Piecewise Surface Reconstruction from Range Data

by

Gene Yu

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2010

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosopy.

Date	Professor George Wolberg, Ph.D. Chair of the Examining Committee
Date	Professor Theodore Brown, Ph.D. Executive Officer

Professor Michael Grossberg, Ph.D.

Professor Ioannis Stamos, Ph.D.

Dr. Michael Reed, Ph.D.

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

PIECEWISE SURFACE RECONSTRUCTION FROM RANGE DATA

by

Gene Yu

Adviser: Professor George Wolberg

Geometric modeling from range data is a long-standing problem in the field of computer graphics. It remains challenging even when the data is replete with simple surfaces such as planes and polynomials, as is common in urban scenes. The need to represent such scenes requires us to segment the data into its components. The collection of these components constitutes the final geometric model. This thesis introduces a method of piecewise surface reconstruction that fits a scene with a model composed of disjoint surfaces. The contribution of this work is the introduction of a surface evaluation method based on quantitative entropy measurements for balancing the tradeoff between accuracy and efficiency. Integrated surface evaluation enables us to produce output models that are accurate to within user-specified tolerances. Since our algorithm minimizes global criteria, it is robust to holes, occlusions, nonplanar surfaces, and missing data. Compared to methods that operate on unorganized point clouds and utilize no segmentation, our approach provides the user with greater control over the final appearance and error characteristics of the output model. A range of shape approximations such as plane, polynomial, and spline mesh surfaces can be used interchangeably. This flexibility is applicable to all scenes involving piecewise models.

Acknowledgements

I would like first of all to thank my advisor, Professor George Wolberg, for supporting me over the years, both in and out of the classroom, and for encouraging me to pursue ever higher goals in education and in science. His integrity and professionalism have set an example that I try to follow in all of my endeavors. Secondly, I would like to thank Professor Michael Grossberg, whose enthusiasm and penetrating mathematical insight spurred me on when I needed it most. I owe a debt of gratitude to Professor Ioannis Stamos, who laid the groundwork for my research, and who has been so generous with his knowledge, time, and equipment. Thanks also to his students Dr. Lingyun Liu and Dr. Cecilia Chao Chen, with whom I have been privileged to collaborate. I am grateful as well to Dr. Michael Reed for sitting on my examining committee and participating in my defense.

I am especially indebted to Dr. Siavash Zokai, whose advice and encouragement have helped me immeasurably, both in sharpening my understanding of the vast field of computer science, and in boosting my morale during the long years of study.

Finally, I wish to thank my family: my brothers Van and Shane, and my sister Fay, who supported me without fail, and to my wife Olivia, who stands by me always.

This work was supported in part by grant DOE DE-FG52-06NA27503.

For my parents

Contents

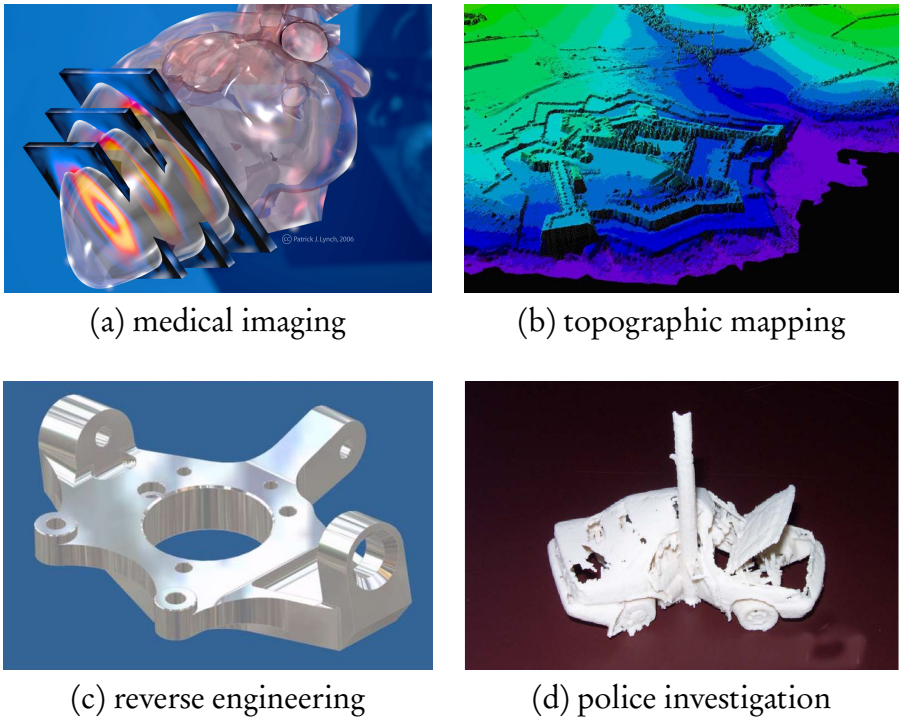
List of Figures

Chapter 1

Introduction

Geometric modeling is the process of generating a mathematical representation of an object from raw data. The purpose of the model is to describe the geometry of the object in a simple but computationally powerful form that is useful for a given application. Modeling applications are numerous and diverse, due in part to the wide variety of range sensors available. Figure 1.1 shows some examples of models of all shapes and sizes that are in use today. The central problem in modeling is evaluation. How do we determine what is a good model? This thesis presents a novel approach to model evaluation for surface reconstruction from range data.

A good model balances accuracy and complexity. It must approximate the geometry closely, without requiring an excessive amount of storage space or computational overhead. Therefore, our evaluation method must measure both the accuracy and the efficiency of the model. Accuracy is described by error, or the deviation of the surface geometry from the data. Efficiency is described by entropy, measured with respect to the information content in the data. A complicated scene requires a more complicated model than a simple scene to achieve the same error. In this thesis, we present a reconstruction method based on model evaluation that enables us to choose, from a range of candidates,



(a) medical imaging (b) topographic mapping

(c) reverse engineering (d) police investigation

Figure 1.1. Applications of 3D data.[1]

the model that best balances accuracy versus abstraction.

To measure the accuracy of a model, we propose to use *generalization error*. Rather than simply measuring deviation, generalization error assesses the prediction capability of the model on an independent data set. In a static data set such as a collection of range samples, we simulate independence by dividing the samples into training and test sets. Modeling is perfomed on the training set, while the test set is used only for validation. Figure 1.2 shows the behavior of the error for both the training and test sets as the model complexity is varied [19]. An increase in complexity lowers the training error, as the model gets closer and closer to interpolating the samples. However, because we are dealing with raw data, there is noise present. If we approximate the data too closely, we risk introducing the noise itself into the model, a condition known as *overfitting*. In con-

[1]Images courtesy of (a) Patrick J. Lynch (b) Blom ASA (c) Creative Dezign Concepts, Inc. (d) Spar Point Research LLC.

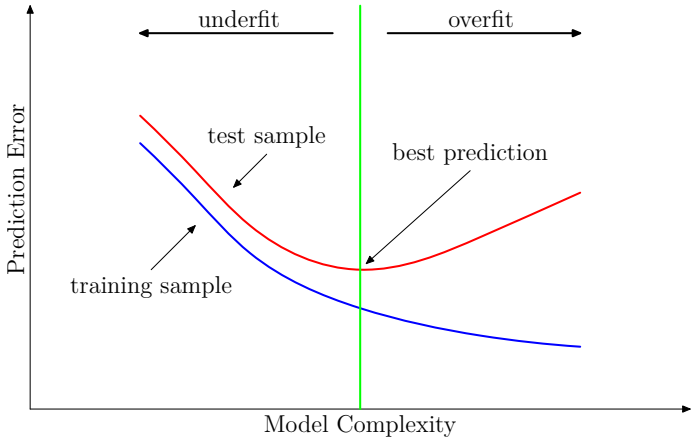


Figure 1.2. To compute generalization error, input data is split into training and test samples. Model fitting is performed on the training sample, while the test sample is used for validation only. As complexity increases, training error decreases, even when the model is overfit. Test error, however, predicts the behavior of the model on independent data.

trast, the test error descends to a minimum value at some level of complexity, after which the error begins to increase. Since the noise model relates to a single instance of data drawn from the underlying distribution, it predicts poorly when new data is drawn from the same underlying distribution. Thus, the minimum test error identifies the optimal model with respect to error.

Generalization error can only be computed however, if the model is parametric. When comparing different reconstruction methods, it is not always the case that the model can be refit at different levels of complexity. In addition, we may have available only the resulting model. If we do not have access to the fitting method, we cannot analyze the generalization error. In such cases, we propose an entropy measure based on data compression. The bit rate relates the compressed and uncompressed sizes of a data set. A low bit rate indicates a large amount of uniformity in the data, which can be compressed effectively by entropy coding. The purpose of the model is to replace large areas of variation by a small number of parameters, such that the residuals exhibit a high degree of

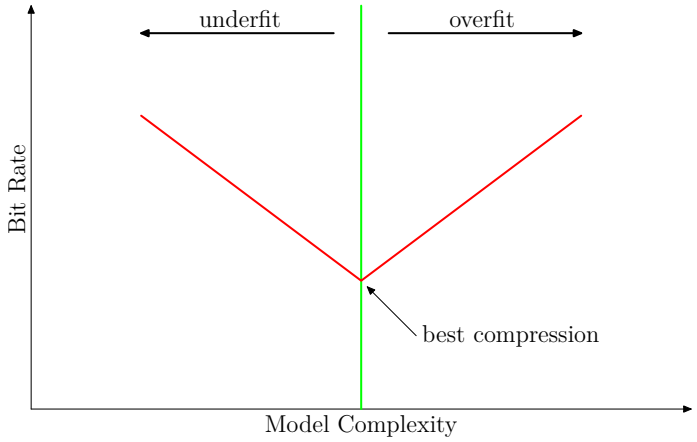


Figure 1.3. Bit rate is the ratio between the uncompressed and compressed sizes of the dataset. A model can increase the amount of compression achieved, as long as it is not too large itself. Since the optimal amount of compression possible is related to the entropy in the dataset, bit rate analysis can be used to measure entropy.

uniformity. Hence, a good model should improve the bit rate.

Figure 1.3 shows the relationship between model complexity and bit rate. Like test error, the bit rate initially improves as the model complexity increases, but at a certain point, a minimum is achieved. Beyond the minimum, an increase in complexity produces no gain in compression performance, so we can conclude that the extra parameters do not capture any additional geometry. The minimum bit rate thus marks the optimal model with respect to efficiency. When generalization error is not available, our entropy measure may be used as a proxy to detect overfitting. In addition, when both generalization error and bit rate are present, we may choose to eschew optimal accuracy if the bit rate indicates that the cost of achieving it is unreasonably high. By combining both measures, our method provides level-of-detail control by means of a small number of parameters.

To investigate the utility of our evaluation-based approach, we perform surface reconstruction on range data of urban scenes. The geometry of urban scenes is of a particular type. Most buildings are composed of many individual planes and smooth curved sur-

faces. Each one can be modeled parametrically on its own, so our goal for the final model is a union of parametric pieces. While we have some idea of what types of models to use, the number and location of the pieces is not known. Therefore, in order to perform surface modeling, we must also solve a segmentation problem simultaneously. Most reconstruction methods in computer graphics concentrate on the case of a single object of unknown type, such as a statuette or a bodily organ. Such objects, while arbitrarily complex, are usually smooth, closed volumes. In our case, however, discontinuities are the most important elements of the geometry. The piecewise problem magnifies the importance of evaluation, because model selection is critical for both segmentation and modeling. In this thesis, we present a new method for obtaining piecewise models that is motivated by our evaluation method. Our method solves a coupled segmentation and modeling problem to produce a union of parametric surfaces.

1.1 Problem Statement

The device used to create the range data shown in this study is a time-of-flight laser scanner, the Leica HDS 2500. Figure 1.4 shows the scanner in action, along with some of the relevant specifications. With a maximum range of 100 meters and a stated accuracy of six millimeters, the HDS 2500 produces very accurate point samples. It also produces a lot of them, up to one million per scan. A data set may contain dozens of scans for a complex scene with a large number of occlusions, so the size of the dataset is of primary importance. Our method must be fast enough to handle the load.

An example of a range image generated by the HDS 2500 appears in Figure 1.5. The range image on the left shows a false color rendering of the intensity response of the laser at each pixel. On the right, a detail of the 3D positional data has been meshed with triangles by connecting adjacent rows and columns in the image grid. Thus, a range image is a

Figure 1.4. Leica HDS 2500 time-of-flight laser range scanner. Maximum image dimensions: 1000 × 1000. Maximum range: 100m. Maximum field of view: 40°. Positional accuracy: ±6mm (1.5 to 50m). Distance accuracy: ±4mm. Angular accuracy: ±12″. Beam spot size: 6mm (0 to 50m).

(a) range image (b) triangulation

Figure 1.5. A range image of Grand Central Terminal, New York. (a) The intensity of the response of the laser at each pixel is rendered in false color. Black denotes no response. (b) A triangulation of the input image based on raster ordering of the scan points. The positional information is very accurate, but the geometry contains holes, occlusions, and depth discontinuities.

combination of geometric and topological information. We aim to preserve both. We assume that the range images are already in registration, since it can be accomplished easily offline [6, 4, 29]. From the registered set of images, our goal is to create a reconstruction algorithm subject to the following requirements:

1. **Fast.** Our algorithm must be able to process large data sets containing dozens of individual range scans, each containing up to one million points per scan.
2. **Global.** Our algorithm must guarantee the quality of the output model by enforcing error tolerances globally.
3. **Parametric.** Our algorithm must produce parameteric surfaces that can be evaluated by generalization error.
4. **Topological.** Our algorithm must produce manifold surfaces that are consistent with physical objects.

Figure 1.6 shows an example of the desired output. The piecewise output model consists of a union parametric surfaces, colored randomly. Red points are classified as nonplanar; they do not belong to any component. Each surface contains a boundary, and may contain one or more holes. Components and boundaries generally correspond to architectural features, an advantage for many potential uses of the range data in computer graphics and computer-aided design.

1.2 Related Work

Generalization error is a standard method for managing noisy data where overfitting is a danger [19]. While prevalent in many fields, it has seen little use in surface reconstruction for computer graphics. A decade ago, Englert [13] used generalization error for reconstruction of buildings, but it was not used in a surface fitting context. Rather, it was used for pattern matching after surfaces had already been generated. A more common

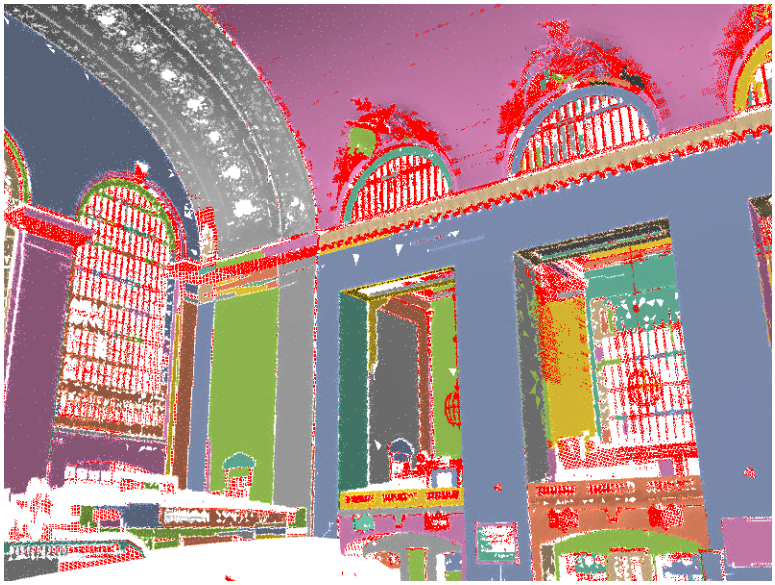


Figure 1.6. A model of Grand Central Terminal generated by our algorithm. Individual surfaces are colored randomly, with red denoting nonplanar points. Both flat and curved surfaces are detected and modeled to a high degree of accuracy, including complex topological features such as holes.

application of generalization error in computer graphics has been to 3D model databases. Hou *et al*. [21] employ support vector machines to classify models for shape-based search and retrieval. Ng *et al*. [26] developed a localized generalization error that they also apply to image classification, but using neural networks instead of support vector machines. Generalization error in the form of the bootstrap method has been used for polynomial fitting by Ramli and Ivrissimtzis [28] in a similar spirit to our approach.

Entropy measurements were used for segmentation 20 years ago by Darrell *et al*. [9]. Their minimum description length encoding was used to evaluate candidate models during segmentation. Schürmann and Grassberger [31] use compression algorithms to measure the entropy of several different kinds of symbol sequences. Grossberg *et al*. [18] also measure entropy in high-dimensional satellite data by means of compression. Since the entropy determines the maximum possible lossless compression, a good estimate can be

used to identify optimal models for a variety of different kinds of data sets.

When approaching the segmentation problem, we seek a method that allows us to use our evaluation method, so we are primarily interested in segmentation that produce parametric models as output. Early range segmentation methods did not do so. For instance, the region growing algorithm [3, 5, 20] operates on a graph connecting range image pixels in small neighborhoods. The output of the algorithm is a graph subset, which can only be evaluated by entropy measures. In addition, because modeling criteria are only applied locally, the resulting graph subsets may or may not conform to the desired shape globally. Nevertheless, region growing is commonly applied to range data because it is very fast.

Improving the quality of the output surfaces requires global information. For instance, Gotardo *et al*. [15] augment region growing by applying robust statistics during model selection. Another region growing variant by Bab-Hadiashar and Gheissari [1] uses energy minimization. An extreme example of global information for graph-based segmentation is the normalized cut algorithm of Shi and Malik [34]. Normalized cut was applied to range data by Yu *et al*. [43], but the data set had to be reduced in size by many orders of magnitude to allow the large matrix optimization to be solved. Aside from feasibility issues, normalized cut also does not output model parameters. In addition, the graph subsets produced do not satisfy a global geometric constraint, even if they are optimal in some sense.

Moving to parametric models means that evaluation of the candidate components can be performed on the fly, so only conforming surfaces are allowed into the output. The Hough transform computes model parameters explicitly. First used for line and circle detection in 2D [12], it can be modified easily to detect parametric shapes in 3D [39]. Thus, output components are guaranteed to conform to the correct model type, but their parameters are not determined by global geometric optimization. Instead, the parameters

are estimated in a quantized voxel space which is not meaningul geometrically. In addition, since graph connectivity is not used, the output models produced are not connected components, so the Hough transform is not topology-aware. High-resolution voxelization requires large amounts of memory, which also limits the scalability of the method in practice.

We developed a segmentation method that combines aspects of the region growing, normalized cut, and Hough transform methods [42]. Our method utilizes global k-means iteration is a similar way to the mesh approximation method of Cohen-Steiner *et al*. [8]. We produce parametric models by means of global geometric optimization, and since our method is graph-based, it also preserves topology. The only problem is that the method performs an exhaustive search for candidate models that severely limits is usefulness for range data. In this thesis, we describe how to speed up our segmentation using the RANSAC model selection method of Schnabel *et al*. [30].

We perform segmentation to isolate subsets of the data that conform to a single parametric model, but reconstruction methods have also been applied to range data without segmentation. Multiple range scans are usually combined prior to reconstruction, to take advantage of oversampling in overlap areas. After high-resolution reconstruction, the resulting mesh may be reduced in size by mesh simplification. For smooth objects, the abundance of data can lead to high accuracy after smoothing, but corners and depth discontinuities are not likely to be well-represented. The marching cubes algorithm of Lorensen and Cline [24] extracts an isosurface from a voxel space. Like the Hough transform, voxelization requires the same resolution to be used for both smooth and discontinuous areas. Thus, detail in discontinuous areas requires high resolution everywhere.

The computational impact of the "marching" approach can be mitigated to some degree by avoiding the construction of the voxel space. The ball pivoting algorithm (BPA) by Bernardini *et al*. [2] is a $2\frac{1}{2}$D variant in which the marching is performed by rolling a

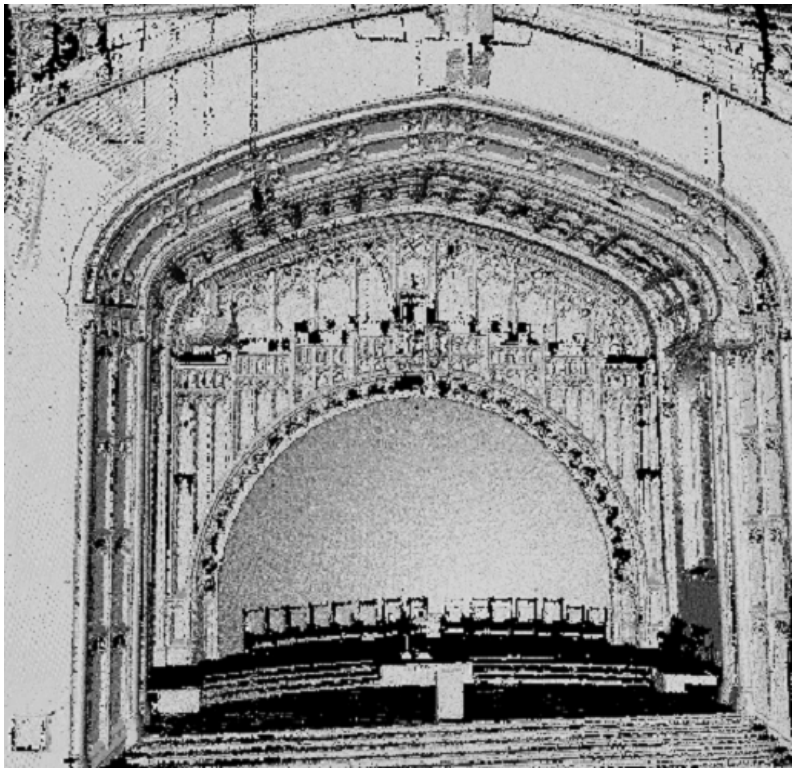


Figure 1.7. Out-of-core ball pivoting algorithm.[1]

ball over the surface and creating a triangle wherever the ball touches three points simultaneously. BPA is better-suited for urban scenes than marching cubes because it is able to reconstruct topological disks while marching cubes is designed for topological spheres. An example of a BPA mesh is shown in Figure 1.7. The BPA mesh captures the abundance of detail in the scene, but it is not much smaller than the input data. Following reconstruction by mesh simplification (MS) provides level-of-detail control. The combination of BPA and MS exemplifies the unorganized approach of performing reconstruction on the complete high-resolution data set.

In contrast, we perform segmentation on a scan-by-scan basis, and merge the results afterward. Turk and Levoy [38] designed the mesh zippering algorithm to merge overlapping surfaces. Since they start with meshes, however, the zippering process leaves

[1]Image courtesy of Ioannis Stamos.

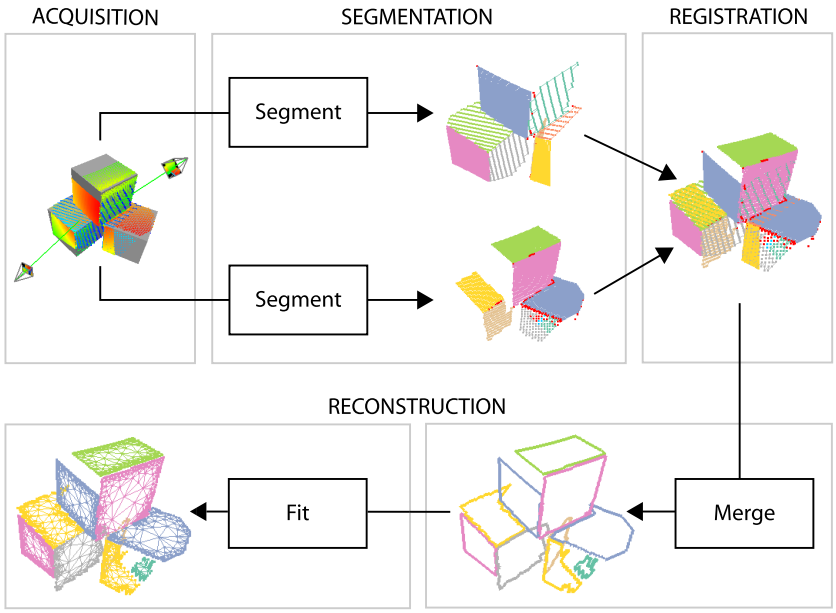


Figure 1.8. Our surface reconstruction algorithm begins by running a planar segmentation algorithm on each scan separately. Then, the registered scans are merged, with overlapping components combined in a common parameter space. Finally, we fit a parametric surface to each group of overlapping components.

visible seams. With parametric models, however, we can combine one or more models in a common parameter space and re-estimate the parameters. Stamos *et al*. [37] merged components in a common Euclidean parameter space. We have generalized this approach to nonplanar shapes that are homeomorphic to Euclidean space.

1.3 Method

An overview of our reconstruction method appears in Figure 1.8. After several overlapping range images have been acquired, each one undergoes planar segmentation. The segmentation produces a set of regions with boundaries that are simple closed curves in the plane. The scans then need to be registered to each other before they can be merged. We

registered our range scans using the point-to-plane algorithm of Chen and Medioni [6]. Registration does not depend on the segmentation, and may be performed in parallel, although an automated algorithm that uses the segmentation results may be substituted instead [35]. During the merging step, overlapping components are combined in a common parameter space, and the Boolean union of the individual boundaries becomes the boundary of the merged component. Finally, we fit a surface to each merged component. The output piecewise model is the disjoint union of all of the merged components.

In this thesis, we describe our reconstruction method in detail. In Chapter 2, we present our novel evaluation method, and explain how to use it for model selection and assessment. As for the algorithm itself, we begin in Chapter 3 by describing the input in detail, including data structures derived from the raw data that capture crucial surface properties that we exploit in later stages of processing. In Chapter 4, we examine several existing methods of segmentation and discuss their advantages and disadvantages with respect to the requirements stated in Section 1.1. By combining elements of several methods, we then tailor a segmentation method for our purposes. Finally, in Chapter 5, we describe how to merge components from overlapping scans, and how to fit surface models to the merged components.

1.4 Contribution

The principal contributions of this thesis are:

- A method for evaluating the goodness-of-fit of a 3D surface model based on a combination of generalization error as a measure of accuracy and bit rate as a measure of entropy.

- A reconstruction algorithm based on our evaluation method that computes a piece-

wise surface model from a set of registered range scans.

- The piecewise surface model itself, a high-level representation of the geometry of a scene that is suitable for analytic computation.

- A discretization algorithm for converting the piecewise surface model to a triangle mesh at a user-specified level-of-detail.

The reconstruction algorithm provides a general framework for segmentation and modeling problems. It can be tailored to other applications by replacing the shape functions with alternative formulations. The evaluation method is also general, in that it can be used to characterize the entropy of a model *a posteriori*, when access to the reconstruction algorithm is unavailable.

Chapter 2

Model Evaluation

The end product of the reconstruction process is a piecewise model, in which different components may be represented by different families of approximating functions. How do we select the best model to use for any individual component? This question is closely related to the more general problem: How do we evaluate geometry models? In this chapter, we present a method of model evaluation based on a combination of generalization error and entropy. As we saw in the introduction, generalization error measures the prediction performance of a model on independent data. For example, if a scene is inadequately sampled in a certain region, the model is likely to be underfit. A new scan that is focused on the same region will exhibit large error with respect to this model, because the scene geometry is more complicated than the model geometry. In contrast, a very high sampling rate can lead to large generalization errors if the model is so fine that it captures positional variations that are entirely due to noise.

To compute the generalization error, we employ a standard technique: 10-fold cross-validation. Cross-validation splits the data into a training set and a test set. In the 10-fold case, the test set comprises 10% of the total size of the data. The test set is reserved only for validation. First, we fit our model to the training set, then we compute the error

against the test set. Since the test set was not used to create the model, this procedure simulates the appearance of new data. Generally, training error tends to underestimate the true error, while test error reflects prediction performance more accurately. More importantly, the training error consistently decreases with model complexity, eventually vanishing as the model approaches interpolation of the input data. On the other hand, the test error reaches a minimum at some point, and starts to rise as the model becomes more overfit. The inflection point at the minimum test error marks the optimal model complexity. Let $X = \{x_1, \ldots, x_n\}$ be a set of input points, and let $f(X)$ be a model estimated from X. For some target variable Y, we define a loss function $L(Y, f(X))$ for measuring the error between $f(X)$ and Y. A natural choice is squared error $\big(Y - f(X)\big)^2$. Since our goal is to minimize the error, the target vector in our case is the zero vector $\mathbf{0}$. Test error is the expected value of the loss function

$$\text{err}(f) = \text{E}[L(\mathbf{0}, f(X))]. \tag{2.1}$$

Using the test error, we derive the following heuristic model selection algorithm:

Algorithm 1 Model Selection

Input: A piecewise linear model $\Psi = \cup_i S_i$, a sequence of approximations $\mathscr{F} = \{f_j\}$, and an error threshold ϵ.

Output: The parameters of the best-fitting model.

1. **for** each S_i **do**
2. **for** each f_j **do**
3. Perform cross-validation using f_j and measure the test error.
4. If $\text{err}(f_i) > \text{err}(f_{i-1})$, then f_i is overfit. Return f_{i-1}.
5. If $\text{err}(f_i) < \epsilon$, then we are within acceptable tolerances. Return f_i.
6. If no f_i has been selected, return f_m.

We now have a means to determine how closely our model approximates the data. We

are also interested in controlling the size of the output model. Our approach is to use compression to measure the entropy of a model. In the same way, we can use compression ratio or its inverse—bit rate—to measure the level of abstraction. Rewriting the 3D

Algorithm 2 Model Evaluation

Input: A piecewise planar model $\Psi = \cup_{i=1}^{m} S_i$.
Output: The bit rate of the model.

1. **Project** the points into their respective coordinate frames.
2. **Quantize** the local parameter coordinates and the residual height.
3. **Write** each model to a flat file, along with the parameter coordinates and the residual for each point.
4. **Compress** the flat file using bzip2 and compute the bit rate.

coordinates of each point as 2D local coordinates plus a residual height tends to reduce the variance in z. Lower entropy should result in a lower bit rate. If there are enough points in the given component, space savings may exceed to storage costs of the model, and then the model is considered to be effective at providing some level of abstraction. Even if the test error can be improved further, there is no point to doing so unless real work is being done. Balancing the test error against the bit rate enables us to negotiate a level of complexity that approaches optimal for any given family of models.

In cases where generalization error is unavailable, the bit rate may be used as a proxy for generalization error since the minimum bit rate approximates the optimal model complexity. We use this proxy error measure to compare our results to the ball-pivoting algorithm (BPA), which is representative of methods that operate on unorganized points and utilize no segmentation. Since the unorganized point cloud is highly redundant, any such reconstruction is expected to be overfit. Therefore, we apply mesh simplification (MS) to control the level of detail in the output model. In Section 2.3, we will show that BPA reconstruction does indeed tend to overfit. Furthermore, we show that, although

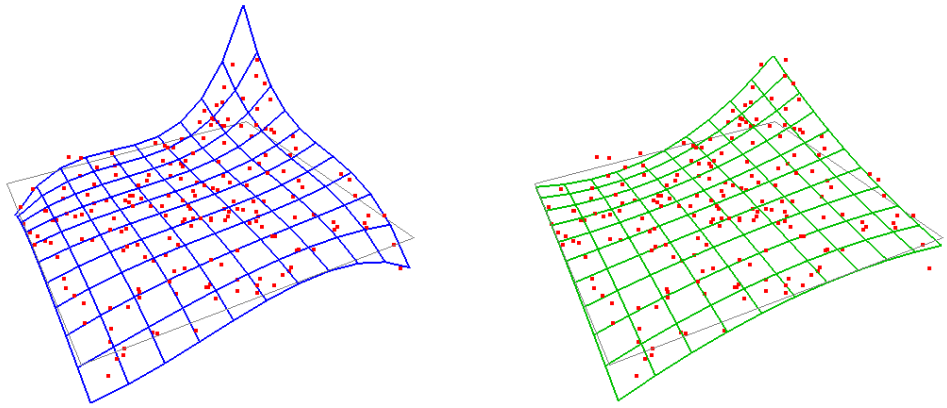
MS can remove much of the redundancy, there is a limit to how much.

The BPA algorithm represents the geometry of the scene using triangle meshes. Since the model is not parametric, we cannot compute generalization error. We need to choose a parametric model that is appropriate for the scene content. Urban scenes contain many planes and higher-order polynomial surfaces, so they constitute the front line of our attack. However, not all curved surfaces in architecture conform to polynomials. For freeform surfaces, we employ spline meshes, which are able to adapt locally to the underlying geometry. We tested the behavior of our evaluation measures on synthetic data in order to verify both that the generalization error can be used to optimize model complexity, and that the bit rate can act as a proxy. The results are presented in Sections 2.1 and 2.2, respectively.

2.1 Polynomial Surfaces

We generated 20 random cubic surfaces and, for each surface, we generated a random sample of points. Then, we perturbed each point's position by applying Gaussian noise to the height. The noise is necessary to induce overfitting at some level of complexity. An example of a test surface is shown in Figure 2.1a. Starting with a plane, we fit polynomials of increasing degree and measure the test error. Figure 2.1b shows the quadratic case. Since the true surface is cubic, we expect to observe a minimum value for the test error at degree 3. Indeed, Figure 2.2a shows the expected minimum for a data set containing 500 points. Note that to the right of the minimum test error, the ground truth error splits the training and test samples.

Now let us compute the bit rate. We write the model parameters and the residual heights to a flat file in the form of scaled integers, and then compress the file using bzip2.



(a) test surface (b) quadratic approximation

Figure 2.1. Synthetic generalization error experiment. (a) Generate a random cubic surface and a set of points near the surface, perturbed by Gaussian noise. (b) Fit polynomials to the point set in increasing order of degree, and measure the generalization error using cross-validation.

The bit rate rate is a ratio between the compressed and uncompressed sizes:

$$\text{Bit Rate} = \frac{\text{Compressed Size}}{\text{Uncompressed Size}}. \tag{2.2}$$

The uncompressed data set contains connectivity in addition to positional information. In the synthetic case, we supply the connectivity by computing a 2D Delaunay triangulation of the sample points, and writing three vertex indices to the flat file for each face of the triangulation.

The bit rate for the example with 500 points is shown in Figure 2.2b. This curve shows a nearly direct relationship between the model complexity and the bit rate. The more complex the model, the more bits it takes to represent it. Evidently, additional complexity yields no gain in space savings. We conclude that, for a sample size of 500 points, we should always use the cheapest model available, the plane. How large does the sample size need to be to gain any benefit from a more complex model? For the cubic

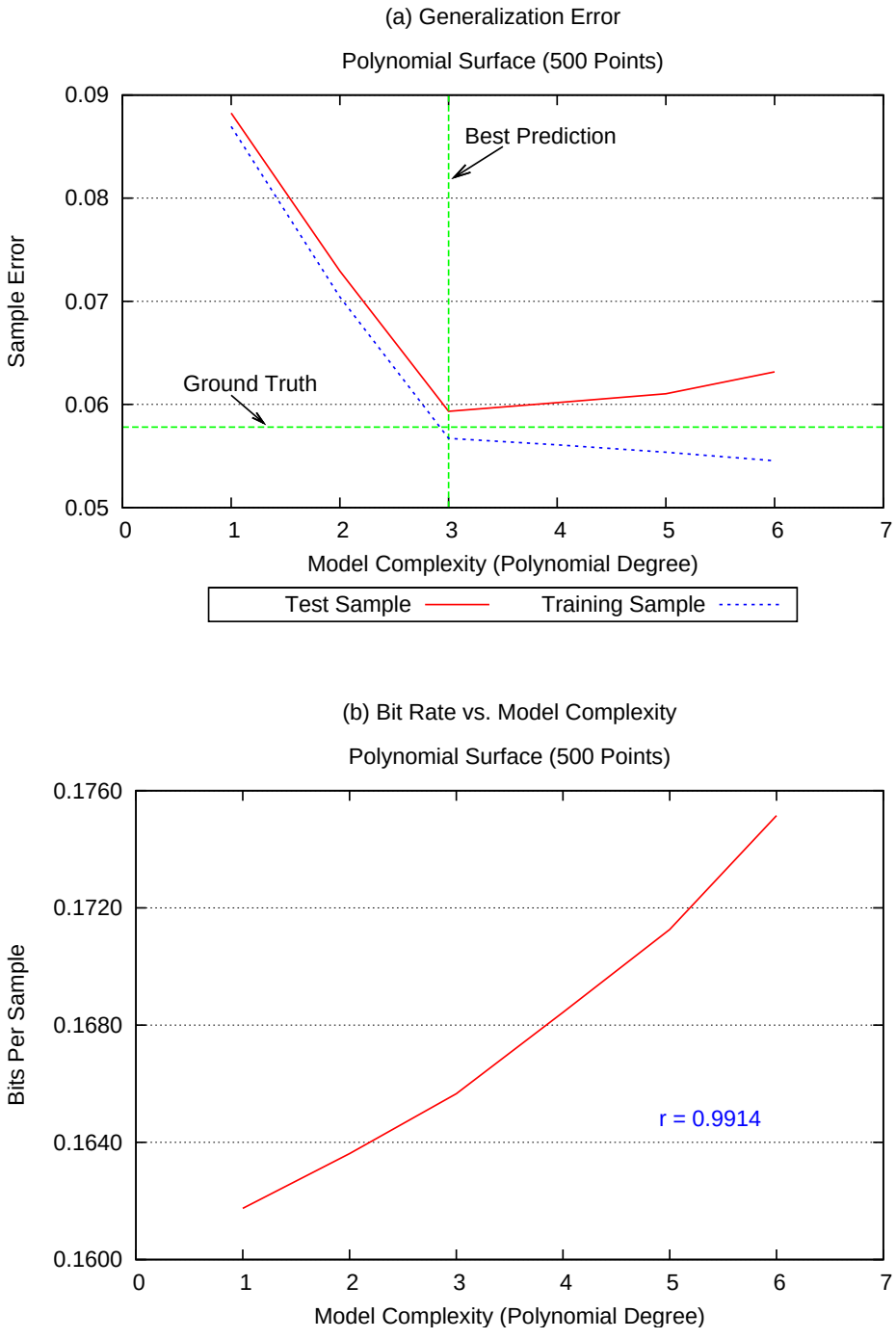


Figure 2.2. Generalization error and bit rate for polynomial surfaces containing 500 points. (a) As expected, the generalization error shows an optimal reconstruction at degree three. (b) The bit rate does not improve, however, because the number of points to be encoded is too small to require a model.

polynomial, consider a sample size of 5,000 points, as shown in Figure 2.3. Although the training error and the test error are very close, they do diverge as expected at degree three, and the test error takes on its minimum value there. But now in Figure 2.3b, the bit rate in also attains a minimum at degree three. Hence, we observe that some abstraction has been achieved: increasing the size of the data leads to increased space savings after compression. Beyond the optimal complexity, however, the bit rate and the model complexity exhibit the same correlation as they did for a sample size of 500 points: as the complexity rises, so does the bit rate.

Let use take a closer look at the relationship between bit rate and model complexity. First of all, if the model is effective, we should observe a relationship between bit rate and test error that is independent of model complexity. This is because both bit rate and test error are measures of entropy. To the extent that the model captures the geometry, it reduces redundancy, thus lowering entropy in the data set. A plot of bit rate versus test error for polynomials of degree less than or equal to three appears in Figure 2.4a. As expected, the bit rate correlates strongly with the test error when the degree is less than or equal to the optimal degree. How strongly correlated depends on the characteristics of the model family. For low-degree polynomials, changes in degree can produce very different surfaces. On the other hand, each level of a spline mesh is a refinement of the previous level, so we expect lower variance in the test error as a result. In general, if the model is underfit, then the bit rate and the test error are related. What if the model is overfit? In Figure 2.4b, we observe a very low correlation between bit rate and test error for polynomials of degree greater than or equal to 3. Since the extra degrees of freedom fit noise, not signal, the test error likewise fluctuates in a noisy way. At this point, changes to the model are ineffective—they do not capture any additional geometry.

In fact, the situation is the same as it was when we used a very small sample size (Figure 2.2). In both cases, the model is larger than it needs to be for the amount of

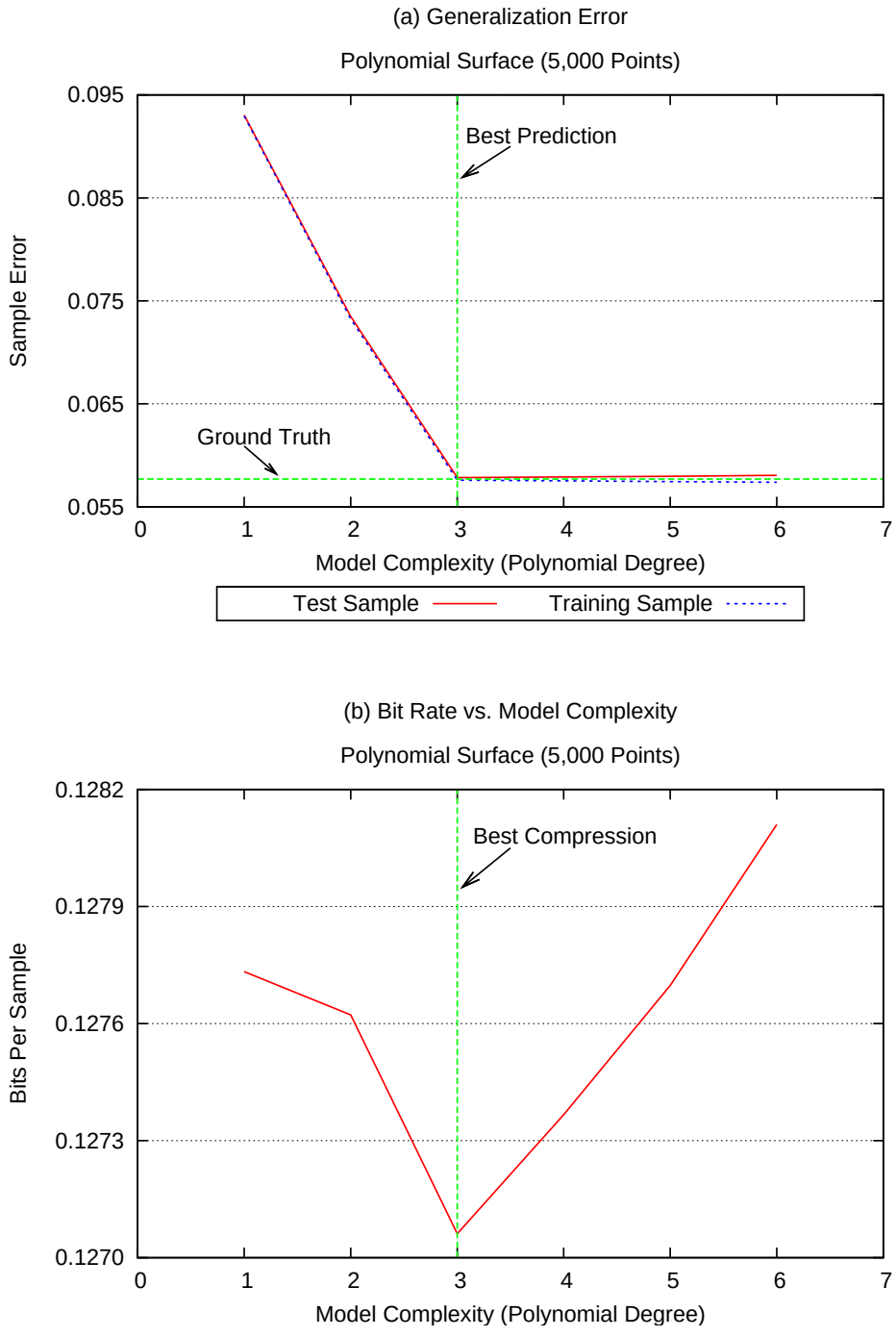


Figure 2.3. Generalization error and bit rate for polynomial surfaces containing 5,000 points. (a) As expected, generalization errors show an optimal reconstruction at degree three. (b) At 5,000 or more points, the cubic polynomial is clearly worth the extra parameters since the bit rate decreases.

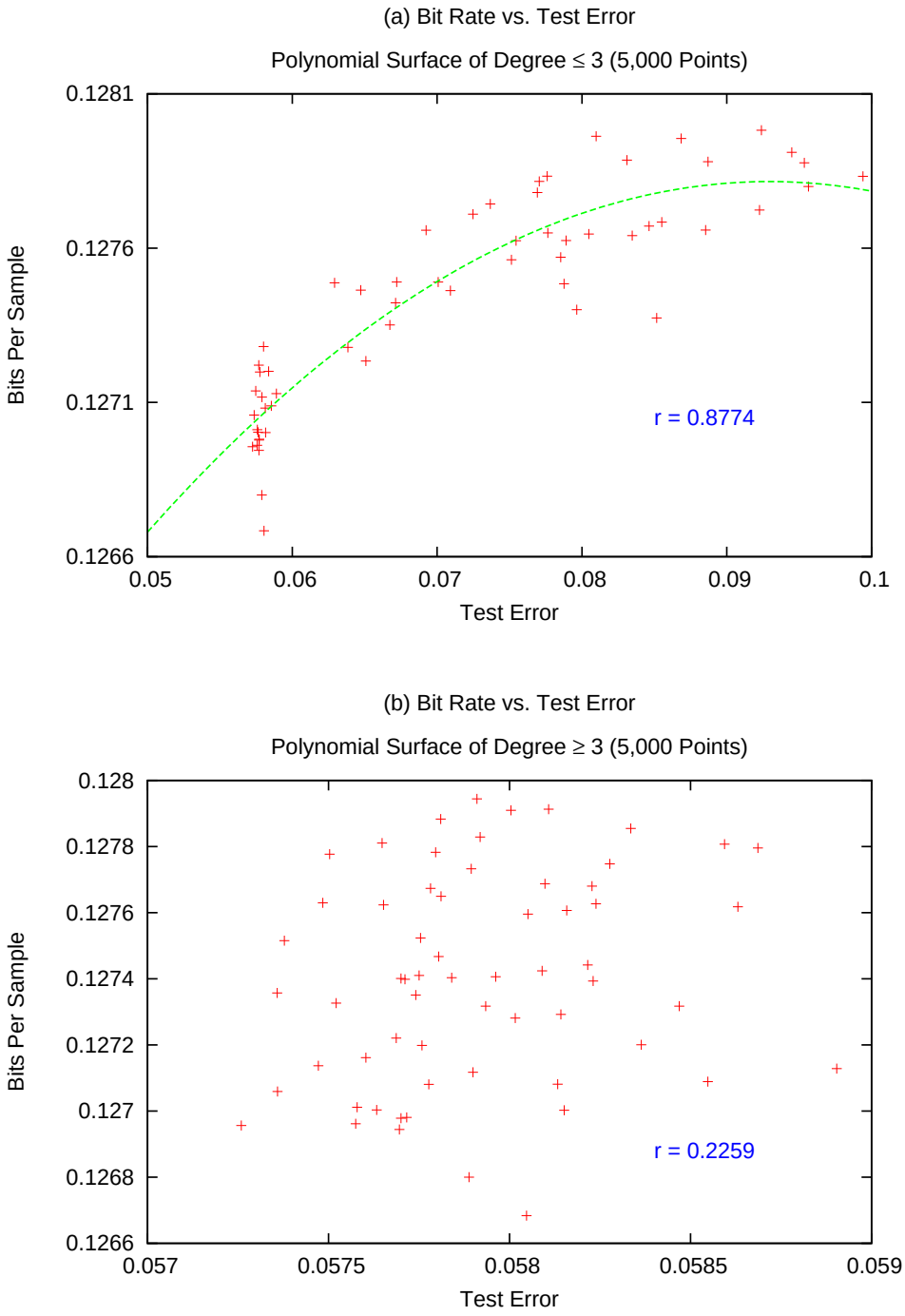


Figure 2.4. Bit rate versus test error for polynomial surfaces. (a) To the left of the minimum test error, there is a high correlation between the bit rate and the model complexity. (b) To the right of the minimum test error, there is no correlation between the bit rate and the model complexity.

information it encodes. To show this, we first note that different model types grow at different rates. For example, an n-degree polynomial requires $(n+1)^2$ parameters of storage, a relatively small rate of growth compared to the spline surface representation, which requires $(2^h+3)^2$ parameters to represent level h of complexity. Therefore, when we are comparing it to the bit rate, we measure the model complexity by the number of parameters required. Figure 2.5a shows the bit rate plotted against the model size for polynomials of degree less than 3. Inverse correlation means a gain in bit rate for every parameter we add to the model. The added expense of storing the new model parameters is more than offset by the space saved after parameterization and compression. For larger sample sizes, the space savings should be even better. Once we exceed the optimal degree, however, the correlation runs strongly in the opposite direction. As Figure 2.5b shows, bit rate is directly related to model size in this case.

Thus, the bit rate behaves in a similar way to the generalization error, in that it reflects the effectiveness of the model—its predictive capability. It is an important measure in model selection because it allows us to address of the cost of the model as a component of the decision process. For example, it may be the case that the level of detail captured by the range scanner exceeds the requirements of the reconstruction. Components often arise that exhibit a minimum test error of one millimeter or less, due to oversampling, or simply due to favorable scanning conditions. This level of accuracy far exceeds the scanner's stated tolerance, and it is certainly overkill for visualization. Bit rate supplies a brake that we can use to determine when the cost of an optimal reconstruction exceeds its value as a modeling abstraction.

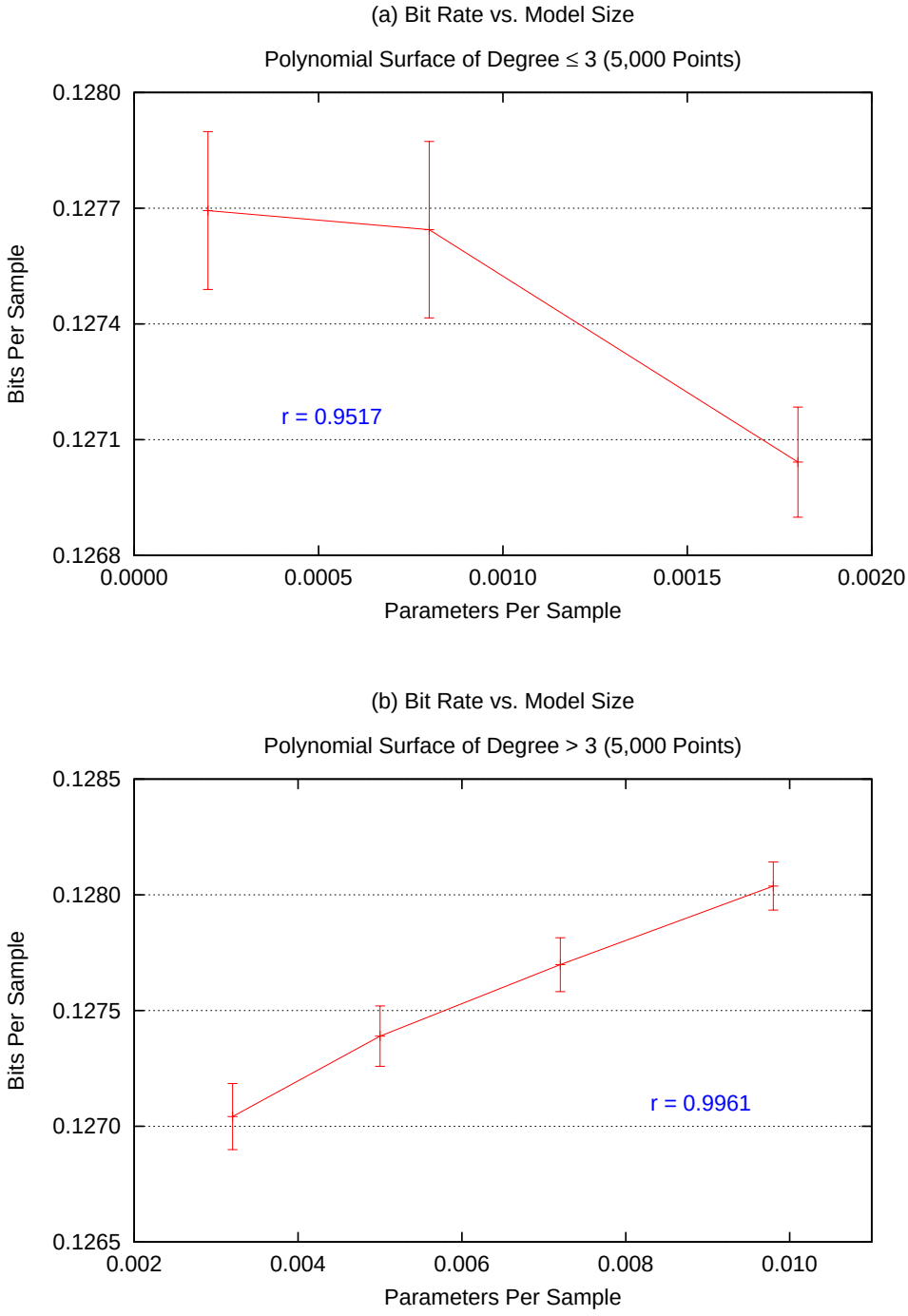


Figure 2.5. Bit rate versus model size for polynomial surfaces. (b) To the left of the minimum test error, the bit rate and the model size are inversely related. (a) To the right of the minimum test error, the bit rate and the model size are directly related.

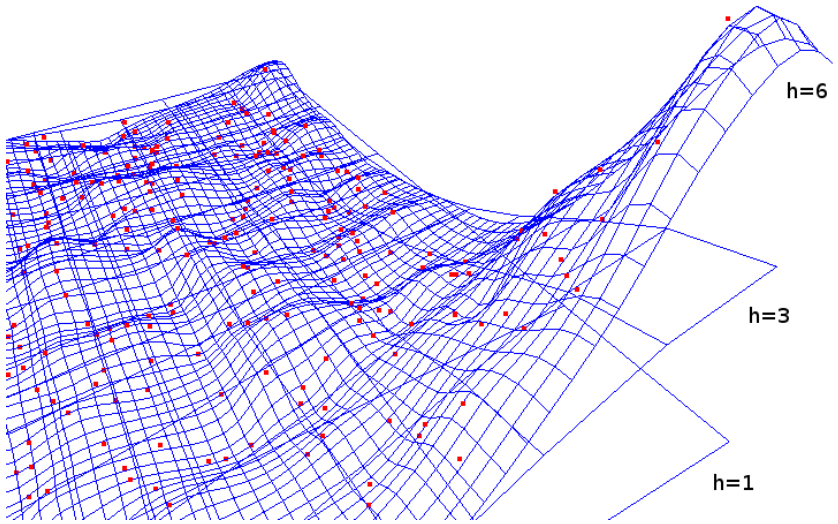


Figure 2.6. Multilevel B-spline mesh. As the spline mesh level h rises, the reconstructed surface approaches the samples. At $h = 6$, the surface is overfit, resulting in ripples that are not featured in the original cubic polynomial.

2.2 Spline Surfaces

Polynomial models grow slowly with degree, so runaway model size is not a concern. Unfortunately, except for planes, they do not fit many components well, especially if the components are large. For freeform surfaces, many possibilities exist. We choose the Multilevel B-spline Algorithm (MBA) [23] for two reasons:

1. The input to the fitting algorithm is a set of scattered data points.
2. The level of detail is controlled by a single integer parameter h.

An example of a hierarchical B-spline mesh is shown in Figure 2.6. At $h = 3$, the reconstructed surface does not model the peak in the near corner very well. On the other hand, at $h = 6$, the surface, which we know to be a cubic function, is clearly overfit. In this section, we will analyze the quality of fit for spline surfaces using the generalization error and the bit rate.

Training and test errors computed over 20 trials of 150,000 points apiece are shown in Figure 2.7a. As expected, the test error reaches a minimum value within the range—at Level 6—while the training error continues to descend, albeit slowly. Now consider Figure 2.7b, bit rate plotted against model complexity. It, too, achieves a minimum value, but at Level 5 it underestimates the true optimal level by 1. What happened? Recall from Section 2.1 that the spline mesh model size grows at a rate of $(2^h+3)^2$ parameters for every level h. At Level 6, the required 4,489 parameters captured a certain amount of noise which survived entropy coding. Evidently, we can achieve optimal prediction accuracy only at the expense of optimal compression, and vice versa. Which choice we prefer is determined by the application. Either way, it is worth noting that the complexity of both polynomial and spline surfaces vary in discrete increments, so some compromise may be necessary. The growth rate of the spline mesh model is also reflected in slope of the bit rate curve in Figure 2.7b. Once overfit, the bit rate suffers exponential degredation. We can straighten out the right tail of the bit rate curve by plotting it against model size instead of model complexity. But first, we will verify the correlation between the bit rate and the test error that we observed for polynomial surfaces.

Figure 2.8a shows a scatter plot for all spline meshes of level five and below. Indeed, we observe high correlation between the bit rate and the test error to the left of the minimum bit rate. Even though the model size is growing exponentially, we still realize improved space savings at the given sample size. At the observed noise levels, we conclude that a component size of 150,000 points can be modeled optimally by multilevel B-splines as long as the level of detail required does not exceed $h = 5$. Overfitting does not actually begin to occur until $h = 6$, after which the bit rate is dominated by the growth of the model. At a given level, however, we can observe low correlation between the bit rate and the test error across different trials, as shown in Figure 2.8b for $h = 6$. In contrast, the bit rate correlates strongly with model size when the model is overfit, as we can see

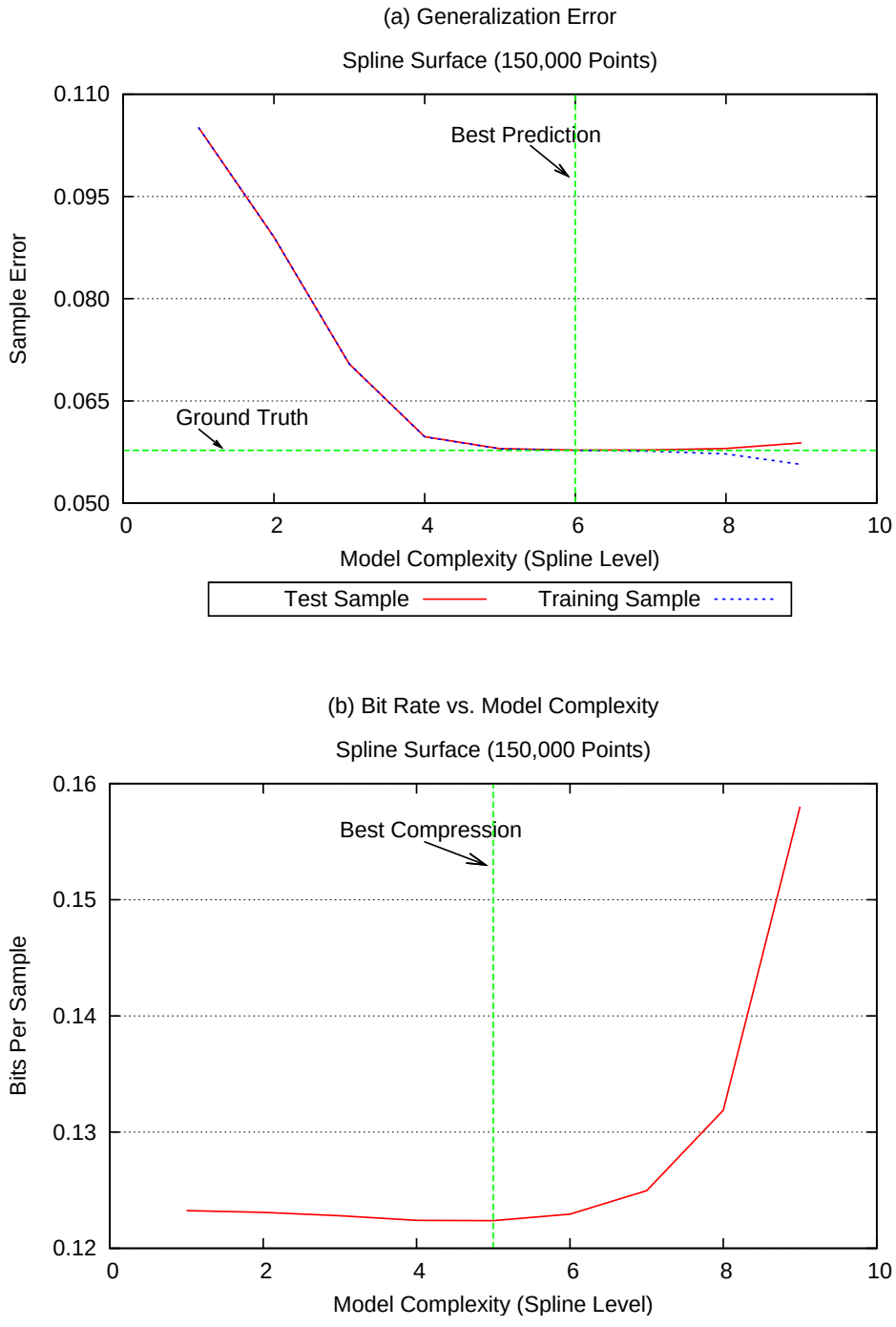


Figure 2.7. Generalization error for spline surfaces. (a) The optimal generalization error comes very close to the ground truth error at spline level six. (b) The bit rate underestimates the optimal spline level by one, but the difference in error between level five and level six in only about 2mm.

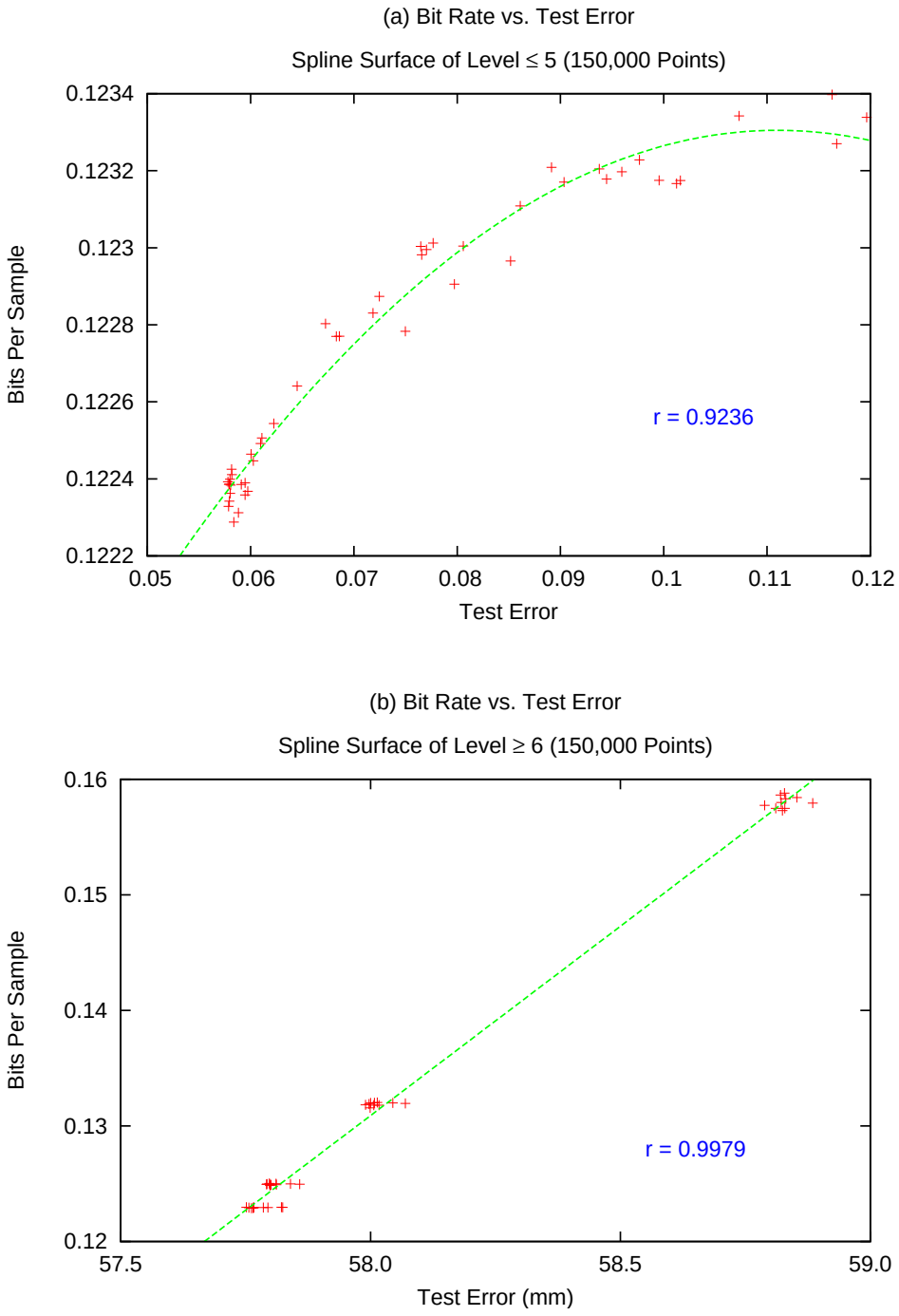


Figure 2.8. Bit rate versus test error for spline surfaces. (a) To the left of the minimum test error, there is a high correlation between the bit rate and the test error. (b) To the right of the minimum test error, the correlation between the bit rate and the test error is dominated by the change in model size. However, the bit rate and test errors are clustered by complexity.

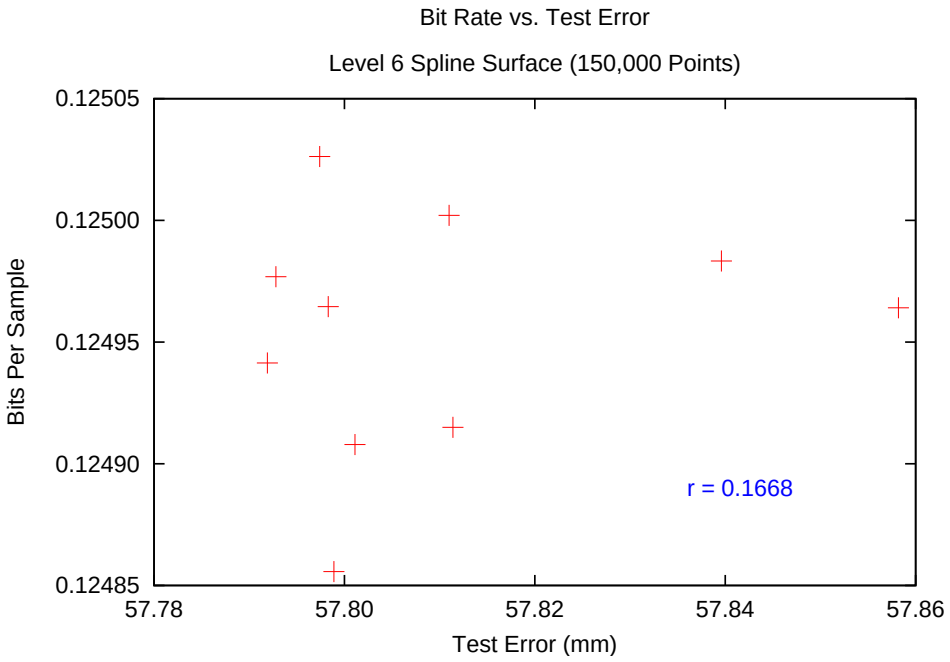


Figure 2.9. Bit rate versus test error for spline surfaces at the same level. To the right of the minimum test error, there is no correlation between the bit rate and the test error from trial to trial if the spline level remains constant.

by the very high correlation coefficient shown in Figure 2.10b. At a certain point, it is clear that we will need more than one parameter per input sample, and that, quickly, the model will exceed the input data in size. Hence, the spline mesh must be very effective indeed at levels below the minimum bit rate in order to achieve the inverse correlation shown in Figure 2.10a. They also prove to be very effective for components that are much larger than 150,000 points. Indeed, one of the main advantages of multilevel techniques in general that they scale very well to large data sets. Using either polynomial or spline surfaces, we were able to model all of the components that we encountered in actual data to within reasonable tolerances. Other data sets may require alternative formulations, which are easy to add. As long as they are parametric, we can compare them during model selection using generalization error.

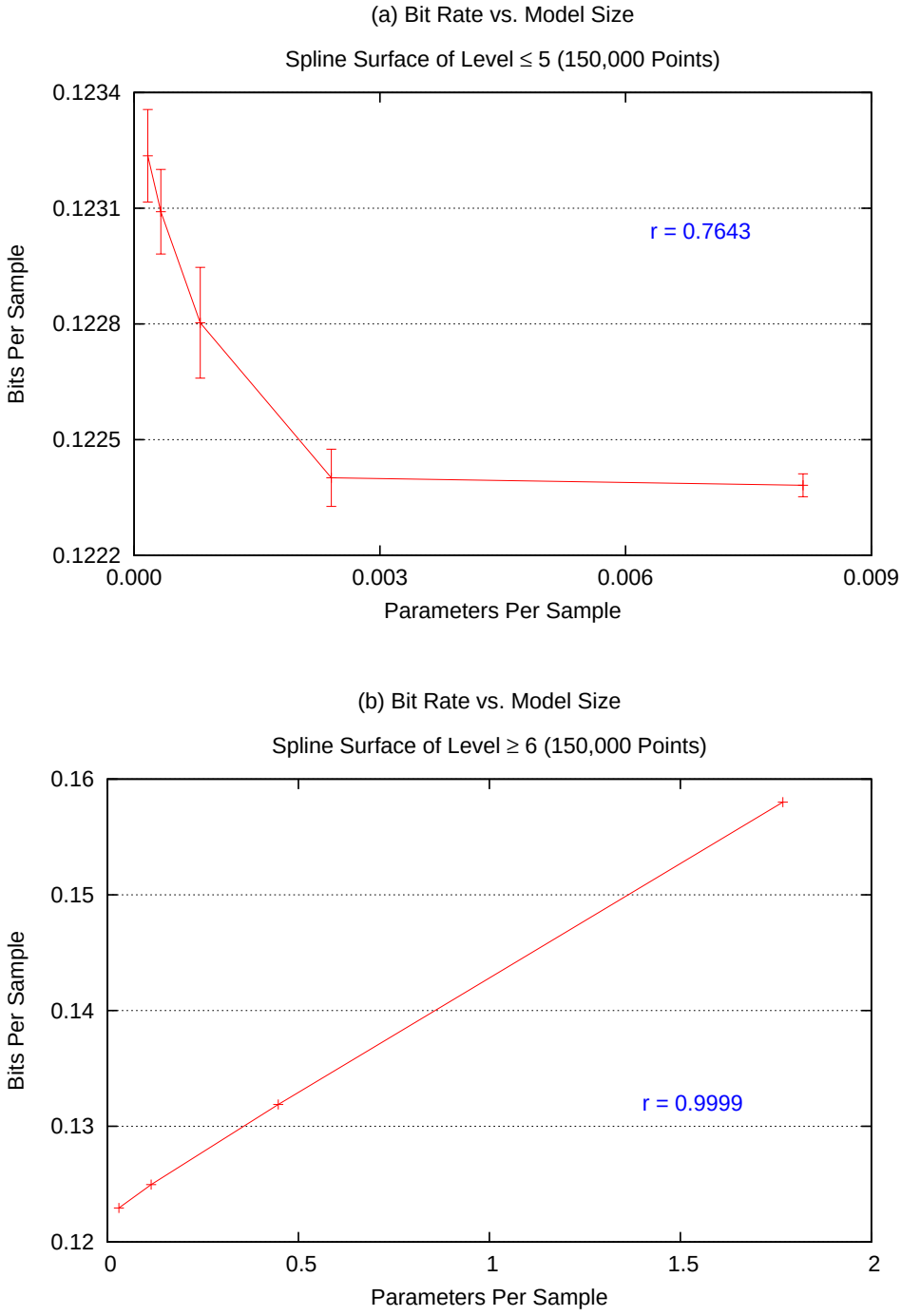
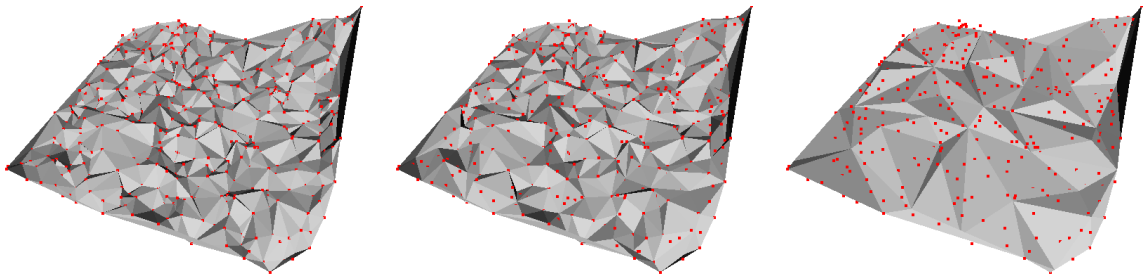


Figure 2.10. Bit rate versus model size for spline surfaces. (a) To the left of the minimum test error, the bit rate and the model size are inversely related. (b) To the right of the minimum test error, the bit rate and the model size are directly related.



(a) Full size (b) 50% (c) 10%

Figure 2.11. Synthetic mesh simplification test. (a) The Delaunay triangulation in parameter space provides a complete triangles to be simplified by the MS algorithm. Two levels of simplification are shown in (b) and (c). The original vertices are projected to the simplified meshes to compute the error.

2.3 Mesh Simplification

When analyzing BPA meshes, we do not have access to cross-validation to determine quality of fit because removing a random selection of points from the data set would change the scale characteristics in the point cloud. Hence, a fixed-radius ball would be likely to produce surfaces that vary topologically from trial to trial. The bit rate serves as an alternative measure of entropy in this case. To assess its usefulness in this role, we varied the model complexity by repeated application of MS, and measured the change in bit rate achieved by parameterizing the complete point set using the simplified model. In the synthetic case, we simulated the full-size BPA mesh by computing the Delaunay triangulation of the sample points, and passing the result on to the MS algorithm as input. For actual data, the vertices and faces of the BPA mesh constitute the complete model, including connectivity information. Note that the set of vertices is a subset of the original points—many points never come into contact with the ball, and are not included in the final output. Our algorithm also processes a subset of the input points, but the two subsets are different. It is difficult to compare reconstructions directly because the

different subsets have different error characteristics. For example, the BPA mesh is biased toward the inside or outside of the surface, depending on which side the ball rolls. We use an absolute distance threshold in our algorithm, so points both above and below a surface are considered equally for membership. Another difficulty in performing a direct comparison is that error cannot be computed in the same way. Since a component in our output comes with a parameter space, we compute the error based on the height of a sample point in the parameter space. But the BPA mesh contains no such space. How can we compute the height at a given point if we cannot determine the direction in which to project it toward the mesh? Without a segmentation, the options are severely limited, because we cannot get a handle on any portion of the data using a single model. In cases such as this, the bit rate can act as a fallback. Though it is expensive to use compression in trial after trial, using the entropy ensures that the level-of-detail of the output representation is optimal.

Figure 2.12 show how sample error and bit rate change with model size. Let us consider the bit rate in Figure 2.12b first. Reading from right to left, we repeatedly remove edges from the model and reparameterize the data set using the newly simplified mesh. As the number of edges decreases, the bit rate also decreases proportionally. As we have seen, this indicates that the entropy of the model remains constant. Only the size of the model changes. As for the sample error, we parameterize the points as follows: To compute the height of a given point over the model, we project the point to the plane and query the Delaunay triangulation for an intersecting triangle. A ray originating at the query point and directed upward intersects the triangle in 3D. The height of the intersection is taken to be the value of the model at that point. The sample error, shown in Figure 2.12a, increases as the number of edges decreases. As the number of edges remaining approaches 10%, the error approaches the ground truth error *from below*, which means that the model is too fine. Recall the graph of the generalization error, reproduced

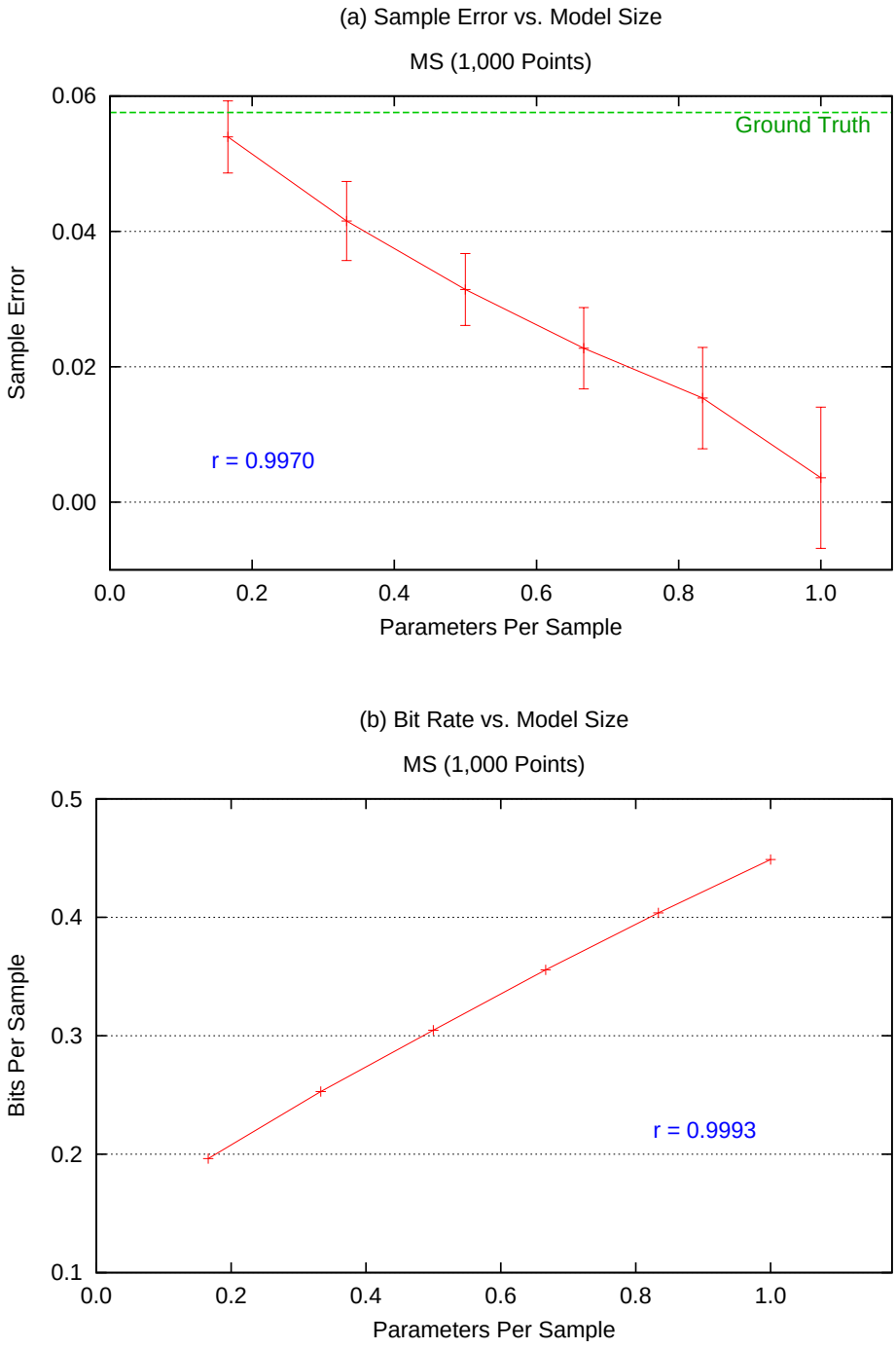


Figure 2.12. Proxy generalization error for MS. (a) An increasing number model parameters reduces the sample error, but there is no way to know if we are overfit. (b) Direct correlation between the bit rate and the model size indicates that the model is ineffective, because increasing the number of parameters does nothing to improve the bit rate.

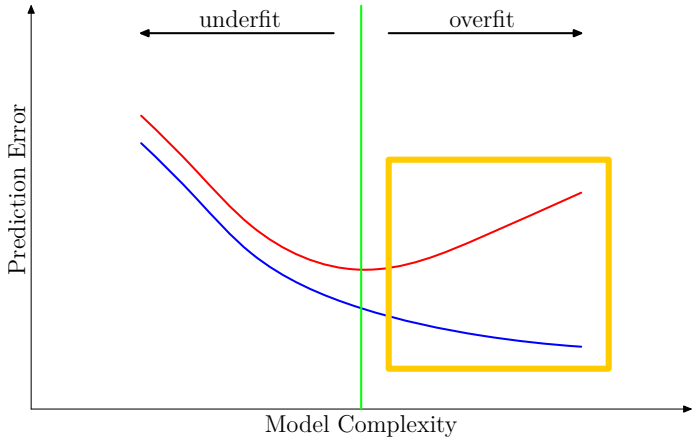


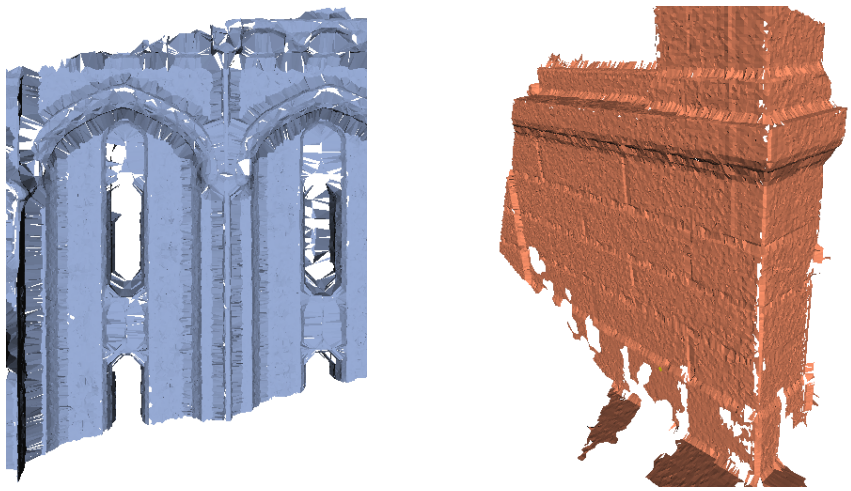
Figure 2.13. To the right of the optimal test error, increasing the model complexity continues to lower the training error, but the test error increases as overfitting causes the prediction capability of the model to suffer.

in Figure 2.13. The two curves slope in opposite directions, placing them in the region to the right of the best prediction. Taken together, they indicate that the model is initially overfit, and remains overfit even after 90% of the edges have been removed.

Two examples of BPA mesh excerpts from results on actual data appear in Figure 2.14. Unlike the synthetic case, we have no parameterization for these meshes, so we must construct one in order to compute the change in bit rate. We do this by computing a normal estimate at each vertex of the full-size mesh using the discrete techniques outlined in Section 3.1. Then, as we remove edges from the model, we project the original vertices to the new model along their normal directions. The results are shown in Figure 2.15 for five data sets of buildings in New York City:

GC – Grand Concourse, Grand Central Terminal (interior);
SH – Shepard Hall, City College of New York (exterior);
GH – Great Hall, City College of New York (interior);
HN – Hunter North, Hunter College (exterior);
CU – Cooper Union (exterior).

Just as we saw in the synthetic case, the bit rate in Figure 2.15b decreases proportionally



(a) Shepard Hall (b) Cooper Union

Figure 2.14. Excerpts from BPA+MS meshes. Both meshes were created from actual scans taken by a Leica HDS 2500. Meshes were generated using an out-of-core BPA implementation by Ioannis Stamos. The MS algorithm is bundled with CGAL.

with the model size as edges are removed. At the same time, the error rises to four or five millimeters, on average, which is commensurate with the manufacturer's stated error tolerance of six millimeters. The two curves again slope in opposite directions, indicating that the data is overfit by the BPA mesh, even after simplification. Furthermore, it is not clear that a path to the optimal prediction error is always possible from the full-size starting point. Note the change in slope at 0.1 parameters per sample for the SH and GH data sets. Shape tolerances can constrain the MS algorithm so that it cannot contract beyond a minimum number of edges. This problem is especially acute if the models are piecewise smooth, as they are in the case of BPA+MS meshes. In the case of both the SH and GH data sets, the minimum is evidently achieved somewhere between 10% and 20%.

Intuitively, it makes sense that the BPA mesh should be overfit. Since it is constructed on the unorganized point cloud, it is liable to be highly oversampled in many areas, a condition that may not be completely eliminated by the smoothing properties of the

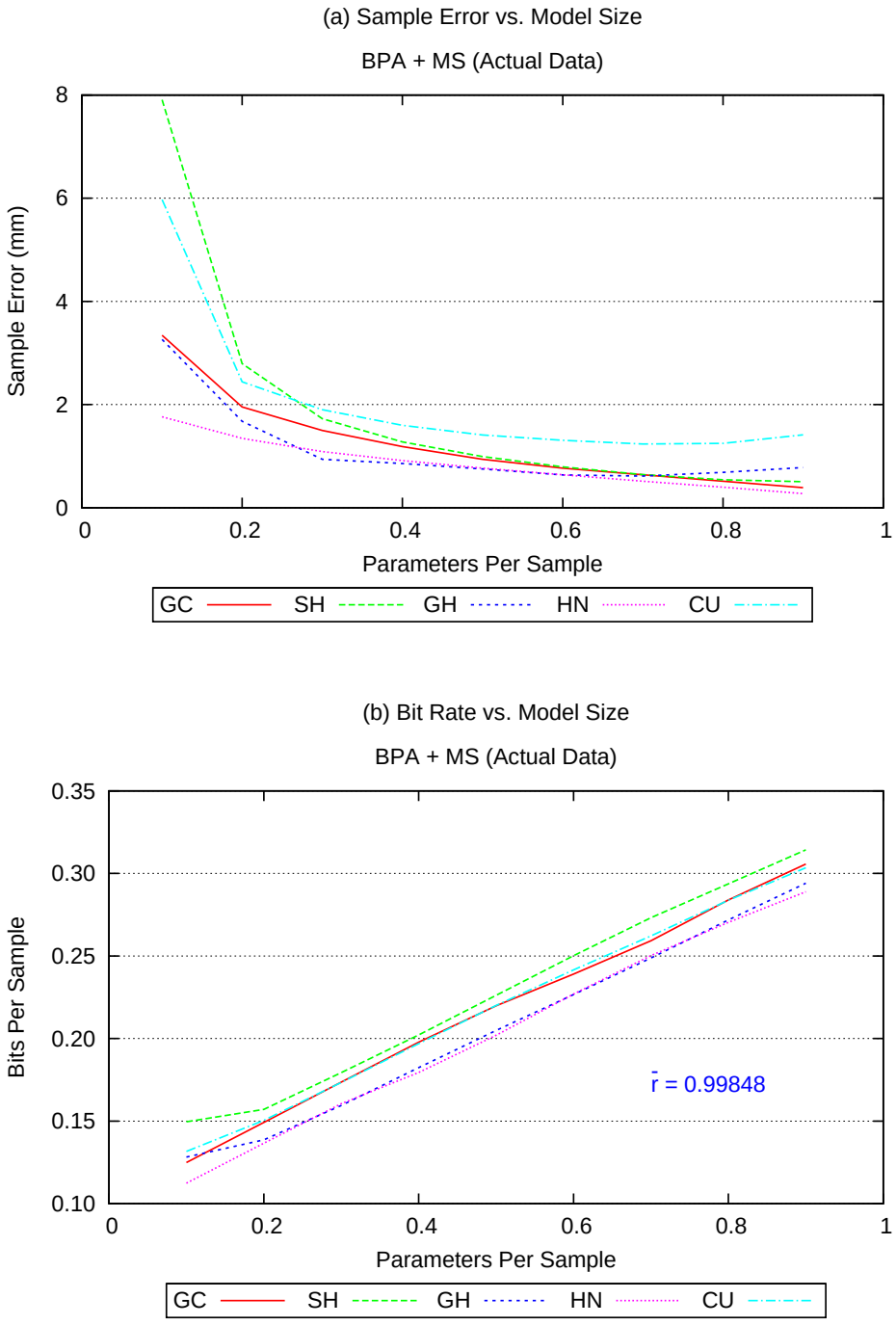


Figure 2.15. Proxy generalization error for BPA+MS results on actual data. (a) Like the synthetic case, the sample error decreases with an increase in model size. (b) Also like the synthetic case, a direct correlation between bit rate and model size indicates that the data is overfit.

algorithm. In particular, any smoothing operation is severely compromised by the presence discontinuites in the mesh. The major advantage of our algorithm is that it models the discontinuities explicitly during the segmentation phase, so smoothing can be applied isotropically during the modelling phase. Our reconstruction method approaches the optimal model complexity from the underfit side, which gives us control over the level of detail. For example, we can elect to cut off the modeling process prematurely in order to produce a simpler model if the application warrants it. In addition, a path to the optimal prediction error is more likely to exist, since a potentially large body of model families can be applied interchangeably.

We have defined what we mean by a good model by showing how to analyze and compare surface reconstructions using generalization error and bit rate. In the remainder of this thesis, we apply model evaluation to the surface reconstruction problem. In Chapter 3, we first describe precisely the input to our algorithm. We define two data structures derived from raw range data that capture local surface properties such as connectivity, orientation, and curvature. Then, in Chapter 4, we show how to segment range data into planes based on a measure of similarity computed using the local surface properties. Finally, once segmentation is complete, we explain how to merge and model the resulting components in Chapter 5.

Chapter 3

Algorithm Preliminaries

Range scanners produce a stream of 3D coordinates, each one representing the location of a point on the surface of an object in the scene. To produce a geometric model, we build data structures atop the input data that can be used to answer queries of a geometric nature that may arise in an application context. For example, a 3D rendering program might need a surface normal in order to compute the color at a given point. At the simplest level, the input coordinates alone with no additional structure can serve as a geometric model. An unorganized collection of coordinates is called a *point cloud*. How well does a point cloud represent the geometry of the scene? Even if the positional information is highly accurate, point samples are not well-suited to the scene content in our case. Urban scenes are composed of many disjoint surfaces such as walls, floors, ceilings, and other architectural features. In general, one surface generates many point samples, but this relationship is not represented in a point cloud. The granularity of this model is too fine.

Ideally, we would like to construct a model that treats each disjoint surface as a separate component. The components are either flat or smoothly curved, so each one can be modeled easily and efficiently using a *Monge patch* [17]. A Monge patch is a surface

$f : U \to \mathbb{R}^3$ specified by a function of two variables

$$f(u,v) = \big(u, v, h(u,v)\big) \tag{3.1}$$

where U is an open set in $\mathbb{R}^2$ and h is differentiable in $\mathbb{R}$. It is a local surface, in that we evaluate the approximating function h in a local parameter space. Our surfaces exist in 3D, so the required parameterization $\Pi : \mathbb{R}^3 \to U$ maps 3D scene points to 2D parameter-space coordinates. The domain of h is further limited by a boundary curve Ω spanning U, which may contain holes. The bounded function is called a *trimmed surface*, which we write as a tuple $S = \langle \Pi, \Omega, f \rangle$. The desired final output representation is thus a union of trimmed surfaces

$$\Psi = \bigcup_{i=1}^{m} S_i \tag{3.2}$$

where m is the number of components in the scene.

The input data contains additional structure that is not utilized by the point cloud. Rather than producing unorganized points, range scanners scan in raster order. The stream of input coordinates is arranged in a grid pattern, which we can use to parameterize the points in a similar manner to the Monge patch parameterization shown above. The mapping $g : U \to \mathbb{R}^3$ is not as simple as that of Equation 3.1, however:

$$g(u,v) = \big(x(u,v), y(u,v), z(u,v)\big). \tag{3.3}$$

Approximated by g, the entire image is represented as a single surface. We can now create a discrete approximation to the surface simply by connecting adjacent rows and columns in the raster grid with triangles. The vertices and edges of the triangulation constitute a graph structure. Combined with the coordinate data stored in the vertices, we may define a range image as an embedding of a graph into $\mathbb{R}^3$.

Figure 3.1. A triangle mesh constructed from a single range image. Note the large number of holes, and the spurious surface areas spanning depth discontinuities.

Figure 3.1 shows an example of a triangle mesh created using the grid parameterization. The shortcomings of the surface are apparent. The surface is noisy, and it contains many holes. A more serious problem is the treatment of depth discontinuities. Adjacent points in the scanner grid are connected by triangles even if they belong to different surfaces. These triangles do not represent actual surfaces in the scene, an ambiguity that must be removed in order to produce a faithful model. On the other hand, this mesh has useful mathematical properties. Let σ be a *k-simplex*, a subset of $\mathbb{R}^3$ that is the convex hull of a finite set of exactly $k+1$ points [10]. Each vertex in the mesh is a 0-simplex, an edge connecting two vertices is a 1-simplex, and a triangle connecting three vertices is a 2-simplex. Altogether, the mesh is a collection $\mathcal{T} = \{\sigma_i\}_{i=1}^{n}$ of n k-simplices. Because the grid parameterization is well-ordered, the collection of simplices exhibits the following property: for $i \neq j$, either $\sigma_i \cap \sigma_j = \emptyset$, or $\sigma_i \cap \sigma_j \in \sigma_i$ and $\sigma_i \cap \sigma_j \in \sigma_j$. Two simplices can intersect only in a single point or a single edge. If the required property holds, then the

collection is called a *simplicial complex*, and $\mathcal{T}$ is said to be a *triangulation* of the input points. This discrete structure comprises both geometric and topological information, both of which we must preserve as we transform the triangulation into a piecewise surface. In the next section, we will learn how to compute useful shape attributes from the triangulation. For example, fast algorithms exist for estimating surface normals and curvature on simplicial complexes. We will also discuss some disadvantages of the discrete surface that stem from the noise inherent in the raw data.

3.1 Discrete Surface Representation

In a triangle mesh, local surface features may be derived from differential geometry by evaluating partial derivatives to estimate the curvature at each point [3]. We use mean curvature H and Gaussian curvature K to classify the local surface into one of the eight types depicted Figure 3.2. A unique surface classification depends only on the signs of these two scalar values. Gaussian curvature distinguishes basic surface types: elliptic, hyperbolic, parabolic, and planar. Mean curvature incorporates an estimate of the normal vector, thus measuring the shape of the local neighborhood with respect to the orientation of the surface. By combining these two values, we create a surface-type-label image called an *HK-sign map*. The HK-sign map can be used to identify homogeneous surface regions. For example, one way to identify a surface in a triangle soup is to analyze connected components in the HK-sign map. We will discuss such a method in Section 4.1, in which we extract a seed component and grow it across a homogeneous surface region.

In order to compute curvature values in our input triangulation, we use the discrete curvature estimates summarized by Meyer *et al*. [25]. Let $\mathcal{T}$ be a triangulation. Figure 3.3a shows the 1-ring neighborhood of a point $x_i \in \mathcal{T}$ and an edge between x_i and its neighbor $x_j \in \mathcal{T}$. We use the two angles α_{ij} and β_{ij} opposite the edge to calculate the

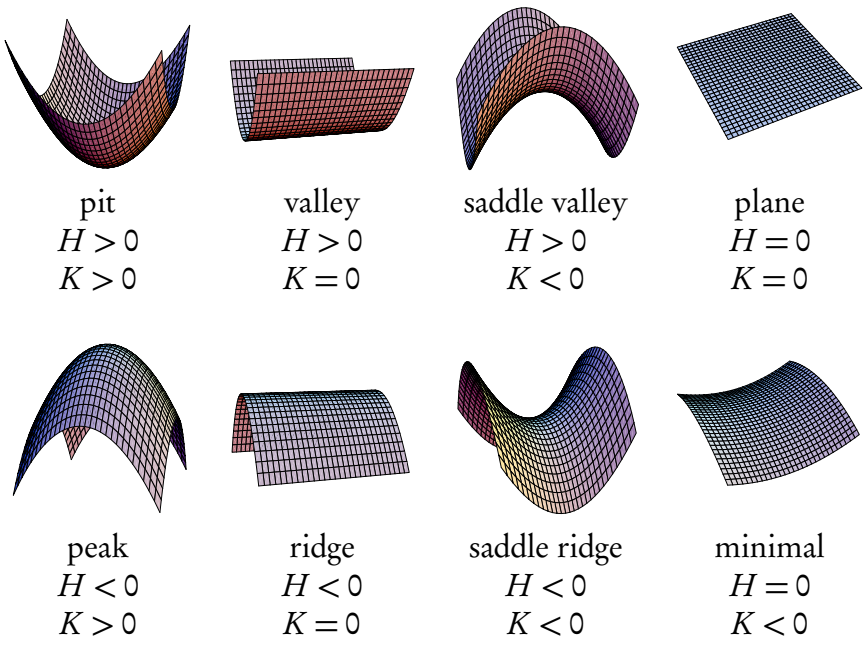


Figure 3.2. Surface Types by Curvature Sign. The sign of the mean curvature H and the Gaussian curvature K classify surfaces into eight types. These classifiers, computed pointwise, are used to create an image with eight levels of quantization called the HK-sign map. We analyze the HK-sign map using a connected components algorithm to identify homogenous surface regions.

Voronoi area, depicted in Figure 3.3b as a partition of the triangle by its perpendicular bisectors meeting at the circumcenter O. Summing Voronoi areas over all edges in the 1-ring neighborhood generates a discrete area element centered at x_i

$$\mathcal{A} = \frac{1}{8} \sum_{j \in \mathcal{N}_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij}) \|x_i - x_j\|^2. \tag{3.4}$$

In turn, summing the areas of the Voronoi regions over all vertices in a triangulation computes the area of the entire surface. This property is highly desirable, because we can

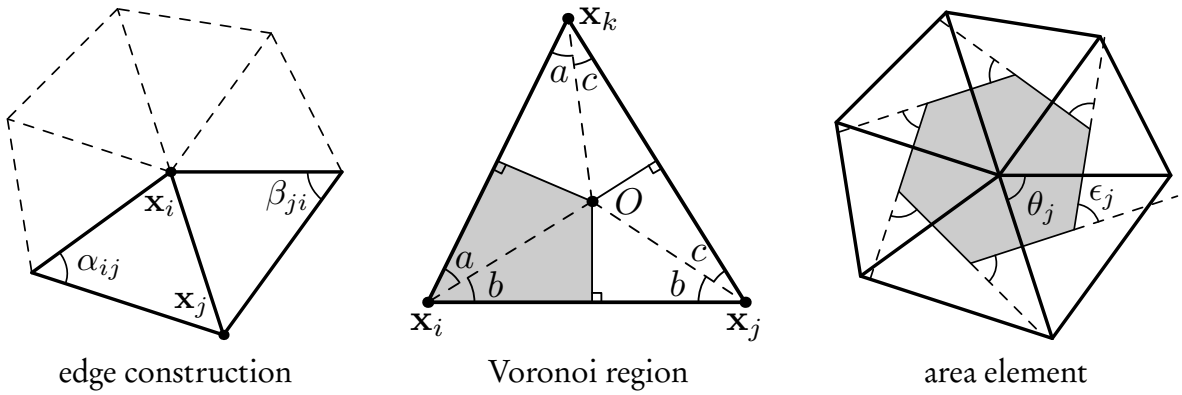


Figure 3.3. Discrete Curvature. (a) The 1-ring neighborhood around a point x_i, an edge from x_i to its neighbor x_j, and angles α_{ij} and βij opposite the edge. (b) The Voronoi region of a triangle partitions its area using the perpendicular bisectors meeting at circumcenter O. (c) The Voronoi area of the 1-ring neighborhood, exterior angles ϵ_j and interior angles θ_j.

use Voronoi regions to discretize continuous surface integrals of the form

$$\int_S f(u,v)\, d^2A. \tag{3.5}$$

If the entire neighborhood lies in a plane, the interior angles θ_j sum to 2π (Figure 3.3c). Any discrepacy in the angle measure must be caused by a local convexity or concavity. This observation suggests the discrete formulation of Gaussian curvature

$$K(x_i) = (2\pi - \sum_{j=1}^{|F_i|} \theta_j)/\mathscr{A} \tag{3.6}$$

where F_i is the set of faces in $\mathscr{N}_1(i)$. The Gaussian curvature measures the *angular defect* or deviation from 2π. We compute the mean curvature by taking half the magnitude of the mean curvature normal

$$H(x_i) = \frac{1}{2\mathscr{A}} \sum_{j \in \mathscr{N}_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(x_i - x_j). \tag{3.7}$$

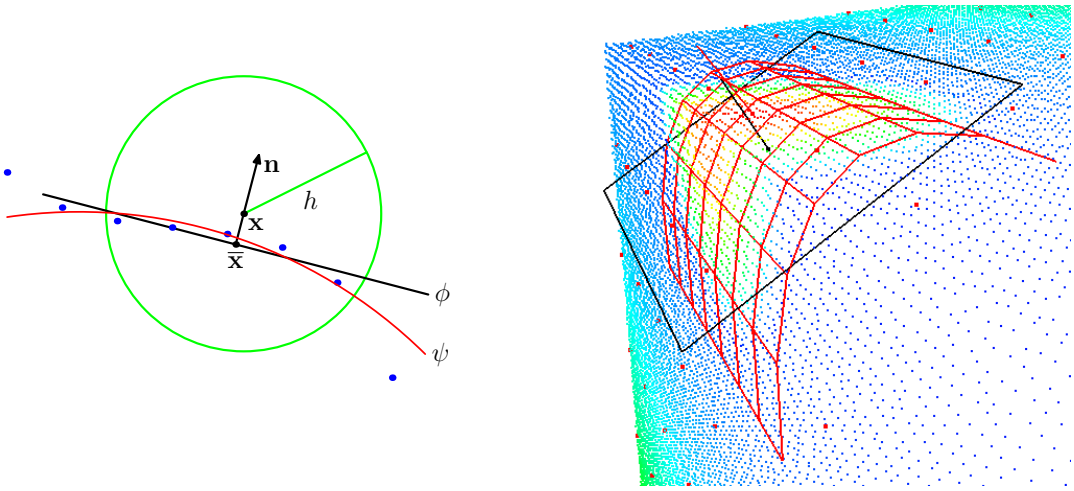
$H(x_i)$ computes the area-weighted average of the triangle normals in $\mathcal{N}_1(i)$.

1-ring operators are fast and exact, but they are very sensitive to noise. In raw range data, the area of the 1-ring neighborhood is usually small in relation to surface features such as texture or molding. Smoothing is required to ensure numerical stability. If the data is suitably organized, we can use convolution operators to estimate the partial derivatives. However, because of discontinuities present in the data, we prefer to use an anisotropic operator. In the next section, we introduce an anisotropic smoothing operator based on moving averages, which we use to field queries about surface attributes such as normal direction and curvature. Nevertheless, the discrete formulation is essential for enforcing consistent orientation, and it can resolve ambiguities where the sampling rate is inadequate. By combining complementary discrete and smooth structures, we aim to preserve both the geometric and the topological accuracy from end to end.

3.2 Moving Least Squares

An alternative approach to triangulation is to dispense with the expense of stored connectivity and define instead a neighborhood operator that can answer queries about the local area of a given point by isolating a small number of points nearby. Moving Least Squares (MLS) is one such operator, and it can be formulated in several different ways surveyed by Cheng *et al*. in [7]. The basic idea is to move a local least squares approximation around the input data in such a way that the surface properties vary smoothly from point to point. Given a query point x, we fit a polynomial to the points in the neighborhood of x:

$$f(x) = \min \sum_i w_i(||x - p_i||)\,||\mathbf{g}^T(p_i)\,\mathbf{c}(x) - f_i||^2. \tag{3.8}$$



(a) MLS projection (b) Surfel

Figure 3.4. Moving Least Squares. (a) The local coordinate frame $\phi = (\overline{x}, a)$ induced by the set of points in a neighborhood of radius h around x. The origin is $\overline{x}$, and the normal is a. (b) In 3D, the neighborhood is circular, and the second-degree approximation is a function of two variables.

The polynomial basis vector is $\mathbf{g}(x)$, and the vector of coefficients to be determined is $\mathbf{c}(x)$. Smoothness comes from the weighting function w_i, which constrains the support of the fitting function to a neighborhood of x. The weighting function is monotone decreasing as the distance from x increases. Thus, the influence of a sample point p_i on a local approximation changes gradually with distance. The Gaussian

$$w_i(d) = e^{-d^2/h^2} \tag{3.9}$$

is commonly given as the prototypical example.

In our implementation, we minimize the MLS fitting function using the singular value decomposition (SVD). At a given query point x, we form the weighted covariance matrix of the points p_i in the neighborhood of x, centered at their mean position $\overline{x}$. The SVD outputs the matrix $\mathbf{V}^T$ of right eigenvectors, which contains the normal vector

corresponding to the smallest eigenvalue. In addition, since $\mathbf{V}^T$ encodes the rotation from world coordinates to the local frame, it is convenient to compute the perpendicular distance from a point x to a plane ϕ as

$$d(x, \phi) = \mathbf{V}^T(x - \overline{x}) \cdot a \tag{3.10}$$

where a is the normal vector at $\overline{x}$.

We define the MLS surface by specifying the domain of the fitting function f. Let Ω be the vicinity of the surface—that is, the space of points that satisfy the sampling condition. To derive the projection operator, restrict f to Ω. Then, the projection operator $f : \Omega \rightarrow \mathbb{R}^3$ maps points from the vicinity of the surface to points on the surface. Hence, the surface is defined as follows:

$$\mathcal{S} = \{x \in \Omega : f(x)\} = \text{range}(f). \tag{3.11}$$

A point on the surface, augmented by a local coordinate frame and a polynomial approximation, is called a *surface element* or *surfel* by analogy to the term pixel (picture element).

In practice, neighborhoods are usually formed by constructing search data structures such as kd-trees. A kd-tree can find all points within a radius h of a given query point in logarithmic time. If a grid parameterization is available, a faster way to collect neighbors is to use a rectangular window in the image grid. The neighborhood size h can chosen independent of the window size, or h can be allowed to vary based on the neighborhood size. In either case, connected points that are far outside the neighborhood will be correctly suppressed. Thus, the MLS surface achieves a degree of anisotropy that we exploit during segmentation. In Chapter 4, we will make use of these local surface approximants to group points based on the likelihood that they belong to the same surface.

Chapter 4

Segmentation

The data structures that we build on top of the raw range data are monolithic representations that use a single model to represent the entire scene. The discrete triangulation contains the complete point set, to which it adds edge and face connectivity. The MLS surface also contains the complete point set, to which it adds a procedural component. In both cases, all of the input data must be present in order for surface properties to be evaluated. A single parametric model, however, cannot be used for urban scenes because of the underlying geometry. It is necessary to segment the image into regions such that each region fits a single model. For example, Figure 4.1 shows a range scan segmented into parametric planes. The coupling of segmentation and fitting is the essential challenge of the piecewise reconstruction problem. Although it is challenging, it also offers an opportunity to divide and conquer a very large dataset. Performing segmentation on one scan at a time enables parallelization. It also means that the reconstruction can be built incrementally, as new images are acquired. We will need to divise an algorithm to merge the images, but the size of the merging problem will be much smaller than the raw data, as we will see in Chapter 5.

We will describe our novel approach later on in this chapter. First, we examine several

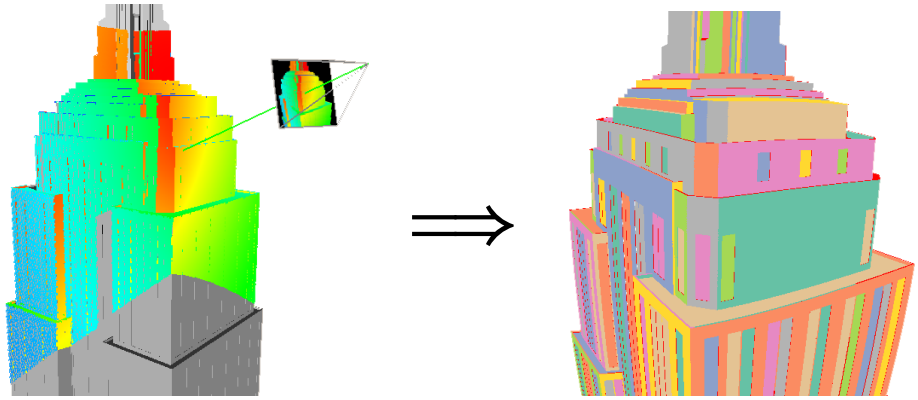


Figure 4.1. Range segmentation is the process of converting a raw range scan into a collection of model. On the left is a range scanner, depicted as a frustum, acquiring an image of a building. The geometry of the building is decomposed into a set of parametric planes.

existing segmentation methods, and discuss their advantages and drawbacks. There are an enormous number of segmentation algorithms in the literature. We confine ourselves to methods from which we draw ideas, and to methods that are relevant from a competitive standpoint. We begin with *region growing*, a popular method for range data because it is very fast. Region growing operates on a graph, and is based solely on local relations between points that are connected by edges in the graph. The advantage of a graph-based approach is that it is topology-preserving, and the speed of the algorithm is a consequence of the fact that it is purely local. The problem with a local algorithm, however, is that a constraint that is applied locally does not necessarily hold globally. For example, a smooth curved surface looks like a plane in a small neighborhood of a point.

To overcome the problem of locality, we turn to global methods. *Normalized cut* is one such method that is popular in image segmentation. While it is also graph-based, it operates in theory on the complete graph connecting all pairs of points. In practice, a somewhat sparser graph is used, but the idea is to allow global relations to influence the segmentation process. Unfortunately, even though global relations are included, normalized cut still does not guarantee that the output components satisfy a global constraint.

Constraints are applied only along edges, and no global test is applied to the results. Furthermore, the large matrix optimization problem at the heart of the normalized cut method is infeasible for very large datasets.

If a global test is needed to ensure output components of the correct type, then a parametric model is the best choice because it is so easy to evaluate. We examine the *Hough transform* because it searches for model parameters directly. The Hough transfers point relations to a global histogram space in which the peaks correspond to the optimal parameter values. There are two problems with this approach. First of all, optimization in Hough space does not necessarily lead to optimization in 3D space. Secondly, graph connectivity is not represented in Hough space, so it is necessary to augment the parameter discovery with some topology-aware postprocess.

Our method combines a graph representation with global histogram analysis and k-means-style iterative refinement. We call it *Cezanne*, because the flat areas of solid color produced by our algorithm are reminiscent of the painter's work. The output components are parametric functions with parameters that are optimized in global 3D space. Although we segment a graph, we do not use global optimization as normalized cuts does. We also do not need to build a large 3D histogram space as Hough transform does. We only use 1D histograms and thresholding to remove edges, which results in smaller subproblems. At the same time, iteratively applying a global parametric constraint enables a more exhaustive search for models of the desired type. Unfortunately, iteration is also time-consuming, and Cezanne is very slow. To speed it up, we focus on the model selection process, a significant bottleneck when ambiguities in the data require a great deal of iteration to resolve. By replacing the iterative component of our algorithm with *RANSAC* model selection, we created *Fast-Cezanne*, which produced the segmentations that we use as input to the merging and modeling algorithm described in Chapter 5.

4.1 Region Growing

The region growing algorithm has many variants [3, 5, 20]. A simple approach that illustrates the basic idea appears in Algorithm 3. A seed region is identified that is known

Algorithm 3 Region Growing

Input: A range image C.
Output: A planar segmentation $\mathcal{S} = \{C_k\}_{k=1}^{m}$.

1. Choose a seed point x_i and let $C_1 = \{x_i\}$.
2. Compute similarity w_{ij} for 1-ring neighbors $x_j \in \mathcal{N}_1(i)$.
3. If $w_{ij} >$ threshold, add x_j to C_1.
4. Repeat for each unprocessed neighbor of x_i.

with high confidence to be planar. This can be accomplished by partitioning the points into classes based on locally computed surface characteristics [3]. Then, a connected components algorithm is used to find a large area of similar points. Then, the selected component is subjected to erosion, isolating a central region away from any corners or boundaries. Thus, the flattest part of the surface is used to estimate the plane parameters. Once a seed has been selected, a flood fill algorithm propagates the labeling recursively along the edges of the graph. Visualize the grid parameterization as a graph like the one shown in Figure 4.2, which depicts a 3×3 region of a range image. The vertices are arrayed in scan order, and edges connect 4-neighbors in the image grid. Local surface charactistics are computed using one of the methods described in Chapter 3. The seed region grows along an expanding wavefront as long as the edge weight exceeds a threshold. Thus, the resulting region is guaranteed to be connected.

Many improvements to the region growing approach have been proposed. Gotardo *et al*. [15] use robust statistics to improve the extraction of seed regions. By identifying and removing outliers, the estimated parameters reflect the shape of the local surface with

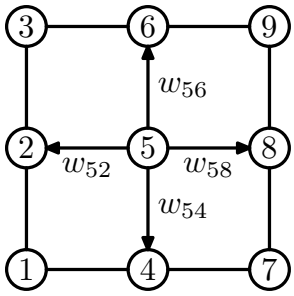
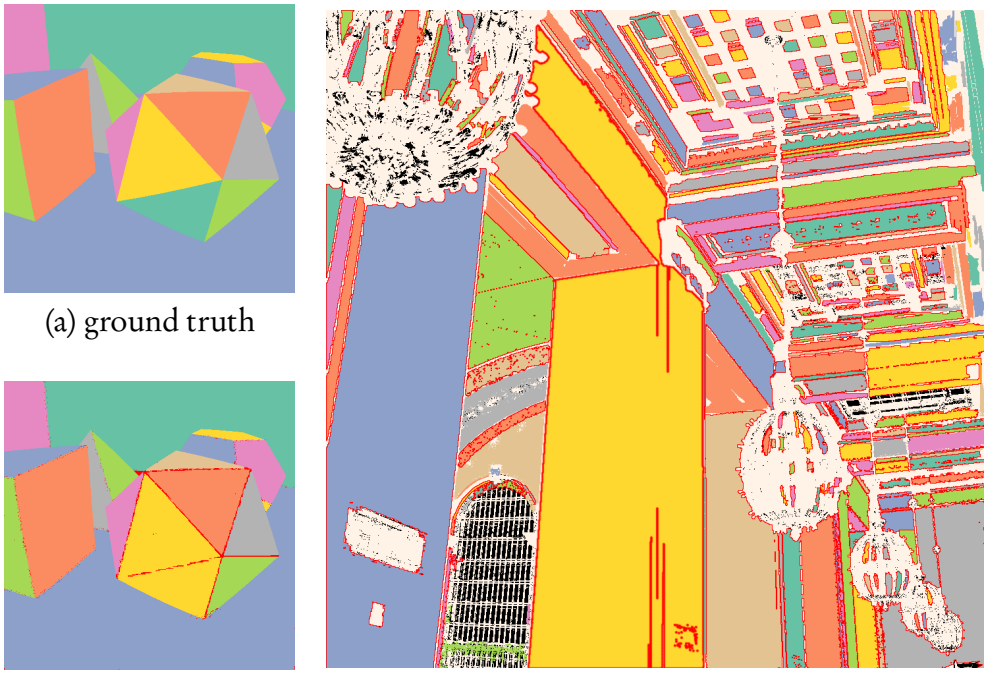


Figure 4.2. Region growing. Let vertex 5 be a seed region. We evaluate a similarity weight along each edge subject to a threshold that limits growth across discontinuities. If a weight exceeds the threshold, the incident vertex exhibits a high level of similarity to the seed region. We add the vertex to the region and recurse. The recursion terminates when no more edges can be added.

greater accuracy. After choosing a seed region, statistical information is further used to restrain the growth of the region so that the error in the final component is proportional to the initial error. In a similar statistical vein, Bab-Hadiashar and Gheissari [1] analyze residuals from parametric fitting to select a surface type from a set of functions of varying order. This step both improves the extraction of seed regions, and improves the region growing process by explicitly testing components using an analytic expression related the output surface type.

Chen and Stamos [5] use least squares fitting augmented with a heuristic classifier to improve normal estimation near discontinuities. The classifier recognizes certain common configurations and attempts to discard outliers before performing the least squares fit. Results appear in Figure 4.3 on simulated and actual data. The problem with a local notion of similarity is that points near component boundaries have no easily-defined membership. If the boundary is a corner connecting two or more planes, smoothing in the local approximation can easily cause the local area to appear as a curved surface instead of a sharp discontinuity. Figure 4.3b shows an example of this type of error where



(a) ground truth

(b) simulated data

(c) actual data

Figure 4.3. Region growing results. Black pixels denote an absence of data from the scanner. Red pixels are classified as nonplanar by the algorithm. (a) Ground truth segmentation of a simulated range image. (b) Region growing segmentation of the simulated image, containing an instance of undersegmentation merging two distinct faces of the icosahedron. (c) Region growing segmentation on actual data. The 999×999 range image was taken inside Grand Central Terminal, New York.

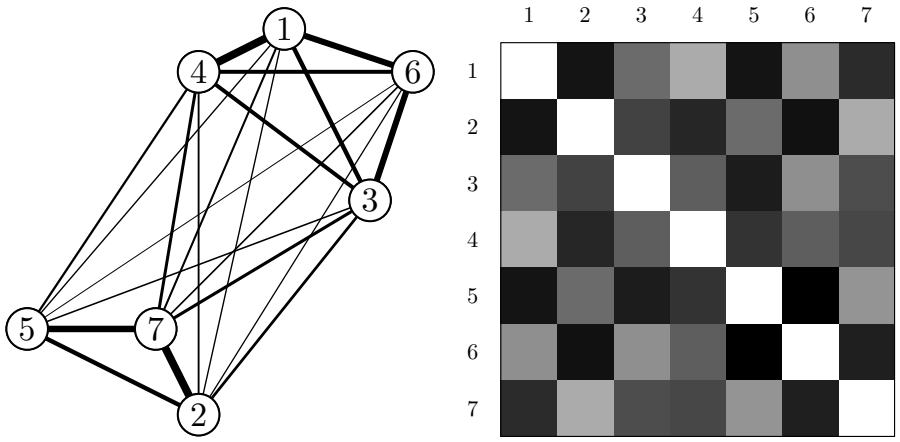
two faces of the icosahedron are merged into one. A large dihedral angle coupled with noise obscures the boundary between the two faces. In Figure 4.3c, the effect is pronounced on the large column in the center of the image. Four planes meet to the left of the capital, causing the local approximations in the area to be smoothed excessively. Even though the critical region represents a small bridge connecting large components that are clearly incompatible, the region growing algorithm cannot break the link without some larger-scale information that can resolve the ambiguities.

4.2 Normalized Cut

In contrast to the local approach exemplified by region growing, the normalized cut algorithm operates on complete graphs, and is thus inherently global. To formulate range image segmentation as a normalized cut problem, the points are represented as vertices in a weighted undirected graph. Edge weight is a function of similarity between two nodes, where similarity may be defined with respect to surface features computed locally, as in the case of region growing. The graph is then partitioned into two disjoint components by removing edges until it becomes disconnected. To evaluate the quality of the partition, a measure of dissimilarity is derived by analyzing the distribution of the edge weights between two components. This measure is called the *cut*. The purpose of a graph cut algorithm is to compute an optimal cut. In this section, we describe two such algorithms, *minimum cut* and *normalized cut*, and their application to range image segmentation. A matrix representation of a weighted undirected graph leads directly to a numerical algorithm for computing an approximation to the normalized cut.

A weighted undirected graph can be represented by an adjacency matrix, as we see in Figure 4.4a, which shows a complete graph $G = (V, E)$ with edge weights represented by varying line weight. The graph contains $n = |V|$ vertices, so the corresponding adjacency matrix in Figure 4.4b is $n \times n$, with each cell colored according to the weight of the edge between the corresponding vertices. Entries along the diagonal are white, since the similarity of any vertex to itself is always perfect. The weights w_{ij} decrease as the distance between vertex i and vertex j increases. Since G is an undirected graph, the corresponding adjacency matrix $\mathbf{W}$ is symmetric.

The *degree* of a vertex $v_i \in V$ is $d_i = \sum_{j=1}^{n} w_{ij}$. The degree measures the total weight associated with v_i. If we were to remove v_i from G, the total weight removed would be $d_i - w_{ii} = d_i - 1$. That is, $\text{cut}(\{v_i\}, V \setminus \{v_i\}) = d_i - 1$. The cut represents the cost of



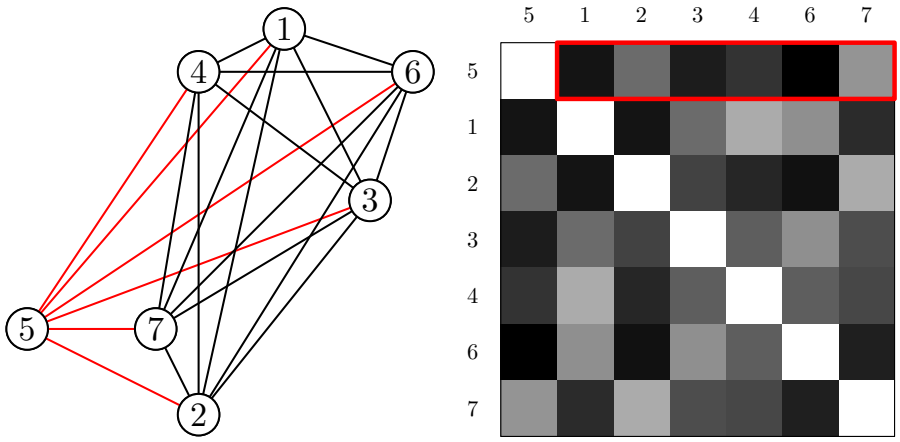
(a) $G = (V, E)$, $n = |V|$ (b) $\mathbf{W} = w_{ij}$

Figure 4.4. Graph representation. (a) A complete weighted undirected graph with edge weights indicated by line thickness. The weights are based on Euclidean distance between vertices: thicker lines represent shorter distances. (b) The corresponding similarity matrix with edge weights indicated by gray values. In this representation, lighter values represent shorter distances.

disassociating v_i from the rest of the graph. A large cut value indicates that v_i is tightly coupled to G, either because it contains many links, or because it contains particularly strong links. This definition of the cut generalizes to arbitrary subsets. Consider a partition of G into two disjoint subsets A and B, $A \cup B = V$, $A \cap B = \emptyset$. Then

$$\text{cut}(A, B) = \sum_{i \in A} \sum_{j \in B} w_{ij}. \tag{4.1}$$

Wu and Leahy [41] computed segmentations of grayscale images by minimizing Equation 4.1. They were able to devise an efficient solution, but, in the process, they observed a bias in the minimum cut criterion. Since the cut value is proportional to the number of edges, minimum cut tends to remove isolated vertices. Indeed, isolating v_5 achieves the minimum cut in our example. The edges to be removed are highlighted in Figure 4.5a. A reordering of the rows and columns of the matrix in Figure 4.5b groups the two compo-



(a) $A = \{5\}, B = \{1,2,3,4,6,7\}$ (b) $\text{cut}(A,B) = \sum_{j \in B} w_{5j}$

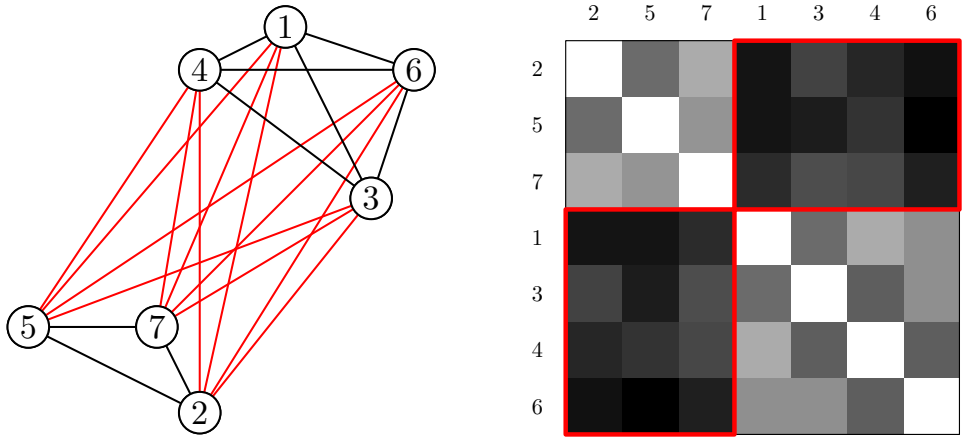
Figure 4.5. Minimum cut. (a) A partition that isolates vertex v_5, and the edge set representing the cut value of the partition. (b) The adjacency matrix $\mathbf{W}$, reordered to group the two subsets A and B in contiguous blocks. The value of the minimum cut is the sum of the outlined edge weights.

nents in contiguous blocks, with cut edges outlined in red. Minimum cut is crucial for computing maximum flows in directed graphs representing flow networks, but segmentation requires a more sophisticated cut criterion.

Shi and Malik [34] proposed the normalized cut as a way to produce more balanced subsets. The balance comes from a normalizing factor $\text{assoc}(A,V) = \sum_{i \in A} d_i$ applied to the cut value for a given subset that takes into account the total weight of the subset, a quantity that is analogous to the degree of a single vertex. The normalized cut

$$\text{Ncut}(A,B) = \frac{\text{cut}(A,B)}{\text{assoc}(A,V)} + \frac{\text{cut}(B,A)}{\text{assoc}(B,V)} \tag{4.2}$$

measures the cut cost of the partition as a fraction of the total weight of the graph. Thus, the bias toward small sets is removed by scaling the cut value in proportion to the size of the set. The cost is small if the cut separates two components that have few edges of low weight between them and many edges of high weight internally. The minimum



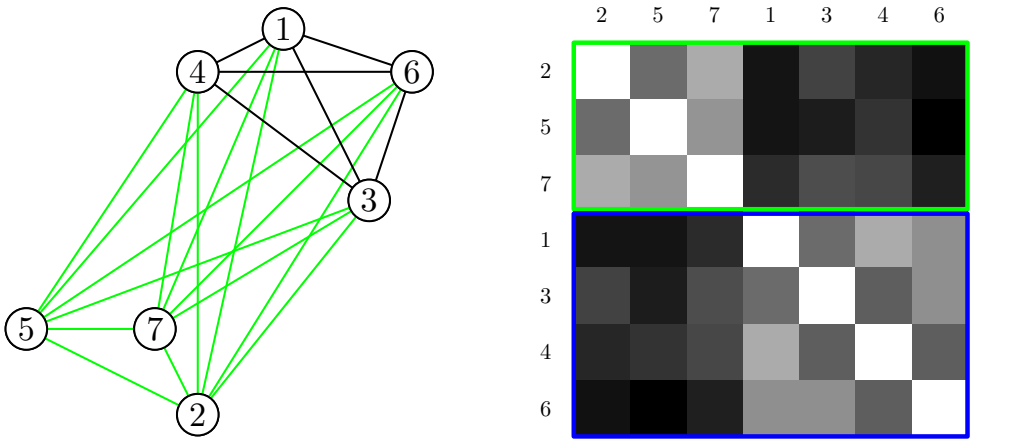
(a) $A = \{2,5,7\}, B = \{1,3,4,6\}$ **(b)** $\text{cut}(A,B) = \text{cut}(B,A)$

Figure 4.6. Normalized cut. (a) Spatially, the vertices are clustered into two groups $A = \{2,5,7\}$ and $B = \{1,3,4,6\}$. The normalized cut algorithm finds the desired cut exactly. (b) Since $\mathbf{W}$ is symmetric, $\text{cut}(A,B)$ and $\text{cut}(B,A)$ are reflections of each other through the diagonal.

normalized cut of the graph G in our example is shown in Figure 4.6a. The matrix representation is shown in Figure 4.6b with the rows and columns grouped by component. The partition produced by the normalized cut matches our intuition that the group of three vertices in the lower left constitute one cluster that is spatially distant from the group of four vertices in the upper right. The normalization factors $\text{assoc}(A,V)$ and $\text{assoc}(B,V)$ are shown in Figures 4.6c and 4.6d.

Solving the normalized cut problem involves stating it in a variational form that can be solved by numerical optimization. First, we define the *degree matrix* $\mathbf{D} = \text{diag}(d_i)$, $i = 1,2,\ldots,n$, which is simply a diagonal matrix with the degree of each vertex stored in the diagonal elements. To encode the membership of a set $A \subset V$, we define an *indicator vector*

$$\mathbf{x}_i = \begin{cases} 1 & \text{if } i \in A \\ -1 & \text{if } i \notin A \end{cases}. \tag{4.3}$$



(a) assoc(A, V) **(b)** assoc$(A, V) + \text{assoc}(B, V) = \sum_{i=1}^{n} d_i$

Figure 4.7. Normalizing factors. (a) The total association between A and V is used as a normalizing factor for cut(A, B). (b) The sum of the normalizing factors is equal to the total weight of G.

Given $\mathbf{D}$, $\mathbf{x}$, and the weight matrix $\mathbf{W}$, we can write the normalized cut as a Rayleigh quotient

$$\text{Ncut}(\mathbf{x}) = \frac{\mathbf{x}^T(\mathbf{D} - \mathbf{W})\mathbf{x}}{\mathbf{x}^T\mathbf{D}\mathbf{x}}. \tag{4.4}$$

The importance of the Rayleigh quotient is that its critical values are the eigenvectors of the Laplacian matrix $\mathbf{D} - \mathbf{W}$. The value of the quotient at these critical points is equal to the corresponding eigenvalue. Since the Laplacian matrix is symmetric, its eigenvalues are real. Thus, the quotient reaches its minimum value when the eigenvalue is smallest and $\mathbf{x}$ is equal to the corresponding eigenvector. The Rayleigh quotient has a physical meaning in many contexts as an expression of the average energy of a system. The variational principle states that we can find the minimum energy normalized cut by minimizing the average energy in the form of a Rayleigh quotient. This integer programming problem is NP-complete.

However, if the indicator vector $\mathbf{x}$ is relaxed to take on real values, the minimization

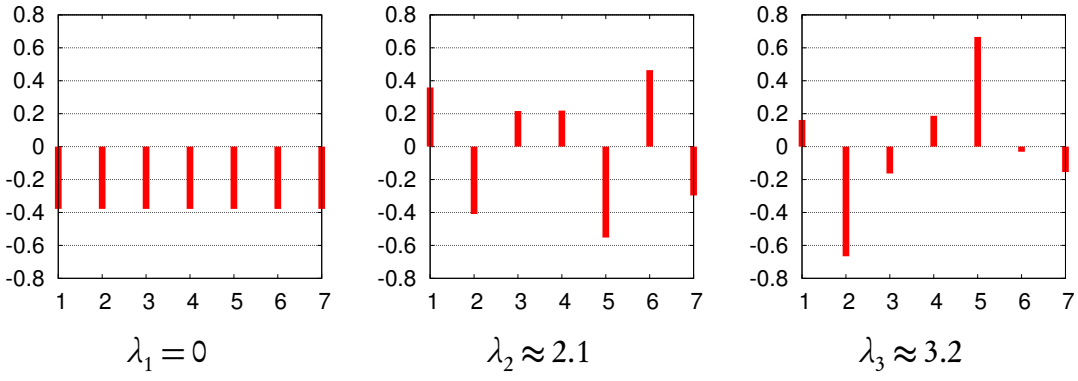


Figure 4.8. Eigenvectors of the Laplacian matrix. (a) The smallest eigenvalue λ_1 is 0, and the corresponding eigenvector is a constant. (b) The eigenvector corresponding to the second smallest eigenvalue is the minimum normalized cut, with 0 as the splitting value. (c) The eigenvectors corresponding to the next smallest eigenvalues optimally partition the first two parts, but they are not often used in practice due to the accumulation of numerical errors.

can be recast in the following form:

$$\min_{\mathbf{x}} \mathbf{x}^T(\mathbf{D}-\mathbf{W})\mathbf{x} \quad \text{subject to} \quad \mathbf{x}^T\mathbf{D}\mathbf{x} = 1. \tag{4.5}$$

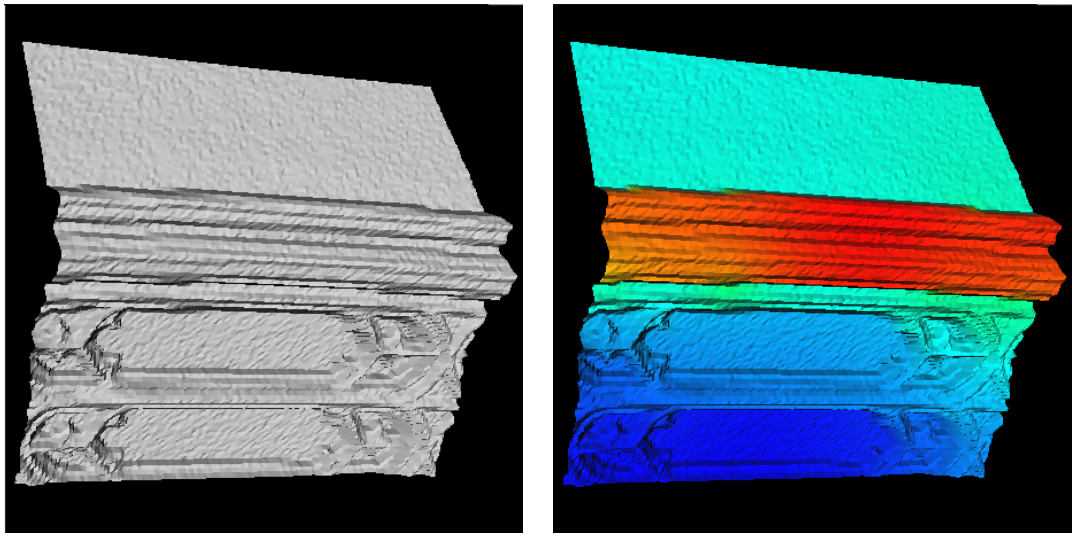
The relaxation in Equation 4.5 is equivalent to the generalized eigenvalue problem

$$(\mathbf{D}-\mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}, \tag{4.6}$$

which can be solved by numerical methods. A generalized eigenvalue solver can be used to search for the smallest eigenvalues, or one can solve the standard eigenvalue problem

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{y} = \lambda\mathbf{y} \tag{4.7}$$

with $\mathbf{y} = \mathbf{D}^{\frac{1}{2}}\mathbf{x}$. The eigenvectors corresponding to the three smallest eigenvalues in our example are shown in Figure 4.8. Since $(\mathbf{D}-\mathbf{W})\mathbf{1} = 0$, the smallest eigenvalue is always



(a) surface patch (b) indicator vector

Figure 4.9. Normalized cut results. (a) A 128 × 128 detail from a range image taken in the Great Hall at City College of New York. (b) Continuous indicator vector produced by an eigenvalue solver. Simple thresholding can easily split the continuous vector into two pieces separating the ridge lines from the flatter surfaces above and below it.

equal to 0 with eigenvector **1**. The eigenvector $\mathbf{x}_2$ corresponding the second smallest eigenvalue encodes the solution. By picking a splitting value that partitions the real-valued elements of $\mathbf{x}_2$ into two parts, two disjoint components result. In the example shown in Figure 4.8b, splitting $\mathbf{x}_2$ around 0 produces the desired partition. Further partitions are achieved by recursively splitting the output components.

A example of the output of the normalized cut algorithm on range images is shown in Figure 4.9. The figure depicts a continuous indicator vector produced by the eigenvalue solver before a splitting point is chosen which bipartitions the vertices. A practical implementation must ignore most of the connections between vertices that are spatially distant. Using a Laplacian matrix containing a large number of zeroes enables a sparse solver to be used. Some random connections are retained, however, because a small amount of truly global information improves numerical stability. Even a sparse solver cannot handle

a problem of the size of the range images in our dataset.

4.3 Hough Transform

Much of the power of the normalized cut algorithm stems from the application of global information to enrich the search space. But the output is not guaranteed to satisfy any global similarity measure. An output component could be the most planar for a given cut even if it is not really planar at all. The output consists only of a labeling, so we have no way to evaluate the result. Since we are searching for planes, we would be better served by a parametric algorithm that is designed to fit a small set of coefficients. One such algorithm is the Hough transform [12], a simple voting procedure in the dual parameter space. Given a parametric equation describing a structure, such as a plane, that we wish to find in a range image, each point votes for a configuration space, or range of possible planes through that point. Maxima in the dual space represent the parameters of planar structures in the primal space. Similar to region growing and normalized cut, the Hough transform uses local surface approximation to construct a notion of similarity. The parameter values that receive a vote at each point are based on the local surface fit. Voting takes place in a discrete array of buckets that quantize the dual space. This parameter space is called the *Hough space*. Hough space is similar to a histogram, generalized to an arbitrary number of dimensions.

The Hough space depends on an interesting relationship between the input points and their representation in the dual space. Figure 4.10a shows three collinear points in the plane. Let us first consider a single point $\mathbf{x}_1$. To plot $\mathbf{x}_1$ in the dual space, all possible parameter values that are consistent with $\mathbf{x}_1$ must be determined. That $\mathbf{x}_1$ votes for all possible lines in the plane that intersect it. Traditionally, a polar formula is used to avoid singularities at vertical lines. The pair (θ, ρ) represents a line, where $r \geq 0$ is the

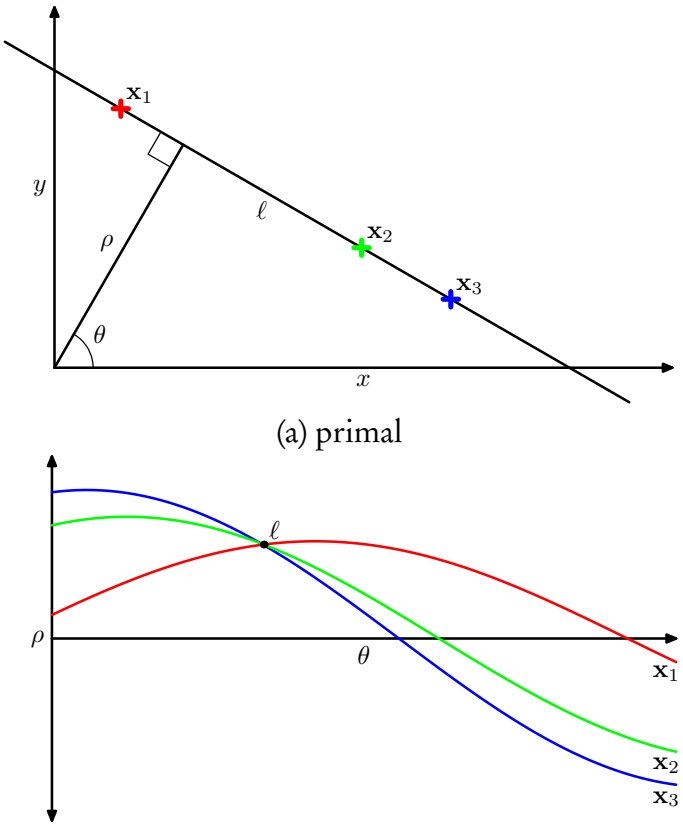


Figure 4.10. Hough space. (a) Three collinear points in Cartesian space, and the parameters (θ, ρ) mapping the line ℓ to a point in the dual space. (b) Sinusoids in the dual space representing the possible lines through $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$, and their common intersection point at ℓ.

perpendicular distance from the line to the origin and $0 \leq \theta \leq \pi$. The parameterization taking $\mathbf{x}_1 = (x_1, y_1)$ to the dual space (θ, ρ) is

$$x_1 \cos\theta + y_1 \sin\theta + \rho = 0. \tag{4.8}$$

Given $\mathbf{x}_1$, we plot equation (4.8) for all values of θ between 0 and π, which produces a sinusoid in the dual space (Figure 4.10b).

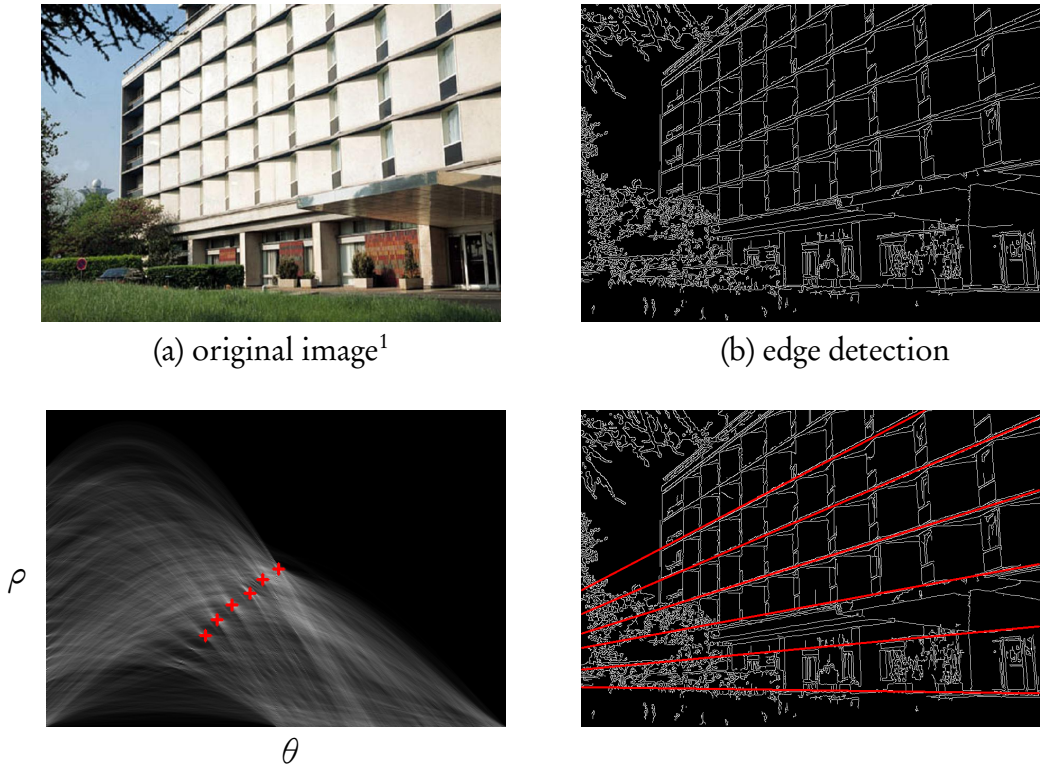
Now, note the three collinear points $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ in Figure 4.10a. One unique line

connects all three points. Sinusoids for each of these points in the dual space intersect wherever they exhibit collinearity. Lines in the primal space map to points in the dual space. If many points lie on a given line in the primal space, the corresponding point receives many votes in the dual space. Thus, the parameters of the lines in an image are estimated by the peaks in the dual space.

To detect lines in a 2D image using the Hough transform, a local feature is defined that reflects the best estimate of where the lines in an image reside. This is usually achieved using an edge detection algorithm. Consider the color image in Figure 4.11a, which contains several strong linear features. A Canny edge detector produced the edge image in Figure 4.11b. An edge image simplifies the voting process significantly. All points that were not classified as edge points by the edge detector can be safely ignored. Secondly, since the Canny edge detector performs nonmaxima suppression, the edges are represented by one-pixel-wide binary curves, so each edge point gets one vote, and no fractional values need to be processed. Of course, the quality of the edge detection is critical for finding accurate parameters. This is just one of the challenges to the practical and robust implementation of the Hough transform.

Another difficult decision is the choice of grid size for the dual space. An appropriate grid size can be difficult to determine, depending on the range and distribution of the input values and the amount of noise present. If the grid is too coarse, many peaks may coalesce into one, and the resulting line will be an average of several lines in the image. If the grid is too fine, one line may be dispersed into several buckets such that no peak can be discerned. In Figure 4.11c, the grid size matches the dimensions of the input image. Each edge point in the edge image corresponds to one sinusoid in the dual image.

Finding the peaks in the dual image is another challenging problem in the Hough transform pipeline. Since the peak may be distributed among several buckets, simply choosing the bucket with the most votes can lead to an inaccurate parameter fit, espe-



(a) original image[1] (b) edge detection

(c) accumulator array (d) output

Figure 4.11. Line detection using the Hough transform. First, linear features are isolated using a Canny edge detector. The edge pixels are then transferred to a histogram known as Hough space, where each one votes for all possible lines passing through it. Peaks in Hough space identify the parameters of the best-fitting lines.

cially if the grid size is relatively coarse. One remedy is to threshold the dual image and then search for connected components. The center of mass of a component found in this manner may represent a better guess at the true parameters. A more sophisticated technique is to fit a surface to the peak region and determine the maximum point analytically. In our example, it is sufficient to choose the bucket with the most votes, but even this procedure is not as simple as it seems. It will usually be that case that buckets near the top peak have similar values. These must be suppressed before searching for a second

[1]Image courtesy of OpenCV.

peak. Using the simple expedient of setting a 10×10 neighborhood to 0 around each peak as it is discovered, we collected the six best peaks in the dual space, corresponding to the lines shown in Figure 4.11d.

The Hough transform exploits global information because it allows points that are widely separated in the image grid to vote in concert. This can lead to robust parameter estimates, but it can also produce phantom lines, especially in the presence of noise. For instance, the trees and other vegetation on the left side of the image in Figure 4.11a result in high frequency edge information that causes votes to accumulate for many short line segments. Noise in the edge image is thus transferred to the dual space. In our simple example, the quality of the parameter estimates degrades rapidly as we move beyond the top six peaks.

Extending the Hough transform to detect planes in 3D is described by Vosselman *et al*. [39]. A parameter space is constructed along similar lines to the 2D case by decomposing the plane equation into an orientation component and a distance component. The orientation is composed of two parameters θ and ρ that encode the direction of the normal vector at a point $\mathbf{x}_i$. The third component ρ then encodes the perpendicular distance to the plane through $\mathbf{x}_i$ with normal $\mathbf{n}$. To estimate the normal vector, a local frame can be obtained by least squares fitting. The local plane Π centered at a point $\mathbf{x}_i$ is shown in Figure 4.12, along with the tuple (θ, φ, ρ) that parameterizes Π in the dual space. Note that, in the case of planes, there is no analogue to the edge detection step. Although the related method of *tensor voting* [16] incorporates a classification of points into spherical, linear, and planar neighborhoods that is similar in principle to the use of edge detection in 2D, extension of the Hough transform to 3D usually relies on the plane parameters only.

It is important to note that the parameters θ, φ, and ρ form a 3D dual space. Given both the large size and high resolution of the input range images, an enormous search

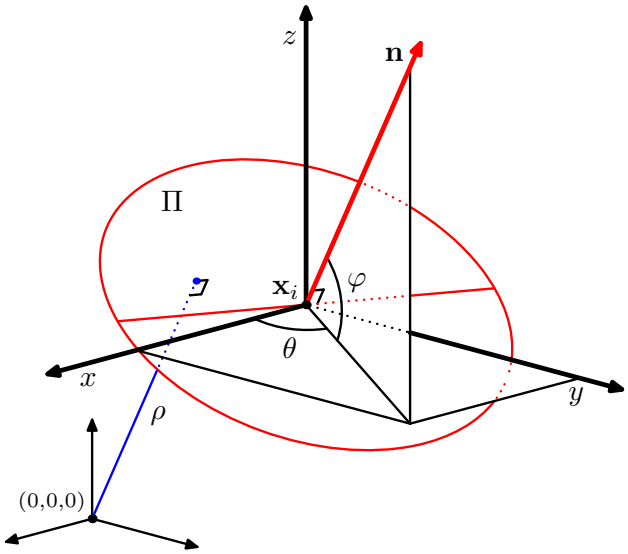


Figure 4.12. 3D Hough transform. Π is a local frame with normal **n**, centered at $\mathbf{x}_i = (x_i, y_i, z_i)$. Two parameters θ and φ encode the orientation of the local frame, and a third parameter ρ encodes the distance from the plane to the origin.

space is required to ensure comparable accuracy in the estimated plane parameters. One way to mitigate the impact of the large search space is to separate the estimation of the orientation parameters θ and φ from the estimation of the distance parameter ρ. The Hough space for the normal vector estimation is a Gaussian sphere, which is discretized to create an accumulator for voting. Once orientation parameters are determined for the most likely normal direction, we construct a 1D histogram to resolve likely distance values for all planes perpendicular to the selected normal.

Results using the 3D Hough transform are shown in Figure 4.13 on range images of the exterior of the Thomas Hunter Building at Hunter College in New York. Plane parameters produced by the Hough transform can achieve high accuracy relative to local methods, because outliers can be screened out easily and effectively using a closed-form point-to-plane distance function. Resolution depends critically on the size of the space, however, because the 3D parameter space can have prohibitively large memory require-

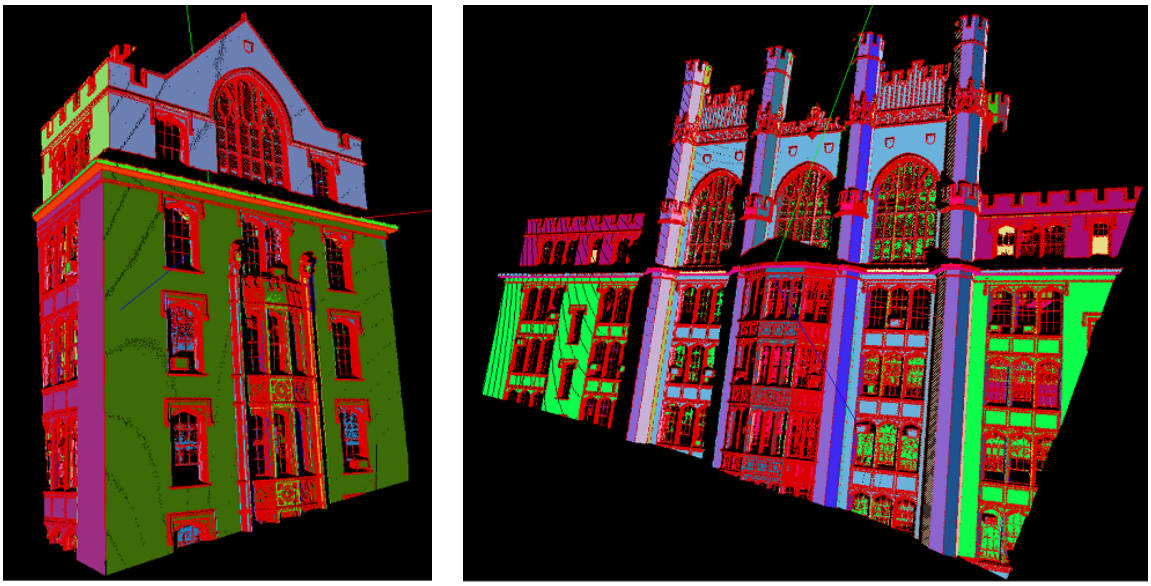


Figure 4.13. Hough transform results. Point renderings of two views of the Thomas Hunter Building, Hunter College, New York. Red indicates points classified as non-planar. Other colors represent groupings of points that voted for the same plane. Note the complex shapes of the coplanar pieces due to window panes, air conditioners, and sculptural elements of the facade.[1]

ments if the desired discretization is too fine. In addition, while this method can be used to find plane parameters on unorganized point clouds, determining the connected components in the output is not a trivial problem. Therefore, though the Hough transform can be used for model selection, some postprocessing would be required to solve the complete segmentation problem.

[1]Images courtesy of Hadi Fadaifard.

4.4 Our Method: Cezanne

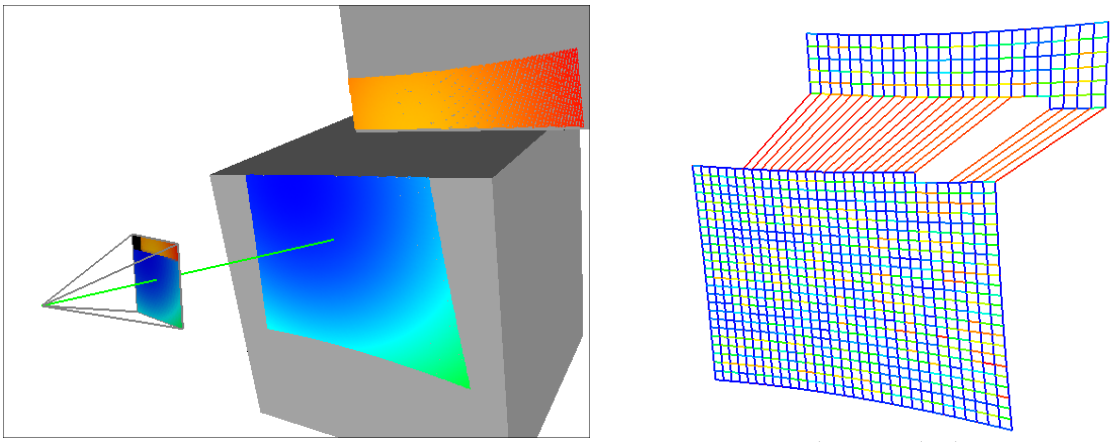
A similarity measure that uses only local relations cannot resolve boundaries cleanly, due to approximation errors that increase near corners and depth discontinuities. In addition, local variations in scale and sampling rate make suitable thresholds difficult to determine, since a single set of parameters is rarely optimal in every area of an image. As a result, a method such as region growing often produces boundaries that are extremely complicated. On the other hand, both global methods that we considered had significant drawbacks: normalized cut is intractable for very large data sets, and the Hough transform is only a partial solution. Cezanne [42] combines the simple but powerful graph formulation used in region growing with a global parametric model to find accurate corners and boundaries.

Algorithm 4 Cezanne

Input: A triangulation $\mathcal{T}$ of a range image, and an error tolerance ϵ.
Output: A piecewise surface $\Psi = \cup_{i=1}^{m} S_i$ where $S_i = \langle \Pi, \Omega, f \rangle$ and f is planar.

1. **Initialize** Ψ to the union of connected components S_i in $\mathcal{T}$.
2. **Fit** a least-squares plane to each S_i, and store the standard error σ_i.
3. **Reproject** all vertices. Points closer to the plane than $3\sigma_i$ are retained. Points that are connected in $\mathcal{T}$ are split into separate components.
4. **Repeat** steps 2–3 until $\sigma_i < \epsilon$. Replace S_i in Ψ and remove the new components from $\mathcal{T}$.
5. **Repeat** steps 1–4 until the data is exhausted.

The method is top-down: all vertices in the grid parameterization are initially connected, and we proceed by cutting away nonplanar edges to break up the mesh into components. Then, we fit plane parameters to each component's points. The key idea is to negotiate between local and global perspectives on what areas of the mesh seem to be planar. In the region growing method, we removed or retained an edge based on a threshold applied to its weight value. In Cezanne, we consider the probability distribution of all



(a) range scanner (b) scored edges

Figure 4.14. Graph representation of a range image with distance-weighted edges. A normal distribution based on the edge weights identifies edges across the depth discontinuity as outliers.

edge weights. We alternate between fitting plane parameters to a set of points, and fitting points to a set of plane parameters. The iteration converges because the error characteristics of the component change with membership. A connected component is easily tested to see if it is planar by computing the standard error over all member points with respect to the current parameter values. If not, then the process repeats on the remaining edges until a planar component becomes detached. We cut away areas of ambiguity until demonstrably planar models remain.

Figure 4.14 shows a depth discontinuity covered by edges from the triangulation. The edges rendered in false color based on their length. The edges covering the depth discontinuity are colored red, since they are the longest edges in the image. We compute point-to-point distance using Equation 3.10 for each edge in $\mathcal{T}$. Then we generate similarity weights by fitting a normal distribution to the distance values. By analyzing the distribution of edge weights, we can isolate the discontinuity edges as outliers, and remove them from the triangulation. A typical distribution of edge weights appears in

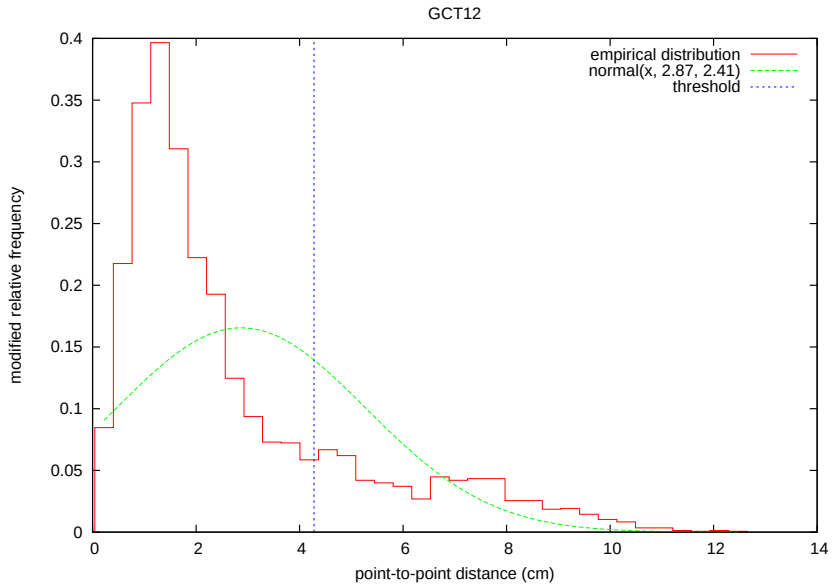


Figure 4.15. Edge removal by histogram thresholding. A typical histogram of edge weights contains one peak representing edges connecting adjacent points on a flat surface, and several smaller peaks representing either more distant surfaces or depth discontinuities. Starting at the mean, we descend to a local minimum, which identifies a good candidate threshold.

Figure 4.15.

Depth discontinuities appear as peaks in the distribution. The higher peak to the left of the mean shows that most of the points in the image were a little over 1cm apart. This distance represents the spread between adjacent vertices in the grid that fell on the same planar surface. The spread distance is determined by the depth of the surface, because the rays travel in a frustum. Thus, the more distant surface produces a second smaller peak to the right of the mean. Using the mean as an initial guess, we descend to the closest local minimum, which separates the two surfaces at the discontinuity. The changing distribution of weights makes it possible for repeated application of the splitting method to find different sets of planes. By contrast, region growing would produce the same components on each pass because there is no mechanism for automatically adjusting the parameters.

After thresholding, we split the graph into connected components. A component is rejected immediately if it has fewer than a given minimum number M. Otherwise, we fit a plane to each candidate component. Parameter estimation is accomplished by constructing an unweighted covariance matrix in homogeneous coordinates, and solving for the eigenvector corresponding to the smallest eigenvalue using SVD. The eigenvector is a plane equation in point-normal form $\Phi = (\mathbf{n}, p)$, where $\mathbf{n}$ is the surface normal and p is the perpendicular distance to the origin. The distance between a point $\mathbf{x}$ in space and a global plane Φ is

$$D(\mathbf{x}, \Phi) = \mathbf{n} \cdot \mathbf{x} + p. \tag{4.9}$$

If the standard error of the residuals is smaller than a predetermined tolerance τ, then the candidate is planar. Recursing on the nonplanar candidates, we construct a hierarchy of nested components, such that the planar components stored in the leaves.

Refinement proceeds bottom-up by the k-means method using objective function

$$J = \min_{\{S_i\}_{i=1}^{k}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} D(\mathbf{x}, \Phi_i). \tag{4.10}$$

Each level of the component hierarchy partitions $\mathcal{T}$ into connected subspaces. At each level, we alternately relabel points and update parameters until component membership stabilizes. Moving up the hierarchy, we fit to larger and larger sets of points, increasing both accuracy and coverage simultaneously. Because the thresholds are based on local minima, overfitting may occur. Redundant planes produced by overfitting can be merged later by analyzing adjacency relations between output components.

While it models corners and depth discontinuities very well, Cezanne is expensive. Figure 4.16 shows that the algorithm scales poorly. Many passes are required to correctly label as many points as possible. Because local approximations degrade so much in quality

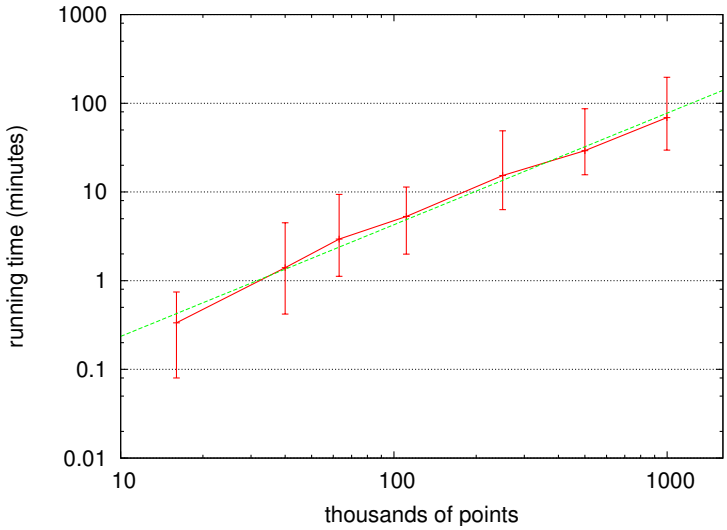


Figure 4.16. Timings for Cezanne. Log-log plot of number of points vs. mean running time for 25 scans on a 3GHz Intel Xeon 5160 processor with $w = 5$, $\tau = 6$mm and m ranging from 400 points at full size down to 40 for the smallest tile. Error bars indicate minimum and maximum times. The best-fitting line has a slope of 1.3.

near discontinuities, isolating them may take several levels of recursion. In the best case, it is a typical recursive divide-and-conquer algorithm, which runs in $\Theta(n \lg n)$ time. In the worst case, however, one component containing the minimum number of points will be extracted on each pass, so total the running time is $O(n^2)$. To investigate the average case, we segmented 25 actual range scans from Grand Central Terminal at their full size of 999×999 pixels. We also tiled the images into 1×2, 2×2, 3×3, 4×4, 5×5, and 8×8 regions. Results appear in figure 4.16. The large errors reflect a large variance in image content. A straight line through the mean values in the log-log graph suggests an average case power law relationship with a slope of 1.3. Even in the average case, this method is still too slow for large datasets. The bottleneck here is the model selection process. In Section 4.5, we will use random sampling to greatly increase the speed of model selection.

4.5 RANSAC

Cezanne discovers planes somewhat laboriously by removing edges from a graph until suitable connected components emerged. This did not yield an efficient algorithm, but we did observe a pattern in the distribution of edge weights in a typical scene: almost all of the edges connect coplanar vertices. We invariably find one very large peak near the minimum distance, and a long right tail, as we saw in Figure 4.15. This shape is evidence of spatial redundancy in the image, a property that we can exploit to improve the efficiency of the model selection process. Schnabel *et al*. [30] devised a sampling method based on the RANSAC algorithm of Fischler and Bolles [14] that greatly simplifies and speeds up the model selection process. RANSAC sampling, like Hough transform, finds model parameters but does not preserve topology. Since they merge the range images before segmentation, they must establish connectivity after the model is discovered, by projecting the points to an image at a carefully-selected resolution and applying a connected components algorithm. We operate on individual images, however, so we can preserve topology from end to end.

RANSAC sampling works by extracting one component at a time. Spatial redundancy is framed in terms of a prior probability that captures the assumption that points close together in 3D space are likely to belong to the same surface. The number of points to extract is determined by the number of parameters in the model. For example, in the planar case, exactly three points are needed to fit a plane unambiguously. Restricting the domain to points in a ball with a fixed radius increases the chances of choosing inliers. However, the radius must be chosen carefully. If it is too large, points that do not belong to the same surface may be included. If it is too small, parameters may be fit to the noise instead of the signal. The RANSAC method selects an appropriate scale using an octree, which partitions space into contiguous regions. An example of an octree search

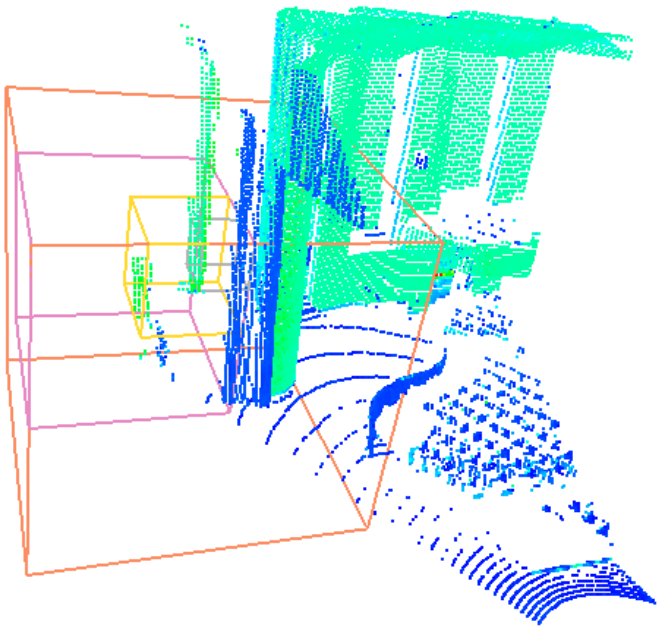


Figure 4.17. Spatial sampling using an octree. The octree decomposes space into nested cuboids. At a given level of the hierarchy, one cell of the octree represents a contiguous region in which a feature of interest may be found.

is depicted in Figure 4.17. Each level of the octree represents a nested enclosure around the query point. By randomly selecting the source set of point samples from the among the enclosing octree levels, the scale adapts to different areas of the image. Speed also may be gained by maintaining statistics on the chosen scales, so subsequent queries can be weighted to favor frequently-occuring levels.

Once a model is chosen, it receives a score based on the size of the largest connected component among points that are close to model. By always selecting the best score among a number of random choices, we are assured of selecting one of the largest remaining planes. Time complexity depends on how many models we need to consider to be sure we will have a good one among the choices. Let $\mathscr{P}$ be an input point cloud containing N points, and assume that a shape of size n resides in $\mathscr{P}$. Given a shape func-

tion Π^m having m degrees of freedom, m points must be selected from $\mathscr{P}$ to construct a candidate shape. The probability of detecting the shape on the first pass is

$$P(n) = \binom{n}{m} \Big/ \binom{N}{m} \approx \left(\frac{n}{N}\right)^m. \tag{4.11}$$

The probability of success after t trials is the complement of the probability of failing t times in a row:

$$P(n,t) = 1 - \big(1 - P(n)\big)^s \tag{4.12}$$

A threshold τ imposes a minimum acceptable probability of success, so the required number of trials can be determined by solving for t:

$$t \geq \frac{\ln(1-\tau)}{\ln\big(1-P(n)\big)}. \tag{4.13}$$

For small $P(n)$, the Taylor series expansion of the denominator is equal to $-P(n) + O\big(P(n)^2\big)$. Since the numerator is constant, the asymptotic complexity of the selection algorithm is $O\big(\frac{1}{P(n)}S\big)$, where S is the cost to score a candidate.

Algorithm 5 RANSAC Model Selection

Input: A point cloud $\mathscr{P} = \{p_1, p_2, \ldots, p_N\}$, an octree containing all of the points, a distance threshold ϵ, and a probability threshold τ.

Output: A model $(\Pi^m, \mathscr{L})$ where $\mathscr{L} \subseteq \mathscr{P}$ and Π is a shape function defined on $\mathscr{L}$ having m degrees of freedom.

1. Draw a random point p_i from $\mathscr{P}$, and select a random octree cell containing p_i.
2. Draw m random points from the selected octree cell, and fit an instance Π_i of the desired shape function to the points.
3. Score model i by counting the number of points that are within ϵ of the surface.
4. Repeat steps 1–3 until $P(|\mathscr{L}|, t) > \tau$
5. Return the model with the highest score.

The output segmentation is produced by iteratively applying the model selection al-

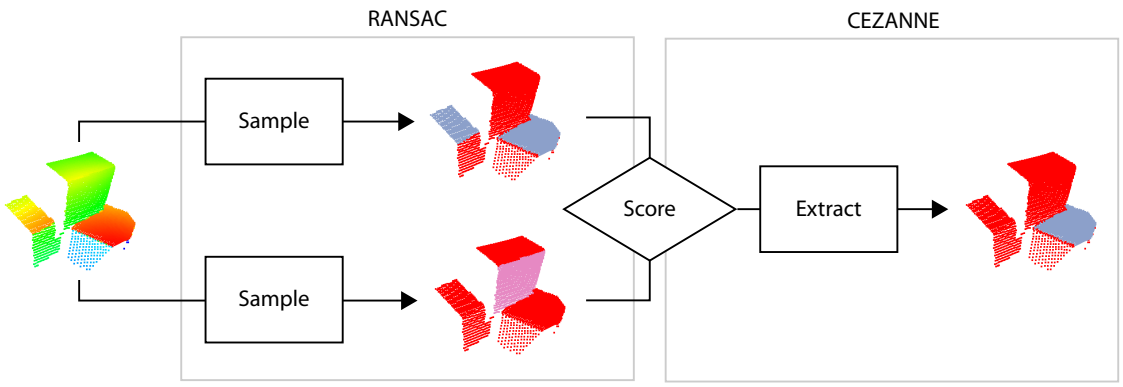


Figure 4.18. The Fast-Cezanne algorithm combines our iterative segmentation algorithm, Cezanne, with RANSAC model selection. Both algorithms employ evaluation to compare potential models with each other.

gorithm, consuming the input points until no more good-sized components remain. After each iteration, thresholds and probability weights can be updated to favor surfaces that have similar attributes to those that have already been accepted. For example, the success rate for each octree level can be used to overweight commonly recurring scales. The running time is highly dependent on content, running much faster if the scene contains a few large planes. On the other hand, curved shapes and complicated façades require more iterations.

4.6 Fast-Cezanne

Table 4.1 shows how well the various segmentation methods described in this chapter address the requirements stated in Section 1.1. None of the methods examined meets all of the requirements, including our own method, Cezanne, which is not fast enough to be able to handle a large amount of data. Region growing is exceptionally fast, and is therefore commonly used to segment range data. Normalized cut, on the other hand,

	Fast	Global	Parametric	Topological
Region Growing	✓			✓
Normalized Cut				✓
Hough Transform	✓		✓	
Cezanne		✓	✓	✓
RANSAC	✓	✓	✓	

Table 4.1: Comparison of Segmentation Methods. None of the methods satisfies all of our requirements. Cezanne and RANSAC both satisfy three out of four requirements, with complementary strengths and weaknesses.

depends on solving a large matrix problem, so the size of a feasible problem is severely limited. The Hough transform can be used for large datasets, but it is sensitive to the choice of grid size, and the tradeoff between speed and accuracy may be difficult to negotiate. RANSAC, on the other hand, can perform relatively fast model selection while enforcing a global error tolerance. Based on their speed, region growing and RANSAC are the clear winners.

The second requirement is that the algorithm should minimize error globally. Why is this important? An example comparing region growing and Cezanne illustrates the reason [42]. The fixed neighborhood size common to region growing algorithms is liable to fail wherever the sampling rate is less than optimal. In Figure 4.19a, the large yellow component in the center contains a particularly challenging feature at the top of the column where five distinct planes meet. Since the area is relatively distant from the scanner, the sampling rate is too low to capture the detail. Region growing could not resolve the ambiguity, because the planar predicate is applied only as an input parameter. The predicate is never revisited to provide feedback on the result.

The only two methods that can be said to minimize error globally are Cezanne and RANSAC. Like region-growing, normalized cut uses a planar predicate locally, but not globally. After computing edges weights during initialization, the output labeling is a function of the weights, while the planar predicate is never applied to the component

(a) Region Growing (b) Cezanne

Figure 4.19. Segmentation results: Region Growing vs. Cezanne. Black denotes holes where the scanner failed to register. Nonplanar areas are colored white. Eight other colors are assigned randomly to differentiate the planar regions, and each region is outlined in red. (a) Region growing produces components that are not necessarily planar, such as the large yellow column in the foreground. (b) Cezanne is less likely to be confused by local aberrations.

points in aggreggate. Thus, a component is not guaranteed to satisfy the predicate that was used to generate it.

In order to achieve true global error control, a parametric model is necessary because it is cheap to evaluate. Only when global information is easy to generate, can it be used in an inner loop. In addition, the predicate must be geometric in nature in order to minimize geometric error. The Hough transform estimates model parameters directly, but it fails to minimize the geometric error globally. Since parameter selection takes place in Hough space, the error is also minimized in Hough space. In addition, the Hough transform does not preserve topology. Components that are connected in Hough space are merely coplanar in 3D space, not necessarily connected.

In contrast, Cezanne operates on a graph, like the region growing and normalized cut methods. Components are created only by the removal of edges, which is topology-

preserving. It also estimates model parameters using a minimization function that operates in 3D space. Unfortunately, it is too slow to be used for our input. RANSAC also fulfills three out of four requirements, but it does not preserve topology. Like the Hough transform, the points are analyzed not in 3D space but in a sample space that is independent of the geometry. However, it does estimate model parameters, and the error is minimized globally by the model selection process.

Fast-Cezanne incorporates RANSAC sampling into Cezanne's graph-based segmentation framework. Instead of top-down iteration which sifts through the data for good

---

Algorithm 6 Fast-Cezanne

Input: A triangulation $\mathcal{T}$ of a range image, and an error tolerance ϵ.
Output: A piecewise planar surface $\Psi = \cup_{i=1}^{m} P_i$ where P_i.

1. **Select** a model P_i using RANSAC.
2. **Project** the points of $\mathcal{T}$ to P_i.
3. **Classify** as inliers the points that are closer to P_i than ϵ.
4. **Extract** from $\mathcal{T}$ the largest component of connected inliers.
5. **Repeat** steps 1–4 until the data is exhausted.

---

models, Fast-Cezanne takes advantage of spatial redundancy to make educated guesses about the scale at which good models reside. Once a model is selected, a conforming surface is extracted from the graph without compromising the connectivity of the graph. Figure 4.18 depicts the two parts of the Fast-Cezanne algorithm and how they are related. Central to both methods is an evaluation based on a global geometric fit that compares candidate models in order to select the best one. The combination of fast model selection and the graph structure of Cezanne fulfill our requirements for a segmentation method that produces a topologically correct piecewise planar model of an individual range image. Figure 4.20 shows the result of Fast-Cezanne on the same area depicted in Figure 4.19. We were able to achieve results that compete with Cezanne in quality, but in much faster

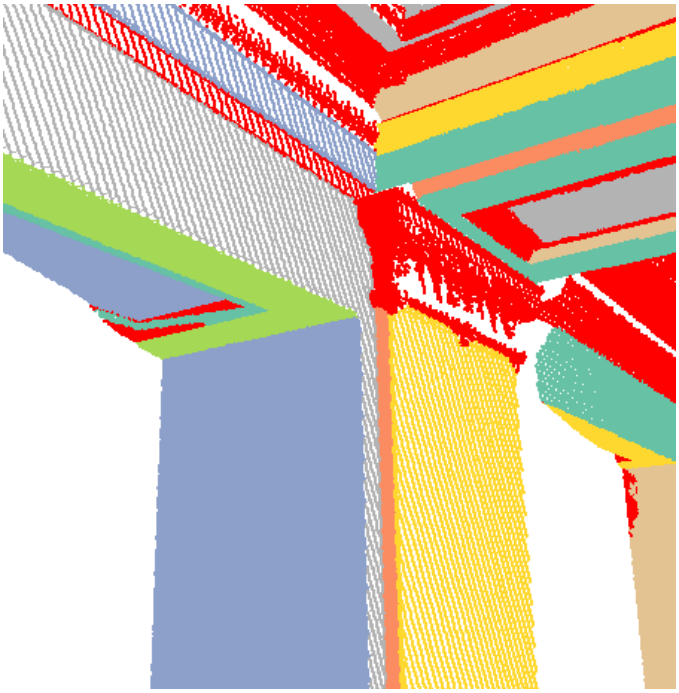


Figure 4.20. Segmentation results for Fast-Cezanne. Like Cezanne, Fast-Cezanne algorithm represents edges and depth discontinuities with high accuracy, and it is able avoid nonplanar areas during planar reconstruction.

time due to the incorporation of RANSAC sampling. In the next chapter, we show how several overlapping segmentations can be merged to produce a piecewise surface model of an entire data set.

Chapter 5

Surface Reconstruction

An important benefit of early segmentation is that it immediately divides the reconstruction problem into smaller parts. Since components are disjoint, we can process them in isolation. More importantly, now each component corresponds to a single model. For an individual scan, the segmented image represents a disjoint union of surface components, as our output requirements demand. However, merging two or more scans destroys the important properties of the model. Many components from registered meshes overlap, so the merged dataset is not a disjoint union. Secondly, merged components lose the Monge property because their parameters spaces are not aligned. Recall from Chapter 3 that we need to fit a parametric function $f(u,v) = \big(u, v, h(u,v)\big)$ which requires the data to be represented as a height field. Overlapping components need to be merged into a common parameter space before we can proceed with modeling.

We used only planar models for segmentation, but now we must reintroduce curved surfaces. Recall from Chapter 4 that we segmented smooth curves into planar pieces. Curved surfaces are approximated differently from scan to scan, leading to complex intersections. Futhermore, Curved surfaces introduce distortion into the parameter space. In some cases, projection to the distorted parameter space causes boundary curves to

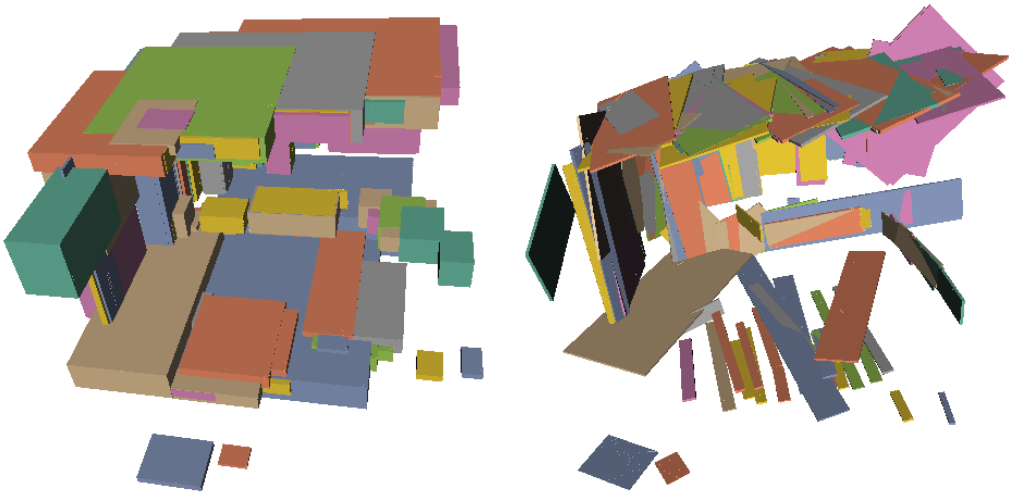
foldover. Our solution is to separate parameter space selection from model selection. The job of parameterization is to embed the surface in $\mathbb{R}^2$. The job of modeling is accomplished by covering the parameter space with a Monge patch. Parameter space selection is covered in Section 5.2. Modeling follows parameterization, and it is always performed in the lower-dimensional space. We present one possible configuration in Section 5.3 which employs two families of surfaces: low-degree polynomials and spline meshes.

5.1 Scan Merging

Once a set of scans is in registration, merging the points is trivial. Since each element is a point, there can be no overlap. Surface components are much harder to merge, but the total number of components is much smaller than the total number of points. Figure 5.1 shows two overlapping scans that have been segmented individually. To find all intersections between the scans, we construct a graph. The vertices of the graph are the surface components, and edges connect those components that intersect. Since the components within an individual scan are disjoint, the graph is bipartite. The intersection algorithm starts with the complete bipartite graph. In three passes, we remove edges from the graph until only true intersections remain. During the first pass, we perform an axis-aligned bounding box intersection test. During the second pass, we perform a tight bounding box intersection test. Figure 5.2 shows the bounding boxes in two overlapping scans. During the final pass, we transform the boundary curves of the two components into a common parameter space and perform a Boolean intersection test. To merge a new scan, we construct a new bipartite graph connecting it to all previously seen components, and repeat the intersection tests on the newly-created edges. This brute-force approach slows progressively as more scans are added, but because the total number of components is small, the impact is likewise small (see Chapter 2).

Figure 5.1. Combining range scans. After segmentation, many components belonging to the same surface may overlap in 3D space. By projecting overlapping components into a common coordinate space, we can merge them using only 2D operations.

Since most of the components in the scene are disjoint, the axis-aligned bounding box test quickly eliminates a majority of the edges in the complete graph. Tight bounding boxes are computed in parameter space and transformed to world coordinates for comparison. Testing against tight bounding boxes eliminates many spurious intersections stemming from the misalignment of the building's orientation from the world coordinate system. In addition, the orientations of tight bounding boxes that are truly coplanar must be close, so we can eliminate some intersections by examining the angle between normal vectors. Once we have isolated two components that are orientated in a similar way, the final intersection test consists of projecting the boundary polygons into a shared parameter space and performing a Boolean intersection test. To find a shared parameter space, a simple heuristic suffices: we choose the space of the larger component, since it



(a) Axis-aligned bounding boxes (b) Tight bounding boxes

Figure 5.2. Bounding boxes. Component intersections are subjected to two types of bounding box tests to limit the number of 2D Boolean intersections that need to be computed during merging.

Algorithm 7 Scan Merging

Input: A registered set of piecewise planar segmentations $\Psi_j = \cup_{i=1}^{n_j} P_i$ where j ranges from 1 to m, and n_j is the number of components in Ψ_j.

Output: A piecewise linear manifold $\Psi = \cup_{i=1}^{n} S_i$ where $S_i \subseteq \cup\Psi_j$ and $n \leq \sum_j n_j$.

1. Initialize $\Psi = \Psi_1$ and a graph G with no edges.
2. **for** each Ψ_j from 2 to m **do**
3. Add a vertex to G for each component in Ψ_j.
4. Compute $\Psi_j \cap \Psi$ and add an edge to G for every bounding box intersection.
5. Test tight bounding boxes, and remove edges that fail to intersect.
6. Select a shared parameter space for each edge.
7. Test 2D boundary intersections, and remove edges that fail to intersect.
8. The connected components of G are the S_i to be returned.

---

has more support in the data.

Shared parameter spaces now exist for every edge in the graph. This construction is known as a *piecewise linear manifold*. Each shared parameter space acts as a *coordinate chart* which is locally Euclidean. Transforming points from from one component to another through the coordinate chart is equivalent to a *transition function*. The collection of charts forms an *atlas*. Thus, the merged scans exhibit a powerful mathematical structure that is compact in comparison to the input data. It still carries redundant information in the form overlaps, however, and it also carries the overhead of the transition functions. Further refinement is possible. Combining the individual components into a single object would remove the transition functions from the model. In the next section, we will compute shared parameter spaces not just for pairwise intersections, but for connected components in the intersection graph.

5.2 Parameterization

If an atlas contains a collection of charts belonging to a surface that is actually flat, parameterization to 2D is a simple matter, and it suffices to choose as a shared parameter space the plane of the largest component, as we did in the pairwise case. All of the charts in the atlas are then transformed into the shared parameter space, and the boundary of the output component is the Boolean union of all of the charts. If the surface is curved, however, a planar parameter space may not be adequate to represent the surface without foldover. In that case, we must reduce distortion using a curved parameter space. We accomplish this task by returning to the source data: the input points. By combining the points from all components in the planar shared parameter space, we fit a curved shape that has a well-defined mapping to $\mathbb{R}^2$.

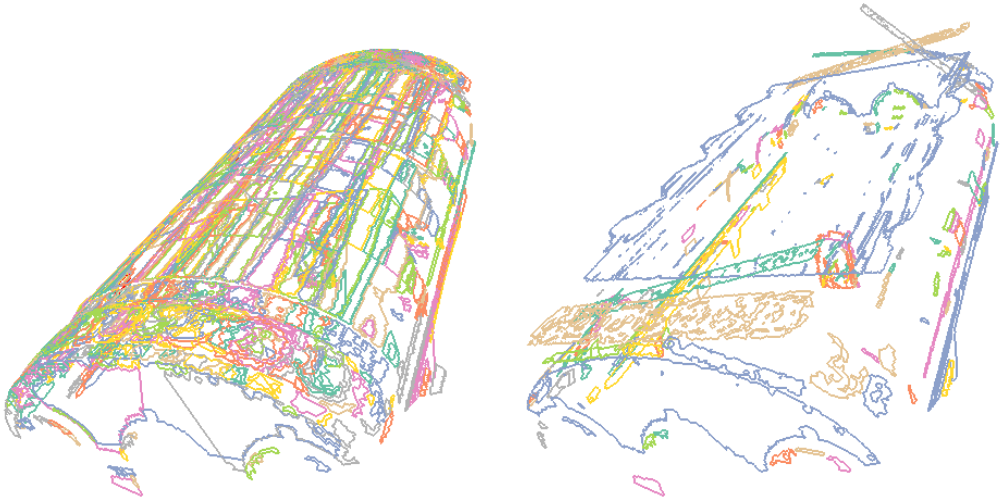
We limit curved parameter spaces to geometric primitives such as cylinders, spheres,

Algorithm 8 Parameterization

Input: A piecewise linear manifold $\Psi = \cup_{i=1}^{n} S_i$.
Output: A shared parameter space Π_i for each $S_i \subseteq \cup\Psi_j$.

1. Arrange the parameter space models Π^k in increasing order of complexity.
2. **for** each S_i **do**
3. **for** each Π^k **do**
4. Fit an instance of Π^k to all of the points in S_i.
5. **for** each plane $P_j \in S_i$ **do**
6. Project P_j into Π^k.
7. If no P_j folds over, accept $\Pi_i = \Pi^k$.

and cones because they have very simple form. Figure 5.3 depicts the segmentation and merging of components generated by a large curved surface. The surface is roughly cylindrical, but it is not a perfect circle because the center of the arch is flatter than the sides. The planar parameter space shown in Figure 5.3b exhibits an increasing amount of distortion moving from the center to the sides. As a result, many components along the sides cannot be projected into the parameter space without folding over. Figure 5.4 shows an example of two such components. By fitting a cylinder to the points, we can easily flatten them into a parameter space in which there is much less distortion. Note that the cylinder does not need to fit closely in the parameterization phase. The main purpose is to reduce distortion prior to modeling.



(a) Merged components (b) Flattened components

Figure 5.3. Merged and flattened components. (a) Merged components form atlases, with charts that are composed of pairwise intersections between polygons in a shared parameter space. (b) Flattening more than two connected components into a shared parameter space precedes surface fitting to the composite component.

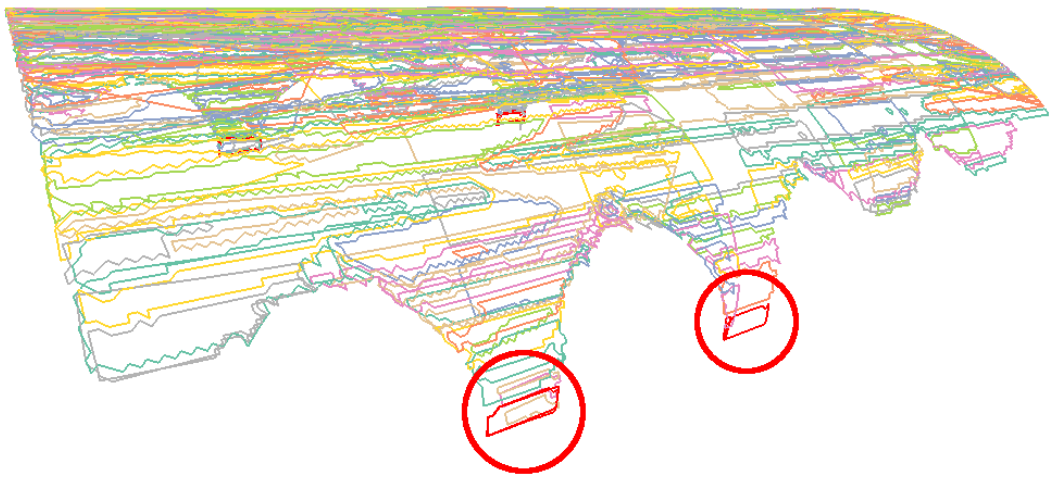
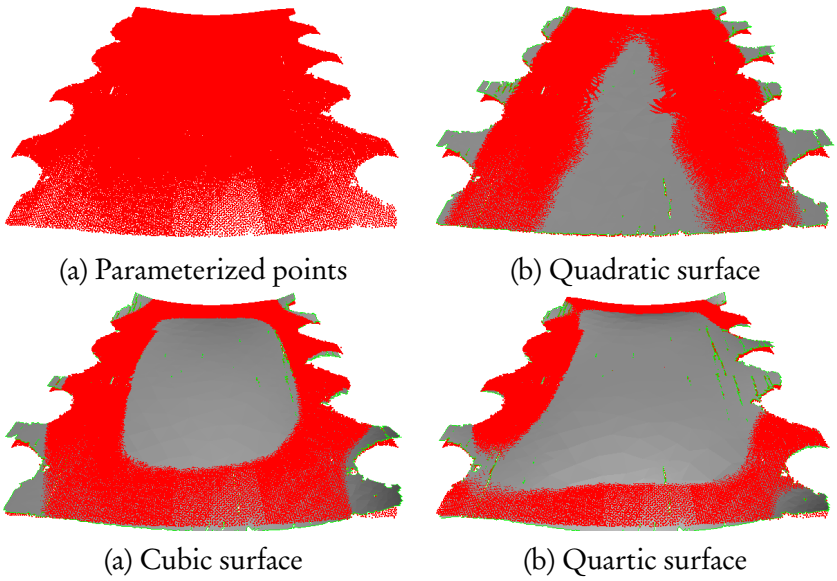


Figure 5.4. Merging with invalid boundaries flagged. The shape of a component sometimes precludes projection to a flat parameter space without spoiling the topology by folding over. We detect surface self-intersection by testing boundary for curve self-intersection.



(a) Parameterized points (b) Quadratic surface

(a) Cubic surface (b) Quartic surface

Figure 5.5. Polynomial fitting. Polynomials of increasing degree fitted to the ceiling of Grand Central Terminal in a cylindrical parameter space. The points are colored red, while the fitted surface is rendered in gray. Although the training error decreases as we raise the polynomial degree, the higher-degree polynomials appear to fit the surface worse than the quadratic model.

5.3 Surface Modeling

Parameterization ensures that each component can be represented by a Monge patch. Therefore, many options exists that can be used as modeling functions. How do we choose between them? Figure 5.5 shows progressive polynomial fits to the ceiling of Grand Central Terminal. For many surfaces found in urban scenes, a suitable low-degree polynomial can easily be found this way. However, the ceiling is too complex to be fit adequately by a polynomial. In the quartic case, for instance, huge deviations from the surface appear while the error continues to decrease, which indicates that overfitting has occurred. When the surface is free-form, we switch to the B-spline surface, a representation that can handle an arbitrarily large number of degrees of freedom. Thus, by arrang-

ing model families in ascending order of complexity, we can select an adequate model heuristically while limiting the number of extraneous fits required.

We approach parameter space modeling as a problem of scattered data interpolation [23]. The scattered data consists of a set of height values z_c defined at arbitrary positions $\{(x_c, y_c)\}$ in the plane. We seek a function of two variables $f(x, y)$ that takes the value z_c at (x_c, y_c). In general, we cannot fit all of the scattered points exactly. We can, however, search for the function with the smallest error.

In a typical urban scene, most of the surfaces are planes. Therefore, we begin with low-degree polynomials. If a plane fits, then we need to store only four parameters in the

---

Algorithm 9 Polynomial Approximation

Input: Scattered data $P = \{(x_c, y_c, z_c)\}_{c=1}^{N}$ and the degree d of the polynomial to be fit.
Output: The d-degree least squares polynomial Φ.

1. **for** each data point $(x_c, y_c, z_c) \in P$ **do**
2. Form the vectors $\mathbf{u} = (x_c^d, x_c^{d-1}, \ldots, x_c^1, 1)$ and $\mathbf{v} = (y_c^d, y_c^{d-1}, \ldots, y_c^1, 1)$.
3. Compute the outer product $\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^T$.
4. Copy the result to row c of a matrix $\mathbf{A}$ with n rows and $(d+1)^2$ columns.
5. Form the right hand side $\mathbf{b} = (z_1, \ldots, z_n)$.
6. Minimize $||\mathbf{b} - \mathbf{A}\Phi||_2$.

---

output. Next is the quadratic polynomial, which is appropriate for surfaces that have a shallow curvature in one direction. The plane computed in the first iteration becomes the domain for the quadratic fit. Let Ω be a rectangular domain in the xy-plane, and let $P = \{(x_c, y_c, z_c)\}_{c=1}^{N}$ be a set of scattered points with $(x_c, y_c) \in \Omega$. To fit a d-degree polynomial surface to the points, we need to solve an equation having the form $\mathbf{A}\Phi = \mathbf{b}$. The unknown solution vector Φ contains the $(d+1)^2$ coefficients of the minimum-norm least-squares surface. The matrix $\mathbf{A}$ and the right hand side vector $\mathbf{b}$ are both derived from the input points. $\mathbf{A}$ is a Vandermonde matrix obtained from the xy components of

each point by computing the outer product $\sum_{i=1}^{d}\sum_{j=1}^{d} x^i y^j$ and arranging the terms row-wise. For example, the quadratic case when $d = 2$ has nine coefficients in its expansion $Ax^2y^2+Bxy^2+y^2+x^2y+xy+y+x^2+x+1$. The z coordinates populate the right hand side $\mathbf{b}$, so that the residuals are given by $\mathbf{r} = \mathbf{b} - \mathbf{A}\Phi$. The sum of the squared residuals is minimized by orthogonal decomposition using the singular value decomposition (SVD).

For spline fitting, we use the Multilevel B-spline Algorithm (MBA) of Lee *et al*. [23] This method of scattered data interpolation uses a hierarchy of control lattices for fast surface reconstruction. MBA is controlled by a pair of shape parameters that allow the domain of the reconstruction to accommodate surfaces with different aspect ratios. A integer height parameter specifies the number of the levels of hierarchy to employ. Increasing height values increases the number of degrees of freedom exponentially. The aspect ratio can be determined by analyzing the shape of the parameter space. Tuning of the height parameter is part of the model selection process described in Chapter 2.

Let $\Omega = \{(x,y) \mid 0 \le x < m, 0 \le y < n\}$ be a rectangular domain in the xy-plane, and let $P = \{(x_c, y_c, z_c)\}$ be a set of scattered points with $(x_c, y_c) \in \Omega$. To approximate P, we overlay a lattice Φ of control points defining a uniform bicubic B-spline function f such that f takes on the value z_c at (x_c, y_c). The advantage of B-spline approximation is that the value of f at can be determined by examining just sixteen control points ϕ_{kl} in a 4×4 neighborhood around (x_c, y_c):

$$z_c = \sum_{k=0}^{3}\sum_{l=0}^{3} w_{kl}\phi_{kl}, \tag{5.1}$$

where w_{kl} is a blending of B-spline basis functions. We minimize Equation 5.1 in a least-squares sense by computing the deviation of f as

$$\phi_{kl} = \frac{w_{kl} z_c}{\sum_{a=0}^{3}\sum_{b=0}^{3} w_{ab}^2}. \tag{5.2}$$

To solve for a control point ϕ_{ij}, we blend contributions from each neighboring data point (x_c, y_c) by minimizing the error $e(\phi_{ij}) = \sum_c (w_c \phi_{ij} - w_c \phi_c)^2$. Differentiation with respect to ϕ_{ij} yields

$$\phi_{ij} = \frac{\sum_c w_c^2 \phi_c}{\sum_c w_c^2}. \tag{5.3}$$

A fast algorithm to compute the ϕ_{ij}'s accumulates the contributions of the numerator δ_{ij} and denominator ω_{ij} of Equation 5.3 in one pass through the data points. Once

Algorithm 10 B-Spline Approximation

Input: Scattered data $P = \{(x_c, y_c, z_c)\}$.
Output: A control lattice $\Phi = \{\phi_{ij}\}$.

1. **for** each data point $(x_c, y_c, z_c) \in P$ **do**
2. Center a window over the neighborhood containing (x_c, y_c).
3. Compute w_{kl}'s and $\sum_{a=0}^{3} \sum_{b=0}^{3} w_{ab}^2$.
4. **for** $k, l \leftarrow 0$ **to** 3 **do**
5. Compute ϕ_{kl} using Equation 5.2.
6. Accumulate $w_{kl}^2 \phi_{kl}$ in $\delta_{(i+k)(j+l)}$ and w_{kl}^2 in $\omega_{(i+k)(j+l)}$.
7. Compute $\phi_{ij} = \delta_{ij} / \omega_{ij}$ for each control point.

all contributions have been accumulated, the final value of each control point ϕ_{ij} is the quotient δ_{ij}/ω_{ij}. The density of the control lattice Φ determines the shape of the approximation function f. If the lattice is coarse, many points fall in each neighborhood, resulting in a smooth function. As the lattice becomes finer, the approximation gains accuracy, until the shrinking neighborhood size leads to overfitting of local peaks.

Even though the number of degrees of freedom and, thus, the number of parameters, can be much larger than it is with polynomial models, there is still enormous space savings associated with spline surfaces in comparison to the hundreds of thousands of sample points that may fall on a single surface in the original scan data. In fact, the collection of surface models plus boundaries generated by the modeling step is the most compact rep-

Algorithm 11 Multilevel B-Spline Approximation

Input: Scattered data $P = \{(x_c, y_c, z_c)\}$.
Output: A control lattice hierarchy $\Phi_0, \Phi_1, \ldots, \Phi_h$.

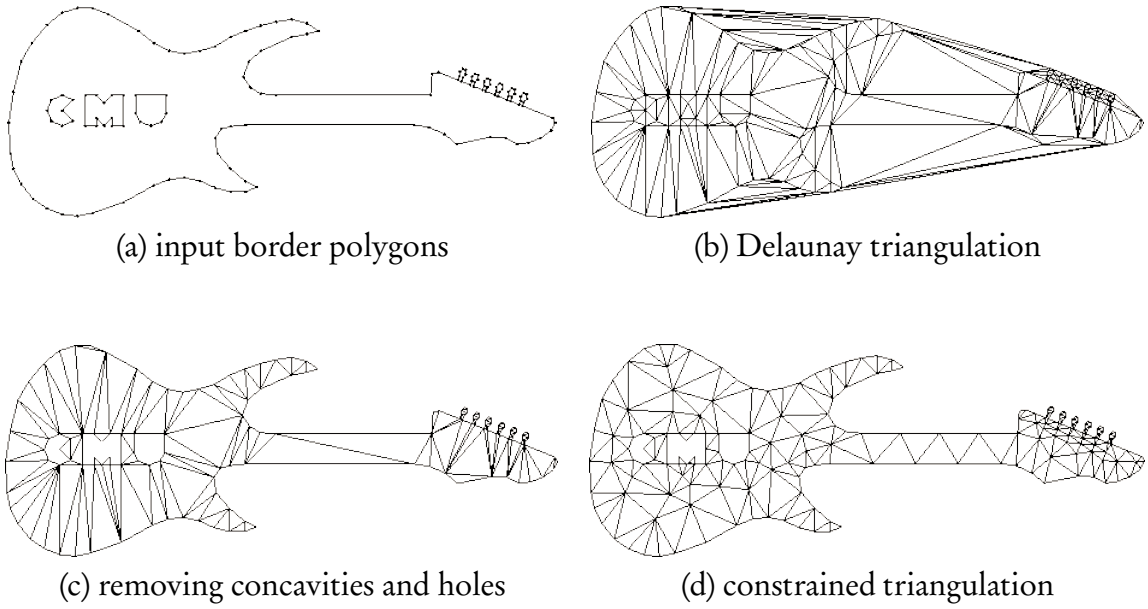
1. **for** $k \leftarrow 0$ **to** h **do**
2. Let $P_k = \{(x_c, y_c, \Delta^k z_c)\}$
3. Compute a B-spline approximation Φ_k from P_k.
4. Compute $\Delta^{k+1} z_c = \Delta^k z_c - f_k(x_c, y_c)$ for each point.

resentation of the range data in our pipeline. If we were to store this representation along with a residual value for each data point, we would have an effective method for lossless compression. We will build such a representation in Chapter 2, and use it to compare our reconstruction with the output of other alternatives. First, we describe how to output our model using a more standard representation, the triangle mesh.

5.4 Triangulation

In order to make our model as general as possible, we created an algorithm to compute a triangulation from the trimmed surface model. Since graphics graphics acceleration hardware is tailored for triangle meshes, this output is useful for visualization. In addition, we produce well-formed triangles that are suitable for use by numerical techniques such as the finite element method. As in modeling, we perform triangulation in parameter space. Thus, distortion is a risk that is best managed by selecting an appropriate parameter space.

Once we have selected a suitable low-distortion parameter space and projected the component boundary into it, we need to fill the boundary with triangles. We use constrained Delaunay triangulation to accomplish this task. Figure 5.6 depicts the process of triangulating the inputs points, and then refining the triangulation until all of the tri-



(a) input border polygons (b) Delaunay triangulation

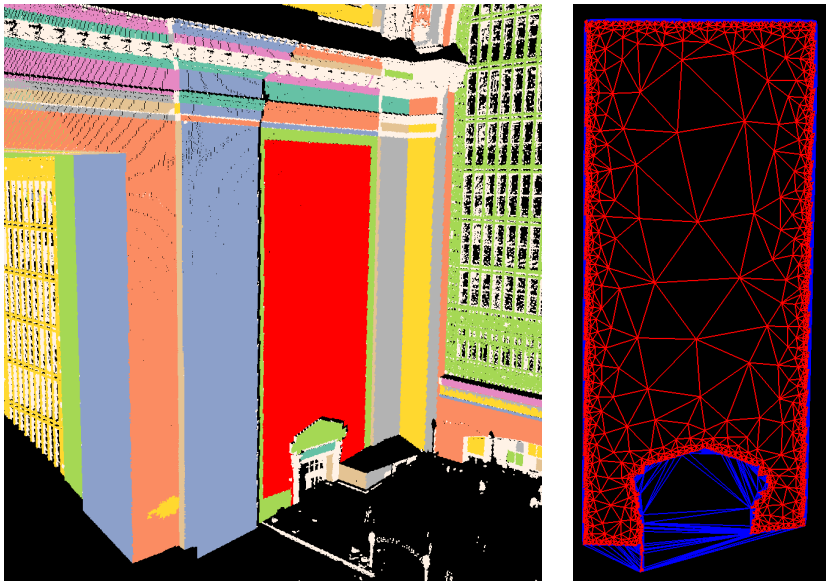
(c) removing concavities and holes (d) constrained triangulation

Figure 5.6. Constrained Delaunay triangulation. (a) Border polygon, with holes. (b) Delaunay triangulation of the convex hull. (c) A simple point-in-polygon test can be used to remove concavities and holes. (d) Constraints are applied iteratively, modifying the mesh until all triangles are conforming.[1]

angles are well-formed. The boundary curves are fixed, and are guaranteed to reappear in the output. Therefore, it is easy to knock out the holes and trim concavities in the convex hull. The user can vary the minimal angle enforced during triangulation, and a maximum area can also be imposed on any triangle in the output.

Once the triangles have been deposited in the parameter space, we then lift them to 3D using the surface function computed in section 5.3. First, we evaluate $f(x,y)$ at each vertex of the constrained Delaunay triangulation to generate a z-coordinate, then bring the 3D point (x,y,z) to world space using the inverse parameter space transformation.

[1]Images courtesy of Jonathan R. Shewchuk.



(a) 38,194 vertices (b) 2,550 vertices

Figure 5.7. Triangulation. (a) A large planar component. (b) A mesh approximating the planar component. The approximation uses less than 7% of the number vertices, but still models the border exactly.

5.5 Complexity

As we derived in Section 4.5, the segmentation step has an asymptotic time complexity of $O(\frac{1}{P(n)}S)$ per component, where $P(n)$ is the probability of finding a component of size n and S is the cost of scoring the component [30]. Merging is dominated by the boundary intersection test, which is performed only when a pair of components passes both the regular bounding box and tight bounding box tests. The running time thus depends on the probability $P(i,j)$ that a component C_i from scan i intersects a component C_j from scan j. We merge scans incrementally, always performing pairwise intersection tests, and accumulating intersections in a graph. For the first pair, the running time is $P(1,2)N_2$, where N_2 is the number of components in scan 2. When we add a third scan, it must be

tested against the first two, and the running time is

$$T = P(1,2)N_2 + \big(P(1,3)N_3 + P(2,3)N_3\big) \tag{5.4}$$

$$= P(1,2)N_2 + \big(P(1,3) + P(2,3)\big)N_3. \tag{5.5}$$

Let k be the number of scans in the data set, and let N be the average number of components per scan. Then, the running time required to merge all scans is

$$T = \sum_{j=2}^{k} \sum_{i=1}^{j} P(i,j)N. \tag{5.6}$$

To bound the sum, note that, although the complete bipartite graph connecting scan i and j has $N_i N_j$ edges, the number of intersections should be much smaller, because the manifold properties of the geometry limit the amount of overlapping that can occur. Intersections are local in the sense that the number of intersections per component cannot exceed a small constant $c \ll N$. Therefore, if we draw one edge from scan i and one edge from scan j, the probability of finding an intersection is

$$p = \frac{c}{N^2}. \tag{5.7}$$

Substituting p for $P(i,j)$ in Equation 5.6, we have

$$\begin{aligned} T &= pN + 2pN + \ldots + kpN && (5.8)\\ &= (1+2+\ldots+k)pN && (5.9)\\ &= \left(\sum_{i=1}^{k} i\right)pN && (5.10)\\ &= \frac{k(1+k)}{2}pN && (5.11)\\ &= O(k^2N). && (5.12) \end{aligned}$$

The space complexity is the same, because the size of the output graph is proportional to the probability of finding an intersection. The expected value for number of intersections in the worst-case is

$$\begin{aligned} S &= p(kN)(k-1) && (5.13)\\ &= pk^2N - pkN && (5.14)\\ &= O(k^2N). && (5.15) \end{aligned}$$

In Equation 5.13, the total number of components in the data set is kN, which is the number of scans multiplied by the average number of components. Each component has worst-case probability p of intersecting a component in another scan, and there are $(k-1)$ scans to be tested. Since all intersections are pairwise, the extra memory required to represent two 2D polygons in memory is bounded by the number of vertices in the boundary of the largest component.

Once merging is complete, composite components must be reparameterized and re-modeled. Parameterization is accomplished by solving a least-squares problem using the singular value decomposition (SVD). Our implementation uses the divide-and-conquer

LAPACK routine `DGESDD` to solve the eigenvalue problem. For each component, we need to test at most a small number of surface types such as planes, cylinders, and spheres. Therefore, the running time is linear in the number of components. Modeling, however, may need to test many levels of detail for a number of function families before finding an appropriate fit. Without loss of generality, let us assume that all of the models come from the same family $\Pi = \{f_h\}_{h=1}^{\infty}$ where h is the level of detail. For every component C, there exists an optimal h where the test error reaches its minimum. The time complexity of model selection is determined by the expected value of the optimal level of detail, because this determines the number of surface fits that need to be performed on average until an optimal model is found. The overall time complexity T is equal to the expected number of surface fits required per component, multiplied by the total number of components:

$$T = \mathrm{E}[h]kN. \tag{5.16}$$

Different families have different running times that depend in general on the number of points in the component n and the level of detail h. Like parameter spaces, polynomial surfaces are computed using SVD, and thus benefit from logarithmic time complexity. Spline fitting using the MBA algorithm runs in $O\big(n + s(h)\big)$ time, where the mesh size $s(h)$ is exponential in the level of detail [23]. As for space complexity, we only retain one model per component, so the it is linear in the number of components.

Finally, we also generate an output triangle mesh. Generating the output mesh depends on the constrained Delaunay triangulation, which requires $O(n \log n)$ time to run [33].

5.6 Results

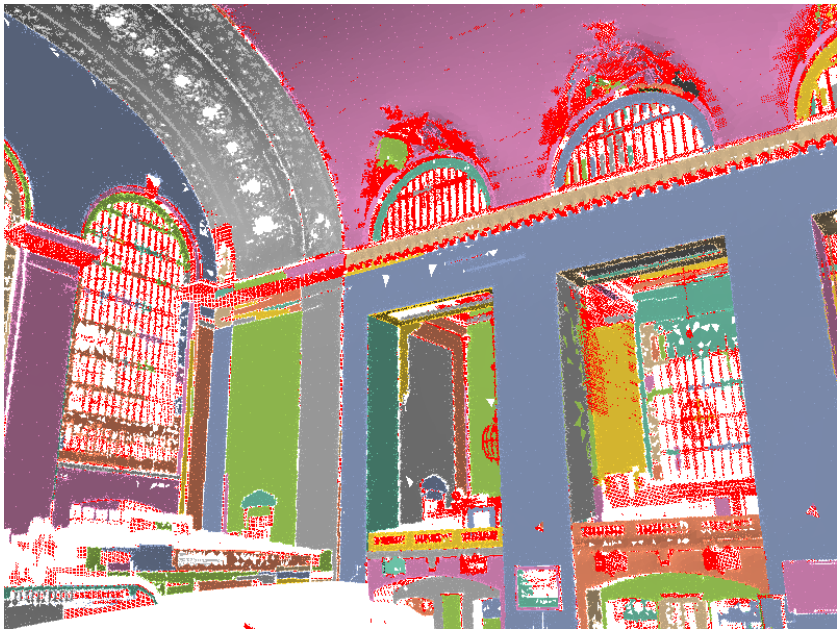
Results from our reconstruction algorithm appear in Figure 5.1. We ran our reconstruction algorithm on the same five data sets analyzed in Section 2.3. All computations were performed on a 3MHz Intel Xeon 5160. Final models had an average error below the

Name	Scans	Components	Segmentation (secs/scan)	Modeling (secs)	Meshing (secs)	Nonplanar (%)	Error (mm)
GC	44	1,314	183 ± 90	1,446	117	13	3 ± 7
SH	20	1,213	270 ± 153	125	7	28	4 ± 6
GH	5	230	209 ± 172	29	2	28	4 ± 7
HN	3	334	334 ± 158	1	4	24	2 ± 4
CU	23	1,373	283 ± 245	5	29	21	4 ± 6

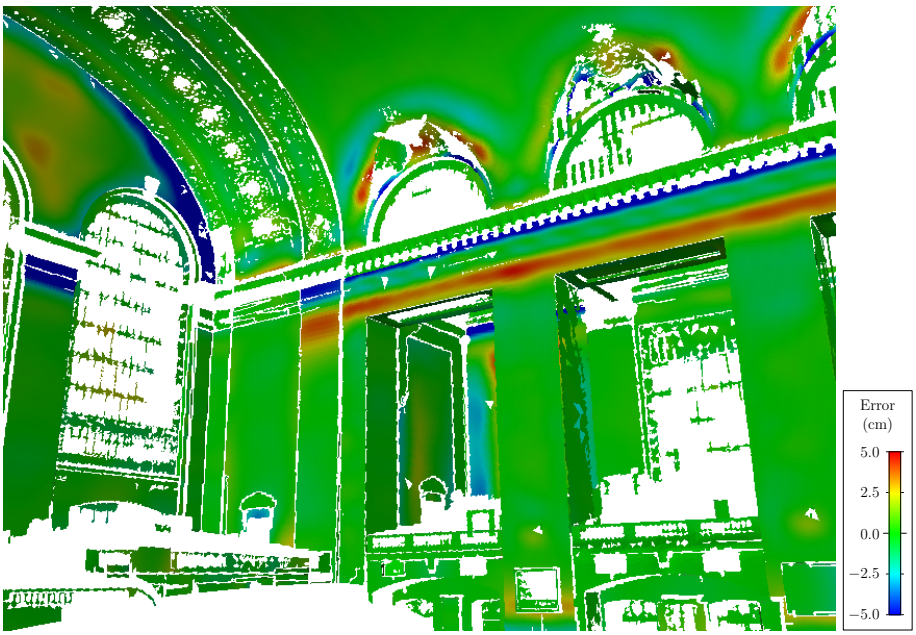
Table 5.1: Surface reconstruction results on a 3MHz Intel Xeon 5160. The large variance in segmentation times reflects large differences in content from scan to scan in a single data set. Likewise, content dictates modeling time. For a scene containing large curved surfaces, such as the GC data set, a larger, more complex model selection and fitting problem needs to be solved. On the other hand, if the scene is full of flat surfaces, such as the CU data set, modeling can be very fast. Similar small average error results indicate that the user-defined error tolerances provide good control over the error characteristics of the output approximation.

manufacturer's estimate of 6mm for the acquisition device. Large variations in segmentation and modeling times reflects sensitivity to scene content.

Appearing on the following pages are snapshots of modeling results on actual data. The output surface components are shown alongside the nonplanar points—colored red—that remain after surface reconstruction. On average, we can expect between 70% and 90% of the points in the raw data to be consumed during reconstruction. Accompanying the output model, we include an error image created by fitting a spline mesh surface to the residuals in parameter space and creating a texture map. False color indicates the average distance of the sample data above or below the surface at every point. Many imperfections in the surface and some scanner artifacts appear in the texture images, indicating effective abstraction that separates high-frequency deviations from the geometry.

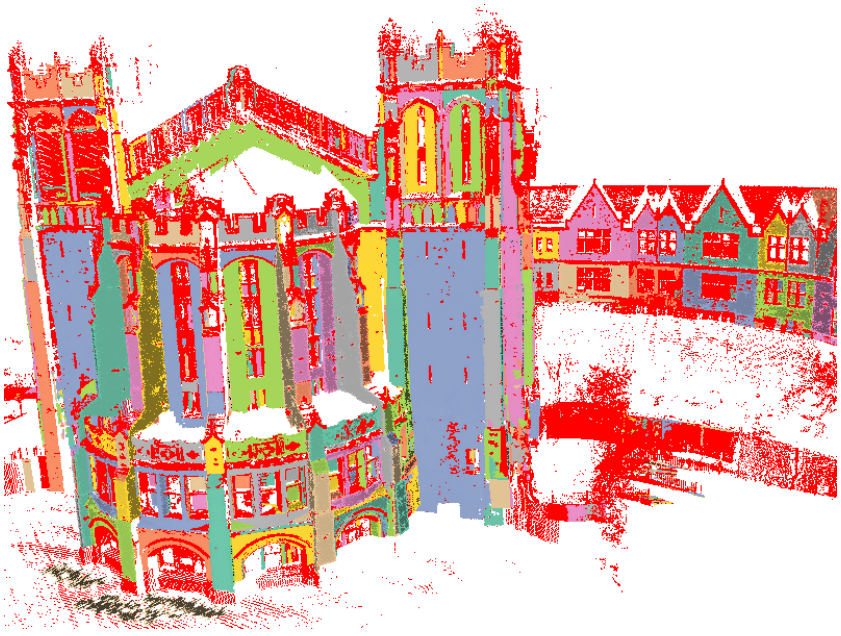


(a) Output model

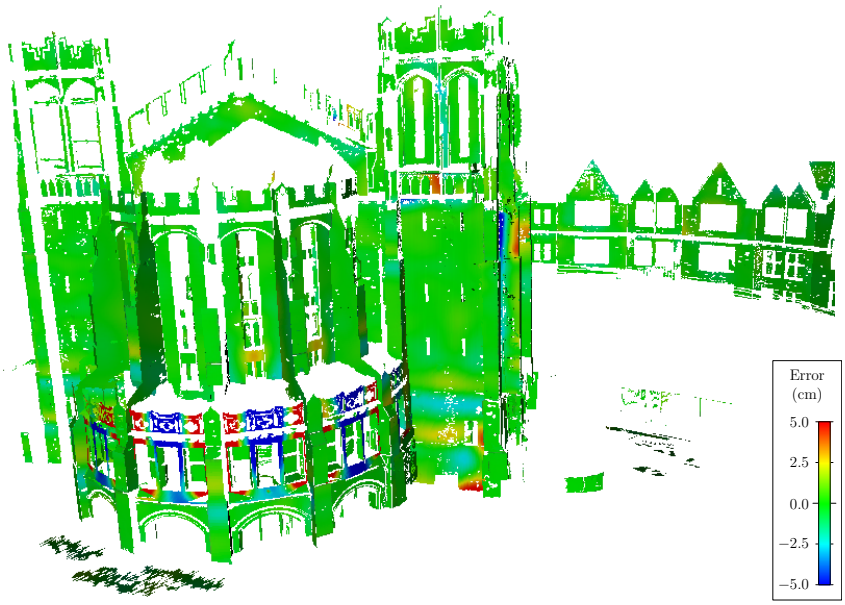


(b) Error

Figure 5.8. Grand Central Terminal, New York.

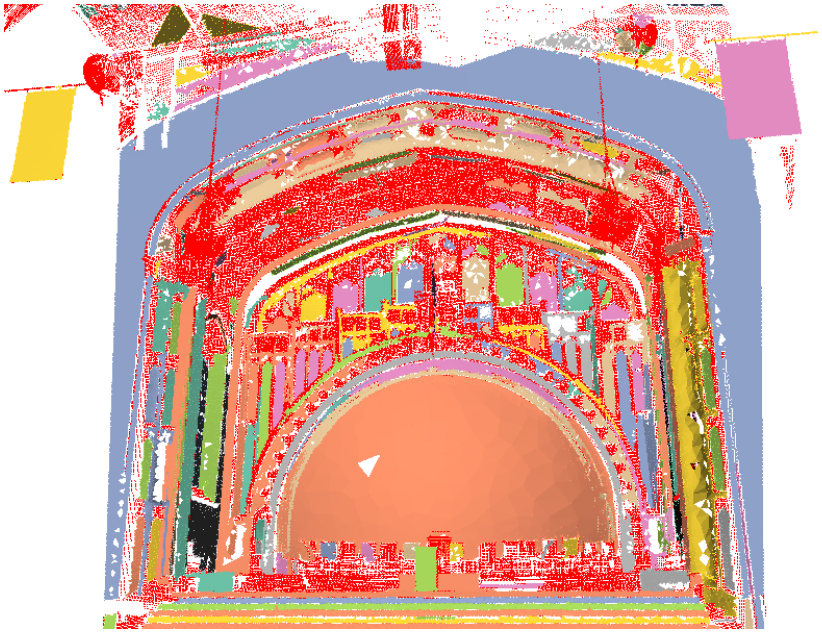


(a) Output mesh

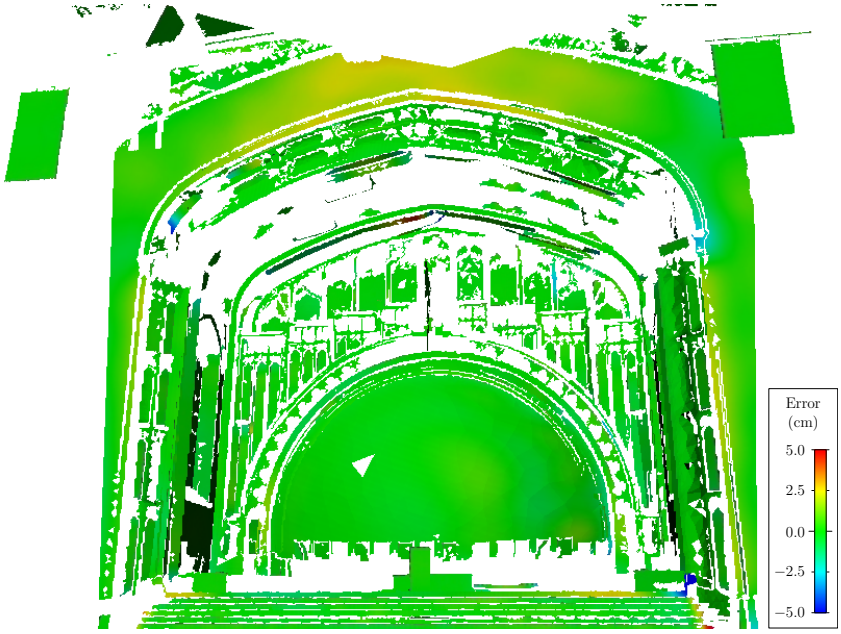


(b) Error

Figure 5.9. Shepard Hall, CCNY.

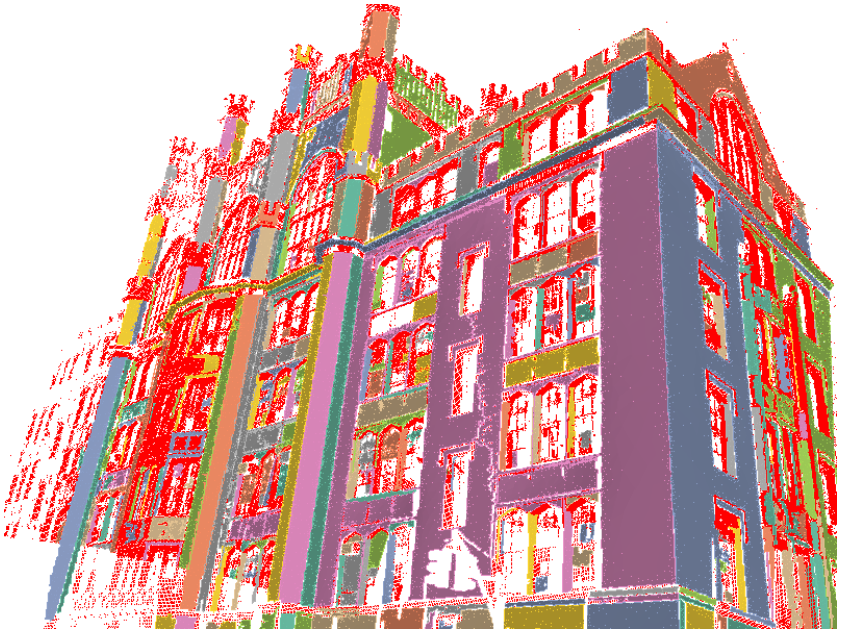


(a) Output mesh

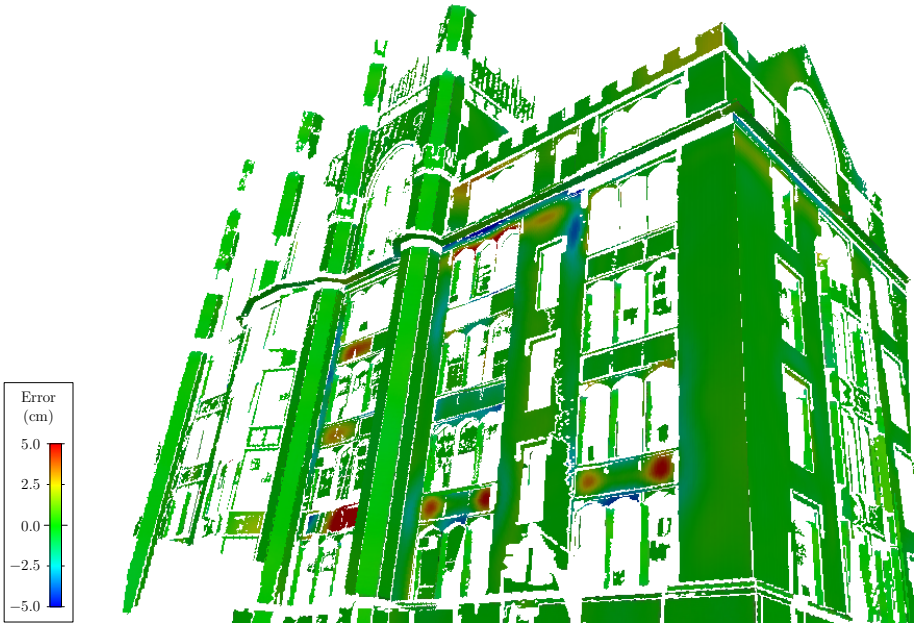


(b) Error

Figure 5.10. Great Hall, CCNY.



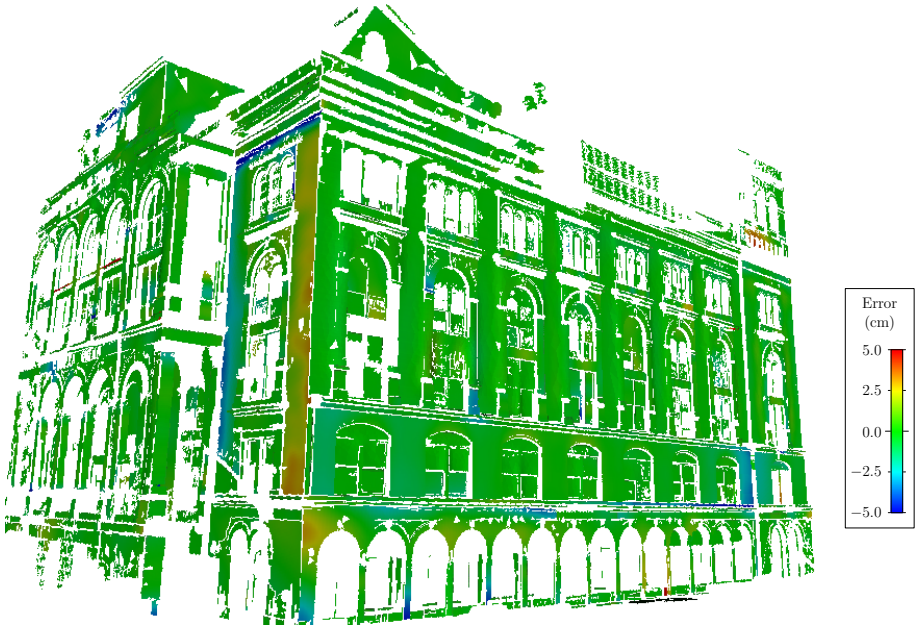
(a) Output mesh



(b) Error

Figure 5.11. Hunter College.

(a) Output mesh



(b) Error

Figure 5.12. Cooper Union.

5.7 Discussion

Our algorithm is designed for scenes that have a large number of disjoint surfaces, each conforming to a simple geometric shape that can be approximated parametrically. If the surfaces are not planes and simple curved shapes, it may not be possible to parameterize them using height fields, thus causing the reconstruction algorithm to fail. Objects such as statues and sculptures may be both highly complex and smooth at the same time, resulting in a merged component that folds over in every possible parameter space. In this case, our method can at least isolate the object before applying a traditional modeling algorithm that is not coupled to a segmentation.

Another point of failure stems from the tendency of the segmentation algorithm to overfit. Although merging of overlapping components removes many seams, scanning constraints may result in insufficient coverage in some areas of the scene. If such areas contain curved surfaces which are poorly approximated by planes during segmentation, they will also be poorly approximated in the final reconstruction. Since we segment each image separately, our algorithm may fail to recover surfaces that are adequately sampled only after the scans have been merged. If a given surface is seen obliquely from several vantage points, there may be no one scan in which the surface is detected. In that case, our algorithm may miss the surface even if the combination of scans would yield a dense enough sampling.

Assuming that the scene does contain parametric surfaces and the sampling rate is sufficient everywhere, how well does our algorithm perform on actual data? Our results must be analyzed in a different way from BPA meshes, because different components have different shapes and error characteristics. Instead, we must consider specific attributes of the surfaces in order to group them together for comparison. For example, we selected a large, flat surface from each of the data sets. Generalization error is depicted in Fig-

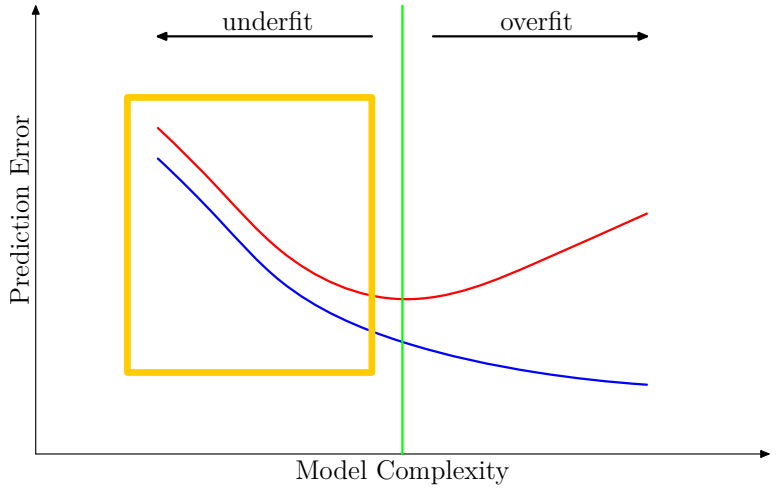


Figure 5.13. To the left of the optimal test error, an increase in model complexity lowers both training and test errors. The optimal prediction capability of the model is achieved when the test error reaches a minimum point.

ure 5.14a. Flat surfaces containing 30,000 to 70,000 points, common in the data sets we encountered, are instructive examples. Often, a low-degree polynomial cannot be used to fit a large component within reasonable error tolerances even if the surface is flat. In that case, we use a spline mesh, which adapts easily to slight bowing and warping. Figure 5.14 shows that all five surfaces have optimal spline mesh representations between level 8 and level 11. Some variation in content can be observed as the optimal generalization error ranges from 4mm to 1cm. The differently-shaped curves also indicate surface features resolving at different scales.

Now we know what it would take to represent the model at full resolution. Is it worth it for a surface that we have already identified to be flat? Recall the generalization error graph reproduced in Figure 5.13. We are looking for both the test error and the bit rate to decrease and the model complexity increases. The results are shown in Figure 5.14. For flat surfaces, the test error does indeed decrease. However, the bit rate increases, indicating that models of increasing complexity do not increase the level a ab-

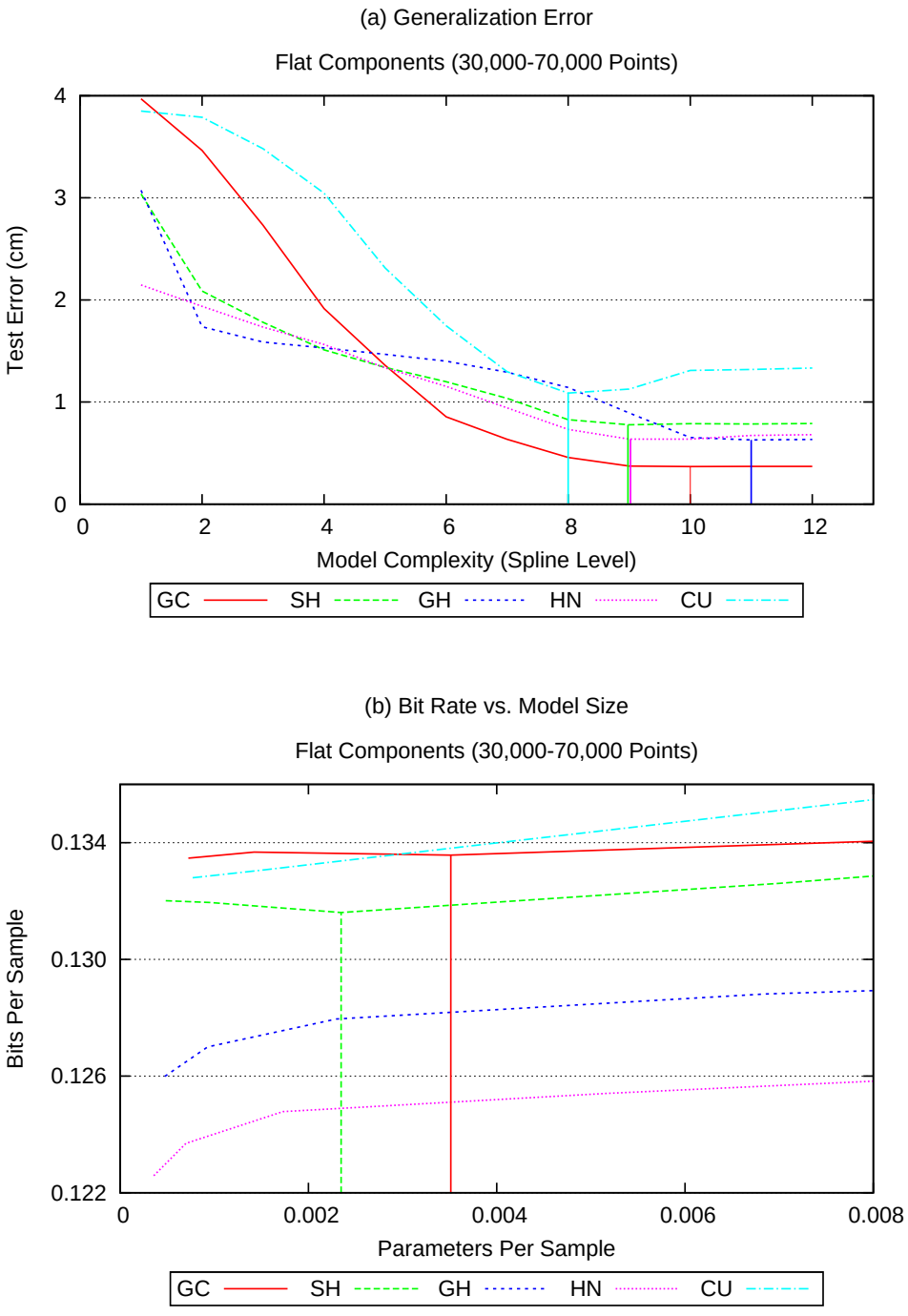


Figure 5.14. Generalization and bit rate analysis for flat components. (a) Optimal generalization error can be achieved for all components using spline meshes of level abetween 8 and 11. (b) Bit rate, on the other hand, does not track the generalization error. Flat surfaces contain more detail than we need.

straction achieved. The reason is unsurprising: pushing the error down to sub-centimeter levels is overkill. In only two cases was it worthwhile to use a spline level greater than the minimum. What if we prefer to represent all such surfaces using a plane? The analysis indicates that we should set the error tolerance to 3–4cm. At 3cm, results over all components appear in Figure 5.1.

Once again, we see evidence of content-based differences among data sets. Note, first of all, the close error numbers across all scans. The consistent error results indicates effective control that is responsive to user-defined tolerances. However, there is a huge discrepancy in modeling time between the GC and CU data sets for a similar number of output components. This variation in running time is due to differences in the scene geometry. The exterior of Cooper Union contains no curved surfaces, and among flat surfaces, few of them are very large. Since small models suffice for small, flat components, the modeling stage is fast. In contrast, the interior of Grand Central Terminal is dominated by the ceiling, a single large curved surface. To model the ceiling, we must attempt many inadequate fits before a surface can be found that satisfies the given error tolerance. In particular, the surface fitting sequence runs to completion in the planar parameter space before advancing to the cylindrical. However, since it is the largest, most important geometric feature in the scene, a large model may be appropriate. Generalization error for the ceiling is shown in Figure 5.15, along with results for the spherical mural in the Great Hall, a similar dominant curved feature. For both surfaces, the optimal model identified by the generalization error is the spline mesh at level nine. The optimal bit rate, however, underestimates the optimal generalization error to a degree determined by the size of the data set. At 1.9 million points, the ceiling is best modeled at level 8. We sacrifice just one millimeter of accuracy as a result. For the mural, however, we lose 3.7cm of accuracy to represent $55,000$ points at the optimal bit rate. Since this is a dominant feature, however, we might be inclined to sacrifice some efficiency in favor of

a consistent range of fidelity across the output model.

Plots of generalization error and bit rate for curved surfaces appear in Figure 5.15. It shows the expected correlation for models below optimal complexity. When plotted against model size, increasing the number of parameters improves the bit rate for both components. In contrast to the BPA+MS result, which is initially overfit and remains overfit after simplification, our approximation approaches optimal from the underfit side. This enables us to control the level of detail in the output, since a range of simple-to-complex instances of a model is generated easily during model selection, any of which can serve as an acceptable, topologically correct substitute for the highest-resolution result if the application is sensitive to size.

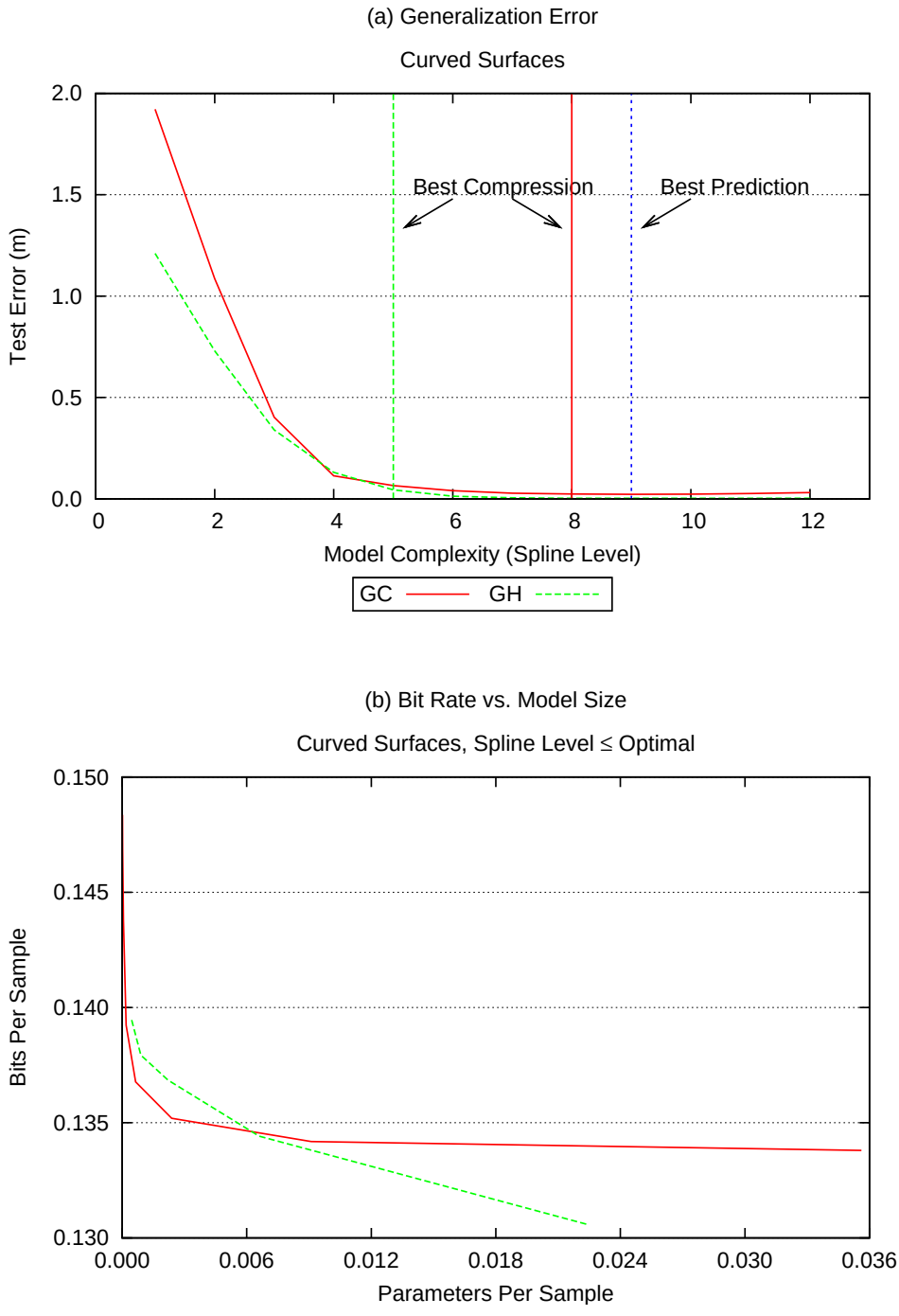


Figure 5.15. Generalization error and bit rate for curved components. (a) An increase in model complexity decreases the test error, so the model is underfit. (b) Adding model parameters improves the bit rate, so the model continues to get more effective as we approach the optimal model complexity.

Chapter 6

Conclusion

Geometric modeling from raw sensor data poses a chicken-and-egg problem requiring us to determine the type of model and to search for instances of the model at the same time. Usually, we need to bring prior information to bear in order to bootstrap the modeling process. For example, range data of urban scenes contains simple models such as planes and polynomials, so we can limit our search to parametric surfaces. But the number and location of model instances must be determined before we can begin to fit parameters to the data. We have developed an algorithm for range data that combines planar segmentation, scan merging, and surface fitting to produce a model of an urban scene composed of a collection of disjoint surface components. An integral part of our method is a surface evaluation technique that we use to assess the quality of a proposed model with respect to both accuracy and efficiency. Our method measures geometric error directly, thus allowing the user to control the level of detail in the output by adjusting the desired error tolerance. Fast and reliable comparisons between different instances and families of surfaces enables us to use a customized model for each component. Inline evaluation also leads to robustness because the selection of the final surface model is delayed until information from all of the scans is available. This permits a wide range of possible surface

approximations to be applied. The only requirement is that the surface formulation must be locally linear. If it can be approximated from a set of linear pieces, then it can be constructed by merging the components of a linear segmentation. Adapting this idea to data other than 3D range scans is a focus of our future work on reconstruction methods.

We identified four requirements that we designed our algorithm to satisfy:

1. **Fast.** Segmentation proceeds one scan at a time, sequentially or in parallel. Afterward, scan merging involves only a small set of planes, and then surface fitting is performed one component at a time. Thus, we never need to process the entire point set at the same time.

2. **Global.** Surface selection and evaluation always involves fitting a geometric model directly to subsets of the raw data. Our algorithm minimizes geometric error globally, so error is always comparable between two different approximations of the same component, or between two different components.

3. **Parametric.** Since we use parameterized surfaces, the final representation produced by our algorithm is much smaller than the raw data while still modeling the geometry closely.

4. **Topological.** Starting with topological information derived from the input range images, our algorithm preserves topology from end to end. Connectivity information is maintained in graph form throughout the segmentation process. Subsequently, a component is always treated as a topological disk.

Our segmentation method combines random sampling with k-means refinement to cut each scan into planar components. A novel merging scheme parametrizes overlapping components in a shared domain. Then, each component undergoes an extensive surface selection and fitting process that is highly customizable.

The principal contributions of this thesis are:

- Our evaluation method, which we use to analyze 3D surface models for both accuracy and efficiency. Accuracy is characterized by generalization error, while efficiency is characterized by computing the bit rate as a measure of entropy.
- Our reconstruction algorithm, which we use to create a piecewise surface model from a set of registered range scans. Our algorithm is controlled by a small number of parameters specifying acceptable error tolerances and minimum size constraints for the output components.
- The piecewise surface model itself, a union of bounded parametric surfaces that provides an analytic description of scene geometry suitable for input to high-level processing such as object recognition.
- A discretization algorithm to compute a triangle mesh from the model, with level-of-detail controlled by simple error tolerances.

Our experience indicates that segmentation should be performed early in the pipeline when the data contains many objects of known type. A piecewise model captures structural information about a scene that is difficult to generate from a monolithic representation in which many surfaces coexist in a single data structure or a single domain. In computer graphics, areas of discontinuity must be modeled accurately because edge information plays such an important role in the human visual system [32]. Our approach treats discontinuity as an important feature of a scene, and explicitly models it by maintaining the boundary of every component. The combination of geomety and topology in our output promises to replace raw data in higher-level recognition tasks.

6.1 Future Work

The ultimate goal of surface reconstruction is to produce a high-level scene description from point samples that is as detailed as a CAD model. CAD models are highly organized. An object is represented not just by its external geometry but also by relationships among its parts. Furthermore, it is complete: each part is a watertight manifold with discernable properties such as volume and surface area. In the world of reverse engineering, this is done by performing surface reconstruction at high resolution, and simplifying the resulting triangle mesh output. For example, in the commercial software package PolyWorks/Modeler, segmentation is achieved using automatic and interactive mesh editing tools. But, as we have seen, the size and complexity of terrestrial range data make full-resolution modeling extremely difficult. Moving segmentation to the front of the pipeline effectively divides the data—to be conquered in parts—at the expense of some coverage lost because we eschew superresolution at segmentation time. But complete coverage is not possible since there are holes and missing parts in the original input scans. Various methods have been designed to fill holes and otherwise complete meshes [40, 22, 27]. One important goal of our surface reconstruction is to extract as much information from the raw data as possible so postprocessing algorithms can focus on higher-level abstractions. For instance, the commercial software package ClearEdge3D implements a pipeline that extracts rectangular shapes from a data set in order to bootstrap interactive CAD modeling. The user imports the shapes into a traditional CAD package for further manipulation. For instance, the flat shapes can be extruded to form walls. Our trimmed surface representation can easily yield simplified CAD-ready input using 2D fitting algorithms, and our trimmed surfaces can likewise be converted to Nonuniform Rational B-Spline (NURBS) curves, the preferred representation in CAD software.

The intelligence in a CAD model stems mainly from the hierarchical ordering of parts. Therefore, orderly decomposition of raw data into parts is a critical first step in a bottom-up reconstruction scheme. Our surface reconstruction algorithm lays the groundwork for such a scheme. We have aggregated primitive planar surface regions into smooth, topologically well-defined components. One direction of future research is to move up the tree to higher-level recognition tasks, where the rich geometric models that we produce offer great advantages over raw data as input. An alternate path is to explore the space of approximations, both in higher dimensions, and in the variety of modeling abstractions employed. Customization to other geometries and surface formulations could yield powerful application-specific reconstruction methods in the future.

In the near term, we plan to apply our algorithm to the task of rendering, which requires us to clean up the model by simplifying boundaries, filling holes, and completing gaps. Boundary simplification in the 2D parameter space can be accomplished using the Douglas-Peucker algorithm [11]. Stamos *et al*. filled gaps between adjacent components by extending surfaces until they meet at corners [37]. They also filled holes using constrained Delaunay triangulation. Many larger gaps can be filled by integration with BPA, which can recover many nonplanar areas, and works at higher resolution than our segmentation. After recovering all the surface information we can from the range data, completing the remaining geometry automatically without actual data remains an open problem. It may be possible, however, to complete the geometry manually. We plan to create an interface to CAD by converting our trimmed surfaces to NURBS and other compatible shape approximations. As a labor-saving tool, an automatic reconstruction preprocess saves the modeler from having to start from scratch. Like rigging for character animation, surface components provide handles in the scene that the modeler would otherwise have to place by hand.

Once the geometry is complete, the options for texture mapping are far greater in a

space of disjoint components than they are in a monolithic surface with ill-defined corners. Texture alignment and blending, like surface fitting, are best performed over continuous, smooth regions. Rather than covering up imperfections in the geometry, texture applied to accurate geometry should be convincing from novel angles. Using automatic 2D-to-3D image registration and multiview geometry [36], we plan to generate texture maps from wide-baseline color images taken independently of the scanning process. As in the case of geometry, piecewise parameterization should lead to a texture merging and blending algorithm that can take full advantage of the resolution of its input.

Bibliography

[1] A. Bab-Hadiashar and N. Gheissari, "Range image segmentation using surface selection criterion," *IEEE Trans. Image Process.*, vol. 15, no. 7, pp. 2006–2018, 2006.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Trans. Vis. Comput. Graphics*, vol. 5, no. 4, pp. 349–359, 1999. [Online]. Available: http://citeseer.ist.psu.edu/bernardini99ballpivoting.html

[3] P. J. Besl and R. C. Jain, "Segmentation through variable-order surface fitting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 10, no. 2, pp. 167–192, 1988.

[4] P. J. Besl and N. McKay, "A method for registration of 3-D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, 1992.

[5] C. Chen and I. Stamos, "Range image segmentation for modeling and object detection in urban scenes," in *Intl. Conf. 3-D Digital Imaging and Modeling (3DIM)*, Montreal, Canada, 2007, pp. 185–192. [Online]. Available: http://www.cs.hunter.cuny.edu/ ~ ioannis/3DIM_2007.pdf

[6] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," in *Proc. IEEE Conf. Robotics and Automation*, 1991.

[7] Z.-Q. Cheng, Y.-Z. Wang, B. Li, K. Xu, G. Dang, and S.-Y. Jin, "A survey of methods for moving least squares surfaces," in *Proc. IEEE/EG Symp. Volume and Point-Based Graphics*, 2008.

[8] D. Cohen-Steiner, P. Alliez, and M. Desbrun, "Variational shape approximation," in *Proc. SIGGRAPH*, Los Angeles, CA, 2004, pp. 905–914. [Online]. Available: ftp://ftp-sop.inria.fr/geometrica/alliez/approximation.pdf

[9] T. Darrell, S. Sclaroff, and A. Pentland, "Segmentation by minimal description," in *Proc. Intl. Conf. Computer Vision (ICCV)*, Osaka, Japan, 1990, pp. 112–116.

[10] T. K. Dey, *Curve and Surface Reconstruction*. Cambridge: Cambridge University Press, 2007.

[11] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a line or its caricature," *The Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, 1973.

[12] R. O. Duda and P. E. Hart, "Use of the Hough transform to detect lines and curves in pictures," *Comm. ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[13] R. Englert, "Acquisition of complex model knowledge by domain theory-controlled generalization," *Computing*, vol. 62, no. 4, pp. 369–385, June 1999.

[14] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Comm. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[15] P. F. U. Gotardo, K. L. Boyer, O. R. P. Bellon, and L. Silva, "Robust extraction of planar and quadric surfaces from range images," in *Proc. Intl. Conf. Pattern Recognition (ICPR)*, Cambridge, England, 2004, pp. 216–219.

[16] V. M. Govindu, "A tensor decomposition for geometric grouping and segmentation," in *Proc. Conf. Computer Vision and Pattern Recognition (CVPR 2005)*, San Diego, CA, 2005, pp. 1150–1157.

[17] A. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica*. Boca Raton: CRC Press, 1999.

[18] M. Grossberg, S. Gottipati, I. Gladkova, M. Goldberg, and L. Roytman, "An analysis of optimal compression for the Advanced Baseline Imager based on entropy and noise estimation," in *Proc. Satellite Data Compression, Communications, and Archiving II*, vol. 6300. Society of Photo-Optical Instrumentation Engineers (SPIE), 2006, p. 63000M.

[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer, 2001.

[20] A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. Bowyer, D. W. Eggert, A. Fitzgibbon, and R. B. Fisher, "An experimental comparison of range image segmentation algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 7, pp. 673–689, 1996. [Online]. Available: http://citeseer.ist.psu.edu/hoover96experimental.html

[21] S. Hou, K. Lou, and K. Ramani, "SVM-based semantic clustering and retrieval of a 3D model database," *Computer-Aided Design & Applications*, vol. 2, pp. 155–164, 2005.

[22] T. Ju, "Robust repair of polygonal models," in *Proc. SIGGRAPH*. New York, NY, USA: ACM, 2004, pp. 888–895.

[23] S. Lee, G. Wolberg, and S. Y. Shin, "Scattered data interpolation with multilevel B-splines," *IEEE Trans. Vis. Comput. Graphics*, vol. 3, no. 3, pp. 228–244, 1997.

[24] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proc. SIGGRAPH*, 1987, pp. 163–169.

[25] M. Meyer, M. Desbrun, P. Schröder, and A. Barr, "Discrete differential geometry operators for triangulated 2-manifolds," in *Proc. Intl. Workshop on Visualization and Mathematics (Vismath)*, Berlin-Dahlem, Germany, 2002. [Online]. Available: http://citeseer.ist.psu.edu/meyer02discrete.html

[26] W. W. Y. Ng, A. Dorado, D. S. Yeung, W. Pedrycz, and E. Izquierdo, "Image classification with the use of radial basis function neural networks and the minimization of the localized generalization error," *Pattern Recognition*, vol. 40, no. 1, pp. 19–32, January 2007.

[27] J. Podolak and S. Rusinkiewicz, "Atomic volumes for mesh completion," in *Proc. 3rd Eurographics Symp. Geometry Processing (SGP)*. Aire-la-Ville, Switzerland: Eurographics Association, 2005, p. 33.

[28] A. Ramli and I. Ivrissimtzis, "Bootstrap test error estimations of polynomial fittings in surface reconstruction," in *Vision, Modeling, and Visualization Workshop*, November 2009.

[29] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Intl. Conf. 3-D Digital Imaging and Modeling*, 2001.

[30] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," *Comput. Graph. Forum*, vol. 26, no. 2, pp. 214–226, 2007.

[31] T. Schürmann and P. Grassberger, "Entropy estimation of symbol sequences," *CHAOS*, vol. 6, no. 3, pp. 414–427, 1996.

[32] R. M. Shapley and D. J. Tolhurst, "Edge detectors in human vision," *J. Physiol.*, vol. 229, no. 1, pp. 165–183, February 1973.

[33] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in *Applied Computational Geometry: Towards Geometric Engineering*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds. Springer-Verlag, May 1996, vol. 1148, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry.

[34] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000. [Online]. Available: http://citeseer.ist.psu.edu/shi97normalized.html

[35] I. Stamos and M. Leordeanu, "Automated feature-based range registration of urban scenes of large scale," in *Intl. Conf. Computer Vision and Pattern Recognition (CVPR)*, June 2003, pp. 555–561.

[36] I. Stamos, L. Liu, C. Chen, G. Wolberg, G. Yu, and S. Zokai, "Integrating automated range registration with multiview geometry for the photorealistic modeling of large-scale scenes," *Intl. J. Computer Vision*, vol. 78, no. 2–3, pp. 237–260, 2008. [Online]. Available: http://www.cs.hunter.cuny.edu/ ~ ioannis/IJCV_2008.pdf

[37] I. Stamos, G. Yu, G. Wolberg, and S. Zokai, "3D modeling using planar segments and mesh elements," in *3rd Intl. Symp. 3D Data Processing, Visualization and Transmission (3DPVT)*, University of North Carolina, Chapel Hill, 2006.

[38] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proc. SIGGRAPH*, 1994, pp. 311–318. [Online]. Available: http://www-graphics.stanford.edu/papers/zipper/

[39] G. Vosselman, B. G. H. Gorte, G. Sithole, and T. Rabbani, "Recognising structure in laser scanner point clouds," in *Intl. Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences (IAPRS)*, vol. 46, part 8/W2, Freiburg, Germany, 2004, pp. 33–38. [Online]. Available: http://www.itc.nl/personal/vosselman/papers/vosselman2004.natscan.pdf

[40] J. Wang and M. M. Oliveira, "Improved scene reconstruction from range images," *Computer Graphics Forum*, vol. 21, pp. 521–530, 2003.

[41] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 11, pp. 1101–1113, 1993.

[42] G. Yu, M. Grossberg, G. Wolberg, and I. Stamos, "Think globally, cluster locally: A unified framework for range segmentation," in *4th Intl. Symp. 3D Data Processing, Visualization and Transmission (3DPVT)*, Georgia Institute of Technology, Atlanta, GA, June 2008.

[43] Y. Yu, A. Ferencz, and J. Malik, "Extracting objects from range and radiance images," *IEEE Trans. Vis. Comput. Graphics*, vol. 7, no. 4, pp. 351–364, 2001. [Online]. Available: http://citeseer.ist.psu.edu/yu01extracting.html