of the edge map produced by Canny edge detector.

### 2.4.4 Modified Canny Edge Detector

The Canny edge detector is one of the most widely used edge detectors due to its characteristics of good detection, good localization, and single response. Some important edge points, however, especially those points at the crossing of two edges, are usually missing in the edge map produced by the Canny edge detector. For an input image shown in Figure 2.17(a), the edge map generated by Canny edge detector is illustrated in Figure 2.17(b). We can see that the edges obtained with the Canny edge detector are not **consistent**, that is, an edge point near the crossing of two edges is missing. By closely examining the gradient magnitude and direction at the missing edge point, we find that, although the gradient mag-

(a) (b)

Figure 2.15: Comparison between (a) edge map generated with hysteresis thresholding, and (b) edge map produced with only one threshold specified.

nitude at the missing edge-point is larger than those of a pair of points at opposing sides 8-connected to them, it is not the maximum in the gradient direction at the missing edge point. Near the crossing of two edges, intensity values of the points are changed in two different directions normal to the two edges. Thus the magnitudes of those points are affected by those two edges and may no longer be the maxima along the gradient directions. Therefore, the edge point near the crossing of two edges in Figure 2.17(b) is missing.

To avoid missing points at the crossings of the edges generated, the **modified Canny edge detector** was proposed [54]. Since the problem is that the gradient magnitudes at those missing edge points are maxima not in the gradient direction, the modified Canny edge detector employs a post-processing step to add those missing points to the final edge map. Let us call edges produced by the Canny edge detector the **main edges**. We will then mark a point as a part of the **secondary edges** if its gradient magnitude is larger than any pair of points 8-connected to it and at opposing sides but not necessarily in the gradient direction. As shown in Figure 2.16, suppose the gradient magnitude of image is denoted as $g$, the point $a$ is marked as a secondary edge point if when $g(a)$ is larger than both $g(a_{-1,-1})$ and $g(a_{1,1})$, or both $g(a_{-1,0})$ and $g(a_{1,0})$, or both $g(a_{-1,1})$ and $g(a_{1,-1})$, or both $g(a_{0,-1})$ and $g(a_{0,1})$.

According to the definitions, the main edges are a subset of the secondary edges. Recall that the main edges are obtained by the Canny edge detector, which are demonstrated to correspond to the true edges. On the other hand, some of the secondary edges correspond to the true edges, while others represent false edges. We need to separate the true secondary

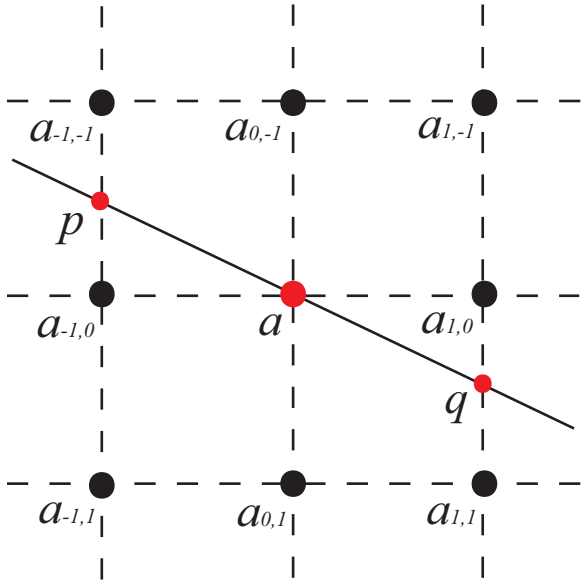



Figure 2.16: Local nonmaxima suppression of modified Canny edge detector. The point $a$ is marked as a secondary edge point when $g(a)$ is greater than any pair of points at opposing sides.

edges from the false ones. To achieve this, we first partition the secondary edges at the branch points, and remove the branches that do not contain any main edge points. We then trace through the secondary edge contours and only mark the points in the secondary edges that connect the main edges. Combining the main edges and the marked points in the secondary edges, we obtain the final edge map shown in Figure 2.17(c). We can see from the figure that, the edges generated by the modified Canny edge detector have no missing point at the crossing of two edges. Thus they are more consistent than those produced by the Canny edge detector.

### 2.4.5 Anti-aliasing and Super-sampling

In the context of computer graphics and rasterization algorithms, **aliasing** refers to the jagged and pixelated edges in a rendered image. In order to improve the quality of an image, we need to minimize aliasing (i.e., reduce the jagged edges) in the image.

The technique of minimizing aliasing is known as anti-aliasing. An example of the effect of anti-aliasing during rasterization is illustrated in Figure 2.18. Figure 2.18(a) and (b) show an image rendered without anti-aliasing and with anti-aliased, respectively. We can see from the example that, with anti-aliasing, the jagged edges in the image without anti-aliasing are eliminated, and the anti-aliased image looks much better than the one

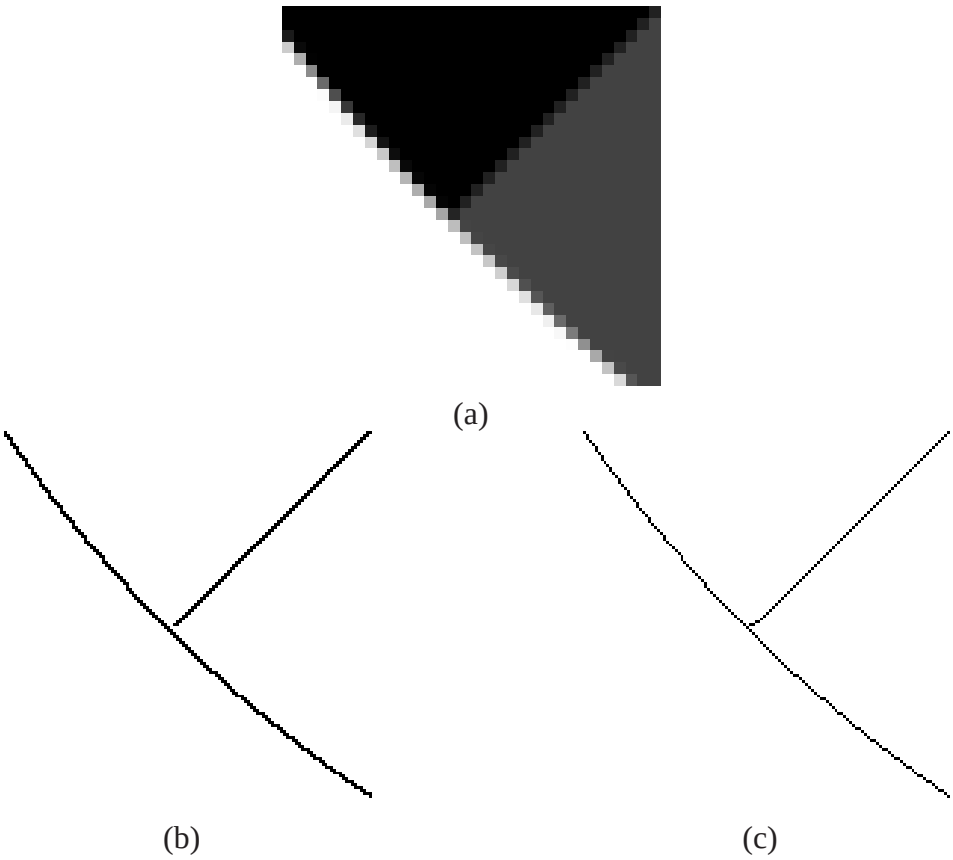
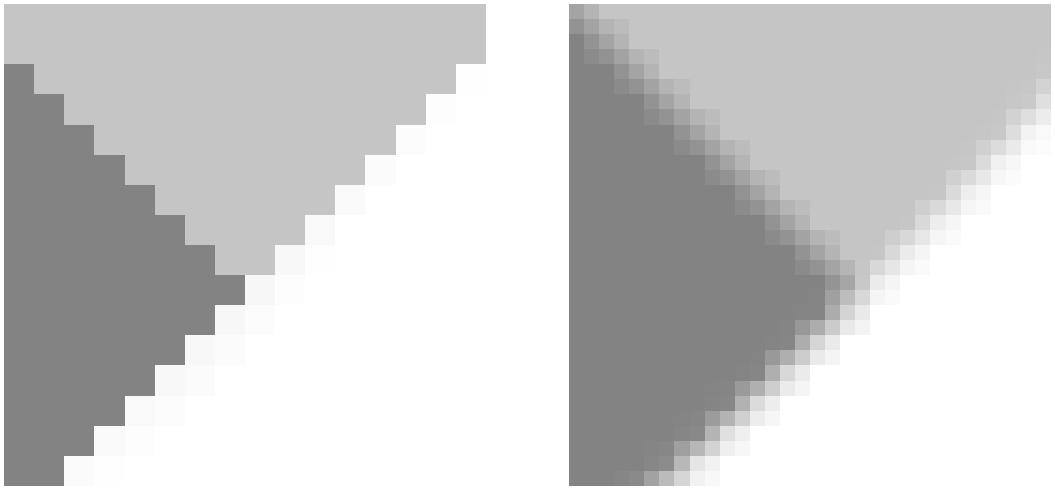



Figure 2.17: For a (a) given input image, the comparison of the effects of (b) the Canny edge detector and (c) the modified Canny edge detector.

produced without anti-aliasing.

One common anti-aliasing technique is called super-sampling [55]. In super-sampling, the image is first rendered at a much higher resolution than the final resolution desired. Extra samples are then taken inside a pixel (usually following some fixed pattern), and the average value of the samples inside the pixel is calculated. The image is downsampled to the desired size at the end, and the value of each pixel in the downsampled image is the average value of the samples inside the pixel of the original higher resolution image.

Many types of super-sampling have been proposed to date. One commonly used type is known as the grid algorithm. The grid algorithm is simple and easy to implement. In the grid algorithm, the pixel is split in several sub-pixels, and a sample is taken from the center of each. An example of $4 \times 4$ grid algorithm is shown in Figure 2.19. In the figure, the pixel is divided into a $4 \times 4$ array of sub-pixels, and 16 samples are selected from the center of the sub-pixels. The value of the pixel is computed as the average value of the sub-pixel



(a) (b)

Figure 2.18: Example of anti-aliasing. (a) the image with jagged edges, and (b) the image after anti-aliasing.

samples.

### 2.4.6 Triangle Scan Conversion

With a given triangle mesh model of an image function $\phi$ defined on $\mathbb{Z}^2$, in order to obtain an approximation $\hat{\phi}$ of $\phi$, we need to interpolate the value of all the points in $\hat{\phi}$. An efficient way of producing an image approximation $\hat{\phi}$ from a triangle mesh of the image $\phi$ is known as **triangle scan conversion**. Generally speaking, triangle scan conversion is a technique for converting geometric object defined on continuous domain to discrete lattice-sampled representation.

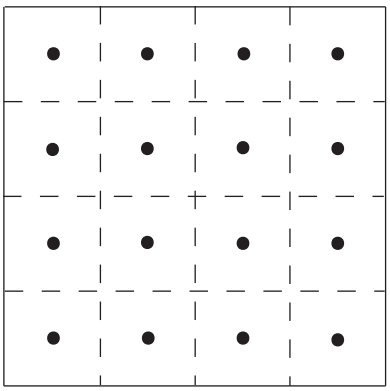


Figure 2.19: Example of super-sampling using $4 \times 4$ grid algorithm,

In triangle scan conversion, each triangle of the mesh is scanned sequentially, and the value of the points of $\hat{\phi}$ in each triangle is then uniquely determined by a planar interpolant that passes through the three vertices of the triangle. The process of scanning a triangle is shown in Figure 2.20. For each triangle in the triangulation, the interpolation is done on each point in the triangle sequentially, starting from top to bottom, left to right. As shown in Figure 2.20, for each triangle, a horizontal scan-line, which start from $y_{max}$, is used to do the interpolation. At each $y \in [y_{min}, y_{max}] \cap \mathbb{Z}$, the horizontal scan-line intersects the triangle edges at two points, $P_l(x_l, y)$ and $P_r(x_r, y)$, respectively. The interpolation is then carried out on the points $p(x_p, y) \in \mathbb{Z}^2$ located between the pair of intersections, $P_l$ and $P_r$, from left to right. The process of triangle scan conversion stops when the scan-line reaches $y_{min}$, which means all the points inside the triangle (including the points on triangle edges) have been interpolated. Due to the properties of the triangle, the points on the edges are interpolated twice, which increases the computational complexity of triangle scan conversion. One of the methods for solving this problem, which is presented in [56], is uniquely assign each point to a triangle. Once the process of triangle scan conversion is completed, the approximation $\hat{\phi}$ of the image function $\phi$ is generated.

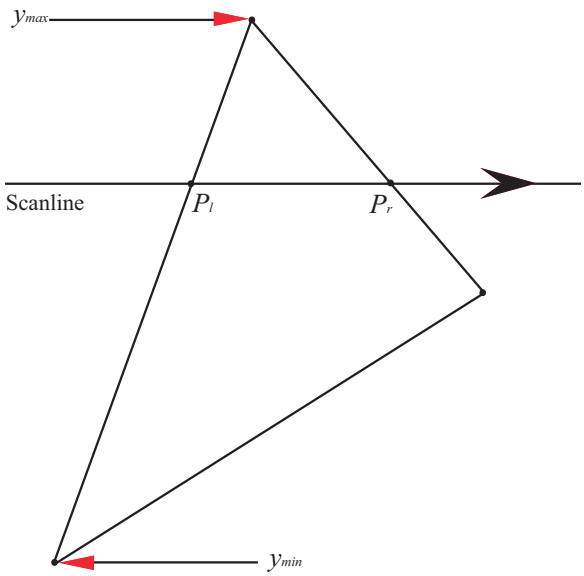
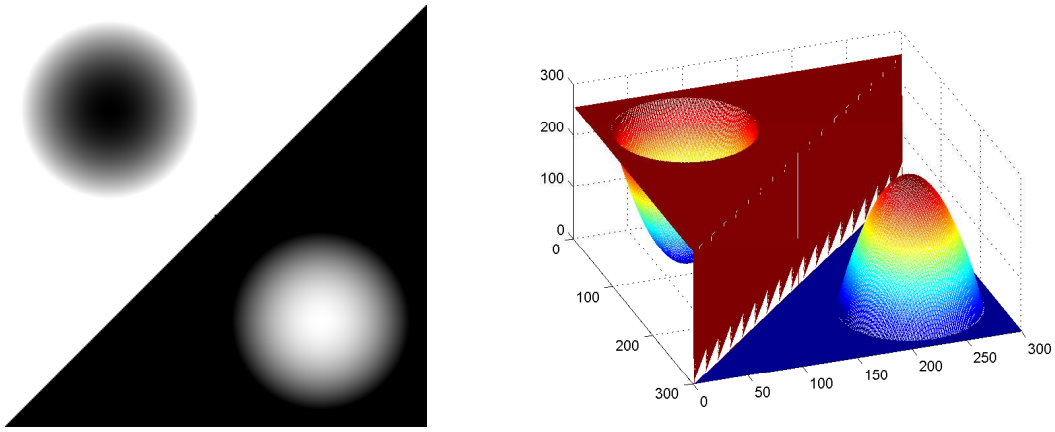



Figure 2.20: Triangle scan conversion.

## 2.5 Mesh Modelling

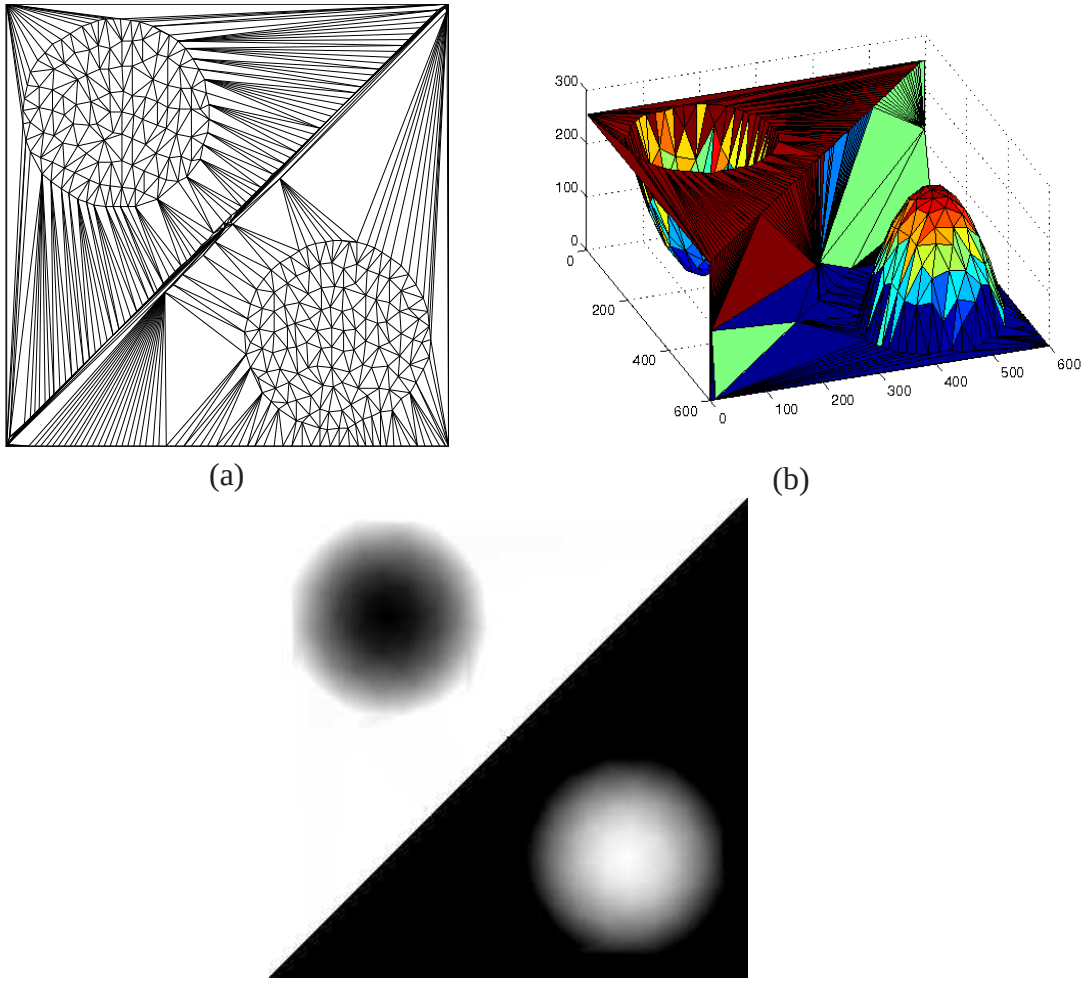
An image can be modeled as a 2-D function defined on a continuous domain (a subset of $\mathbb{R}^2$), where the value of the function corresponds to the brightness of the image. For example, the image in Figure 2.21(a) can be modeled as a surface as illustrated in Figure 2.21(b) by using the brightness of the image as the height of the surface above the plane. In image processing and computational geometry, mesh modelling is an approach for modelling objects by representing them using mesh elements (i.e., polygons). Particularly, mesh modelling of an image is an approach for approximating the image function $\phi$, which involves partitioning the image domain into a collection of non-overlapping mesh elements (e.g., triangles). In other words, a mesh model of an image can be considered as a geometric representation of the image using non-uniform samples, of which the image samples are adaptively placed according to the local content of the image. Among the numerous possibilities of polygon meshes, triangle meshes have received considerable attention. To form a triangle mesh model of an image, we first need to select a set of sample points from the image, and then construct a triangulation of the sample points.




(a) (b)

Figure 2.21: Image modeled as a function defined on continuous domain. (a) The original image, and (b) image modeled as surface.

A mesh approximation of the image in Figure 2.21(a) with a sampling density (which will be defined later) of 0.5% is shown in Figure 2.22. The mesh model used here is the most commonly used mesh model, which we refer to as the **basic model** herein. The basic model is quite simple and employs Delaunay triangulations. With the basic model, the sample points $P$ are chosen as a subset of $\Lambda$, and $P$ is triangulated using the Delaunay triangulation. Over each (triangle) face in the triangulation, $\hat{\phi}$ is defined as the unique




(a) (b)

(c)

Figure 2.22: Mesh approximation of Image (sampling density 0.5%). (a) The triangulation of the original image, (b) resulting triangle mesh, and (c) the reconstructed image.

linear function that interpolates $\phi$ at the three vertices of the face. Thus, the approximating function $\hat{\phi}$ is continuous and interpolates $\phi$ at each point in $P$. Furthermore, this model is completely characterized by the sample points $P$ and $\phi(p)$ for $p \in P$. Figure 2.22(a) shows the triangulation of the original image domain and the resulting triangle mesh is illustrated in Figure 2.22(b).

With a mesh model of an image, an approximation of the image can be reconstructed. Among the numerous image construction methods, triangle scan conversion, which is introduced earlier in Section 2.4.6, is an efficient and extensively used scheme. In triangle scan conversion, the value of the point $p \in \mathbb{Z}^2$ inside each triangle is determined by a unique planar interpolant that passes through the sampled function values at three vertices

of the triangle. With triangle scan conversion, the image approximation shown in Figure 2.22(c) is reconstructed from the mesh model illustrated in Figure 2.22(b). From the Figure 2.22(c), we can see that, although the sampling density is quite low (0.5%), the image approximation obtained is still of good quality.

# Chapter 3

# Proposed Mesh-Generation Methods

## 3.1 Overview

In this chapter, we begin with an introduction of two previously proposed mesh-generation methods that are employed in our work, namely the ED and GPI schemes. Then, we propose a mesh model that explicitly represents image discontinuities along with two mesh-generation approaches that select the model parameters for a given input image. After that, we discuss the selection of the parameters and variations of the proposed approaches that affect the quality of the results. Experimental results show that our proposed mesh-generation methods can produce image approximations of higher quality than those generated by the highly effective ED and GPI schemes in terms of both PSNR and subjective quality. Some of our results presented in this chapter have also been published in [35].

## 3.2 Previously Proposed Mesh Generation Methods

Before presenting our approaches, we first introduce two other extensively used mesh-generation methods, namely the **error diffusion** (ED) scheme and the **greedy point insertion** (GPI) scheme. The mesh model used in these two mesh-generation methods is the basic model described in Chapter 2, which is based on DT and can be uniquely characterized by a set of sample points along with the corresponding sample values. In what follows, the ED scheme and the GPI scheme are presented in detail.

### 3.2.1 ED Scheme

The ED scheme was proposed by Yang, Wernick, and Brankov [13] in 2003. It is fast and easy to implement. It uses Floyd-Steinberg error diffusion [24] to generate the set of desired sample points, hence the name. For a given image $\phi$ defined on the domain $\Lambda = [0, W-1] \times [0, H-1] \bigcap \mathbb{Z}^2$, to select a desired number $K$ of sample points, the ED scheme first calculates a sample-point density function $\sigma$ from the **maximum magnitude second-order directional-derivative (MMSODD)** of $\phi$ at each point $(x,y) \in \Lambda$, and then employs Floyd-Steinberg error-diffusion to select a set $S$ of $K$ sample points according to the density function $\sigma$. One desired property of the ED scheme is that, the local spatial density of the sample points is proportional to the MMSODD of $\phi$.

In more detail, the ED method consists of the following steps (in order):

1. We first compute $d$, which is the MMSODD of $\phi$, at each point $p = (x,y) \in \Lambda$ as

$$d(x,y) = \max\{|\lambda_1 + \lambda_2|, |\lambda_1 - \lambda_2|\}, \tag{3.1}$$

where $\lambda_1$ and $\lambda_2$ are computed by

$$\lambda_1 = \frac{1}{2}\left(\frac{\partial^2}{\partial x^2}\phi(x,y) + \frac{\partial^2}{\partial y^2}\phi(x,y)\right), \text{ and} \tag{3.2}$$

$$\lambda_2 = \sqrt{\frac{1}{4}\left(\frac{\partial^2}{\partial x^2}\phi(x,y) - \frac{\partial^2}{\partial y^2}\phi(x,y)\right)^2 + \left(\frac{\partial^2}{\partial x \partial y}\phi(x,y)\right)^2}. \tag{3.3}$$

The density function $\sigma$ is then computed as

$$\sigma(x,y) = \left[\frac{d(x,y)}{d_{max}}\right]^{\gamma}, \tag{3.4}$$

where $d_{max} = \max_{(x,y)\in\Lambda} d(x,y)$, and $\gamma$ is a positive real constant adjusting the sensitivity of the location of sample points to edges in the image, which we refer to as contrast sensitivity parameter herein.

2. In order to select approximately $|S| = K$ sample points, the threshold $\rho$ of Floyd-Steinberg error diffusion can be chosen as,

$$\rho = \frac{1}{2K}\left[\sum_{i=1}^{M}\sum_{j=1}^{N}\sigma(i,j)\right]. \tag{3.5}$$

A binary image $b$ defined on $\Lambda$ is generated from $\sigma$ using Floyd-Steinberg error diffusion with $\rho$ in serpentine scanning order [13]. Each point $(x,y) \in \Lambda$ satisfying $b(x,y) \neq 0$ is then placed in a set $S'$ of sample points.

3. If $|S'|$ is close enough to $|S|$ (i.e., $K$), set $S$ to $S'$; otherwise, the threshold $\rho$ need to be adjusted, and then go to step 2.

4. At last, a DT is constructed on the set $S$ of sample points to produce a mesh model.

Since the ED method has a number of degrees of freedom, we note that, in our work, we consider the variant of this method that employs a third-order binomial filter for noise removal during MMSODD estimation, a contrast sensitivity parameter $\gamma$ of 1, and the serpentine scan order for error diffusion. Also, the first- and second-order derivative operators used in the calculation of the MMSODD were approximated by filters with transfer functions $\frac{1}{2}z - \frac{1}{2}z^{-1}$ and $z - 2 + z^{-1}$, respectively. Lastly, we note that, during filtering, signal boundaries are handled by zero extension (i.e., padding with zeros).

### 3.2.2 Greedy Point Insertion Scheme

Garland and Heckbert proposed a mesh-generation scheme in 1995 [10] which is referred to as Garland-Heckbert(GH) method. The GH scheme refines an initial model successively until either a certain approximation error is reached or a certain number of sample points is obtained. The refinement of the GH scheme is done by inserting one point into a triangulation in each iteration based on the maximum absolute error. A modified version of the GH scheme was proposed by Adams [29], which employs the squared error of the faces instead of the maximum absolute error, and achieves a great improvement in the quality of the resulting image approximations. In the remainder of the paper, the modified version of the GH scheme proposed by Adams is called greedy point insertion (GPI) scheme.

Before presenting the GPI scheme in detail, we first introduce some concepts and notations used in the method. For a given image $\phi$ defined on $\Lambda$, in each iteration, the GPI scheme derives an approximation $\hat{\phi}$ from $\phi$. The absolute error $\varepsilon_{p(x,y)}$ of a point $p(x,y)$ is computed as

$$\varepsilon_p = \left|\phi(x,y) - \hat{\phi}(x,y)\right|, \tag{3.6}$$

and the sum of squared error $\varepsilon_{sum}$ of a face $f$ is the sum of the squared errors $\varepsilon$ of all the points in $f$,

$$\varepsilon_{sum} = \sum_{p \in f} \varepsilon_p^2. \tag{3.7}$$

In the triangulation of a set $P$ of points, let $F$ denote the set of all the faces in the triangulation, and each point $p \in \Lambda$ is assigned to exactly one face $f$ in $F$, denoted by $face(p) = f$. The **candidate point** of a triangle is defined as the point with the maximum absolute error $\varepsilon$ within the triangle, denoted by $cand(f)$.

The GPI scheme is involved in the following four steps.

1. Construct a DT with the set $P$ of four corner points on the convex hull of $\phi$.

2. Derive a mesh approximation from the DT using triangle scan conversion, and then calculate the absolute error $\varepsilon$ of each point. For each face $f$ of the DT, find its candidate point $cand(f)$.

3. For each face in the DT, calculate the sum of squared error $\varepsilon_{sum}$. Then find the face $f_m$ with maximum $\varepsilon_{sum}$.

4. Insert the candidate point of face $f_m$ into the DT. If the total number of vertices of the triangulation reaches the desired number $K$, the algorithm stops. Otherwise, go to step 2.

To implement the algorithm efficiently, a heap based priority queue is used to maintain the information of the faces and the candidate points. All the faces are stored in the priority queue keyed on their sum of squared errors $\varepsilon_{sum}$. During each iteration, we simply extract the candidate point of the face $f_m$ with maximum $\varepsilon_{sum}$, $cand(f_m)$, from the top of the priority queue and insert it in to the DT. One problem, which is computationally expensive, is the recalculation of the approximation errors after a candidate point is inserted in each iteration. Once a candidate point is inserted, the structure of the triangulation will be changed, consequently the approximation errors need to be recalculated. In fact, only a small number of triangles in the triangulation are affected by the insertion of a candidate point. Therefore, to solve this problem, we first find the set of triangles which are affected by the insertion of the candidate point, and then only recalculate the errors in those triangles.

## 3.3 A Mesh Model with Explicit Discontinuities

The basic mesh model used by the ED and GPI schemes (introduced earlier) is always associated with an approximating function that is continuous. Images, however, often contain a significant number of discontinuities (i.e., image edges). This observation motivated us to propose a new mesh model that explicitly represents discontinuities in images, known as

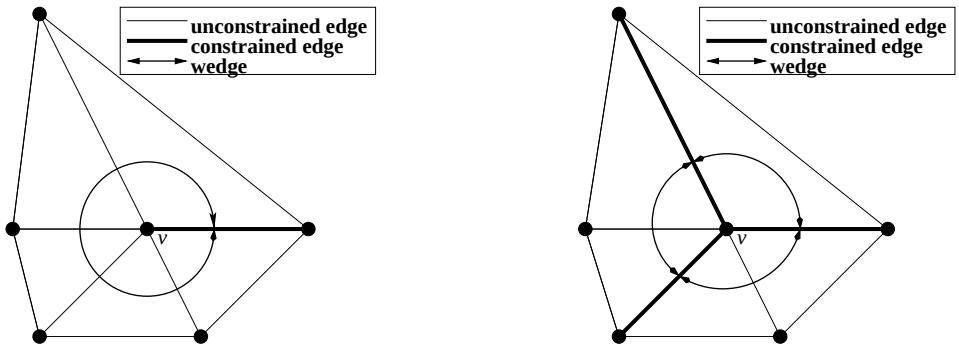
the **ERD model** (where ERD stands for "explicit representation of discontinuities"). Our ERD model makes use of constrained Delaunay triangulations [42]. Consider an image $\phi$ defined at the points $\Lambda = \{0,1,\ldots,W-1\} \times \{0,1,\ldots,H-1\}$ (i.e., an image sampled on a rectangular grid of width $W$ and height $H$). Let $\Gamma = [0,W-1] \times [0,H-1]$. A mesh model for $\phi$ is completely characterized by:

1. a set $P = \{p_i\}$ of sample points, where $p_i = (x_i, y_i) \in \frac{1}{2}\mathbb{Z}^2 \cap \Gamma$;
2. a set $E$ of constrained edges (i.e., a set of pairs of sample points from $P$); and
3. for each sample point $p_i$, one or more wedge values (where the term ẅedge value" will be defined precisely later).

The quantities $P$ and $E$ along with the associated wedge values are used to determine a function $\hat{\phi}$ defined on $\Gamma$, where $\hat{\phi}$ is an approximation of $\phi$. Note that, with our model, the sample points in $P$ are chosen on twice as fine a grid as the original image being represented (i.e., $\Gamma \cap \frac{1}{2}\mathbb{Z}^2$ as opposed to $\Gamma \cap \mathbb{Z}^2$). This is done in order to allow for more accurate representation of image edges. As we will see, $\hat{\phi}$ is chosen to interpolate $\phi$ at each point in $\mathbb{Z}^2 \cap P$. As a matter of terminology, we refer to the quantity $|P|/|\Lambda|$ as the **sampling density**. In what follows, we explain how $\hat{\phi}$ is defined in terms of $P$, $E$, and the wedge values.

First, we construct a constrained DT of $P$ with the constrained edges $E$, which serves to partition the image domain $\Gamma$ into triangle faces. The constrained edges are chosen to correspond to image edges. For each vertex $v \in P$, the set of faces incident on $v$ is partitioned into what are called wedges. In particular, a **wedge** is a set of consecutive faces in a loop around a vertex $v$ that are not separated by any constrained edge. This definition is illustrated in Figure 3.1. If the number of constrained edges incident on the vertex $v$ is zero or one, all faces incident on $v$ form a single wedge, as shown in Figure 3.1(a). Otherwise, if $n$ constrained edges are incident on $v$ (where $n \geq 2$), the faces incident on $v$ form $n$ wedges, as shown in Figure 3.1(b). Wedges are used to facilitate the modelling of discontinuities (i.e., image edges). Since constrained edges are chosen to correspond to image edges, a vertex $v \in P$ that has more than one wedge must be located along a discontinuity (i.e., image edge). Each wedge of a vertex has associated with it what is called a wedge value. The **wedge value** $z$ of the wedge $w$ belonging to vertex $v$ specifies the limit of $\hat{\phi}(p)$ as $p$ approaches $v$ from points inside the wedge $w$.

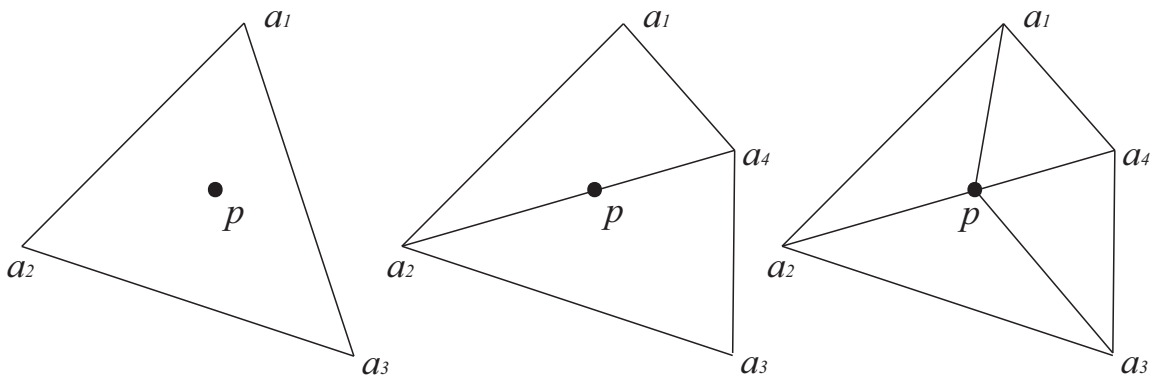
Now, we specify precisely how the function $\hat{\phi}$ is defined at each point $p \in \Gamma$. There are two cases to consider: 1) $p$ is not on a constrained edge; 2) $p$ is on a constrained edge.




(a) (b)

Figure 3.1: The relationship between vertices, constrained edges, and wedges. The (a) single-wedge, and (b) multiple-wedge cases.

*Case 1.* First, let us consider the case that $p$ is not on a constrained edge. Let $f$ denote a face of the triangulation with vertices $p_i = (x_i, y_i)$, $p_j = (x_j, y_j)$, and $p_k = (x_k, y_k)$ that contains the point $p$. Let $z_i$, $z_j$, and $z_k$ denote the wedge values for the face $f$ corresponding to the vertices $p_i$, $p_j$, and $p_k$, respectively. Then, $\hat{\phi}(p) = g(p)$, where the function $g$ is the unique planar interpolant that passes through the points $(x_i, y_i, z_i)$, $(x_j, y_j, z_j)$, and $(x_k, y_k, z_k)$. The three possible situations when $p$ is not on a constrained edge are shown in Figure 3.2.

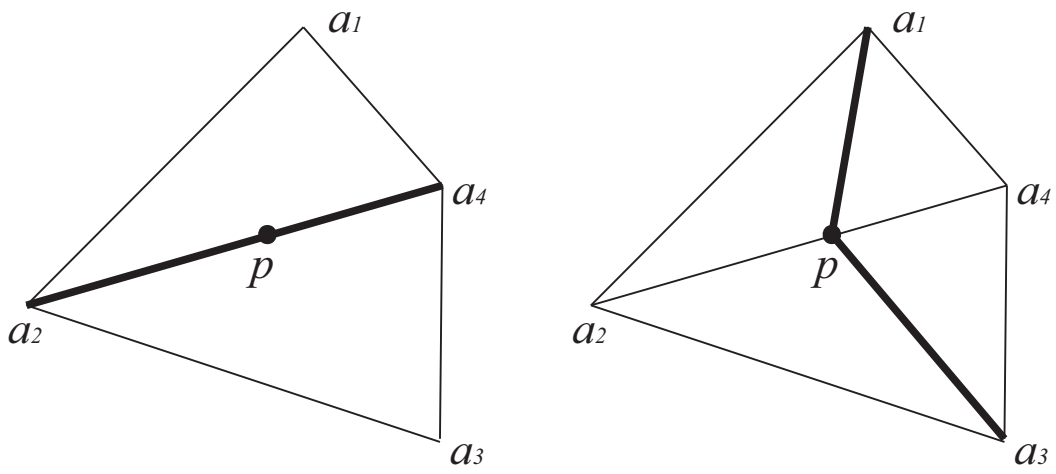



(a) (b) (c)

Figure 3.2: Three cases when the point $p$ is not on a constrained edge. (a) $p$ is inside a triangle, (b) $p$ is on a edge that is not a constrained edge, and (c) $p$ is a vertex of a triangle, but not incident to a constrained edge.

*Case 2.* Next, let us consider the case that $p$ is on a constrained edge. If $p$ is not an endpoint of a constrained edge, $\hat{\phi}(p)$ is the average of the values on the two sides of the image discontinuity (computed as in case 1). On the other hand, if $p$ is an endpoint of a constrained edge (i.e., a vertex in the triangulation), $\hat{\phi}(p)$ is the average of all wedge

values for (the vertex) $p$. The two possible situations when $p$ is on a constrained edge are illustrated in Figure 3.3.




(a) (b)

Figure 3.3: Two cases when the point $p$ is on a constrained edge. (a) $p$ is not an end point of a constrained edge, and (b) $p$ is an end point of a constrained edge.

From the mesh model $\hat{\phi}$, a lattice-sampled image can be reconstructed by straightforward rasterization algorithms. Due to the fact that our ERD model explicitly represents discontinuities, image edges could produce undesirable aliasing effects if the samples of the (discrete) image reconstruction were generated by simply evaluating $\hat{\phi}$ at points in $\Lambda$. Consequently, in the case of our ERD model, rasterization is performed using the well-known $4 \times 4$ supersampling technique [55], as this approach avoids such aliasing effects.

## 3.4 Proposed Mesh-Generation Methods

Having introduced our ERD mesh model, we now propose two mesh-generation methods, called ERDED and ERDGPI, to be used in conjunction with this model. For a given image $\phi$ sampled at the points of the rectangular grid $\Lambda$, each of these methods selects the parameters of the model (i.e., $P$, $E$, and wedge values) so as to obtain the best possible approximation of $\phi$ for a specified target number $N$ of sample points (i.e., $|P| = N$). Since the ERDED and ERDGPI methods fit into the same general algorithmic framework, we first introduce this framework. Then, we give the specifics of each of these methods. The algorithmic framework employed by both methods consists of the following steps:

1. **INITIAL TRIANGULATION.** Select initial values for $P$ and $E$. This determines the initial triangulation (i.e., the constrained Delaunay triangulation of $P$ with edge constraints $E$). Let $N_0 = |P|$ (i.e., $N_0$ is the initial mesh size).

2. **INITIAL WEDGE VALUES.** For each vertex $v \in P$, calculate the wedge value for each wedge of $v$.

3. **POINT SELECTION.** Select a new sample point $p^*$ to add to the mesh.

4. **POINT INSERTION.** Insert the point $p^*$ in the (constrained Delaunay) triangulation. If $p^*$ is on a constrained edge, split the edge at $p^*$ into two constrained edges, and compute the wedge value for each wedge of the vertex $p^*$. If $p^*$ is not on a constrained edge, the wedges remain the same and no wedge values need to be recomputed.

5. **STOPPING CRITERION.** If $|P| < N$, go to step 3 (i.e., add another sample point to the mesh).

In the framework above, steps 1 and 2 choose an initial coarse mesh of size $N_0$, and steps 3 to 5 iteratively refine the mesh by adding $N - N_0$ sample points to the mesh. The initial coarse mesh selection (i.e., steps 1 and 2) is performed in an identical manner for both the ERDED and ERDGPI methods. The two methods differ only in the approach used to refine the mesh (i.e., steps 3 to 5). First, we describe in more detail steps 1 and 2, which are identical for both the ERDED and ERDGPI methods.

### 3.4.1 Initial Coarse Mesh Selection

Recall that, with our ERD model, the sample points $P$ are chosen as a subset of $\Gamma \cap \frac{1}{2}\mathbb{Z}^2$. That is, the grid on which the sample points lie is a grid with its spacing in the horizontal and vertical directions each reduced by half relative to the grid $\Lambda$ on which $\phi$ is originally sampled. Since the original image $\phi$ is sampled on a $W$ by $H$ grid, this implies that the sample points (in $P$) are chosen to fall on a $(2W-1)$ by $(2H-1)$ grid. The relationship between these two grids is illustrated in Figure 3.4 for the case of a $4 \times 4$ image (i.e., $W = H = 4$). In what follows, let $\overline{\phi}$ denote the function defined on $\Gamma$ formed by the bilinear interpolation [46] of $\phi$. By definition, $\overline{\phi}$ satisfies $\overline{\phi}(p) = \phi(p)$ for all $p \in \Lambda$.

**STEP 1.** In step 1 of our framework, the selection of the initial triangulation consists of four substeps, which are numbered 1.1 to 1.4 below. These substeps are described in

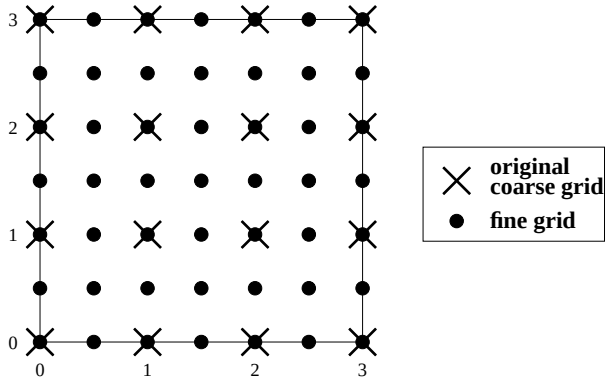



Figure 3.4: Relationship between grids for a $4 \times 4$ image.

detail in the paragraphs that follow and are also illustrated by way of the example shown in Figure 3.5.

**1.1) LOCATE EDGES.** First, we employ the (modified) Canny edge detector in [54] to locate edges in the image $\phi$ with half-pixel resolution. To accomplish this, we apply the edge detector to $\bar{\phi}$ sampled on the rectangular grid $\Gamma \cap \frac{1}{2}\mathbb{Z}^2$ to produce a binary edge map of dimensions $(2W-1) \times (2H-1)$. (An entry in the edge map is one if it corresponds to an edge pixel, and zero otherwise.) Note that the grid on which $\bar{\phi}$ is sampled here is twice as fine (in each dimension) as the grid $\Lambda$ on which the original image $\phi$ is sampled. By applying the edge detector to this higher resolution version of the original image, we can locate edges with half-pixel accuracy. The Canny edge detector works by computing the gradient magnitude and direction and then using this information in conjunction with hysteresis thresholding to select edge pixels. Two parameters must be specified as input to the edge detector, namely, the low and high thresholds for hysteresis thresholding, denoted herein as $\tau_{\mathsf{low}}$ and $\tau_{\mathsf{high}}$, respectively. In our method, these edge-detector thresholds are controlled by the parameters $\beta$ and $r$. The quantity $\tau_{\mathsf{high}}$ is chosen such that the fraction of pixels (from $\bar{\phi}$) whose corresponding gradient magnitude is greater than or equal to $\tau_{\mathsf{high}}$ is $\beta$ (i.e., the edge detector will nominally produce at least $\beta \left|\Gamma \cap \frac{1}{2}\mathbb{Z}^2\right|$ edge pixels). Then, $\tau_{\mathsf{low}}$ is selected as $\tau_{\mathsf{low}} = r\tau_{\mathsf{high}}$. To reduce the effects of noise, a smoothing operation (i.e., lowpass filter) is included in the convolution kernel used for gradient calculation. That is, the filter used to estimate each partial derivative is the composition of a first-order derivative operator and smoothing operator, where the first-order derivative operator is a filter with transfer function $\frac{1}{2}z - \frac{1}{2}z^{-1}$ and the smoothing operator is a fifth-order binomial filter [44]. Since edges in the edge map can be more than one pixel wide, we apply the line thinning algorithm from [53] to reduce the thickness of edges. The edge detection process is illustrated in Figure 3.5. In particular, given the input image in Figure 3.5(a), edge

detection would produce a binary image resembling that shown in Figure 3.5(b), where edge pixels are shown in black.

**1.2) CONSTRUCT POLYLINES FOR EDGES.** Having generated the edge map, we next construct a polyline representation of each edge in the edge map. To accomplish this, 8-connected edge pixels in the edge map are joined (by line segments) to form polylines. In cases where a polyline has one or more self-intersections (excluding loops), the polyline is split at each intersection point. In this way, the final set of polylines is guaranteed not to have any self-intersections (excluding loops). This process is illustrated in Figure 3.5. Given the edge map shown in Figure 3.5(b), we would produce a set of polylines like that shown in Figure 3.5(c).

**1.3) SIMPLIFY POLYLINES.** After polylines have been constructed to represent image edges, we need to simplify these polylines. In other words, for each polyline we find a new polyline with fewer vertices (i.e., control points) that well approximates the original polyline. To perform polyline simplification, we employ the well known **Douglas-Peucker (DP)** [43] algorithm. For a given polyline, the DP scheme repeatedly (using a greedy approach) adds points to a trivial two-point approximation of the polyline until the resulting approximation error is less than a prespecified tolerance. In our method, each polyline is simplified using the DP algorithm with tolerance $\varepsilon$. Then, we discard any polylines with fewer than $\ell$ vertices, where $\ell$ is a parameter of our method. Polylines with only a few points are eliminated, as such polylines tend to be associated with false edges introduced by noise and degrade mesh quality. This process is illustrated in Figure 3.5. Given the set of polylines shown in Figure 3.5(c), we would produce a set of simplified polylines like that shown in Figure 3.5(d).

**1.4) SELECT $P$ AND $E$ FROM POLYLINES.** Having obtained the set of simplified polylines, we now use those polylines in order to select $P$ and $E$. Since the extreme convex-hull points of $\Lambda$ (i.e., the four corner points of image bounding box) must be included in $P$, these four points are always forced to be included in $P$. We choose $P$ as the union of all of the polyline vertices and select $E$ as the set of line-segments from all of the polylines. Then, we form the constrained Delaunay triangulation of $P$ with edge constraints $E$. This process is illustrated in Figure 3.5. Given the simplified polylines shown in Figure 3.5(d), we would produce the triangulation shown in Figure 3.5(e), where constrained edges are denoted by thick lines.

**STEP 2.** Having selected the initial triangulation, we now need to choose the wedge values. In particular, for each wedge $w$ of each vertex $v \in P$, we must select the corresponding wedge value $z$. The selection of $z$ is performed in one of two ways, depending

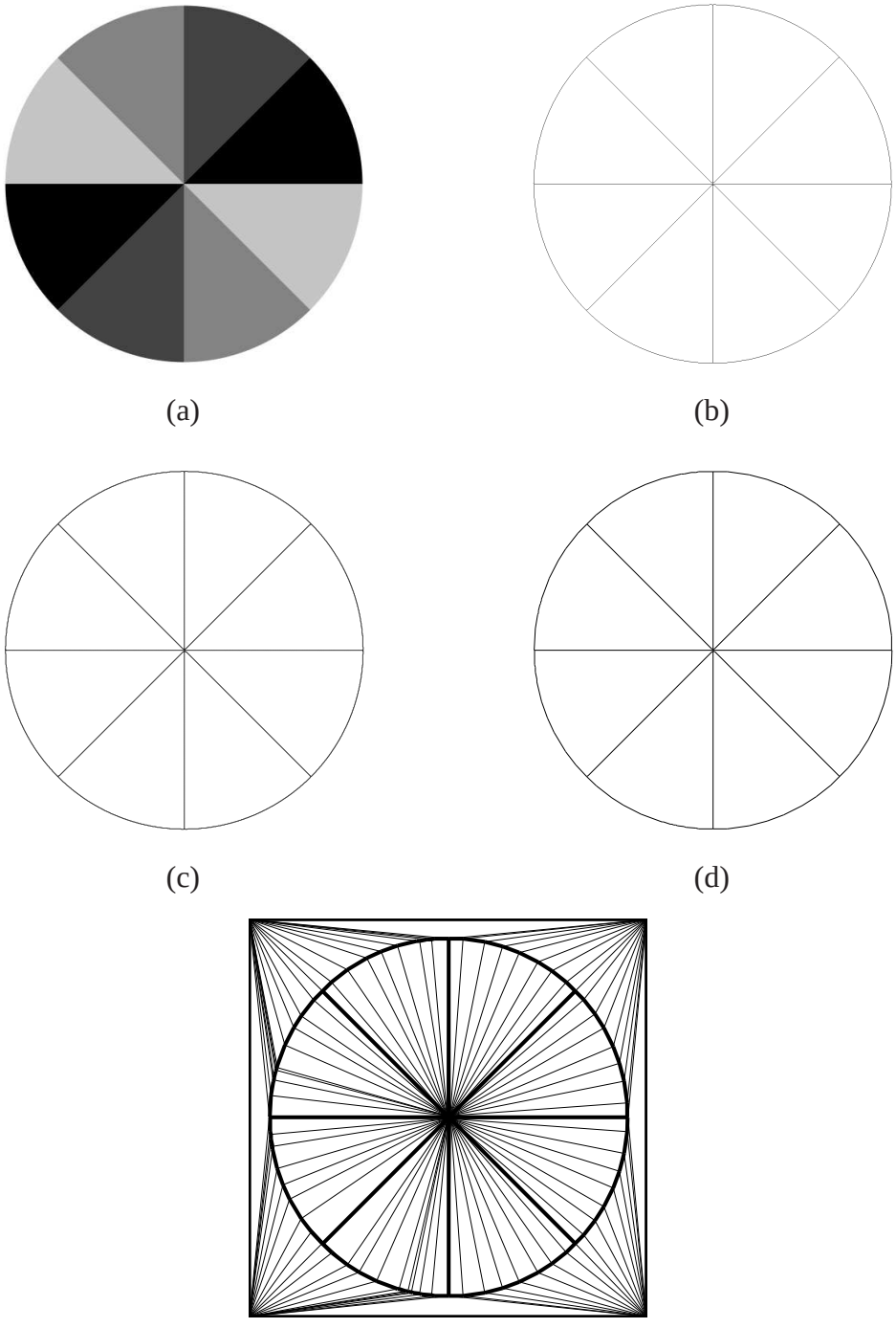


Figure 3.5: Selection of the initial triangulation. (a) Original image. (b) Binary edge map. (c) Unsimplified polylines representing image edges. (d) Simplified polylines representing image edges. (e) Initial triangulation (with constrained edges denoted by thick lines).

Table 3.1: Choice of $\beta$ for the ERDED method

| Samp. Density (%) | $\beta$ |
|---|---|
| $[0.00, 0.20)$ | 0.0095 |
| $[0.20, 0.35)$ | 0.0095 |
| $[0.35, 0.75)$ | 0.0118 |
| $[0.75, 1.20)$ | 0.0280 |
| $[1.20, 2.50)$ | 0.0550 |
| $[2.50, 3.50)$ | 0.0830 |
| $[3.50, 5.00)$ | 0.0950 |

Table 3.2: Choice of $\beta$ for the ERDGPI method

| Samp. Density (%) | $\beta$ |
|---|---|
| $[0.00, 0.20)$ | 0.0110 |
| $[0.20, 0.35)$ | 0.0115 |
| $[0.35, 0.75)$ | 0.0118 |
| $[0.75, 1.20)$ | 0.0280 |
| $[1.20, 2.50)$ | 0.0550 |
| $[2.50, 3.50)$ | 0.0830 |
| $[3.50, 5.00)$ | 0.0950 |

on the number $n$ of wedges associated with the vertex $v$. If $n = 1$, we simply choose $z$ as $z = \overline{\phi}(v)$. Otherwise (i.e., if $n \geq 2$), we proceed as follows. Let $f$ denote the MMSODD of $\phi$ (which is calculated according to equation (6) in [13]) and let $d$ denote a unit vector in the direction of the ray originating from $v$ and bisecting the wedge $w$. We select $z$ as $z = \overline{\phi}(v')$, where $v' = v + \alpha^* d$ and $\alpha^* = \arg\max_{\alpha \in [1,1.5]} f(v + \alpha d)$. In other words, $v'$ is chosen as the result of a local line search that maximizes the MMSODD along the ray bisecting $w$. The line search is restricted to $\alpha \in [1, 1.5]$ in order to prevent $v'$ from falling far outside of the (triangle) face with which $w$ is associated. As a practical matter, $f$ (as defined above) is calculated with a fifth-order binomial filter for smoothing.

**SELECTION OF $\beta$, $r$, $\ell$, AND $\varepsilon$.** As seen above, the selection of the initial triangulation (in step 1 of our framework) requires the specification of the parameters $\beta$, $r$, $\ell$, and $\varepsilon$. Rather than requiring these parameters be chosen manually, we propose an automated scheme for their selection, which was developed based on extensive experimentation. In our framework, we always choose $r = 0.4$. The remaining parameters $\beta$, $\ell$, and $\varepsilon$ are chosen as described below. In what follows, let $N$ and $D$ denote the number of sample points (i.e., $N = |P|$) and the sampling density of the mesh model, respectively.

As a matter of terminology, we refer to an image as **simple** if it contains an abnormally low amount of edges. How we select $\beta$, $\ell$, and $\varepsilon$ depends on whether the image is simple. First, we make a determination of whether the image is simple. To do this, we perform step 1 of our framework with the fixed choices of $\beta = 0.055$, $\ell = 5$, and $\varepsilon = 1$. If this results in an initial triangulation where the number of vertices that are endpoints of constrained edges is less than $0.001 \left|\Gamma \cap \frac{1}{2}\mathbb{Z}^2\right|$, the image is deemed to be simple; otherwise, it is deemed not to be simple.

Next, we make an initial choice for $\beta$, $\ell$, and $\varepsilon$. If the image is not simple, we proceed

as follows. Set $\ell = 5$. If $D > 0.01$, set $\varepsilon = 1$; otherwise, set $\varepsilon = 2$. Finally, make the initial choice of $\beta$ based on the sampling density as given by Tables 3.1 and 3.2 for the ERDED and ERDGPI methods, respectively. If the image is simple, we make the initial choice of $\beta$, $\ell$, and $\varepsilon$ as $\beta = 0.03$, $\ell = 1$, and $\varepsilon = 1$.

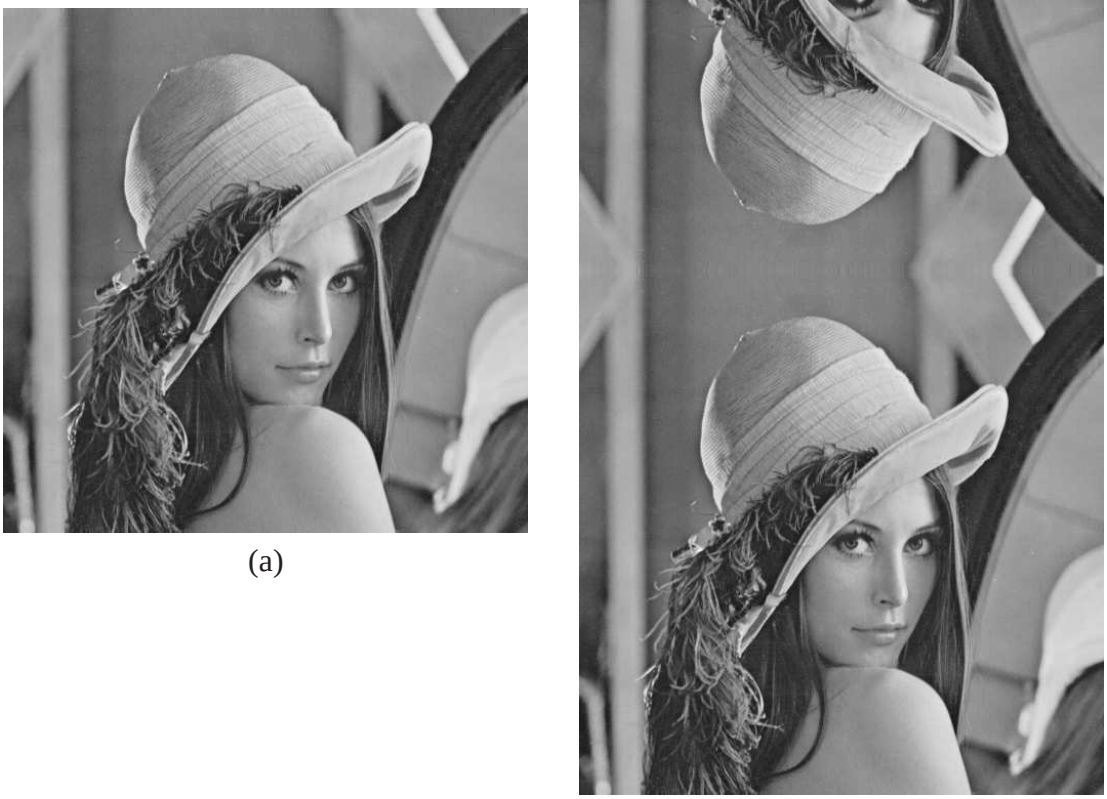
Next, we iteratively update $\beta$, $\ell$, and $\varepsilon$. This is accomplished by the following steps: 1) Perform edge detection (i.e., step 1.1 which uses $\beta$ and $r$). 2) Perform polyline simplification (i.e., steps 1.2 and 1.3 which uses $\varepsilon$ and $\ell$). If the number of sample points on constrained edges is less than $0.35N$, (i.e., too few sample points are obtained), set $\varepsilon = 1$ and redo polyline simplification. 3) If the actual number of constrained sample points is greater than $0.7N$, set $\beta := 0.75\beta$, set $\ell := 2\ell$, and go to step 1 (i.e., the start of the loop). 4) Output the current values of $\beta$, $r$, $\ell$, and $\varepsilon$ as the final selected values.

### 3.4.2 Mesh Refinement for the ERDED Method

As mentioned above, mesh refinement is performed differently in our ERDED and ERDGPI methods. In particular, in step 3 of our framework, the strategy used to select the new point $p^*$ to add to the mesh is different in these two cases. In what follows, we describe how step 3 is performed in the ERDED case.

Let $S$ denote the set of sample points to be added to the mesh. Since the initial mesh has size $N_0$, we require that $|S| = N - N_0$. With the ERDED method, $S$ is selected all at once. So, if step 3 is being encountered for the first time, $S$ is chosen (in its entirety) before any other processing is performed. Then (with $S$ having been initialized), we arbitrarily assign a point from $S$ to $p^*$ and then let $S := S \setminus \{p^*\}$. As for how $S$ is initially chosen, we will describe this shortly. Theoretically, it is possible for one or more of the points in $S$ to fall on a constrained edge. To avoid unnecessarily complicating our ERDED method, we discard any such points. Since it is extremely rare for this situation to arise, the impact on the target sampling density is negligible.

The set $S$ is chosen using the error-diffusion technique from the ED method [13] as described earlier. In order to permit the error-diffusion technique to work more effectively with our ERDED method, we made several modifications to this technique. First, the edge sensitivity parameter $\gamma$ was chosen as $\gamma = 0.5$ and the smoothing operator employed (for MMSODD calculation) was selected as a fifth-order binomial filter. Second, the density function $d$ used for error diffusion was modified. Instead of simply choosing $d$ as the MMSODD, $d$ was chosen such that it equals the MMSODD at points where the corresponding edge map entry (obtained from edge location in step 1.1 of our framework) is zero, and



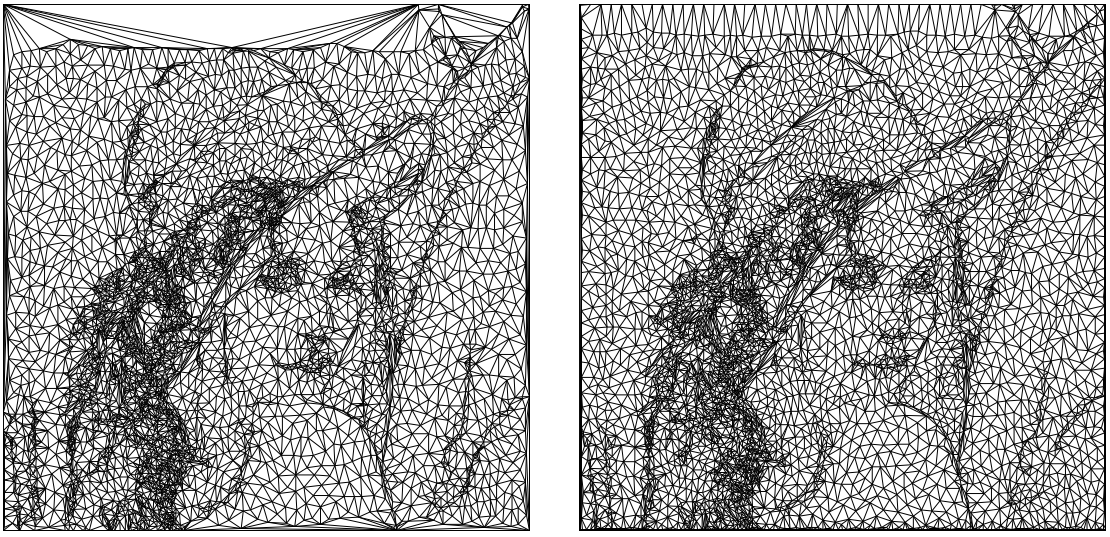
(a)

(b)

Figure 3.6: Mirroring the image. (a) original image and (b) image obtained after mirroring.

zero otherwise.

The third modification to the error diffusion scheme serves to eliminate an undesirable startup effect. In particular, when the number of sample points to be chosen is sufficiently low (i.e., at low sampling densities), error diffusion will often result in an abnormally low number of sample points being selected in the region of the image processed first (namely, the top of the image). This abnormally low number of sample points leads to very high distortion in this region, degrading overall performance. To eliminate this startup effect, we extend the image to be processed by mirroring it about its first row so as to obtain an image of twice the original height. That is, for the original image shown in Figure 3.6(a), we extend it by mirroring and obtain the resulting image shown in Figure 3.6(b). Then, we



(a) (b)

Figure 3.7: Startup effect in error diffusion. Triangulation obtained (a) without mirroring and (b) with mirroring.

apply error diffusion to extended image, discarding any sample points that are chosen in the mirrored region. To demonstrate the benefit of this mirroring process, we provide an example in Figure 3.7. In particular, Figures 3.7(a) and (b) show the triangulation obtained without and with the use of mirroring. Observe that in the no-mirroring case, an abnormally low number of sample points is selected in the first few rows (i.e., top) of the image, which ultimately leads to higher approximation error. In contrast, the mirroring case does not suffer from this problem.

### 3.4.3 Mesh Refinement for the ERDGPI Method

Now, we describe how step 3 of our proposed framework is performed in the ERDGPI case. In the ERDGPI case, a new point $p^*$ to add to the mesh is selected using the process described in what follows. During mesh refinement (i.e., steps 3 to 5 of our framework), we maintain the image approximation $\hat{\phi}$ generated from the current mesh model. This image approximation $\hat{\phi}$ is generated from the mesh model parameters as specified in Section 3.3. Each time a new point is added to the mesh, the image approximation $\hat{\phi}$ is updated to reflect the change in the mesh. For a face $f$ in the triangulation, let points$(f)$ denote all points in $\Lambda$ belonging to $f$. Using the current image approximation $\hat{\phi}$, we choose $p^*$ in two steps.

Table 3.3: Test images

| Image | Size, Bits/Sample | Description |
|---|---|---|
| `bull` | 1024×768, 8 | computer-generated bull [60] |
| `ct` | 512×512, 12 | CT scan of head [57] |
| `glasses2` | 1024×768, 8 | raytraced glasses [60] |
| `lena` | 512×512, 8 | woman [59] |
| `mri` | 256×256, 11 | MRI scan of head [57] |
| `peppers` | 512×512, 8 | collection of peppers [59] |

First, we select the face $f^*$ with the greatest squared error. That is,

$$f^* = \arg\max_{f \in F} \sum_{p \in \text{points}(f)} (\hat{\phi}(p) - \phi(p))^2,$$

where $F$ is the set of all faces in the triangulation. Then, we select the point $p^*$ in $f^*$ with the greatest absolute error. That is,

$$p^* = \arg\max_{p \in \text{points}(f^*)} \left|\hat{\phi}(p) - \phi(p)\right|.$$

## 3.5 Evaluation of Proposed Methods

Before proceeding further, a brief digression is in order concerning the test data used herein. In our work, we employed 41 (grayscale) images that were taken mostly from standard test sets, such as the JPEG-2000 test set [57], Kodak test set [58], and USC image database [59]. Herein, we focus our attention on the representative subset of six images listed in Table 3.3, which were deliberately chosen to include computer-generated, medical, and photographic imagery.

**ERROR DIFFUSION STARTUP IN ERDED METHOD.** Earlier, in the context of our ERDED method, we mentioned that error diffusion can often exhibit an undesirable startup behavior and to combat this problem we introduced a mirroring scheme. Now, we present some results to demonstrate the effectiveness of this mirroring scheme. For several test images and sampling densities, we generated a mesh using ERDED with and without mirroring and measured the resulting approximation error in terms of **peak signal-to-noise ratio (PSNR)**. The results obtained are shown in Table 3.4, with the best result in each case being highlighted in **bold**. Clearly, the mirroring scheme employed in our ERDED method is highly effective, outperforming the approach without mirroring in 22/24 of the test cases by a margin of up to 4.31 dB. It was due to this excellent performance of mirror-

Table 3.4: Effectiveness of the strategy for mitigating the startup effect in error diffusion

| Image | Samp. Density (%) | PSNR (dB) | |
|---|---|---|---|
| | | No Mirroring | Mirroring |
| bull | 0.125 | 21.97 | **24.68** |
| | 0.250 | 28.28 | **28.86** |
| | 0.500 | 34.69 | **35.15** |
| | 1.000 | **39.22** | 39.02 |
| ct | 0.125 | **15.83** | 15.63 |
| | 0.250 | 24.09 | **25.97** |
| | 0.500 | 25.75 | **30.06** |
| | 1.000 | 35.55 | **36.51** |
| glasses2 | 0.500 | 18.19 | **20.54** |
| | 1.000 | 23.21 | **24.80** |
| | 2.000 | 26.80 | **28.71** |
| | 3.000 | 30.06 | **30.85** |
| lena | 0.500 | 20.19 | **20.56** |
| | 1.000 | 24.65 | **25.82** |
| | 2.000 | 28.09 | **29.28** |
| | 3.000 | 30.64 | **31.31** |
| mri | 0.250 | 11.56 | **12.55** |
| | 0.500 | 22.46 | **24.79** |
| | 1.000 | 28.69 | **30.33** |
| | 2.000 | 32.13 | **33.14** |
| peppers | 0.500 | 21.37 | **22.14** |
| | 1.000 | 24.75 | **25.97** |
| | 2.000 | 28.56 | **29.00** |
| | 3.000 | 29.91 | **30.18** |

ing that we chose to include it in our ERDED method (as introduced earlier).

**COMPARISON WITH ED AND GPI METHODS.** Having introduced our proposed ERDED and ERDGPI methods, we now compare their performance to that of two competing methods, namely the ED and GPI schemes (described earlier). In terms of computational complexity, the ERDED method is most comparable to the ED scheme, and the ERDGPI method is most comparable to the GPI scheme. Therefore, we compare the ERDED method to the ED scheme and the ERDGPI method to the GPI scheme. For the images in our test set and several sampling densities, we used each of the ERDED, ED, ERDGPI, and GPI methods to generate a mesh and then measured the resulting approximation error in terms of PSNR. A representative subset of the results obtained is shown in Table 3.5 for the ERDED and ED methods and in Table 3.6 for the ERDGPI and GPI

Table 3.5: Comparison of the mesh quality obtained with the ERDED and ED methods

| Image | Samp. Density (%) | PSNR (dB) ERDED | PSNR (dB) ED |
|---|---|---|---|
| bull | 0.125 | **24.68** | 14.66 |
| | 0.250 | **28.86** | 17.36 |
| | 0.500 | **35.15** | 27.79 |
| | 1.000 | **39.02** | 34.16 |
| ct | 0.125 | **15.63** | 12.99 |
| | 0.250 | **25.97** | 13.23 |
| | 0.500 | **30.06** | 17.47 |
| | 1.000 | **36.51** | 20.82 |
| glasses2 | 0.500 | **20.54** | 16.55 |
| | 1.000 | **24.80** | 21.78 |
| | 2.000 | **28.71** | 26.02 |
| | 3.000 | **30.85** | 27.98 |
| lena | 0.500 | **20.56** | 17.60 |
| | 1.000 | **25.82** | 22.45 |
| | 2.000 | **29.28** | 26.90 |
| | 3.000 | **31.31** | 28.54 |
| mri | 0.250 | **12.55** | 10.87 |
| | 0.500 | **24.79** | 16.10 |
| | 1.000 | **30.33** | 15.55 |
| | 2.000 | **33.14** | 19.94 |
| peppers | 0.500 | **22.14** | 17.53 |
| | 1.000 | **25.97** | 22.42 |
| | 2.000 | **29.00** | 27.10 |
| | 3.000 | **30.18** | 29.06 |

Table 3.6: Comparison of the mesh quality obtained with the ERDGPI and GPI methods

| Image | Samp. Density (%) | PSNR (dB) ERDGPI | PSNR (dB) GPI |
|---|---|---|---|
| bull | 0.125 | **35.47** | 30.56 |
| | 0.250 | **38.33** | 35.30 |
| | 0.500 | **40.16** | 38.72 |
| | 1.000 | **41.39** | 40.95 |
| ct | 0.125 | **27.19** | 24.69 |
| | 0.250 | **32.11** | 29.93 |
| | 0.500 | **36.31** | 35.12 |
| | 1.000 | 39.49 | **39.82** |
| glasses2 | 0.500 | **24.94** | 23.65 |
| | 1.000 | **28.56** | 27.01 |
| | 2.000 | **32.11** | 31.00 |
| | 3.000 | **33.86** | 33.53 |
| lena | 0.500 | **25.58** | 24.22 |
| | 1.000 | **28.35** | 26.96 |
| | 2.000 | **30.79** | 29.74 |
| | 3.000 | **32.30** | 31.36 |
| mri | 0.250 | **27.14** | 26.34 |
| | 0.500 | **29.55** | 29.07 |
| | 1.000 | **33.01** | 32.14 |
| | 2.000 | **35.21** | 35.08 |
| peppers | 0.500 | **25.99** | 24.66 |
| | 1.000 | **28.40** | 27.49 |
| | 2.000 | **30.42** | 29.99 |
| | 3.000 | **31.50** | 31.19 |

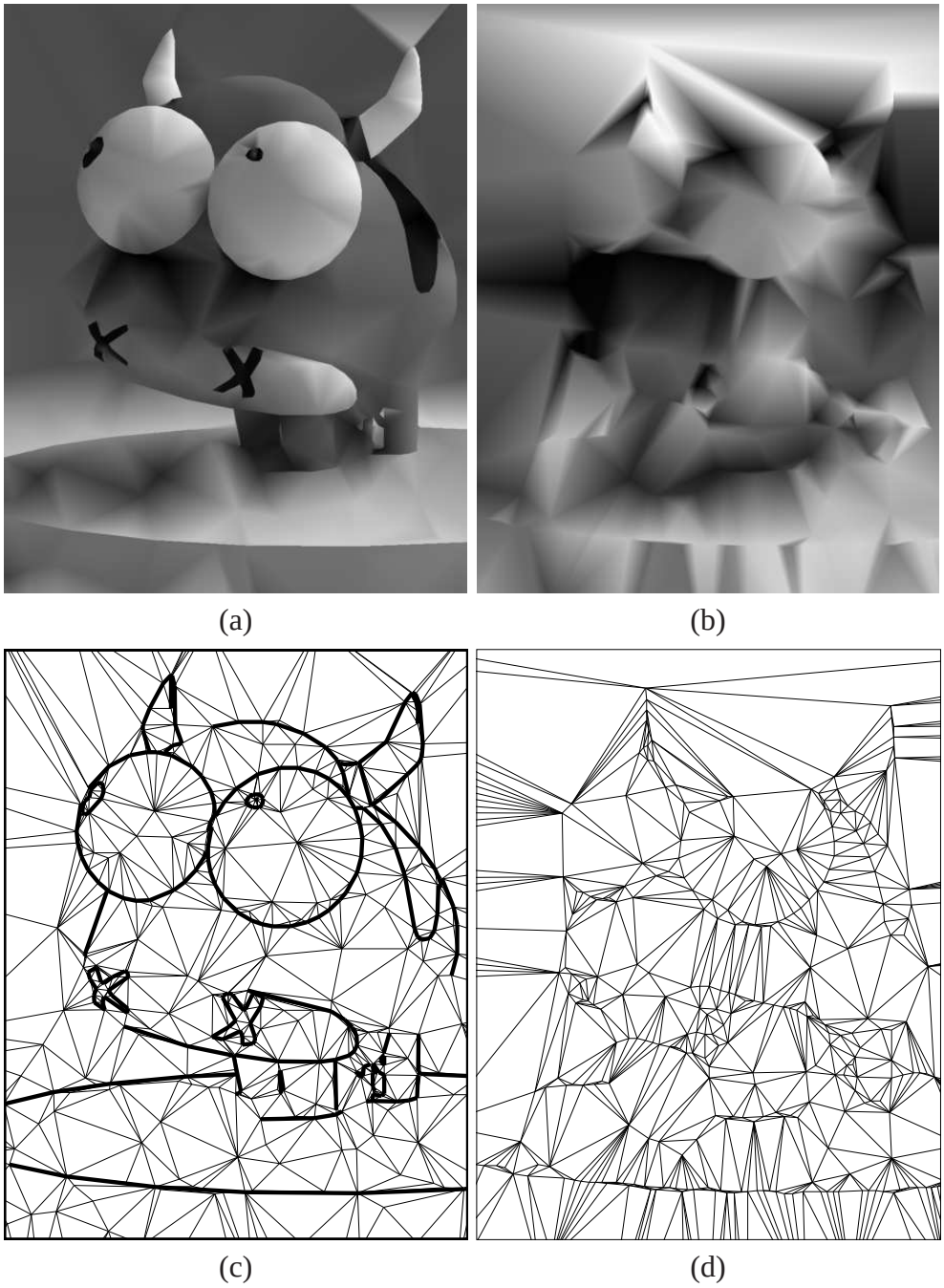
methods. In these tables, the best result in each test case is highlighted in **bold**.

*ERDED versus ED.* First, we compare the ERDED and ED methods. From the results of Table 3.5, we can see that the ERDED method outperforms the ED scheme in 24/24 of the test cases, by a margin of 1.12 to 15.69 dB (with a median of 3.77 dB). Subjective image quality was found to correlate reasonably well with PSNR. As examples to illustrate subjective quality, the image approximations for two of the test cases from Table 3.5 are shown in Figure 3.8 and Figure 3.9, respectively. The corresponding image-domain triangulations are also shown, with constrained edges (in the ERDED case) denoted by thick lines. Clearly, our ERDED method produces vastly superior image approximations (relative to the ED scheme), preserving image edges much more faithfully.

*ERDGPI versus GPI.* Now, we compare the ERDGPI and GPI methods. From the results of Table 3.6, we can see that the ERDGPI method outperforms the GPI scheme in 23/24 of the test cases, by margin of up to 4.91 dB (with a median of 1.08 dB). Again, subjective quality was found to correlate well with PSNR. As examples to illustrate subjective quality, the image approximations for two of the test cases from Table 3.6 are shown in Figure 3.10 and Figure 3.11, respectively. The corresponding image-domain triangulations are also shown, with constrained edges (in the ERDGPI case) denoted by thick lines. A close inspection of the two image approximations shows that the ERDGPI method more faithfully reproduces image edges and generally has less significant distortion, relative to the GPI scheme. With the ERDGPI method, the constrained edges in the triangulation align well with image edges, allowing for better image-edge reproduction.

*Computational complexity.* For the images in our test set and sampling densities in the range 0.5% to 3%, we used each of the ERDED, ED, ERDGPI, and GPI methods to generate a mesh and then measured the computational complexity of each method in terms of execution time. A representative subset of the results obtained is shown in Table 3.7. These execution times were obtained on relatively modest hardware (namely, a three-year old notebook computer with a 2.0 GHz Intel Core2 Duo CPU and 1 GB of RAM) with an implementation of our methods that was not optimized for execution time. We note that our ERDED and ERDGPI schemes are both quite modest in terms of their computational requirements compared to the ED and GPI methods. From Table 3.7, we can see that for the images in our test set and sampling densities in the range 0.5% to 3%, the ERDED, ED, ERDGPI, and GPI methods have average execution times of approximately 3.1234 seconds, 0.1293 seconds, 5.0134 seconds, and 1.2468 seconds, respectively. From the results, we notice that, although our proposed ERDED and ERDGPI methods are slower than the ED and the GPI schemes, they are still fairly fast. The differences (in absolute terms) between the execution times of all the four methods are not large. Since our proposed ERDED can produce much better results than the ED method, and the ERDGPI methods can also generate much better image approximations than the GPI method, such small differences in execution times can be neglected due to the large improvement in the quality of the resulting image approximations. Further more, since the implementations of our proposed ERDED and ERDGPI methods are not optimized, we can optimize the implementations of our ERDED and ERDGPI methods to reduce the execution times.

**COMPARISON WITH GVS METHOD.** As an additional point of reference, we compare our ERDED and ERDGPI methods to the **Garcia-Vintimilla-Sappa (GVS)** scheme



(a) (b)

(c) (d)

Figure 3.8: Comparison of the ERDED and ED methods. Part of the image approximation obtained for the `bull` image at a sampling density of 0.125% with the (a) ERDED (24.68 dB) and (b) ED (14.66 dB) methods, and (c) and (d) their corresponding triangulations.

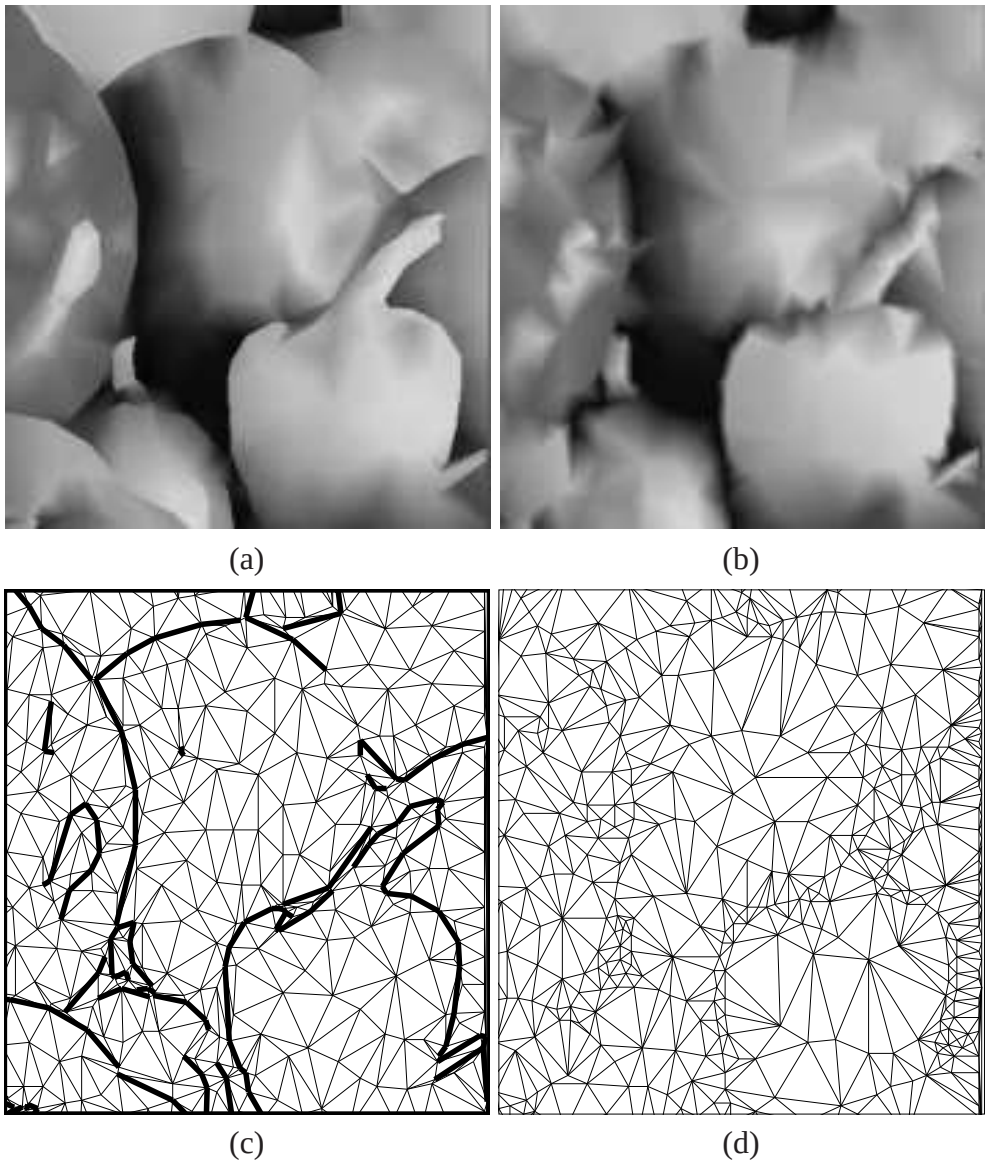



Figure 3.9: Comparison of the ERDED and ED methods. Part of the image approximation obtained for the `peppers` image at a sampling density of 0.125% with the (a) ERDED (25.97 dB) and (b) ED (22.42 dB) methods, and (c) and (d) their corresponding triangulations.

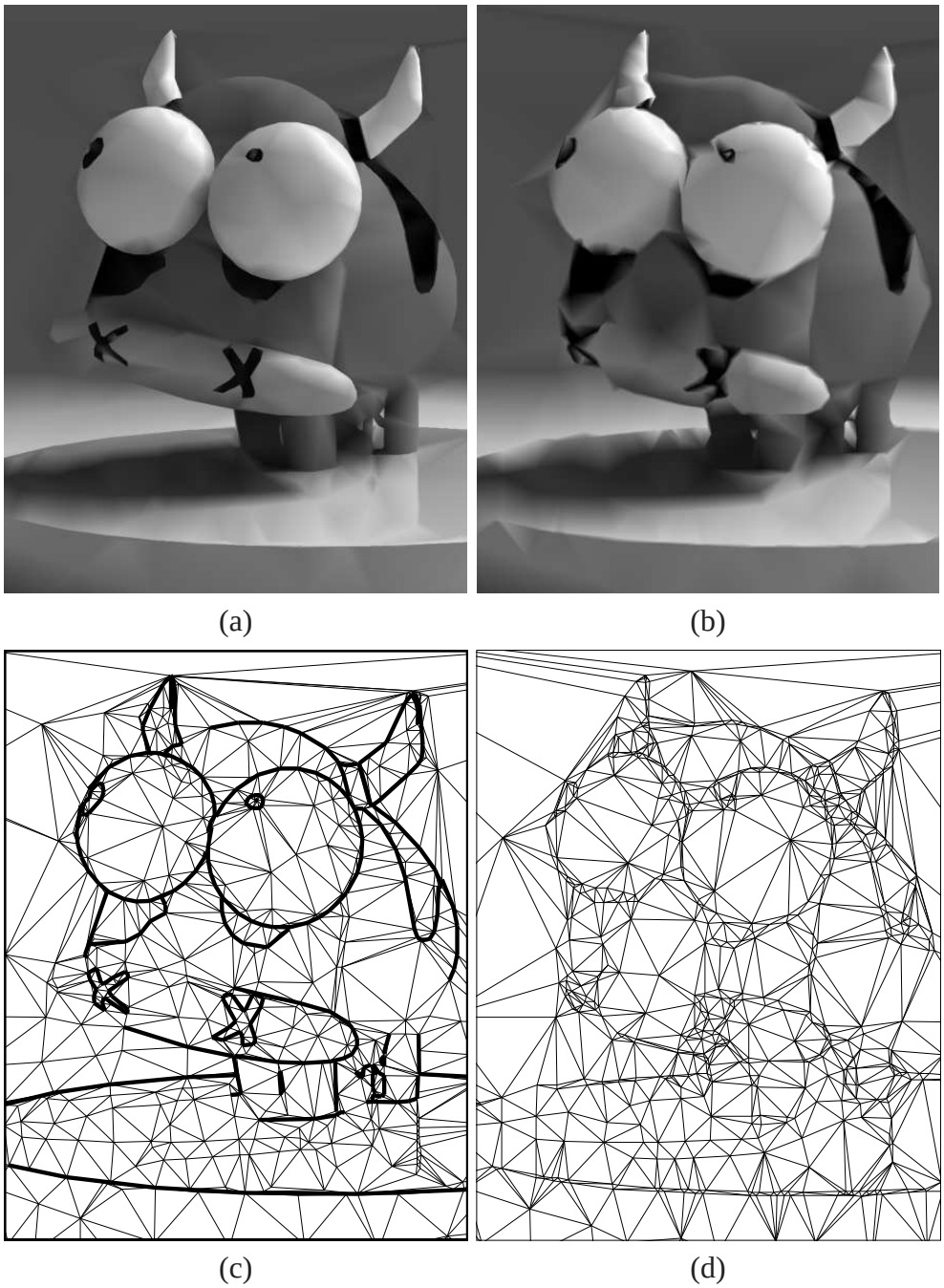



Figure 3.10: Comparison of the ERDGPI and GPI methods. Part of the image approximation obtained for the `bull` image at a sampling density of 0.125% with the (a) ERDGPI (35.47 dB) (b) GPI (30.56 dB) methods, and (c) and (d) their corresponding image-domain triangulations.

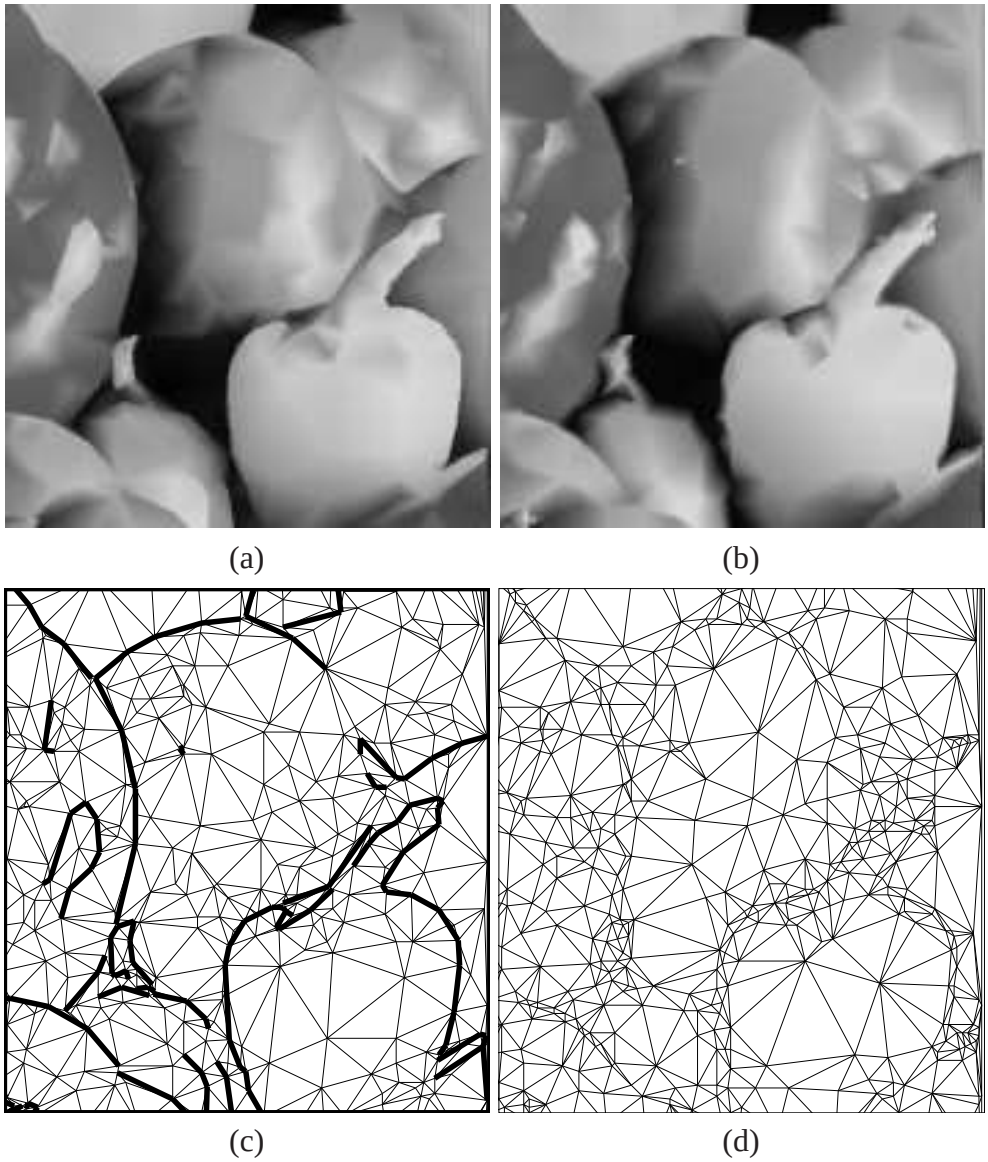



Figure 3.11: Comparison of the ERDGPI and GPI methods. Part of the image approximation obtained for the `peppers` image at a sampling density of 1% with the (a) ERDGPI (28.40 dB) (b) GPI (27.49 dB) methods, and (c) and (d) their corresponding image-domain triangulations.

Table 3.7: Comparison of the computational complexity of the ERDED, ED, ERDGPI, and GPI methods

| Image | Samp. Density (%) | Time (s) | | | |
|---|---|---|---|---|---|
| | | ERDED | ED | ERDGPI | GPI |
| lena | 0.5 | 1.9777 | 0.0587 | 2.7205 | 0.5528 |
| | 1.0 | 2.0850 | 0.0665 | 2.7773 | 0.6498 |
| | 2.0 | 2.0639 | 0.0694 | 3.0731 | 0.8632 |
| | 3.0 | 2.2055 | 0.0773 | 3.2111 | 0.9745 |
| peppers | 0.5 | 2.0551 | 0.0587 | 2.5124 | 0.4578 |
| | 1.0 | 2.1145 | 0.0653 | 2.8348 | 0.5612 |
| | 2.0 | 2.1054 | 0.0700 | 3.1626 | 0.7954 |
| | 3.0 | 2.1858 | 0.0809 | 3.2087 | 0.8787 |
| glasses | 0.5 | 6.1296 | 0.1758 | 8.4193 | 1.9404 |
| | 1.0 | 6.4190 | 0.1845 | 9.0583 | 2.2983 |
| | 2.0 | 6.6875 | 0.2346 | 10.444 | 2.8436 |
| | 3.0 | 6.6337 | 0.3000 | 10.714 | 3.5033 |
| mri | 0.5 | 0.4666 | 0.0143 | 0.5664 | 0.0996 |
| | 1.0 | 0.4912 | 0.0185 | 0.6816 | 0.1233 |
| | 2.0 | 0.5024 | 0.0168 | 0.7521 | 0.1681 |
| | 3.0 | 0.5141 | 0.0182 | 0.8227 | 0.1920 |
| ct | 0.5 | 1.9978 | 0.0621 | 2.4622 | 0.4809 |
| | 1.0 | 1.9759 | 0.0640 | 2.7363 | 0.5699 |
| | 2.0 | 2.0957 | 0.0719 | 3.2155 | 0.7663 |
| | 3.0 | 2.1037 | 0.0779 | 3.5545 | 0.8995 |
| bull | 0.5 | 5.9541 | 0.1787 | 10.379 | 1.8231 |
| | 1.0 | 6.0737 | 0.1979 | 10.889 | 2.2144 |
| | 2.0 | 6.2095 | 0.5043 | 10.529 | 2.7384 |
| | 3.0 | 6.0999 | 0.5805 | 11.597 | 3.5295 |

Table 3.8: Comparison of the mesh quality obtained with the ERDED, ERDGPI, and GVS methods

| Image | Mesh Size | PSNR (dB) | | |
|---|---|---|---|---|
| | | ERDED | ERDGPI | GVS |
| house | 1649 | 29.52 | 31.52 | 27.62 |
| lena | 7492 | 29.69 | 31.76 | 28.30 |
| peppers | 7224 | 28.82 | 31.12 | 26.48 |

from [12], which is based on constrained Delaunay triangulations. In [12], some mesh-generation results are provided for three standard test images (from the USC image database), namely, the house, lena, and peppers images. Using our ERDED and ERDGPI methods, we generated meshes to match the sizes of the meshes produced in [12] and then measured the resulting approximation error in terms of PSNR. Our results along with the ones from [12] are given in Table 3.8 for comparison. Examining the results of this table, we can see that our ERDED and ERDGPI methods each outperform the GVS scheme in every case by a margin of 1.39 to 2.34 dB and 3.46 to 4.64 dB, respectively. Clearly, our ERDED and ERDGPI methods are both superior to the GVS scheme.

**COMPARISON WITH THE RESULTS OBTAINED WITH CHOOSING PARAMETERS MANUALLY.** As we mentioned earlier, the parameters $\beta$, $r$, $\ell$, and $\varepsilon$ in the ERDED and ERDGPI methods are chosen by the automated scheme proposed in Section 3.4.1. Although the automated scheme chooses the parameters intelligently, the results obtained can still be improved by manually choosing a better set of parameters. To demonstrate this, for the images in our test set and several sampling densities, we used each of the ERDED and ERDGPI methods with the set of parameters carefully chosen manually to generate meshes, measured the quality of the results in terms of PSNR, and then compared with the results obtained by choosing the parameters automatically. A representative subset of the results obtained is shown in Table 3.9 for ERDED method and Table 3.10 for ERDGPI method. We can see that, with the parameters chosen manually, both the ERDED and ERDGPI methods outperforms the proposed methods with an automated parameter-choosing scheme in all 24/24 of the test cases by margin of up to 2.64 dB (with an average of 0.55 dB) and 0.41 dB (with an average of 0.22 dB), respectively.

Although we can obtain better results by manually choosing the set of parameters, the process of parameters selection is fairly tedious and time-consuming. We need to try many times before we can obtain a better set of parameters than that is chosen automatically. Also, for those people who are not familiar with the parameters, it is not easy to choose the parameters. Therefore, although the proposed automated parameter selection scheme

Table 3.9: Comparison of the mesh quality obtained with the ERDED method

| Image | Samp. Density (%) | PSNR (dB) Auto | PSNR (dB) Manual |
|---|---|---|---|
| bull | 0.125 | 24.68 | **25.65** |
| | 0.250 | 28.86 | **30.61** |
| | 0.500 | 35.15 | **35.52** |
| | 1.000 | 39.02 | **39.16** |
| ct | 0.125 | 15.63 | **17.29** |
| | 0.250 | 25.97 | **26.02** |
| | 0.500 | 30.06 | **30.53** |
| | 1.000 | 36.51 | **36.85** |
| glasses2 | 0.500 | 20.54 | **20.96** |
| | 1.000 | 24.80 | **25.17** |
| | 2.000 | 28.71 | **28.97** |
| | 3.000 | 30.85 | **30.89** |
| lena | 0.500 | 20.56 | **23.20** |
| | 1.000 | 25.82 | **26.56** |
| | 2.000 | 29.28 | **29.48** |
| | 3.000 | 31.31 | **31.35** |
| mri | 0.250 | 12.55 | **13.06** |
| | 0.500 | 24.79 | **25.17** |
| | 1.000 | 30.33 | **30.75** |
| | 2.000 | 33.14 | **33.51** |
| peppers | 0.500 | 22.14 | **22.58** |
| | 1.000 | 25.97 | **26.43** |
| | 2.000 | 29.00 | **29.10** |
| | 3.000 | 30.18 | **30.25** |

Table 3.10: Comparison of the mesh quality obtained with the ERDGPI method

| Image | Samp. Density (%) | PSNR (dB) Auto | PSNR (dB) Manual |
|---|---|---|---|
| bull | 0.125 | 35.47 | **35.85** |
| | 0.250 | 38.33 | **38.41** |
| | 0.500 | 40.16 | **40.23** |
| | 1.000 | 41.39 | **41.43** |
| ct | 0.125 | 27.19 | **27.25** |
| | 0.250 | 32.11 | **32.51** |
| | 0.500 | 36.31 | **36.62** |
| | 1.000 | 39.49 | **39.61** |
| glasses2 | 0.500 | 24.94 | **25.34** |
| | 1.000 | 28.56 | **28.97** |
| | 2.000 | 32.11 | **32.21** |
| | 3.000 | 33.86 | **33.94** |
| lena | 0.500 | 25.58 | **25.96** |
| | 1.000 | 28.35 | **28.73** |
| | 2.000 | 30.79 | **30.98** |
| | 3.000 | 32.30 | **32.39** |
| mri | 0.250 | 27.14 | **27.34** |
| | 0.500 | 29.55 | **29.95** |
| | 1.000 | 33.01 | **33.24** |
| | 2.000 | 35.21 | **35.60** |
| peppers | 0.500 | 25.99 | **26.12** |
| | 1.000 | 28.40 | **28.55** |
| | 2.000 | 30.42 | **30.55** |
| | 3.000 | 31.50 | **31.55** |

usually yields somewhat poorer results than manual selection, it is vastly superior to manual selection in a practical sense (i.e., in terms of usability).

# Chapter 4

# Conclusions and Future Research

## 4.1 Conclusions

In this thesis, we studied geometric image representations based on triangle meshes. In particular, we proposed a new mesh model that explicitly represents image discontinuities along with two mesh-generation methods that select the model parameters for a given input image. The main contributions of the thesis are summarized in what follows.

As mentioned earlier, the discontinuity (i.e., image edge) is an important feature of an image that is easily captured by human eyes. Many of the previously-proposed mesh models do not explicitly take image discontinuities into consideration. In our work, we proposed a new mesh model that explicitly represents image discontinuities. This mesh model is based on constrained DTs, where the constrained edges correspond to image edges.

Having introduced this mesh model, we then proposed two approaches that select the model parameters for a given input image. One of the proposed methods is named ERDED, which employs the ED scheme to select sample points that are not on the constrained edges. The other method is called ERDGPI, where the GPI method is employed to select a subset of the sample points.

The proposed methods have several parameters that affect the quality of the image approximation obtained. The selection of these parameters was analyzed, including the selection of contrast parameter $\gamma$ of ED scheme, $\beta$ and $r$ in hysteresis thresholding, and the tolerance $\varepsilon$ in the Douglas-Peucker polyline simplification algorithm. Furthermore, we proposed an automated parameters selection scheme that chooses these parameters intelligently.

Through experimental results, we evaluated the performance of our proposed ERDED

and ERDGPI methods in terms of both PSNR and subjective image quality. By comparison, we found that the image approximations produced by our proposed ERDED method are often about 3.77 dB higher in PSNR than those produced by the ED scheme, and our proposed ERDGPI method can generate reconstructed images of about 1.08 dB higher PSNR than those produced by the GPI scheme.

## 4.2 Future Research

In this thesis, we first presented a mesh model that explicitly represents image discontinuities, and then proposed two mesh-generation methods that select the model parameters for a given input image. Although our proposed mesh-generation methods work fairly well, additional work to further refine these methods could be beneficial and may lead to higher quality meshes.

As discussed earlier, edge detection is a key step in our method to find the constrained edges for a constrained DT, and the accuracy and consistency of the edges obtained are extremely important. In our methods, the modified Canny edge detector is employed for the sake of the consistency of the edges, and the edges are detected in half-pixel resolution to improve the accuracy. As future work one might try to find a way to further improve the accuracy and the consistency of the edges. For example, since many edge detection methods have been proposed to date, a better edge detector which focuses on the consistency of the edges might be beneficial. Furthermore, we can also apply the edge detector to an even higher resolution version of the input image to enhance the accuracy of the detected edges.

In the hysteresis thresholding employed by the edge detector in our work, two parameters need to be specified, namely $\beta$ and $r$. These parameters greatly affect the quality of the image approximations that are obtained. It is not easy, however, to choose good values for these parameters, since the best values are dependent on the sampling density. Although in our work, we propose an automated parameters selection scheme, based on numerous of experiments, this scheme could probably be improved. Finding better means for selecting these hysteresis thresholding parameters would be another area for future work.

In the mesh model presented in our work, the wedge is an important feature, which facilitates the modelling of image discontinuities. Therefore, the way in which the wedge value is chosen for wedges is very important. In our work, we presented a wedge value selection method that is based on the MMSODD of the image. One defect, however, of proposed wedge selection method is that the second-order derivatives are very sensitive to noise. Although a binomial filter is employed to smooth the image before calculating

the MMSODD, this blurs the image, which degrades the resulting image approximations obtained with the method. Therefore, the development of wedge value selection scheme that does not have this weakness might be beneficial.

# Bibliography

[1] D. L. Donoho, "Orthonormal ridgelets and linear singularities," *Journal on Mathematical Analysis of the Society for Industrial and Applied Mathematics*, vol. 31, no. 5, pp. 1062–1099, Dec. 2000.

[2] E. Candes and D. Donoho, "Curvelets: A surprisingly effective nonadaptive representation of objects with edges," In Curves and Surfaces. Vanderbilt University Press, Nashville, TN, USA, 1999, pp. 105–120.

[3] R. H. Chan, S. D. Riemenschneider, L. Shen, and Z. Shen, "High-resolution image reconstruction with displacement errors: A framelet approach," *International Journal of Imaging Systems and Technology*, vol. 14, no. 3, pp. 91–104, Sep. 2004.

[4] D. Donoho and X. Huo, "Combined image representation using edgelets and wavelets," vol. 3813, Wavelet Applications in Signal and Image Processing VII, in Proceedings SPIE. Denver, Colorado, Oct. 1999, pp. 468–476.

[5] M. N. Do and M. Vetterli, "The contourlet transform: an efficient directional multiresolution image representation," *IEEE Transactions on Image Processing*, vol. 14, no. 12, pp. 2091–2106, Dec. 2005.

[6] D. D.-Y. Po and M. N. Do, "Directional multiscale modeling of images using the contourlet transform," *IEEE Transactions on Image Processing*, vol. 15, no. 6, pp. 1610–20, Jun. 2006.

[7] D. L. Donoho, "Wedgelets: nearly minimax estimation of edges," *Annals Statistics*, vol. 27, no. 3, pp. 859–897, 1999.

[8] E. L. Pennec and S. Mallat, "Sparse geometric image representation with bandelets," *IEEE Transactions on Image Processing*, vol. 14, no. 4, pp. 423–438, Apr. 2005.

[9] R. B. M. Jansen and S. Lavu, “Multiscale approximation of piecewise smooth two-dimensional functions using normal triangulated meshes,” *Applied and Computational Harmonic Analysis*, vol. 19, no. 1, pp. 92–130, Jul. 2005.

[10] M. Garland and P. S. Heckbert, “Fast polygonal approximation of terrains and height fields,” vol. 95-181. School of Computer Science, Carnegie Mellon University, Sep. 1995.

[11] D. Terzopoulos and M. Vasilescu, “Sampling and reconstruction with adaptive meshes,” IEEE International Conference on Computer Vision and Pattern Recognition. Hawaii, USA, Jun. 1991, pp. 70–75.

[12] M. A. Garcia, B. Vintimilla, and A. Sappa, “Approximation and processing of intensity images with discontinuity-preserving adaptive triangular meshes,” Sixth European Conference on Computer Vision. LNCS Vol. 1842, Springer Verlag, Dublin, Ireland, Jul. 2000, pp. 844–855.

[13] Y. Yang, M. N. Wernick, and J. G. Brankov, “A fast approach for accurate content-adaptive mesh generation,” *IEEE Transactions on Image Processing*, vol. 12, no. 8, pp. 866–881, Aug. 2003.

[14] M. D. Adams, “An improved content-adaptive mesh-generation method for image represnetation,” IEEE International Conference on Image Processing. Hong Kong, China, Sep. 2010, pp. 873–876.

[15] M. Sarkis and K. Diepold, “A fast solution to the approximation of 3-d scattered point data from stereo images using triangular meshes,” in Proceedings of IEEE-RAS International Conference on Humanoid Robots. Pittsburgh, PA, USA, Nov. 2001, pp. 235–241.

[16] ——, “Texture recognition from sparsely and irregularly sampled data,” *Computer Vision and Image Understanding*, vol. 102, no. 1, pp. 95–104, Apr. 2006.

[17] J. G. Brankov, Y. Yang, and N. P. Galatsanos, “Image restoration using content-adaptive mesh modeling,” vol. 2. in Proceedings of IEEE International Conference on Image Processing, 2003, pp. 997–1000.

[18] M. D. Adams, “Progressive lossy-to-lossless coding of arbitrarily-sampled image data using the modified scattered data coding method,” in Proceedings of IEEE Interna-

tional Conference on Acoustics, Speech, and Signal Processing. Taipei, Taiwan, China, Apr. 2009, pp. 1017–1020.

[19] G. Ramponi and S. Carrato, "An adaptive irregular sampling algorithm and its application to image coding," *Image and Vision Computing*, vol. 19, no. 7, pp. 451–460, May 2001.

[20] P. Lechat, H. Sanson, and L. Labelle, "Image approximation by minimization of a geometric distance applied to a 3d finite elements based model," vol. 2. in Proceedings of IEEE International Conference on Image Processing, 1997, pp. 724–727.

[21] Y. Wang, O. Lee, and A. Vetro, "Use of two-dimensional deformable mesh structures for video coding, part iithe analysis problem and a region-based coder employing an active mesh representation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 6, pp. 647–659, Dec. 1996.

[22] 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*, vol. 5, no. 2, pp. 338–346, Feb. 1996.

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

[24] R. Floyd and L. Steinberg, "An adaptive algorithm for spatial greyscale," *Proceedings of the Society for Information Display*, vol. 17, no. 2, pp. 75–77, 1976.

[25] C. L. Huang and C. Y. Hsu, "A new motion compensation method for image sequence coding using hierarchical grid interpolation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, pp. 44–51, Sep. 1994.

[26] T. Gevers and A. W. Smeulders, "Combining region splitting and edge detection through guided delaunay image subdivision." IEEE International Conference on Computer Vision and Pattern Recognition, Jun. 1997, pp. 1021–1026.

[27] M. A. Garcia, B. X. Vintimilla, and A. D. Sappa, "Approximation of intensity images with adaptive triangular meshes: Toward a processible compressed representation," in Proceeding Irish Machine Vision and Image Processing Conference. Dublin, Ireland, Sep. 1999, pp. 241–249.

[28] ——, “Efficient approximation of gray-scale image through bounded error triangular meshes,” vol. 1, IEEE International Conference on Image Processing. Kobe, Japan, Oct. 1999, pp. 168–170.

[29] M. D. Adams, “An evaluation of several mesh-generation methods using a simple mesh-based image coder,” in Proceedings of IEEE International Conference on Image Processing. San Diego, CA, USA, Oct. 2008, pp. 1041–1044.

[30] L. Demaret and A. Iske, “Advances in digital image compression by adaptive thinning,” vol. 3, in Annals of the Marie-Curie Fellowship Association. Marie Curie Fellowship Association, Feb. 2004, pp. 105–109.

[31] B. Delaunay, “Sur la sphere vide, izvestia akademii nauk sssr,” *Otdelenie Matematicheskikh i Estestvennykh Nauk*, vol. 7, no. 6, p. 793800, 1934.

[32] N. Dyn, D. Levin, and S. Rippa, “Data dependent triangulations for piecewise linear interpolation,” *The IMA Journal of Numerical Analysis*, vol. 10, no. 1, pp. 137–154, Jan. 1990.

[33] S. Rippa, “Adaptive approximation by piecewise linear polynomials on triangulations of subsets of scattered data,” *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 5, pp. 1123–1141, Sep. 1992.

[34] D. Salesin, D. Lischinski, and T. DeRose, “Reconstructing illumination functions with selected discontinuities,” in *In Third Eurographics Workshop on Rendering*, 1992, pp. 99–112.

[35] X. Tu and M. D. Adams, “Image representation using triangle meshes with explicit discontinuities,” Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. Victoria, BC, Canada, Aug. 2011, pp. 97–101.

[36] J. O’Rourke, *Computational Geometry in C*. Cambridge University Press, 1997.

[37] M. Berg, *Computational Geometry Algorithms and Applications*. Springer, 1998.

[38] O. Devillers and M. Teillaud, “Perturbations and vertex removal in a 3d delaunay triangulation,” *14th ACM-Siam Symposium on Algorithms*, pp. 313–319, 2003.

[39] H. Edelsbrunner and E. P. Mucke, “Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms,” *ACM Transaction on Graphics 9*, pp. 67–104, 1990.

[40] E. P. Mucke, “A robust implementation for three-dimensional delaunay triangulations,” *International Journal of Computational Geometry and Applications 8*, pp. 255–276, 1998.

[41] C. Dyken and M. S. Floater, “Preferred directions for resolving the non-uniqueness of delaunay triangulations,” *Computational Geometry Theory and Applications*, vol. 34, no. 2, pp. 96–101, 2006.

[42] L. P. Chew, “Constrained Delaunay triangulations,” *Algorithmica*, vol. 4, pp. 97–108, 1989.

[43] D. Douglas and T. Peucker, “Algorithms for the reduction of the number of points required to represent a digitized line or its caricature,” *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, Oct. 1973.

[44] M. Aubury and W. Luk, “Binomial filters,” *Journal of VLSI Signal Processing*, vol. 12, no. 1, pp. 35–50, Jan. 1996.

[45] R. Haddad and A. Akansu, “A class of fast gaussian binomial filters for speech and image processing,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 39, no. 3, pp. 723–727, Mar. 1991.

[46] W. K. Pratt, *Digital Image Processing*, 4th ed. Hoboken, NJ, USA: Wiley, 2007.

[47] D. Marr and E. Hildreth, “Theory of edge detection,” vol. B-207, no. 1167, Proceedings of the Royal Society. London, Feb. 1980, pp. 187–217.

[48] R. Haralick, “Digital step edges from zero crossing of second directional derivatives,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Jan.

[49] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[50] R. Malladi, J. A. Sethian, and B. C. Vemuri, “Segmentation and object recognition using edge detection techniques,” *International Journal of Computer Science and Information Technology*, vol. 2, no. 6, pp. 153–161, Dec. 2010.

[51] R. Malladi and J. A. Sethian, “A topology independent shape modeling scheme,” *In SPIEs Geometric Methods in Computer Vision II*, vol. SPIE 2031, pp. 246–255, Jun. 1993.

[52] N. Paragios and R. Deriche, “Geodesic active contours and level sets for the detection and tracking of moving objects,” *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 22, no. 3, pp. 266–280, Mar. 2000.

[53] Z. Guo and R. W. Hall, “Parallel thinning with two-subiteration algorithms,” *Communications of the ACM*, vol. 32, no. 3, pp. 359–373, Mar. 1989.

[54] L. Ding and A. Goshtasby, “On the canny edge detector,” *Pattern Recognition*, vol. 34, no. 3, pp. 721–725, Mar. 2001.

[55] M. K. Agoston, *Computer Graphics and Geometric Modeling: Implementation and Algorithms*. London, UK: Springer, 2005.

[56] K. Fleischer and D. Salesin, “Accurate polygon scan conversion using half-open intervals,” in *Graphics Gems III*, D. Kirk, Ed. Academic Press, 1992, pp. 362–365.

[57] “JPEG-2000 test images,” ISO/IEC JTC 1/SC 29/WG 1 N 545, Jul. 1997.

[58] “Kodak lossless true color image suite,” 2011, http://r0k.us/graphics/kodak.

[59] “USC-SIPI image database,” 2011, http://sipi.usc.edu/database.

[60] “Michael Adams’ research datasets,” 2011, http://www.ece.uvic.ca/~mdadams/datasets.

# Appendix A

# Software

## A.1 Overview

During the course of his research, the author of this thesis developed software implementations of several methods described herein. This software was written in C++, involves some fairly complicated algorithms and data structures, and consists of more than 14,000 lines of code. The software implements four mesh-generation methods, the ED scheme, the GPI scheme, the proposed ERDED method, and the proposed ERDGPI method. With a given input image and several appropriate parameters, the software can produce a mesh model of the original image with a prespecified mesh-generation method, reconstruct an image approximation from the mesh model, and calculate the approximation error in terms of PSNR. The software utilizes the classes provided by the CGAL library for 2-D triangulations, and 2-D constrained Delaunay triangulations. The input image is restricted to be in the plain PNM format, which can be easily converted from other image formats.

## A.2 Extracting the Software

The software is distributed in the form of a Zip file. Therefore, in order to extract the contents of this file, a program capable of handling Zip archives is required. Such software is readily available for many different computing platforms, and can be obtained from:

- Info-Zip Web Site (i.e., `http://www.info-zip.org`). Unzip software for many different computing platforms (e.g., UNIX, DOS, MacOS, etc.).
- WinZip Web Site (i.e., `http://www.winzip.com`). Zip/Unzip software for Mi-

crosoft Windows.

## A.3 Building the Software

Building software refers to the process of converting source code files into into an executable program. The software is intended to be built using the standard UNIX make utility, a utility that automatically builds executable programs and libraries from source code by reading files called makefiles which specify how to derive the target program. The Make utility needs to compile and link the various files, in the correct order. If the source code in a particular file has not changed then it may not need to be recompiled. A C++ compiler is also needed to build the software. If you need a C++ compiler, you can obtain the GNU Compiler Collection (GCC) from the GNU Project web site (i.e., http://www.gnu.org). If you need an implementation of the make utility, you can also obtain GNU Make from the GNU Project web site. All GNU software is free software.

### A.3.1 Building Process

In what follows, `$TOPDIR` denotes the top level directory of the software distribution. To build the software, the following steps are required (in order):

1. *Extract the contents of the archive file containing the software distribution to* `$TOPDIR`.

2. *Set the current working directory to the top level directory of the software distribution.*

   To set the current working directory as required, type:

   ```
   cd $TOPDIR
   ```

   (where `$TOPDIR` is defined as described above).

3. *Compile and link the software.*

   This is accomplished via the `make` command. To run the make program, type:

   ```
   make
   ```

4. *Install the software.*

This step may require special (e.g., superuser/administrator) privileges depending on the target directory for installation. (The default installation directories are normally under `$TOPDIR/bin`.) To install the executables, type:

```
make install
```

Presuming that the build was successful, the executables for the software programs can be found in the directory `$TOPDIR/bin`.

### A.3.2 Dependencies on Other Software

In order to have access to the full functionality of the software, you may need to install some additional libraries on your system. Since the software utilizes many data structures and algorithms provided by CGAL library, to build the software, CGAL must be installed. You can download and install the free CGAL library which is available from the CGAL web site (i.e., `http://www.cgal.org`).

## A.4 Image Models Used in the Software

Two image models are utilized in the software. The first image model is the basic model described in Chapter 2, which is based on the DT, and can be uniquely characterized by a set of sample points together with the associated sample values. The basic model is generated by the ED and GPI methods. The second image model is the ERD model introduced in Section 3.3, which can be generated by our proposed ERDED and ERDGPI methods. In our software, the simple model is represented by a file of extension `.model`, and the ERD model is represented by a file of extension `.erd`. The `.model`(i.e., the basic model) and `.erd`(i.e., the ERD model) formats used in our software are described in detail in what follows.

### A.4.1 The format of the basic model.

For the basic model, the file format typically uses the "`.model`" file name extension. The format consists of a maximum intensity value(i.e., a number), followed by vertex and face data.

The vertex data consists of the following information (in order): 1) The number of vertices in the mesh, 2) the information of each vertex in the mesh, followed by a newline

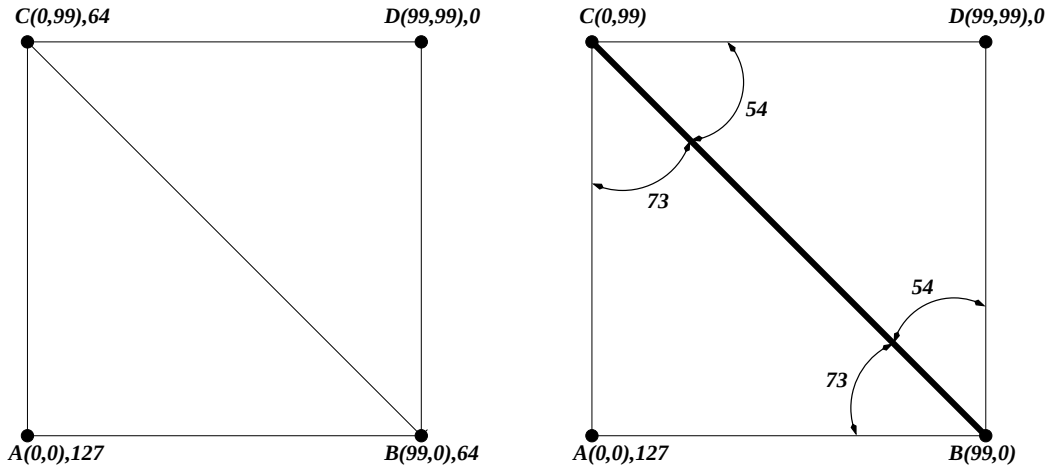



Figure A.1: Example of (a) the basic model and (b) the ERD model.

character. In particular, for each vertex in the mesh, the following information is given: the *x*, *y* coordinates of the vertex and the value of the approximating function at the point $(x,y)$ (in that order) separated by whitespace and followed by a newline character.

The above vertex data is then followed by face data. The face data consists of the following information (in order): 1) the number of faces in the mesh, 2) the information of each face in the mesh, followed by a newline character. In particular, for each face in the mesh, the following information is provided separated by whitespace: a) for each vertex in the face (in CCW order), the index of the vertex, b) a newline character. Note that vertices are indexed starting from zero.

An example of a file for a mesh model with two faces and, four vertices(i.e., $A(0,0)$, $B(0,99)$, $C(99,0)$, and $D(99,99)$ along with the corresponding intensity values) named `simple.model` is shown below. The image-domain mesh of `simple.model` is illustrated in Figure A.1(a).

```
simple.model

255

4
0 0 127
0 99 64
99 0 64
99 99 0
```

```
2
0 1 2
1 2 3
```

### A.4.2 The format of the ERD model.

For the ERD model, the file format typically uses the ".erd" file name extension. The `.erd` format is similar to the `.model` format. The `.erd` format consists of a maximum intensity value, followed by vertex, face and wedge data.

The vertex data consists of the following information (in order): 1) The number of vertices in the mesh. 2) the information of each vertex in the mesh, followed by a newline character. In particular, for each vertex in the mesh, the following information is given: the *x*, *y* coordinates of the vertex and the value of the approximating function at the point $(x,y)$ (in that order) separated by whitespace and followed by a newline character. For any vertex that has more than one wedge, the vertex's function value is ignored since the value should be the average values of all the wedges around the vertex.

The above vertex data is then followed by face data. The face data consists of the following information (in order): 1) the number of faces in the mesh, 2) the information of each face in the mesh, followed by a newline character. In particular, for each face in the mesh, the following information is provided separated by whitespace: a) for each vertex in the face (in CCW order), the index of the vertex, b) a newline character. Note that vertices are indexed starting from zero.

The above face data is finally followed by wedge data (only the vertices have more than one wedges are presented). In particular, for each wedge in the mesh, the following information is provided separated by whitespace: 1) The index of the vertex of the wedge, 2) The index of a face that belongs to the wedge, 3) The wedge value, 4) a newline character. Note that faces are indexed starting from zero.

An example of a file for a mesh model with two faces, four vertices(i.e., $A(0,0)$, $B(0,99)$, $C(99,0)$, and $D(99,99)$ along with the corresponding intensity values) and four wedge values named `simple.erd` is shown below. The image-domain mesh of `simple.erd` is illustrated in Figure A.1(b), where a constrained edge is represented by a thick line. Note that the vertices in the ERD model fall on a grid with half-pixel resolution. That is, a vertex $v_l = (x_l, y_l)$ must be such that $v_l \in \frac{1}{2}\mathbb{Z}^2 \bigcap [0, W-1] \times [0, H-1]$. In the data file, however, vertex coordinates are scaled so that the coordinate values specified in the file are always

integer value. In particular, for a vertex with coordinates $(x_l, y_l)$, the values stored in the data file correspond to the point with the coordinates $(2x_l, 2y_l)$, which will always be a point with integer coordinates.

```
simple.erd

255

4
0 0 127
0 99 64
99 0 64
99 99 0

2
0 1 2
1 2 3

4
1 0 73
1 1 54
2 0 73
2 1 54
```

## A.5 Application Programs

The software can accomplish three tasks, that are producing a mesh model for a given input image, reconstructing an image approximation from the mesh model, and evaluating the reconstructed image approximation in terms of PSNR. Therefore, the software is divided into three parts, and each part accomplishes a certain functionality described above. The detail of each part is described as below.

### A.5.1 Producing the Mesh Model

Besides our proposed ERDED and ERDGPI methods, two other schemes, namely the ED scheme and GPI scheme, are also implemented for the purpose of comparison. This soft-

ware contains four commands, `ywb`, `gpi`, `erd_ed`, and `erd_gpi`, respectively. These commands can produce the mesh models described earlier, with a given input image and some appropriate parameters. The details of the four commands are described as follows.

## The `ywb` Command

### Synopsis

```
ywb [options]
```

### Description

The `ywb` command is an implementation of the ED scheme introduced in Section 3.2.1. With an input image, and a desired number $K$ of sample points or a desired sampling density $d$, the `ywb` command uses the ED scheme to generate a basic model (`.model` file) of the given input image.

### Options

The ywb command accepts the following options:

`-i $inputFile`

: Read the input image from the file named `$inputFile` in the format of plain pnm.

`-d $sampleDensity`

: Set the sampling density to `$sampleDensity`(%). The default value is $3(\%)$.

`-n $num`

: Set the number of sampling points to `$num`. The default value is 7864. If `$sampleDensity` is set, `$num` is ignored.

`-o $outputFile`

: Output the image model to the file named `$outputFile` in the format of `.model`.

`-t $triangulationFile`

: Output the triangulation data to the file named `$triangulationFile` in the triangulation format used by iviewer software.

`-p $polylineFile`

Output the polyline data to the file named `$polylineFile` in the polyline set format used by iviewer software.

`-i $exitMode`

1) If `$exitMode`= 0, then the program will print an error message to standard error and then exit with an exit status of 3 (by calling "exit(3)") if the sampling density cannot be satisfied exactly; the error message should be of the form (all on one line) "ERROR: target sampling density not achieved exactly (requested XXX but got YYY)".

2) If `$exitMode` = 1, then the program will still complete normally if the sampling density cannot be satisfied exactly, but the program should print a warning message of the form "WARNING: target sampling density not achieved exactly (requested XXX but got YYY)" to standard error.

3) If `$exitMode`= 2, then the program will still complete normally if the sampling density cannot be satisfied exactly and no warning message is issued.

The default value of `$exitMode` is 0.

### Examples

For example, a basic model `lena.model` of the `lena.pnm` image at a sampling density of 3% is produced by running the following command:

```
ywb -i lena.pnm -d 3 -o lena.model
```

## The `gpi` Command

### Synopsis

```
gpi [options]
```

### Description

The `gpi` command is an implementation of the GPI scheme introduced in Section 3.2.2. With an input image, and a desired number $K$ of sampling points or a desired sampling density $d$, the `gpi` command uses the GPI scheme to generate a basic model of the given input image.

**Options**

The `gpi` command accepts the following options:

`-i $inputFile`

Read the input image from the file named `$inputFile` in the format of plain pnm.

`-d $sampleDensity`

Set the sampling density to `$sampleDensity`(%). The default value is 3(%).

`-n $num`

Set the number of sampling points to `$num`. The default value is 7864

`-o $outputFile`

Output the image model to the file named `$outputFile` in the format of `.model`.

`-t $triangulationFile`

Output the triangulation data to the file named `$triangulationFile` in the triangulation format used by iviewer software.

`-p $polylineFile`

Output the polyline data to the file named `$polylineFile` in the polyline set format used by iviewer software.

`-x $exitMode`

1) if `$exitMode` = 0, then the program will print an error message to standard error and then exit with an exit status of 3 (by calling "exit(3)") if the sampling density cannot be satisfied exactly; the error message should be of the form (all on one line) "ERROR: target sampling density not achieved exactly (requested XXX but got YYY)".

2) if `$exitMode` = 1, then the program will still complete normally if the sampling density cannot be satisfied exactly, but the program should print a warning message of the form "WARNING: target sampling density not achieved exactly (requested XXX but got YYY)" to standard error.

3) if `$exitMode` = 2, then the program will still complete normally if the sampling density cannot be satisfied exactly and no warning message is issued.

The default value of `$exitMode` is 0.

### Examples

After running the example shown below, a basic model `lena.model` of the `lena.pnm` image at a sampling density of 1% is produced.

```
gpi -i peppers.pnm -d 1 -o peppers.model
```

## The `erd_ed` command

### Synopsis

```
erd_ed [options]
```

### Description

The `erd_ed` command is an implementation of our proposed ERDED method. With an input image, a desired number $K$ of sampling points or a desired sampling density $d$, and some appropriate parameters, the `erd_ed` command uses the ERDED method to generate an ERD model of the given input image.

### Options

The `erd_ed` program accepts the following options:

`-i $inputFile`

: Read the input image from the file named `$inputFile` in the format of plain pnm.

`-d $sampleDensity`

: Set the sampling density to `$sampleDensity`(%). The default value is $3(\%)$.

`-n $num`

: Set the number of sampling points to `$num`. The default value is 7864. If the parameter `$sampleDensity` is set, `$num` is ignored.

`-o $outputFile`

: Output the image model to the file named `$outputFile` in the format of `.erd`.

`-t $triangulationFile`

: Output the triangulation data to the file named `$triangulationFile` in the triangulation format used by iviewer software.

`-p $polylineFile`

Output the polyline data to the file named `$polylineFile` in the polyline set format used by iviewer software.

`-l $minLength`

Set the minimum length of the polylines to be kept to `$minLength`. Any polyline generated with fewer points than `$minLength` is discarded. The default value of `$minLength` is 5.

`-b $beta`

Set the proportion parameter $\beta$ used in hysteresis thresholding of edge detection to `$beta`. The default value of `$beta` is 0.095.

`-r $ratio`

Set the ratio parameter $r$ used in hysteresis thresholding of edge detection to `$ratio`. The default value of `$ratio` is 0.4.

`-e $tolerance`

Set the tolerance $\varepsilon$ applied in DP polyline simplification algorithm to `$tolerance`. The default value of `$tolerance` is 1.

`-f $controlFlag`

Set the flag used in choosing the point selection method to `$controlFlag`. If the value of `$controlFlag` is 1, the program uses ED scheme to select sample points; otherwise, random selection method is applied. The default value of `$controlFlag` is 1.

`-w $dFlag`

Set the flag used in choosing the wedge value selection scheme to `$dFlag`. If the value of `$dFlag` is 1, the program uses MMSODD method to select wedge value; otherwise, fixed distance method is applied. The default value of `$dFlag` is 1.

`-s $ksize`

Set the size of the binomial filter used in MMSODD method of wedge value selection to `$ksize`. The default value of `$ksize` is 5.

`-x $exitMode`

1) if `$exitMode`= 0, then the program will print an error message to standard error and then exit with an exit status of 3 (by calling "exit(3)") if the sampling density cannot be satisfied exactly; the error message should be of the form (all on one line) "ERROR: target sampling density not achieved exactly (requested XXX but got YYY)".

2) if `$exitMode`= 1, then the program will still complete normally if the sampling density cannot be satisfied exactly, but the program should print a warning message of the form "WARNING: target sampling density not achieved exactly (requested XXX but got YYY)" to the standard error.

3) if `$exitMode`= 2, then the program will still complete normally if the sampling density cannot be satisfied exactly and no warning message is issued.

The default value for `$exitMode` is 0.

`-B $startUp`

1) if `$startUp`= 0, then the mirroring method will be applied to solve the start-up effect;

2) if `$startUp`= 1, then the forcing method will be applied to solve the start-up effect;

3) if `$startUp`= 2, then nothing will be done about the start-up effect.

The default value for `$startUp` is 0, that is, using the mirroring method to solve the start-up problem.

`-S $isSimple`

If `$isSimple`= 0, set the type of the input image to simple; otherwise, set the type of the input image to nonsimple. The default value for `$isSimple` is determined by the software automatically based on the content of the input image.

**Examples**

Suppose we want to generate an ERD model of the `lena` image at a sampling density of 4% with a particular manually chosen set of parameters using our proposed ERDED method, we can run the `erd_ed` command as follows:

```
erd_ed -i lena.pnm -d 4 -l 3 -b 0.17 -r 0.4  -o lena.erd
```

## The `erd_gpi` command

### Synopsis

```
erd_gpi [options]
```

### Description

The `erd_gpi` command is an implementation of our proposed ERDGPI method. Given an input image, and a desired number $K$ of sampling points or a desired sampling density $d$, the `erd_gpi` command uses the ERDGPI method to generate an ERD model of the input image.

### Options

The `erd_gpi` program accepts the following options:

`-i $inputFile`

: Read the input image from the file named `$inputFile` in the format of plain pnm.

`-d $sampleDensity`

: Set the sampling density to `$sampleDensity`(%). The default value is $3(\%)$.

`-n $num`

: Set the number of sampling points to `$num`. The default value is 7864. If the parameter`$sampleDensity` is set, `$num` will be ignored.

`-o $outputFile`

: Output the image model to the file named `$outputFile` in the format of `.erd`.

`-t $triangulationFile`

: Output the triangulation data to the file named `$triangulationFile` in the triangulation format used by iviewer software.

`-p $polylineFile`

: Output the polyline data to the file named `$polylineFile` in the polyline set format used by iviewer software.

`-l $minLength`

Set the minimum length of the polylines to be kept to `$minLength`. Any polyline generated with fewer points than `$minLength` is discarded. The default value of `$minLength` is 5.

`-b $beta`

Set the proportion parameter $\beta$ used in hysteresis thresholding of edge detection to `$beta`. The default value of `$beta` is 0.095.

`-r $ratio`

Set the ratio parameter $r$ used in hysteresis thresholding of edge detection to `$ratio`. The default value of `$ratio` is 0.4

`-e $tolerance`

Set the tolerance $\varepsilon$ applied in DP polyline simplification algorithm to `$tolerance`. The default value of `$tolerance` is 1.

`-w $dFlag`

Set the flag used in choosing the wedge value selection scheme to `$dFlag`. If `$dFlag`$=$ 1, the program uses the MMSODD method to select the wedge values; otherwise, the fixed distance method is applied. The default value of `$dFlag` is 1.

`-s $ksize`

Set the size of the binomial filter used in MMSODD method in wedge value selection to `$ksize`. The default value of `$ksize` is 5.

`-x $exitMode`

1) If `$exitMode`$=$ 0, then the program will print an error message to standard error and then exit with an exit status of 3 (by calling "exit(3)") if the sampling density cannot be satisfied exactly; the error message should be of the form (all on one line) "ERROR: target sampling density not achieved exactly (requested XXX but got YYY)".

2) If `$exitMode`$=$ 1, then the program will still complete normally if the sampling density cannot be satisfied exactly, but the program should print a warning message of the form "WARNING: target sampling density not achieved exactly (requested XXX but got YYY)" to standard error.

3) If `$exitMode`= 2, then the program will still complete normally if the sampling density cannot be satisfied exactly and no warning message is issued.

The default value of `$exitMode` is 0.

`-S $isSimple`

If `$isSimple`= 0, set the type of input image to simple; otherwise, set the type of input image to non-simple. The default value of `$isSimple` is determined by the software automatically based on the content of the input image.

**Examples**

Suppose we want to generate an ERD model of the `lena` image at a sampling density of 3% with a manually chosen set of parameters using our proposed ERDGPI method, we could run the `erd_ed` command as follows:

```
erd_gpi -i lena.pnm -d 3 -l 5 -e 1 -b 0.4 -o lena.erd
```

### A.5.2 Image reconstruction

In order to reconstruct an image approximation from the mesh model, triangle scan conversion is applied to scan convert each triangle in the meshes, interpolating the value of the points inside the triangles. Since there are two kinds of mesh models in our software, two commands, namely the `imgsyn` and `imgsyn_aa`, are used to reconstruct images from these two mesh models.

#### The `imgsyn` command

**Synopsis**

`imgsyn`

**Description**

The `imgsyn` command reads the basic model of an image (`.model` format) from the standard input and writes the reconstruct image in the format of plain pnm to the standard output.

**Example**

An image approximation is reconstructed from the simple model `lena.model` by running the following command:

```
imgsyn < lena.model > lena_reconstructed_model.pnm
```

### The `imgsyn_aa` command

**Synopsis**

```
imgsyn_aa
```

**Description**

The `imgsyn_aa` utilizes $4 \times 4$ super-sampling to reconstruct an image approximation from the ERD model(`.erd` format). The command reads ERD model from the standard input in `.erd` format and writes the reconstructed image approximation in plain pnm format to the standard output.

**Example**

To reconstruct an anti-aliased image approximation from the ERD model `lena.erd`, we run the commands as follows:

```
imgsyn_aa < lena.erd > lena_reconstructed_erd.pnm
```

## A.5.3 Quality evaluation

To evaluate the quality of the reconstructed images, the PSNR is calculated using `Cal_PSNR` command.

### The `Cal_PSNR` command

**Synopsis**

```
Cal_PSNR $original_image $reconstructed_image
```

**Description**

The `Cal_PSNR` takes two images of plain pnm format as input, the original image `$orignal_image` and the reconstructed image `$reconstructed_image`, and writes the PSNR of the reconstructed image to standard output.

**Example**

The PSNR of the reconstructed image `lena_reconstructed.pnm` can be obtained by running the command:

```
Cal_PSNR lena.pnm lena_reconstructed.pnm
```

## A.6 A Bug in CGAL

With regard to CGAL(version 3.5.1), a bug was found and fixed in the function:

```
template <class OutputItFaces, class OutputItBoudaryEdges>
std::pair<OutputItFaces, OutputItBoundaryEdges>
cdt.get_conflicts_and_boundary(Point p, OutputItFaces fit
OutputItBoundaryEdges eit, Face_handle start)
const,
```

The preceding function is a member function of the class

```
Constrained_Delaunay_triangulation_2<Traits, Tds, Itag>
```

The function is intended to query the set $F$ of faces and the set $E$ of boundary edges in conflict with a point $p$. In the constrained Delaunay triangulation, the constrained edges are considered as obstacles blocking the view from a point to the interior of a face. A point $p$ is said to be in conflict with a face $f$ if and only if $p$ is visible from the interior of $f$ and included in the circumcircle of $f$.

When the point $p$ is inserted, only the faces in $F$ are affected. For efficiency, in each iteration after $p$ is inserted, we only want to recalculate the absolute error $\varepsilon_p$ of the points of the faces in $F$. Therefore, this function `get_conflict_and_boundary` is extensively used and extremely important.

When the point $p$ inserted lies on a constrained edge, however, the set $F$ of faces and the set $E$ of boundary edges that are in conflict with $p$ are incorrectly calculated by the

function and only contain part of the desired sets. The function starts to search the face that has $p$ in its interior, then propagates to the faces incident to it in three directions until a constrained edge is met or the faces searched are no longer in conflict with $p$. The problem is that, when the point $p$ lies on a constrained edge, it searches the faces only on one side of the constrained edge, the faces and boundary edges on the other side of the constrained edge are missing. We found and fixed this bug, making $F$ and $E$ calculated by the function contain all the faces and boundary edges that are in conflict with the point $p$.

## A.7 Listing and Description of Source Files

The source code consists of a large number of files of various functionalities. To have a better understanding of our software, in what follows, all the source files of the software along with the corresponding functionalities are described.

`ywb.cpp`

The file contains the main function of the ED scheme.

`gpi.cpp`

The file contains the main function of the GPI scheme.

`erd_ed.cpp`

The file contains the main function of our proposed ERDED method.

`erd_gpi.cpp`

The file contains the main function of our proposed ERDGPI method.

`imgsyn.cpp`

The file contains the main function of reconstructing images from the basic model.

`imgsyn_aa.cpp`

The file contains the main function of reconstructing images from the ERD model, using $4 \times 4$ super-sampling.

`Cal_PSNR.cpp`

The file contains the main function of calculating PSNR to evaluate the quality of an image approximation.

`config.h`

A configuration file which defines the parameters used through out the programs.

`typedef_ip.h`

The file contains header file declaration and type definition of GPI scheme.

`typedef_myED.h`

The file contains header file declaration and type definition of our proposed ERDED method.

**`typedef`**`.h`

The file contains header file declaration and type definition of our proposed ERDGPI method.

`Array2.hpp`

The file contains the definition of a 2-D matrix class, `Array2`, which is used to represent images in our software.

`PriQue.hpp`

The file contains the definition of a class of a heap-based priority queue, which is used to extract the face with the maximum squared error along with its candidate point during greedy point insertion.

`WedgeTri.hpp`

The file contains the definition of a triangulation with wedges, which is derived from the constrained Delaunay triangulation in CGAL.

`PolylineG.hpp`

The file contains the definition of a class used for polyline generation.

`PolylineClass.hpp`

The file contains the definition of a class for polyline simplification using the Douglas-Peucker algorithm.

`incremental_ip.h`

The file contains the definition of a method which inserts points into the triangulation based on the error measurement obtained.

`scanface_ip.h`

The file contains the definition of a method which recalculates the absolute errors of the points in the faces that are affected by inserting a point.

`funcDeclaration.h`

The file contains the declaration of all the functions used in our software.

`imageProcessing.h`

The file contains the implementations of image processing algorithms, such as convolution, modified Canny edge detector, bilinear interpolation and so on.

`binomialFilter.h`

The file contains the method of generating a 2-D binomial filter with a given size.

`creatConstraint.h`

The file contains the definition of a method which generates constrained edges used in the constrained DT.

`geoModel.h`

The file contains the definition of a method which constructs a constrained DT on a set of points with a set of constrained edges, and produces the ERD model.

`Mgeneration_Yang.hpp`

The file contains the definition of a method which uses the ED scheme to select nonedge points with the mirroring method to solve the start-up effect.

`Mgeneration_Yang1.hpp`

The file contains the definition of a method which uses the ED scheme to select nonedge points with the forcing method to solve the start-up effect.

`Mgeneration_Yang2.hpp`

The file contains the definition of a method which uses the ED scheme to select nonedge points without anything to solve the start-up effect.

`random_selection.hpp`

The file contains the definition of a method which randomly selects nonedge points.

`scanFace.h`

The file contains the definition of a method which calculates the absolute error of each point in the face and the squared error of the face.

`scanTriangle.h`

The file contains the definition of a method which interpolates each point in the triangle during the greedy insertion process.

`secondOrderDerivative.h`

The file contains the definition of a method which calculates the maximum magnitude second-order directional-derivative of a given image.

`incremental.h`

The file contains the definition of a method which inserts a point into the triangulation in each iteration based on the error measurement until a certain number of points is reached.