Surface Reconstruction from Imperfect Point Models

A DISSERTATION PRESENTED

BY

JIANNING WANG

TO

THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

STONY BROOK UNIVERSITY

May 2007

Stony Brook University

The Graduate School

JIANNING WANG

We, the dissertation committee for the above candidate for

the Doctor of Philosophy degree,

hereby recommend acceptance of this dissertation.

Professor Arie E. Kaufman, Dissertation Advisor
Computer Science Department

Professor Hong Qin, Chairman of Defense
Computer Science Department

Professor Dimitris Samaras, Committee Member
Computer Science Department

Professor George Wolberg, External Committee Member
Computer Science Department
City College of New York (CUNY)

This dissertation is accepted by the Graduate School.

Dean of the Graduate School

Abstract of the Dissertation

Surface Reconstruction from Imperfect Point Models

by

JIANNING WANG

Doctor of Philosophy
in
Computer Science

Stony Brook University

2007

This thesis presents work on surface reconstruction from imperfect point models. Here "imperfections" refer to the cases such as missing information about point normals, unknown level of noise, irregular sampling, and/or non-manifold surfaces, resulting in incorrect reconstruction results using existing approaches. Due to the diversity of point acquisition approaches (laser range finders, computer vision methods, etc), imperfections occur often in practice and pose a great challenge to surface reconstruction algorithms. In this thesis, several novel algorithms, targeted towards different imperfections, are proposed. To recover the geometry and the texture of insufficiently sampled regions, we exploit user knowledge about the planarity and the symmetry of man-made objects in indoor scenes and fill the holes in a natural way. After that, there might still be some remaining holes. We then develop a hole filling algorithm as a post processing module to complete the resulting mesh. This algorithm interpolates the interior of a hole based on its vicinity. It can also be used as an independent module for mesh repairing. In addition, we propose a distance-field based algorithm to handle very noisy points without point normals. A global distance field

is constructed by propagating the information from interior/exterior regions in a hierarchical and adaptive way. A mesh is extracted later as the zero-set surface of the distance field. Furthermore, a generalized approach is introduced to naturally handle both manifold and non-manifold surfaces, including both non-orientable surfaces and surfaces with boundaries. The input points are first voxelized, enabling easy classification of the local shape for each point. We then locate non-manifold regions (junctions and boundaries) and take special care to mesh them. We also propose to enforce the regularity of the resulting mesh reconstructed from point models. Using a nearly isometric point parameterization, we are capable of controlling the shape of output triangles. The aforementioned algorithms cover a wide range of imperfections and we believe they could handle many problems encountered in practice.

To My Wife, Huan Ni

And To My Parents

With Love!

Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I am deeply grateful to my advisors, Professor Arie E. Kaufman and Professor Manuel M. Oliveira, for their inspiration for the research directions, and for their invaluable advices and countless efforts guiding me during my research. In addition, I want to express my sincere gratitude to Professor Hong Qin, Professor Dimitris Samaras, and Professor Klaus Mueller, who have provided insightful suggestions to my research and served on various committees associated with my graduate studies. I also thank Professor George Wolberg for taking the time to serve as the external member of my dissertation committee.

The members of Center for Visual Computing offer me tremendous help during my study and research. I would like to thank everyone for their inspiring ideas and selfless contribution. I want to thank Huamin Qu, Xiaoming Wei, Wei Li, Nan Zhang, Susan Frank, Kevin McDonnell, Ye Duan, Jing Hua, Hui Xie, and Wei Hong for their helpful suggestions. And many thanks to many of other members of the lab, especially Yang Wang, Haitao Zhang, Ye Zhao, Feng Qiu, and Zhe Fan. The members of the excellent support staff in the Computer Science Department were also of great assistance, especially Bin Zhang, Brian Tria, Stella Mannino, Betty Knittweis, Kathy Germana and Edwina Osmanski. I extend my gratitude to the faculty and students of the Computer Science department, who

over the years have helped me in many ways.

I also thank my parents for their understanding and constant support. I would not be here without them. Finally, and most importantly, I want to thank my girl friend, Huan Ni, for her endless love, cheerful encouragement, and perpetual support. She stands by me and supports me all the time. I would not be finishing my Ph.D study without her encouragement and inspiration. This dissertation is dedicated to my parents and Huan Ni.

Publications

1. J. Wang, and A. Kaufman. 3D Reconstruction from Endoscopic Videos. submitted.

2. J. Wang, H. Zhang, M. Oliveira, and A. Kaufman. Reconstructing Regular Meshes from Points. submitted to *the Visual Computer*.

3. J. Wang, and M. Oliveira. Filling Holes on Locally Smooth Surfaces Reconstructed from Point Clouds. *Image and Vision Computing*, Elsevier. 25, pp. 103-113, 2007.

4. J. Wang, M. Oliveira, and A. Kaufman. Reconstructing Manifold and Non-Manifold Surfaces from Point Clouds. *IEEE Visualization*. pp. 415-422, 2005.

5. J. Wang, M. Oliveira, H. Xie and A. Kaufman. Surface Reconstruction Using Oriented Charges. *Computer Graphics International*. pp. 122-128, 2005.

6. J. Wang, O. Hall-Holt, P. Konecny and A. Kaufman. Per-Pixel Camera Calibration for 3D Range Scanning. *SPIE Electronic Imaging*. 5665, pp. 342-352, 2005.

7. F. Qiu, Y. Zhao, Z. Fan, X. Wei, H. Lorenz, J. Wang, S. Yoakum-Stover, A. Kaufman and K. Mueller. Dispersion Simulation and Visualization For Urban Security. *IEEE Visualization*. pp. 553-560, 2004.

8. H. Xie, J. Wang, J. Hua, H. Qin and A. Kaufman. Piecewise C_1 Continuous Surface Reconstruction of Noisy Point Clouds via Local Implicit Quadric Regression. *IEEE Visualization*. pp. 91-98, 2003.

9. J. Wang and M. Oliveira. A Hole Filling Strategy for Reconstruction of Smooth Surfaces in Range Images. *XVI Brazilian Symposium on Computer Graphics and Image Processing*. pp. 11-18, 2003.

10. J. Wang and M. Oliveira. Improved Scene Reconstruction from Range Images. *Computer Graphics Forum (Eurographics)*, 21(3), pp. 521-530, 2002.

11. W. Corrêa, M. Oliveira, C. Silva and J. Wang. Modeling and Rendering of Real Environments. *RITA - Revista de Informtica Teórica e Aplicada*, 9(2), pp. 127-156, 2002.

Chapter 1

Introduction

There has been a considerable amount of research in the past decade on the development of techniques for surface reconstruction from points. However, only few attempts have been made to handle the encountered imperfections, which occur often in practice and pose a great challenge to surface reconstruction algorithms. Despite the recent development of scanning devices, these problems still remain. In this thesis, we present a set of novel surface reconstruction algorithms to handle them.

1.1 Problem Statement

Recently, point models have received a growing amount of attention in computer graphics. However, the polygonal mesh remains the most popular graphics primitive, with the widest support from existing hardware and software. Therefore, we need surface reconstruction algorithms to convert points to surface meshes.

The problem of surface reconstruction could be formalized as follows: given a point

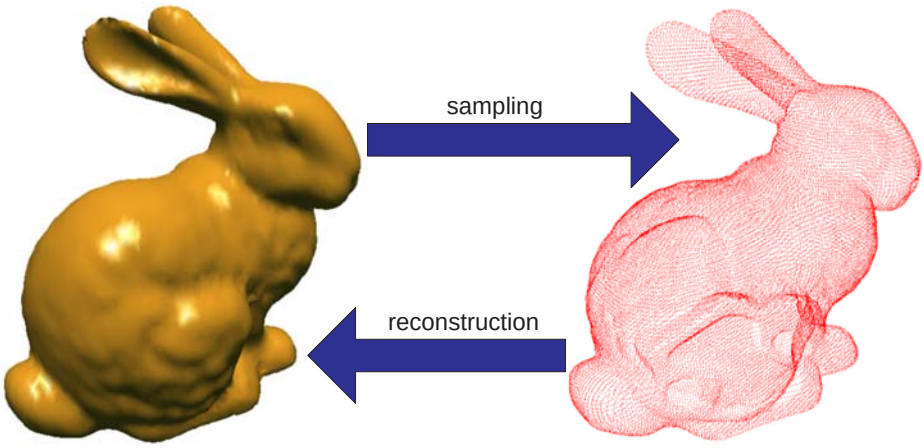


Figure 1.1: Sampling and reconstruction

set $\mathcal{P}$ sampled from a surface $\mathcal{S}$, compute a mesh $\mathcal{M}$ that approximates $\mathcal{S}$. Reconstruction is the reverse process of sampling (Figure 1.1). The sampling process (from $\mathcal{S}$ to $\mathcal{P}$) entails information loss if the sampling rate is below the Nyquist frequency [95]. In practice, the samples might also contain noise and outliers. At the same time, a sample point could be equipped with very limited information (*e.g.*, only its location). Therefore, surface reconstruction is an ill-posed problem and several assumptions are usually required to enforce proper reconstruction. Common assumptions include: (1) dense sampling; (2) surface regularity (smoothness); and (3) no non-manifold regions.

Each reconstruction algorithm has its limitations. These reconstruction algorithms often make assumptions on the input points and the surface to be reconstructed. With less information stored in the input points and/or more requirements about the resulting mesh, it becomes more difficult to design a reconstruction algorithm. These difficulties are termed *imperfections* and pose a great challenge to surface reconstruction algorithms. These imperfections are often encountered in practice and should be tackled with a specific emphasis. Although a number of efforts have been taken to handle the problems, we could not find a systematic one to push reconstruction algorithms to their limits.

Our research is targeted towards designing novel reconstruction algorithms suitable for weaker assumptions (more defect tolerant) about input points and stronger requirements for the resulting mesh. In this thesis, we consider the following defects of the input point clouds:

1. **Noisy points:** The location of points may be perturbed by unknown levels of noise.
2. **Missing point normals:** The points may not be equipped with point normals, which indicate the orientation of the local shape.
3. **Irregular, incomplete sampling:** The surface may not be well-sampled. This often leads to holes, which are to be filled additional efforts.

Additional difficulties for surface reconstruction algorithms follow from the further requirements for the resulting mesh:

1. **Holes:** Holes, resulting from the insufficiently sampled regions, might need to be filled.
2. **Boundaries:** The boundary of the surface might need to be preserved.
3. **Non-manifold surface:** A non-manifold surface includes surface junctions or surface boundaries, which can not be reconstructed correctly by traditional surface reconstruction algorithms.
4. **Fair mesh:** The triangles in the resulting mesh should be well-shaped. It is required by some follow-up operations on the mesh, such as Finite Element Analysis.
5. **Sharp features:** Sharp edges (creases) and corners should be preserved, even though they break the assumption of smoothness.

1.2 Contributions

In this thesis, we present a number of techniques and algorithms in an attempt to partially solve the aforementioned problems in surface reconstruction. Specifically, our contributions can be summarized as follows:

1. A novel algorithm to recover missing regions in indoor scenes. User knowledge such as the planarity and the symmetry of man-made objects are exploited to fill holes. Textures are repaired using a similar technique [134].

2. A post processing technique to fill holes in the reconstructed mesh. By interpolating/approximating the vicinity of holes using moving least squares, we are able to recover the geometry and texture of the holes [132].

3. A new framework to compute the global distance field from points without point normals and reconstruct the surface. After propagating the information about the interior and exterior in an octree, a hierarchical and adaptive distance field is created. The mesh is later extracted from the distance field. This approach works for very noisy points [137].

4. A generalized framework to reconstruct manifold and non-manifold surfaces. In addition, we are able to reconstruct both closed surfaces and surfaces with boundaries. By transforming the input points to voxels, local shape is determined more easily than in the continuous space. Then, surface junctions and boundaries could be identified and reconstructed [136].

5. A novel algorithm to reconstruct regular meshes from point models. It is important for some applications, such as Finite Element Analysis, to have an input mesh with well-shaped triangles. We first obtain a natural parameter map for the input points.

With special handling along the cuts between parameter patches, we create a mesh of triangles with nearly the same shape and size.

THE ABOVE CONTRIBUTIONS ARE ALSO SUMMARIZED IN FIGURE 1.2, WHERE THE DIFFERENT KINDS OF IMPERFECTIONS ARE HANDLE BY THE PROPOSED ALGORITHMS. FOR GENERAL PURPOSE USE, THE FOLLOWING TWO ALGORITHMS MIGHT BE SUFFICIENT FOR SURFACE RECONSTRUCTION IN MOST CASES. IF THE INPUT DATA SET IS WELL SAMPLED AND HAS WITH LITTLE NOISE, WE RECOMMEND THE USER TO USE THE BALL-PIVOTING ALGORITHM [13] FOR FAST AND ROBUST RECONSTRUCTION. IF THE POINT DATASET IS EQUIPPED WITH POINT NORMALS AND REPRESENTS A CLOSED SURFACE, USER MIGHT USE THE MULTI-LEVEL PARTITION OF UNITY APPROACH [91]. FOR IMPERFECT POINT MODELS, WE HAVE PROPOSED A NUMBER OF ALGORITHMS, AS AN INTEGRATED ATTEMPT TO AT LEAST PARTIALLY SOLVE THE PROBLEMS. THEY SHARE THE CAPABILITY TO HANDLE THE SAME SET OF IMPERFECTIONS AND THEY ALL REQUIRE NO INFORMATION ABOUT POINT NORMALS. HOWEVER, THE PURPOSE AND THE FACED CHALLENGES OF THESE ALGORITHM ARE DIFFERENT.

THE ALGORITHM FOR RECOVERING MISSING REGIONS ONLY WORKS FOR INDOOR SCENES. IT REQUIRES THE SCENE TO INCLUDE SEVERAL LARGE PLANAR REGIONS AND/OR OBJECTS WITH BILATERAL SYMMETRY TO RECOVER THE MISSING INFORMATION. AND THE USER NEEDS TO HAVE THESE KNOWLEDGE ABOUT THE SCENE. IN ADDITION, THE SAMPLED SURFACE COULD BE NON-CLOSED ONE. THE HOLE-FILL ALGORITHM WORKS NOT ON POINT CLOUDS, BUT RECONSTRUCTED MESHES. THIS IS INDEED AN POST-PROCESSING MODULE TO FILL THE HOLE REGIONS WITH A SMOOTH SURFACE. THE NEW FRAMEWORK OF COMPUTING GLOBAL DISTANCE FIELDS WORKS ONLY ON POINT CLOUDS REPRESENTING CLOSED-SURFACES. HOWEVER, IT HAS THE

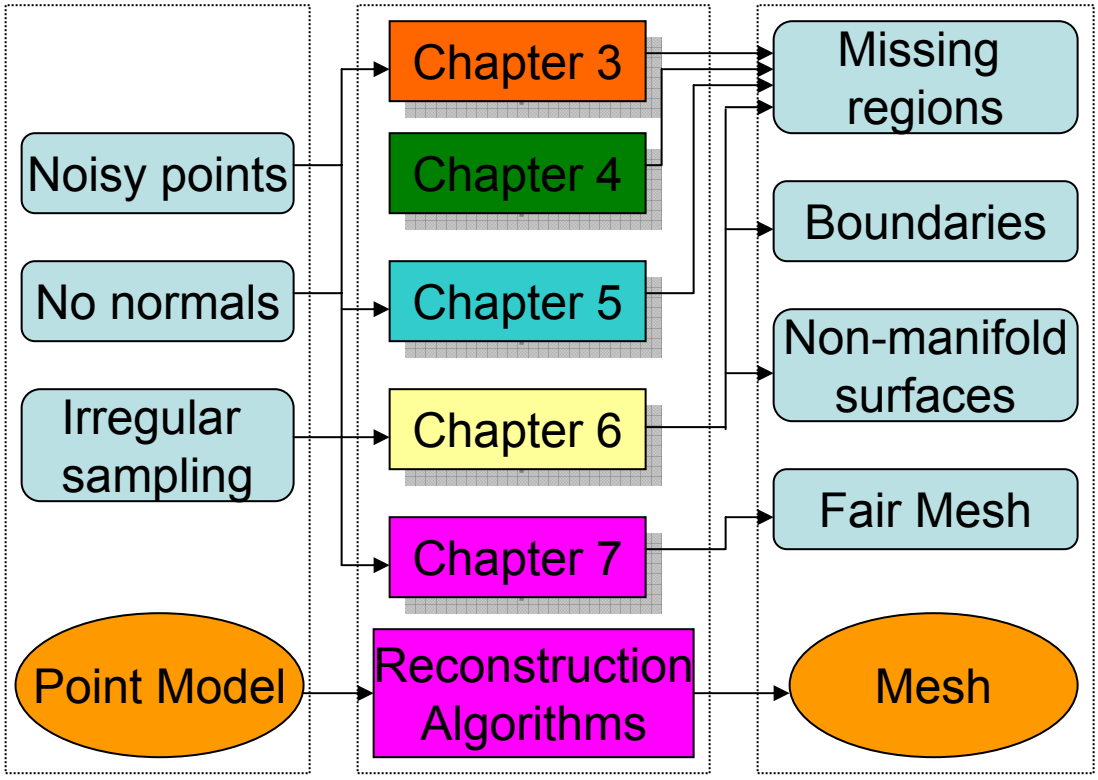


Figure 1.2: The imperfections handled by different algorithms.

ABILITY TO HANDLE VERY NOISY DATASET AND FILL SMALL HOLES. THE FRAMEWORK TO RECONSTRUCT MANIFOLD AND NON-MANIFOLD SURFACES IS ALSO A GENERAL FRAMEWORK FOR NON-CLOSED SURFACES AND CLOSED SURFACES. IT CAN PRESERVE THE BOUNDARY OF NON-CLOSED SURFACES AS WELL. THE ALGORITHM GENERATING REGULAR MESHES COULD ALSO WORK ON THE NON-CLOSED SURFACES. HOWEVER, THE POINT PARAMETERIZATION EMBEDDED IN THE ALGORITHM IS A LITTLE SENSITIVE TO NOISE AND VARYING SAMPLING RATES.

USER SHOULD SELECT THE MOST APPROPRIATE ALGORITHM FOR HIS RECONSTRUCTION BASED ON HOW MUCH INFORMATION IS ASSOCIATED WITH THE INPUT DATA, WHAT ASSUMPTION HE COULD MAKE ABOUT THE MISSING REGION, AND WHAT IS REQUIRED FOR THE RESULTING MESH.

1.3 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 gives an overview of the previous work related to point acquisition and surface reconstruction. Chapter 3 details the reconstruction algorithm to recover missing regions in indoor scenes that takes advantage of user knowledge. Chapter 4 describes the hole filling algorithm as a post-processing. In Chapter 5, we discuss the hierarchical and adaptive framework to compute the global distance field and reconstruct the surface. Chapter 6 describes the generalized framework for surface reconstruction of both manifold and non-manifold surfaces. Reconstructing regular mesh is discussed in Chapter 7. We conclude and propose the plan for the future work in Chapter 8.

Chapter 2

Background

In recent years, there has been a growing interest in digitizing the shape of real 3D objects [66, 70]. The popular pipeline from the real object to its digital replica is shown in Figure 2.1. There exist a number of methods to capture samples of objects [60, 77, 96, 105]. The problem of reconstructing a continuous function from a point dataset in R^n has been extensively studied over the past decades [41, 42, 52]. In recent years, the problem of surface reconstruction in 3D has attracted the attention of several researchers [7, 26, 85, 91, 129, 151]. After reconstruction, the resulting mesh may have several holes and there are several algorithms to fill these holes [30, 31].

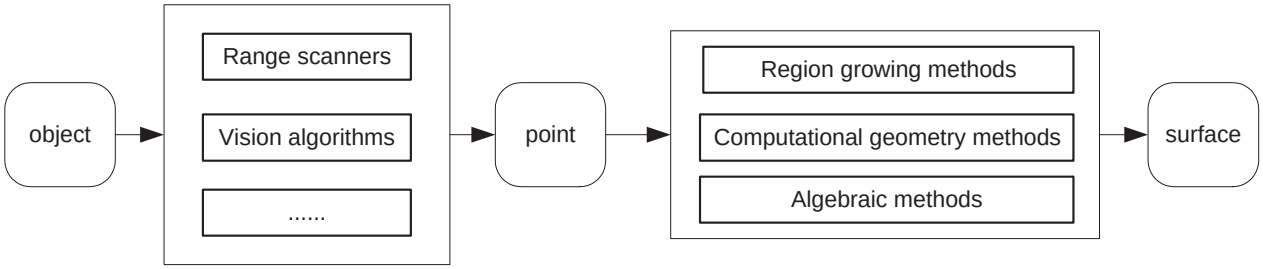


Figure 2.1: The common pipeline for making digital replicas of objects

2.1 Point Model Acquisition

The interest in making replicas of real objects has stimulated the development of different range scanning techniques. Kanade et al. develop a stereo machine [60] to reconstruct the depth value for each pixel. Matusik et al. [77] use object silhouettes on the image to carve 3D space. With different views, they obtain in the end the 3D shape of the object. Proesmans and Cool [105] use a structured light system to extract shape and texture of an object from a single image. We [133] propose an algorithm to calibrate a structured light scanning system, namely, to calibrate the camera and the projector of the system. This approach handles all types of distortions and produces results in high accuracy using a non-parametric model. Palojärvi et al. [96] devise a laser radar and use the time of flight for the depth.

In the computer graphics community, a number of approaches are proposed to obtain the point samples of an object in an accurate and complete manner. The Digital michelangelo project [70] combines the use of laser triangulation rangefinders, laser time-of-flight rangefinders, and digital cameras to acquire the point samples. The 3D model acquisition pipeline proposed by Bernadini et al. [14] studies how to integrate texture and normal maps with the geometric data. Recently, Nehab et al. [86] enhance the estimation of the underlying 3D surface by combining point positions and normals from different sources. The proposed linear algorithm treats high- and low-frequency components separately to recover details and to remove bias. The common challenge of above algorithms is that an individual range scan only covers a small region. As a consequence, multiple range scans need to be registered in a common framework. This is often done using the iterative closest

POINTS ALGORITHMS. ONE EXAMPLE IS THE REAL-TIME MODEL ACQUISITION APPROACH, PROPOSED BY RUSINKIEWICZ ET AL. [107]. AFTER REGISTRATION, THE INCONSISTENCE BROUGHT ABOUT DIFFERENT RANGE SCANS WILL BE REMOVED IN AN INTEGRATION STAGE. FOR THIS PURPOSE, TURK AND LEVOY [128] PROPOSE A ZIPPER ALGORITHM TO MERGE RANGE IMAGES AND CURLESS ET AL. [30] ADOPT A VOLUMETRIC APPROACH.

In computer vision, shape from X techniques could be used to reconstruct 3D shapes as well. Shape from motion algorithms [126] are used to compute the motion of the camera and recover the shape of the object from a video sequence. Shape from shading algorithms [150] take a single image as input. They use the variation of color/intensity to induce the shape of the underlying surface. However, shape from shading algorithms have an inherent ambiguity for concavity and convexity. To resolve that, Prados [102] assumes that the light source is at the optical center.

2.2 Surface Reconstruction

In general, current surface reconstruction methods can be grouped into three different categories: (1) region growing methods, (2) computational geometry methods, and (3) algebraic methods. Here, we only present some of the most popular approaches. Note some of these methods fail in the presence of noisy data or incomplete information, while others are only computationally suitable for simple models.

2.2.1 Region Growing Methods

Region growing methods propagate information and thus reconstruct surfaces in a progressive style. Hoppe [52] fits a plane to a neighborhood around each data point, thus providing an estimate of the surface normal for the point. He then constructs a graph that

connects neighboring points using arcs weighted by the similarity between the surface normals. The surface normals are propagated by traversing the graph as a minimal spanning tree. Lee and Medioni's tensor voting method [65] is similar in that neighboring points are used to estimate the orientations of data points. The tensor is the covariance matrix of the normal vectors of a neighborhood of points. Each data point votes for the orientation of other points in its neighborhood using its tensor field. Tang and Medioni [120] reconstruct the surface by growing planar, edge, and point features until they encounter neighboring features. Both methods described above are sensitive to noise because they rely on good estimates for the normal vector at each data point. Bernardini's Ball-Pivoting algorithm [13] grows a mesh from an initial seed triangle that is correctly oriented. A ball of specified radius is pivoted across edges of each triangle bounding the growing mesh. If the pivoted ball hits vertices that are not yet part of the mesh, a new triangle is instantiated and added to the growing mesh. Non-manifold constructions are avoided in the above process.

SHARF ET AL. [115] DEPLOY AN EXPLICIT DEFORMABLE MESH REPRESENTATION COMPOSED OF MULTIPLE COMPETING EVOLVING FRONTS. THESE FRONTS CONFORM TO THE FEATURES IN A COARSE-TO-FINE MANNER. WITH TOPOLOGICAL CHANGES MONITORED, THIS APPROACH COULD RESULT IN A WATER-TIGHT RECONSTRUCTION FOR VARYING SAMPLING AND AND MISSING DATA.

RECENTLY, LEVEL SET IS ALSO ADOPTED FOR SURFACE RECONSTRUCTION. NILSSON ET AL. [88] USE THE EULERIAN FORMULATION AUGMENTED WITH LAGRANGIAN PARTICLES TO PROPAGATING THE CONTOURS AS 2D LEVEL SETS. ZHAO ET AL. [151] PROPOSE FORMULATIONS FOR A WEIGHTED MINIMAL SURFACE MODEL. THIS LEVEL SET APPROACH HANDLES COMPLICATED TOPOLOGY AND NOISY DATA.

2.2.2 Computational Geometry Methods

Computational geometry methods depend on mechanisms, such as Delaunay triangulation and Voronoi diagram [7, 35]. Such methods interpolate the original points and basically they are sensitive to the presence of noise.

Several algorithms based on computational geometry construct a collection of simplexes that form the shape or surface from a set of unorganized points. These methods exactly interpolate the data and the vertices of the simplexes consist of the given data points. A consequence is that noise and aliasing in the data become embedded in the reconstructed surface. Of such methods, two most successful are Alpha Shapes [35] and the Crust algorithm [7]. In Alpha shapes, the shape is carved out by removing simplexes of the Delaunay triangulation of the point set. A simplex is removed if its circumscribing sphere is larger than the alpha ball. In the Crust algorithm, Delaunay triangulation is performed on the original set of points along with Voronoi vertices that approximate the medial axis of the shape. The resulting triangulation distinguishes triangles that are part of the object surface from those that are on the interior because interior triangles have a Voronoi vertex as one of their vertices. Both the Alpha Shapes and Crust algorithms need no other information than the locations of the data points and perform well on dense and precise data sets. The object model that these approaches generate, however, consists of simplexes which occur close to the surface. The collection of simplexes is not a manifold surface, and extraction of such a surface is a non-trivial post-processing task.

AMENTA ET AL. [8] SELECT CANDIDATE TRIANGLES FOR THE RECONSTRUCTED SURFACE USING *co-cones*. A MANIFOLD SURFACE IS EXTRACTED LATER BY A "PROVABLY CORRECT" COMBINATIONAL ALGORITHM. THE POWER CRUST ALGORITHM [9] FIRST EXTRACT THE MEDIAL AXIS FROM THE POINT CLOUDS AND THEN USE AN INVERSE MEDIAL AXIS TRANSFORM TO PRODUCE THE RECONSTRUCTED SURFACE. DEY

ET AL. [33] ADOPT A DIVIDE-AND-CONQUER IDEA TO MAKE THE DELAUNAY TRIANGULATION APPROACH SUITABLE FOR RECONSTRUCTING MODELS WITH MILLIONS OF POINTS.

2.2.3 Algebraic Methods

Algebraic methods try to fit a function to the data points. They avoid creating noisy surfaces by fitting a smooth function, and by not requiring that the function pass through all data points. The reconstructed surface may consist of a single global function or many local functions, which are pieced together. Two examples of reconstruction by global algebraic fitting are the works of Taubin [121, 122], and Gotsman and Keren [63]. Taubin fits a polynomial implicit function to a point set by minimizing the distance between the point set and the implicit surface. He points out that calculating Euclidean distances for implicit functions requires an iterative process because implicit functions are not often Euclidean distance functions. Taubin develops a first order approximation of the Euclidean distance [121] and improves the approximation [122]. Gotsman and Keren create parameterized families of polynomials that satisfy desirable properties, such as fitness to the data or continuity preservation. Such a family must be large so that it can include as many functions as possible. This technique leads to an over-representation of the subset, in that the resulting polynomial will often have more coefficients for which to solve than the simpler polynomials included in the subset, thus requiring additional computation. The primary limitation of global algebraic methods is their inability to reconstruct complex models. These methods become too computationally expensive for the high degree polynomials that are necessary to represent complex objects.

Bajaj [12] overcomes the complexity limitation by constructing piecewise polynomial

patches (called A-patches) that combine to form one surface. Bajaj uses Delaunay triangulation to divide the point set into groups delineated by tetrahedrons. An A-patch is formed by fitting a Bernstein polynomial to the data points within each tetrahedron. By constructing a piecewise surface, Bajaj's approach loses the compact characteristic of a global representation, and operations such as collision detection, morphing, blending, and modeling with constructive solid geometry become more difficult to perform since the representation is no longer a single analytical function. Examples of algebraic methods developed earlier in the vision community that provide both smooth global fitting and accurate local refinement include the works of Terzopoulos and Metaxas on deformable superquadrics [124] and Pentland and Sclaroff on generalized implicit functions [100, 113]. Both methods use superquadric ellipsoids as the global shape and add local deformations to fit the data points. Terzopoulos and Metaxas separate the reconstructed model into global parameters defined by the superquadric coefficients, and local displacements defined as a linear combination of basis functions. The global and local deformation parameters are solved using dynamics. Pentland and Sclaroff define a generalized implicit model that consists of a superquadric ellipsoid, a modal deformation matrix that acts on the ellipsoid, and a displacement map that pushes the implicit surface along the surface normal towards data points. The modal deformation parameters are found by iteratively finding the minimum RMS error to the data points. The residual error after the deformation parameters have been found are incorporated into a displacement map that may exactly interpolate or just approximate the data. As with most of the algebraic methods, the primary drawback of these techniques is their inability to handle arbitrary topologies. Complex models are constructed by combining multiple superquadrics. Terzopoulos and Metaxas' example of reconstructing a humanoid doll consists of separate deformable superquadrics for the torso, head, arms, and legs.

Dinh [34] proposes a reconstruction approach based on a summation of non-polynomial

basis functions whose domain is a scalar value obtained from the distance between sample points. Surface regularization restricts the class of permissible surfaces to those which minimize a selected energy functional. The work of Boult and Kender [24] and the work of Terzopoulos [123] are examples of regularization applied to height-field surfaces, and [37] is an example of regularization applied to parametric curves. Terzopoulos [123] pioneers finite-differencing techniques to compute approximate derivatives used in minimizing the thin-plate energy functional of a height-field. He develops computational molecules from the discrete formulations of the partial derivatives. Regularization is performed by iterating between coarse and fine levels in a multi-resolution hierarchy. Boult and Kender [24] compare classes of permissible functions and discuss the use of basis functions to minimize the energy functional associated with each class. Using synthetic data, they show examples of overshooting surfaces that are often encountered in surface regularization. As exemplified by these two methods, many approaches based on surface regularization are restricted to height fields because surface derivatives are required in the process of regularization. Derivatives with respect to the major axis are naturally defined for height fields.

Fang and Gossard [37] reconstruct piecewise continuous parametric curves. The advantage of parametric curves and surfaces over height fields is the ability to represent closed curves and surfaces. Each curve in their piecewise reconstruction minimizes a combination of first, second, and third order energies. Unlike the examples above, the derivative of the curve in this method is evaluated with respect to the parametric variable. Each curve is formulated as a summation of weighted basis functions. Fang and Gossard show examples using Hermite basis. Dinh [34] uses basis functions to reconstruct a closed surface which minimizes a combination of first, second, and third order energies. She reconstructs complex 3D objects using a single implicit function based on volumetric regularization. By fitting smooth functions to the input points, algebraic methods [124] handle noisy datasets, but cannot deal with arbitrary topologies.

It is worthy pointing out that most popular implicit function based methods [26, 85, 91, 129] belong to this category as well. Essentially, these methods are based on the use of a signed distance function and can naturally handle different topologies. They lift the reconstruction problem to a higher dimension and then the resulting surface becomes the zero-set surface of the distance field. These methods require a way to distinguish between the inside and outside of (closed) surfaces, which is usually achieved by using information of surface normals at the samples. Turk and O'Brien [129] build signed distance functions using non-compact radial basis functions (RBF). Carr [26] accelerates the computation of the global distance field using Fast Multipole Methods. Morse [85] use compactly-supported RBFs based on the work by Wendland [140] to achieve local control and improved memory and computational time requirements. Ohtake et al. [91] uses a method based on the partition of unity to achieve similar results. They adaptively fit piecewise quadrics to the local shape and piece them together to form a global distance field. Carr et al. [27] handle surface reconstruction from noisy datasets by fitting an RBF and then changing the basis function during the evaluation. As described by Carr et al. [26], the method requires normals for defining the interior and exterior of the surfaces.

KANZHDAN [61] ADOPTS STOKES' THEOREM TO COMPUTE THE CHARACTERISTIC FUNCTION OF THE SOLID MODEL. FIRST THE FOURIER COEFFICIENTS OF THE CHARACTERISTIC FUNCTION ARE COMPUTED. THEN THE INVERSE FOURIER TRANSFORM ARE COMPUTED AND THE SURFACE IS EXTRACTED AS THE ISO-SURFACE. SIMILAR TO THE RADIAL BASIS FUNCTION APPROACH [26], KAZHDAN ET AL. [62] FIT THE ORIENTED POINTS USING A POISSON SYSTEM, WHICH ALLOWS FOR LOCALLY SUPPORTED BASIS FUNCTIONS. IN CONSEQUENCE, THE RECONSTRUCTION RESULT TURNS OUT TO THE SOLUTION OF A LINEAR SYSTEM.

ALEXA ET AL. [3] PROJECT THE POINTS ONTO A SURFACE DEFINED BY A MOVING LEAST SQUARES SCHEME. THIS APPROACH PROVIDES A MESHLESS APPROXIMATION

OF THE UNDERLYING SURFACE AND ALLOWS FOR UNSAMPLING AND DOWNSAMPLING WITH EASE. FLEISHMAN ET AL. [38] EXTEND THIS APPROACH BY CONSTRUCTING A HIERARCHY OF POINTS WHICH COULD EFFICIENTLY REPRESENT THE SHAPE IN DIFFERENT VIEWING CONDITIONS. AMENTA ET AL. [10] GIVE OUT A NEW EXPLICIT DEFINITION IN TERMS OF THE CRITICAL POINTS OF AN ENERGY FIELD. CONSEQUENTLY, A MAXIMAL SURFACE WILL BE EXTRACTED.

AN IMPORTANT PROBLEM OF ALGEBRAIC METHODS IS HOW TO PRESERVE SHARP FEATURES DUE TO THE FACT THAT THE FITTED FUNCTION IS SMOOTH IN MOST CASES. OHTAKE ET AL. [91] FIND OUT THE SHARP FEATURES AS REGIONS CONTRADICTING NORMALS. THEY FIT THESE REGIONS WITH TWO OR MORE QUADRICS TO REPRESENT THE ABRUPT TRANSITION OF THE DISTANCE FIELD. FLEISHMAN ET AL. [39] USE A FORWARD-SEARCH APPROACH TO FIND THE SMOOTH NEIGHBORHOOD FOR MOVING LEAST SQUARES COMPUTATION. THE SHARP FEATURES THEN BECOMES THE OUTLIERS IDENTIFIED BY THE FORWARD SEARCH APPROACH.

JENKE ET AL. [54] SEARCH FOR A LOCAL MAXIMUM OF POSTERIOR PROBABILITY IN THE RECONSTRUCTION SPACE, PARAMETERIZED AS POINT CLOUDS. THIS APPROACH IS ROBUST TO THE PRESENCE OF NOISE AND IS CAPABLE OF HANDLING SHARP FEATURES.

FOR DISTANCE FIELD BASED METHODS, IT IS OFTEN IMPORTANT TO DISTINGUISH THE INSIDE/OUTSIDE OF THE OBJECT. MELLO ET AL. [80] PROPOSE AN APPROACH TO ESTIMATE THE IN/OUT FUNCTION OF THE SURFACE REPRESENTED BY A POINT MODEL. XIE ET AL. [143] FIND THE IN/OUT INFORMATION FOR PIECEWISE SURFACE PATCHES, AND REGISTER THESE INFORMATION VIA A VOTING PROCESS.

2.3 Hole Filling

The most popular approach for hole filling is also based on the use of implicit functions [26, 30, 31, 34, 85, 91, 110, 129, 144]. Curless's and Levoy's VRIP algorithm [30] is a good example. It uses an implicit method for surface reconstruction and hole filling. The approach consists of computing signed distance functions from a set of aligned meshes obtained from range scans. These functions are then blended and the final surface is extracted using the marching cubes algorithm [73]. In order to perform hole filling, the algorithm requires information about the line of sight of the scanner, and tends to perform poorly if the available information does not appropriately cover the entire volume enclosing the object. According to Davis et al. [31], this method may reconstruct surfaces that look less plausible than a smooth interpolation of the observed surfaces.

Davis et al. [31] use a volumetric diffusion approach, analogous to inpainting techniques [15, 93], to fill holes in range scans. This technique is targeted toward the reconstruction of densely sampled closed surfaces. The process consists of converting a surface into a voxel-based representation with a clamped signed distance function. The diffusion algorithm consists of alternating steps of blurring and compositing, after which the final surface is extracted using marching cubes [73]. This technique performs hole filling after surface reconstruction and the processing is constrained to the neighborhood of the holes. This algorithm is based on the use of an implicit function and can handle holes with more complex topology and twisted geometry. Moreover, the reconstructed patch is not guaranteed to smoothly blend with the rest of the surface, and, like in the algorithm of Curless et al. [30], may look little plausible.

Verdera et al. [131] use an approach similar to the one described by Davis et al. [31]. They turn a mesh into an implicit representation, then use a PDE system to inpaint the

missing regions. Savchenko and Kojekine [109] deploy a space mapping approach to extend existing surface boundaries and fill the gaps using RBFs. Clarenz et al. [29] minimize the Willmore energy to ensure continuity of the normal field using PDEs.

Liepa [71] presents an algorithm targeted toward filling holes in oriented connected manifold meshes. It fills holes by first identifying hole boundaries, triangulating the holes, and finally smoothing the resulting patches. The triangulation is based on an $O(n^3)$ algorithm that produces a minimum area triangulation, thus requiring the resulting patches to be smoothed during the last stage of the algorithm.

More recently, Ju [57] introduces a volumetric algorithm that takes a polygonal mesh and creates a closed surface. It starts by scan converting the mesh into voxels, where holes are filled by patching boundary cycles using minimal surfaces. This technique is appropriate for filling very small holes or larger holes in locally flat surfaces.

HOLES COULD ALSO BE FILLED DIRECTLY ON THE POINT MODELS BEFORE SURFACE SURFACE. Sharf et al. [114] reconstruct fine details in missing regions on geometric complex surfaces represented by point clouds. It uses a coarse to fine approach based on an octree to copy and paste samples from a set of example regions (at the same level of detail) to the missing ones. The technique can produce some impressive results, but in the absence of appropriate examples, or for noisy or poorly sampled surfaces, the algorithm tends to produce poor results. PARK ET AL. [97] DIRECTLY FIND THE HOLES FROM THE POINT CLOUDS. THEN THE SIMILARITY BETWEEN THE LOCAL SURFACE PATCHES IS COMPUTED USING THEIR LOCAL PARAMETERIZATION AND THE CURVATURE. WHEN A CANDIDATE PATCH IS SELECTED FOR A HOLE REGION, IT IS WARPED AND GLUED TO THE HOLE BY SOLVING A POISSON EQUATION IN 2D.

Chapter 3

Reconstructing Missing Regions of Indoor Scenes

(a) (b)

Figure 3.1: View of the reading room: (a) before reconstruction; and (b) after reconstruction by our system.

The availability of detailed geometric models is a critical factor for achieving realism in most computer graphics applications. Despite the great progress observed in rendering

techniques during last two decades, the process of modeling 3D objects and scenes has experienced significant fewer advances in the same period. In the past few years, we have observed an increasing demand for faithful representations of real scenes, primarily driven by applications whose goal is to create extremely realistic experiences by building virtual replicas of real environments. Potential uses of this technology include telepresence, entertainment, training and simulation, analysis of forensic records, remote walkthroughs and 3D TV, and have attracted the attention of several researchers [4, 59, 90, 106].

Creating models of real scenes is, however, a complex task for which the use of traditional modeling techniques is inappropriate. Aiming to simplify the modeling and rendering tasks, several image-based techniques have been proposed in recent years [28, 78, 79, 117]. Among these, the combined use of laser range finders [16] and color images appears as one the most promising approaches due to its relative independence of the sampled geometry and short acquisition time [78, 90]. Renderings of scenes modeled with such a technique can potentially exhibit an unprecedented degree of photorealism. Unfortunately, in many situations, it is not possible or practical to guarantee appropriate sampling of all surfaces in the scene. For example, occlusions and accessibility limitations to certain regions of the scene may cause some areas not to be sampled, resulting in incomplete or incorrectly reconstructed models. The occurrence of such areas is a major source of artifacts, usually appearing as "holes" in the rendered images. Figure 3.1 illustrates this situation for a partial model of the UNC Reading Room acquired from a single laser scan. Note the "holes"are exphasized by the blue regions in the figure. Creating high-quality mesh representations for each object in the scene from incomplete data remains a challenging task [148].

This chapter describes a pipeline for improving the reconstruction of scenes represented as sets of range images and introduces new algorithms for exploiting the use of symmetry

for reconstruction. The pipeline consists of a segmentation step followed by the reconstruction of missing geometric and textural information for individual objects (Figure 3.2). To achieve our goal, we take advantage of two simple but powerful observations: (1) real indoor scenes usually contain a number of large planar surfaces; (2) symmetry pervades both human-created and natural environments [48, 72, 138, 141]. We use these observations to design new algorithms and tools that greatly simplify the reconstruction task. Texture reconstruction from samples of the original texture is performed using a synthesis algorithm [36] and by exploring symmetry in periodic patterns.

Due to the segmentation of individual objects, our system also allows the user to edit the scene. We illustrate its use for reconstruction of both synthetic and real scenes. The resulting models are shown to be significantly better than the ones obtained by simply meshing the original range data.

The main contributions of this chapter include:

- A pipeline for scene reconstruction from range images;
- New algorithms for identifying approximate bilateral and rotational symmetric patterns in point clouds. These algorithms are relatively robust to noise and incomplete data;
- New algorithms for automatic identification of regions whose sampling rate/reconstruction can be improved by exploiting redundancies present in incomplete symmetric objects;
- New algorithms for performing/improving reconstruction of the areas identified above;
- A new algorithm for reconstruction of incomplete textures presenting complex symmetric patterns.

3.1 Related Work

Our effort to improve reconstruction of scene models from range images benefits from previous work in several related areas, including registration and segmentation of range images [18, 89], surface reconstruction from point clouds [7, 12, 35, 43, 52] and texture synthesis [36, 139].

Several algorithms for range image registration and for registration of range and color images [76, 87, 148] have been published in recent years. We rely on these results and assume that the input to our pipeline consists of a registered set of range and color images.

Nyland's registration algorithm [89] uses a 3D Hough transform to identify large planar areas in different datasets, which are then used to constrain the registration process. In our pipeline, a 3D Hough transform is also used to identify large planar areas, but the goal here is to replace them with texture-mapped polygons. This has three important benefits: (1) it can significantly reduce the number of points in the later stages of the pipeline; (2) reconstruction of planar areas is straightforward; (3) it facilitates de identification of clusters among the remaining points.

Curless [30] uses a hole filling technique to interpolate non-sampled surfaces in concave regions of objects. In this case, however, the added surfaces have little or no impact on the appearance of the objects and were intended to produce "watertight" models for reproduction using rapid prototyping techniques [30]. In our work, we try to derive information to fill in missing geometry and texture that would, otherwise, result in major artifacts.

A lot of work has been done in segmentation of range images [17, 49, 51, 67], but even in the simplest case of planar range image segmentation, current automatic procedures often fail to produce the desired results [51]. Due to ambiguities, segmentation usually requires some knowledge about the scene, and achieving good results often requires user intervention[148]. Our system uses preprocessing and interactive tools to assist users to

perform segmentation and reconstruction.

Yu et al. [148] use range images and photographs to segment, reconstruct and simplify objects from real scenes. Their goal is to edit the scene (e.g., move objects around) and no attempt to reconstruct missing areas was made. Since our pipeline uses a segmentation step, our scenes can also be edited.

Mark [75] discusses the issue of removing disocclusion artifacts in the context of post-rendering 3D warps. In this case, the hole-filling step is performed in image space.

One of the novel aspects of our work is the identification and use of approximate symmetry in point clouds and its use for reconstruction. Identification of symmetry is an important issue in pattern recognition and several algorithms have been proposed for finding symmetry in 2D images [68, 98, 145, 146]. Yip [145] used a Hough transform to identify approximate rotational symmetry in 2D binary images. The 2D Hough transform is used to identify the center of a set of concentric circles. The resulting algorithm has cost $O(n^3)$ on the number of pixels of the image. More recently, Yip exploited the use of 2D Hough transform to identify approximate bilateral and skew symmetry [146]. In this case, a three-step Hough transform is used to find the common bisector of a group of trapezoids. In our approach, a 3D Hough transform is used to identify both bilateral and rotational symmetry in unorganized point clouds. Our algorithm is conceptually much simpler and has a cost $O(n^2)$ on the number of points.

O'Mara and Owens [94] find the dominant plane of bilateral symmetry for 3D magnetic resonance images (MRI). The 3D MRI dataset is treated as a binary object for which a centroid and a covariant matrix are computed. The eigenvectors of the covariant matrix are computed and used as normals for three candidate planes. For each such plane, a measure of symmetry is computed by checking differences between intensity values in opposite sides of the plane. The one with highest symmetry measure is selected. PODOLAK ET AL. [101] PROVIDE AN EFFICIENT MONTE CARLO ALGORITHM FOR FINDING THE PLANAR

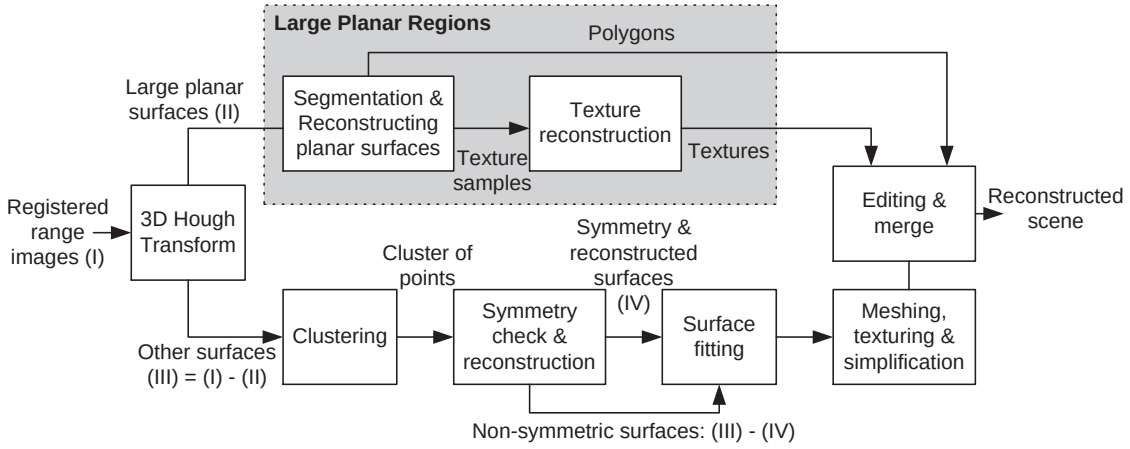


Figure 3.2: Segmentation and reconstruction pipeline.

REFLECTIVE SYMMETRY TRANSFORM FOR SURFACES AND ALSO PROPOSE AN ITERATIVE REFINEMENT ALGORITHM TO FIND THE LOCAL MAXIMA PRECISELY. THEY ALSO DEMONSTRATE VARIOUS APPLICATIONS OF THE SYMMETRY INFORMATION IN COMPUTER GRAPHICS, INCLUDING SHAPE MATCHING, SEGMENTATION, AND AUTOMATIC VIEWPOINT SELECTION. MITRA ET AL.[84] ACCUMULATE THE EVIDENCE FOR POSSIBLE SYMMETRIES IN THE TRANSFORMATIONAL SPACE. THE SIGNIFICANT PARTIAL AND APPROXIMATE SYMMETRY IS EXTRACTED VIA A CLUSTERING STAGE FOLLOWING BY A VERIFICATION STEP. THE PROPOSED ALGORITHM IS THEORETICALLY GUARANTEED TO HAVE A CERTAIN SUCCESS RATE.

Mills et al. [83] present an algorithm for finding global approximate symmetry for boundary representation (B-rep) models built from planes, spheres, cylinders, cones and tori. The symmetry check consists of trying a certain number of rigid motions (up to 120 permutations) and verifying whether each point lies within a certain tolerance from the original position of another point in the set. This algorithm works for relatively simple models with small number of vertices, such as simple polyhedra.

3.2 The Reconstruction Pipeline

The segmentation and reconstruction pipeline is shown in Figure 3.2. It exploits the observations that real scenes usually contain a significant number of planar and symmetric surfaces. The following subsections describe the steps of the pipeline.

3.2.1 Segmentation and Reconstruction of Planar Surfaces

The 3D Hough transform is a standard technique for identifying planar regions represented by sets of points in 3D. For each input point, a vote is cast to all cells of the Hough space representing planes passing through that point. To avoid finding spurious planes, the normal of every point is computed before vote accumulation and each point only votes for a few Hough cells. This decision is based on the normal at the point, which is computed based on its local neighborhood. The parameters of the final extracted planes are further refined fitting a plane through their selected points using singular value decomposition (SVD).

All salient planes can be identified using the technique described, but extracting polygons from the scene requires user assistance to specify the appropriate boundaries. In our system, this is done interactively, allowing for the creation of polygons with arbitrary shapes and sizes. This is particularly useful for reconstruction of occluded planar surfaces whose shapes are known a priori.

3.2.2 Texture Reconstruction for Planar Areas

Texture reconstruction for planar surfaces is based on isotropic and stochastic assumptions. For each texture, the synthesis procedure consists of:

- Orthographically project all texture samples onto the underlying plane and splat the

color information onto a texture buffer;

- Find the biggest square in the texture buffer whose interior is completely filled with the splats;

- Use this square texture as input for an Image Quilting procedure[36] and discard the original texture.

For the reconstruction of symmetric texture patterns, such as the one shown in the Oriental rug (Figure 3.3), a 2D Hough transform is used. In this case, the extracted texture is scanned and, for each texel t, all other texels with similar colors (based on an L2 norm in RGB space) are identified. A vote is then cast to each of the bisectors between t and all its similar texels. After the scanning of all input samples is completed, the bisectors with most votes are selected as the symmetry axes for the texture and used to mirror filled texels to unfilled ones. Since the accuracy of this procedure depends on the quality of the input data, user assistance is usually required. After copying texels according to the symmetry axis, some texels may still be empty. These texels are filled using a pull-push strategy [44] based on a texture pyramid. Figure 3.3 illustrates the intermediate steps produced by this technique for the reconstruction of an oriental rug. Figure 3.3(a) shows the extracted texture and one of its symmetry axes. The image in the center shows the reconstruction obtained by mirroring, while on the right one sees the final result produced by the pull-push algorithm.

3.2.3 Clustering

After eliminating all points belonging to large planar surfaces, the pipeline proceeds by segmenting clusters of points spatially close to each other, which are treated as individual objects. Clusters are identified using an incremental surface construction algorithm [43]

Figure 3.3: Reconstruction of an oriental rug: (a) extracted texture; (b) reconstruction by mirroring; and (c) final result produced by the pull-push algorithm.

based on the projected distance, onto local tangent planes, of adjacent points in 3D.

For each pair of points in the point cloud do

Compute the bisector plane for the two points

Cast a vote for the computed plane

If plane Π_i alone received most votes then

Π_i is a symmetry plane //bilateral symmetry

elseif Π_i , Π_j ... Π_n all received many votes then

a = intersection(Π_i , Π_j ... Π_n) is a symmetry axis

else

no symmetric pattern found.

Algorithm 3.1: Identifying approximate symmetry in point clouds.

3.2.4 Symmetry Check and Reconstruction

Symmetric features are pervasive in both natural and man-made environments [48, 72, 138, 141]. The redundancy embodied in symmetric shapes is exploited to allow reconstruction to proceed from incomplete data.

Due to its relative insensitivity to noise and missing data, the Hough transform is a

natural tool for identifying approximate symmetries in unorganized point clouds. Algorithm 3.1 describes a procedure for identifying such patterns. In the case multiple planes presenting approximately the same number of votes (this indicates the presence of rotational symmetry), the symmetry axis is computed as the intersection of all identified planes using SVD.

Put all points in a queue
While the queue is not empty do
  Get one point **p** from the queue
  Find the samples in the local neighborhood of **p** and the
    samples in its mirrored neighborhood
  Project both neighborhoods onto the tangent plane of **p** and
    sort them by angle around **p**
  If the angle between two projected adjacent points a_i and a_j
    from the local neighborhood of **p** is bigger than 120 degrees,
    find a point **q** from the mirrored neighborhood, whose
    projection falls in between (angular wise) a_i and a_j and
    whose distance to **p** is between μ*local_min_dist(**p**) and
    local_min_dist(**p**) is the minimum distance between **p** and
    any point in its local neighborhood. μ is a user specified
    parameter for the entirereconstruction process
  Add **q** to the queue.

Algorithm 3.2: Symmetry reconstruction.

Once a symmetry plane is has been identified, points can be mirrored to the other side of the plane to fill holes or simply to improve the sampling rates locally. The identification of regions that can potentially benefit from mirroring is performed in the following way: for each point *p* in the point cloud, we project its local (3D) neighborhood onto its tangent plane and sort the projections by angle around *p*. If the angle between two projected adjacent points a_i and a_j from the local neighborhood of *p* is bigger than 120 degrees, a possible hole (or boundary) may exist. Only in this case, reconstruction by mirroring is applied. The reconstruction process is described in Algorithm 3.2 and was inspired by the algorithm used for surface reconstruction [43]. Figure 3.4 illustrates this situation.

Because the computed position for the symmetry plane is usually only approximate, the distances between each point in the mirrored neighborhood and the tangent plane of the current point (p, in Algorithm 3.2) are computed as well. If a distance is bigger than a certain threshold, the corresponding point is discarded. A dexel data structure [50] is used to accelerate the search of neighborhoods.

Depending on the input data, there may still be some non-sampled regions even after symmetry reconstruction. We currently perform surface reconstruction using Gopi's algorithm [43]. Like other surface reconstruction strategies from point clouds [7, 12, 35, 52], it cannot guarantee filling in holes in areas with local variations of sampling density. Our system currently does not handle these cases and we intend to implement an interactive tool for reconstructing these regions by locally fitting surfaces using a moving least squares approach [64].

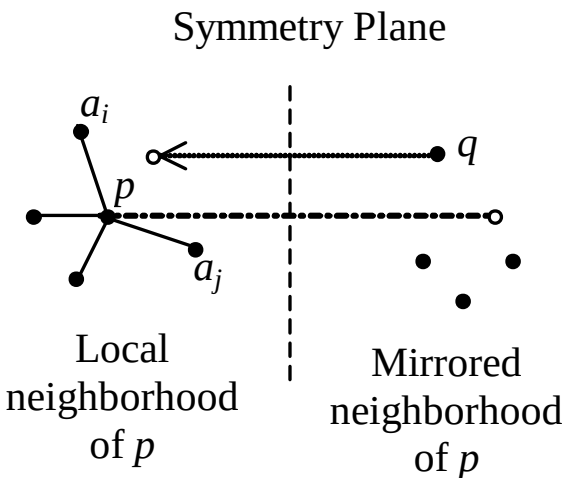


Figure 3.4: Illustration of the symmetry reconstruction in the neighborhood of point /textitp.

After incorporating new samples by exploiting symmetry and surface fitting, each cluster is then meshed and possibly textured and simplified. Meshing is currently performed using the same algorithm used for cluster segmentation [43].

If a planar region does not have enough samples, the corresponding plane may not be correctly extracted. For these cases, we have implemented a tool for cloning already reconstructed polygons, which can then be repositioned and resized to make up for large missing areas. This technique was used to create the top of the furniture next to the window, a wall and the back of the shelves not visible in the original dataset. The approach is also illustrated in one of the accompanying animations.

Textures can also be reconstructed by cloning. We use a cloning brush tool similar to the one available in Photoshop to reconstruct some kinds of textures. This tool was used to help reconstruct the window texture occluded by the small plant (Figure 3.7) and textures for the lower portion of the bookshelves occluded by the chairs.

3.3 Results

We have implemented the pipeline described in Section 3.2, with the exception of the surface fitting step. The code was written in C++ and times were measured using a debug version of the code on a 2.0 GHz Pentium 4 PC with 512 MB of memory. Our prototype was used for reconstructions of a real and a synthetic environment. Hough tables with sizes 60x60x160 were used in both cases for the detection of large planar areas. In Hough space, this corresponds to angular increments of 3 degrees. The size of the increment in ρ varied with the length of the scene's bounding box's diagonal (which was divided by 160). For the case of the real scene, $\Delta\rho = 0.51$ units, and for the synthetic scene $\Delta\rho = 1.93$ units.

3.3.1 Reconstruction of a Real Scene

The real dataset used is a partial model of the Sitterson Hall (UNC) reading room, consisting of ten 864x240 range images containing a total of 1,805,139 valid samples. These are shown as a panorama in Figure 3.5. For this scene, the computation of the Hough

transform for identifying planar regions took approximately 17 minutes and its results were saved for later use during an interactive session.

Although the Hough transform identified all major planar areas, we decided to replace only five of them with texture-mapped polygons: the wall containing the window, the wall next and parallel to it, the front of furniture also next to the window, the floor and the bottom part of the bookshelf. After this step, the scene was left with 1,351,235 points. Replacing other planar regions of walls would have brought this number down to 863,391 points. We decided not to do so and allow a larger number of points to proceed to the clustering step of the pipeline. The clustering step isolated the three chairs and the remaining walls as four

Figure 3.5: Sitterson Hall reading room. Panorama obtained by concatenating ten 864x240 range images.

different clusters. Reconstruction then proceeds with the middle chair in Figure 3.5. The symmetry plane computed by the Hough transform (using a table of size 70x70x70) was slightly off the center, requiring minor user intervention. This is due to the fact that only one external side of the chair is visible (Figure 3.5). Figure 3.6 shows two views of the arm-chair before and after the reconstruction. The resulting model contains 64,290 points and 127,525 triangles. Despite the existence of small holes (in areas where no data was available in either side of plane), the reconstructed model (Figure 3.6(b)) shows a significant

improvement when compared to the original chair (Figure 3.6(a)). The resulting polygonal model was instantiated three times, replacing the original chairs in Figure 3.1. The

(a) (b)

Figure 3.6: Views of the armchair: (a) before reconstruction; and (b) after reconstruction.

texture for the wall containing the window was extracted from the original dataset by orthographically projecting all samples within a certain tolerance to the plane (Figure 3.7(a)). Notice the existence of holes, mostly due to occlusions, shown in black. The final texture (Figure 3.7(b)) was obtained with the use of the clone-brush tool to reconstruct the area corresponding to the shadow of the plant on the shades. The remaining holes were filled with the use of the push-pull algorithm.

(a) (b)

Figure 3.7: Reconstruction of texture: (a) texture extracted from the original point clouds; (b) reconstructed texture using a clone-brush tool and pull-push.

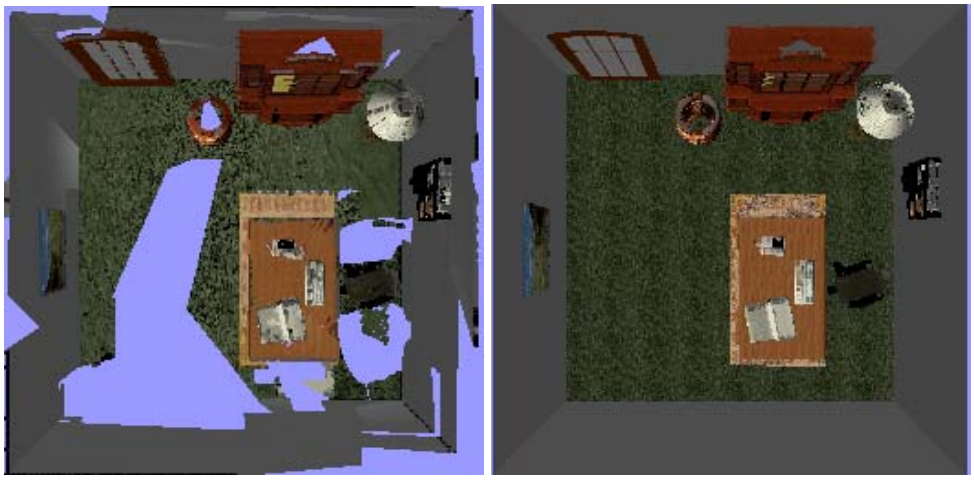
The carpet texture was synthesized from a small sample seen in Figure 3.8 using the

method described by Efros et al. [36]. The top of the furniture next to window and a wall not visible in Figure 6 were created using the polygon cloning technique. The textures for bottom of the three bookshelves were created with the clone-brush tool. The final edited scene is shown in Figure 3.1(b). Essentially, all disocclusion artifacts have been removed.

3.3.2 Reconstruction of a Synthetic Scene

We also tested our system on a larger and more complex synthetic dataset of an office environment. It consists of nineteen 640x480 range images, containing a total of 4,397,683. Top views of the scene before reconstruction are shown in Figure 3.8.

The office scene contains a chair, a desk, a floor lamp, a vase, a bookshelf, an oriental rug, computer monitor, keyboard and a phone. The floor lamp and the vase are used to illustrate the reconstruction of rotationally symmetric objects. The reconstruction of the chair, monitor and bookshelf are based on bilateral symmetry. The rug illustrates our attempt to reconstruct complex texture patterns, which is shown in Figure 3.3.



(a) (b)

Figure 3.8: Top view of a synthetic office: (a) before reconstruction; and (b) after reconstruction.

First, the major planes of the scene (walls, floor, the top and some faces of the desk)

were identified using the Hough transform. Textures for the carpet and desk were synthesized from the original samples using the procedure described in Section 3.2.2. After the major planar surfaces have been removed, all the remaining objects were segmented automatically and submitted to a symmetry check. Figure 3.9 shows three views of the chair. The image on the left shows the chair before reconstruction. The image on the center shows the symmetry plane, and to its right, the resulting model computed automatically. Figure 3.10 shows renderings of the monitor before and after its automatic reconstruction. Little holes in the final model, not present in the original incomplete model, are caused by the distance criterion used by the surface reconstruction algorithm used [43]. All the original points are preserved. Figure 3.11 illustrates the reconstruction of a rotationally

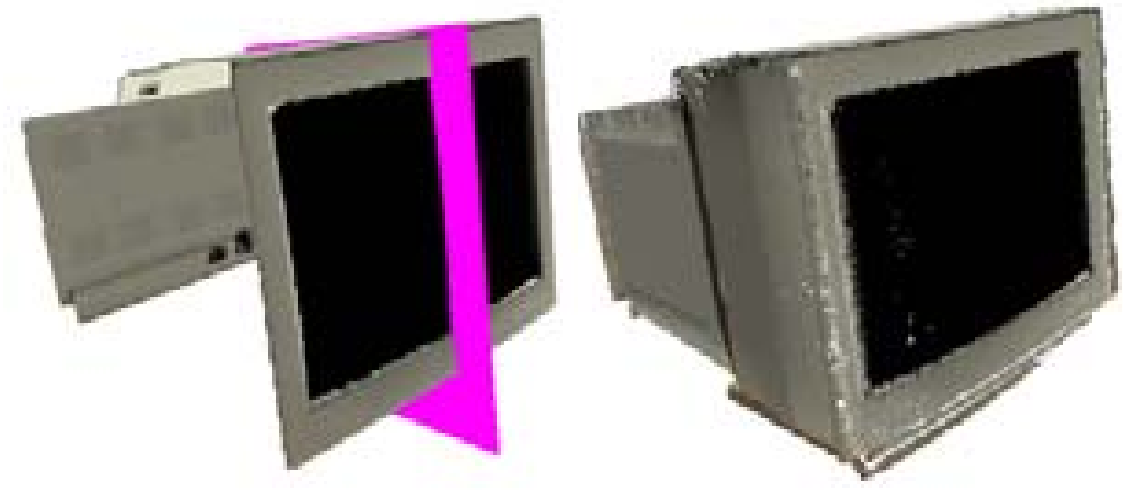
(a) (b) (c)

Figure 3.9: Office chair: (a) before reconstruction; (b) with the symmetry plane computed; and (c) after reconstruction.

symmetric object. The model is shown before reconstruction, with a symmetry plane and after reconstruction. Notice that although there is a symmetry axis, reconstruction becomes much simpler when performed as a series of reflections with respect to planes. The number of points per object before and after reconstruction is shown in Table 3.1.

The algorithm looks for symmetric regions containing unbalanced number of samples and tries to balance them, it is possible that areas whose surfaces were already filled in

the incomplete model receive new samples. Since at this point, our system copies samples to the new location, it may introduce texture artifacts due to differences in shading. This situation is illustrated in the case of the floor lamp (Figure 3.11), for which samples with darker shade samples were copied onto brighter regions.



(a) (b)

Figure 3.10: Monitor: (a) before reconstruction, but with the symmetry plane computed; and (c) after reconstruction.

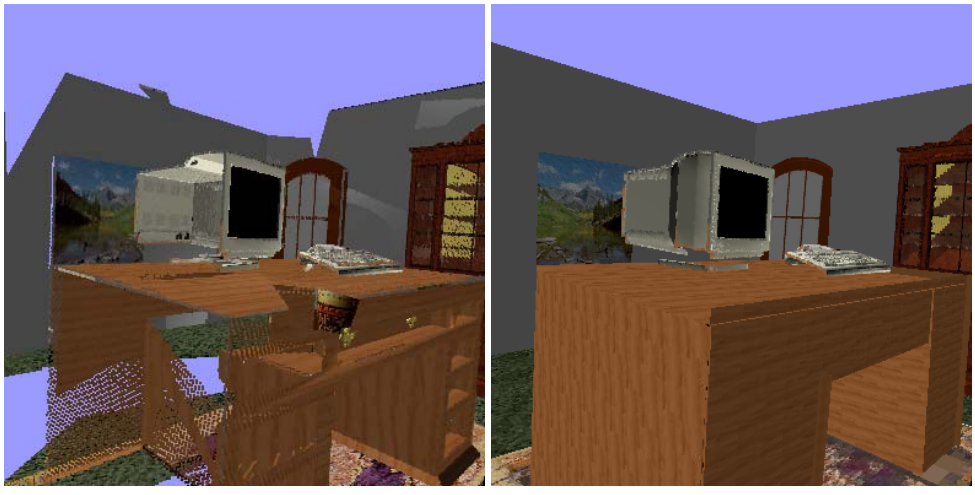
The oriental rug was reconstructed using the algorithm described in Section 3.2.2, and its results are illustrated in Figure 3.3. Figure 3.12 shows views of the office before and after reconstruction.

Table 3.1: Number of points per object before and after reconstruction

Object	# original points	# points after	# triangles after
Chair	41,511	64,290	127,525
Office chair	51,237	70,595	136,464
Monitor	81,329	103,243	206,982
Lamp	47,968	64,722	129,771
Vase	187,031	268,061	555,228

(a) (b) (c)

Figure 3.11: Floor lamp: (a) before reconstruction; (b) with the symmetry plane computed; and (c) after reconstruction.



(a) (b)

Figure 3.12: View of the office: (a) before reconstruction; and (b) after reconstruction.

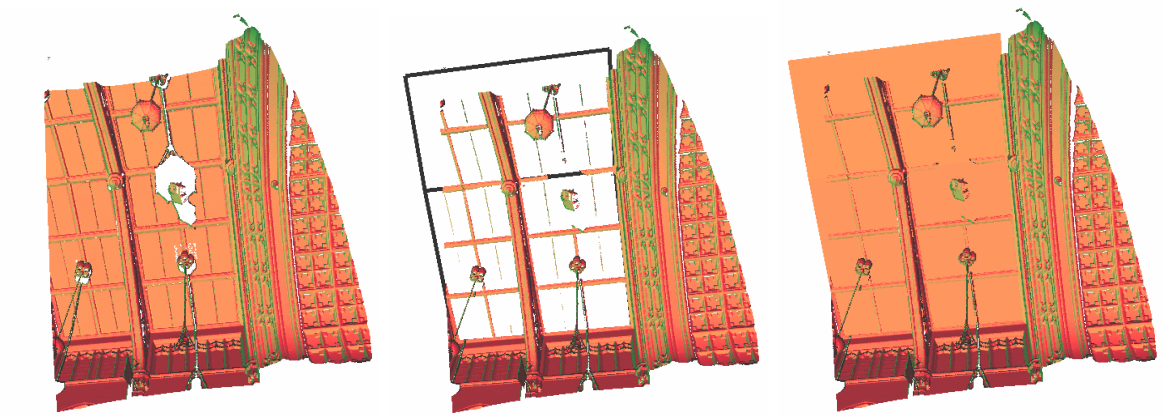
3.3.3 Discussion

OCCLUSIONS AND SCANNER ACCESSIBILITY LIMITATIONS TO CERTAIN REGIONS CAUSE SOME SURFACES NOT TO BE SAMPLED, RESULTING IN INCOMPLETE OR INCORRECTLY RECONSTRUCTED MODELS. THIS CHAPTER PRESENTS A PIPELINE FOR RECONSTRUCTING MISSING GEOMETRY AND TEXTURE DATA, THUS IMPROVING THE QUALITY OF THE RESULTING MODELS. THE CENTRAL IDEA OF THE PIPELINE IS BASED ON THE OBSERVATIONS THAT REAL SCENES USUALLY CONTAIN A NUMBER OF LARGE PLANAR SURFACES AND SYMMETRIC SURFACES. WE FIRST IDENTIFY LARGE PLANAR AREAS, REPLACING THEM WITH TEXTURED POLYGONS. IF THE COLOR INFORMATION IS NOT AVAILABLE WITH THE POINT DATASET, THE GENERATED POLYGON WILL NOT BE ASSOCIATED WITH A TEXTURE. THE REMAINING SAMPLES ARE SEGMENTED INTO SPATIALLY COHERENT CLUSTERS, WHICH ARE THEN ANALYZED FOR THE EXISTENCE OF SYMMETRY.

TO FIND LARGE PLANAR SURFACES, WE DISCRETIZE THE SYMMETRY PLANE SPACE WITH ITS ORIGIN DELIBERATELY SET AS THE CENTER OF THE BOUNDING BOX. THE SIZE OF CELLS OF THE SPACE IS SELECTED EMPIRICALLY IN THE EXPERIMENTS. WITH A SMALLER CELL, WE MAY OBTAIN A MORE ACCURATE SYMMETRY PLANE. HOWEVER, SINCE THE COMPLEXITY OF THE HOUGH TRANSFORM IS $O(ns)$, WHERE n IS THE NUMBER OF INPUT POINTS AND s IS THE SIZE OF THE HOUGH TABLE, THE RUNNING TIME WILL ALSO INCREASE DRAMATICALLY.

ONE PROBLEM OF FINDING LARGE PLANAR SURFACES USING HOUGH TRANSFORM IS THAT A NEARLY PLANAR SURFACE WITH DETAILS MAY BE REPLACED AS A SIMPLE POLYGON. TO PRESERVE THE DETAILS, USER MAY MARK THIS REGION AS INAPPROPRIATE FOR REPLACING. IN FIGURE 3.13, AN INDOOR POINT MODEL HAS SOME MISSING REGIONS IN THE CEILING. THE CEILING IS REPLACED BY TWO POLYGONS. SINCE

THE DISTANCE OF SOME POINTS REPRESENTING THE DETAILS ON THE CEILING FALLS WITHIN THE THRESHOLD, THESE POINTS ARE CONSIDERED TO BE PART OF THE PLANE AND ARE REMOVED. IN THIS EXAMPLE, THIS REMOVAL IMPLY THE LOSS OF DETAILS.



(a) (b) (c)

Figure 3.13: Recovery of some missing regions of a indoor model using the planarity information: (a) the original model has some missing regions; (b) points belonging to the ceiling are replaced by two planar shapes; and (c) the recovered missing regions.

SYMMETRY IMPLIES REDUNDANCY AND CAN BE EXPLOITED FOR MODEL RECONSTRUCTION FROM INCOMPLETE DATA. WE USED THIS OBSERVATION TO DESIGN A FAMILY OF NEW ALGORITHMS BASED ON 3D AND 2D HOUGH TRANSFORMS FOR IDENTIFYING APPROXIMATE SYMMETRIC PATTERNS IN POINT CLOUDS AND TEXTURES, AND USING SUCH PATTERN TO PERFORM RECONSTRUCTION. THE HOUGH TRANSFORM IS AN APPROPRIATE TOOL FOR THIS TASK DUE TO ITS RELATIVE INSENSITIVITY TO NOISE AND MISSING DATA. SYMMETRY-BASED RECONSTRUCTION DEPENDS ON THE QUALITY OF THE INPUT DATA AND USER ASSISTANCE MAY BE REQUIRED. NEVERTHELESS, IT IS A POWERFUL CONCEPT FOR RELAXING THE REQUIREMENTS OF FULL DIGITIZATION OF ALL SURFACES IN THE SCENE, AS CURRENTLY REQUIRED FOR THE CONSTRUCTION OF COMPLETE MODELS. IN ESSENCE, PODOLAK'S SYMMETRY CHECKING ALGORITHM [101] EMPLOYS THE SIMILAR IDEA AS OURS.

THE PROPOSED *planar reflective symmetry transform* (PRST) PROVIDES A CONTINUOUS MEASURE OF THE REFLECTIVE SYMMETRY. PODOLAK ET AL. USE A MONTE CARLO INTEGRATION FOR COMPUTING A DISCREET VERSION OF THE PRST, WHICH IS ALSO SIMILAR TO THE HOUGH TRANSFORM USED IN OUR ALGORITHM. THE DIFFERENCE IS THAT PODOLAK'S APPROACH COMPUTES THE PRST FOR VOLUMETRIC FUNCTIONS AND SURFACE MESHES WHILE OURS WORKS ON POINT CLOUDS. WHEN THE INPUT BECOMES PURE POINTS, IT IS NECESSARY TO FIRST COMPUTE A DISTANCE FIELD FROM THESE POINTS FOR PODOLAK'S ALGORITHM, WHICH IS STILL A CHALLENGING PROBLEM THOUGH. PODOLAK'S APPROACH ALSO INCLUDES AN ITERATIVE REFINEMENT ALGORITHM FOR FINDING LOCAL MAXIMA WITH ARBITRARY PRECISION. IN CONTRAST, MITRA ET AL. ADOPT THE SIMILAR CONCEPT TO COMPUTE SYMMETRIES AS WELL. HOWEVER, THE PRIMITIVES BECOME LOCAL SHAPE DESCRIPTORS INSTEAD OF POINTS. THIS IDEA SIGNIFICANTLY REDUCES THE NUMBER OF POSSIBLE REFLECTIVE PLANES. HOWEVER, THIS WILL NEED THE NORMAL AND CURVATURE FOR THE POINTS, WHICH MAY NOT BE EQUIPPED WITH GENERAL POINT MODELS. AND THE COMPLEXITY OF THE CURRENT ALGORITHM IS $O(n^2)$, WHERE n IS THE NUMBER OF POINTS.

WE CURRENTLY COPY COLOR AND TEXTURE INFORMATION REGARDLESS OF SHADING DIFFERENCES BETWEEN THE SOURCE AND DESTINATION REGIONS. REFLECTANCE PROPERTIES EXTRACTED DIRECTLY FROM SURFACES [147] COMBINED WITH LIGHTING INFORMATION CAN BE USED TO IMPROVE THESE RESULTS.

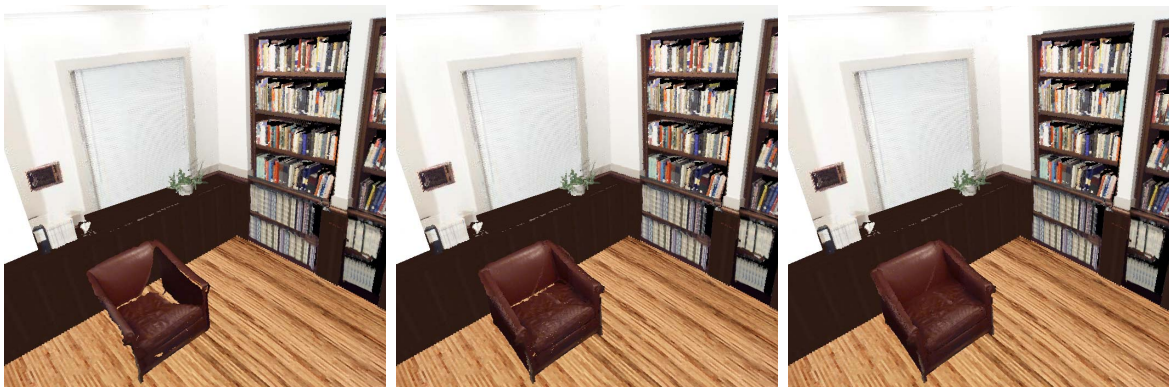
IN PRACTICE, IT OFTEN TURNS OUT THAT THE PROPOSED PIPELINE COULD ONLY RECOVER A LIMITED PORTION OF THE MISSING REGIONS. WE SUGGEST USERS USE THIS PIPELINE THE FIRST STEP FOR A MORE COMPLETE RECONSTRUCTED SURFACE. WITH MANY MISSING REGIONS RECOVERED USING USER KNOWLEDGE ABOUT THE PLANARITY AND SYMMETRY, WE HAVE MOVED ONE STEP FURTHER TOWARDS A MORE

REASONABLE RECONSTRUCTION. AND THE REMAINING MISSING REGIONS MAY BE RECOVERED USING OTHER KINDS OF INFORMATION AND ASSUMPTIONS. ONE EXAMPLE OF THE ASSUMPTION IS DEMONSTRATED IN THE NEXT CHAPTER.

Chapter 4

Hole Filling as a Post-process

Creating accurate models of real objects and environments is a non-trivial task for which the use of traditional modeling techniques is inappropriate. In these situations, the use of laser rangefinders [16] seems attractive due to its relative independence of the sampled geometry and short acquisition time. The combined use of range and color images



(a) (b) (c)

Figure 4.1: Partial model of the UNC reading room, a real environment digitized with a 3D laser scanner. The floor texture has been replaced to emphasize the holes in the chair model: (a) chair model reconstructed from the original samples only. Notice the big holes and missing outer surface on its left; (b) model obtained from (a) using our symmetry-based techniques [134]. Despite the clear improvement, many small holes are still visible; and (c) complete model obtained by applying our hole-filling algorithm to the model shown in (b).

is very promising and has been demonstrated to produce an unprecedented degree of photorealism [78, 90]. Unfortunately, surface properties (*e.g.*, low or specular reflectance), occlusions and accessibility limitations cause scanners to miss some surface areas, leading to incomplete reconstruction and introducing holes in the resulting models. This makes hole filling an important component of object and scene reconstruction. Its importance can be better appreciated when considering that, often times, it will not be possible to re-scan the original scene for acquiring extra samples. This may happen because the scene might have changed or due to cost limitations. In these situations, one should try to obtain the best possible reconstruction using only the available samples. Creating high quality mesh representations for objects in the scene based on such incomplete information remains a challenge [148].

The problem of filling holes in range data can be divided into two sub-problems: (i) identifying the holes, and (ii) finding appropriate parameterizations that allow the reconstruction of the missing parts using the available data. Unfortunately, none of these problems are trivial because holes arising from the scanning of geometrically complex surfaces (*e.g.*, twisted, self-occluding surfaces) can be quite intricate [31]. However, in many situations of practical interest, holes occurring in range images present simple topologies. This is the case, for example, of many holes found when scanning interior environments and most objects. For these cases, the quality of the reconstructed 3D meshes can be significantly improved with the use of relatively simple algorithms.

The proposed algorithm can be applied to manifolds and surfaces with boundaries. In the later case, user assistance is required to avoid the inherent ambiguity of deciding which holes are artifacts of undersampled regions and which ones may define true surface boundaries. The technique does not provide a general solution to the hole-filling problem. In particular, it does not handle holes with arbitrary topologies or on highly twisted geometry. Nevertheless, it can be applied to a large range of practical situations, is conceptually very

simple and its implementation is straightforward. Essentially, the algorithm takes a point cloud as input and produces an intermediate representation consisting of a triangle mesh, which is then analyzed for the existence of boundary edges (edges belonging to a single triangle). The occurrence of a hole implies the existence of a cycle defined by boundary edges. Once a boundary edge is found, the algorithm traces the entire boundary. A ring of points around the boundary, called the *boundary vicinity*, is then used to interpolate the hole using a *moving least squares* (MLS) procedure.

Our algorithm presents several desirable features:

- Hole filling is performed after meshing and, therefore, the algorithm can use any surface reconstruction technique that produces a triangle mesh;
- Since MLS interpolates the original samples, the algorithm guarantees that the reconstructed patches smoothly blend into the original mesh;
- It can be used to fill holes on both closed surfaces as well as on surfaces with boundaries;
- The reconstructed patches preserve the original sampling rate of their vicinities;
- As the new primitives are distinguished from the original points, they can be processed further;
- The processing is limited to the vicinities of holes.

We demonstrate the effectiveness of our approach on both real and synthetic datasets and show that it can significantly improve the overall quality of the models. We also show that for locally smooth surfaces, the described technique produces better results than the state-of-the-art hole-filling techniques [31, 57]. In particular: (i) the reconstructed patches blend with the original model in a smoother way, (ii) it preserves the original mesh, and (iii) the

resulting models contain a smaller number of vertices and triangles than the ones produced by other techniques [31, 57] (see Section 4.4).

Figure 4.1(a) shows a partial model of the UNC reading room, an interior environment scanned with a laser rangefinder. Color, except for the floor texture, was obtained from photographs taken with a digital camera [78]. The clear floor texture was chosen in order to highlight the existence of holes in the chair model. Figure 4.1(b) shows the chair model after it has been processed using the techniques described by Want et al. [134]. Despite the considerable improvement, many small holes are still visible in the chair. Figure 4.1(c) shows the same scene after our hole-filling algorithm has been applied to the chair model of Figure 4.1(b). Notice that the holes have been removed.

4.1 Moving Least Squares

Moving least squares (MLS) provides a class of complete solutions to the problem of fitting smooth functions to scattered data [64]. This is performed by interpolating the original data points, which may not be desirable in case the dataset is noisy. Our algorithm, however, uses MLS only to fill holes. Therefore, surface reconstruction can be performed with any reconstruction technique and may even include a low-pass filtering step to minimize the effects of noise. By using MLS to guide the hole filling process, our algorithm guarantees that the reconstructed patches smoothly blend into the reconstructed mesh. This is accomplished with the use of a relatively small number of samples in the vicinity of the hole boundaries. The entire algorithm is described in Section 4.2. The remaining of this section provides a quick review of MLS interpolation. For a more in-depth discussion of the subject, we refer the reader to [64].

Let s be a height function defined over a two-dimensional subspace ($s \models U \subset \Re^2 \rightarrow \Re$) and let $p_i \in U$ be a point in the domain of s. f_i is a height measurement associated with p_i.

The fitness of s to a set of values f_1 to f_N can be measured by the error

$$E(s) = \sum_{i=1}^{N} w_i (s(p_i) - f_i)^2 \tag{4.1}$$

where N is the total number of points and w_i is the weight associated with point p_i. The best fit to the given set of values is obtained by minimizing the error $E(s)$. In practice, s is usually approximated by simple polynomial functions, such as the one shown in Equation 4.2 [64].

$$s(u, v) = a_0 + a_1 u + a_2 v + a_3 u^2 + a_4 v^2 + a_5 uv \tag{4.2}$$

In this case, the a_i coefficients that minimize the error are obtained by solving

$$a = (BWB^T)^{-1} BWf \tag{4.3}$$

where B is the matrix shown in Equation 4.4 and W is an n by n diagonal matrix with diagonal elements equal to w_i.

$$B = \begin{bmatrix} 1 & \cdots & 1 \\ u_1 & \cdots & u_n \\ v_1 & \cdots & v_n \\ u_1^2 & \cdots & u_n^2 \\ v_1^2 & \cdots & v_n^2 \\ u_1 v_1 & \cdots & u_n v_n \end{bmatrix} \tag{4.4}$$

Such a weighted least squares solution can only represent low order surfaces, often resulting in poor approximations. To reflect the fact that samples near the resampling positions should have more influence than far away samples, the error function should take into

account weight factors w_i, which vary with the evaluation point:

$$E_p(s) = \sum_{i=1}^{N} w_i(p)(s(p_i) - f_i)^2 \tag{4.5}$$

For this case, a good choice of weight function is given by[64]:

$$w_i(p) = \frac{e^{-\alpha d_i^2(p)}}{d_i^2(p)} \tag{4.6}$$

Here, $d_i(p)$ is the distance from the new sampling position p to the ith original sample p_i (in the vicinity). When evaluated right at an input point, this weight function becomes infinity, thus interpolating the point. To avoid numerical problems, evaluation right at input points are handled individually. The parameter α controls the influence of vicinity features on the region to be resampled. As the weight functions depend on the resampling positions, new coefficients a_0 to a_5 for Equation 4.2 need to be computed for every resampled point as:

$$a(p) = (BW(p)B^T)^{-1}BW(p)f \tag{4.7}$$

where the elements of the diagonal matrix $W(p)$ are computed using Equation 4.6. Compared to the weighted least squares method, which creates a quadric surface for the entire hole, moving least squares compute one quadric surface for each evaluation point and blends them all. Therefore it can fit higher order surfaces.

4.2 The Hole Filling Algorithm

In order to fill holes, new points need to be added to the unsampled regions. To accomplish this, the algorithm first identifies hole boundaries and their vicinities. For each hole, it fits a plane through the vicinity points and, for each such a point, computes its

distance to this plane as well as its projection onto the plane. The set of distances define a height field around the hole which is then used for surface fitting. This way, the problem of reconstructing holes in 3D is reduced to a simpler interpolation problem. Once a surface has been fitted to the height field using MLS, new points for filling the hole can be obtained by resampling the fitted surface. The basic version of the algorithm is presented in Algorithm 4.1, and its details are explained next.

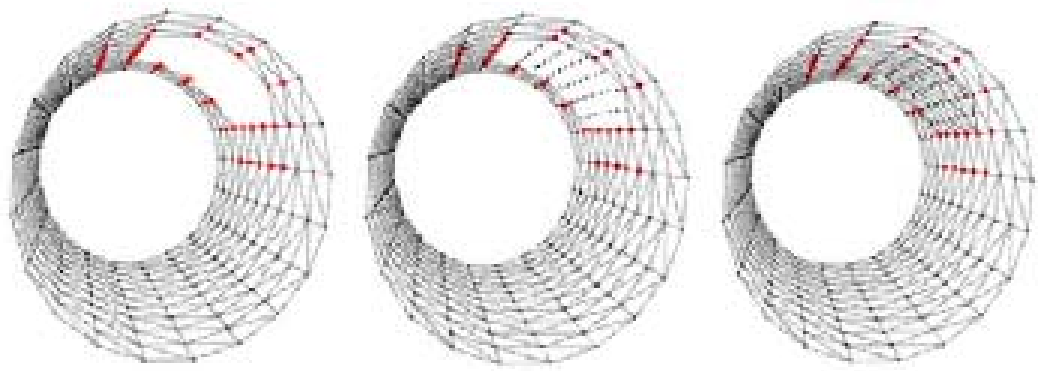
4.2.1 Finding Holes

In order to identify holes, we start by creating a triangle mesh from the input point cloud. A number of algorithms exist for this purpose [7, 12, 13, 35, 43, 52]. For the results shown in this chapter, we have used an incremental surface reconstruction algorithm [43], which was chosen due to its simple implementation.

A hole consists of a loop of boundary edges. A *boundary edge* is defined as an edge belonging to a single triangle, as opposed to *shared edges*, which are shared by two triangles. By tracking boundary edges, holes can be identified automatically. Note, however, that there are two kinds of distinct boundaries: internal and external ones. An *internal* boundary delimits a hole on a surface. An *external* boundary, in turn, delimits either a patch ("island") inside a hole, or the limits of a surface, such as in the case of the end portions of the cylindrical surface shown in Figure 4.2. From the example of the cylinder, it becomes

Create a triangle mesh from the input point cloud
Repeat
  Automatically find a hole boundary and its vicinity
  Compute a reference plane for the hole vicinity
  Compute the distances of the vicinity points to the plane
  Fit a surface through this height field using MLS
  Fill the hole by resampling the fitted surface
Until no holes exist

Algorithm 4.1: The hole-filling algorithm



(a) (b) (c)

Figure 4.2: Cylinder: (a) triangle mesh with the hole and its boundary vicinity identified; (b) new points added; and (c) reconstructed mesh.

clear that not all holes should be necessarily filled and that user assistance is required in order to guarantee proper reconstruction.

4.2.2 Computing the Reference Plane

Once a hole has been identified, the next step is to use a ring of points around the boundary of the hole to provide a context for its interpolation. A height field is obtained by computing the distances from these points to a reference plane, which is the best fit plane to the set of points in the vicinity of the hole. The plane's position and orientation are computed as follows: first, the average $O = (\bar{x}, \bar{y}, \bar{z})$ of all vicinity points is computed as the origin of a new coordinate system associated to the plane. A matrix M is obtained by subtracting O from all points in the vicinity (Equation 4.8). Then, *Singular Value Decomposition* (SVD) [104] is used to compute the eigenvectors and eigenvalues of $M^T M$. The two eigenvectors with the largest absolute values span the reference plane and correspond to the U and V axes shown in Figure 4.3. The third eigenvector represents the plane normal (S axis).

$$M = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} & z_1 - \bar{z} \\ x_2 - \bar{x} & y_2 - \bar{y} & z_2 - \bar{z} \\ . & . & . \\ x_{N-1} - \bar{x} & y_{N-1} - \bar{y} & z_{N-1} - \bar{z} \\ x_N - \bar{x} & y_N - \bar{y} & z_N - \bar{z} \end{bmatrix} \tag{4.8}$$

4.2.3 Determining the Resampling Positions

Each vicinity point is orthographically projected into the reference plane (UV plane), producing a pair of coordinates (u, v) and a height s computed as its distance to the reference plane. These values are used to fit the surface using MLS.

It is important that the set of points used to resample the hole have the same sampling density as the vicinity points. Two criteria are used for determining the resampling positions:

- The projections of new points should fall on the projection of the hole on the reference plane;
- The minimum distance from any new point to any other one (either new or vicinity point) should be bigger than a threshold.

While the first criterion seems self-evident, the second one is used to guarantee good remeshing results, since some reconstruction techniques require the input points to be spaced as evenly as possible [43, 52]. The vicinity mesh is orthographically projected onto the UV plane, defining a mask. This situation is illustrated in Figure 4.4 for the case of a hole topologically equivalent to a disk. In case the hole contains "islands", they should also be projected and will be part of a disconnected mask. The mask image is traversed in scan line order using the step size compute using Equation 4.9. New sampling positions are

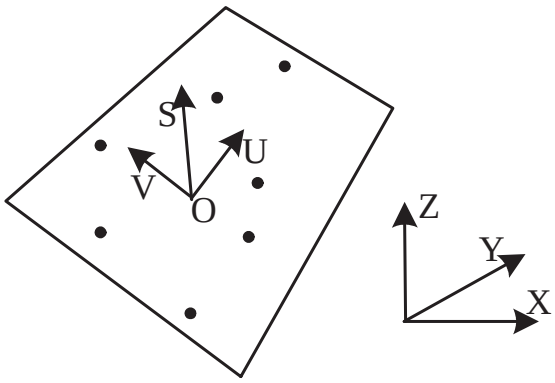


Figure 4.3: The UV projection plane

then set over a regular grid inside the hole in the UV plane. If the distance between a point and the vicinity mask is less than $0.5 \times stepsize$, such a point is not used as a resampling point.

$$stepsize = \sqrt{\frac{area}{3n}} \tag{4.9}$$

Equation 4.9 provides a heuristic for spacing the resample positions inside a hole. n is the number of points on the boundary of the hole and *area* is the sum of the areas of all triangles connected with these points. The vicinity of the hole is then defined as the set of points whose distances from the boundary of the hole is less than $\beta \times stepsize$. The "thickness" of the vicinity ring is controlled by the parameter β.

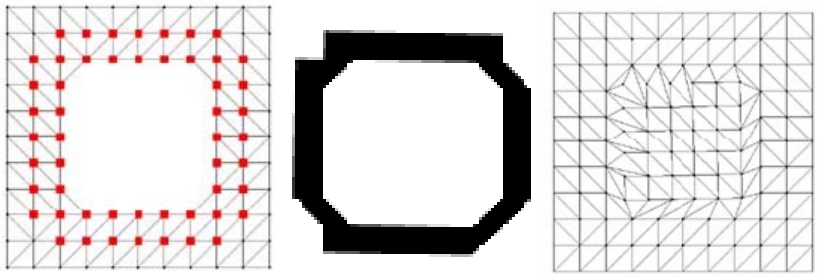
4.2.4 Fitting the Surface

For each new point created to fill holes, a solution of Equation 4.7 provides the coefficients of Equation 4.2 necessary for determining $s(u, v)$. After that, the transformation from the UVS coordinate system to the XYZ coordinate system is straightforward. Similarly, the colors (R,G,B channels) associated to the points in the vicinity of a hole are

Table 4.1: Statistics and running time for different datasets

Data Set	# Origingal Points	# Vicinity Points	# Final Points	# Origingal Triangles	# Final Triangles	MLS time (seconds)
Armchair	64,159	1,873	65,114	126,588	136,002	10.93
Bunny	35,947	1,246	36,478	69,451	70,733	6.43
Bust	12,853	1,506	16,162	21,757	29,357	72.85
Angel	10,189	347	11,264	18,737	20,921	4.16
Buddha	143,298	1,154	144,701	290,936	293,707	10.32

treated as three separate height functions, from which the colors of the new points are re-sampled. This is achieved by replacing the height value of each vicinity point with the value of one of its associated color channels at a time. The resulting three height functions representing the three color channels are reconstructed using MLS and are resampled using a procedure similar to the one used to recover the missing geometric information. After the new points have been introduced, the final step is to remesh the complete model. Figure 4.4 illustrates the intermediate steps of the algorithm for the simple case of a planar surface.



(a) (b) (c)

Figure 4.4: Intermediate results of the algorithm: (a) a triangle mesh with the projection of the vicinity points highlighted; (b) a mask image for the projection of the vicinity points onto the reference plane; and (c) the reconstructed mesh.

Table 4.2: Comparison among four hole-filling algorithms. n is the number of points in the point cloud, e is the number of edges in the polygonal model, and v is the number of voxels used for spatial subdivision.

Algorithm	Input	Properties	Cost
MLS	points	smooth transitions	$O(n \log n)$
MLS	mesh	smooth transitions	$O(e + mk)$
Davis et al.	partial meshes	non-smooth transitions	$O(v)$
Ju	polygonal mesh	minimal surfaces	$O(v)$
Liepa	oriented connected manifold meshes	minimal surfaces later smoothed	$O(e^3)$
Sharf et al.	points	surface features copied from vicinity	$O(v)$

4.3 Results

We have implemented the described algorithm and used it to fill holes in models of both real and synthetic objects. In all cases, the input to the algorithm consisted of unorganized point clouds. For real objects, we used datasets acquired with laser scanners. The point clouds for synthetic objects were obtained by rendering 3D models and saving the contents of the corresponding color and depth buffers. Such information was later used to reproject colored points in 3D. We used the surface reconstruction algorithm described by Gopi et al. [43] to create the initial triangle meshes and to remesh the model after new points were added. For the examples shown in this chapter, we used $\alpha = \frac{1}{16}$ (Equation 4.6). The experiments were performed on a 2.0 GHz Pentium 4 PC with 512 MB of memory. Table 4.1 presents some statistics and the running times for applying our algorithm to the examples shown in the chapter.

In order to illustrate the effectiveness of our algorithm, we used the techniques described by Wang et al. [134] to segment and reconstruct a chair model from range images acquired from a real environment (the UNC reading room). We then applied the proposed algorithm to the resulting model. Figure 4.6(a) shows the original samples of the chair

rendered as a triangle mesh. Notice the existence of a big hole. Figure 4.6(b) shows the reconstruction of the same chair model after the use of the symmetry-based techniques described by Wang et al. [134]. Many small holes are still left due to the nonexistence of data in either side of the symmetry plane and to the inability of the surface reconstruction algorithm used [43] to work in areas containing local variations of sampling density. Figure 4.6(c) shows the resulting chair model after applying our hole filling algorithm to the model shown in (b). Notice that the holes have been eliminated. The original incomplete chair model has 41,511 points and 82,065 triangles. Those numbers change to 64,159 points and 126,588 triangles after reconstruction using symmetry information [134]. The complete chair model (geometry and color), reconstructed with the hole-filling algorithm, has 65,114 vertices and 136,002 triangles, and was obtained in 10.93 seconds.

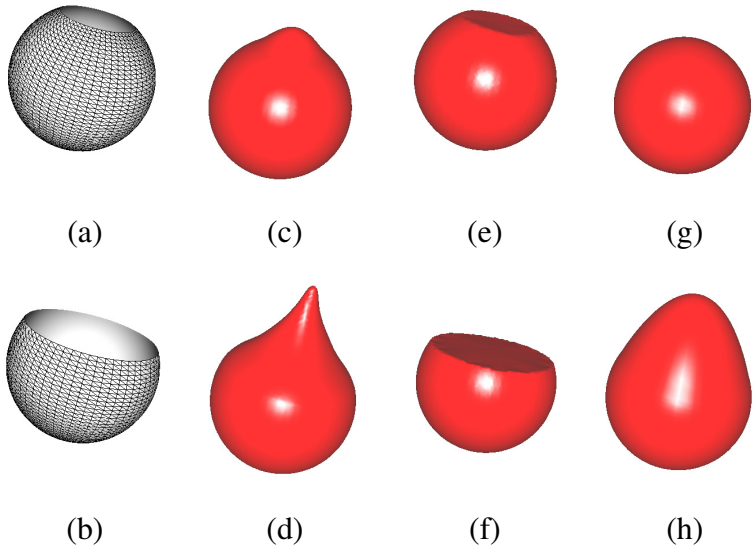


Figure 4.5: Reconstruction of the surfaces shown in using different algorithms: (a)-(b) the model with a hole of different sizes; (c)-(d): results produced by the algorithm of Davis et al. [31]; (e)-(f) results produced by the algorithm of Ju [57]; and (g)-(h) results produced by our MLS algorithm.

Figure 4.1 shows the three stages of the reconstruction of the chair model in a real environment. The remaining of the scene (walls, shelves, etc.) was also reconstructed

Table 4.3: Parameters used to reconstruct the models shown in Figure 4.5. The original sphere has radius equal to 5.0 units and the side of a voxel was set to 0.16 units.

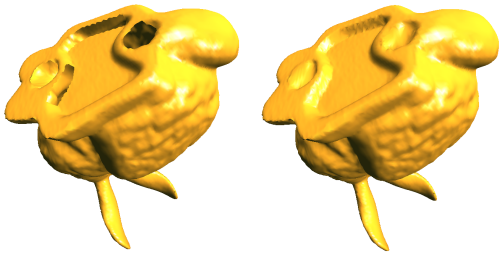
Algorithm / Model	Figure 4.5(a)(c)(e)(f)	Figure 4.5(b)(d)(f)(h)
Davis et al.	$73 \times 73 \times 104$ volume	$73 \times 73 \times 135$ volume
Ju	$64 \times 64 \times 64$ volume	$64 \times 64 \times 64$ volume
MLS	180 points	180 points

using the pipeline presented by Wang et al. [134]. The original reading room scene was edited by replacing the floor texture with a clear one and by repositioning the chair in order to emphasize the existence of holes in the model (Figures 4.1(a) and (b)).

(a) (b) (c)

Figure 4.6: UNC reading room armchair: (a) model reconstructed as a triangle mesh using only the original samples. Notice the large missing areas; (b) model obtained after processing the samples in (a) with our symmetry-based techniques [134]; and (c) final model without holes obtained by applying our hole filling algorithm to the model shown in (b).

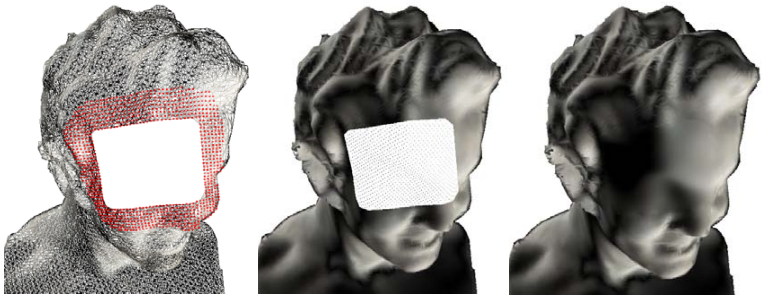
Figure 4.7(a) shows the Stanford bunny, which is known for containing a few holes in its bottom. The dataset consists of 35,947 points and 1,246 of these points were used as vicinity points for filling all the holes. Figure 4.7(b) shows the resulting model after hole filling. It contains 36,478 points and was obtained in 6.43 seconds. The previous two examples illustrate that our algorithm can be applied to both surfaces with boundaries as well as closed surfaces.



(a) (b)

Figure 4.7: The Stanford bunny: (a) the original model contains some holes; and (b) bunny model after hole filling performed with our algorithm.

The bust and angel models shown in Figures 4.8 and Figure 4.9, respectively, are synthetic models used illustrate the steps of the algorithm. In Figure 4.8(a), one sees the triangle mesh reconstructed from the point cloud. The highlighted points represent the vicinity of the hole, used as input for the MLS interpolation. Figure 4.8(b) shows the points resampled from the interpolated surface. The final model is shown in Figure 4.8(c). The larger reconstruction time for this model is as explained by the bigger number of resampled points (3,309), computed as the number of final point minus the number of original points (see Table 4.1).

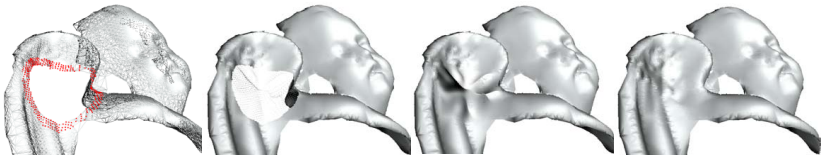


(a) (b) (c)

Figure 4.8: Bust with a hole in the head used to illustrate the steps of our algorithm: (a) triangle mesh with the hole and vicinity identified; (b) points resampled from the interpolated patch; and (c) final model after hole filling.

Figure 4.9(a) shows an angel model with a hole in one its wings. The vicinity points are highlighted to show the identified hole. Figure 4.9(b) shows the points resampled from the

patch that fills the hole. Figure 4.9(c) presents the reconstructed model, while Figure 4.9(d) shows the original model for comparison. Notice that although not the same as the original surface, the reconstructed patch is a plausible one.



(a) (b) (c) (d)

Figure 4.9: Angel with a hole on one wing: (a) triangle mesh highlighting the vicinity of the hole; (b) points added inside the hole; (c) reconstructed model after hole filling; and (d) the actual model for comparison.

Figure 4.10 illustrates the local nature of our algorithm. Figure 4.10(b) shows the Happy Buddha model after a few patches (highlighted in Figure 4.10(a)) have been removed, partially taking away some surface details. The locations where the patches were removed from were chosen to cover both low and high-frequency surface areas. The surfaces visible through the holes correspond to the back of the statue. The resulting model, after the five patches have been removed, contains 143,298 samples . The total number of vicinity points for the five holes is 1,154 samples and the number of resampled (new) points is 1,075. The time required to perform hole filling was 10.32 seconds. Figure 4.10(c) shows the result produced by our hole filling algorithm. Notice that such a reconstruction is quite plausible. Figure 4.10(d) displays the original model for comparison.

The cost of searching for boundary edges is linear in the number e of edges of the mesh. Once such an edge is found, tracing the boundary requires, in the worst case, following $e-1$ edges. Since each boundary edge has exactly two adjacent boundary edges (which can be identified by their shared vertices), the cost of automatically finding hole boundaries is $O(e)$.

The cost of the MLS fitting depends on the number k of new points to be added and

(a) (b) (c) (d)

Figure 4.10: Happy buddha: (a) holes are obtained by cutting off dark regions; (b) the model after hole cutting; (c) the model after hole filling performed by our algorithm; and (d) the original model for comparison.

on the size of vicinity, m, thus $O(mk)$. Therefore, the cost of the hole-filling algorithm is $O(e + mk)$. In practice, however, we observe that the running time of the algorithm is mostly influenced by the size of the vicinity and by the number of resampled points used to fill the holes, as can be seen in Table 4.1 (for instance, compare the running times for the Bust and Buddha models). If a point cloud is provided as input, the cost of creating a mesh from the point cloud, $O(n \log n)$ on the number of points [43], needs to be added to the total cost of the algorithm.

4.4 Discussion and Comparisons

The approaches presented by Davis et al. [31] and by Ju [57] use contouring algorithms to reconstruct polygonal models from intermediate voxel-based representations. Thus, unlike in our approach, the resulting models do not preserve the original meshes. Ju's approach [57] fills holes by reconstructing minimal surfaces, which tend to blend poorly with the original meshes as the sizes of the holes increase. Essentially, hole-filling techniques based on minimal surfaces only perform well on very small holes or when the surface is locally flat. Liepa's technique [71] uses an $O(n^3)$ algorithm for filling holes with minimal-surface patches. These are later processed to produce smoother surfaces. The approach presented by Sharf et al. [114] is intended for filling holes on highly-complex geometric regions and, therefore, is target toward a different domain than ours. Table 4.2 summarizes the major features of all these algorithms, including input type, properties of the interpolated patches, and cost.

Figures 4.5(a) and (b) shows two smooth surfaces with boundaries obtained after removing caps of different sizes from the surface of a sphere. The surface on the left contains 3,001 vertices and 5,940 triangles, while the one on the right contains 2,401 vertices and 4,740 triangles. These surfaces were used to compare how smoothly the patches reconstructed by the different techniques blend with the original meshes. A sphere was chosen as a reference shape because of its smoothness and symmetry, and because one has a clear idea about what to expect from an exact reconstruction.

We reconstructed the surfaces shown in Figure 4.5 using the techniques by Davis et al. [31], Ju [57], and our MLS approach. Davis's et al. and Ju's techniques were chosen because they are representatives of the state-of-the-art in hole filling and their source codes are available on the web. Table 4.3 shows the parameters used by the different algorithms.

Table 4.4: Number of vertices and triangles in the models produced by each technique. The percentages are computed with respect to the corresponding values in the input models.

Algorithm	Figure 4.5(a)(c)(e)(g)		Figure 4.5(b)(d)(f)(h)	
	♯ of vertices	♯ of triangles	♯ of vertices	♯ of triangles
Input	3,001 (100.0%)	5,940 (100.0%)	2,401 (100.0%)	4,740 (100.0%)
Davis et al.	10,323 (343.9%)	20,904 (351.9%)	10,454 (435.4%)	20,904 (441.0%)
Ju	6,858 (228.5%)	13,712 (230.8%)	5,978 (248.9%)	11,952 (252.1%)
MLS	3,542 (118.0%)	7,080 (119.2%)	3,672 (152.9%)	7,098 (149.7%)

In order to use Davis's et al. code, the triangle meshes were first converted to the PLY format. The resolution of the voxel space was defined experimentally by setting the voxel size to 0.16 in the Ply2Vri program (73 $\approx$ 10.0/0.16 + 10 fringe voxels). The third dimension of the voxel space was chosen so that the resulting surface could fit inside the volume.

Figure 4.5 shows the reconstructed results produced by the three algorithms for the surfaces shown in Figure 4.5. Figures 4.5 (c), (e) and (g) were reconstructed from the surface shown in Figure 4.5 (a), whereas Figures 4.5 (d), (f) and (h) are reconstructions of the surface shown in Figure 4.5 (b). The results shown in the second column of Figure 4.5 were produced by the algorithm of Davis et at [31]. Notice the existence of some protrusions on the reconstructed patches, which become more evident as the size of the hole increases. The abrupt cut on the largest highlight visible in Figure 4.5(d) indicates that: (i) the reconstructed patch does not smoothly blend into the original mesh and (ii) the original mesh was not preserved. The results produced by Ju's technique [57] are shown in Figures 4.5(e) and (f). Notice the minimal surfaces reconstructed by the technique, showing that it can only be used for filling small holes.

Figures 4.5(g) and (h) show the results produced by the MLS technique using 180 samples in the vicinity of each hole. The results are quite smooth, with the reconstructed patches naturally blending themselves into the original meshes. In Figure 4.5(e), the sphere was very satisfactorily recovered. In the case of the bigger hole, the algorithm reconstructed

an egg-like shape, which is a plausible solution for the ill-posed problem of surface interpolation. The extended highlight indicates the smooth blending of the reconstructed patch with the original mesh. Table 4.4 shows the numbers of vertices and triangles produced by the three algorithms. The percentage values were computed with respect to the corresponding numbers in the input models. The meshes obtained with the algorithm by Davis et al. [31] present the largest numbers of primitives, followed by the meshes produced by Ju's algorithm [57]. This is due to the fact that these meshes are extracted using contouring techniques [58, 73]. As a result, the number of resulting polygons depends on the resolution of the voxel space used, and not on the number of original vertices. Notice the significant increase on the numbers of vertices and triangles in the models produced by these techniques, even though a relatively low-resolution voxel space has been used (see Table 4.3) and the space is mostly empty. In these techniques, the entire model, and not only the interpolating patch, is extracted at voxel-space resolution.

An important feature of our algorithm is that the shape of the resulting patches can be reasonably predicted by the vicinities of the holes. For example, as we project the vertices belonging to the vicinity of the hole shown in Figure 4.5 (b) onto the reference plane (the cutting plane for the missing cap), the distances between any two pairs of projected (U, V) coordinates will be smaller than the difference between their corresponding heights (the S coordinates). As a result, $abs(\frac{\partial S}{\partial U}) > 1.0$ and $abs(\frac{\partial S}{\partial V}) > 1.0$, producing the resulting egg-like shape.

Our algorithm might fail if the vicinity region presents folds or twists, which will not define a one-to-one mapping when projected onto the reference plane. In this case, that part of the hole might not be filled properly. When the size of a hole increases, the possibility of facing such a situation tends to grow. This problem could be addressed by, instead of a plane, using a curved domain for projecting the vicinity points onto.

Chapter 5

Noise Tolerant Surface Reconstruction

The problem of surface reconstruction from unorganized point clouds is challenging and ill-posed. Moreover, in range scans acquired from real objects, noise tends to contaminate the data, making the reconstruction task harder. Whenever it is difficult to accurately estimate normals at the samples (*e.g.*, in noisy datasets), existing implicit methods tend to fail to create distance fields that correctly represent surface details and topology.

Figure 5.1: Reconstruction of David's head using our algorithm.

One way is to assume that the surface to be reconstructed are 2-manifolds of piecewise C^1 continuity, with isolated small irregular regions of high curvatures, sophisticated local

topology or abrupt burst of noise. At each sample point, a quadric field is locally fitted quadric fields vis modified moving least squares method moving least squares method [69]. These locally fitted quadric fields are then blended together to produce a pseudo-signed distance field using Shepard's method [116]. We introduce a prioritized front growing scheme in the process of local quadrics fitting. Flatter surface areas tend to grow faster. The already fitted regions will subsequently guide the fitting of those irregular regions in their neighborhood. We refer interested readers to [144] for the details.

As an alternative, we introduce, in this chapter, a new implicit method that only requires information about the positions of points and, therefore, can be used for surface reconstruction from both clean and noisy datasets. Our method is based on two key observations: (1) a highly accurate distance field is not necessary for reconstruction when the data is quite noisy. In these cases, it is sufficient to have an approximate smooth distance field whose error near the surface is bounded by the amount of noise in the data. (2) Assuming the surface to be reconstructed is closed, one can use a bounding box containing the point cloud to orient the surface (*i.e.*, to define its exterior). Based on these observations, we approximate the distance field using an adaptive and hierarchical framework. Although no formal proofs are presented, we provide some intuitive evidence that the proposed approach guarantees the orientability of the reconstructed surfaces, preserves topology, and that the reconstruction error is bounded by the amount of noise and by the sample density of the input point cloud.

The contributions of this chapter include:

- An improved implicit method for surface reconstruction from point clouds that only requires information about the positions of the samples and is robust to the presence of noise and missing data (Sections 5.1 and 5.2).

- A new algorithm for computing approximate distance fields based on oriented

charges (Section 5.1);

- A hierarchical and adaptive representation for performing surface reconstruction at different levels of detail as well as hole filling (Section 5.5).

The idea of oriented charges was inspired by the notion of *oriented particle systems* [119], but it distinguishes itself from the latter, which has no relation to distance fields. In our approach, a global distance field is formed by blending local distance fields of nearby oriented charges. Our approach also differs from Ohtake's algorithm [91] because we compute the implicit function based only on the location of points while Othake's approach also uses information about normals. Moreover, our approach is less sensitive to noise, producing reasonable results even for very noisy datasets.

5.1 Oriented Charges

An *oriented charge* $Q_i(p_i, q_i, n_i)$ (OC) is a charge instantiated at a point p_i, with magnitude q_i and orientation n_i, where $\|n_i\| = 1$. It implies a linear local distance field at position x given by

$$F_i(x) = q_i - (x - p_i) \cdot n_i \tag{5.1}$$

Let O be an octree defining a subdivision of the space containing a point cloud. The adaptive subdivision of the octree is trigged by the number of points inside a given node, up to a maximum specified octree level. In our current implementation, we recursively subdivide an octree node until it contains no more than six samples. In order to approximate the distance field induced by the surface represented by the point cloud, a set of oriented charges are instantiated at the neighborhood of the nodes containing the original samples (Section 5.2).

Each charge Q_i has an associated linear distance field, whose iso-surfaces are the set

of parallel planes perpendicular to n_i. This can be easily seen by rewriting Equation 5.1 as $F_i(x) = q_i - \alpha$, where $\alpha = (x - p_i) \cdot n_i$. Although, according to Equation 5.1, F_i can assume arbitrarily large values both in the positive and negative ranges, the local distance field of an oriented charge is made effective only within a $3t \times 3t \times 3t$ neighborhood centered at the oriented charge itself. Here, t is the side of the associated octree node. To combine overlapping distance fields into a global one, we use a scaled version of a Gaussian function as the weight for each oriented charge Q_i

$$\omega_i(x) = 2^l e^{-\frac{\|p_i - x\|^2}{\sigma_l^2}} \tag{5.2}$$

where l is the level of its associated octree node and σ_l varies with the size of that octree node (in our experiments we used $\sigma_l = 1.5t$). This weight function was designed such that oriented charges at coarser levels have larger areas of influence than at finer levels. The field induced by multiple oriented charges is a weighted average of Equation 5.1:

$$F(x) = \frac{\sum_i F_i(x) * \omega_i(x)}{\sum_i \omega_i(x)} \tag{5.3}$$

Given an octree O enclosing a set of samples in 3D, its nodes can be classified as either *exterior*, *interior*, *boundary*, *hole* or *unknown* nodes. The notion of octree-node inside-outside classification was first introduced by Samet [108] and has also been recently used to support CSG operations in point-based graphics [1].

Initially, each octree node is labeled either as a boundary or as an unknown node, depending on whether it contains some or no samples, respectively. After that, all nodes of the octree are traversed and once an unknown node is found, it is re-labeled as either exterior, interior or hole node. Exterior/interior nodes are outside/inside the object (*i.e.*, outside/inside the object's closed surface representation). Boundary nodes and hole nodes

intersect the zero-set surface of the object. We use the term *crust* to refer to a closed thin shell of octree nodes, all at the same level of the octree. The *surface crust* is composed only of boundary and hole nodes. The *exterior* and *interior crusts*, on the other hand, only have exterior and interior nodes, respectively, and together tightly bound the surface crust. *Oriented charges are only instantiated in the exterior and interior crusts*.

5.2 Surface Representation with Oriented Charges

In order to identify the exterior and interior crusts where oriented charges will be instantiated, we first find two crusts which are absolutely outside and inside the surface and refer to them as the *exterior* and *interior frontiers*, respectively. The propagation of frontiers is based on an adaptive refinement approach that re-labels unknown nodes and finally finds the surface, the exterior and the interior crusts (Figure 5.2 and Section 5.2.2).

The algorithm for instantiating oriented charges is shown in Algorithm 5.1. Starting from a coarser to a finer level of the octree, first the exterior and interior frontiers are identified (Section 5.2.1). These frontiers are propagated to compute the crusts at that given level of the octree (Section 5.2.3). The resulting information is then used as input to the subsequent finer level. Once we reach the finest level, oriented charges are instantiated at the exterior and interior crusts.

for *octree_level= 1 to MAX* **do**
    Find the exterior and interior frontiers for the current level;
    Propagate the frontiers to find the exterior and interior crusts;
end
Instantiate OCs in the exterior and interior crusts;

Algorithm 5.1: Instantiation of oriented charges

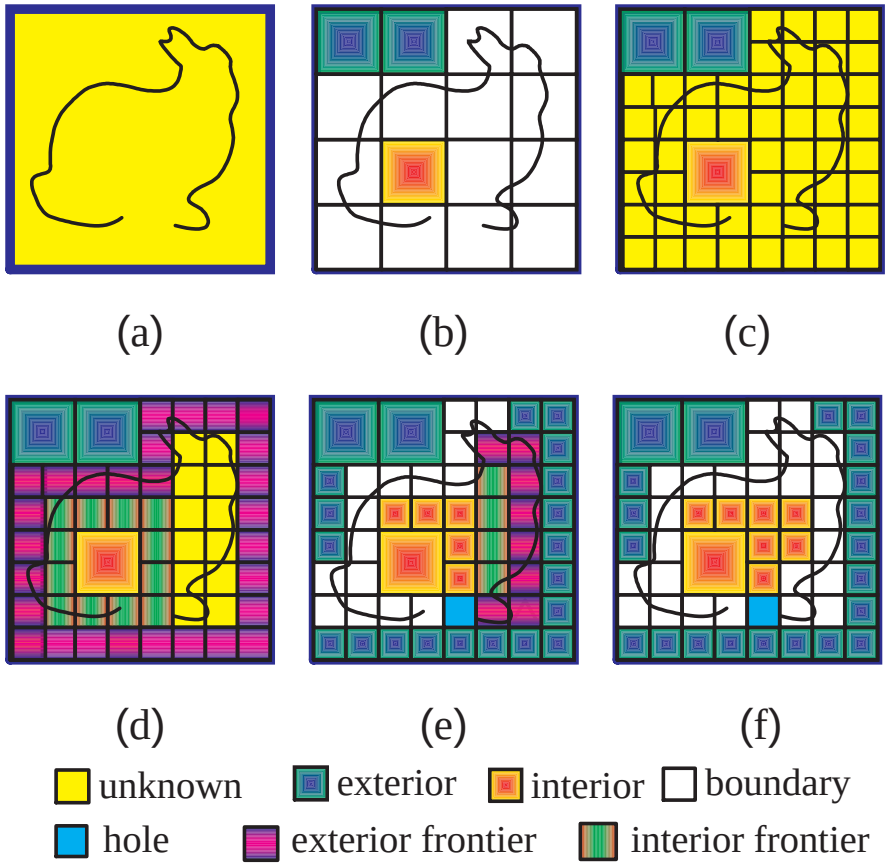


Figure 5.2: Propagation stages of the frontiers: (a) initial exterior frontier; (b) exterior and surface crusts at the coarsest one. Remaining unknown nodes set as inside; (c) boundary nodes are subdivided and labeled as unknown; (d) new exterior/interior frontiers; and (f) final classification.

5.2.1 Finding and Propagating Frontiers

We revise the tagging algorithm of Zhao et al. [151] to find the exterior and interior surface crusts. We make the original algorithm adaptive to sampling density and progressively enlarge the trusted exterior/interior regions. The algorithm assumes that the outer nodes of the octree contain no samples and, therefore, can be safely set as exterior nodes (Figure 5.2(a)). Starting at a coarser level, the exterior frontier is propagated until nodes containing at least one sample are reached. These nodes are labeled as boundary nodes.

The remaining empty nodes (*i.e.*, that could not be reached from the exterior frontier) are then labeled as interior nodes (Figures 5.2(b)). At this point, the algorithm refines this initial tagging (Figures 5.2 (c) to (f)).

During a node refinement, the children of an exterior/interior/hole node are also labeled as exterior/interior/hole nodes. The children of a boundary node, however, can become exterior, interior, boundary or hole nodes. Thus, when all the nodes at a coarser level have been labeled, at the next level the algorithm only needs to classify the children of boundary nodes. Initially, we mark all children of boundary nodes as unknown nodes (Figure 5.2(c)). The refinement proceeds using the children of the of previous-level exterior/interior frontier as the new exterior/interior frontier (Figure 5.2(d)). The final result is illustrated in Figure 5.2(f).

5.2.2 The Crusts

The next step is to find the interior, exterior and surface crusts. First, we compute the average location of samples inside each boundary node. Then, we compute the Euclidean distances between the centers of exterior/interior frontier nodes and the average locations of the samples in the boundary nodes under their radius of influence (a $3\times3\times3$ neighborhood). We record the minimal distance in each frontier node. Based on their minimal distances, nodes in the frontiers are heap sorted. For each iteration, the node at the top of the heap (*i.e.*, with the biggest distance) is taken out and processed as follows. If this node has the smallest distance compared to its adjacent unknown nodes, it will be marked as a hole node. Otherwise, it is discarded and all its adjacent unknown nodes are inserted into the heap. When the iteration ends, boundary and hole nodes compose the surface crust and exterior/interior nodes touching the boundary/hole nodes become the exterior/interior crust (Figure 5.2(e) and (f)).

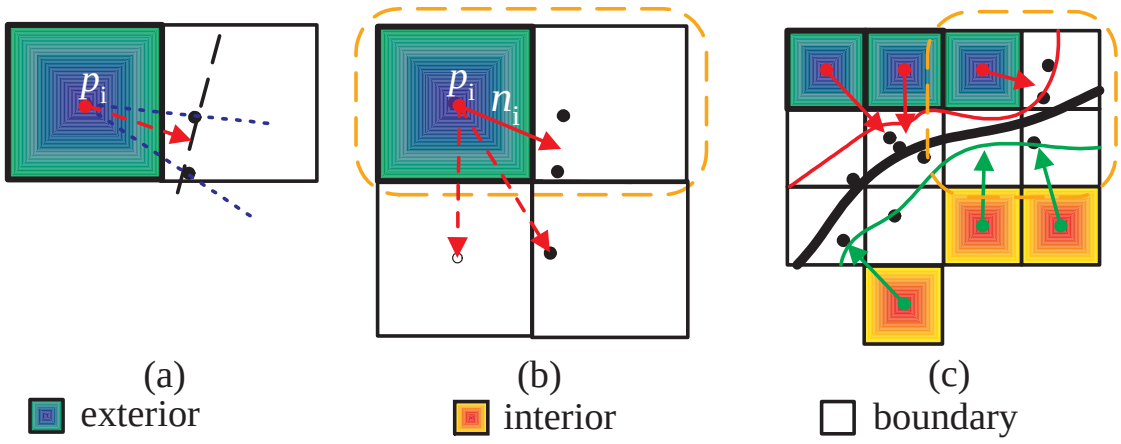


Figure 5.3: Creating the oriented charge: (a) a candidate oriented charge; (b) the candidate with smallest magnitude is selected; and (c) the global distance field (the thick black curve) is created combining the distance fields of the exterior and interior crusts.

5.2.3 Instantiating Oriented Charges

One oriented charge Q_i is instantiated at the centers of each node in the exterior and the interior crusts. For a given node N_i in the exterior/interior crust, the final magnitude, q_i, and orientation, n_i, of Q_i is defined by the set S of surface-crust nodes in the $3 \times 3 \times 3$ neighborhood of N_i. First, for each node s_k in S, we compute a candidate oriented charge cQ_j for N_i. The orientation of cQ_j is computed as the centerline of the cone with apex at p_i (the center of N_i) that bounds all samples in s_k. This is illustrated in Figure 5.3 (a). The magnitude cq_j of cQ_j is computed as

$$\min_{cq_j} |cq_j| \tag{5.4}$$

such that $(cq_j - (p_i - d_k)) \cdot n_j \geq 0$ holds for at least $\gamma\%$ of the d_k samples inside s_k. γ is a parameter to counteract the effect of noise and outliers. Intuitively, Equation 5.4 finds, for each external and internal crust node, the closest plane perpendicular to n_j that separates the samples inside s_k into two groups, such that at least $\gamma\%$ of the points lie on

one side of the plane. We obtain such a plane using linear programming. Then, the tuple (p_i, cq_i, n_i) with the smallest $|cq_i|$ is selected as the oriented charge Q_i for N_i. The sign of cq_i is set to "+" or "-", depending on whether the node is in the exterior or in the interior crust, respectively. If a node s_k in S is a hole node (*i.e.*, it contains no samples), a virtual sample is instantiated at the center of s_k before computing cq_i and n_i using the procedure just described. This situation is illustrated in Figure 5.3(b), where a virtual sample has been instantiated at the center of the bottom left surface crust.

To create oriented charges at a coarser level of the octree, the oriented charges in adjacent nodes are considered. Suppose we have all oriented charges at a finer level. For every such oriented charge Q_i, a virtual point is computed as located at $p_i - q_i n_i$. To compute oriented charges at the coarser level, we collect virtual points instead of real points within its neighborhood before Equation 5.4 is applied. This bottom-up process computes oriented charges from the finest level to the coarsest level.

5.3 OC Properties

Surface Orientability Preservation. Since the exterior/interior crust bounds the surface of the object tightly, it represents a manifold itself. Besides, all oriented charges in the exterior/interior crust are pointing to the object's surface inwardly/outwardly. Thus, the exterior/interior distance field has correct orientability. The associated gradient field is nonzero everywhere in the surface crust (the internal and external distance fields have opposite signs: negative and positive, respectively) having a direction from outside to inside. When weighted averaging the exterior and interior distance fields, the resulting gradient field is nonzero everywhere on the surface crust. Therefore, inside the region of interest, the resulting distance field has no twists or singular points, and the reconstructed surface is well oriented.

Topology Preservation. Given that the original surface has been sampled at the appropriate sampling rate, the topology of the surface is preserved. The thickness of the thin volume composed by the exterior, interior and surface crusts decreases as the octree nodes are subdivided. When the thickness is small enough, the manifold represented by the thin volume has the same topology as the surface. Since the reconstructed surface lies within the thin volume, it will approximate the surface accurately (*e.g.*, showing the correct connection and genus). The assumption that surfaces are sampled at appropriate rates guarantees correct topology for clean datasets. In the case of noisy ones, we are limited by the amount of noise relatively to the sizes of the local structures of the objects. Thus, small (relative to the amount of noise) structures may not be correctly reconstructed. This situation is illustrated in Figure 5.8(d), where the small holes under buddha's arms have collapsed due to the presence of large amounts of noise.

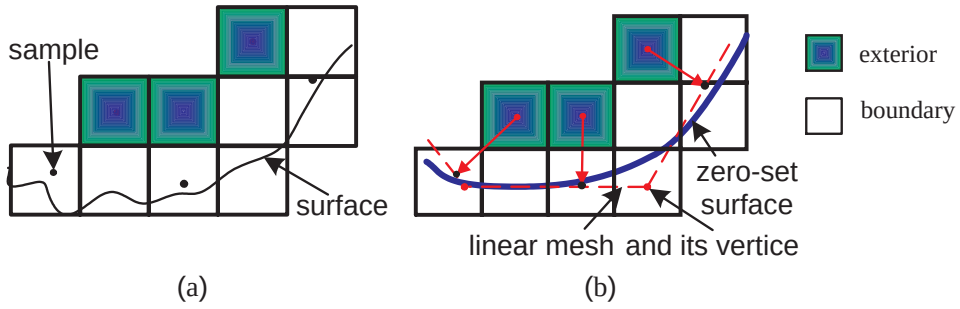


Figure 5.4: Comparison of the actual surface and the reconstructed surface: (a) the actual surface and samples; and (b) the reconstructed surface.

Error Bounding. A given oriented charge induces a local linear distance field, whose zero-set surface is an infinite plane. When weighted averaging the influence of two adjacent oriented charges, the intersection line of two zero-set planes is part of the resulting zero-set surface. To estimate the error bound, the zero-set surfaces of the exterior and interior distance fields are approximated by piecewise meshes (shown in 2D in Figure 5.4). Let τ be the error bound in the input samples. Because the influential region of oriented charge

is limited within a $3 \times 3 \times 3$ subvolume, the error bound on the reconstructed surface is $\tau + \frac{3\sqrt{3}}{2}t$ (*i.e.,* τ plus half of the diagonal of the $3 \times 3 \times 3$ subvolume), where t is the size of the octree's leaf nodes.

5.4 Iso-Surface Extraction and Cost Analysis

Iso-surface extraction is obtained using a variant of the Marching Cubes algorithm [20]. To efficiently improve the accuracy, the size of the marching cubes is made compatible to the smallest node of the octree. Since the details of the surfaces are mainly captured by the oriented charges at the octree's leaf nodes, we use all oriented charges at neighboring nodes at the same and direct coarser levels to compute the corresponding distance fields using Equation 5.3. While the traditional Marching Cubes algorithm tends to unnecessarily reconstruct many small triangles over flat areas, an adaptive solution could be used to avoid this problem [111].

5.4.1 Mesh Refinement

Since oriented charges define piecewise linear implicit functions, they may lead to coarse reconstruction in regions with high curvatures. Jeong et al. [55] use subdivision surfaces with adaptive sampling to improve a coarse model. We, instead, address this problem with a mesh-refinement step that follows surface extraction. Since the reconstructed model is already quite close to the real surface, one can move vertices of the mesh along their normals to make them closer to their nearest samples (Figure 5.5). Notice these normals are computed at the vertices, after mesh extraction and are not used for computing the distance field. The range of adjustment is limited by the size of the finest octree node to avoid mesh self-intersection.

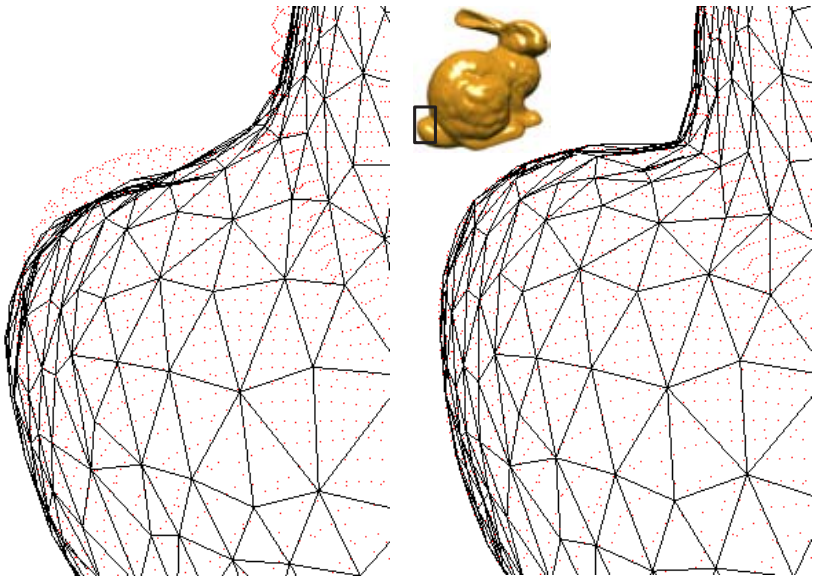


Figure 5.5: Mesh refinement: (a) mesh extracted by the Marching Cubes algorithm. Small dots are original samples; and (b) final mesh after vertex displacement.

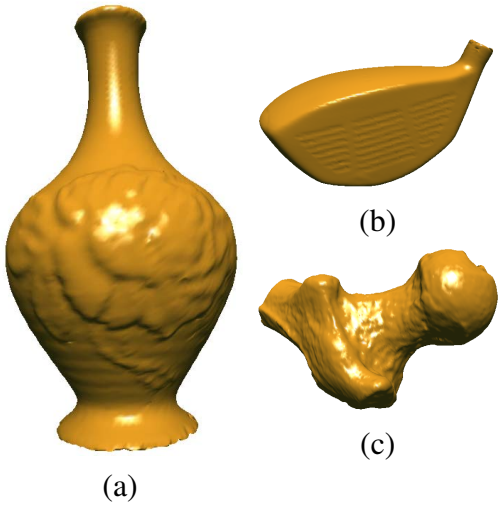


Figure 5.6: Reconstruction of (a) a vase; (b) a golf club; and (c) a balljoint, using our algorithm.

5.4.2 Cost Analysis

Let n be the number of samples in a given point cloud and assume that each octree leaf node contains at most one sample. Since each octree node, except for the first level, must have a parent node, the total number of occupied cells that need to be stored can be expressed as $\Sigma = n + n/8 + n/64 + ... + 1$. Σ converges to $(8/7)n$. In our current implementation, for each occupied node we currently store links to its 26 neighbors, making the total memory requirements of the algorithm $27(8/7)n \approx 31n = O(n)$. The cost is linear on the number of samples and the big constant is due to the data structure we used, not inherent to the algorithm itself.

The tagging procedure is based on heap sort and has cost $O(n \log n)$. Since surface extraction is done locally at octree neighborhoods, each evaluation of the distance field has cost $O(1)$. Thus, the total time complexity of the algorithm is $\Omega(n \log n)$ and its upper bound will depend on the level of refinement of the octree (*i.e.*, on the number of times the implicit function is evaluated during surface extraction).

5.5 Results

We have tested our algorithms on a series of clean and noisy datasets, including range scans of real objects. Because it is hard to tell the amount of noise present on such range scans, in order to keep control of the amount of noise in each case, we have added noise to originally "clean" models. For each model, the bound to the perturbation added to the position of its original samples is expressed as a fraction of the largest size of the bounding box that contains the point cloud (Table 5.1). No scaling was applied to the input data. Except when explicitly stated otherwise, reconstruction was performed at level 9 of the octree. The relative error bounds shown in Table 5.1 were obtained by dividing each error

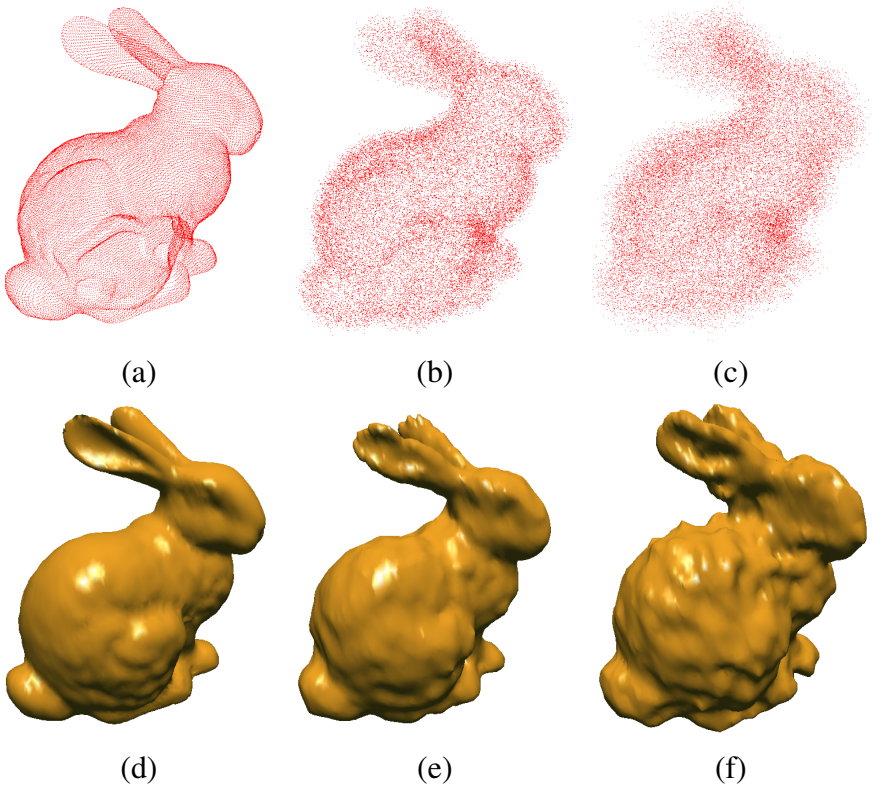


Figure 5.7: Bunny with noise bounded by (a) 0; (b) 1/60; or (c) 1/30 of the largest side. Reconstructions: (d); (e); or (f), respectively.

bound by the smallest dimension of the corresponding dataset's bounding box. Entries with an asterisk indicate that the input data were obtained from range scans (no precise information about the amount of noisy was available - N/A).

Figures 5.7 and 5.10(a) show several reconstruction results for the Stanford bunny with different levels of noise. Figure 5.10(a) also shows the bunny model, this time reconstructed after merging ten range scans containing noise, registration errors and outliers. Note the quality of the reconstructed model. Figure 5.10(b) shows the dragon model reconstructed from a set of 60 range scans also containing noise, outliers and registration errors. These examples demonstrate the effectiveness of the proposed technique to handle data captured from the real world.

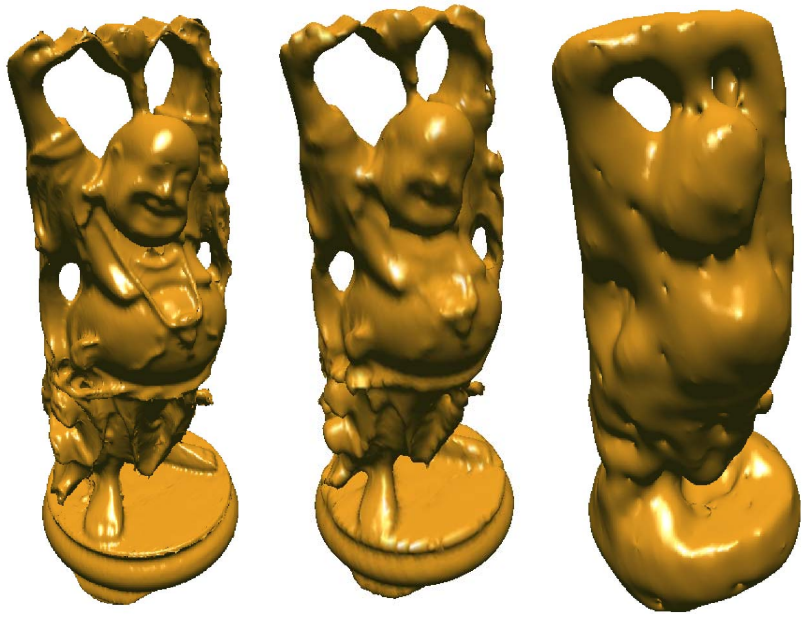
Figures 5.1, 5.6, 5.8(a), and 5.9(a) show examples of surface reconstruction from

clean datasets using the proposed technique. David's head (Figure 5.1), the dragon (Figure 5.9(a)) and the buddha (Figure 5.8(a)) illustrate the case of surface reconstruction involving large datasets. Table 5.1 provides some statistics associated with the models. The measurements were performed on a Pentium 4 2.2GHz PC with 1GB of memory. The clean models show a bound on the relative error varying from 0.2% to 0.5%. This error varies with the size of the octree leaves and can be further reduced with extra refinement of the octree. For the very noisy example shown in Figure 5.7(f), the relative error bound is 3.9%.

Although, in practice, it may not be desirable to perform surface reconstruction of very noisy datasets (without previous smoothing), we have included some examples to illustrate the robustness of our approach in the presence of noise (Figures 5.7(f), 5.8(c), and 5.9(b)). The hierarchical representation of oriented charges provides a natural way for creating smoother versions from both clean and noisy datasets. This is illustrated in Figure 5.8(b), where the Buddha model has been reconstructed using a coarser level of the octree. This is similar to applying a low-pass filter to the geometric information [92]. However, when a very coarse level is used for reconstruction, some dimples may be created (Figure 5.8(c)), as some mesh vertices may be moved too far by the vertex displacement procedure.

We have compared the results of our algorithm with the ones produced by the MPU approach [91]. For noiseless datasets, the visual quality of the reconstructed models are similar, although MPU does a better job in reconstructing sharp features. This situation is illustrated on the left part of Figure 5.9, where the image at the top was produced using our algorithm and the one at the bottom was created using Ohtake's own code. For the dragon example, MPU's running time was 93 seconds, while our algorithm took 119 seconds.

Our approach is, however, more general in the sense that it handles datasets with minimal information and is considerably more robust to noise. In order to verify this, we added noise bounded by 1% to both the bunny and the dragon point clouds and tried to



(a) (b) (c)

Figure 5.8: The reconstructed buddha model: (a) no noise (level 9); (b) no noise (level 7); and (c) with noise bounded by 1/50 of the bounding box size (level 9).

reconstruct the resulting models using MPU and our approach. Since MPU requires normals, these were computed using Hoppe's normal estimation algorithm [52], considering different numbers of neighbor samples. Figure 5.11 illustrates the normals obtained using a neighborhood of ten samples for the bunny containing no and 1% noise. For these noisy datasets with estimated normals, MPU was unable to reconstruct the models. The RBF-based technique described by Tobor et al. [125] was also unable to perform the reconstruction. Figure 5.9(b) shows the reconstructed dragon model produced by our algorithm and illustrates its robustness to the presence of noise. Figure 5.9(d) shows the result produced by MPU when the noisy dragon dataset and the normals of the clean model were given as input. For the noisy dragon example, MPU's running time was 466 seconds while our algorithm took only 272 seconds.

WE HAVE ALSO RECONSTRUCTED THE NOISY (1% NOISE) BUNNY MODEL USING A VARIANT [82] OF THE BALL-PIVOTING ALGORITHM [13] FOR COMPARISON. WE USE

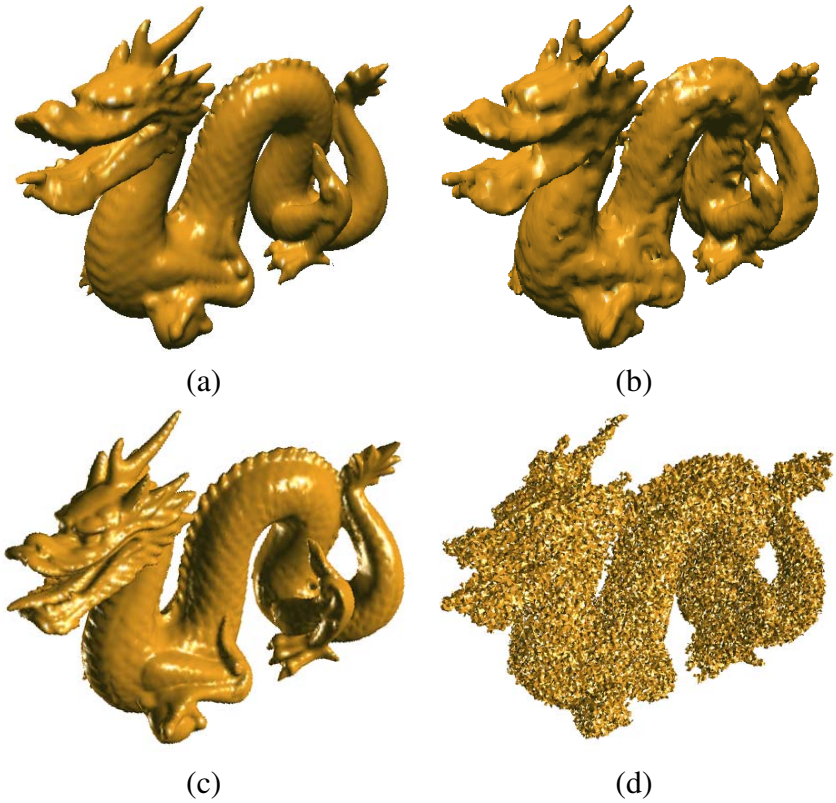


Figure 5.9: Dragon model reconstructed using our approach ((a) and (b)) and MPU ((c) and (d)). Datasets: clean ((a) and (c)) and 1% noise ((b) and (d)).

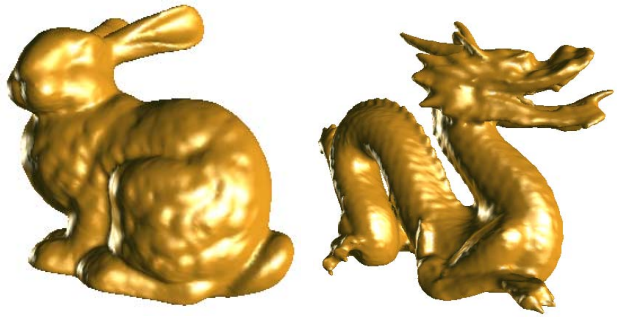
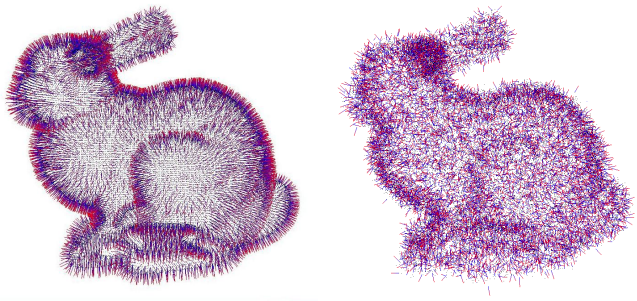


Figure 5.10: Models reconstructed by our algorithm from actual range scans.

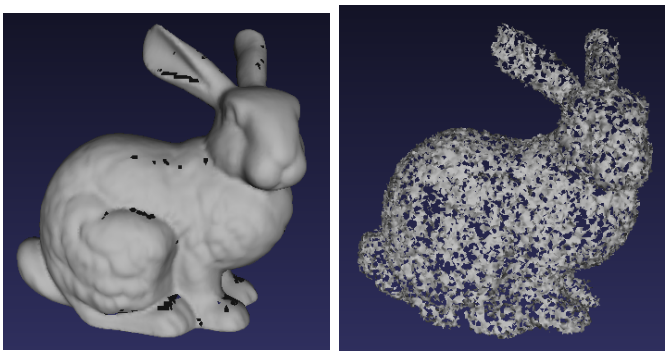
THE "AUTOGUESS" MODE FOR THE BALL SIZE IN THE EXPERIMENTS. THE RESULTS ARE SHOWN IN FIGURE 5.12. THE RECONSTRUCTED MODEL OF THE CLEAN DATASET IS GOOD EXCEPT FOR SOME SMALL HOLES IN HIGH CURVATURE REGIONS. IN COMPARISON, THE RECONSTRUCTION OF THE NOISY DATASET HAS NUMEROUS HOLES. IT

SHOWS THAT THE BALL-PIVOTING ALGORITHM DOES NOT WORKS WELL FOR VERY NOISY DATASETS OR DATASETS WITH SIGNIFICANT CHANGES OF SAMPLING RATES.



(a) (b)

Figure 5.11: Bunny normals computed from: (a) a clean point cloud and; (b) a 1% noisy point cloud.



(a) (b)

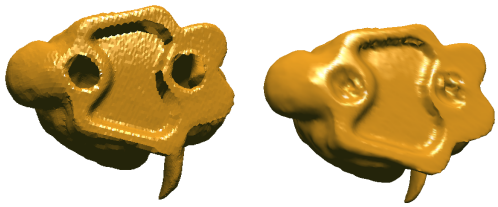
Figure 5.12: Bunny models reconstructed using a variant of the ball-pivoting algorithm from: (a) a clean point cloud and; (b) a 1% noisy point cloud.

The implicit function induced by oriented charges provides a natural way for filling small holes in the model. The hierarchical nature of OC's allows for the filling of larger ones. This situation is illustrated in Figure 5.13. Due to the adaptive subdivision of the octree, nodes inside large holes tend to be at much coarser levels than nodes on the boundary crust. Because the tips of the vectors representing the oriented charges covering such regions will be at the centers of these larger octree nodes, this tends to introduce discrepancies around the boundary of large holes. Therefore, the reconstructed surface for relatively

Table 5.1: Running times and error bounds for various models. Times are measured in seconds and the number of points are shown in thousands.

Model	# of points	Noise	Time OC	Time mesh	Error bound
Balljoint	137.06	0	8.42	86.80	0.5%
Buddha	543.65	0	12.32	120.32	0.5%
Buddha	543.65	1/50	12.52	123.32	1.5%
Bunny	34.83	0	10.74	90.80	0.5%
Bunny	34.83	1/30	10.66	90.32	3.9%
Merged Bunny	362.27	N/A	11.28	91.32	N/A
Dragon	437.64	0	11.04	107.22	0.5%
Dragon	437.64	1/100	11.21	108.43	1.5%
Merged Dragon	1,769.51	N/A	39.28	233.19	N/A
Golf club	209.77	0	8.74	82.80	0.2%
Head	394.18	0	16.32	194.32	0.5%
Vase	68.09	0	7.32	69.77	0.2%

large holes might present some small bumps, as illustrated in Figure 5.13(b).



(a) (b)

Figure 5.13: Hole filling: (a) holes at the bottom of the Bunny; and (b) bunny model with holes filled.

5.6 Discussion

In our current implementation, the evaluation of the distance field slows down the reconstruction process. This can be improved using either an adaptive polygonizer or a field pre-evaluation approach similar to the fast multipole method [26].

When surface features are smaller than the size of the smallest octree nodes, they may not be appropriately reconstructed. The solution, in this case, is to further subdivide some of the octree cells.

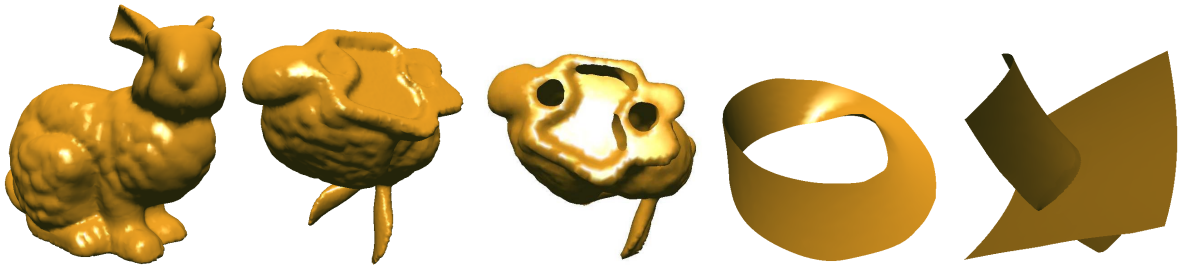
THE CREATION OF ORIENTED CHARGES RELY ON THE ACCURATE CLASSIFICATION OF EXTERIOR/INTERIOR OCTREE NODES. FOR THIS REASON, THE PROPOSED FRAMEWORK CAN NOT BE USED TO RECONSTRUCT A NON-CLOSED SURFACE, FOR EXAMPLE THE INDOOR SCENE MODEL IN FIGURE 3.13. APPARENTLY, IF THE POINT CLOUD IS EQUIPPED WITH POINT NORMALS, THEY BE USED TO HELP DISTINGUISH THE EXTERIOR/INTERIOR. HOWEVER, THE PROPOSED ALGORITHM ASSUMES THAT THE INPUT HAS MINIMAL INFORMATION AND DOES NOT TAKE ADVANTAGE OF ADDITIONAL INFORMATION SUCH AS NORMALS.

The local nature of oriented charges and of the evaluation process used for reconstruction suggests that the entire procedure (tagging, instantiation of oriented charges and surface extraction) can be performed on the fly for small neighborhoods. This should allow the reconstruction of arbitrarily large models with fine details, using out-of-core techniques.

Chapter 6

Non-Manifold Surface Reconstruction

Surface reconstruction from unorganized points has many practical applications ranging from reverse engineering [130], entertainment, and analysis of forensic records, to digitization of cultural heritage [70] and creation of virtual museums. This subject has gotten considerable attention in recent years due to the increasing availability of 3D scanning devices, which are capable of sampling complex geometric objects at very high resolutions. Surface reconstruction from unorganized point clouds is, however, a challenging and ill-posed problem and although much progress has been made in the last few years [7, 26, 52, 91],



(a) (b) (c) (d) (e)

Figure 6.1: Examples of surfaces with different topologies reconstructed with our algorithm. Here a single framework handles all cases: (a) a view of the Stanford bunny; (b) surface reconstruction with hole filling; (c) surface reconstruction with boundary preservation; (d) non-orientable surface; and (e) non-manifold surface.

currently no single approach can consistently handle all possible surface configurations. Moreover, essentially all approaches seem to assume that point clouds represent the surfaces of solids, and very little attention has been given to reconstructing surfaces with boundaries or non-orientable surfaces [2].

In this chapter, we present a novel approach for surface reconstruction from unorganized point clouds. Our approach is general in the sense that it naturally handles manifold and non-manifold surfaces, surfaces with boundaries, as well as non-orientable surfaces, all in a consistent way. It does not require any extra geometric information other than the positions of the samples. Depending on no extra information makes this algorithm suitable for a broader range of applications. Our method is robust to irregular sampling and surface gaps, and relatively robust to the presence of noise. Furthermore, it is fast, parallelizable and easy to implement because it is based on simple local operations.

In addition to its central contribution, the new reconstruction algorithm, this chapter also introduces three other original results:

- An extension to Tsao and Fu's algorithm [127] to perform thinning while preserving surface boundaries (Section 6.2.2);
- An extension to Azernikov's meshing algorithm [11] to support the creation of polygonal meshes for non-manifold surfaces represented as point clouds (Section 6.2.3);
- A new algorithm for smoothing surface boundaries that significantly improves the quality of the meshes reconstructed for surfaces with boundaries (Section 6.2.3.1).

In our approach, surface reconstruction is performed using a three-step process. First, the space containing the point cloud is subdivided, creating a voxel representation. Gap filling is performed at this stage (Section 6.2.1). Then, a voxel surface with the same topology of the sampled surface is computed using topological thinning operations based

on Tsao and Fu's algorithm [127] (Section 6.2.2). Finally, a polygonal mesh is extracted using a modified version of Azernikov's algorithm (Section 6.2.3). Figure 6.1 illustrates the versatility of our technique to reconstruct surfaces with different topologies.

6.1 Related Work

According to Bloomenthal et al. [21], boundaries are usually specified, in the implicit function based methods, via additional functions that are required to have all the same sign. Bloomenthal and Ferguson [22] describe an algorithm for polygonizing non-manifold implicit surfaces defined by multiple regions of space. The algorithm can handle surfaces with boundaries and intersections, but the existence of multiple regions significantly adds to the complexity of the polygonizer. Park et al. [97] identify the surface boundaries on the point model before filling the hole by copying similar region into the hole.

Adamson and Alexa [2] present a point-based algorithm for rendering (not reconstructing) surfaces with boundaries and non-orientable surfaces. In their approach, a dense set of points defines an implicit surface, which is identical to a moving least squares (MLS) surface approximation [3]. The technique can only be used for rendering manifold surfaces and is based on ray casting. A surface with boundary is then locally defined as the set of points x (on the implicit surface), whose distances to a weighted average position in a neighborhood Ω that contains x is less than a user-specified threshold. Thus, sparsely sampled regions are rendered as holes. Unlike [2], our technique actually reconstructs a polygonal representation for these surfaces.

Azernikov's approach [11] constructs a connectivity graph and then creates facets by traversing the graph and finding minimal loops. In the resulting mesh, each voxel yields one vertex, computed as the centroid of input points inside the associated voxel. The algorithm cannot guarantee that the voxel surface has a well-defined local topology. Therefore,

a number of heuristics are used to assure the correctness of the resulting mesh. It also assumes that its input consists of a densely sampled point cloud and does not fill gaps. There are three major differences between our modified meshing strategy and Azernikov's algorithm: (1) we only find a connectivity graph for *surface* and *border voxels* (see Section 6.2.2 for a definition of these terms); (2) our approach handles non-manifold surfaces; and (3) because some empty octree voxels corresponding to holes need to be incorporated in the voxel surface (in order to automatically fill them), we employ a new way to identify such voxels and assign vertices to them.

6.2 The Surface Reconstruction Algorithm

Our surface reconstruction algorithm constructs a polygonal mesh from a point cloud using voxels as an intermediate representation. The steps of the algorithm are shown in Algorithm 6.1. First, the point cloud is turned into a voxel representation. At this stage, gaps are filled using a new gap-filling algorithm. Then, topological thinning is used to create a voxel surface representation. Finally, a polygonal mesh is extracted from the voxel surface.

1. voxelization and gap filling;
2. topological thinning;
3. meshing.

Algorithm 6.1: Surface reconstruction algorithm

6.2.1 Voxelization and Gap Filling

The algorithm starts by computing a tight bounding box for the point cloud, which is then subdivided into voxels. The voxel size should be small enough to avoid merging distinct surface features, but large enough to avoid unnecessary computations. Currently, the

voxelization process is performed based on a user-specified voxel size. Voxels containing original points are called *p-voxels*. Some empty voxels (*i.e.*, not containing points) falling in between p-voxels and representing non-sampled surface patches are called *g-voxels*. g-voxels will later be used for gap filling (Section 6.2.1.2 explains how to identify g-voxels). We call *foreground voxels* the set defined by the union of p-voxels and g-voxels. Foreground voxels are used for topological thinning (Section 6.2.2). Because they only occupy a small portion of the space, we use a dexel data structure [50] to store them. The complement of the foreground voxels with respect to the bounding box is called *background voxels*.

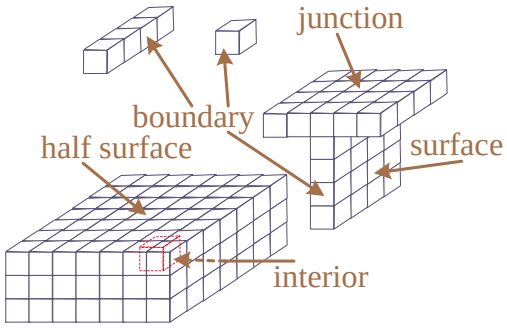


Figure 6.2: Topological classification of voxels.

6.2.1.1 Topological Classification of Voxels

We are interested in reconstructing surfaces and surface junctions, while preserving surface boundaries. The *local topological type* of a voxel is determined by the number of foreground and background connected components in its neighborhood. A voxel is called a *junction voxel* if it is at the junction of two or more surfaces. Likewise, a voxel is called a *boundary voxel* if it is at the boundary of some surface, along some curve, or just an isolated voxel. *Half surface voxels* are on one side of a thick voxel surface and are the only ones to be deleted during topological thinning. The existence of thick voxel surfaces

Table 6.1: Voxel topological types

Topological type	NFC_3	NBC_3	$NBC_{>3}$
interior	1	0	any
boundary	>1 or 0	any	any
boundary	1	1	1
surface	1	2	2
junction	1	≥ 2	> 2
half surface	1	1	2

is usually associated with the presence of noise in the dataset. Figure 6.2 illustrates the different topological types of a voxel.

To compute the local topological type of a voxel v, one needs to compute NFC and NBC, the number of connected components formed exclusively by foreground (18-connected) and background (6-connected) voxels, respectively, in the neighborhood of v. Before computing NFC, v should be deleted from its neighborhood. The appropriate size for the neighborhood depends on the level of noise in the point cloud and on the surface features to be reconstructed. According to our experience, a maximum neighborhood size of $5 \times 5 \times 5$ voxels seems to work well in practice for most situations. All examples shown in the chapter were reconstructed using a $5 \times 5 \times 5$ voxel neighborhood.

The topological type of a voxel v is determined by looking up Table 6.1 using the values of NFC and NBC. The subscripts of NFC and NBC in Table 6.1 represent the size of a neighborhood (*e.g.*, a subscript value k indicates a $k \times k \times k$ neighborhood). For efficiency reasons, we first search the $3 \times 3 \times 3$ neighborhood of v. If the $NBC_3 = 0$, we have an interior voxel, and the classification process stops. If, on the other hand, $NFC_3 > 1$ or $NFC_3 = 0$, v is a boundary voxel. Although Table 6.1 shows the necessary conditions for identifying v's topological type on a $3 \times 3 \times 3$ neighborhood, such a small neighborhood size can only be safely used with clean datasets. Noise tends to cause voxel surfaces to appear thicker. In this case, the use of a $3 \times 3 \times 3$ neighborhood is often insufficient to

correctly characterize the different components in v's neighborhood, thus leading to a incorrect topological classification. If v's topological type cannot be safely identified based on the values of NFC_3 and NBC_3, we increase the neighborhood size to $5 \times 5 \times 5$ and recompute NBC. In this case, if $NFC_3 = 1$ and $NBC_3 = 1$ and $NBC_5 = 1$ then v is a boundary voxel. If $NFC_3 = 1$ and $NBC_3 = 1$ and $NBC_5 = 2$ then v is a half surface voxel. Otherwise, if $NBC_5 = 2$, v is a surface voxel. If $NBC_5 > 2$, v is a junction voxel. These conditions are summarized in Table 6.1 and in Algorithm 6.2.

```
compute NFC_3 and NBC_3 for v;
if (NBC_3 = 0)
    return interior voxel;
if (NFC_3 > 1 or NFC_3 = 0)
    return boundary voxel;
else
    compute NBC_5 for v;
    if (NBC_3 = 1)
        if (NBC_5 = 1)
            return boundary voxel;
        else
            return half surface voxel;
    else
        if (NBC_5 > 2)
            return junction voxel;
        else
            return surface voxel;
```

Algorithm 6.2: Topological classification (v: voxel)

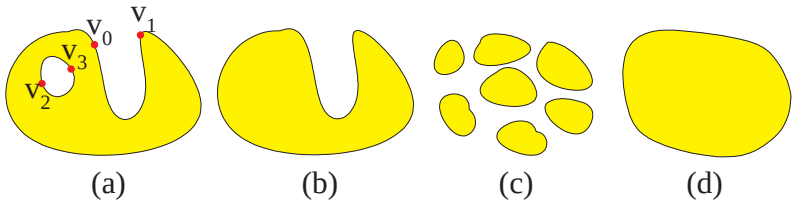


Figure 6.3: Possible situations in gap filling: (a) a surface with an inner hole. Here $< v_2, v_3 >$ is a valid voxel line segment, but $< v_0, v_1 >$ is not; (b) the surface reconstructed from (a) with hole filled; (c) several surface patches with in-between gaps; and (d) the surface reconstructed from (c).

6.2.1.2 Identifying g-voxels and Filling Gaps

The voxel surface defined by the set of p-voxels may contain holes and/or gaps due to insufficient sampling, as illustrated in Figure 6.3(a) and (c). We will not make a distinction between a hole (Figure 6.3(a)) and a gap (Figure 6.3(c)). For simplicity, we use the term "gap" to refer to hole and gap. Because we adopt a more general definition of "gap", the classical hole-filling schemes (*i.e.*, finding hole boundary before filling the hole [57]) cannot be applied here. Algorithm 6.3 fills gaps before a polygonal mesh is extracted from the voxel surface. The algorithm uses two dexel data structures D_{pg} and D_b, which are both initialized with all p-voxels. At the end of the procedure, D_{pg} will contain the set of foreground voxels (p-voxels and g-voxels) and will be used as input for the topological thinning procedure; D_b, on the other hand, will contain the set of boundary voxels, and will be used to prevent boundary voxels in D_{pg} from being eroded during topological thinning (Section 6.2.2).

The idea behind Algorithm 6.3 is straightforward: after initializing D_{pg} and D_b with p-voxels, the algorithm tries to fill gaps by filling voxel line segments with increasing lengths ranging from 1 to L (a maximum user-specified value). Each segment should connect a pair $< v_0, v_1 >$ of boundary voxels (classified using Algorithm 6.2). Before a voxel line

segment is filled, the segment has to pass a validation test.

Gap refers to the background voxels between adjacent patches of the same surface component. And we call the background voxels separating two different surface components *separation*. In Algorithm 6.3, we first fill gaps with small width. When the length of voxel line segments become larger, we need to test whether they are connecting already connected patches.

store all p-voxels in both D_{pg} and D_b;
for d=1 to hole size L;
  for each voxel v in D_b;
    check the topological type of v in D_{pg};
    if v is not a boundary voxel;
      delete v from D_b;
  for each voxel v_0 in D_b;
    for each v_1 in D_b with $\|v_1 - v_0\| \leq d$;
      if voxel line segment $< v_0, v_1 >$ is valid;
        create g-voxels connecting v_0 to v_1;
        add these g-voxels to D_{pg} and D_b;

Algorithm 6.3: Gap filling in voxel space

The validity tests of voxel line segment $< v_0, v_1 >$ include the following: (1) except the two end *boundary* voxels, the voxel line segment should be solely composed of background voxels; (2) v_0 and v_1 do not come from the same outmost boundary (*e.g.*, v_0 and v_1 in Figure 6.3(a)). If the voxel line segment passes the tests, all (empty) voxels along the segment are turned into g-voxels, and used to fill gaps. g-voxels are automatically incorporated into the set of foreground voxels.

A boundary is a closed voxel curve. We start from an arbitrary voxel and propagate it via adjacency along the curve until all voxels along the curve have been reached. The

number of propagation becomes then the length of the curve. If we propagate a boundary towards adjacent non-boundary regions, we obtain a new voxel curve. If the boundary is longer than the new curve, it is considered to be an outmost one. In practice, however, this criterion could fail and users might want to preserve some hole boundaries (*e.g.*, the holes in the bottom of the bunny in Figure 6.1(b)). For these reasons, the user is allowed to mark any boundary as outmost later.

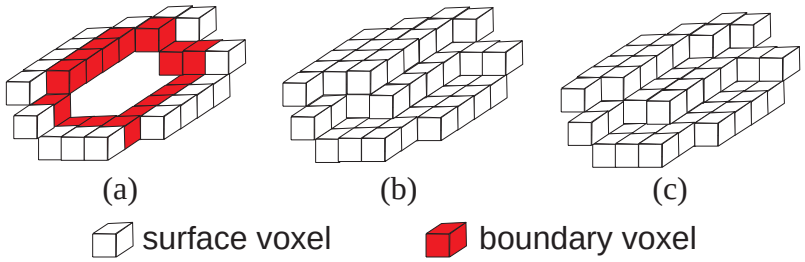


Figure 6.4: Gap filling and topological thinning: (a) fragment of a voxel surface with a boundary highlighted in red; and (b) result of the gap filling procedure; and (c) resulting voxel surface after topological thinning.

6.2.2 Topological Thinning

Before converting the voxel representation into a polygonal mesh, we should remove all voxels that cause the "surface" to be unnecessarily thicker than one voxel deep. Such voxels are called half-surface voxels (see Figure 6.2) and usually result from the existence of noise in the point cloud or inappropriately large voxel size. In order to create a thin voxel shell, we perform a topological thinning operation. By iteratively removing boundary voxels and reclassifying the remaining voxels, Tsao and Fu [127] find the skeleton (medial axis or medial surface) from voxel representations. However, their algorithm erodes surface boundaries as well, which we want to preserve. Therefore, we have extended Tsao's and Fu's original algorithm to preserve boundaries. In the extended algorithm, thinning is applied to the set of foreground voxels D_{pg}, while D_b (the set of boundary voxels) is

checked to avoid improper removal of boundary voxels. Both D_{pg} and D_b are computed using Algorithm 6.3. At the end of the thinning process, D_{pg} contains a *voxel surface* (more precisely, D_{pg} will contain a set of connected components representing the surface).

Figure 6.4 illustrates the process of gap filling and topological thinning. The red voxels represent a surface boundary and the result of gap filling is shown in Figure 6.4(b). Note the existence of some extra voxels. Figure 6.4(c) shows the output produced by our topological thinning algorithm, which will be used as input for the meshing stage (last step of our surface-reconstruction algorithm).

6.2.3 Meshing

Meshing is performed using a variation of the algorithm presented by Azernikov et al. [11]. Since the local topological type of all voxels in D_{pg} is known, the step in Azernikov's algorithm required to eliminate "false facets" (*i.e.*, facets that might appear in the connectivity graph but do not belong to the original surface) can be safely skipped. For the case of manifold surfaces, this simpler version of the algorithm can be used. Since Azernikov's algorithm does not handle non-manifold surfaces, we modified the original algorithm by allowing one surface to be composed of several *sub-surfaces* connected at junction voxels.

Given two surface patches P_i and P_j, their intersection defines a set of junction voxels. Figure 6.5 illustrates this situation in 2D showing a slice (side view) through two interpenetrating surface patches. In Figure 6.5, blue squares represent junction voxels. Red squares represent surface voxels. Surface voxels touching junction voxels are called *border voxels*. A set of border voxels of a given patch defines a *border curve* for the patch. We call *intersection curve* the curve defined by the set of junction voxels.

Given an intersection between two patches P_i and P_j, there are three possible types of

border curves for one participating patch, as illustrated in Figure 6.6 for the case of patch I: two disconnected border curves (Figure 6.6a), one open border curve (Figure 6.6b), and one closed border curve ((Figure 6.6c). For the cases (b) and (c), each patch still consists of a single connected component. Meshing can then be performed after junction voxels are connected to voxels of the border curve, and the resulting set of voxels is submitted to a thinning operation.

If the intersection causes a patch to be split into two disjoint connected components (Figure 6.6(a)), the algorithm needs to regroup all components into a single mesh. Figure 6.5 illustrates this situation for a pair of intersecting patches P_i and P_j, resulting in four disjoint connected components S_0, S_1, S_2 and S_3, shown in red in Figure 6.5(a).

The process of regrouping starts by identifying all components and, for each one, creating a mesh using the simplified version of Azernikov's algorithm. After these meshes have been created, a normal vector is computed for each border voxel by averaging the normals of all faces sharing that particular voxel (Figure 6.5(a)). We start with a voxel in the intersection curve and walk along this curve. Let v_j be the current junction voxel in the intersection curve. Also, let v_a and v_b be any two border voxels adjacent to v_j and belonging to different border curves C_p and C_q. We compute $m_{a,b} = |n_a \cdot n_b|$, where n_a and n_b are the normals of border voxels v_a and v_b, respectively. We use a triangular matrix M to store in element $M_{p,q}$ the average of all values $m_{a,b}$ such that v_a is in border curve C_p and v_b is in border curve C_q. In this case, averaging is used to compensate for the fact that different border curves may have different numbers of border voxels.

After the walk through all junction voxels in the intersection curve is over, let $M_{p,q}$ be the element of matrix M with the biggest value. Then, we connect the components associated with border curves C_p and C_q and set $M_{p,q}$ to zero. The rationale here is that the orientation of border voxels belonging to these two components correlate the most. This is interpreted as a strong indication that the two components belong to the same surface.

Then, we identify the element in M with the next biggest value, connect their corresponding components and set the matrix cell to zero. This process is repeated until M contains no non-zero elements. The entire process is illustrated in Figure 6.5. First, components S_1 and S_2 are connected (Figure 6.5(b)), then S_0 and S_3 are also connected (Figure 6.5(c)). Connecting two components amounts to computing the union between the voxels belonging to the components themselves and the associated junction voxels, performing a thinning operation, and finally using the simplified version of Azernikov's algorithm to create a mesh. Figure 6.5(c) shows the final result after the algorithm is applied to the four components shown in Figure 6.5(a). The position of the vertex associated with a junction voxel is given

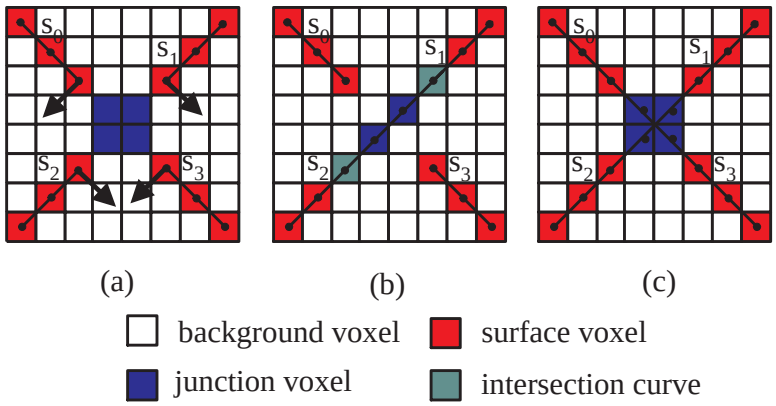


Figure 6.5: Creating non-manifold surfaces: (a) individual surfaces are reconstructed. s_1 and s_2 are part of the same surface intersected by the surface defined by s_0 and s_3; (b) surface (s_1, s_2) is reconnected; and (c) surface (s_0, s_3) is reconnected, thus reconstructing the non-manifold surface.

by the average position of all points falling inside the corresponding voxel. On the other hand, the vertex position inside a g-voxel is initially at the center of the voxel. In order to produce smoother meshes, these positions can be relaxed using any active contour techniques [19]. Currently, we minimize the following energy with a simplistic mass-spring system:

$$E = \sum_{i \in G} (\| \sum_{k \in M(v_i)} (v_k - v_i) \|)^2 \tag{6.1}$$

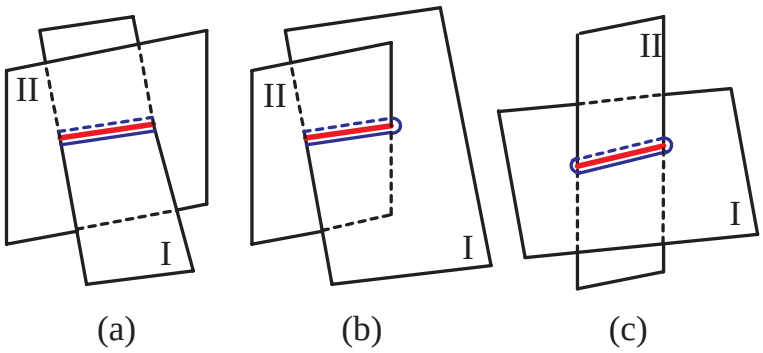
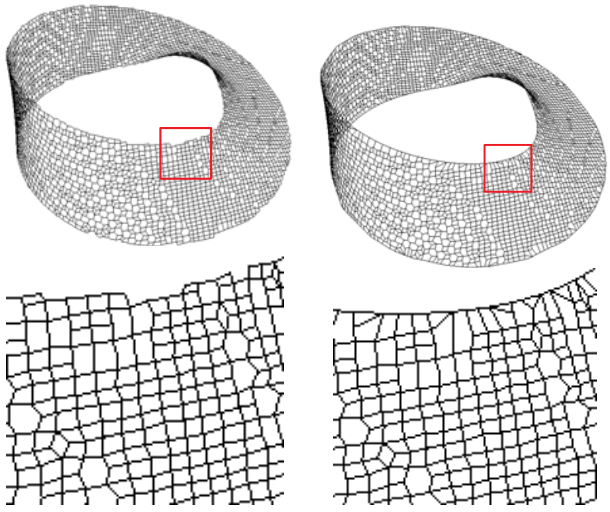


Figure 6.6: Three types of border curves in surface intersections (considered with respect to patch I): (a) two border curves; (b) a single non-closed border curve; and (c) a single closed border curve.

where G is the set of junction voxels and g-voxels, and $M(v)$ is the set of voxels which are directly connected to v in the resulting mesh. In order to accelerate the convergence, we solve the system iteratively until E becomes negligible.

6.2.3.1 Smoothing Boundaries

The representation of boundaries of surfaces is usually affected by under-sampling and noise. Also, the spatial discretization imposed by the voxelization process tends to worsen the problem. As a result, the meshing procedure described in section 6.2.3 tends to reconstruct ragged boundaries. This situation is illustrated in Figure 6.7(a) for the case of a mesh fragment showing a surface boundary. In order to improve the appearance of boundaries, we take groups of three consecutive boundary vertices and set the position of the middle one as a weighted average of the other two. Thus, let c be the vertex position associated with a voxel v at the boundary of a given surface S, and let c_l and c_r be the positions of the vertices associated with v_l and v_r, the left and right neighbors, respectively, of v at the same boundary. We then make c be collinear with c_l and c_r. As we proceed to the next group of three neighbor boundary voxels, the collinearity among c_l, c and c_r is lost, but the boundary



(a) (b)

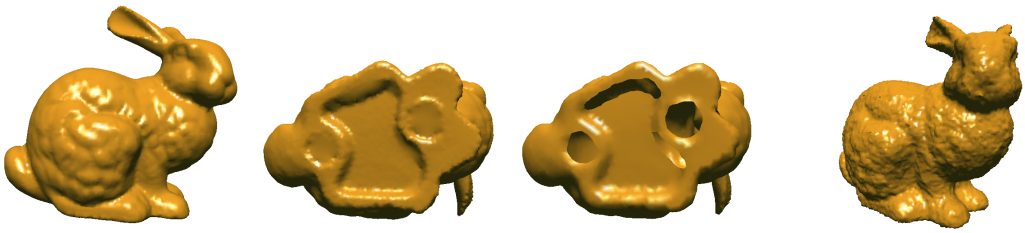
Figure 6.7: Smoothing the jagged boundaries. Here the meshes in the bottom are the zoom-in view of the red square regions in the top: (a) low sampling, noise, and spatial discretization imposed by the voxels can lead to the reconstruction of ragged boundaries; and (b) new mesh obtained after boundary smoothing.

smoothing effect is preserved. Using this simple procedure, the resulting boundary (Figure 6.7b) becomes much smoother, leading to more pleasing results. Figure 6.10 compares the results produced by mesh extraction (a) without and (b) with boundary smoothing and illustrates the significant improvement obtained with this technique.

6.2.4 Cost of the Algorithm

Let n be the number of samples in the point cloud and let m be the number of voxels used to discretize its bounding box. The cost of assigning a given input point to a voxel is $O(1)$, adding up to $O(n)$ for the entire input data. The topological classification of one voxel is done by analyzing a finite neighborhood and, thus is $O(1)$. Therefore, the classification of all voxels is performed in $O(m)$. During gap filling, we have at most $(2L+1)^3$ voxel line segments and may have up to $L(2L+1)^4 m$ voxel operations, where L the maximum length allowed for a segment. Gap filling is then performed in $O(m)$.

Thus, the first step of the surface reconstruction algorithm has cost $O(n+m)$. Topological thinning is performed in $O(m)$, since each voxel is visited and either deleted, if it is a half surface voxel, or kept, otherwise. Finally, for meshing, we need to process all foreground voxels (*i.e.*, p- and g-voxels) in order to compute the average position of the sets of points inside each voxel. Since the cost of creating the mesh itself is $O(m)$, the cost of the entire meshing stage is $O(n+m)$. Therefore, the total cost of our surface reconstruction algorithm is $O(m+n)$.



(a) (b) (c) (d)

Figure 6.8: Reconstruction of the Bunny model: (a) Stanford bunny reconstructed from 10 range scans using the proposed algorithm; (b) reconstruction of the bunny as a closed surface; (c) reconstruction of the bunny preserving the holes (boundaries); and (d) noisy bunny.

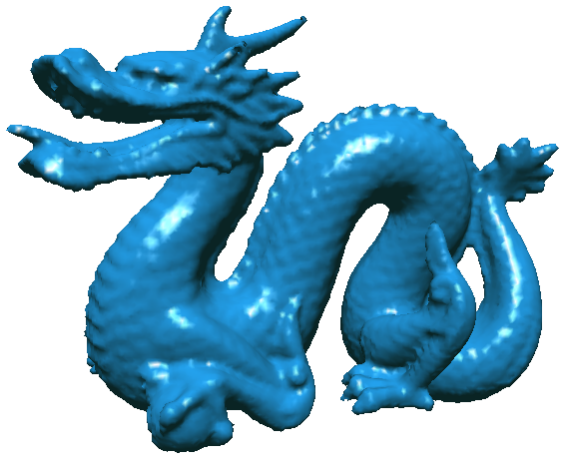


Figure 6.9: The reconstruction of the dragon model obtained from 60 range scans (containing registration errors, noise and outliers) using our algorithm.

6.3 Results

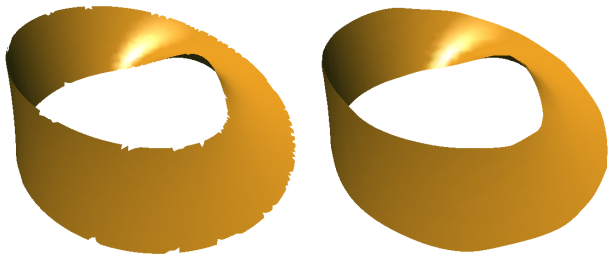
We have implemented the described algorithm and used it to reconstruct manifold and non-manifold surfaces with different topologies. The point clouds used as input for the reconstruction process were obtained from actual range scans as well as from synthetic datasets. Figure 6.1 illustrates the versatility of our approach showing reconstruction results for different kinds of topologies. Our algorithm can be used to reconstruct closed surfaces (Figure 6.1(b)), surfaces with boundaries (Figure 6.1(c)), non-orientable surfaces (Figure 6.1(d)) and non-manifold surfaces (Figure 6.1(e)), all using a single framework. For rendering, the meshes were obtained using the procedure described in section 6.2.3 and were then triangulated. Vertex normals were approximated by averaging the normals of all faces sharing the given vertex.

Relying on the voxelization, topological classification and gap filling procedures, our algorithm is robust to the presence of noise and irregular sampling. Figures 6.8 and 6.9 illustrate the results of the proposed algorithm applied to actual range scans. Figure 6.8 shows several reconstructions of the Stanford bunny obtained from a point cloud created after merging 10 range scans (containing noise and registration errors), with a total of 362,272 points. The bunny model (Figure 6.8a) is well known to have holes at the bottom. Figure 6.8(b) shows the reconstructed bunny with the holes filled, whereas in Figure 6.8(c) the boundaries were preserved. We also added Gaussian noise with magnitude of 1% of the size of the original bounding box to the point cloud. The resulting reconstructed model is shown in Figure 6.8(d). The large number of g-voxels in the noisy bunny model (see Table 6.2) results from the existence of many small holes due to the added noise.

Figure 6.9 shows the Stanford dragon reconstructed with our algorithm after merging 60 range scans containing noise and registration errors. The merged dataset for this genus-2 model has about 1.7 million points and includes a number of noticeable outliers. Using

our approach, the global topology is correctly reconstructed and delicate surface details are nicely preserved. No pre- or post-processing step is required for removing the outliers.

Figure 6.10 shows the reconstruction of a Möbius strip model (a non-manifold) and illustrates the ability of our approach to handle non-orientable surfaces with boundaries. On the left, we show the reconstructed model before boundary smoothing. Notice the ragged edges. On the right, one sees the result after our smoothing boundary algorithm has been used. The new boundaries are significantly smoother, resulting in a more pleasant model. Figure 6.7 shows close-up views of the two meshes for comparison. One should note that the ability to reconstruct non-manifold and non-orientable surfaces has more than just theoretical importance. For example, these surfaces have been represented using points [2], which one may want to visualize using polygonal models.



(a) (b)

Figure 6.10: Reconstruction results of a non-orientable surface (Möbius strip): (a) before; and (b) after smoothing of boundaries.

Figure 6.11 shows another view of the non-manifold model shown in Figure 6.1(e), which consists of two intersecting surfaces. This is a particularly challenging test and has been perfectly reconstructed by our approach. To the best of our knowledge, no other contemporary surface-reconstruction algorithm is capable of reconstructing such a model.

Figure 6.12 shows reconstruction results for a vase model with varying sampling rates and illustrates the ability of our approach to work with sparse datasets. Figure 6.12(a) shows a point cloud obtained by randomly selecting only 20% of the points (13,619 points)

from a digitized vase model. The small square highlights the irregular sampling in the resulting point cloud. Undersampling can also be observed by comparing the number of g-voxels in Table 6.2. The number of such voxels for the reduced model is considerably bigger, thus implying severe undersampling. Figure 6.12(b) shows the reconstructed model obtained from the point cloud shown in (a). For comparison, Figure 6.12(c) shows the reconstructed vase using the full dataset (68,097 points). Although it is possible to observe differences on the vase's relief and at the boundary of its base, all major features were faithfully reconstructed from a much smaller dataset.

Table 6.2 presents some statistics associated with the models shown in the chapter. Measurements were performed on a Pentium 4, 2 GHz PC with 1GB of memory. The last three columns of the table show the running times, in seconds, of the three steps of our surface reconstruction algorithm.

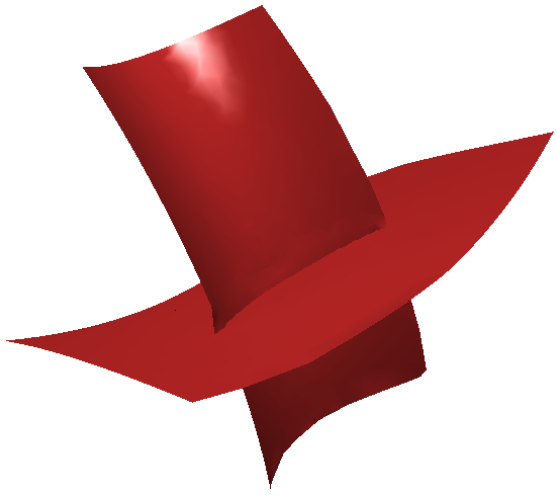
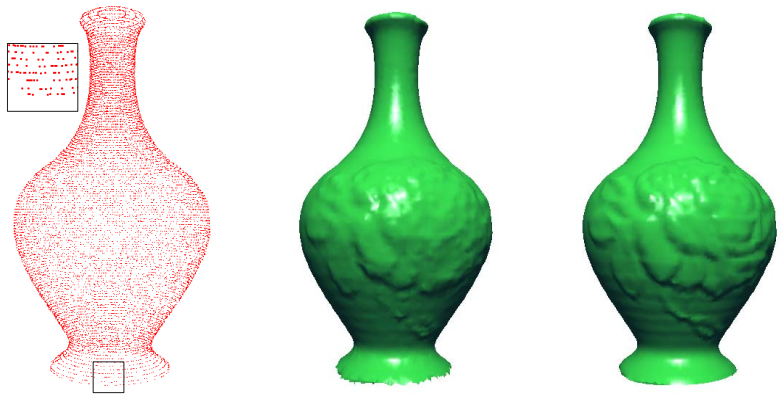


Figure 6.11: Non-manifold surface reconstructed with our algorithm. The surface consists of two intersecting patches.

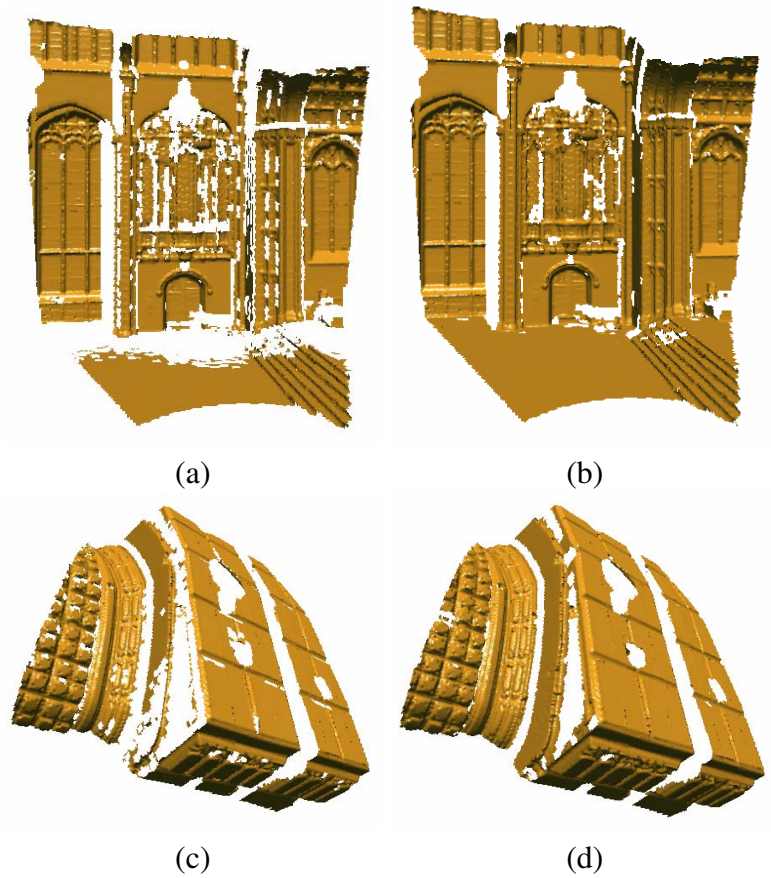
WE ALSO APPLIED THE PROPOSED ALGORITHM TO AN INDOOR SCENE. WE HAVE TWO DIFFERENT RANGE SCANS OF THE SCENE. THE SCENE HAS SOME CHANDELIERS SUSPENDED FROM THE CEILING AND HIGHLY SPARSE AND VARYING SAMPLING RATES



(a) (b) (c)

Figure 6.12: Reconstruction of sparsely and unevenly sampled objects: (a) A point cloud obtained by randomly selecting 20% of the points (13,619 points) of a digitized vase model. The small square highlights the irregular sampling. (b) Vase model reconstructed from the point cloud in (a). (c) Vase reconstructed from the full point cloud (68,097 points). Notice that all important features were appropriately reconstructed in (b).

FOR SOME REGIONS. THE STRINGS ATTACHING THE CHANDELIERS ARE SAMPLED AS POINTS ALONG A SINGLE LINE. THE CHANDELIERS OCCLUDE THE RANGE SCANNER AND LEAVE SEVERAL MISSING REGIONS IN THE CEILING. IN FIGURE 6.13, THE TWO SCANS ARE RECONSTRUCTED USING THE PROPOSED ALGORITHM IN THIS CHAPTER. WE TRIED TO RECONSTRUCT THE MODEL WITH ALL GAPS PRESERVED. IN ADDITION, WE RECONSTRUCT THE MODEL WITH A SELECTED GAP SIZE OF 2. GAP-FILLING WITH A LARGER SIZE IS NOT SUCCESSFUL BECAUSE IT WILL CONNECT UNRELATED SURFACE COMPONENTS. FOR EXAMPLE, ONE END OF THE STRING OF A CHANDELIER IS TOO CLOSE THE BOUNDARY OF THE HOLE CAUSED BY THIS CHANDELIER. THE GAP-FILLING ALGORITHM MAY CONNECT THEM BEFORE FILLING THE HOLES, LEADING TO A TOPOLOGICALLY INCORRECT RECONSTRUCTION. THE STRINGS OF THE CHANDELIERS ARE NOT RECONSTRUCTED BECAUSE THEY ARE PRESENTED MERELY BY *boundary* VOXELS IN VOXELIZED MODEL.

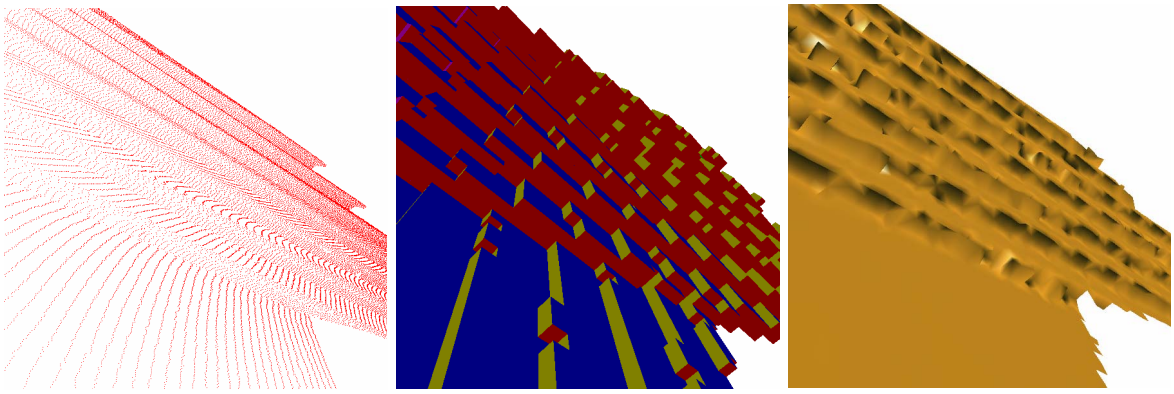


(a) (b)

(c) (d)

Figure 6.13: Reconstruction of the indoor scene: (a) reconstructed surface with all boundaries preserved of the first range scan; (b) reconstructed surface with small gaps filled of the first scan; (c) reconstructed surface with all boundaries preserved of the second range scan; and (d) reconstructed surface with small gaps filled of the second scan.

THE PROPOSED ALGORITHM DOES NOT WORK WELL FOR SHARP FEATURES. IN FIGURE 6.14, WE HAVE POINTS REPRESENTING THE STEPS OF THE INDOOR SCENE. AFTER THINNING, WE HAVE A VOXEL SURFACE REPRESENTING THE STEPS. BECAUSE THE MAJOR VOLUME AXIS DO NOT ALIGN WITH THE DIRECTION OF THE STEPS, WE END UP HAVING A ZIGZAG STYLE FOR THE SHARP EDGES. SINCE EACH SURFACE VOXEL WILL YIELD ONE VERTEX FOR THE RESULTING MESH AND THE LOCATION OF THIS VERTEX IS COMPUTED AS THE AVERAGE OF THE SAMPLES INSIDE. IF MULTIPLE SAMPLES ARE INSIDE, THE VERTEX WILL BE NOT EXACTLY ALONG THE EDGE. FOR



(a) (b) (c)

Figure 6.14: The proposed framework is not capable of preserve sharp features: (a) the points representing steps in the indoor scene; (b) the surface voxels representing the steps in our algorithm; (c) the reconstructed steps.

THIS REASON, THE SHARP FEATURES WILL NOT BE PRESERVED.

For models with dense and regular samples, we obtain voxel surfaces with correct local topological types for each voxel. In the presence of noise, sample positions tend to be shifted, creating "inflated" voxel surfaces and possibly introducing small gaps in the surface. In most cases, the level of noise in the dataset is smaller than the voxel size. Thus the introduced gaps are at most one voxel wide and can be filled with ease using Algorithm 6.3 with $L = 2$.

Our approach subdivides the bounding box of the point cloud into voxels with uniform sizes. While the use of an adaptive data structure, such as an octree, has advantages in terms of memory requirements, the topological thinning algorithm requires a constant voxel size. Note that using small voxels impacts the running time of the algorithm. Thus, the most suitable voxel size can be understood as the biggest one that still allows the reconstruction of the intended surface details. Such a value can be obtained based on the user experience or using a binary-search-based trial and error process.

Our approach does not require point normals. However, if the dataset contains normals, they can be used to help thinning and meshing. For instance, the sub-directions for thinning

could be reduced from 6 to 2 and one could further optimize the obtained mesh by applying the constraints on surface normals.

The major limitation of the proposed approach is that the hole filling algorithm relies on the assumption that the size of gaps to be filled is smaller than the size of separation between surface components, which should not be filled. In the case of increased noise level or missing samples, the assumption might be broken and the proposed algorithm will not work.

Table 6.2: Statistics of some reconstruction results (time in seconds)

Model	# of input points	voxel grid	# of p-voxels	# of g-voxels	# of facets	time of gap filling	time of thinning	time of meshing
Merged Bunny	362,272	128^3	25,751	419	27,953	5.64	11.00	0.56
Merged Dragon	1,769,513	256^3	41,243	662	101,997	20.80	28.60	3.79
Möbius Strip	7,560	64^3	2,912	0	2,838	0.89	1.23	0.11
Intersected surfaces	20,402	32^3	2,171	0	2,021	0.43	0.00	0.11
Noisy bunny (with 1% noise)	34,834	128^3	19,122	13,917	35,531	75.61	29.23	1.32
Vase	68,097	64^3	11,830	459	13,654	1.40	2.78	0.35
Undersampled Vase	13,619	64^3	6,413	6,092	13,855	7.50	4.20	0.37

Chapter 7

Reconstructing Regular Meshes from Points

Given a set of points $\mathcal{P} = \{p_i = (x_i, y_i, z_i)\}$ sampled from an object's surface, the goal of surface reconstruction techniques [52, 91] is to compute a mesh $\mathcal{M}$ that approximates the underlying surface. This is essentially an ill-posed problem as it admits multiple solutions. Usually, the solution with minimal energy in some sense will be chosen as the optimal result.

Thus, for instance, one may want to minimize the distance between the original points and the reconstructed surface. Often, however, it will be important to have well-shaped (*e.g.*, nearly equilateral) triangles for the best visual quality and for some subsequent use by other procedures, such as finite element analysis [23]. Triangles with nice aspect ratios are also important for animation and rendering (shading) [25].

While one would like to preserve the detailed information available in the original data as much as possible, surface reconstruction from points often leads to some loss. In this chapter, we propose a new approach for reconstructing regular meshes from point clouds. Here the point connectivity is inferred using a neighborhood relation and detailed geometric

features are preserved with the use of a new point parameterization. Given the set of input points, meshes with arbitrary resolutions can be extracted by sampling the patches using a regular grid.

The proposed approach consists of two steps as shown in Figure 7.1. First, a parameterization is computed for the point cloud. Then, a mesh is extracted based on the resulting parameterization. The parameterization process can be summarized as follows: given a 3D point cloud representing a surface with arbitrary topology, it is segmented into several non-overlapping patches, each topologically equivalent to a disk. The points in any patch are assigned a 2D parameterization via a propagation procedure. This results in a set of 2D *parameter patches*. Although patches do not overlap, a patch boundary may be adjacent to several other patches. During meshing, shared vertices are created along the patch boundaries, while the interior of the patches are tiled using a regular triangular pattern.

Surface remeshing techniques [5, 6, 103, 118] create regular meshes, but they take another mesh as input and, therefore, already have connectivity information. Basically, these techniques compute parameters for each original vertex, and new vertices are created by resampling [118]. In this chapter, an alternative paradigm is employed: first we compute the point parameterization, and then resample the parameterization domain for meshing. This breaks the direct coupling between the original points and the resulting mesh, and has some advantages. For instance, it allows the preservation of details on the point side, and for more flexibility during meshing. Thus, meshes with arbitrary resolutions can be easily created. It is also a *direct* reconstruction approach without involving an intermediate mesh. The use of intermediate representations tends to cause loss of information.

The proposed algorithm requires only the location of the points and, therefore, is general enough to handle all kinds of point datasets. Since a point parameterization is produced as a by-product of the reconstruction process, several operations can be directly applied to the resulting mesh, such as texture and bump mapping. As the digitization of real models

using laser scanners is becoming increasingly popular, the automatic parameterization of reconstructed models provided by our approach can potentially save modelers some significant amount of work.

The main contributions of this chapter include:

- A new approach for reconstructing regular meshes from point clouds that produces nearly equilateral triangles. It is based on a new paradigm that decouples parameterization from meshing, thus improving the flexibility of the meshing process, while the obtained parameterization can also be used for other applications (Section 7.2);
- A robust boundary-identification algorithm to find surface boundaries, including hole boundaries (Section 7.3.2);
- A novel approach for parameterizing cuts between parameter patches that allows points along cuts to have more than one associated parameter value (Section 7.3.3);
- A solution for creating triangles from the borders towards the center of the parameter patch, thus counteracting the effect that some patches might overlap (Section 7.4);
- A new approach to optimize the location of interior vertices so as to adjust the triangles around the patch boundary (Section 7.4.2).

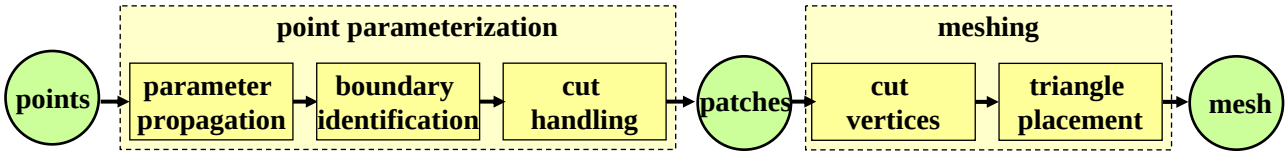


Figure 7.1: Pipeline for creating regular meshes from point clouds. Given a point cloud, a point parameterization is used to create a set of 2D parameter patches, which are in turn used as input to the meshing process.

7.1 Related Work

Our approach falls into the category of representing geometrical information using images [74]. Geometry images [46] parameterize a 3D mesh to a rectangular domain based on geometric stretch. More theoretically sound results are obtained using spherical parameterization [45] and conformal mapping [56]. In comparison, we start with points without any connectivity information. In our approach, parameterized points form several patches, which may overlap and have non-rectangular shapes. The use of patches reduces the distortion and solves the nonuniform sampling problems inherent in the aforementioned approaches.

Our approach is also similar to Floater and Reimers [40] and Hormann and Reimers [53]. However, these methods force the parameterized surface to fit within a convex shape (*e.g.*, a disk), thus large distortion will be introduced. Mencl and Muller [81] first create some skeleton edges from the point dataset. Then triangles are created to fill the surface under the constraint of the skeleton edges. In these algorithms, the original data points are used as vertices. As a result, no guarantees can be given to the aspect ratio of the resulting triangles. In contrast, vertices created by our approach are not necessarily located at the original data points.

Zwicker et al. [152] have proposed an interactive approach to locally compute a parameterization with minimal distortion. However, user interaction is needed to assign some feature points and this parameterization approach could not be applied to the entire surface of a manifold object. Moving least squares surface [10, 112] finds the location of the extremal surface from the point clouds. Its major advantage is to reduce noise. In [112], an adaptive direct meshing approach is employed. A guide field is used to control the triangle size according to curvature and other features. The major difference between this

approach and ours is that the former can not output nearly equilateral triangles. In addition, it pre-computes the guide field while we compute the 2D parameter for each point via propagation.

7.2 Algorithm Overview

The pipeline of the proposed algorithm is illustrated in Figure 7.1. It includes two steps: point parameterization and meshing. For point parameterization, we compute several *3D patches* from the point cloud via propagation. The parameter values of the points inside the patches are computed, forming several 2D *parameter patches*. Each point p_i on a *3D patch* will then have an associated parameter $q_i = (u_i, v_i)$. Within any given 3D patch, there is a *nearly isometric mapping* to its *parameter patch*. That is, for any two 3D points p_i and p_j in a local neighborhood, their Euclidean distance nearly equals the distance computed in 2D using their corresponding parameters ($\|p_i - p_j\| \approx \|q_i - q_j\|$). Since each *3D patch* and its related *parameter patch* both refer to the same set of points, we often make no distinction between them and refer to both simply as *patches*. Whenever we need to refer to a particular kind of patch, we will use the specific terms.

A *patch* boundary may represent the presence of non-sampled regions on the surface. Alternatively, it may represent a parameter discontinuity with respect to adjacent patches. The patch boundaries are termed *cuts* because they represent discontinuities in the parameter values.

In the stage of meshing (Figure 7.1), we first add vertices along cuts and force them to be evenly distributed along the cuts. Then triangles are placed inside each patch using a regular pattern. In the algorithm, a Kd-tree is used to expedite the query of the k nearest neighborhood (kNN) of a point in 3D space.

7.3 Point Parameterization

Isometric mappings only exist for developable surfaces [32]. In this chapter, we instead find a *nearly isometric parameterization* for patches as large as possible. A propagation scheme is used to compute the parameter q for each point p. During the propagation, cuts may be introduced to separate different patches where a parameter discontinuity occurs. Since a cut is related to multiple patches, points along a cut might have multiple parameter values. This situation is illustrated in Figure 7.3.

We parameterize a point cloud via propagation. During the process, a tag selected from the set {*unvisited, patch, gap, cut, boundary* } will be assigned to each point to indicate its current status. Initially, all points are set as *unvisited*. After propagation, points belonging to any patch become *patch* points. The remaining points are *gap* points, representing the boundaries of the *patches*. In the next step, we find among the *patch* points the exterior and interior (hole) boundaries of the surface and mark the related points as *boundary* points. Finally, we convert the *gap* and *boundary* points into several *cuts* and parameterize the cuts. Algorithm 7.1 presents the steps associated with the point parameterization stage of

the pipeline shown in Figure 7.1.

```
// Point parameterization stage:
// (1) parameter propagation (Section 4.1)
for each point p in the point cloud do
    if p is not visited then
        CreateOnePatch(p)
    end
end
// (2) boundary identification (Section 4.2)
for each edge in the patch adjacency graph do
    if this is part of the boundary then
        Mark the edge points as boundary points
    end
end
// (3) cut handling (Section 4.3)
construct the gap point adjacency graph;
find cut points by extracting pruned minimum spanning trees from the graph;
find the patch boundary curve from the boundary points;
traverse the patch boundary curve to parameterize the cuts

function CreateOnePatch(point p);
begin
    push p to a stack;
    while stack is not empty do
        pop the point at the top;
        compute its parameters;
        add adjacent unvisited points into the stack
    end
end
```

Algorithm 7.1: Point parameterization pipeline.

7.3.1 Parameter Propagation

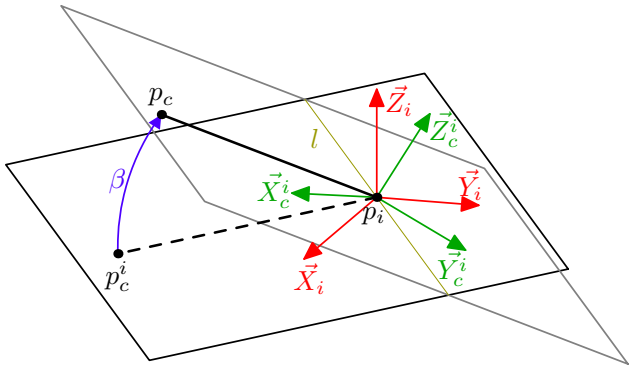


Figure 7.2: Parameter propagation from a *patch* point p_i.

We use our algorithm [149] to turn *unvisited* points into *patch* points, if applicable. Accordingly, each *patch* point p_i will be assigned a local coordinate frame $(\overrightarrow{X_i}, \overrightarrow{Y_i}, \overrightarrow{Z_i})$ and a parameter $q_i = (u_i, v_i)$. Initially, all *unvisited* points are put into a priority queue, sorted by the number of *patch* neighbors in their kNN. While the queue is not empty, we pop the current top point p_c, which has the largest number of *patch* neighbors. If p_c has no *patch* neighbors in $kNN(p_c)$, it is the first *patch* point in this patch. Its $\overrightarrow{Z_c}$ is computed as the eigenvector associated with the smallest eigenvalue of the covariance matrix defined in the points in the $kNN(p_c)$. $\overrightarrow{X_c}$, $\overrightarrow{Y_c}$ and q_c are chosen arbitrarily ($q_c = (0, 0)$ in our experiments), given that $\overrightarrow{X_c}$, $\overrightarrow{Y_c}$ and $\overrightarrow{Z_c}$ form an orthonormal basis.

In case p_c already has some patch neighbors in $kNN(p_c)$, $\overrightarrow{X_c}$, $\overrightarrow{Y_c}$, $\overrightarrow{Z_c}$ and q_c are computed from the *patch* neighbors of p_c. Let $N = \{p_i | p_i \in kNN(p_c)\}$ be the set of points in $kNN(p_c)$. For each $p_i \in N$, we project p_c onto the $\overrightarrow{X_i}\overrightarrow{Y_i}$ plane as p_c^i (see Figure 7.2). p_c^i is moved along the vector $\overrightarrow{p_i p_c^i}$ such that $\|p_c - p_i\| = \|p_c^i - p_i\|$. The new local coordinate system $(\overrightarrow{X_c^i}, \overrightarrow{Y_c^i}, \overrightarrow{Z_c^i})$ is computed for p_c by rotating $(\overrightarrow{X_i}, \overrightarrow{Y_i}, \overrightarrow{Z_i})$ about l by β degrees, where

l is a line passing through p_i and perpendicular to plane $\overline{p_c^i p_i p_c}$ (see Figure 7.2). Parameters $q_c^i = (u_c^i, v_c^i)$ are computed from $q_i = (u^i, v_i)$ as $(u_i + \overrightarrow{p_i p_c^i} \cdot \overrightarrow{X_c^i}, v_i + \overrightarrow{p_i p_c^i} \cdot \overrightarrow{Y_c^i})$. A weight is also computed for p_c from p_i as $w_c^i = 1/\|p_c - p_i\|$. We compute $\overrightarrow{X_c}$, $\overrightarrow{Y_c}$, $\overrightarrow{Z_c}$ and q_c as the weighted average of all $\overrightarrow{X_c^i}$, $\overrightarrow{Y_c^i}$, $\overrightarrow{Z_c^i}$ and q_c^i, respectively.

Two kinds of distortions are computed for point p_c: position distortion $D_{pos}(p_c)$ and orientation distortion $D_{ori}(p_c)$ using a variation of the estimates described in [149]. Note that the normalization of the weights w_c^i is embodied in the definition of D_{pos} and D_{ori}.

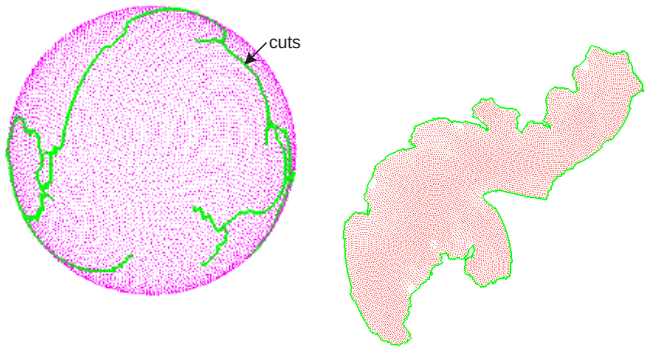
$$D_{pos}(p_c) = \max\left(\frac{w_c^i \|q_c^i - q_c\|}{\sum w_c^i}\right) \tag{7.1}$$

$$D_{ori}(p_c) = \max \frac{(w_c^i(2 - \overrightarrow{X_c^i} \cdot \overrightarrow{X_c} - \overrightarrow{Y_c^i} \cdot \overrightarrow{Y_c}))}{\sum w_c^i} \tag{7.2}$$

If $\alpha D_{pos} + (1 - \alpha) D_{ori} \geq E$, this point becomes a *gap* point, where α is the relative weight for the two errors and E is the maximum error. Otherwise, it becomes a *patch* point. we record the edges connecting p_c to all its *patch* neighbors in a *patch point adjacency graph* $\mathcal{G}_p$, which will be used later to parameterize the *cuts*. In all our experiments, we used $\alpha = 0.5$ and E is computed adaptively as the largest distance between a point and any neighbor in its kNN.

Once p_c becomes a *patch* point, the number of adjacent $patch$ points for its adjacent *unvisited* points needs to be updated and the priority queue is adjusted accordingly. After parameter propagation, we obtain several *patches*. Note that a *parameter patch* might overlap with itself in parameter space. This might happen, for instance, if the boundary follows a curved path and overlaps itself in the 2D parameter space. Consider, for example, two *patch* points p_1 and p_2, which are far from each other in 3D, but whose 2D parameter values are very close. During propagation, the parameters of the *unknown* neighbors of p_1 are computed based only on the 3D point location. Thus, the existence of a nearby point

(*i.e.*, p_2) in the 2D parameter space will not be taken into account.



(a) (b)

Figure 7.3: (a) The cuts (green curves) separating patches, created from the point parameterization; (b) The 2D parameter patch is bounded by the cuts.

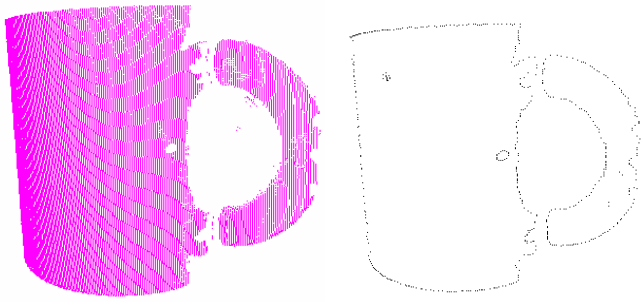
7.3.2 Boundary Identification

In the case the point model does not represent a closed 2-manifold surface, we need an algorithm to identify the surface boundaries (exterior and interior, *i.e.*, holes). In the literature, an angle criterion [47] is often used to check whether a point p_i belongs to the boundary. It consists of projecting the neighborhood N of p_i (including itself) onto its tangent plane (the plane fitted to N). By connecting p_i to its neighbor points on the tangent plane, one has a number of edges sharing p_i. The algorithm then finds the maximum angle between any two such adjacent edges. If such an angle is larger than a threshold, p_i is considered to be part of the boundary. Unfortunately, this approach is not robust in the case of varying sampling rates or high curvatures.

We propose a new criterion for identifying boundary edges that takes the computed local parameterization into account. Note that a boundary edge can be either a surface-boundary edge (in the case of a manifold with boundary) or a regular patch-boundary edge. Any surface-boundary edge also belongs to the edge of a given patch. The two kinds can

be distinguished from one another because internal patch-boundary edges have *gap* points as some of their neighbors. Let $e = (v_0, v_1)$ be an edge in the *patch point adjacency graph* $\mathcal{G}_p$, connecting vertices v_0 and v_1. We say that e is part of the surface boundary if e is not shared by two triangles. Thus, let u_i be the 2D parameter value associated to vertex v_i and expressed in homogeneous coordinates. Thus, the line supporting e in parameter space can be represented as $u_0 \times u_1$, where $\times$ is the cross-product operator. Thus, for any vertex v_j with associated parameter u_j, such a vertex is at one side of e if $(u_j \cdot (u_0 \times u_1)) > 0$ and is at the other side if $(u_j \cdot (u_0 \times u_1)) < 0$. The vertices of the adjacency graph represent *patch* points, while the edges represent neighboring relations. Thus, $e = (v_0, v_1)$ is considered a boundary edge if and only if for all vertices v_k in $\mathcal{G}_p$ and adjacent to both v_0 and v_1, the triple product $u_k \cdot (u_0 \times u_1)$ has the same sign for all v_k.

The algorithm for identifying boundary points has cost $O(m)$, where m is the number of edges in $\mathcal{G}_p$. Figure 7.4 (b) shows the boundary points identified by the algorithm on a single range image of a mug model (Figure 7.4 (a)), whose point cloud was obtained using a structured light scanner [107]. Note the small holes on the model due to high reflectivity and some noise. This example illustrates the robustness of our algorithm to find boundaries.



(a) (b)

Figure 7.4: (a) One range image of a mug; (b) The identified boundary points.

7.3.3 Cut Handling

The last step of the point parameterization stage (Figure 7.1) is cut handling. A *cut* is a curve defined by points for which the accumulated error in the propagated parameter values is bigger than some threshold. Cuts separate different patches but can also define a limit between two regions on the same patch (Fig 7.3).

Points along the cuts are special in the sense that they might have multiple parameters while points inside any patch have a single parameter value. *Cut points* are computed from *gap points*. For each gap point p_g we find its kNN using an approach similar to the one of Pauly et al. [99]. Edges connecting p_g and its *gap* neighbors are added to a *gap point adjacency graph* $\mathcal{G}_g$. Minimum spanning trees (MST) are then extracted from all connected subgraphs of $\mathcal{G}_g$[1]. From each spanning tree T_i, we remove all branches whose lengths are shorter than $3E$, where E is the average length of the kNN for all *patch* points. The value $3E$ was defined empirically based on our experiments. The pruned version of T_i becomes a *cut curve* C_i and its points are stored in a graph $\mathcal{G}_c$ (later, cut branches will be extracted from this graph and parameterized). Cut curves are shown as green curves in Figure 7.3, which depicts the parameterization of a point cloud representing a sphere.

Starting with the set of boundary points, we also build MSTs and compute pruned versions of them. Each resulting curve is then closed by creating an edge between the first and last point, and is called a *patch boundary curve*. The set of all patch boundary curves tightly enclose the cuts. Patch boundary curves are shown in red in Fig 7.5, while cuts are shown in green.

Conceptually, each cut point can be assigned an arbitrary number of different parameter values, which are obtained from patch boundary points. Note that, in this case, each parameter value is associated to a different patch and is only used in the context of such a

[1] A minimum spanning tree connects all vertices of a connected (sub)graph while minimizing the sum of the weights of the selected edges.

patch. Thus, let C be a cut and let p_c be a cut point with degree 1 (*i.e.*, connected to only one other cut point). We proceed the cut parameterization by first finding, for each p_c, its nearest point in the patch boundary curves separated by C (Figure 7.5). In case p_c separates two parts of the same patch, we find its two closest points in the patch boundary curve (one at each side of p_c).

The order of these nearest points along the patch boundary curve determines the order of *cut branches* (*i.e.*, the minimum sets of connected edges in $\mathcal{G}_c$ enclosed by two cut points of degree 1). For each point on a *cut branch*, we find the nearest patch point along the corresponding portion of the patch boundary curve (Figure 7.5). The approach described in Section 7.3.1 is used to parameterize the cut points.

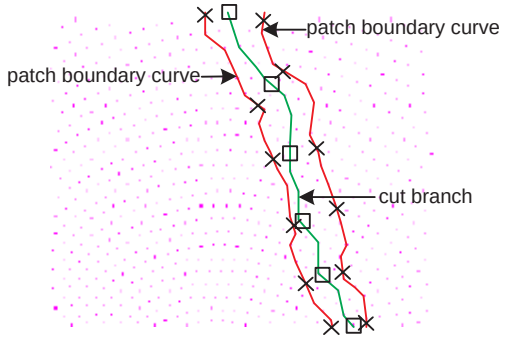


Figure 7.5: We use *patch boundary curves* (shown in red) to help order and parameterize the points along a cut (green curve). Here, the squares represent points along the same *cut branch*, while the $\times$'s are their corresponding points along the patch boundary curves.

7.4 Meshing from Parameterization

Once the parameterizations of all patches are available, one can proceed to create a triangle mesh with arbitrary resolution. To create a complete mesh, the user needs to specify the side length L of the equilateral triangles. To preserve geometric details and avoid the occurrence of T-vertices in the resulting mesh, we first create vertices along the *cuts*,

forming the boundary vertices of the parameter patches. Then, interior vertices are created using a regular sampling pattern. Algorithm 7.2 presents some pseudo code describing this process.

```
Procedure Meshing()
    create shared vertices along the cuts
    place vertices inside patches
    created triangles via edge flipping
    optimize the location of interior vertices
```

Algorithm 7.2: The meshing pipeline

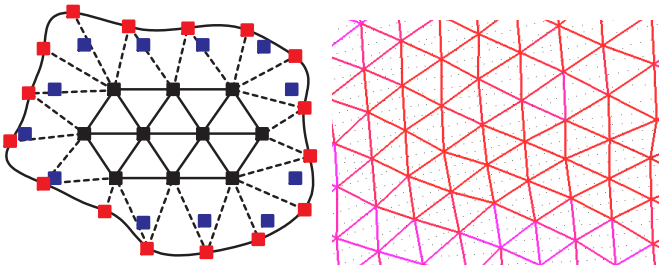
7.4.1 Vertices Along Cuts

We create vertices along each *cut branch* by marching from one end to the other, adding vertices in-between. It is enforced that the distance in 3D between adjacent vertices is L. If the distance between one end and its nearest vertex becomes less than $\frac{L}{2}$, we delete that vertex. After that, this distance is in the range of $[\frac{L}{2}, \frac{3L}{2}]$. We relax the location of vertices, except those at the end points, along the *cut branch* and make the distances between every vertex and its previous/next neighbors equal. The 2D parameters and the 3D location of in-between cut vertices are obtained by linearly interpolating adjacent cut points.

7.4.2 Triangle Placement

Interior vertices of a *parameter patch* are created by sampling it using a grid pattern composed of equilateral triangles, as shown in Figure 7.6(a). Interior vertices are located on the grid points and their parameters are determined implicitly. Conceptually, the meshing process can be understood as follows: let β be the set of points along the boundary of a patch P_i (matching its surrounding cut curves) and represented in 2D parameter space.

Such points are shown in red in Figure 7.6 (a). For each pair (β_i, β_j) of adjacent points along the boundary, we compute the coordinates of the interior vertex β_k that, together with β_i and β_j, would result in an equilateral triangle. However, in order to keep the mesh as regular as possible, we do not use β_k. Instead, we use the grid point closest to β_k as the third triangle vertex. This causes all interior vertices of the mesh to be on the grid pattern, and the triangles with vertices on the borders to be the only ones that might not be equilateral (Figure 7.6 (a)). An optimization procedure later tries to improve the aspect ratio of such triangles, which may slightly change the aspect ratio of some interior triangles.

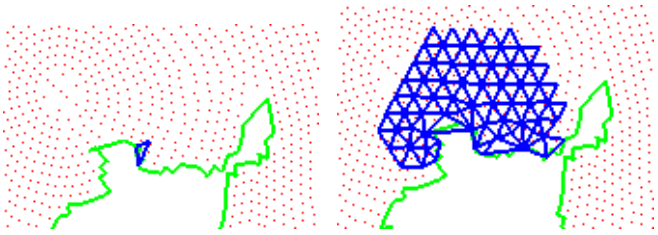


(a) (b)

Figure 7.6: (a) Vertex pattern used for resampling a patch. Red squares are samples along the boundary. Black and blue squares are samples inside the patch. The blue ones are too close to the red squares and are rejected (deleted). The red and black ones, together with the connection between them comprise a mass-spring system. (b) The resulting nearly regular mesh.

In practice, we start from one triangle on the boundary of the parameter patch which takes one interior vertex and two cut vertices, and propagate triangles around using an approach similar to the ball-pivoting strategy of Bernardini et al. [13]. Starting from one triangle, we keep flipping the triangle about its edges to create new triangles. The flipping stops when the current edge is part of the patch boundary or the flipped triangle has already been created. We show the triangles created after different numbers of propagation steps in Figure 7.7.

For each interior vertex, we need to find its *nearest patch point* (NPP). For cut vertices, their NPP are their nearest points in the *patch boundary curve*. When we find a new triangle



(a) (b)

Figure 7.7: Triangles created inside a patch after (a) 1 step, and (b) 100 steps. Red dots are the original points and the green line represents a cut curve. The resulting triangles are shown in blue.

by flipping, we will need to find the NPP of the new vertex from the NPP of the vertices of the edge. Starting from the NPP of these two vertices of the edge, we repeatedly find the nearest neighbor point in the parameter space to the new vertex until the process stops. This process is illustrated in Figure 7.8. Here the triangle $\overline{v_0v_1v_2}$ is flipped about edge (v_0, v_1) to create a new triangle $\overline{v_0v_1v_3}$. For the new interior vertex v_3, we need to compute its 2D parameter values as well as its 3D location. The 2D parameters are obtained with ease. To compute the 3D location, we first need to find the NPP of v_3, using the process described above. During this triangle-flipping process, we check the shortest distance between v_3 and any cut vertex. If this is $\leq \frac{L}{2}$, v_3 is too close to the boundary and is discarded.

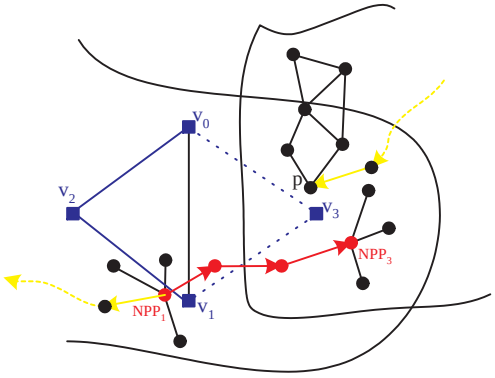


Figure 7.8: The path (red) to find the NPP of an interior vertex v_3. Due to patch overlapping, NPP_3 (the NPP of v_3) is found by following the path from NPP_1. Note that although p is closest to v_3 in the 2D parameter space, NPP_3 is closest to v_3 in 3D space.

Note that the triangles around the boundary are not equilateral, even in the 2D parameter space. To reduce this problem, we relax the obtained mesh using a mass-spring system. Each vertex becomes a mass point and any edge connecting vertices becomes a spring. The 2D parameters of the interior vertices are then relaxed to minimize the following energy function, while the 2D parameters of the vertices along the cut are fixed:

$$E = \sum_i (\sum_{q_j \in kNN(q_i)} (q_i - q_j))^2 \tag{7.3}$$

The 3D location of an interior vertex is computed from the NPP and its kNN in $\mathcal{G}_p$. We use a moving-least-square fitting scheme [64, 135] to find the 3D location of this interior vertex. In this case, we compute the X, Y, Z coordinates separately. We try to minimize the following error function:

$$E(s) = \sum_{i=1}^{N} w_i(p_i)(s(p_i) - f_i)^2 \tag{7.4}$$

where

$$s(u, v) = a_0 + a_1 u + a_2 v + a_3 u^2 + a_4 v^2 + a_5 uv \tag{7.5}$$

is a low order polynomial surface and f_i is the height value related with each point. $w_i(p) = \frac{e^{-\alpha d_i^2(p)}}{d_i^2(p)}$ is the point-wise weight function and $d_i(p)$ is the distance from the query location p to point p_i. The parameter α controls the influence of vicinity features. Using the X, Y, Z coordinates of the points to define f, one obtains the various sets of coefficients a_0 to a_5, from which the the 3D location of the interior vertices can be resampled.

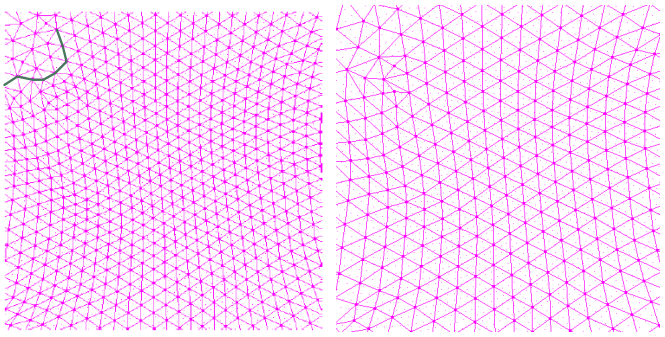
7.5 Results and Discussion

We have used our algorithm to reconstruct meshes from a variety of point cloud datasets, including some real-world medical and 3D scanning data. All the experiments were carried on a Pentium 4 2.2 GHz PC with 768M memory. The human colon dataset was obtained from CT scans of a real human body. The 3D points are computed using the Marching Cubes algorithm. We show the cuts generated by our algorithm for the parameterization of the virtual colon dataset in Figure 7.11(a). The reconstruction result is shown in Figure 7.11(b) and Figure 7.11(c). Figure 7.12 presents a view of the inside of the colon. The resulting mesh is composed of triangles with good aspect ratios. Note that since the 3D mesh is seen in perspective, the projection of the triangles whose normals are almost perpendicular to the viewing direction appears distorted. The obtained mesh allows very nice colon visualizations during the exploration of the model, which contains 61,334 points. It took 30.23s for parameter propagation and 50.04s for meshing using $L = E$. The number of resulting triangles and vertices are 33,566 and 18,743, respectively.

The sphere model is used to show the quality of the resulting mesh. Figure 7.9 shows two meshes with different resolutions created from the original point cloud. Due to the constraint of shared vertices along cuts, one can expect some irregular arrangement of edges along the cuts. While within a patch, edges of triangles only have six possible directions, edges along a cut can have arbitrary directions. This tends to happen regardless of the resolution used to extract the resulting mesh. The sphere model has 10,270 points. It took 3.02s for the propagation, and 4.87s and 2.44s to obtain the meshing results for these two models, respectively. For the model with $L = E$, we have 987 triangles and 710 vertices. For the model with $L = 1.5E$, we obtain 455 triangles and 320 vertices.

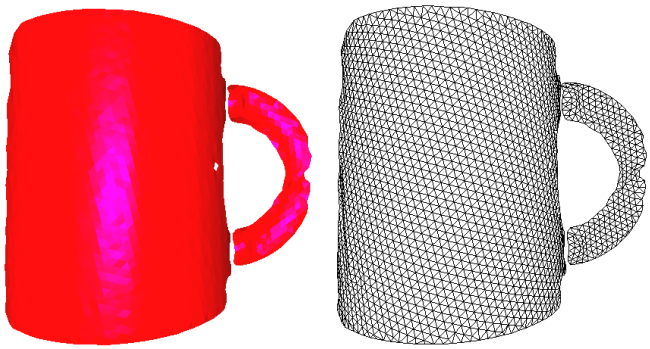
The reconstruction of the mug model is shown in Figure 7.10. Note that the boundaries of this model, including a small hole on the surface, are well preserved by our algorithm.

One should note the regular shape of the resulting triangles (Figure 7.10 b). This model has 34,916 points. It took 6.92s and 6.77s for the propagation and meshing, respectively. The resulting model with $L = E$ contains 3,504 triangles and 1,872 vertices.



(a) (b)

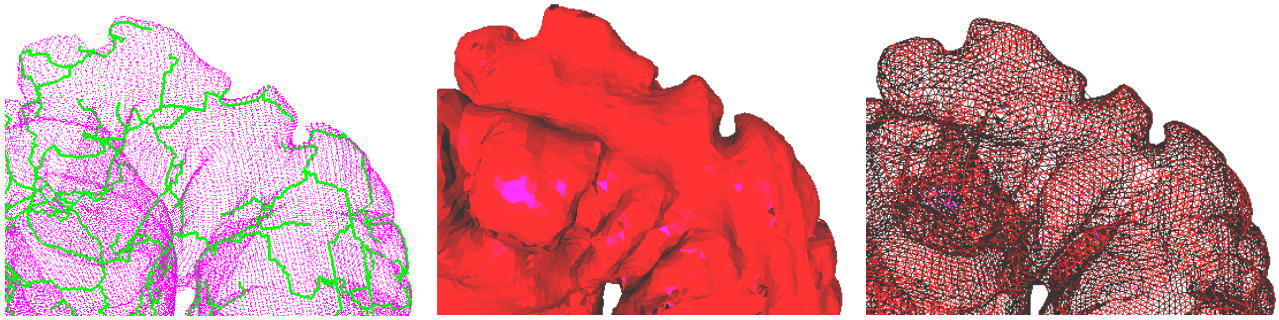
Figure 7.9: Reconstruction result of the sphere model: (a) $L = E$. (b) $L = 1.5E$. The green curve is a cut, where the triangles created around it have distorted shapes.



(a) (b)

Figure 7.10: Reconstruction result of the mug model: (a) Shaded model. (b) Wireframe. Note the regular shapes of the resulting triangles.

The resulting parameterization can be used to perform texture mapping (Figure 7.13a) and bump mapping (Figure 7.13b). While cuts (parameter discontinuities) may introduce seams when using textures containing regular patterns, this problem can be alleviated with the use of stochastic textures (Figure 7.13a). Haitao et al. [149] have proposed an approach to synthesize a seamless textures using this kind of parameterization.



(a) (b) (c)

Figure 7.11: Reconstruction of a human colon model: (a) the point model and the cuts generated using point parameterization; (b) the shaded view of the reconstructed model; and (c) the wireframe view.

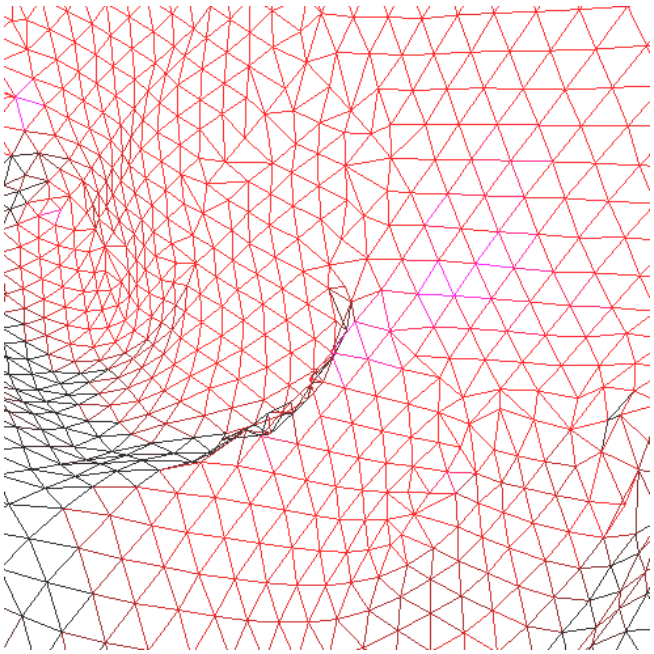
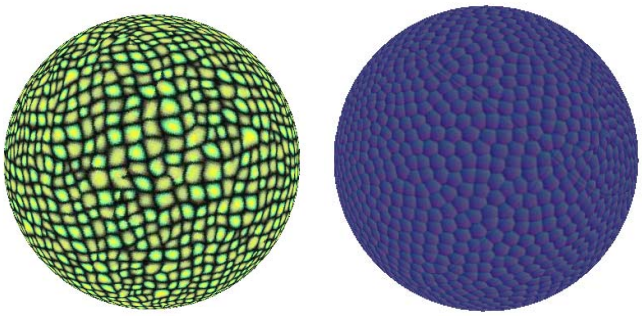


Figure 7.12: An inside view of the reconstructed colon model seen in perspective, which cause the projection of some triangles to appear to have poor aspect ratio. In fact, all triangles are nearly equilateral.

In our approach, a global parameterization assures a nearly globally isometric mapping between the 3D space and the parameterization space, producing nearly equilateral triangles. Since it is a two-step algorithm, the quality of the parameterization is critical for the meshing result. The chosen parameterization approach is robust to varying sampling rate



(a) (b)

Figure 7.13: Applications of our point parameterization for (a) texture mapping and (b) bump mapping.

and a moderate amount of noise. Because the parameterization relies on local operations, it could handle complex topologies as long as the local neighborhood represents a topological disk. However, the error accumulates due to the progressive nature of the parameterization algorithm. The number of the parameter patches also increase with increased noise level. In this case, the mesh tends to become more irregular with longer cuts.

In this approach, we do not fill the holes. Instead, we preserve surface boundaries, including hole boundaries. One may apply the algorithm of Sharf et al. [114] to fill the holes directly on the point model before generating the mesh.

IF THE POINT CLOUD BE NATURALLY SEPARATED INTO SEVERAL CLUSTERS, THE PARAMETERIZATION AND RECONSTRUCTION OF THESE CLUSTERS ARE CONDUCTED INDEPENDENTLY. IF ONE CLUSTER REPRESENTS A SURFACE PATCH WITH A NUMBER OF SMALL HOLES, THE RESULT COULD BE VERY BAD DUE TO THE FACT THAT THERE EXIST SO MANY CUTS AND THESE CUTS ARE CLOSE TO EACH OTHER, DISTORTING THE TRIANGLE MESH GREATLY.

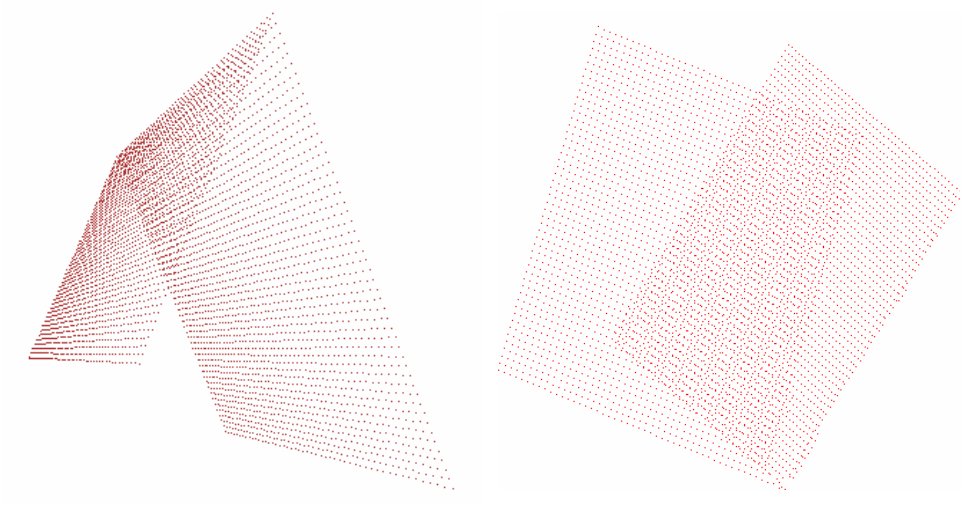
IN ADDITION, THE POINT PARAMETERIZATION APPROACH ADOPTED HERE IN THE ALGORITHM DOES NOT TAKE SHAPE FEATURES INTO ACCOUNT. THE PARAMETERS OF A POINT WILL BE DISTORTED GREATLY BECAUSE THE EUCLIDEAN DISTANCE DIFFERS

GREATLY WITH THE GEODESIC DISTANCE. IN THE MESHING STAGE, THE ALGORITHM ALSO DOES NOT PAY ATTENTION TO THE SHARP FEATURES.

IF THE FEATURE IS A REALLY SHARP (ACUTE ANGLE) ONE, THE PARAMETERIZATION MAY FOLD OVER IN AN INCORRECT DIRECTION. AN EXAMPLE OF THIS PROBLEM IS SHOWN IN FIGURE 7.14. WE EXPECT ITS POINT PARAMETERIZATION PATCH OF THIS MODEL WITH SHARP EDGES IS A 2D RECTANGLE. THE REASON FOR THIS PROBLEM IS THAT THE PARAMETERS OF A POINT ON ONE SIDE OF THE SHARP EDGE MAY BE COMPUTED BASED ON ITS k-NEIGHBORHOOD ON THE OTHER SIDE. BECAUSE THE SHARP EDGE FORMS AN ACUTE ANGLE, THE POINT WILL BE PROJECTED ONTO THE WRONG SIDE OF THE 2D EDGE ON THE PARAMETER SPACE.

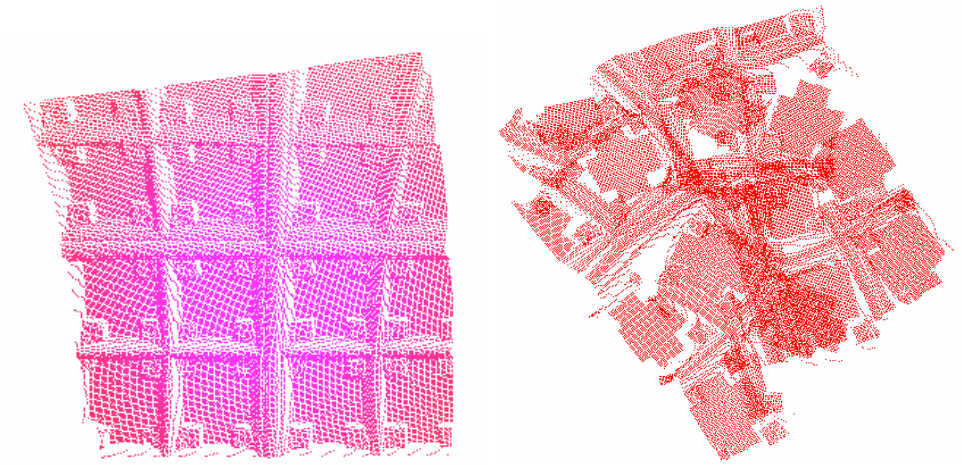
IF WE HAVE A MODEL WITH ABRUPT CHANGE OF SAMPLE RATES, THE POINT MODEL IS ACTUALLY SEPARATED INTO SEVERAL SMALL PATCHES. ALTHOUGH THE PARAMETERIZATION OF EACH SMALL PATCH IS REASONABLE, THESE SMALL PATCHES ARE NOT WELL ORIENTED GLOBALLY, AS SHOWN IN FIGURE 7.15. FOR THOSE POINTS WITH A LOWER SAMPLING RATE, THEY ARE PARAMETERIZED BY ONE *patch* POINT MOST OF THE TIME. THEREFORE, THE ERROR ACCUMULATES AND PROPAGATES EASILY.

The main limitation of our approach lies in that the parameterization procedure is not optimized within each parameter patch. Due to error propagation that happens during parameter propagation, points visited later in the same patch might accumulate considerable distortion, for which we do not have a measurement/control for the error bound. In addition, the discontinuity of the parameterization across cuts tends to exhibit seams when it is used for mapping highly regular texture patterns. Note that even in the presence of parameter discontinuities, the resulting meshes will be nice and regular. Thus, one could compute some global parameterization to the mesh as a post-processing using, for instance, the approaches described in [46, 56]. Alternatively, one could apply a transition rule at the



(a) (b)

Figure 7.14: Problems of parameterization with respect to abrupt change of sampling rates: (a) a point model with a sharp edge; and (b)the parameterization patch of the model.



(a) (b)

Figure 7.15: The indoor scene model has abruptly varying sample rates: (a) the point model; and (b) the parameterization patch.

vicinity of the cuts consisting of blending the multiple parameters (one from each adjacent patch that shares the vertex) associated to the cut vertices.

This chapter presented a new surface reconstruction algorithm for producing meshes with regular triangles from unorganized point clouds. The algorithm consists of generating

a point parameterization before obtaining the resulting mesh. This approach preserves detailed geometric information and gives the meshing process great flexibility. The obtained meshes consist of nearly equilateral triangles, thus leading to better visualizations and making them suitable for specialized operations such as finite element analysis. In the future, we would like to explicitly find certain kinds of geometric details before parameterization, such as surface boundary [142] and sharp features [99]. These features could then be used as additional constraints for propagation. By saving them as part of the cuts, vertices could be created along them, guaranteeing that they are more faithfully preserved. In addition, we intend to use a different re-sampling pattern to create meshes with the size of triangles adapted to some geometrical properties, such as curvature. Right now, all extracted triangles have approximately the same size.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, we have described several novel techniques and algorithms as an attempt to solve a number of challenging problems in surface reconstruction from point clouds. Our work includes missing region recovery, hole filling, reconstructing noisy and non-manifold point models, creating regular meshes, and reconstructing surfaces from endoscopic videos. In short, our completed projects can be summarized as follows.

We have proposed an novel framework to efficiently utilize user knowledge to recover missing regions in point models from indoor environments. In this framework, large planar regions are first identified by Hough transform, thus separating the points into two parts: planar regions and non-planar regions. Planar regions are replaced directly by polygons while the associated texture is recovered by exploiting the symmetry information. Non-planar surfaces are reconstructed with some missing regions are recovered using the symmetry of man-made objects. The complete model is obtained in the end by combining planar polygons and non-planar surfaces.

We have implemented a hole-filling algorithm for meshes. This algorithm could be

incorporated into the system mentioned above to eliminate the remaining holes. Otherwise it could be used as an independent postprocess module for hole filling. We first find the vicinity points of each hole. These points provide clues for the shape and shading of the missing region. A moving least squares fitting approach is then employed to compute vertices inside that hole. These vertices are later triangulated to create a tight and smooth filling.

We have also introduced a novel method to reconstruct surfaces from noisy points without point normals. This approach creates an oriented charge for each voxel adjacent to the actual surface. Oriented charge represents the local distance field and approximates the local shape. This novel representation enables a hierarchical and adaptive construction of the global distance field. The resulting mesh is extracted from the distance field using the Marching Cubes algorithm.

In addition, we have presented a generalized framework to reconstruct non-manifold surfaces as well as manifold surfaces. The point cloud is first converted into voxels. We then thin the voxel surface to eliminate noise and identify the local shape for each voxel. Voxels representing non-manifold regions are thus located. We use a direct meshing algorithm to convert voxels into a mesh while the non-manifold regions are specially handled.

We have described a new algorithm to reconstruct regular meshes from point models. Regular meshes are composed of nearly equilateral triangles, which provide a great visual quality for rendering and a suitable input for some specific applications, such as Finite Element Analysis. We first obtain a nearly isometric point parameterization for the points. Holes and patch boundaries are identified and vertices are created along them. A regular sampling pattern is then imposed on the 2D parameter domain to create meshes with regular triangles.

THE PROPOSED ALGORITHMS AND FRAMEWORKS ARE DIRECTLY TARGETED TOWARDS IMPERFECTIONS. IN OTHER CASES, ONE COULD USE THE EXISTING POPULAR RECONSTRUCTION ALGORITHMS, SUCH AS THE BALL-PIVOTING ALGORITHM [13] AND THE MULTIPLE PARTITION OF UNITY ALGORITHM [91]. IF THE MODEL IS IN THE CASE THAT USER HAS A PRIORI KNOWLEDGE ABOUT THE SCENE TO BE RECONSTRUCTED, ESPECIALLY THE EXISTENCE OF LARGE PLANAR REGIONS AND OBJECTS WITH BILATERAL SYMMETRY, THE ALGORITHM FOR RECOVERING MISSING REGIONS COULD BE USED TO RECOVER THE MISSING REGIONS. THE POST-PROCESS HOLE-FILLING MODULE CAN BE USED TO FILL THE HOLES IN THE MESH, WHICH COULD BE OBTAINED USING VARIOUS RECONSTRUCTION ALGORITHMS. THIS MODULE ENSURES A SMOOTH AND REASONABLE FILLING OF THE HOLE REGIONS. FOR CLOSED-SURFACE MANIFOLD MODELS, OUR ALGORITHM OF COMPUTING ORIENTED CHARGES FOR A GLOBAL DISTANCE FIELD IS CAPABLE OF FILLING SMALL HOLES AND HANDLE SIGNIFICANT NOISE. TO PRESERVE BOUNDARIES OR TO RECONSTRUCT NON-MANIFOLD SURFACES, ONE COULD USE THE PROPOSED FRAMEWORK, WHICH ALSO HAS THE CAPABILITY TO FILL GAPS. TO GENERATE REGULAR MESHES FROM THE POINT MODEL, THE PROPOSED ALGORITHM COMPUTES THE NEARLY ISOMETRIC PARAMETERIZATION IN THE FIRST PLACE AND NEARLY EQUILATERAL TRIANGLES ARE CREATED THEN FROM THE OBTAINED PARAMETERIZATION.

USER SHOULD SELECT THE MOST APPROPRIATE ALGORITHM FOR HIS RECONSTRUCTION BASED ON HOW MUCH INFORMATION IS ASSOCIATED WITH THE INPUT DATA, WHAT ASSUMPTION WE COULD MAKE ABOUT THE MISSING REGION, AND WHAT IS REQUIRED FOR THE RESULTING MESH. IN A NUTSHELL, ONE MIGHT SELECT THE SPECIFIC ALGORITHM INTRODUCED IN THE PREVIOUS CHAPTERS ACCORDING TO TABLE 8.1.

Table 8.1: The simple rules to select the appropriate algorithm

Features	Algorithm
Chapter 3	Known knowledge of planarity and symmetry
Chapter 4	Triangular mesh with holes
Chapter 5	Very noisy closed-surface model
Chapter 6	Preserved boundaries or non-manifold model
Chapter 7	Require a regular mesh as output

8.2 Future Work

8.2.1 Short-term Future Work

During the range scanning process, missing regions are unavoidable in most cases due to occlusion and limited accessibility. In addition, the sampling rate for different directions may vary dramatically. For example, samples are more sparsely scattered in the Y direction for the top of a tall building when we scan this building from the ground. We plan to propose a generic hole filling algorithm which exploits more user knowledge about the object, besides planarity and symmetry, to fill the holes. A possible solution is to construct a shape database. And then we can query in the for the best guess of the missing regions. Furthermore, we plan to propose a novel texture synthesis algorithm to automatically analyze the type of texture of the object from the "good" regions and synthesize new textures for the missing regions.

We have presented several algorithms to reconstruct surfaces from noisy points without point normals. However, these algorithms are not capable of preserving important features. To enhance these algorithms, a global operation is required to identify features. The challenges are how to design an effective means to define the features, how to identify these features robustly in noisy environments, and how to effectively use the features as the constraints for the existing algorithms. In Chapter 7, we have proposed an algorithm to create

regular meshes based on the nearly isometric parameterization. Certain geometric details such as surface boundary [142] and sharp features [99], can be applied as the constraints for the parameterization. Besides, we intend to use a different re-sampling pattern to create meshes with the size of triangles adapted to the geometrical properties such as curvature.

We also want to accelerate the proposed algorithms by exploring the inherent parallelism of these algorithms. For example, in Chapter 6, the local topological type of each voxel could be identified in parallel. Thanks to the rapid development of graphics hardware, we are able to do so using the Graphics Processing Unit (GPU). One issue is that the data sent to the GPU have to be organized in the texture format. The challenge is how to make it an effective process. The other issue is how to balance the load between CPU and GPU to achieve the best performance.

8.2.2 Long-term Future Work

Our long-term future work will be to devise a smart system for surface reconstruction from points. We found that the type of inputs for reconstruction may vary dramatically, depending on the acquisition devices, environment, lighting conditions, and the material of the surface. At the same time, user may want different results (*e.g.*, different resolutions, different error bounds and different processing times). And we have a number of algorithms, which could be selected. Our plan is then to develop a system which could analyze the requirements of users and find the best algorithm to meet their needs. The features of this system should include:

- Specification of the input points and the performance. Users need to specify the type and certain parameters of the input points. They also need to specify the requirement for the resulting mesh and the performance.

- Algorithm selection and assembly. Different algorithms should be implemented as

plug-ins with clearly defined input/ouput. The system will select algorithms based on user's requirements. If the requirements could not be met by the current collection of algorithms, the system should report to the user. User may change his requirements or the system could select the optimal algorithms. After selecting the best algorithms, the system may assemble them into a pipeline.

- Executable export. Base on the result of algorithm selection and assembly, the system should output an optimized executable program or an optimized library to the user.

In our vision, the ranger scanning process will become more accurate, more efficient, more robust, and more automatic. The acquired samples will cover the whole surface with more detailed information. In addition, range scanners will automatically find the best position for the next shot. With the aid of high resolution GPS, a ranger device will be well aware of its motion during the scanning process, thus enabling an accurate registration. And the sampling density might be always sufficient enough. Furthermore, point graphics will be well developed in ten years and graphics hardware will support points directly as well as polygons. From this point of view, we will face a new set of challenges, such as efficient retrieval of features from point models and animation using point models.

Bibliography

[1] B. Adams and P. Dutré. Interactive boolean operations on surfel-bounded solids. *SIGGRAPH*, pages 26–31, 2003.

[2] A. Adamson and M. Alexa. Approximating bounded, non-orientable surfaces from points. *Shape Modeling International*, pages 243–252, 2004.

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. Silva. Point set surfaces. *IEEE Visualization*, pages 21–28, 2001.

[4] D. Aliaga and I. Carlbom. Plenoptic stitching: A scalable method for reconstructing 3D interactive walkthroughs. *SIGGRAPH*, pages 443–450, 2001.

[5] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. Anisotropic polygonal remeshing. *SIGGRAPH*, pages 485–493, 2003.

[6] P. Alliez, M. Meyer, B. Levy, and M. Desbrun. Interactive geometry remeshing. *SIGGRAPH*, pages 347–354, 2002.

[7] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. *SIGGRAPH*, pages 415–421, 1998.

[8] N. Amenta, S. Choi, T. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. *16th ACM Symposium on Computational Geometry*, pages 213–222, 2000.

[9] N. Amenta, S. Choi, and K. Kulluri. The power crust. *Sixth ACM Symposium on Solid Modeling and Applications*, pages 249–260, 2001.

[10] N. Amenta and Y. J. Kil. Defining point-set surfaces. *SIGGRAPH*, pages 264–270, 2004.

[11] S. Azernikov, A. Miropolsky, and A. Fischer. Surface reconstruction of freeform objects based on multiresolution volumetric method. *8th ACM Symposium on Solid Modeling and Applications*, pages 115–126, 2003.

[12] C. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. *SIGGRAPH*, pages 109–118, 1995.

[13] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taublin. The ball-pivoting algorithm for surface reconstruction. *IEEE Trans. on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[14] F. Bernardini and H. Rushmeier. The 3D model acquisition pipeline. *Computer Graphics Forum*, 21(2):149–172, 2002.

[15] M. Bertalmio, G. Shapiro, V. Caselles, and C. Ballester. Image inpainting. *SIGGRAPH*, pages 417–424, 2000.

[16] P. Besl. *Advances in Machine Vision, Chapter 1 - Active Optical Range Sensors*. Springer Verlag, 1989.

[17] P. Besl and A. Jain. Segmentation through variable-order surface fitting. *IEEE Trans. on Pattern Analysis and Machine Inteligence*, 10(2):167–192, 1988.

[18] P. Besl and N. Mckay. A method for registration of 3D shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(5):239–256, 1992.

[19] A. Blake and M. Isard. *Active Contours*. Springer-Verlag, 1998.

[20] J. Bloomenthal. An implicit surface polygonizer. In P. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[21] J. Bloomenthal, C. Bajaj, J. Blinn, M. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[22] J. Bloomenthal and K. Ferguson. Polygonization of non-manifold implicit surfaces. *SIGGRAPH*, pages 309–316, 1995.

[23] M. Botsch and L. Kobbelt. A robust procedure to eliminate degenerate faces from triangle meshes. *Vision Modeling and Visualization Conference*, pages 283–290, 2001.

[24] T. Boult and J. Kender. Visual surface reconstruction using sparse depth data. *Computer Vision and Pattern Recognition*, pages 68–76, 1986.

[25] O. Bunsen and G. Fleischmann. Mesh optimization for animation purposes. *Simulation & Animation, Society for Computer Simulation International*, pages 66–75, 1997.

[26] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans. Reconstruction and representation of 3D objects with radial basis functions. *SIGGRAPH*, pages 67–76, 2001.

[27] J. Carr, R. Beatson, B. McCallum, W. Fright, T. McLennan, and T. Mitchell. Smooth surface reconstruction from noisy range data. *ACM GRAPHITE*, pages 119–126, 2003.

[28] E. Chen. Quicktime vr - an image-based approach to virtual environment navigation. *SIGGRAPH*, pages 29–38, 1995.

[29] U. Clarenz, U. Diewald, G. Dziuk, M. Rumpf, and R. Dusu. A finite element method for surface restoration with smooth boundary conditions. *Computer Aided Geometric Design*, 21(5):427–445, 2004.

[30] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *SIGGRAPH*, pages 303–312, 1996.

[31] J. Davis, S. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffision. *First Symposium on 3*D *Data Processing, Visualization, Transmission*, pages 428–438, 2002.

[32] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum (Eurographics)*, 21(3):209–218, 2002.

[33] T. Dey, J. Giesen, and J. Hudson. Delaunay based shape reconstruction from large data. *IEEE Symposium in Parallel and Large Data Visualization and Graphics*, pages 19–27, 2001.

[34] H. Dinh, G. Turk, and G. Slabaugh. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(10):1358–1371, 2002.

[35] H. Edelsbrunner and E. Muche. Three-dimensional alpha shapes. *ACM Trans. on Graphics*, 13:43–72, 1994.

[36] A. Efros and W. Freeman. Image quilting for texture synthesis and transfer. *SIGGRAPH*, pages 341–348, 2001.

[37] L. Fang and D. Gossard. Multidimensional curve fitting to unorganized data points by nonlinear minimization. *Computer-Aided Design*, 21(1):48–58, 1995.

[38] S. Fleishman, D. Cohen-Or, M. Alexa, and C. Silva. Progressive point set surfaces. *ACM Transactions on Graphics*, 22(2):997–1011, 2003.

[39] S. Fleishman, D. Cohen-Or, and C. Silva. Robust moving least-squares fitting with sharp features. *SIGGRAPH*, pages 544–552, 2005.

[40] M. Floater and M. Reimers. Meshless parameterization and surface reconstruction. *Comp. Aided Geom. Design.*, 18:77–92, 2001.

[41] R. Franke. Scattered data interpolation: tests of some methods. *Mathematics of Computation*, 38:181–200, 1982.

[42] R. Franke and G. Nielson. Smooth interpolation of large sets of scattered data. *Int. Journal for Numerical Methods in Engeneering*, 15:1691–1704, 1980.

[43] M. Gopi and S. Krishnan. A fast and efficient projection based approach for surface reconstruction. *International Journal of High Performance Computer Graphics, Multimedia and Visualisation*, 1(1):1–12, 2000.

[44] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. *SIGGRAPH*, pages 43–54, 1996.

[45] G. Gotsman, X. Gu, and A. Sheffer. Fundamentals of spherical parameterization. *SIGGRAPH*, 2003.

[46] X. Gu, S. Gortler, and H. Hoppe. Geometry images. *SIGGRAPH*, pages 355–361, 2002.

[47] S. Gumhold, X. Wang, and R. McLeod. Feature extraction from point clouds. *10th International Meshing Roundtable*, pages 293–305, 2001.

[48] I. Hargittai and M. Hargittai. *Symmetry: A Unifying Concept*. Shelter Publications Inc., 1994.

[49] R. Hoffman and A. Jain. Segmentation and classification of range images. *IEEE Trans. on Pattern Analysis and Machine Inteligence*, 9(5):608–620, 1987.

[50] T. Hook. Real-time shaded NC milling display. *SIGGRAPH*, pages 15–20, 1986.

[51] B. Hoover, G. Jean-Baptiste, X. Jiang, P. Flynn, H. Bunke, D. Goldgof, K. Bowyer, D. Eggert, A. Fitzgibbon, and R. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Trans. on Pattern Analysis and Machine Inteligence*, pages 673–689, 1996.

[52] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH*, pages 71–78, 1992.

[53] K. Hormann and M. Reimers. Triangulating point clouds with spherical topology. *Curve and Surface Design*, pages 215–224, 2003.

[54] P. Jenke, M. Wand, M. Bokeloh, A. Schilling, and W. Strasser. Bayesian point cloud reconstruction. *EUROGRAPHICS*, 2006.

[55] W. Jeong, K. Kähler, J. Haber, and H. Seidel. Automatic generation of subdivision surface head models from point cloud data. *Graphics Interface*, pages 181–188, 2002.

[56] M. Jin, Y. Wang, S. Yau, and X. Gu. Optimal global conformal surface parameterization. *IEEE Visualization*, pages 267–274, 2004.

[57] T. Ju. Robust repair of polygonal models. *SIGGRAPH*, pages 888–895, 2004.

[58] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *SIGGRAPH*, pages 339–346, 2002.

[59] T. Kanade, P. Rander, and P. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–37, 1997.

[60] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. *Computer Vision and Pattern Recognition*, pages 196–202, 1996.

[61] M. Kazhdan. Reconstruction of solid models from oriented point sets. *Symposium on Geometry Processing*, pages 73–82, 2005.

[62] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. *Symposium on Geometry Processing*, pages 61–70, 2006.

[63] D. Keren and C. Gotsman. Fitting curve and surfaces with constrained implicit polynomials. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(1):21–31, 1999.

[64] P. Lancaster and K. Salkauskas. *Curve and Surface Fitting: An Introduction*. Academic Press, 1986.

[65] M. Lee and G. Medioni. Segmented surface description from stereo data. *Computer Vision and Pattern Recognition*, 1998.

[66] H. Lensch, J. Kautz, M. Goesele, J. Lang, and H. Seidel. Virtualizing real-world objects. *Computer Graphics International*, pages 134–141, 2003.

[67] A. Leonardis, A. Gupta, and R. Bajcsy. Segmentation of range images as the search for geometric parametric models. *International Journal of Computer Vision*, 14(3):253–277, 1995.

[68] J. Leou and W. Tsai. Automatic rotational symmetry determination for shape analysis. *Pattern Recognition*, 20(6):571–582, 1987.

[69] D. Levin. The approximation power of moving least squares. *Mathematics of Computation*, 67(224):1517–1531, 1998.

[70] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. *SIGGRAPH*, pages 131–144, 2000.

[71] P. Liepa. Filling holes in meshes. *Symposium on Geometry Processing*, pages 200–205, 2003.

[72] Y. Liu. Computational symmetry. Technical Report CMU-RI-TR-00-31, The Robotics Institute, Carnegie Mellon University, 2000.

[73] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH*, pages 163–169, 1987.

[74] F. Losasso, H. Hoppe, S. Schaefer, and J. Warren. Smooth geometry images. *Eurographics Symposium on Geometry Processing*, pages 138–145, 2003.

[75] W. Mark. Post-rendering 3D image warping: Visibility, reconstruction, and performance for depth-image warping. Ph.D. Dissertation. UNC Computer Science Technical Report TR99-022, University of North Carolina, 1999.

[76] K. Matsushita and T. Kaneko. Efficient and handy texture mapping on 3D surfaces. *Computer Graphics Forum (Eurographics)*, 18(3):349–357, 1999.

[77] W. Matusik, C. Buehler, R. Raskar, S. Gortler, and L. McMillan. Image based visual hulls. *SIGGRAPH*, pages 367–374, 2000.

[78] D. McAllister, L. Nyland, V. Popescu, A. Lastra, and C. McCue. Real time rendering of real world environments. *Rendering Techniques*, pages 145–160, 1999.

[79] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *SIGGRAPH*, pages 39–46, 1995.

[80] V. Mello, L. Velho, and G. Taubin. Estimating the in/out function of a surface represented by points. *8th ACM symposium on Solid modeling and applications*, pages 108–114, 2003.

[81] R. Mencl and H. Müller. Graph-based surface reconstruction using structures in scattered point sets. *Computer Graphics International*, pages 298–311, 1998.

[82] Meshlab. http://meshlab.sourceforge.net.

[83] B. Mills, F. Langbein, A. Marshall, and R. Martin. Approximate symmetry detection for reverse engineering. *Proceedings of the ACM Symposium on Solid Modeling and Applications*, pages 241–248, 2001.

[84] N. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. *SIGGRAPH*, pages 560–568, 2006.

[85] B. Morse, T. Yoo, P. Rheingans, D. Chen, and K. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. *Shape Modeling International*, pages 89–98, 2001.

[86] D. Nehab, S. Rusinkiewicz, J. Davis, and R. Ramamoorthi. Efficiently combining positions and normals for precise 3D geometry. *SIGGRAPH*, pages 536–543, 2005.

[87] P. Neugebauer and K. Klein. Texturing 3D models of real world objects from multiple unregistered photoraphics views. *Computer Graphics Forum (Eurographics)*, 18(3):245–256, 1999.

[88] O. Nilsson, D. Breen, and K. Museth. Surface reconstruction via contour metamorphosis: An eulerian approach with lagrangian particle tracking. *IEEE Visualization*, pages 407– 414, 2005.

[89] L. Nyland, A. Lastra, D. McAllister, V. Popescu, and C. McCue. Capturing,processing and rendering real-world scenes. *Videometrics and Optical Methods for 3*D *Shape Measurement, Electronic Imaging, SPIE*, 4309, 2001.

[90] L. Nyland, D. McAllister, V. Popescu, C. McCue, A. Lastra, P. Rademacher, M. Oliveira, G. Bishop, G. Meenakshisundaram, M. Cutts, and H. Fuchs. The impact of dense range data on computer graphics. *Multi-View Modeling and Analysis Workshop*, pages 3–10, 1999.

[91] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel. Multi-level partition of unity implicits. *SIGGRAPH*, pages 463–470, 2003.

[92] Y. Ohtake, A. Belyaev, and H. Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. *Shape Modeling International*, pages 292–300, 2003.

[93] M. Oliveira, B. Bowen, R. McKenna, and Y. Chang. Fast digital image inpainting. *International Conference on Visualization, Imaging and Image Processing*, pages 261–265, 2001.

[94] D. O'Mara and R. Owens. Measuring bilateral symmetry in three dimensional magnetic resonance images. *TENCON Digital Signal Processing Applications*, pages 151–156, 1996.

[95] A. Oppenheim, R. Schafer, and J. Buck. *Discrete-Time Signal Processing (2nd Edition)*. Prentice Hall, 1999.

[96] P. Palojärvi, K. Määttä, and J. Kostamovaara. Integrated time-of-flight laser radar. *IEEE Transactions on Instrumentation and Measurement*, 46(4):996–999, 1997.

[97] S. Park, X. Guo, H. Shin, and H. Qin. Surface completion for shape and appearance. *The Visual Computer*, 22(3):168–180, 2006.

[98] S. Parui and D. Majumder. Symmetry analysis by computer. *Pattern Recognition*, 16(1):63–67, 1983.

[99] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. *SIGGRAPH*, 22(3):641 – 650, 2003.

[100] A. Pentland. Automatic extraction of deformable part models. *International Journal of Computer Vision*, 4:107 – 126, 1990.

[101] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A planar reflective symmetry transform for 3D shapes. *SIGGRAPH*, pages 549–559, 2006.

[102] E. Prados and O. Faugeras. Shape from shading: a well-posed problem? *IEEE Conference on Computer Vision and Pattern Recognition*, pages 870–877, 2005.

[103] E. Praun and H. Hoppe. Spherical parametrization and remeshing. *SIGGRAPH*, 2003.

[104] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, second edition*. Cambridge University Press, 1992.

[105] M. Proesmans, L. V. Gool, and F. Defoort. Reading between the lines - a method for extracting dynamic 3D with texture. *IEEE International Conference on Computer Vision*, pages 1081–1086, 1998.

[106] R. Raskar, et al. The office of the future: A unified approach to image-based modeling. *SIGGRAPH*, pages 179–188, 1998.

[107] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3D model acquisition. *SIGGRAPH*, 21(3):438–446, 2002.

[108] H. Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[109] V. Savchenko and N. Kojekine. An approach to blend surfaces. *Computer Graphics International*, pages 139–150, 2002.

[110] V. Savchenko, A. Pasko, O. Okunev, and T. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[111] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. *Pacific Graphics*, pages 70–76, 2004.

[112] C. Scheidegger, S. Fleishman, and C. Silva. Triangulating point set surfaces with bounded error. *Symposium on Geometry Processing*, pages 63–72, 2005.

[113] S. Sclaroff and A. Pentland. Generalized implicit functions for computer graphics. *SIGGRAPH*, 25(4):247–250, 1991.

[114] A. Sharf, M. Alexa, and D. Cohen-Or. Context-based surface completion. *SIGGRAPH*, pages 878–887, 2004.

[115] A. Sharf, T. Lewiner, A. Shamir, L. Kobbelt, and D. Cohen-Or. Competing fronts for coarse-to-fine surface reconstruction. *EUROGRAPHICS*, pages 389–398, 2006.

[116] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. *23rd ACM National Conference*, pages 517–524, 1968.

[117] H. Shum and L. He. Rendering with concentric mosaics. *SIGGRAPH*, pages 296–306, 1999.

[118] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski. Bounded-distortion piecewise mesh parametrization. *IEEE Visualization*, pages 355–362, 2002.

[119] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. *SIGGRAPH*, 26:185–194, 1992.

[120] C. Tang and G. Medioni. Inference of integrated surface, curve, and junction description from sparse 3D data. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20:1206–1223, 1998.

[121] G. Taubin. Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations, with applications to edge and range image segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13:1115–1138, 1991.

[122] G. Taubin. An improved algorithm for algebraic curve and surface fitting. *Proceedings Fourth International Conference on Computer Vision*, pages 658–665, 19913.

[123] D. Terzopoulos. The computation of visible surface representations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(4):417–438, 1998.

[124] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: Deformable superquadrics. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13:703–714, 1991.

[125] I. Tobor, P. Reuter, and C. Schlick. Efficient reconstruction of large scattered geometric datasets using the partition of unity and radial basis functions. *Journal of WSCG*, 12:467–474, 2004.

[126] E. Trucco and A.Verri. *Introductory Techniques for 3*-D *Computer Graphics*. Prentice Hall, 1998.

[127] Y. Tsao and K. Fu. A parallel thinning algorithm for 3D pictures. *Computer Graphics Image Processing*, 17:315–331, 1981.

[128] G. Turk and M. Levoy. Zippered polygon meshes from range images. *SIGGRAPH*, pages 311–318, 1994.

[129] G. Turk and J. O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Georgia Institute of Technology, 1999.

[130] T. Varady, R. Martin, and J. Cox. Reverse enginnering of geometric models - an introduction. *Computer Aided Design*, 29(4):255–268, 1997.

[131] J. Verdera, V. Caselles, M. Bertalmio, and G. Sapiro. Inpainting surface holes. *International Conference on Image Processing*, pages 903–906, 2003.

[132] J. Wang, O. Hall-Holt, P. Konecny, and A. Kaufman. A hole filling strategy for reconstruction of smooth surfaces in range images. *XVI Brazilian Symposium on Computer Graphics and Image Processing*, pages 11–18, 2003.

[133] J. Wang, O. Hall-Holt, P. Konecny, and A. Kaufman. Per-pixel camera calibration for 3D range scanning. *SPIE Electronic Imaging*, 5665:342–352, 2005.

[134] J. Wang and M. Oliveira. Improved scene reconstruction from range images. *Computer Graphics Forum (Eurographics)*, 21(3):521–530, 2002.

[135] J. Wang and M. Oliveira. Filling holes on locally smooth surfaces reconstructed from point clouds. *Image and Vision Computing*, 25:103–113, 2007.

[136] J. Wang, M. Oliveira, and A. Kaufman. Reconstructing manifold and non-manifold surfaces from point clouds. *IEEE Visualization*, pages 415–422, 2005.

[137] J. Wang, M. Oliveira, H. Xie, and A. Kaufman. Surface reconstruction using oriented charges. *Computer Graphics International*, pages 122–128, 2005.

[138] D. Washburn and D. Crowe. *Symmetries of Culture: Theory and Practice of Plane Pattern Analysis*. University of Washington Press, 1988.

[139] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. *SIGGRAPH*, pages 479–488, 2000.

[140] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial basis functions of minimum degree. *Advances in Comp. Math.*, 4:389–396, 1995.

[141] H. Weyl. *Symmetry*. Princeton University Press, 1952.

[142] C. Xia, W. Hsu, M. Lee, and B. Ooi. Border: Efficient computation of boundary points. *IEEE Transactions on Knowledge and Data Engineering*, 18(3):289–303, 2006.

[143] H. Xie, K. McDonnell, and H. Qin. Surface reconstruction of noisy and defective data sets. *IEEE Visualization*, pages 259–266, 2004.

[144] H. Xie, J. Wang, J. Hua, H. Qin, and A. Kaufman. Piecewise c_1 continuous surface reconstruction of noisy point clouds via local implicit quadric regression. *IEEE Visualization*, pages 91–98, 2003.

[145] R. Yip. A hough transform technique for the detection of rotational symmetry. *Pattern Recognitions Letters*, 15:919–928, 1994.

[146] R. Yip. A hough transform technique for the detection of reflectional symmetry and skew-symmetry. *Pattern Recognitions Letters*, 21:117–130, 2000.

[147] Y. Yu, P. Debevec, J. Malik, and T. Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs. *SIGGRAPH*, pages 215–224, 1999.

[148] Y. Yu, A. Ferencz, and J. Malik. Extracting objects from range and radiance images. *IEEE Trans. on Visualization and Computer Graphics*, 7(4):351–364, 2001.

[149] H. Zhang, F. Qiu, and A. Kaufman. Fast hybrid approach for texturing point models. *Computer Graphics Forum*, 23(4):715–725, 2004.

[150] R. Zhang, P. Tsai, J. Cryer, and M. Shah. Shape from shading: A survey. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(3):690–706, 1999.

[151] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. *First IEEE Workshop on Variational and Level Set Methods*, pages 194–202, 2001.

[152] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop3D: An interactive system for point-based surface editing. *SIGGRAPH*, pages 322–329, 2002.