INFERENCE-BASED GEOMETRIC MODELING FOR THE GENERATION OF COMPLEX CLUTTERED VIRTUAL ENVIRONMENTS

A Dissertation

by

KEITH EDWARD BIGGERS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2011

Major Subject: Computer Science

INFERENCE-BASED GEOMETRIC MODELING FOR THE GENERATION OF COMPLEX CLUTTERED VIRTUAL ENVIRONMENTS

A Dissertation

by

KEITH EDWARD BIGGERS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	John Keyser
	Glen Williams
Committee Members,	Scott Schaefer
	John Junkins
Head of Department,	Valerie Taylor

May 2011

Major Subject: Computer Science

ABSTRACT

Inference-based Geometric Modeling for the

Generation of Complex Cluttered Virtual Environments. (May 2011)

Keith Edward Biggers, B.S., Texas Wesleyan University;

M.S., Texas A&M University

Co–Chairs of Advisory Committee: Dr. John Keyser
Dr. Glen Williams

As the use of simulation increases across many different application domains, the need for high-fidelity three-dimensional virtual representations of real-world environments has never been greater. This need has driven the research and development of both faster and easier methodologies for creating such representations. In this research, we present two different inference-based geometric modeling techniques that support the automatic construction of complex cluttered environments.

The first method we present is a surface reconstruction-based approach that is capable of reconstructing solid models from a point cloud capture of a cluttered environment. Our algorithm is capable of identifying objects of interest amongst a cluttered scene, and reconstructing complete representations of these objects even in the presence of occluded surfaces. This approach incorporates a predictive modeling framework that uses a set of user provided models for prior knowledge, and applies this knowledge to the iterative identification and construction process. Our approach uses a local to global construction process guided by rules for fitting high quality surface patches obtained from these prior models. We demonstrate the application of this algorithm on several synthetic and real-world datasets containing heavy clutter and occlusion.

The second method we present is a generative modeling-based approach that can construct a wide variety of diverse models based on user provided templates. This technique leverages an inference-based construction algorithm for developing solid models from these template objects. This algorithm samples and extracts surface patches from the input models, and develops a Petri net structure that is used by our algorithm for properly fitting these patches in a consistent fashion. Our approach uses this generated structure, along with a defined parameterization (either user-defined through a simple sketch-based interface or algorithmically defined through various methods), to automatically construct objects of varying sizes and configurations. These variations can include arbitrary articulation, and repetition and interchanging of parts sampled from the input models.

Finally, we affirm our motivation by showing an application of these two approaches. We demonstrate how the constructed environments can be easily used within a physically-based simulation, capable of supporting many different application domains.

To Autumn, I could never have done this without you.

ACKNOWLEDGMENTS

This dissertation is the result of my work at Texas A&M whereby I have been guided and supported by many different people. I would like to take this opportunity to personally express my gratitude to each and all of you.

I would like to thank my advisors, John Keyser and Glen Williams. The mentorship you have both provided has helped me grow tremendously as a researcher. I would also like to thank the other members of my committee, Scott Schaefer and John Junkins. I am grateful for the time and effort you have contributed to my work.

I would like to acknowledge Jim Wall and the many wonderful researchers at the Texas Center for Applied Technology that worked with me throughout this process. Working a full-time job, combined with working on a Ph.D., can be an arduous process and all of you helped me get through it.

I would like to thank the I/ITSEC community and scholarship committee for providing me with the scholarship that allowed this research to happen. The inspiration and support you provided helped me tremendously.

I would like to acknowledge the National Institute of Standards and Technology and the Texas Engineering Extension Service who provided the Disaster City® datasets and photographs that inspired my work. I would also like to thank Ann McNamara for the use of her NextEngine 3D Scanner HD and lab space which supported my results.

Finally, I would like to thank my family and friends. Without your love, guidance, and support I would have never taken on and completed this endeavor. I hope now that it is all over that I can make up for all of the lost time.

Thank you all, without you I would not be where I am today!

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION . . . 1
- A. Motivation . . . 1
- B. Overview of Research . . . 6
- C. Thesis Statement . . . 8
- D. Accomplishments . . . 9
- E. Overview of Chapters . . . 10

II BACKGROUND . . . 12
- A. Problem Background . . . 12
  - 1. Geometric Modeling . . . 12
    - a. Preparation . . . 13
    - b. Construction, Placement, and Usage . . . 13
  - 2. Reconstruction . . . 14
    - a. Data Acquisition and Processing . . . 15
    - b. Reconstruction and Usage . . . 20
- B. Previous Work . . . 21
  - 1. Surface Reconstruction . . . 21
    - a. Point-based Surface Reconstruction . . . 21
    - b. Patch-based Surface Reconstruction . . . 22
    - c. Standard Surface Reconstruction . . . 23
    - d. Interactive Surface Reconstruction . . . 24
    - e. Progressive Surface Reconstruction . . . 25
    - f. Feature-based Surface Reconstruction . . . 26
    - g. Hole Filling and Augmentation . . . 26
  - 2. Procedural Modeling . . . 27
  - 3. Parts and Example-based Modeling . . . 29
  - 4. CAD/CAM Feature-based Modeling . . . 30
  - 5. Object Recognition and Matching . . . 31
- C. Challenges and Solutions . . . 32

III INFERENCE-BASED POINT CLOUD RECONSTRUCTION . 35
- A. Overview . . . 35
- B. Algorithm Details . . . 37
  - 1. Phase I: Setup . . . 39
    - a. Prior Construction . . . 39

CHAPTER Page

b. Graph Construction 45
2. Phase II: Object Recognition 46
a. Initial Matching 46
b. Structure Recognition 51
3. Phase III: Object Reconstruction 55
a. Solid Model Construction 56
b. Final Processing 58
C. Results 62
D. Discussion 77
E. Limitations and Future Work 79

IV INFERENCE-BASED GENERATIVE MODELING 82

A. Overview 82
B. Algorithm Details 83
1. Structure Sampling and Annotation 84
2. Fitting Process Initialization 87
3. Parameterization 89
4. Inference-Based Fitting 92
a. Initialization and Stepwise Fitting 93
b. Patch Mapping 95
c. Handling Articulation 97
d. Handling Repetition 99
e. Handling Interchanging 104
5. Solid Model Reconstruction and Integration 106
C. Results 109
D. Discussion 110
E. Limitations and Future Work 115

V EXTENSION TO SIMULATION 118

A. Overview 118
1. Convex Hull Generation 118
B. Point Cloud Reconstruction Examples 120
C. Generative Modeling Examples 123
1. External Simulation Integration 124

VI CONCLUSIONS AND FUTURE WORK 127

A. Conclusions 127
B. Future Work 128

CHAPTER . . . Page

REFERENCES . . . 130

APPENDIX A . . . 146

APPENDIX B . . . 151

APPENDIX C . . . 158

VITA . . . 163

LIST OF TABLES

TABLE Page

1 Details of the Reconstruction Examples 61

2 Details of the Reduced Reconstructed Datasets 75

3 Details of the Generative Modeling Examples 109

LIST OF FIGURES

FIGURE Page

1 Real-world complex environment that serves as an inspiration for our work. 2

2 Surface splatting applied to a point cloud dataset. 3

3 Moving Least Squares and contouring applied to a point cloud dataset. 4

4 Our solid modeling algorithm applied to a point cloud dataset. . . . 5

5 Overview of the geometric modeling process for an environment. . . . 13

6 Overview of the full reconstruction process for an environment. . . . 15

7 A ground-based scanning process capable of scanning very large environments. 16

8 A table-based scanning process capable of scanning smaller environments. 17

9 Examples of ground-based point clouds of cluttered environments. . . 18

10 An example of a table scanner-based point cloud of a cluttered environment. 19

11 An example of a photo-based point cloud of a cluttered environment. 19

12 Point cloud samples and fitted surface splats of increasing size. . . . 22

13 An example cluttered environment (left) and corresponding captured point cloud (right). 36

14 A high-level overview of the setup, recognition, and reconstruction phases of our reconstruction process. 38

15 On the left is a sampled model and on the right is a sub-sampled prior patch (the samples are shown in green, the sub-samples in blue, and the mesh vertices in red). 41

FIGURE Page

16 Feature density and ambient occlusion weights. Lighter colors represent higher weights. 44

17 An example neighbor graph (a green node corresponds to a prior sample and a red edge connects two priors whose support spheres overlap in space). 45

18 The various elements that are constructed during the setup stage for use during the recognition and reconstruction stages. 46

19 An octree is used to uniformly search the point cloud dataset. 48

20 Several iterations of the patch fitting process. 53

21 A final contoured model obtained from integrating the fitted patches. 57

22 A bounding box is used to locate the contributing point samples that should be removed once an object has been reconstructed. . . . 59

23 A synthetically generated example that demonstrates the identification and reconstruction of multiple objects in clutter (Example 1). 64

24 A synthetically generated example that demonstrates our approach's ability to handle locally similar, but globally different objects (Example 2). 64

25 A synthetically generated example that demonstrates our approach's ability to handle noisy data (Example 3). 65

26 A synthetically generated example that demonstrates our approach's ability to handle heavy occlusion (Example 4). 65

27 A synthetically generated example that demonstrates our approach's ability to identify an object of interest surrounded by heavy clutter (Example 5). 66

28 A laser scan-based point cloud and corresponding set of reconstructed models (Example 6). 67

29 A laser scan-based point cloud and corresponding set of reconstructed models (Example 7). 68

FIGURE Page

30 A laser scan-based point cloud and corresponding set of reconstructed models (Example 8). 69

31 A photo-based point cloud and corresponding set of reconstructed models (Example 9). 70

32 A photo-based point cloud and corresponding set of reconstructed models (Example 10). 71

33 A photo-based point cloud and corresponding set of reconstructed models (Example 11). 72

34 A photo-based example illustrating the handling of similar, but differently structured objects (Example 12). 73

35 A photo-based example illustrating the handling of similar, but differently scaled objects (Example 13). 74

36 A series of tests were performed on reduced data to evaluate the robustness of our overall process with respect to dataset resolution. . 76

37 Overview of the generative construction process. 84

38 Example model sampling (green points), patch sub-sampling (blue points), and neighbor graph (nodes in green and edges in red). 85

39 Example annotated model (left) and corresponding annotated samples (right). 87

40 Colored Petri net structure defining the process for fitting a given node in the neighbor graph. Places are illustrated as circles and transitions as rectangles. 88

41 Our sketch-based interface for defining a parameterization. 91

42 Example parameterizations we have experimented with. 91

FIGURE Page

43 An illustration of the patch mapping process used to fit a patch from the template model to its correct position and orientation along the defined parameterization. The extracted patch is translated along the parameterization to the determined t value, and then rotated such that its principal axes are aligned with the sketched curve. 96

44 Mapping of samples and sub-samples along a defined parameterization. 97

45 The before/after states for cleanly mapping patches across articulated joints. 98

46 Gaps in the surface definition caused by articulation can be filled by “stretching” a patch across the void. This is performed by identifying a fixed width region on either side of the crease in template space, locating the samples in each region and their corresponding nearest neighbor in the opposite region, and then sub-sampling the line connecting each pair of samples. 100

47 Points from an inter-surface intersection caused by articulation can be removed by finding the bisection plane at the joint, and performing a simple spatial test to identify the points that lie on one side of this plane in template space and the opposite side in construction space. 100

48 Results of the patch fitting process using articulation and repetition of parts. The top image shows the fitted patches and the bottom shows the repeated parts. 102

49 Results of the patch fitting process using interchangeable parts. The top image shows the fitted patches and the bottom shows the interchanged parts. 105

50 An example constructed object (center) from a block template (top left). 107

51 Example environment composed of several constructed objects. . . . 108

52 Constructed blocks using articulation and part repetition (Example 1). 110

FIGURE Page

53 Constructed bricks using only part repetition (Example 2). 111

54 Constructed pipes using articulation and part repetition (Example 3). 112

55 Constructed blocks using articulation, and repetition and interchanging parts (Example 4). A peg is interchangeable with a flat region, and these elements can be repeatedly swapped across the model.113

56 Constructed environment similar to the provided real-world example that inspired this approach (Example 5). The different configurations of pallets were generated using random interchanging and repetition of board configurations. 114

57 Constructed object with its corresponding convex hull. 119

58 Results from two different synthetic datasets (Examples 1 and 2) reconstructed and incorporated into a physically-based simulation. . 120

59 Results from a scan-based dataset (Example 7) reconstructed and incorporated into a physically-based simulation. 121

60 Results from a photo-based dataset (Example 9) reconstructed and incorporated into a physically-based simulation. 122

61 Results of a generative modeling-based environment incorporated into a physically-based simulation. 123

62 A reconstructed environment incorporated into a commercial game engine. 125

63 An initial random sampling (left) and uniform sampling after relaxation (right). 148

64 An illustration of the mapping of samples onto a common plane for both adjacent faces (left) and remote faces (right). 149

65 A simple least squares fitting for a random set of points. 152

66 A Moving Least Squares fitting for a random set of points. The top illustration shows the local fitting process, and the bottom illustration shows how the set of these local fittings form the approximated surface. 153

FIGURE Page

67 An illustration of the iterative convergence process for projecting a point onto the surface. After n iterations, the point x will reside on the surface $\mathcal{S}$. 156

68 A simple Petri net composed of three places and a single transition for validating user access to a system. 159

69 An example of the iterative token passing process used within a Petri net. Places are illustrated as circles and transitions as rectangles. 160

CHAPTER I

INTRODUCTION

A. Motivation

As the use of simulation increases across many different domains, the need for high-fidelity three-dimensional virtual representations of complex environments has never been greater. This need has driven the research and development of both faster and easier methodologies for creating such environments. The objective of these methods is to allow for the construction of very realistic and detailed virtual representations more quickly and easily. These constructed environments can then be incorporated into different types of applications and utilized in areas such as virtual testing and evaluation of equipment, rehearsal of real-world operations, and analysis of concepts related to the introduction of new technology.

Figure 1 shows a very complex real-world environment that serves as an inspiration for our work. This environment contains *clutter*, a collection of objects residing in a small amount of space. Accurately modeling by hand an environment such as that shown would be an extremely difficult and time consuming process. Capturing the full detail in a faithful fashion would involve great attention to detail, and an intricate modeling process. In many cases such an approach is not a feasible option due to the underlying cost and complexity required. Thus, more automated methods capable of automatically re-creating an environment to an appropriate level of detail are necessary.

Due to the portability and availability of three-dimensional data acquisition de-

This dissertation follows the style of *IEEE Transactions on Visualization and Computer Graphics.*

Fig. 1. Real-world complex environment that serves as an inspiration for our work.

vices, efficient and accurate representations of even extremely complex scenes such as that shown in Figure 1 can be rapidly and easily obtained. The challenge then becomes how to process the collected data samples (i.e., a *point cloud*) into a representation that adequately captures a scene's details in an automated and timely fashion. Surface reconstruction from point cloud data provides one such option. The underlying problem of surface reconstruction focuses on, given a set $\mathcal{C}$ of unstructured three-dimensional points (i.e., a point cloud assumed to sample an unknown surface), generate an approximated surface representation $\mathcal{S}$ that characterizes these point samples.

Many existing automated reconstruction algorithms are capable of developing such an accurate representation while minimizing involvement by the user (i.e., requiring only minimal parameter tuning). This trade-off between automated reconstruction and interactive geometric editing has been a topic of previous study [1].

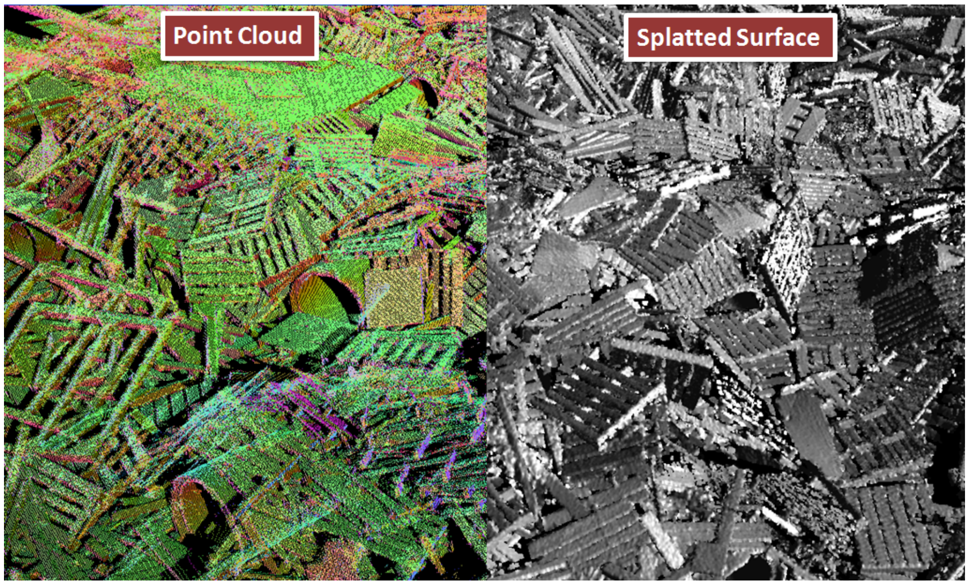


Fig. 2. Surface splatting applied to a point cloud dataset.

However, the underlying complexity of environments such as that shown in Figure 1, and the level of fidelity of which the reconstructed environment may need to entail, can create major challenges for any automated or manual modeling process.

Depending on the underlying application requirements in which a virtual representation is to be used, the level of detail of the reconstructed environment may vary. There is a wide spectrum of techniques capable of generating different fidelity results. At one end of this spectrum is a simple point-based representation of the sampled surfaces in an environment. Figure 2 shows an example output of such an approach generated using a surface splatting technique [2][3]. These methods reconstruct visually continuous surfaces by leveraging the graphics hardware and using the point samples themselves as display primitives. In many applications this simple representation may suffice (e.g., for visual navigation and analysis of an environment), but these approaches do not provide a physically continuous surface representation

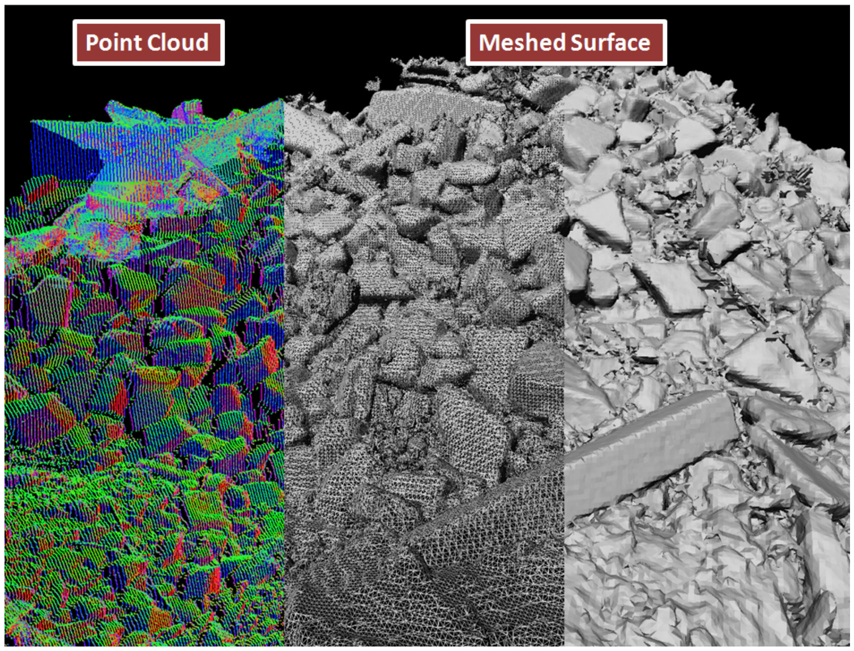


Fig. 3. Moving Least Squares and contouring applied to a point cloud dataset.

necessary in most simulation-based domains.

Residing in the middle of the spectrum are those approaches that reconstruct a single continuous representation of only the visible surfaces captured by the sampled data. Figure 3 shows an example of such an approach generated by constructing a Moving Least Squares surface representation [4], followed by a contouring algorithm to obtain a polygonal mesh representation [5]. In many applications this higher fidelity representation is necessary (e.g., for virtual testing of navigation algorithms and sensor equipment). These approaches are typically slower due to a more expensive reconstruction process, but they provide a smooth and continuous reconstruction of the sampled surfaces. These approaches are limited to working with the point cloud data only, and thus reconstruct only a single continuous surface of the sampled surfaces captured in the point cloud (i.e., only the visible surfaces). As a result, they

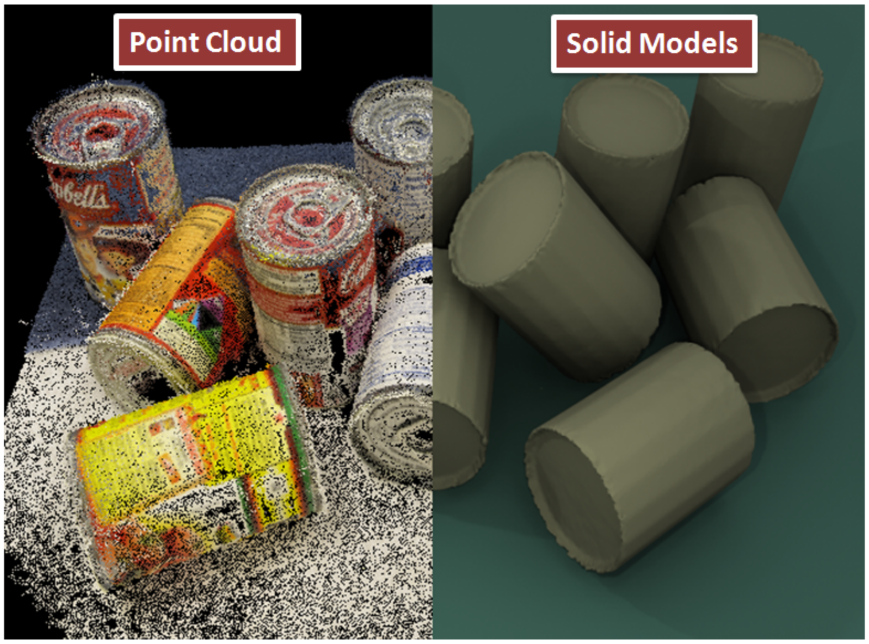


Fig. 4. Our solid modeling algorithm applied to a point cloud dataset.

are not able to distinguish between nor break apart individual objects.

At the far opposite extreme of the spectrum are those techniques that perform a full reconstruction of a scene, where each object is a distinct and complete element capable of being transformed and interacting with other objects in the environment. For many simulation-based applications this highest fidelity reconstruction is required (e.g., for analysis of structural stability or environmental analysis). Figure 4 shows an example of such an approach generated using an inference-based solid modeling algorithm presented in this dissertation. This method provides a full reconstruction of an environment using individual solid models, but must deal with many challenges. These challenges include handling the complexity involved with clutter, and the resulting occlusion that can result within a point cloud. A reconstruction approach capable of generating a result to this level of detail would greatly benefit any simulation-based domain that requires full dynamics of objects within an environment.

All of the previously discussed methods provide approaches for reconstructing a real-world environment from a captured point cloud representation. In some cases this captured data may not be readily available during the modeling process. Thus, alternative methods to reconstruction must also exist to allow for quick generation of environments in these situations. These modeled environments may not be exact reconstructions, but still need to embody similar characteristics and be very realistic in form.

The real-world environment shown in Figure 1 contains a large number of objects composed of similar features, yet each individual object consists of a different underlying composition. Thus, to more easily re-create such an environment, generative techniques that allow for a wide range of objects to be developed very quickly requiring only minimal effort by the user are critical. Using one of these techniques, a user would simply provide an object template that is then used to guide the underlying construction process. A series of different objects with common underlying traits, but different overall form could then be constructed.

Both the reconstruction and generative modeling approaches discussed allow for the construction of virtual representations of a wide variety of different types of complex environments. These virtual representations can then be used to help support applications across many different domains.

B. Overview of Research

In this dissertation, we propose two alternative modeling approaches capable of developing solid model representations for different situations. Both approaches center on the idea of taking one or more user provided template models as input, and using these templates to define and guide a construction process. A set of patches is extracted from each input model, and then iteratively fit together under different conditions to fully define the boundary of an object. The output of both methods is a set of one or more solid models defined from these fitted patches.

We illustrate two different approaches based on this underlying idea. The first approach focuses on reconstruction from point cloud data, and using this technique we demonstrate the ability to recognize and reconstruct objects within a captured cluttered environment. The second approach focuses on generative modeling, and using this technique we demonstrate the ability to easily construct different object configurations that include articulation, and repetition and interchanging of parts from a provided template. These two proposed methods build on and extend existing methods by providing a new technique for constructing solid models from user provided templates.

The motivation of our work is to create virtual environments with characteristics similar to the real-world cluttered environment shown in Figure 1. Our approaches illustrate how solid model reconstruction can be performed within cluttered point cloud datasets, as well as how to construct a wide range of varying objects when trying to model similar environments. These constructed environments need to consist of a set of solid model objects, in order to allow for their most effective use within simulation-based applications.

We provide our proposed techniques and a detailed analysis of their capabilities in this dissertation. We also show many different synthetic and real-world results using our approaches, and to affirm our motivation, we demonstrate the application of generated results from our techniques to the simulation domain. The discussions in the remainder of this dissertation will provide a detailed look at, and an understanding of the work that we performed.

C. Thesis Statement

The thesis of this dissertation is as follows:

> Inference-based modeling, where locally defined surface patches along with corresponding patch interaction rules are extracted from a provided template model and incrementally fit around observed locally available information, provides an effective means of constructing solid models in a variety of situations. Inference-based modeling allows for reconstruction of statically defined objects within point cloud data, as well as dynamically defined objects around a characterizing parameterization.

There are several key ideas within this statement. Inference-based modeling relies on a provided template model in which to extract the underlying elements used as part of a fitting process. As these patches are incrementally fit, they fully define the boundary of an object, and can be integrated together to define a solid representation. These patches are fit around observed locally available information which may include surface samples (i.e., obtained from a point cloud) during reconstruction, or a defined parameterization (i.e., characterized by a user or an algorithm) during generative modeling.

The objective of this approach is to provide a means for developing a wide variety of solid models under different types of circumstances. This approach can be used to recognize and reconstruct statically defined objects from within a point cloud representation of a complex environment through using the patch fitting process and validating the object in an iterative fashion. This approach can also be used to construct dynamically defined objects with differing underlying characteristics by simply fitting patches together in a consistent and correct fashion around an underlying parameterization.

D. Accomplishments

The goal of this dissertation is to understand and evaluate a modeling technique for constructing solid models by consistently fitting together patches obtained from a provided template object. We propose two different methods centered on the previously defined concept of inference-based modeling.

The primary contributions of this dissertation are:

- We propose and evaluate an inference-based surface reconstruction approach that is capable of identifying objects of interest amongst a cluttered scene, and reconstructing solid model representations even in the presence of occluded surfaces.
- We propose and evaluate a generative modeling approach that centers on an inference-based construction process and is capable of developing a diverse set of models from a provided set of templates.
- We affirm our motivation by demonstrating an application of our proposed approach through incorporating the generated results of each method into a physically-based simulation.

Our reconstruction approach also provides three sub-contributions:

- We provide an organized and efficient weighted sampling strategy to recognize objects of interest within a point cloud dataset containing clutter.
- We provide a predictive modeling technique that uses an extracted set of surface patches and rules regarding their relationships to incrementally identify and reconstruct the complete structure of an object, even under very uncertain

situations. This avoids the more difficult approach of performing a global rigid object fitting used by many existing methods.

- We demonstrate how our recurrent local to global matching and fitting approach can be used to iteratively fit objects in a cluttered scene, beginning with those that are easily identified, and over time handling those that are more difficult to recognize and reconstruct.

Finally, our generative modeling approach provides three sub-contributions as well:

- We provide an efficient algorithm that locally fits patches around a defined parameterization in a globally consistent fashion, and is capable of generating a solid model representation of the object.
- We provide a means in which this process can function in both a semi-automated and a fully automated fashion using a series of techniques for obtaining the underlying parameterization around which the object is constructed.
- We provide several extensions of our basic algorithm that allow for more complex object definitions through the use of articulation, repetition of parts, and interchangeable parts.

E. Overview of Chapters

The organization of the remainder of this dissertation is as follows:

- Chapter II provides background material relevant to our work. We first provide a detailed look at the typical construction process. Next, we provide a description of the relevant prior work. Finally, we overview some of the primary challenges behind our rationale.

- Chapter III describes our proposed inference-based reconstruction approach that is capable of reconstructing solid models from a point cloud capture of a cluttered scene.
- Chapter IV describes our proposed inference-based generative modeling approach that allows the construction of a wide variety of models from a user provided template.
- Chapter V describes an extension of our work to the simulation domain. We show how the results of both proposed approaches can be easily integrated into a physically-based simulation.
- Chapter VI concludes the dissertation with a review of the major aspects and a look at future directions for this work.

CHAPTER II

BACKGROUND

A. Problem Background

The process of developing a virtual environment with clutter can be performed in several different fashions, and depending on the environment's underlying complexity, may entail a great deal of work. One alternative is to use state-of-the-art modeling tools to create complex sets of geometric models. This approach typically requires manual construction and placement of each individual object into an environment, and may incorporate a variety of different techniques. This approach does not require any specialized resources which allows greater flexibility, but can be complex and expensive (in both time and resources) to perform.

A second alternative is to acquire a three-dimensional representation of a real-world environment using a scanner or other spatial sampling device. Different reconstruction techniques can then be used to re-create a representation at an appropriate level of detail. This approach requires specialized hardware, but can be very efficiently and effectively performed for even complex scenes.

1. Geometric Modeling

Geometric modeling (or simply *modeling*) is the process of defining the underlying geometrical and topological properties for an object of interest. The general process for modeling an environment can be broken down into five stages (shown in Figure 5). The first and second stages prepare for the modeling process. In the third and forth stages, the object is constructed and then placed into an environment. This process is iteratively performed until the environment is complete and ready to be

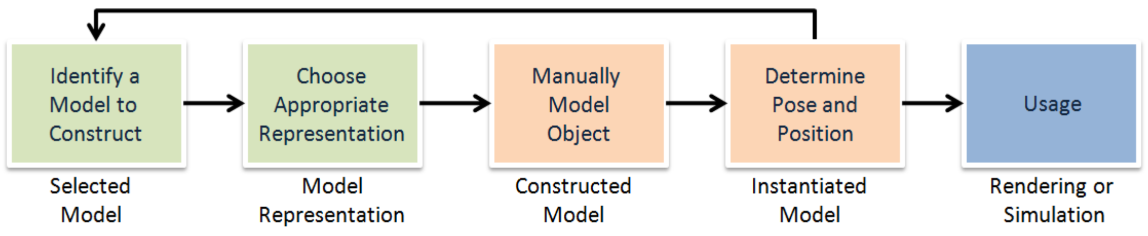


Fig. 5. Overview of the geometric modeling process for an environment.

used in an application for rendering or simulation.

a. Preparation

The process of modeling an environment can be decomposed into modeling one or more individual objects. Thus, it usually begins with the modeler identifying an object of interest. There are many different representations that can be used when modeling an object (i.e., ranging from polygonal meshes, to parametric representations, to constructive solid geometries and many others). The modeler must then decide on an appropriate representation. Once a selection has been made, the modeler must adequately characterize the object using the selected representation. This process is typically performed using an interactive editing environment.

b. Construction, Placement, and Usage

There are a wide variety of applications that allow for geometric editing of different types of objects using different representations. These range from computer-aided design-based applications, to those used as part of the animation, movie, and game industries. In general these applications allow for the interactive definition and construction of objects and environments. These editors can allow for the construction of an object in either a bottom-up or top-down fashion, and the modeling process is performed by iteratively adjusting the properties of an object until it simply "looks correct."

Once an object is constructed using the tools provided by the editor, it then needs to be placed within the virtual environment in a realistic pose and position. This overall approach is repeated iteratively until an adequate number of objects have been developed and placed within a scene. The result is an environment that can be used in a variety of applications.

As environments become larger and more complex, these manual methods for modeling an environment become more time consuming and complicated. Thus, improvements to this overall process are necessary, as well as the incorporation of more automated techniques.

2. Reconstruction

The process of *reconstruction* uses collected data and specialized techniques to develop a virtual representation in a more automated fashion. This process has been used to obtain high-fidelity representations of everything from sculptures [6], to architecture and historical sites [7]. A reconstruction-based process for constructing a virtual environment can be decomposed into four stages (shown in Figure 6).

The first stage, data acquisition, collects a digital sampling of the environment through a laser scanning process. The second stage, data processing, integrates the individual scans into a common coordinate frame and generates a normal for each point sample. The third stage, data reconstruction, generates a continuous representation of the surface to the desired level of fidelity. Finally, the fourth stage takes the resulting reconstructed environment, converts it into a portable format for use by other systems.

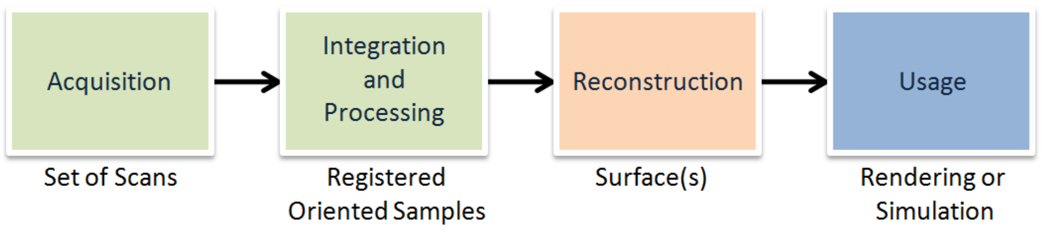


Fig. 6. Overview of the full reconstruction process for an environment.

a. Data Acquisition and Processing

The first stage in the process is to collect a representative dataset. Data can be collected using different methods. The first method is through a standard laser scanning process. This data is typically collected using a series of scans, where each scan is captured from a fixed position and within a fixed field of view (Figures 7 and 8 show two examples of this scanning process). In order to capture adequate detail of an environment, multiple scans from different positions and angles must be captured. As the complexity of the environment increases, more scans must be taken to capture enough detail amongst the clutter of objects. Note that this scanning process is only able to acquire samples from visible surfaces. Thus, any hidden surfaces will not be captured.

The output of this scanning process is a series of individual scans. These scans may be in the form of a point cloud, or in some cases may contain simple surface definitions. Figures 9 and 10 show several scans of real-world environments captured using both the ground-based and table-based scanners, and integrated into point clouds.

Scanning is not the only alternative for capturing a point cloud representation though. A second alternative uses a series of digital photographs taken from different

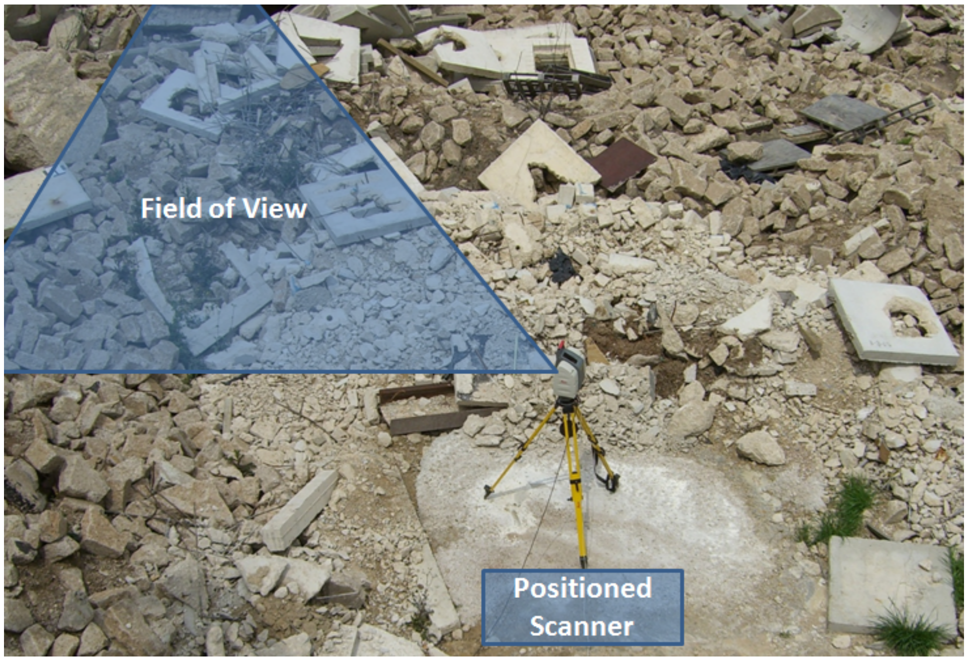


Fig. 7. A ground-based scanning process capable of scanning very large environments.

locations and angles. The collection of these pictures serves as a capture of the environment. Software such as the Bundler package [8] can then be used to produce a reconstruction of the camera locations and sparse scene geometry. The output from Bundler can then be passed to the CMVS software package [9] to reconstruct the sampled three-dimensional structure from the set of collected images. The final output of this process is an extracted point cloud representation of the scene in a similar fashion as obtained with a digital scanner. Figure 11 shows an example point cloud obtained using this method. One important note is that these point clouds usually contain a higher degree of noise, and in some cases may contain voids in the data, due to the feature-based methods used to align and transform the images in space. The differences can clearly be seen between the two different datasets shown in Figures 10 and 11.

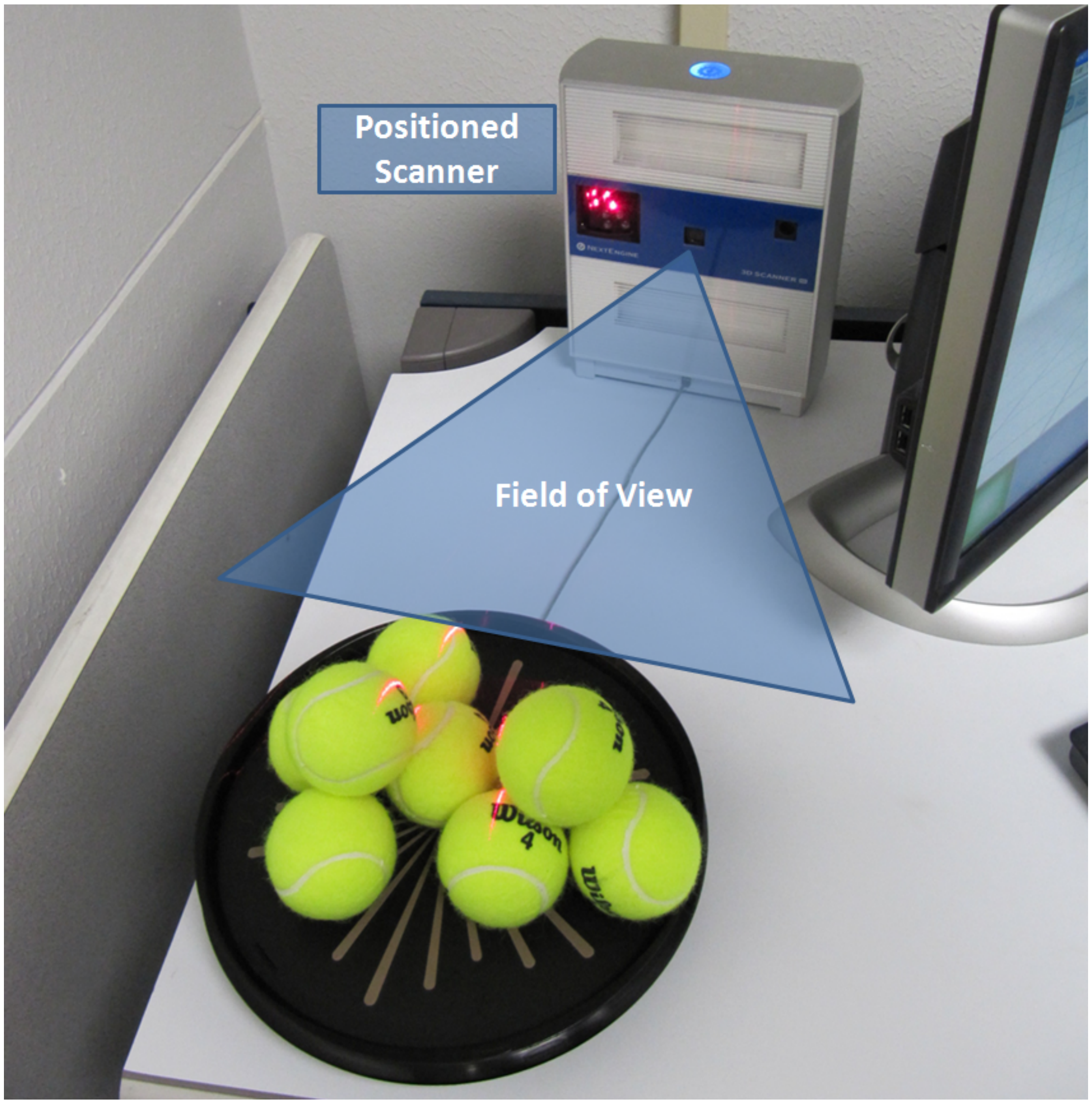


Fig. 8. A table-based scanning process capable of scanning smaller environments.

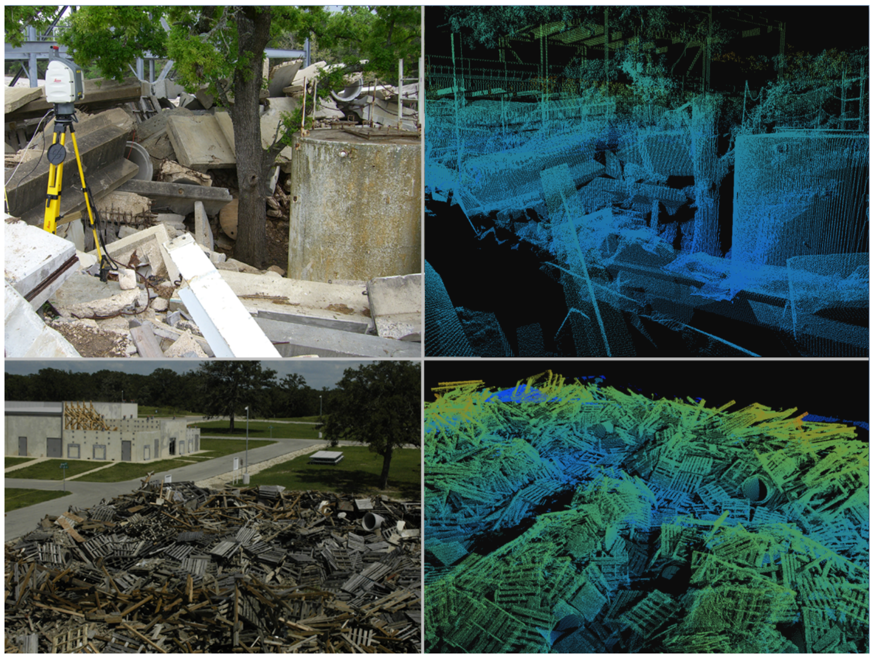


Fig. 9. Examples of ground-based point clouds of cluttered environments.

Once the data has been acquired, the second stage then processes the collected data into a form for use by the reconstruction algorithms. There are several steps involved with this processing. For data that was collected using a series of different captures, a registration and integration of the individual captures into a common coordinate system must be performed. Registration is the process of aligning two different datasets, and the method commonly used for this is an Iterative Closest Point (ICP) algorithm [10]. ICP begins with an initial rough estimation, and then iteratively refines the fitting by rigidly transforming one dataset while trying to minimize the error between the two. There are many variations of this classic technique that incorporate different methods for point selection, matching, weighting, rejecting, and error terms [11]. After all of the datasets have been registered, the result is an integrated set of surface samples residing within a common basis (i.e., a point cloud).

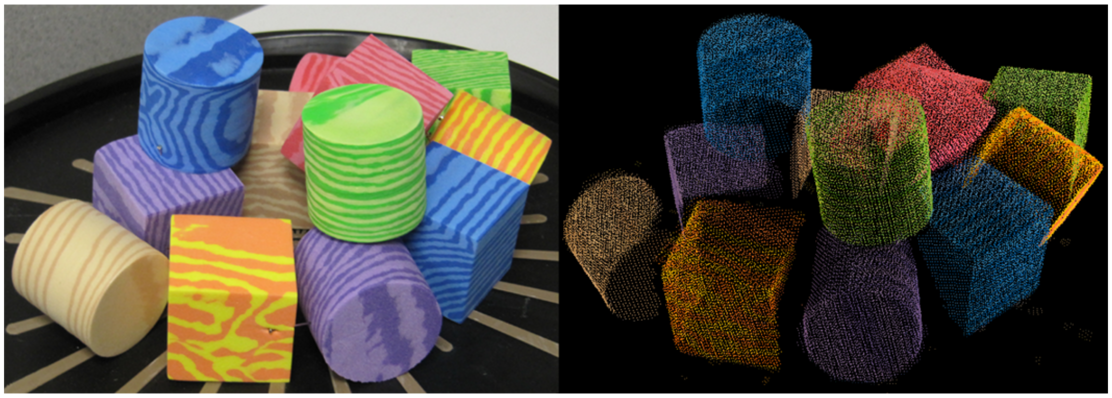


Fig. 10. An example of a table scanner-based point cloud of a cluttered environment.

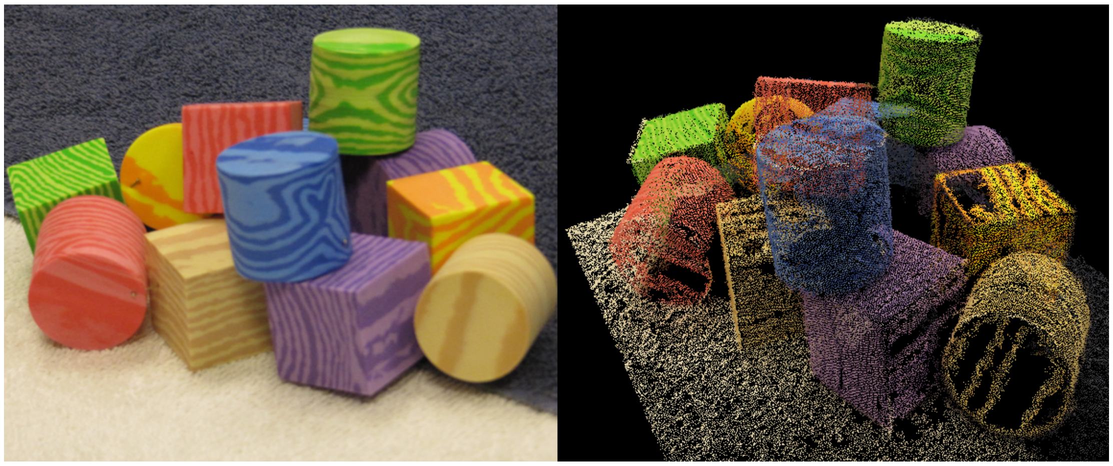


Fig. 11. An example of a photo-based point cloud of a cluttered environment.

Most reconstruction methods work with oriented point samples and require an associated normal with each sample. Thus, the next step must process the data and estimate the normals, if they are not already provided. The most commonly used technique for this process is to perform a least squares fitting using a Principal Component Analysis (PCA) of the covariance matrix of the k-nearest neighbors around a given point [12]. PCA does not guarantee a consistent orientation of surface normals, thus an extra step must be performed to ensure all normals are correctly oriented. For all datasets in this dissertation, we use the known scanner location and a simple dot product between this normal, and the vector from a sample contributing to the plane to the scanner location, to determine if a normal needs to be flipped. The output of this stage is an integrated point cloud consisting of oriented point samples.

b. Reconstruction and Usage

The next stage in the process is to develop or reconstruct a surface representation. There are many techniques that can be used for this step, each with their own strengths and weaknesses, and we will summarize these techniques within the next section. These methods range in complexity from a simple point-based approximation, to a more complex surface-based representation, and finally to solid model reconstructions.

Once the reconstruction is complete, the final constructed environment can be imported into an external application for usage. One important aspect that must be addressed is that the environment must be stored in an understandable and portable format for easy integration into an external application.

B. Previous Work

There is a large body of research related to object modeling and reconstruction in Computer Graphics. Our work draws on ideas related to the areas of surface reconstruction, procedural/generative modeling, parts-based/example-based/feature-based modeling, and object recognition. In the following sections, a discussion of the major previous work in each of these areas is provided.

1. Surface Reconstruction

The area of surface reconstruction has been widely studied, and as a result many different techniques exist. These techniques have resulted in different representations that range from point-based to those that construct a continuous surface description.

a. Point-based Surface Reconstruction

Point-based surface representations provide a simple alternative for visualizing continuous surfaces through the use of the point samples themselves as display primitives. The most widely used of these techniques is surface splatting [2][3]. These approaches are fast and have been shown to scale well to very large datasets [13]. Surface splatting fits a small fixed-size elliptical disc such that it is centered on the point and is perpendicular to the point's normal. Then, using a weighted contribution where the influence diminishes towards the edge of the disc according to a Gaussian distribution, these overlapping discs can be blended together into a visually smooth surface. An illustration of this process is provided in Figure 12 and a full example was previously shown in Figure 2. These approaches rely heavily on graphics hardware, and can provide very efficient, high-quality rendering that does not require the storage of topological information.

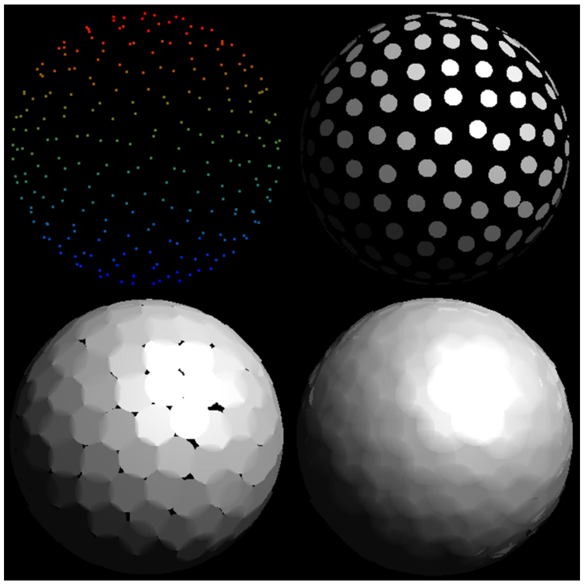


Fig. 12. Point cloud samples and fitted surface splats of increasing size.

b. Patch-based Surface Reconstruction

Patch-based representations provide a slightly more complex method of reconstruction. These approaches fit localized surface patches to groups of related neighboring points. These patches can then be blended together in a follow-on step to provide a continuous surface visualization. The approach by Boubekeur et al. generates locally overlapping 2D Delaunay triangulations of the point set, and then locally aggregates the meshes to obtain a visually continuous surface [14]. Jenke et al. proposed a robust and efficient patch-graph reconstruction algorithm that builds a graph of locally constructed surface patches (from the point cloud data) which is then used as part of a feature-preserving reconstruction process [15].

c. Standard Surface Reconstruction

There are many standard reconstruction-based approaches that generate a continuous representation of the surface the samples represent. Some of the more popular techniques are triangulation-based and implicit-based methods, where each category has their own methodologies as well as advantages and disadvantages.

Triangulation-based approaches represent the classical method of reconstructing a surface using a series of connected triangles (i.e., a piecewise linear surface commonly referred to as a mesh). Some example approaches are Alpha Shapes [16][17], Crust [18], and Cocone [19][20]. The triangulation-based methods generate an accurate surface, interpolating and precisely fitting the data, which may be good or bad depending on the accuracy and noise-level of the dataset itself. These surfaces are both easily stored and rendered with standard graphics hardware, but are often computationally expensive to generate. The Delaunay-based triangulation algorithms provide certain theoretical guarantees that can ensure the integrity of the resulting mesh (e.g., composing angles of the generated triangles). However, these guarantees come at a price as these algorithms typically run slower and do not scale (due to time and memory requirements) to larger datasets without using some form of spatial decomposition and a series of local reconstructions, followed by a patch integration/stitching to define a globally continuous surface [20].

Implicit-based approaches generate a smooth and continuous surface that approximates the data. Some example approaches are Signed Distance Functions [12][21], Radial Basis Functions [22], Moving Least Squares [23][4], and Poisson reconstruction [24]. In general, these approaches tend to work well with noisy data but make generating sharp and concave features, as well as fine-grained detail, very difficult without some form of post-processing. These approaches are fast and can usually scale to larger datasets. The result of these approaches is an implicit surface that is difficult to store and render without first converting it into a meshed representation. The previously shown example in Figure 3 illustrates the result of generating an implicit surface and contouring this representation. Aside from these general surface reconstruction methods, there are many other specialized extensions that have been developed.

d. Interactive Surface Reconstruction

The first specialized extension to surface reconstruction centers around interactive reconstruction. Interactive surface reconstruction is the method of allowing user involvement to dynamically control parameters during the reconstruction process. This could involve anything from parametric ’tweaking,’ to controlling the level of detail of the final reconstructed surface. This area of research has resulted in only a few methods.

Kobbelt and Botsch introduced an interactive approach that works with different types of data (e.g., point clouds, polygons, and NURBS-patches) [25]. This approach allows for adjustment of the orientation and resolution of the triangulated mesh based on the user’s interactively specified demands. However, this approach is a very manual process requiring the user to select the area in which to place a patch, scale and orient it, and then the algorithm automatically stitches the patch into the surrounding elements in the mesh.

Mencl also describes an interactive approach to surface reconstruction [26]. This approach allows manual insertion/deletion of vertices, manual adjustment of the underlying feature description graph, manual surface connection of disjoint surfaces, and a manual point selection driven reconstruction. This interactivity overrides the underlying rules used by the automated process.

e. Progressive Surface Reconstruction

The second specialized extension to surface reconstruction uses a progressive scheme for reconstruction. Progressive surface reconstruction (also referred to as incremental reconstruction) involves a stepwise process that performs the reconstruction in stages. Each stage results in a more accurate surface and can provide visual feedback to the user. This process executes in either a bottom-up or top-down fashion [27].

A bottom-up approach develops a surface based on a predefined set of seed elements and the surface grows outwards to surrounding points of the same surface type. This process continues until encountering a boundary. There are many different bottom-up approaches. Early work took more of a brute force approach [28][29]. Later work used a divide-and-conquer approach to decompose the problem into a series of smaller sub-problems which are locally solved, and then the results are merged [20][30][31][32]. A third approach uses an incremental approach where the surface is constructed element by element in a growing fashion until a boundary is reached [33][34][35].

In a top-down approach the process begins with an assumption that all points belong to a single surface. As the algorithm incorporates additional data, it adds/removes detail to/from this surface and performs an integration process to merge elements. The top-down approach is a less commonly used technique. Ivrissimtzis et al. introduced Growing Cell Structures which function as an incrementally expanding Neural Network (defined as a Neural Mesh) that randomly samples the dataset and adjusts the connectivity of the network based on the selected data points [36]. The nodes and connectivity within this network represent a dual to the vertices and connectivity in the mesh. This approach uses the edge collapse and vertex split transformations as previously defined by Hoppe [37] for evolving a mesh as more data is incorporated.

f. Feature-based Surface Reconstruction

A third specialized extension to surface reconstruction centers around feature-based reconstruction methods. Feature-based methods attempt to identify major elements within a point cloud to ensure that they are accurately represented in the generated surface. This area of research has also resulted in multiple methods functioning in different fashions.

An early approach by Mencl and Muller proposes a method that first creates a surface description graph (i.e., a wire frame of the surface that characterizes certain basic elements identified), and then uses the graph to drive the reconstruction process [38]. Gumhold et al. propose a similar approach that extracts several different feature types (e.g., crease lines, crease junctions, crease loops, border loops, and singleton ends) as part of a preprocessing step, and these features are then used to refine and ensure an accurate surface gets generated during the reconstruction process [39]. A later approach by Fleishman et al. uses a robust Moving Least Squares technique to reconstruct a piecewise smooth surface, and is capable of defining sharp features while using implicit surfaces [40].

g. Hole Filling and Augmentation

Finally, as scene complexity rises, occlusion can cause trouble for many reconstruction algorithms. Some implicit methods can inherently handle smaller voids in data (e.g., [22]). Other approaches have been developed to deal with larger holes by using underlying assumptions (e.g., information from similar surrounding areas [41][42]) or additionally provided information (e.g., geometric/shape prior information [43][44][45]) to fill in the missing details. The approach by Shalom et al. uses the point samples for both position estimation, and to obtain global visibility information to define a better approximated signed distance function and implicit surface [46]. These different approaches have been shown to work well on a wide variety of models. There has also been work attempting to fill larger voids in the data where significant pieces are missing [47]. This approach makes the assumption that the surface can be restored using simple primitives and has been shown to work well for CAD models.

2. Procedural Modeling

Procedural/generative modeling methods have been used in a wide range of applications. In general, many different approaches have been proposed for generating textures, environmental effects, and modeling [48]. Modeling methods have been developed for automatically generating buildings and roads [49][50][51], trees and plants [52][53], and terrain [54][55]. The output from these different methods has been used in real-time games and simulations, and incorporated into animation and movies. These techniques are being used to develop entire mathematically-based worlds [56]. Procedural approaches leverage techniques such as fractals [57], Perlin noise [58], L-Systems [59], tiling [60], and shape grammars [61] to drive the underlying generation process.

Many methods have been developed for generating cities, buildings, architecture, and roads. Kelly and McCabe provide a survey of common techniques [49], and Watson et al. provide an overview of state of the art techniques and applications for city generation [50]. The common approach used begins with the definition of a road network, followed by the generation of a set of buildings around this network. Common methods for developing road networks include L-systems, Voronoi diagrams, and tensor fields [49]. Buildings can then be generated using techniques such as stochastic L-Systems where, based on a building's selected style, it is generated from its footprint using a series of L-system modules/operations that include transformation, extrusion, branching and termination, and selection of geometric templates for defining roofs and various other features [62]. Other approaches include the use of shape grammars where context sensitive shape rules are used to define the interaction between entities of a hierarchical shape description, thereby allowing for a wider range of buildings to be developed [63]. These techniques focus on generation of realistic external structures, but other approaches go beyond just modeling the exterior properties and focus on developing more realistic interior structure as well. Whiting et al. focus on developing structurally sound buildings by incorporating physical constraints and static analysis into each step of the procedural modeling process [64]. Merrell et al. propose a method for generating realistic residential floor plans along with the corresponding three-dimensional structure by using a Bayesian network trained on data obtained from real-world houses [65].

Procedural generation of trees and plants has also resulted in many methods. The classic approach for generating plants is through the use of L-Systems [52]. Extensions of this approach include incorporation of environmental parameters into the construction process [66] and more expressive attributes for developing more complex models [67]. In a different approach by Weber and Penn, a model that depicts the underlying structural development of trees is proposed [68]. In a different area, Deussen et al. focus on the distribution of plants in a realistic and natural fashion to form ecosystems [69]. Multi-resolution generation of plants and trees is also important because it allows generation at different levels of detail for more efficient rendering [70].

Procedural generation of terrain has been another key area of research. The classic approach is to generate a height field (also referred to as a height-map) representation that stores relative altitude at regular intervals. As a result of it being stored in a regular grid, this structure can be easily converted into a meshed representation. Ebert et al. [48] and Smelik et al. [54] both provide good overviews of the many different methods used for this type of terrain generation. Fractals and Perlin noise provide two classic methods for generating height-maps. Additional techniques such as image filtering and cellular automata can also be used to develop the effects of physical phenomena (e.g., erosion and weathering). Other more advanced techniques build on these ideas and are able to leverage the GPU to evaluate and polygonize signed density functions to obtain very complex dynamic terrains at interactive frame rates [71].

Of the many different approaches described, the most closely related to our proposed work is the model synthesis approach by Merrell [72]. His approach extends the 2D texture synthesis problem into higher dimensions, and is capable of generating very large consistent models from a provided example. In later work Merrel et al. extend this idea to continuous model synthesis [73], and allowing for additional geometric constraints to provide a user greater control over the final results [74].

3. Parts and Example-based Modeling

Parts-based and example-based modeling methods have become increasingly popular. These methods commonly use information from known/high quality models for a variety of operations with other unknown/lower quality models. These approaches are not limited to just models, and some have been shown to work well with point cloud data also. These approaches can be broken down into several categories.

The first category fits simple primitives (i.e., planes, cylinders, spheres, etc.) to point cloud data [75][76]. An extension of this idea uses constrained graphs of different primitive shape configurations to describe features, and works well with point clouds from architectural domains [77].

A second category goes beyond simple primitives and relies on a database of object models for extracting the fitting elements. These approaches usually focus on fitting more functional items (i.e., an arm, leg, handle, wheel, etc.). They have made use of ideas such as interchangeable parts [78][79], salient parts [80], and hierarchical analogies between parts [81]. They typically deal with meshed models as opposed to point clouds.

Gal et al. propose another approach that uses a database of local shape priors to augment point clouds during reconstruction. Their approach matches and fits shape priors extracted from provided high quality models to specific regions in a point cloud. These fitted patches are then integrated as part of the reconstruction process. Their approach produces higher quality results by using the fitted priors to smooth out noisy data and fill small gaps [45].

A third category assumes no prior knowledge and simply uses data available from the model itself. These approaches focus on the identification of symmetry and regular geometry within models (i.e., reoccurring parts/features) [82]. Extensions of this work use graphs of salient features [83] and feature lines [84] to help with the process. In general, these approaches are feature-oriented and deal with structural regularity across a dataset (e.g., finding regular patterns of window facades across the surface of a building). These features can then be iteratively transformed across the dataset to define a regular pattern. These approaches have been shown to work with both point cloud data and meshed models and can be used for model repair, compression, and geometry synthesis [82].

4. CAD/CAM Feature-based Modeling

Our work is partially inspired by the large area of feature-based modeling from the CAD/CAM communities. In general, these approaches commonly use a set of defined primitives (i.e., the data) along with a set of rules for primitive interaction (i.e., the relationships between data) to construct solid model representations. There are many survey papers that provide overviews of the different feature-based techniques that have been developed (e.g., [85][86]).

5. Object Recognition and Matching

One major component of our approach is the identification of an object of interest within a larger scene. Many techniques have been developed to solve various problems in object recognition and matching [87][88]. As the complexity of a scene increases, recognition becomes more complex because objects may be only partially visible. This problem is well understood in Computer Vision.

In 2D, many partial recognition techniques have been developed to segment a scene into smaller pieces and then iteratively group the segmented pieces to identify an object [89][90]. Similarly, in both 2D and 3D, geometric hashing can detect local features in a scene using coordinate frames [91]. In 3D, the Spin Image technique uses a local surface matching process followed by a global iterative fit, and has been shown to work well with complex datasets [92]. Other regional point descriptors such as 3D shape contexts and harmonic shape contexts have been proposed as well [93]. The parts-based classification of Huber et al. is a technique that matches local parts (from a database of objects) to possibly occluded objects in a scene and can recognize different classes of known objects [94]. Pairwise alignment techniques such as the 4PCS algorithm allow for partial matching/fitting of two three-dimensional point sets, and can work effectively in the presence of noise [95]. Finally, many of these techniques were developed to support applications for automatic target detection and recognition [96].

Object/shape matching addresses the problem of how to effectively determine the similarity/dissimilarity between two shapes. Tangelder and Veltkamp provide a survey of the overall matching process, as well as a description of some of the more common approaches used [97]. They decompose the matching methods into three primary categories: feature-based (e.g., global features, spatial maps, and local features), graph-based (e.g., model graphs, skeletons, and Reeb graphs), and other methods (e.g., view-based, volumetric error, weighted point set, and deformation-based similarity). Bustos et al. provide a detailed survey of the feature-based methods [98]. Finally, the research at Princeton's Shape Retrieval and Analysis Group has addressed many related issues in the context of matching for shape retrieval [99].

C. Challenges and Solutions

There are many challenges with both geometric modeling and reconstruction of environments such as that shown in Figure 1. All of the methods discussed as part of the previous work attempt to address different aspects of this problem domain, but none provide a complete solution. Thus, as the environments to be modeled become larger and the level of complexity increases, new methodologies must be developed to address these challenges. The methods we propose in this dissertation provide two such alternatives.

Most of the prior work in surface reconstruction focuses on generating a single continuous surface representation from a point cloud. In all of the methods presented, an assumption is made that the point cloud provided as input represents a single element (i.e., an object, building, etc.), and as a result a single surface representation would suffice. Our work takes a different approach. We instead assume that multiple objects exist in the point cloud dataset, and we reconstruct each distinct object individually until no more objects can be identified. As a result, information from provided template models must be used to distinguish between different objects within a scene, and to fill occluded regions that were not captured during the acquisition process due to occlusion.

The prior work in generative modeling provides many alternatives for constructing different elements of an environment such as architecture, vegetation, and terrain. In general, these approaches are limited to developing models of a specific type. The construction process for many of these methods center on an underlying formal grammar (e.g., L-systems for building generation) or a mathematical basis (e.g., Perlin noise for terrain generation). In the case of formal grammars, the underlying rules used must be designed and carefully specified by a user to ensure proper models are obtained. In the case of the mathematical approaches, careful selection and tuning must occur to ensure interesting results are obtained. We take a very different approach. Our method takes a template model as input, and automatically constructs a rule set based on it. The user then has a choice whether to provide an object parameterization for an object, or to allow for a more automated approach. Our approach is not limited to a particular type of object, and differs from many existing approaches whose primary focus is on developing realistic visuals only, in that we focus on constructing arbitrary models in a solid model fashion.

Both of our proposed approaches extend on parts-based and example-based modeling. Our reconstruction approach relies on matching and fitting of extracted patches (from provided object models) to a point cloud. These fitted parts are used to define a complete solid representation for each object in the environment. In our generative modeling approach, defined regions on the template model (which in many cases correspond to parts from an object) are replicated and interchanged to define new variations of the provided template model. We show how the idea of fitting basic elements based on an established set of rules, both of which are obtained from a template model, can provide basic building blocks necessary to construct entire objects of different forms.

Object recognition approaches typically focus on the identification of an object within a larger scene. Our approach is not only able to recognize an object, but also reconstructs a solid representation of it. Our approach can also distinguish between similar yet different objects, and identify objects of varying scale. Most partial recognition techniques use localized information within a point cloud to make a decision about whether an object exists or not, followed by a rigid global fitting of the object to that space. We use an iterative approach that fits extracted patches from the input model in a stepwise fashion to incrementally recognize an object. Thus, localized information is used across a larger region in the point cloud to correctly identify and construct an object.

The two approaches that we propose and evaluate within this dissertation attempt to provide easier and more efficient alternatives for constructing virtual representations of cluttered environments. Our approaches build on and extend from these previous works, and provide new alternatives focused on constructing solid model representations for easier generation of virtual environments.

CHAPTER III

INFERENCE-BASED POINT CLOUD RECONSTRUCTION

A. Overview

As three-dimensional data acquisition devices have become more portable and widely used, the range of applications for digital reconstructions and the complexity of environments being captured has steadily grown. Thus, there is an increasing need for more efficient and accurate reconstruction methods capable of handling more complex environments.

Figure 13 shows an example environment containing clutter, a collection of objects residing in a small amount of space. Many automated reconstruction methods run into trouble with such environments due to the number of distinct surfaces and their spatial proximity. Reconstruction algorithms typically develop a representation of only the visible surfaces which the points sample, combining these points into a continuous surface representation that does not distinguish between separate objects nor capture the complete volumetric structure of distinct objects. Such a representation is not adequate for many application areas, where an unambiguous complete solid representation of each individual object is necessary (e.g., physically-based modeling, environmental analysis, etc.). Thus, taking information from observed portions of an object and inferring details about missing or occluded regions is important for distinguishing between objects and for solid model reconstruction.

A method to reconstruct environments such as the example in Figure 13 must deal with this inherent clutter. Prior research [91][92][93] on cluttered environments has identified several key challenges:

- Clutter hinders the environment capture process by occluding portions of an

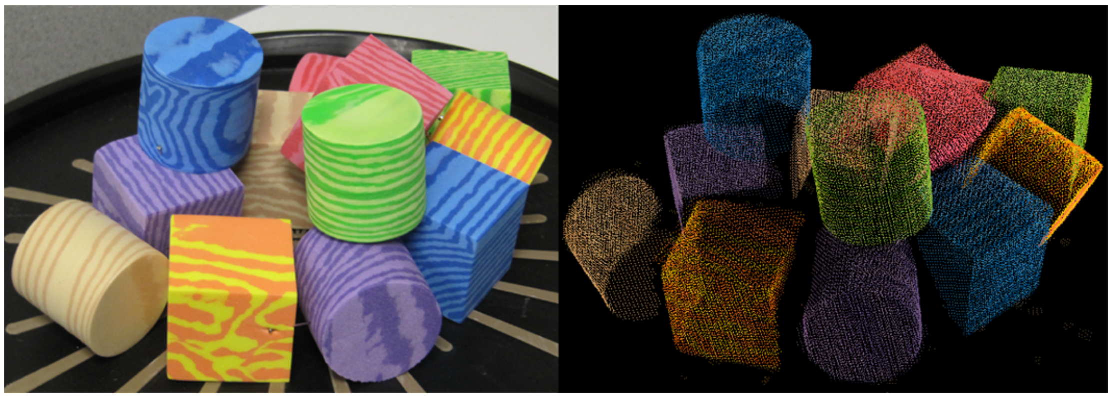


Fig. 13. An example cluttered environment (left) and corresponding captured point cloud (right).

object's surface, resulting in an incomplete surface sampling.

- It can complicate the segmentation of point samples for one object due to the close proximity of other objects in space, often requiring partial recognition techniques to correctly identify an object.
- If two objects have similar local features but different overall global structures, distinguishing between the two objects can be hard. Occluded surfaces make this determination even more difficult.

Our approach uses a partial object recognition strategy along with prior knowledge to infer the details of hidden or unclear structure in point clouds of cluttered environments. This allows us to reconstruct solid models from the point cloud, rather than just a single continuous surface. Our work builds on existing techniques and provides three main contributions:

- We provide an organized and efficient weighted sampling strategy to recognize objects of interest within a point cloud dataset containing clutter.

- We provide a predictive modeling technique that uses an extracted set of surface patches and rules regarding their relationships to incrementally identify and reconstruct the complete structure of an object, even under very uncertain situations. This avoids the more difficult, and often times less accurate, approach of performing a global rigid object fitting used by many existing methods.
- We demonstrate how our recurrent local to global matching and fitting approach can be used to iteratively fit objects in a cluttered scene, beginning with those that are easily identified, and over time handling those that are more difficult to recognize and reconstruct.

B. Algorithm Details

Our approach is decomposed into three phases (shown in Figure 14) and extends the augmentation work of Gal et al. [45]. The first phase constructs the underlying data structures used by the inference process. The second phase identifies individual objects in the scene and fits a series of patches to define an object's boundary. The final phase uses the set of fitted patches to construct a solid representation, followed by removing the contributing point cloud samples so they will not be considered as part of other objects.

The recognition and reconstruction phases function in an automated and iterative fashion, starting with the very apparent and easier to identify objects, and moving on to those that are more difficult to handle. The algorithm continues until there is no longer sufficient evidence to identify an object.

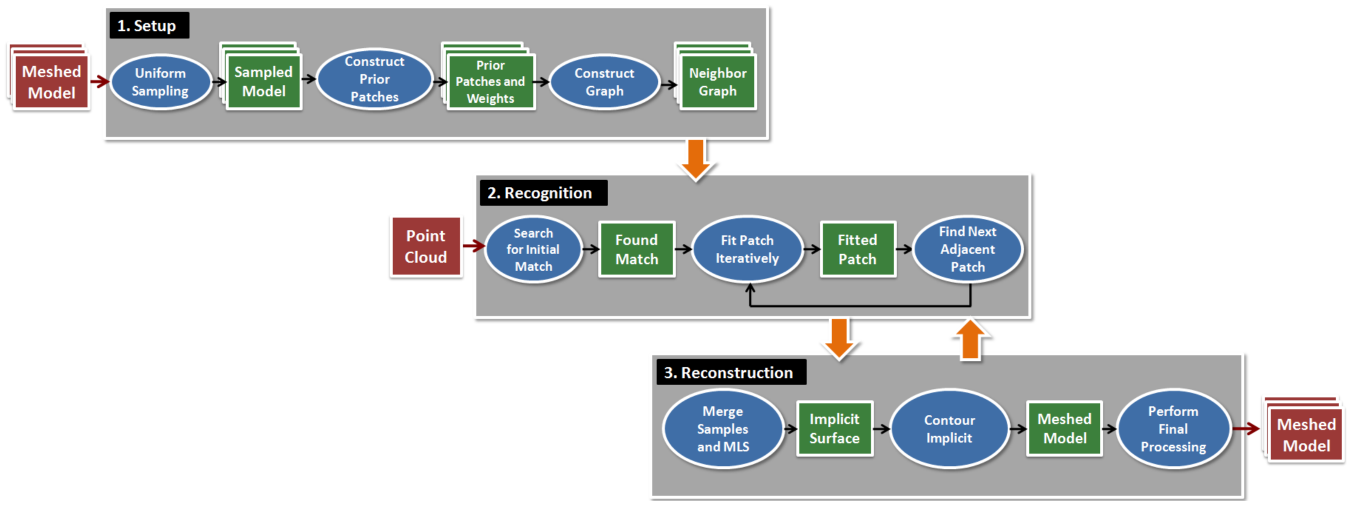


Fig. 14. A high-level overview of the setup, recognition, and reconstruction phases of our reconstruction process.

1. Phase I: Setup

The setup phase (described in Algorithm 1) is a precomputation step in which information is collected for objects to be identified and reconstructed within a scene. We take as input a set of one or more models. We then uniformly sample these models to construct localized surface patches, capture properties of each patch for weighting purposes, and organize the patches into a graph structure that describes their relationship across the object.

Algorithm 1

```
1.  $model_{sel} \in InputModels$;
2.  $uniform\_samples = UniformSampleModel(model_{sel})$;
3.
4.  for $sample \in uniform\_samples$
5.          $p = ConstructShapePrior(sample, \delta, \alpha)$;
6.          $UniformSubsamplePatch(p, s)$;
7.          $T_{p \to n} = NormalizePatch(p)$;
8.          $sig_p = BuildSignature(p, \omega)$;
9.          $w_{feat} = FindFeatureDensityProperty(p)$;
10.         $w_{occl} = FindAmbientOcclusionProperty(p)$;
11.         $StorePatchInList(P, p, T_{p \to n}, sig_p, w_{feat}, w_{occl})$;
12.
13. $G = BuildNeighborGraph(P)$;
```

a. Prior Construction

The first step in the setup process constructs a set of shape priors for each input model. A shape prior (or simply *prior*) is defined as a sampling of the local structure of a surface (i.e., a surface patch). The collection of these patches across an object serves as the core elements leveraged by both the recognition and reconstruction phases of our algorithm.

Sampling

The set of priors P is constructed by uniformly sampling the faces of a model's mesh using an area weighted scheme, followed by a relaxation procedure to ensure an even distribution of samples across the surface [100]. This uniform sampling process is explained in Appendix A. A prior $p \in P$ at a sample μ is defined as the set of faces F (with corresponding face vertices V and face normals N) residing inside a fixed support region s. A support region is defined as a sphere of radius δ located at μ that filters out any samples with a normal angle deviation greater than α. Thus, p is formally defined in Equation 3.1 as:

$$p = \{F \bigcup V \bigcup N \mid dist(v_F, \mu) \leq \delta, \arccos(n_F \cdot n_\mu) \leq \alpha\} \tag{3.1}$$

where the vertex $v_F \in V$, the normal $n_F \in N$, and n_μ is the surface normal at μ. Configuring the support distance δ and support angle α allows customizable control over the discrimination the patches provide, which can be important for recognition in heavily cluttered scenes [92]. We found that an α of 95° provided adequate discrimination for all of the examples used in this dissertation. However, a smaller/larger angle could be used for situations where less/more surface variance is necessary. The value of δ varied based on the input models used, with values ranging from 1.5 to 3.75. An example model sampling is shown in Figure 15.

Signatures

In order to efficiently match the shape priors to regions in the point cloud data, a unique defining signature must be computed for each prior $p \in P$. We use geometric moments (described by Elad et al. [101]) for this surface signature, due to several key traits:

- They are fast to compute and compare, and provide a good descriptor of the essence of a shape.

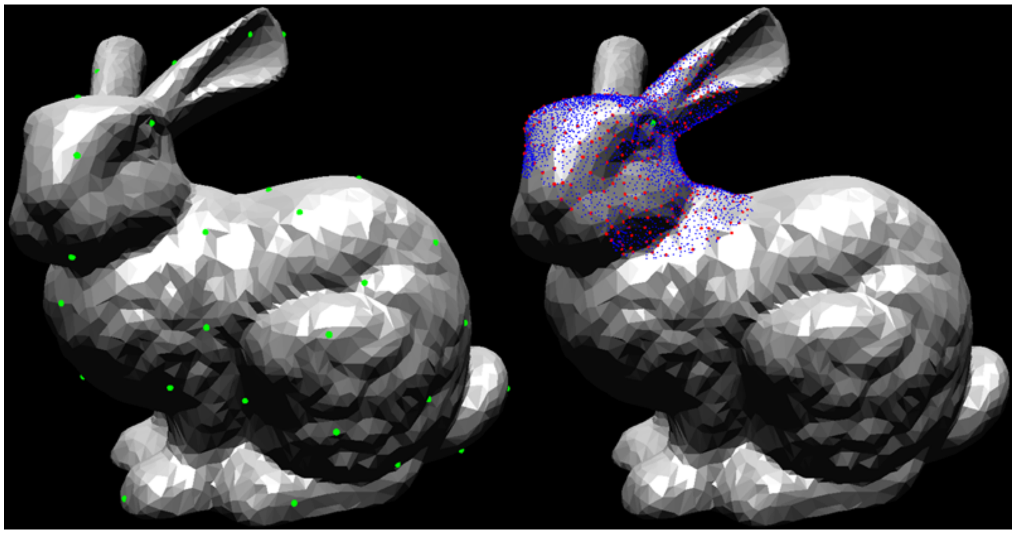


Fig. 15. On the left is a sampled model and on the right is a sub-sampled prior patch (the samples are shown in green, the sub-samples in blue, and the mesh vertices in red).

- They can effectively measure the similarity between two surfaces.
- They work well with point sampled data and do not require extra information (e.g., normals, connectivity information, etc.).
- When computed on normalized data, they are position, rotation, and uniform-scale invariant.

As described by Elad et al. [101], the moment integral can be approximated by a summation over a set of sub-sampled points (shown in Equation 3.2). Thus, for a shape consisting of a sub-sample size N the *(p, q, r)*-th moment is defined as follows:

$$\hat{m}_{p,q,r} = \frac{1}{N}\sum_{i=1}^{N} x_i^p y_i^q z_i^r \tag{3.2}$$

To develop a signature $sig(p)$ for a prior p, the faces making up the prior are first uniformly sub-sampled (using the process as defined by Turk [100] with the additional constraint that the sub-samples must reside inside s; a more detailed explanation of this process is provided in Appendix A). An example sub-sampling of a patch is shown in Figure 15.

These sub-sampled points need to be normalized to a common basis so the comparison of two calculated signatures is invariant to spatial position, rotation, and uniform-scale. Our normalization process follows that defined by Elad et al. [101], and first aligns the points' center of mass with the origin $(0, 0, 0)$, then aligns the points' principal components with the primary coordinate axes, and finally uniformly rescales them so their largest principal component is of unit length. The canonical information $T_{p \to n}$ used to normalize p is stored for use during the later recognition and reconstruction phases. The resulting $sig(p)$ is then defined by a finite vector of scalars from these normalized points (shown in Equation 3.3).

$$sig(p) = \langle m_{1,0,0}, m_{0,1,0}, m_{0,0,1}, \ldots, m_{0,0,\omega} \mid p + q + r \leq \omega \rangle \tag{3.3}$$

Depending on the complexity of the data, different dimensions of moments ω can be used. With higher dimension the signature is more descriptive, but also takes longer to compute and compare. In our experiments we found that an ω of 5 provided adequate results, but a higher dimension could easily be used for capturing more detail. Finally, $sig(p)$ is a simple finite vector and two different signatures sig_a and sig_b representing two different patches can be compared using a distance metric $d(sig_a, sig_b)$ (shown in Equation 3.4).

$$d(sig_a, sig_b) = \|sig_a - sig_b\|^2 \tag{3.4}$$

While a variety of metrics are reasonable, including ones weighting some moments more than others, we found a simple Euclidean distance-based metric to be sufficient.

Two signatures are similar if this resulting distance is small; for evaluation we typically use $1/d(sig_a, sig_b)$ as our similarity score/value. This value can be used to gauge partial similarity between two patches as well.

To allow for more effective matching of patches, we compute two additional "weights" for each prior. The first is based on feature density, and the second on ambient occlusion. These weights, which will be presented next, are used during the recognition process explained in detail within Section 2.

Feature Density Weighting

The feature density weight w_{feat} (shown in Equation 3.5 and illustrated in Figure 16) characterizes the level of features represented within a patch. Sharp features are often the most distinguishing characteristics of objects. As a simple example, the corners of a cube are better for matching than the flat centers of faces. Thus, for each patch we determine the total feature length by first analyzing the edges between adjacent faces and marking those that have a dihedral angle greater than a defined angle (we use 45° in all of our examples). We then sum the lengths of all such feature edges in a patch, and assign a weight for a prior p as the ratio of this sum over the maximum from all of the priors.

$$w_{feat}(p) = \left(\frac{\sum length_{feature\_edge}(p)}{max(total\_length(p_1)..total\_length(p_n))} \right)^2 \tag{3.5}$$

We use a quadratic because it allows better variability of the weights based on the magnitude of feature density near a sample. Finally, for objects that do not contain any features (i.e., in the case of a sphere), all weights are set equally to 1.0.

Ambient Occlusion Weighting

The ambient occlusion weight w_{occl} (shown in Equation 3.6 and illustrated in Figure 16) characterizes the degree to which a sample is occluded from the outside

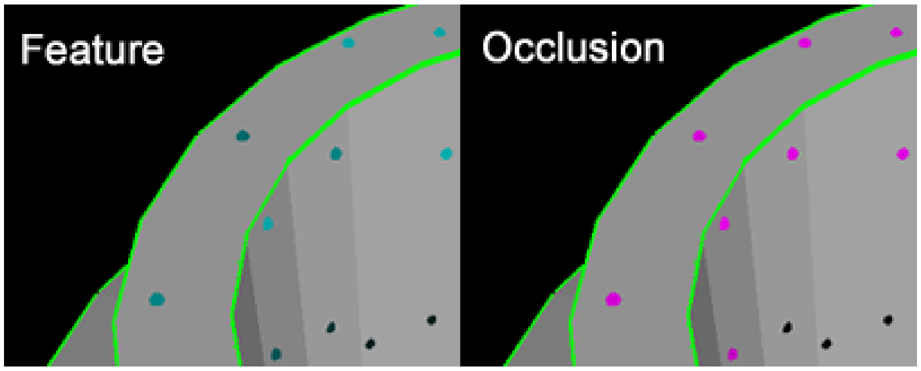


Fig. 16. Feature density and ambient occlusion weights. Lighter colors represent higher weights.

environment. Since the point cloud is collected using line-of-sight sensors from locations surrounding the cluttered scene, we assume that areas of high ambient occlusion are less likely to be sampled and thus are less likely to be matched. As an example, a sample on the inside center of a long tube is less likely to be seen than one on the outside. Thus, w_{occl} measures how likely a sample is to be chosen based on its visibility.

$$w_{occl}(p) = \sqrt{\frac{count(rays_{visible}(p))}{count(rays_{visible}(p)+rays_{not\_visible}(p))}} \tag{3.6}$$

For each model, a bounding sphere of radius τ is computed and uniformly sampled using Hammersley points (using a fixed sample size). A standard ray tracing procedure is then used to shoot rays from each prior sample μ to the samples on the bounding sphere. The weight for a given prior p is the percentage of these rays that hit the sphere (i.e., the fraction of directions from which the sample can be seen from at least τ distance away). We use a square root function for this weight to moderate the effect of its measure.

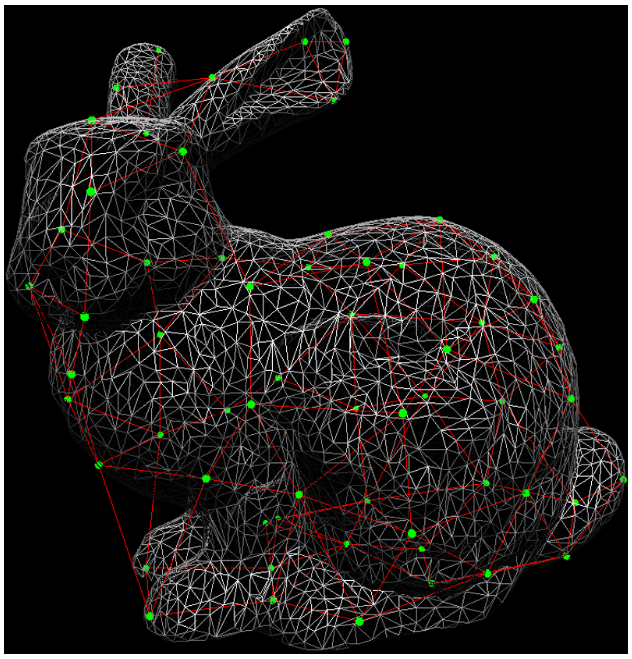
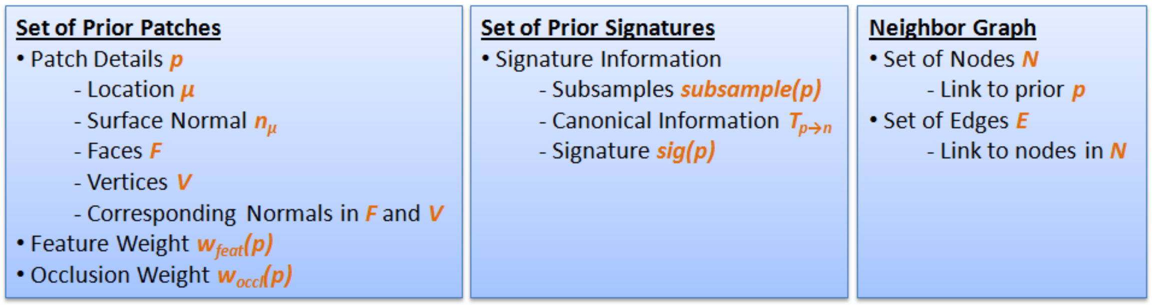


Fig. 17. An example neighbor graph (a green node corresponds to a prior sample and a red edge connects two priors whose support spheres overlap in space).

b. Graph Construction

The next step in our setup phase determines the relationship between different priors, thereby representing the structural behavior across an object. We construct a *neighbor graph* $G = (N, E)$ where the nodes N correspond to the individual prior samples from P. Two nodes n_i and n_j are connected with an edge e_i if the defining support spheres for the two priors overlap. This graph will be used by the matching algorithm to ensure an unambiguous surface definition. An example graph construction is shown in Figure 17.

The result of the setup process (organized in Figure 18) is a collection of prior patches that accurately describe both the local behavior of the model's surface (the individual prior signatures), as well as the interconnectivity of these different local



Set of Prior Patches
- Patch Details p
  - Location μ
  - Surface Normal n_μ
  - Faces F
  - Vertices V
  - Corresponding Normals in F and V
- Feature Weight $w_{feat}(p)$
- Occlusion Weight $w_{occl}(p)$

Set of Prior Signatures
- Signature Information
  - Subsamples $subsample(p)$
  - Canonical Information $T_{p \to n}$
  - Signature $sig(p)$

Neighbor Graph
- Set of Nodes N
  - Link to prior p
- Set of Edges E
  - Link to nodes in N

Fig. 18. The various elements that are constructed during the setup stage for use during the recognition and reconstruction stages.

behaviors across the global structure of the object (the prior neighbor graph). These elements will be the basis for the recognition and reconstruction process explained in the next section.

2. Phase II: Object Recognition

Our primary motivation is to identify and reconstruct objects within a larger cluttered environment. Our recognition algorithm centers on a predictive modeling technique using the graph of prior patches. This recognition phase (described in Algorithm 2) takes as input a point cloud dataset consisting of a set of oriented points. We then try to identify instances of objects in this dataset by matching one of the provided prior models.

a. Initial Matching

To begin reconstructing an object, we must first identify a possible object within the point cloud. The first step of this recognition phase searches the dataset to find a high likelihood local surface match, and then iteratively fits patches to surrounding areas of less certainty.

Algorithm 2

```
1.  $matchFound =$ **false**;
2.  **repeat**
3.      **for** $o \in O$
4.          $\chi = PickRandomSample(o)$;
5.          $c = ConstructCloudPatch(\mathcal{C}, \chi, \delta, \alpha)$;
6.          $T_{c \to n} = NormalizePatch(c)$;
7.          $sig_c = BuildSignature(c, \omega)$;
8.
9.          **for** $p \in P$
10.             $score_{root} = GetRootScore(p, c)$;
11.             **if** $score_{root} < tol_{root}$
12.                **then** $Discard(c)$ and $continue$;
13.
14.             $score_{ring} = GetRingScore(p, c, r_{max})$;
15.             **if** $score_{ring} < tol_{ring}$
16.                **then** $Discard(c)$ and $continue$;
17.                **else**  $matchFound =$ **true** and $break$;
18. **until** $matchFound ==$ **true**;
19.
20. $p_{fitted} = FitPatch(\mathcal{C}, p, c, T_{p \to n}, T_{n \to c})$;
21. $RefinePatchFitting(\mathcal{C}, p_{fitted})$;
```

Searching

To efficiently and evenly search a point cloud $\mathcal{C}$, we use an octree (an example is shown in Figure 19). The octree provides both an organization of the data into a grid for uniform sampling, and an efficient tool for neighbor searching. Our sampling process keeps a list of the active octree leaf nodes O (i.e., those with unselected point samples), and in each iteration a single point cloud sample χ is selected from each octree node $o \in O$. A node is removed from O when it no longer has any unselected points.

For each selected sample χ, a cloud patch c is captured. All cloud samples residing inside a spherical support region (of radius δ and centered at χ) are captured, and the support angle α is used to filter out (based on normal angle deviation) any potential points belonging to other objects. The cloud patch is then normalized and

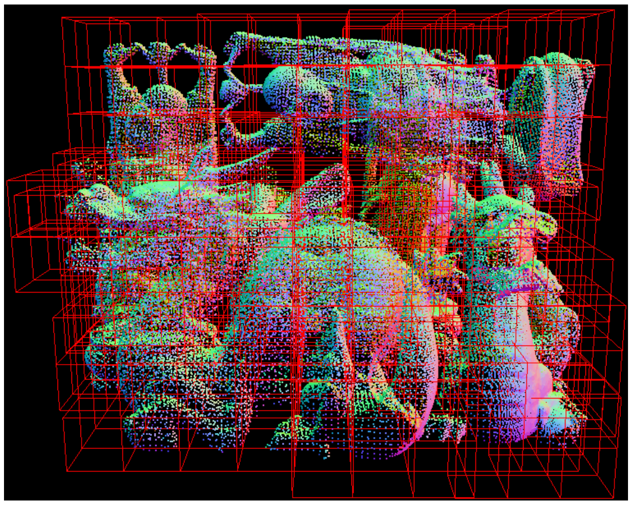


Fig. 19. An octree is used to uniformly search the point cloud dataset.

the canonical information $T_{c \to n}$ used to normalize it is stored. Finally, a geometric moment signature $sig(c)$ is constructed using the cloud points themselves as the corresponding sub-samples. Note that in areas void of data this signature may be a very inaccurate representation, and will thus never be adequately matched to a prior in P.

Root Matching

Once a $sig(c)$ has been generated, the next step is to compare it with the set of prior patches and identify any potential matches. To provide effective recognition, a two phased comparison is employed. In the first phase the cloud patch c is compared to each prior patch $p \in P$ using a weighted similarity score $score_{root}(p, c)$. This weighting scheme incorporates several key components about a prior patch's local behavior, and these weights ensure an adequate patch is chosen to begin the fitting process. This root comparison score (shown in Equation 3.7) takes into account the similarity $s(p, c)$ between the two patches, as well as the weights describing the prior patch's feature density $w_{feat}(p)$ and degree of ambient occlusion $w_{occl}(p)$. The similarity $s(p,c)$ is defined as the distance (we use the Euclidean distance as previously described in Equation 3.4) between the moment signatures $sig(p)$ and $sig(c)$.

$$score_{root}(p,c) = w_{feat}(p) \cdot w_{occl}(p) \cdot s(p,c) \tag{3.7}$$

Given $score_{root}(p,c)$, we then compare this score to a defined tolerance tol_{root}, keeping any items with sufficiently high score, and discarding others. The process for determining tol_{root} will be described later in Section 3.

Often, this one-to-one matching of a single prior may not be sufficient to identify an object since only a locally defined similarity is determined. In some cases an object may be locally similar, but globally different. There are two extensions that allow for better identification of such objects. First, we could make the support neighborhoods substantially larger to capture enough information to distinguish between patches. This approach diminishes the robustness of these patches when matching amongst clutter, as a larger area must now be confidently matched. A second alternative, the approach we use, is to analyze the areas surrounding the patch of interest and incorporate their match quality to further filter potential matches.

Ring Matching

The second phase in our patch comparison analyzes a set of neighbors surrounding each patch being compared to ensure a proper match is obtained. For a prior patch $p \in P$, the graph provides an easy lookup of a neighboring element p_i. For a cloud patch c, finding the corresponding neighboring element c_i within $\mathcal{C}$ is slightly more difficult. The prior's normalization information $T_{p \to n}$ and the inverse of the canonical information found during searching $T_{n \to c}$ are used to define an initial basis. Then the relative oriented offset for each node in the graph (i.e., σ_{prior_offset}) is used to find the corresponding relative position within cloud space σ_{cloud_offset} (i.e., a vector between the two nodes in graph space can be transformed to define a relative position in cloud space). This transformation is described formally in Equation 3.8.

$$\sigma_{cloud\_offset} = T_{n \to c}(T_{p \to n}(\sigma_{prior\_offset})) \tag{3.8}$$

A defined ring size r_{max} is used to control how far from the root node to incorporate additional elements. For all examples provided in this dissertation, an r_{max} ranging from zero to two was used depending upon the level of similarity between the objects being analyzed. For each selected node a support sphere is determined, the intersecting cloud points are found, and a signature is calculated.

The signatures of these nearby elements are then used to determine an overall matching score $score_{ring}(p, c)$ for the patch of interest (shown in Equation 3.9). This score does not incorporate the w_{feat} and w_{occl} weights used during matching of the root patch; the weights help identify a good starting patch, but we want the ring score to solely reflect the similarity between the two surfaces. The ring score is computed as a weighted average based on the ring distance back to the root node where: $s(p_i, c_i)$ is the similarity between the p_i and c_i patches on a ring; r_i is the ring length for the prior p_i, given by the number of edges in the shortest distance to the root node $r_{root} = 0$; the summations are taken over the set of priors such that a prior p_i's distance must be less than r_{max} from r_{root}. The weights falloff exponentially based on ring distance, and 0.5 provided a good base that worked well in all of our examples.

$$score_{ring}(p, c) = \frac{\sum\limits_i s(p_i, c_i) \cdot 0.5^{r_i}}{\sum\limits_i 0.5^{r_i}} \tag{3.9}$$

The ring score is then compared to a tolerance threshold tol_{ring}. If the patch passes, it next needs to be fit to its respective position in the point cloud. A prior's oriented position σ_{prior} and normalization information $T_{p\to n}$, along with the inverse of the canonical information $T_{n\to c}$, are used to determine this initial fitting σ_{fitted} within $\mathcal{C}$ (described in Equation 3.10).

$$\sigma_{fitted} = T_{n\to c}(T_{p\to n}(\sigma_{prior})) \tag{3.10}$$

Finally, an Iterative Closest Point (ICP) algorithm [10][11] is used to refine this initial fitting σ_{fitted}. ICP allows for the refinement between two datasets by iteratively trying to minimize the error between them. The result is the starting element for recognizing and reconstructing the remaining structure of the object within $\mathcal{C}$.

b. Structure Recognition

In the previous subsection we described the process for matching an initial high likelihood prior patch to a region within the point cloud. Based on this initial fitting and through the use of the rules defined as part of the graph, the process can quickly expand outwards, iteratively matching and fitting patches until a complete boundary of the object has been defined. This stepwise procedure is the basis for recognizing an object (described in Algorithm 3). Figure 20 shows an example of this process.

The recognition algorithm uses a breadth first search of the prior graph (beginning at the node of the matched root prior) to fit the remaining priors. This process fits patches to both regions with samples, as well as occluded regions missing samples, to ensure a complete object definition.

Validity Testing

As each node in the graph is visited (with corresponding prior p_{sel}), several tests must be performed to ensure that a surface is properly matched and fit to the corresponding cloud patch c_{sel}. The first test evaluates whether there is an adequate number of point samples to perform the process. The number of cloud points contained in c_{sel} is compared to a predefined threshold *min_sample_sz* (i.e., a fixed percentage of the prior sub-sample size; we used 50% of the sub-sample size for all examples in this dissertation).

Algorithm 3

```
1.  $FindNeighborsToFit(p_{fitted}, nodesToFit)$;
2.  repeat
3.      $p_{sel} = GetNextNode(nodesToFit)$;
4.      $c_{sel} = FindCloudPatch(\mathcal{C}, p_{sel}, T_{p \to n}, T_{n \to c})$;
5.      $FindNeighborsToFit(p_{sel}, nodesToFit)$;
6.
7.      if $GetNumSamples(c_{sel}) < min\_sample\_sz$
8.        then $FitPatch(\mathcal{C}, p_{sel}, c_{sel}, T_{p \to n}, T_{n \to c})$;
9.             $continue$;
10.
11.     $skew_p = GetThirdCentralMoment(sig_{p_{sel}})$;
12.     $skew_c = GetThirdCentralMoment(sig_{c_{sel}})$;
13.     $skew_{diff} = abs(skew_p - skew_c)$;
14.     if $skew_{diff} > tol_{skew}$
15.       then $FitPatch(\mathcal{C}, p_{sel}, c_{sel}, T_{p \to n}, T_{n \to c})$;
16.            $continue$;
17.
18.     if $GetRootScore(p_{sel}, c_{sel}) < tol_{root}$
19.       then $exit$ as an invalid match was found;
20.
21.     $p_{fitted} = FitPatch(\mathcal{C}, p_{sel}, c_{sel}, T_{p \to n}, T_{n \to c})$;
22.     $RefinePatchFitting(\mathcal{C}, p_{fitted})$;
23. until $nodesToFit == \emptyset$;
```

If enough samples exist, then a second test evaluates the distribution of the corresponding normalized points as compared to the normalized sub-samples in the prior patch to ensure the data is not skewed (i.e., allowing for a feasible match between the two). This test simply evaluates the difference between the third central moments from the two signatures $sig(p_{sel})$ and $sig(c_{sel})$. If either of these tests fail, then p_{sel} is simply fit using the default transformation σ_{fitted} without ICP refinement because this region involves a potentially occluded surface (or at least a partially occluded

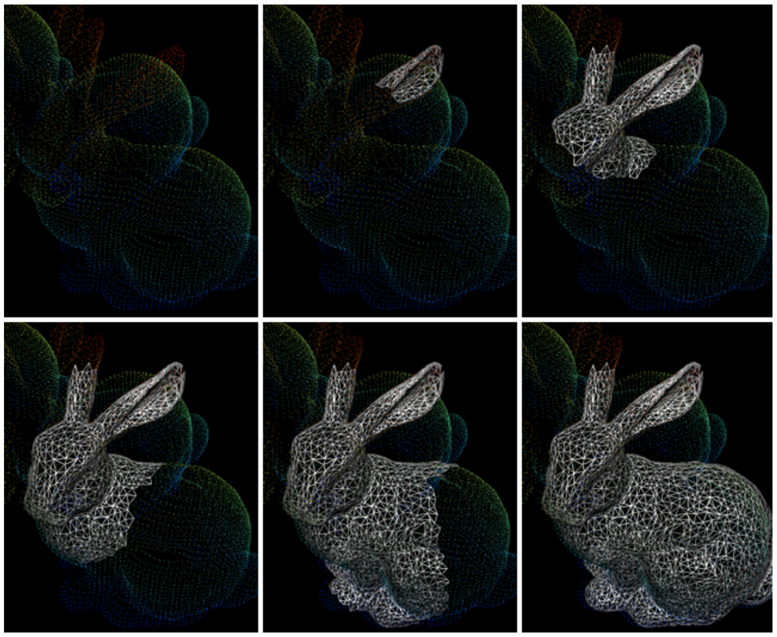


Fig. 20. Several iterations of the patch fitting process.

surface lacking enough data to make a valid determination). If both tests pass then the matching/fitting process continues.

A third test, one that we do not currently perform, could be included to check for samples residing in expected locations only. This test would identify when samples reside in a region where they should not if the algorithm is correctly reconstructing an object (e.g., it might discover samples residing in the interior of an object and alert the matching process of an invalid state).

Handling Matches Versus Errors

If enough samples exist in the region of interest in the point cloud, then the next step matches the signature obtained from these samples $sig(c_{sel})$ to the expected signature of the corresponding node in the graph $sig(p_{sel})$. If a match is found, then it can be fit and refined. The process then continues on to the next queued patch.

If an invalid match is found, then the matching/fitting process has failed. In this case, all fitted patches for this object need to be removed and the point cloud samples returned for later use. By allowing failure during the matching process, objects with similar local surface qualities can be distinguished. As shown later in the results, this provides an effective procedure for only identifying objects of interest within a scene, and can be used to distinguish between similar yet different objects. However, for an object to fail it requires the distinguishing characteristics to be visible during the initial capture process. If these characteristics are entirely hidden, our algorithm has no way of determining a difference and will simply perform a best determination based on the available information.

When a valid match is found, the prior p_{sel} must be fit to the point cloud data. As in the fitting of the matched root patch, an initial estimation of fitting is determined (i.e., using a relative oriented offset σ_{prior_offset} between the two graph nodes, and applying Equation 3.8) and the patch fitting must then be refined.

Iterative Patch Refinement

The initial fitting of each patch is iteratively refined using a weighted ICP algorithm [10][11]. This refinement helps fit the surface patch more closely to nearby point cloud samples, ensuring a better overall fit in well-sampled regions. The patch positioning σ_{fitted} obtained from applying the transformation ensures that a patch begins with a reasonable default fitting (i.e., even in poorly sampled regions). The refinement process then improves the fitting based on the locally available point cloud data. In cases with extreme variation between the point cloud samples and a fitted patch (a distance metric of 20% of the support distance δ was used in all of our examples), we ignore all of these extreme point cloud samples and do not consider them during the ICP process.

Scaling

In some situations it is beneficial to match objects of varying scales (e.g., several balls of different sizes). To address this need, a special stepwise support distance δ_{step} can be used when sampling the point cloud. This stepwise distance allows for sampling at different ranges around a sample (i.e., using concentric spheres) and constructing a scaled patch $\tilde{c}$. Only the samples that reside within a given range's sphere are captured, they are then filtered and normalized, and a signature determined as previously defined in Section 2. For $\tilde{c}$, the canonical information $T_{\tilde{c}\rightarrow n}$ is stored and since the calculated moments are scale invariant, they can be easily matched with the priors in P. The fitting process for $\tilde{c}$ is then performed by using Equation 3.10 and substituting $T_{n\rightarrow\tilde{c}}$ for $T_{n\rightarrow c}$. This simple extension allows the graph to be scale invariant and allows handling objects across differently defined scales (i.e., both smaller and larger) from a defined model.

3. Phase III: Object Reconstruction

Given an object recognized by the previous phase, the goal of the third and final phase is to reconstruct a solid model representation and prepare the dataset for further object recognition (described in Algorithm 4). This multi-phase recognition and reconstruction process is repeated until no more objects are found.

Algorithm 4

```
1.  $\overset{\curvearrowright}{\mathcal{C}} = MergePatchesIntoPointCloud(P_{fitted})$;
2.  $\mathcal{S}_{impl} = GenerateMLSSurface(\overset{\curvearrowright}{\mathcal{C}})$;
3.  $\mathcal{S}_{mesh} = GenerateMarchingCubesSurface(\mathcal{S}_{impl})$;
4.
5.  **if** $DoesUserAcceptReconstruction(\mathcal{S}_{mesh})$
6.      **then** $KeepModel(\mathcal{S}_{mesh})$;
7.           $b = FindOrientedBoundingBox(\mathcal{S}_{mesh})$;
8.           $RemoveCloudPointsInsideBox(\mathcal{C}, b)$;
9.      **else** $Discard(\mathcal{S}_{mesh})$ and start over;
```

a. Solid Model Construction

Once the full boundary of the object has been identified and fitted with a set of priors P_{fitted}, the next step in our process is to create a solid model representation by joining the fitted priors $p_{fitted} \in P_{fitted}$ together to form a watertight and smooth surface $\mathcal{S}$. The sub-samples from each p_{fitted} are combined into an integrated point cloud $\mathscr{P}$. Note that as described by Gal et al. [45], these sub-samples are high quality with accurate normals since they were generated from the original input model. Thus, we use these sub-samples as the basis of the reconstruction process to follow.

Next, a projection-based Moving Least Squares (MLS) reconstruction [4] is performed on $\mathscr{P}$. MLS is beneficial because it provides a smooth surface approximation, can handle noisy data, and is capable of approximating a continuous surface, even with noisy data (e.g., in the case of overlapping patches that may not perfectly align, causing somewhat noisy and/or discontinuous samples; these are easily smoothed using this procedure). There are many other variants of MLS that could also be used [102]. A detailed description of the MLS process we use is provided in Appendix B.

The result of the MLS process is an implicit representation $\mathcal{S}_{impl}$ that smoothly and continuously defines the surface. It is a concise representation, but one that is not easily rendered or used by many applications (e.g., simulations). Thus, a more portable representation is desired. We use Marching Cubes to contour this MLS implicit surface [5]. Marching Cubes is a simple and commonly used approach that calculates a polygonal representation from an implicit definition.

Marching Cubes subdivides the space into a voxel grid, and then iteratively traverses and evaluates each voxel's contribution to the surface. An iso-value is determined at each corner of a voxel and tested to see if it is inside/outside the surface. If a voxel is found to intersect a surface, then the points of intersection along each

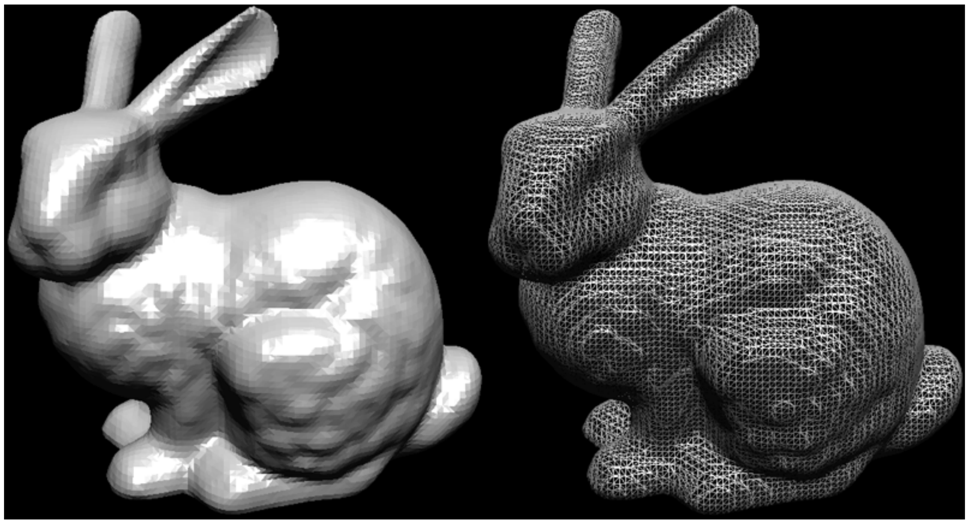


Fig. 21. A final contoured model obtained from integrating the fitted patches.

edge can be found through interpolation. A set of rules (i.e., a pre-calculated array of configurations) then defines the correct set of facets to model the surface within the voxel. The original Marching Cubes algorithm [5] contained several ambiguities which may result in a discontinuous surface definition across two adjacent voxels. As a result, different variants to resolve these ambiguities [103][104] and alternative approaches to voxel interpolation [105] have been proposed.

Marching Cubes generates a piecewise smooth surface that approximates the implicit surface, but may have trouble when trying to reconstruct sharp features and may also result in large meshes. Alternative contouring approaches exist that are both feature-preserving and can generate better quality meshes. Some alternative approaches include Extended Marching Cubes [106], Dual Contouring [107][108], Dual Marching Cubes [109], and Unconstrained Isosurface Extraction on Arbitrary Octrees [110]. A robust MLS fitting [40] provides a different alternative for obtaining sharp features from the implicit surface.

The final output of this step is a meshed representation $\mathcal{S}_{mesh}$ of the recognized object. Figure 21 shows an example contoured model obtained from the patch fitting shown earlier in Figure 20.

b. Final Processing

The final step in our process is to remove the contributing point samples for the constructed object. In addition, the tolerances used during the matching process can be adjusted based on the overall progress.

Contributing Sample Removal

Our approach functions in both fully automated and semi-automated modes. During semi-automated reconstructions, the user can decide whether to keep a reconstructed model or discard it in the case of an undesired fitting. In both modes, after the contoured representation has been generated and deemed acceptable, the final step is to remove the point cloud samples that contributed to this construction to avoid negatively influencing future object recognitions and reconstructions. Since we assume the environment is composed of more than a single object, these points must be removed so they are not noise for later fittings. We perform this operation by first finding an oriented bounding box b around the constructed object and use the octree discussed previously to efficiently find the set of points that reside inside this box.

Depending on the level of clutter in the scene, we allow two options for selecting the points to remove:

- Remove all points that lie inside b.
- Find the iso-value of each point sample inside b (i.e., using MLS to calculate the distance from the point to the surface), and determine if it lies inside or

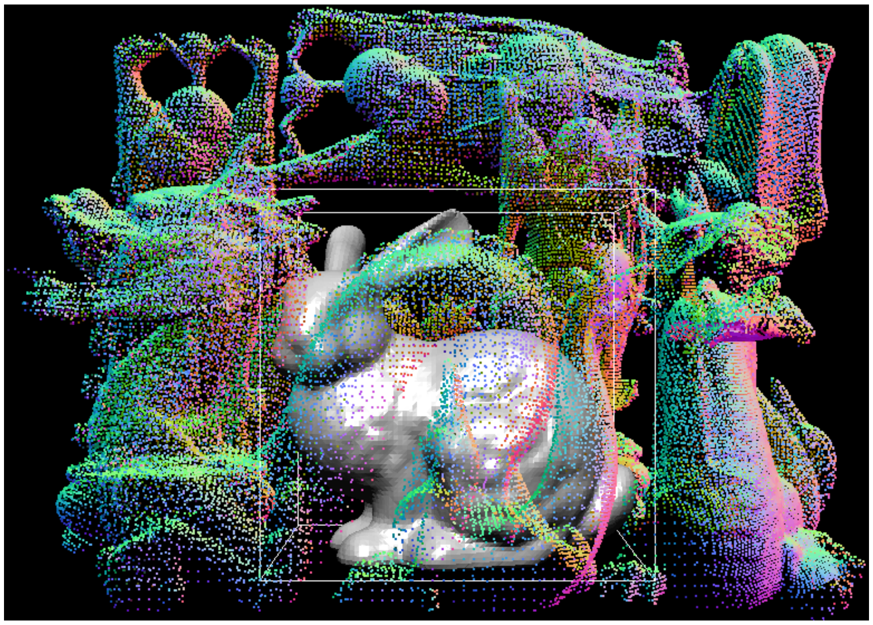


Fig. 22. A bounding box is used to locate the contributing point samples that should be removed once an object has been reconstructed.

within a certain distance from the surface. If so, then the point is discarded, and if not the point is kept for use by later iterations.

The first approach is an efficient solution, but may remove excess points that do not belong to the object (i.e., if part of another object intersects b). The second alternative is much less efficient, but provides a more accurate solution. We use the first approach in all of our examples. Figure 22 shows an example bounding box used for removing the contributing samples.

Incremental Recognition

After the points have been removed, the process then begins again to find and reconstruct another object. As the iterative search process continues, it gradually becomes more efficient as the number of points being analyzed gets reduced. However, the degree to which the samples represent a remaining object's surface may decline. Our approach begins by setting the matching tolerances tol_{root} and tol_{ring} high (i.e., requiring a better matching score), and then iteratively performs the recognition process, decreasing the tolerances over time. The initial tolerances are determined based on a user-guided trial and error process where the user decides when a valid match has been obtained and the score gets recorded. After performing a series of these tests, the maximum identified scores provide a good starting estimation for the tolerance values. More automated approaches could be investigated and used to reduce the front-end work required.

These initially established tolerances are reduced over time for more probable matching. If at any point, greater than 50% of the remaining points in the point cloud have been marked as selected but a sufficient match is not yet found, then we assume there is a lack of evidence to identify an object. Rather than aimlessly moving forward, the matching process is stopped and reset, the matching tolerances are each decreased by 5%, and the matching process can then move on and match against slightly less strict criteria.

This stepwise process allows matching of objects in an iterative fashion where those easily identified are found and handled first, and then over time the more difficult or harder to identify objects are found. Note that the reconstructions of those objects identified later in the process (i.e., using lower tolerances) may not be quite as accurate due to the lower quality matching requirements and reduced set of point cloud samples. Once there are not enough points left to identify an object, the process is terminated and the result is a set of models representing the objects within the captured environment.

Table 1. Details of the Reconstruction Examples

Dataset	**Point Cloud Size**	**# Fit Objects**	**Total Setup**	**Ave. Root Match/Fit**	**Ave. Link Match/Fit**	**Ave. Object Recon.**	**Total Runtime**
Example 1	49,910	24	27.1s	13.1s	3.2s	10.1s	662.2s
Example 2	122,465	17	16.7s	41.1s	6.0s	19.3s	1144.5s
Example 3	49,576	7	16.2s	4.9s	4.7s	29.5s	289.7s
Example 4	38,429	21	15.9s	8.5s	2.4s	21.5s	695.1s
Example 5	94,398	1	67.8s	77.2s	60.9s	9.3s	215.2s
Example 6	143,087	9	15.9s	0.6s	15.2s	12.0s	265.8s
Example 7	80,940	12	22.5s	13.4s	11.1s	29.8s	674.0s
Example 8	170,237	7	26.2s	20.0s	62.7s	53.7s	980.7s
Example 9	278,812	13	15.7s	3.6s	16.0s	7.5s	367.3s
Example 10	198,600	12	18.8s	34.2s	13.4s	30.9s	960.8s
Example 11	642,279	11	21.5s	92.7s	66.9s	35.0s	2161.2s
Example 12	179,644	4	23.0s	47.3s	36.4s	28.5s	471.6s
Example 13	112,095	5	15.6s	2.6s	8.9s	9.8s	122.7s

C. Results

In order to adequately demonstrate the different capabilities of our proposed approach, we provide several examples containing various levels of object clutter and complexity. We experimented with three different methods for generating the underlying point cloud datasets used, and the database of priors consisted of a set of models representative of the objects appearing in the corresponding synthetic and real-world scenes. All results were generated using an Intel Xeon 2.67 GHz CPU with 4GB of memory. Currently, our algorithm implementation performs all computations on the CPU and uses the GPU only for rendering. The dataset details and respective execution times for each example are shown in Table 1.

The first method for evaluation uses an application that simulates the scanning process to build several synthetic datasets. This simulation allows control over the number and density of scans performed, as well as the incorporation of various degrees of noise. Figures 23, 24, 25, 26, and 27 (i.e., Examples 1-5) show the point clouds and corresponding reconstructions generated using our approach. The examples shown in these figures illustrate many different capabilities of our algorithm:

- Example 1 demonstrates the ability of our algorithm to distinguish amongst the clutter and identify different object types, and reconstruct each accordingly.
- Example 2 demonstrates the ability of our algorithm to distinguish between similar yet different objects. Objects such as those shown can complicate the partial recognition process. Our approach uses a combination of the different matching weights and the trial and error process to correctly identify and reconstruct each distinct object.
- Example 3 demonstrates the ability of our algorithm to work with noisy data.

Gaussian noise was added to all samples in the point cloud where one standard deviation corresponds to 5% of the largest dimension of the object. Since our algorithm heavily relies on correctly matching the surface patches, if the noise becomes too excessive the overall process can break down.

- Example 4 demonstrates our algorithm working with limited data and heavy occlusion. Here the point cloud consists of a single scan taken from a single fixed viewpoint and view direction. Our algorithm is able to iteratively identify and correctly reconstruct each of the objects. Initially some of the objects with more point samples are handled, and gradually over time those with fewer samples (i.e., due to occlusion) are handled based on the available information.
- The objective of Example 5 is to locate the bunny amongst a larger pile of more geometrically complex objects. In this example, the initial root search time is slower because finding an accurate identifiable match for the object is more difficult. In addition, fitting of the linked patches is slower due to lengthy convergence times of the ICP algorithm. However, once the patches have been fit a high quality reconstruction is obtained as shown in the figure.

The second method for evaluation uses a standard laser scan (a NextEngine 3D Scanner HD) for generating the point clouds. Objects were placed on a turntable and scanned from multiple directions. Figures 28, 29, and 30 (i.e., Examples 6-8) show the original scenes, the corresponding point clouds, and the generated reconstructions using our approach.

The third method for evaluation uses a photo-based approach for generating the point clouds. A series of digital photographs was taken from different locations and angles, and the Bundler software package [8] was used to produce a reconstruction of the camera locations and sparse scene geometry. The output from Bundler was then

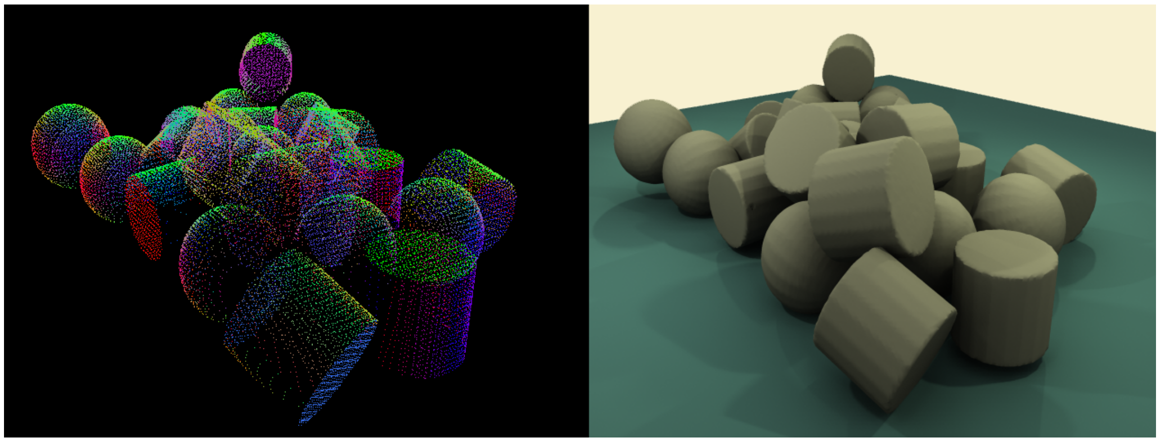


Fig. 23. A synthetically generated example that demonstrates the identification and reconstruction of multiple objects in clutter (Example 1).

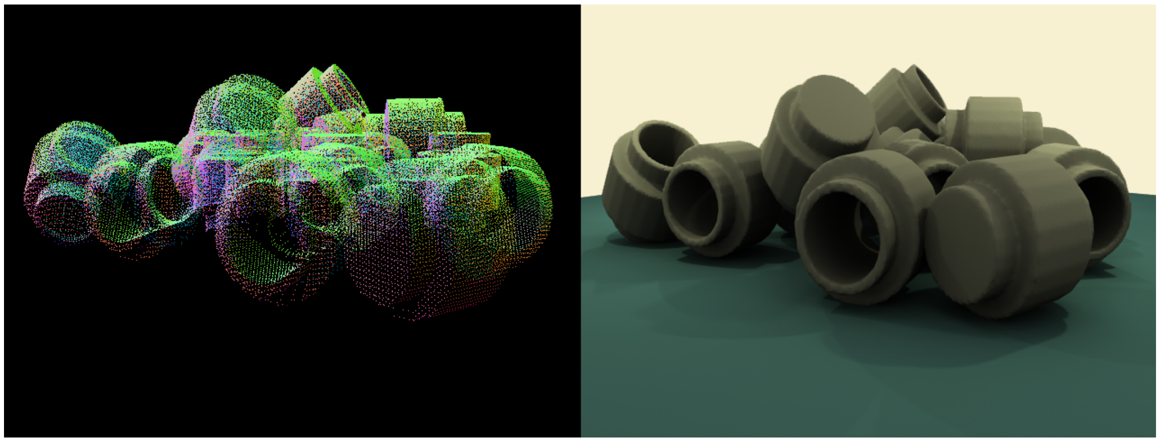


Fig. 24. A synthetically generated example that demonstrates our approach's ability to handle locally similar, but globally different objects (Example 2).

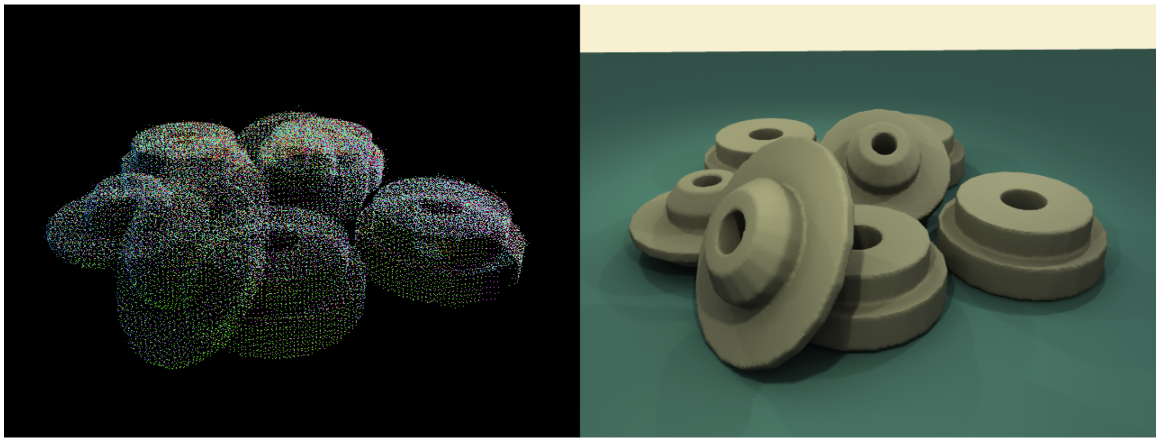


Fig. 25. A synthetically generated example that demonstrates our approach's ability to handle noisy data (Example 3).

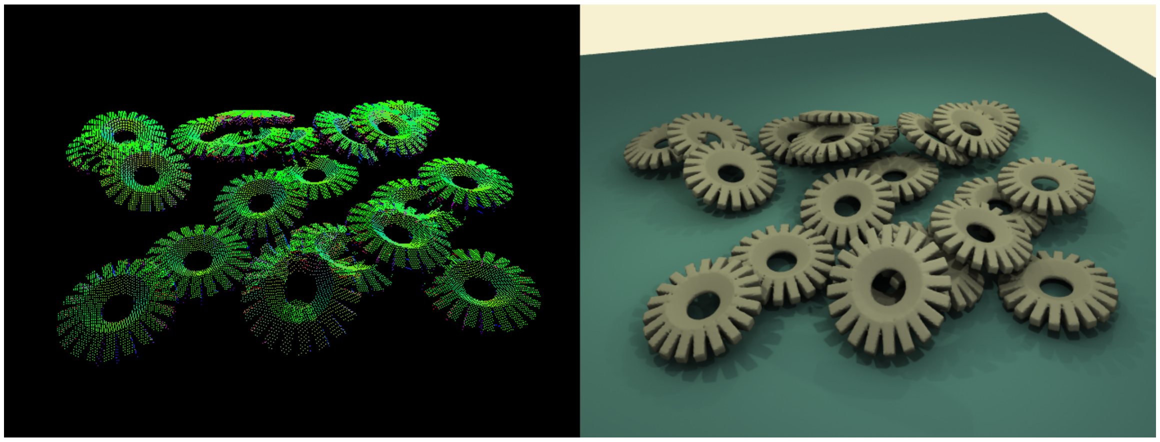


Fig. 26. A synthetically generated example that demonstrates our approach's ability to handle heavy occlusion (Example 4).

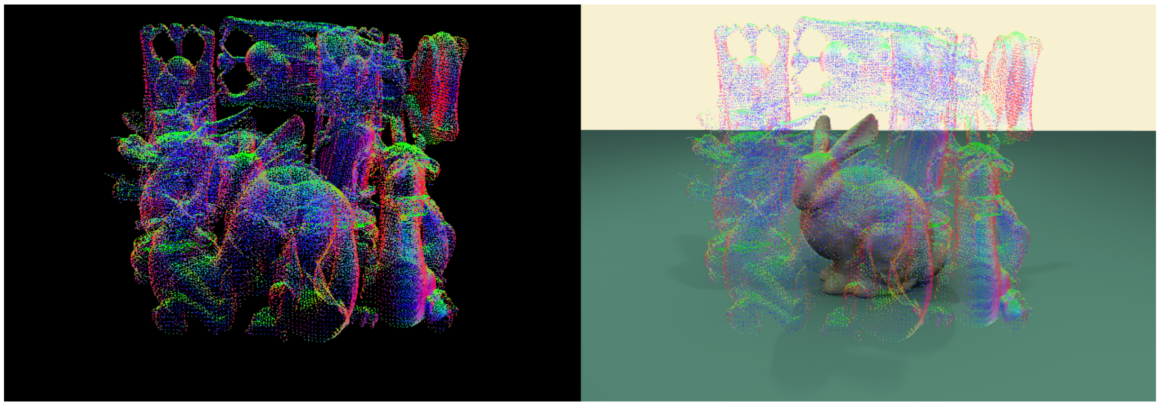


Fig. 27. A synthetically generated example that demonstrates our approach's ability to identify an object of interest surrounded by heavy clutter (Example 5).

passed to the CMVS software package [9] to reconstruct the sampled 3D structure from the set of collected images. The final output of this process is a point cloud representation of the scene. Figures 31, 32, and 33 (i.e., Examples 9-11) show the original scenes, the corresponding point clouds, and the generated reconstructions using our approach.

Figures 34 and 35 (i.e., Examples 12-13) show two special applications of our approach using real-world data. Figure 34 shows the ability of our algorithm to recognize and reconstruct similar yet different objects. In this example a ring size of 2 and a smaller support sphere were used to allow proper distinguishability between the two different objects. Figure 35 shows an example of our approach using a single input model to match multiple objects at different scales. In this example, the method as defined earlier was used to match the two differently sized objects.

Finally, in order to evaluate the robustness of our approach with respect to dataset resolution, we performed a series of tests with datasets of varying reduced sizes. We began with the dataset from Example 7 (Figure 29), and then randomly

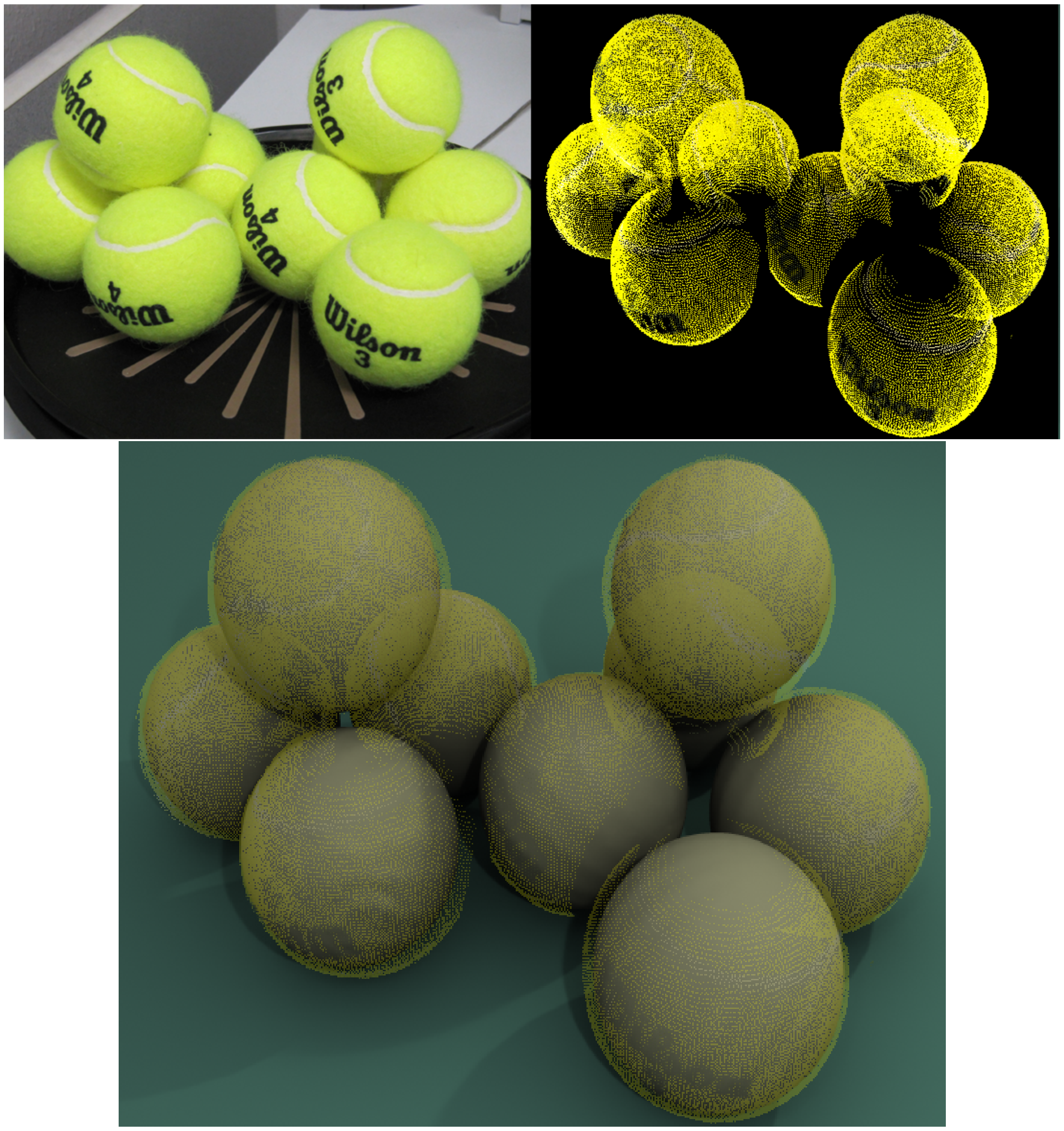


Fig. 28. A laser scan-based point cloud and corresponding set of reconstructed models (Example 6).

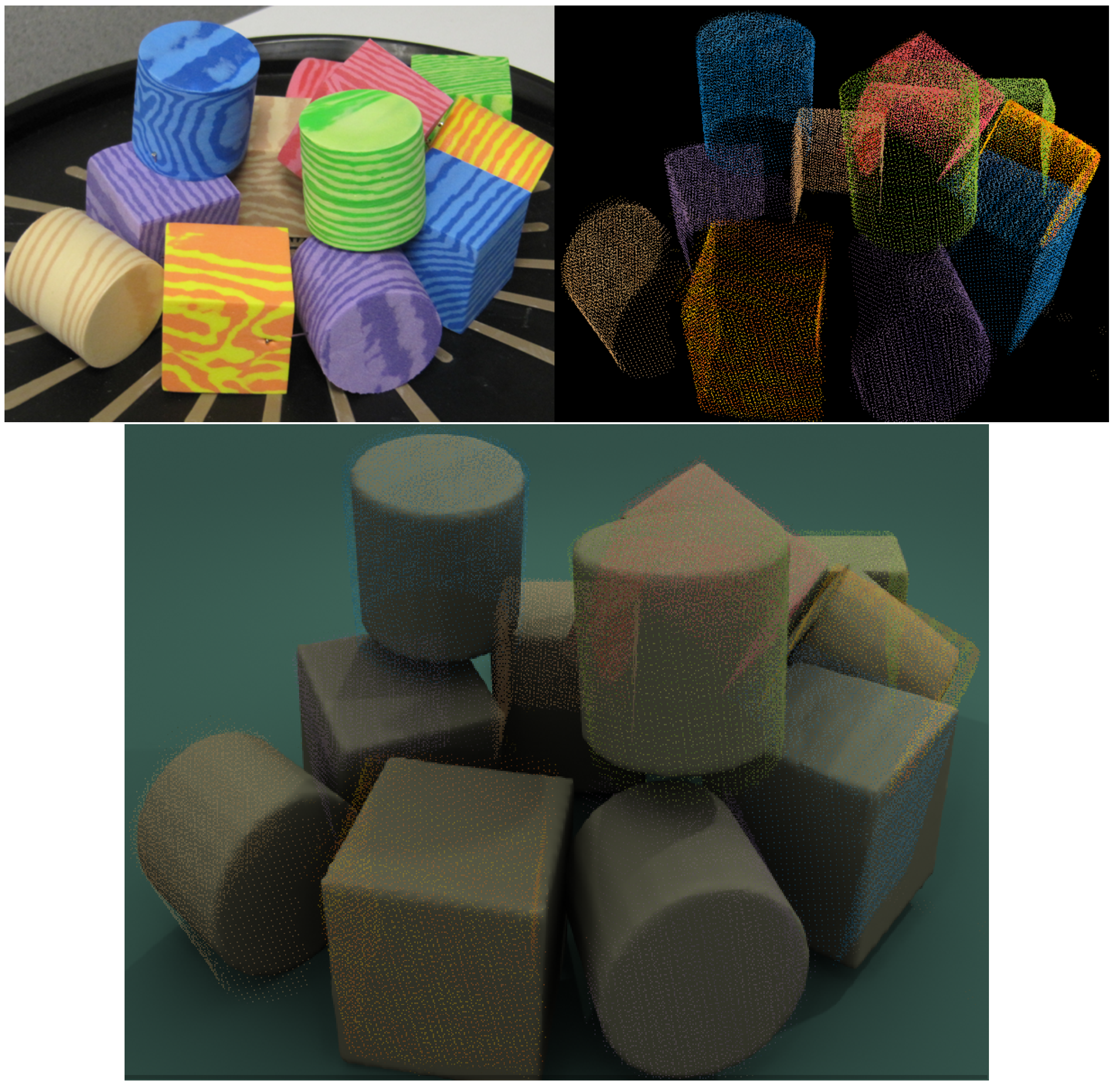


Fig. 29. A laser scan-based point cloud and corresponding set of reconstructed models (Example 7).

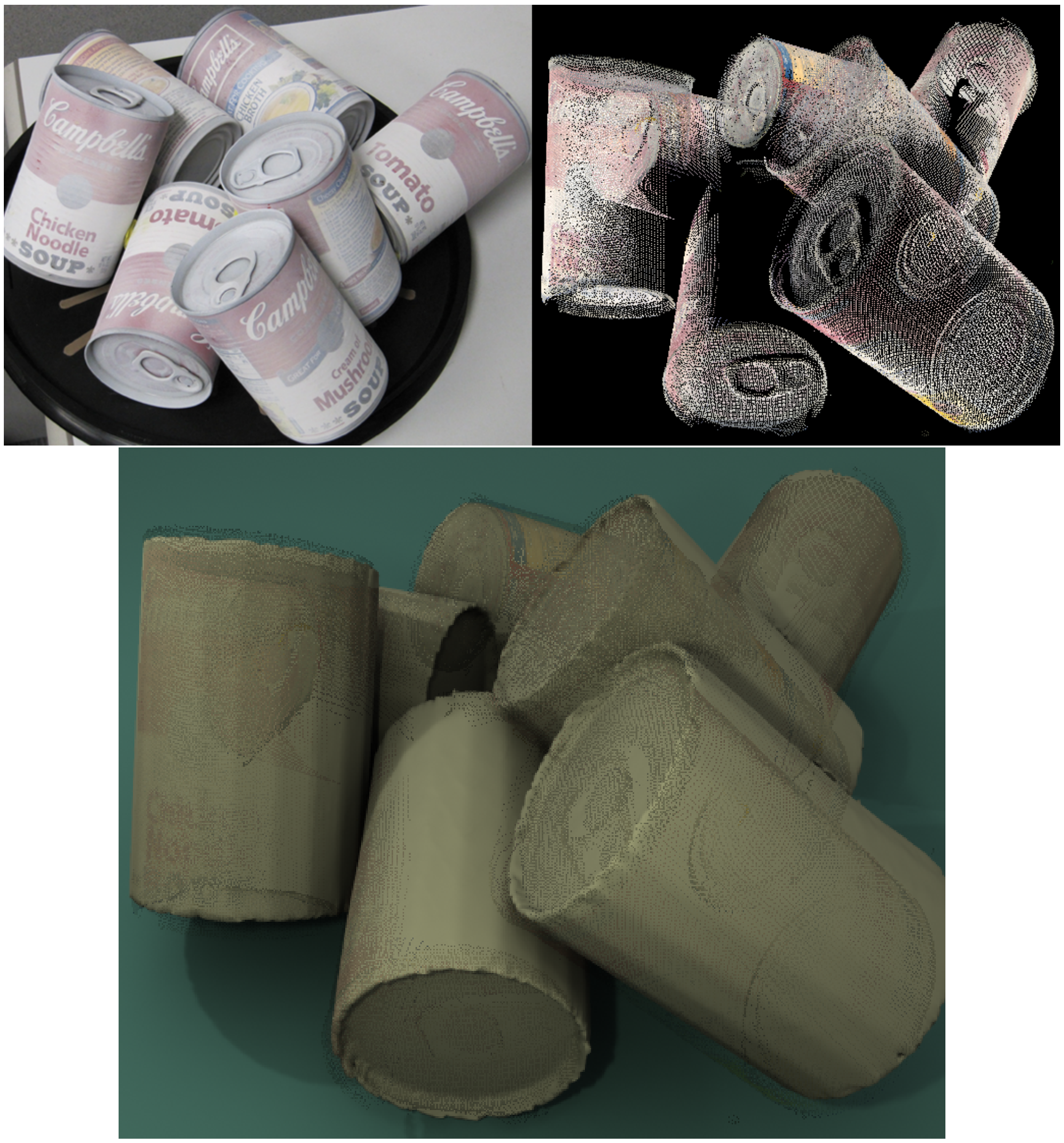


Fig. 30. A laser scan-based point cloud and corresponding set of reconstructed models (Example 8).

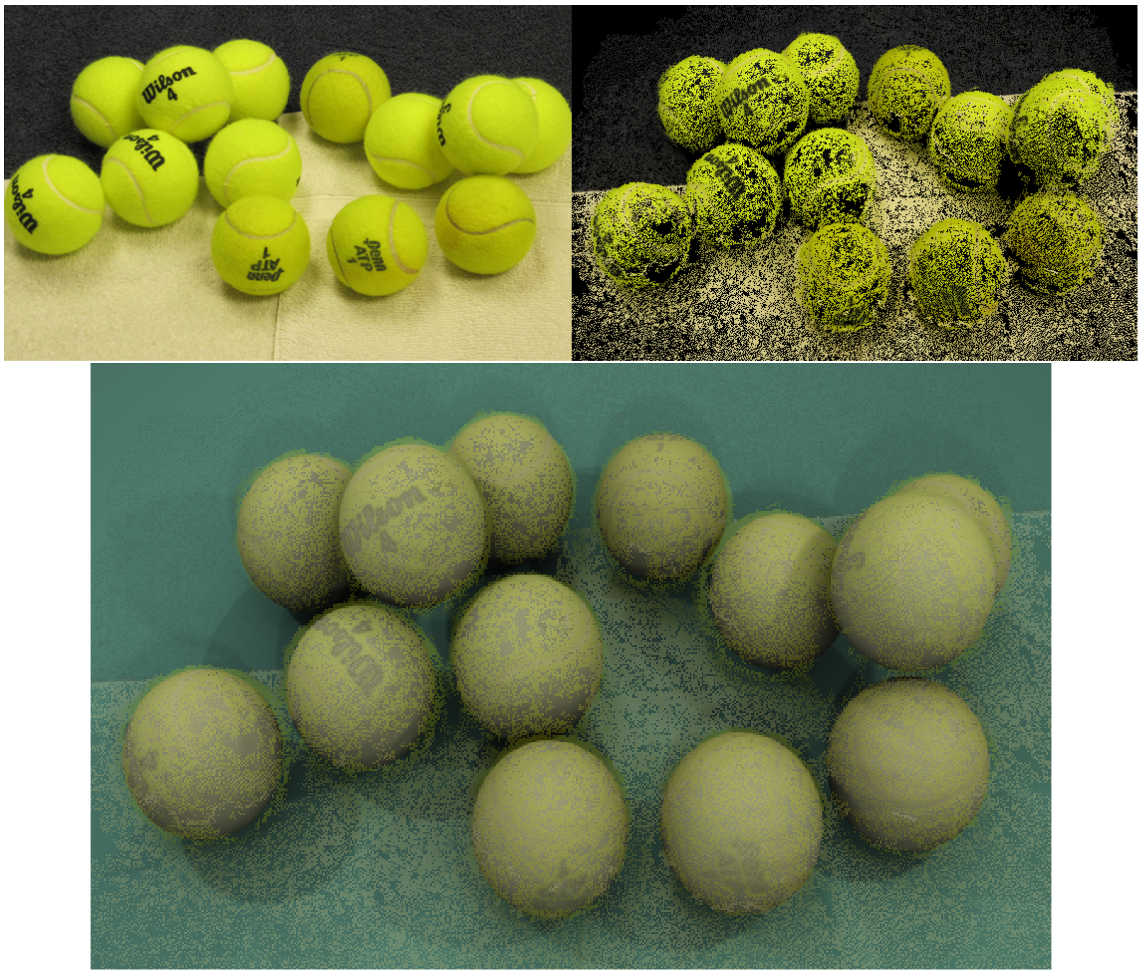


Fig. 31. A photo-based point cloud and corresponding set of reconstructed models (Example 9).

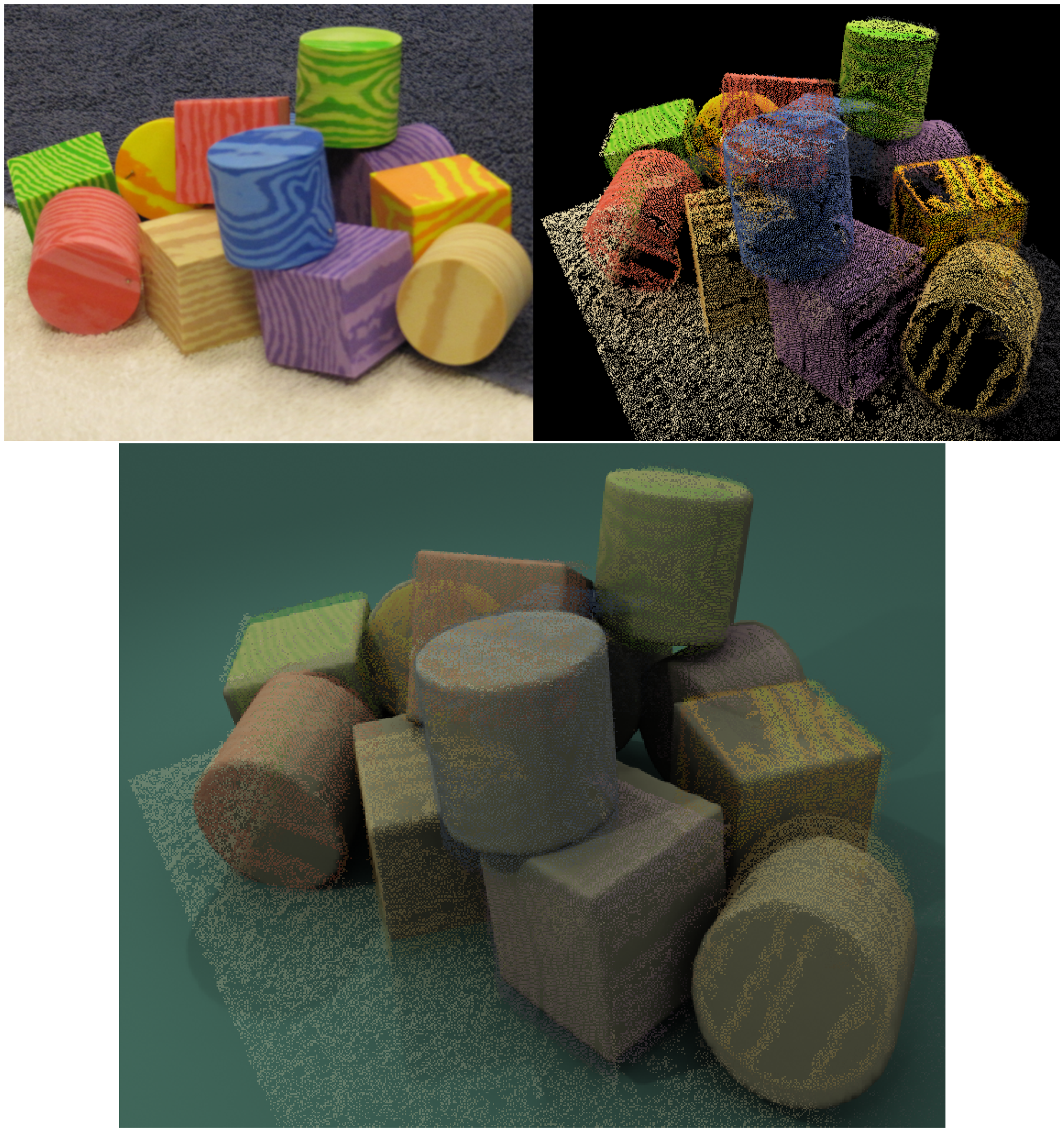


Fig. 32. A photo-based point cloud and corresponding set of reconstructed models (Example 10).

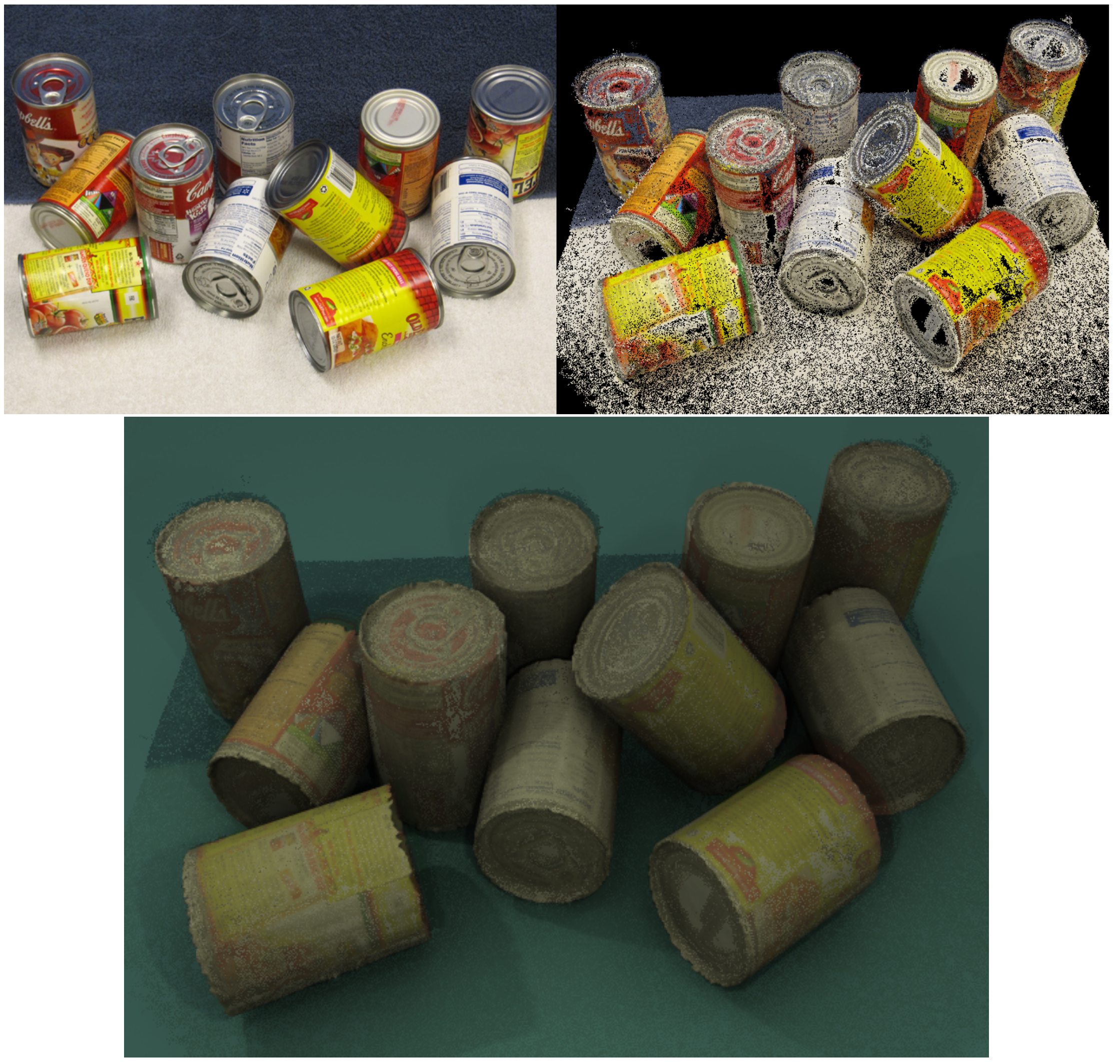


Fig. 33. A photo-based point cloud and corresponding set of reconstructed models (Example 11).

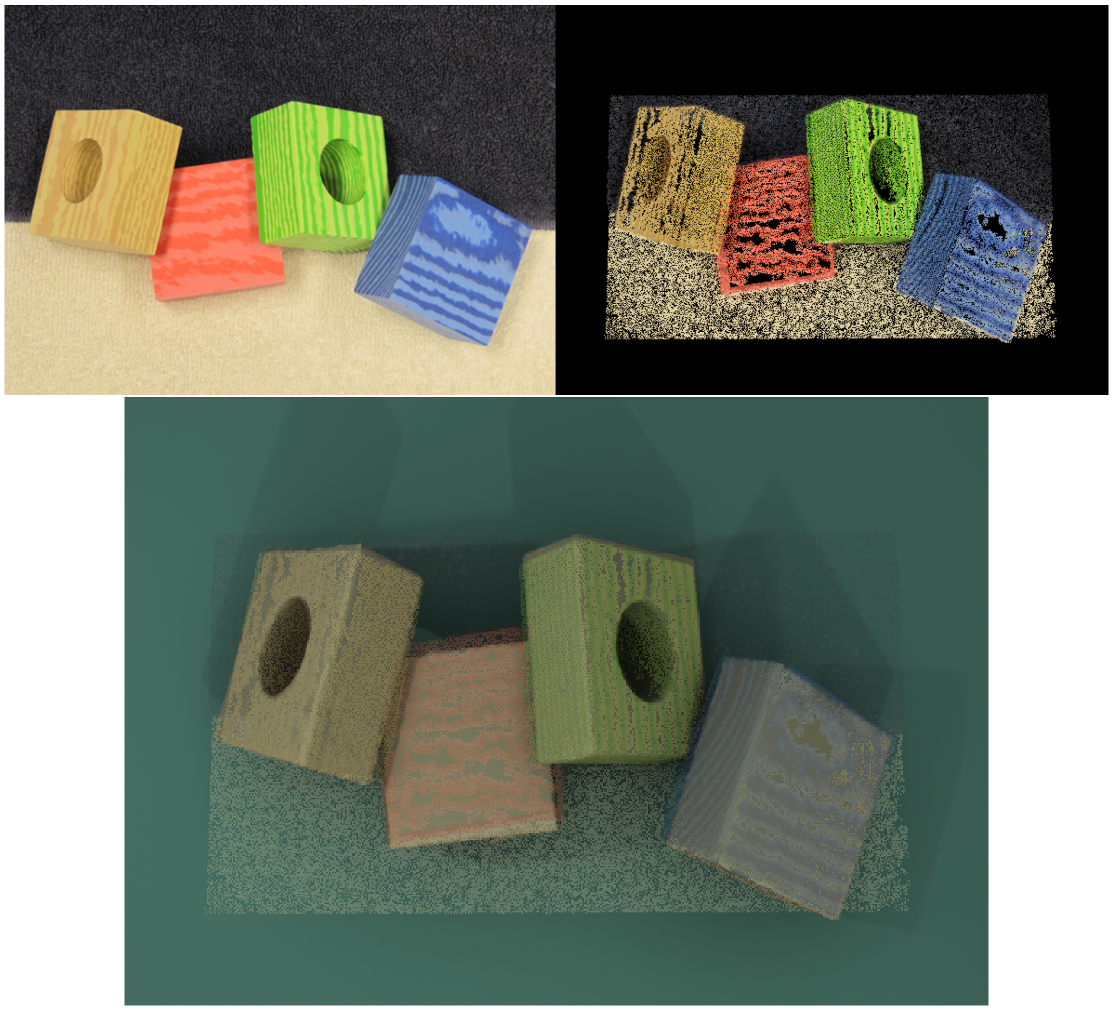


Fig. 34. A photo-based example illustrating the handling of similar, but differently structured objects (Example 12).

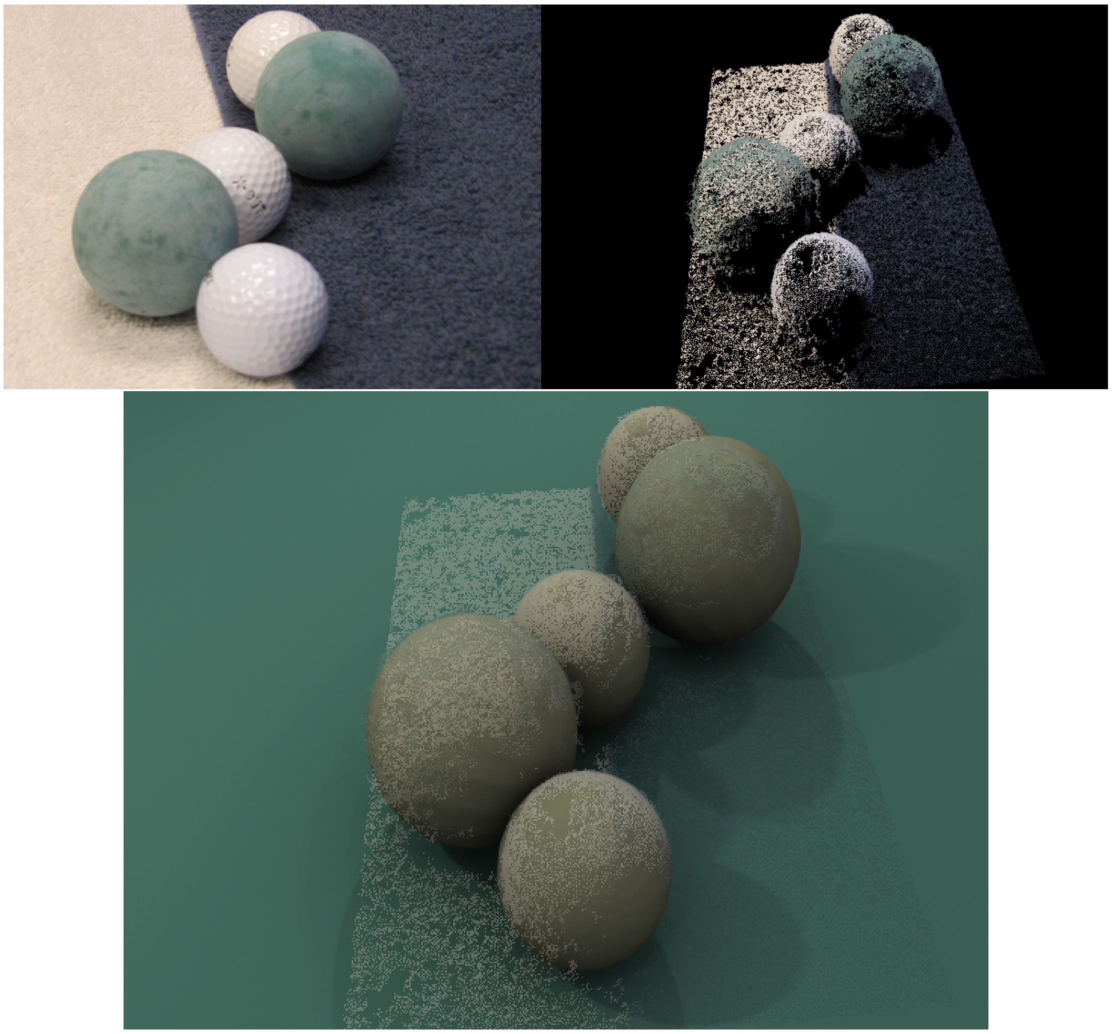


Fig. 35. A photo-based example illustrating the handling of similar, but differently scaled objects (Example 13).

Table 2. Details of the Reduced Reconstructed Datasets

Dataset Size	**Point Cloud Samples**	**Total Runtime**	**Total Objects Found**	**Errors**
100%	80,940	653.2s	12	0
50%	40,470	595.5s	12	6
25%	20,235	567.3s	12	5
12.5%	10,117	485.8s	12	6
6.25%	5,058	516.7s	12	6
3.125%	2,529	366.5s	9	9

reduced it by 50% over several iterations. Our reconstruction process was then performed using a fixed set of parameters in a semi-automated fashion so the user could validate when an identification and reconstruction was successful. The details for each dataset are shown in Table 2 and the final results are shown in Figure 36. Each dataset is half the size of the previous dataset, and as data is reduced, more errors occur. An error is defined as the matching and fitting of a wrong object, fitting of an object in a wrong orientation, or not recognizing an object at all.

At 100%, our approach functions without any errors. At 50%, it is able to handle the more heavily sampled objects on the outer areas easily, but begins to have trouble on some of the inner objects that are more occluded and have fewer samples. In this example, a couple of objects are misidentified and others reconstructed in an incorrect orientation. As data is reduced in already poorly sampled regions, the matching process breaks down and can misidentify the starting element for an object (i.e., these objects are typically matched later in the process with lower tolerance

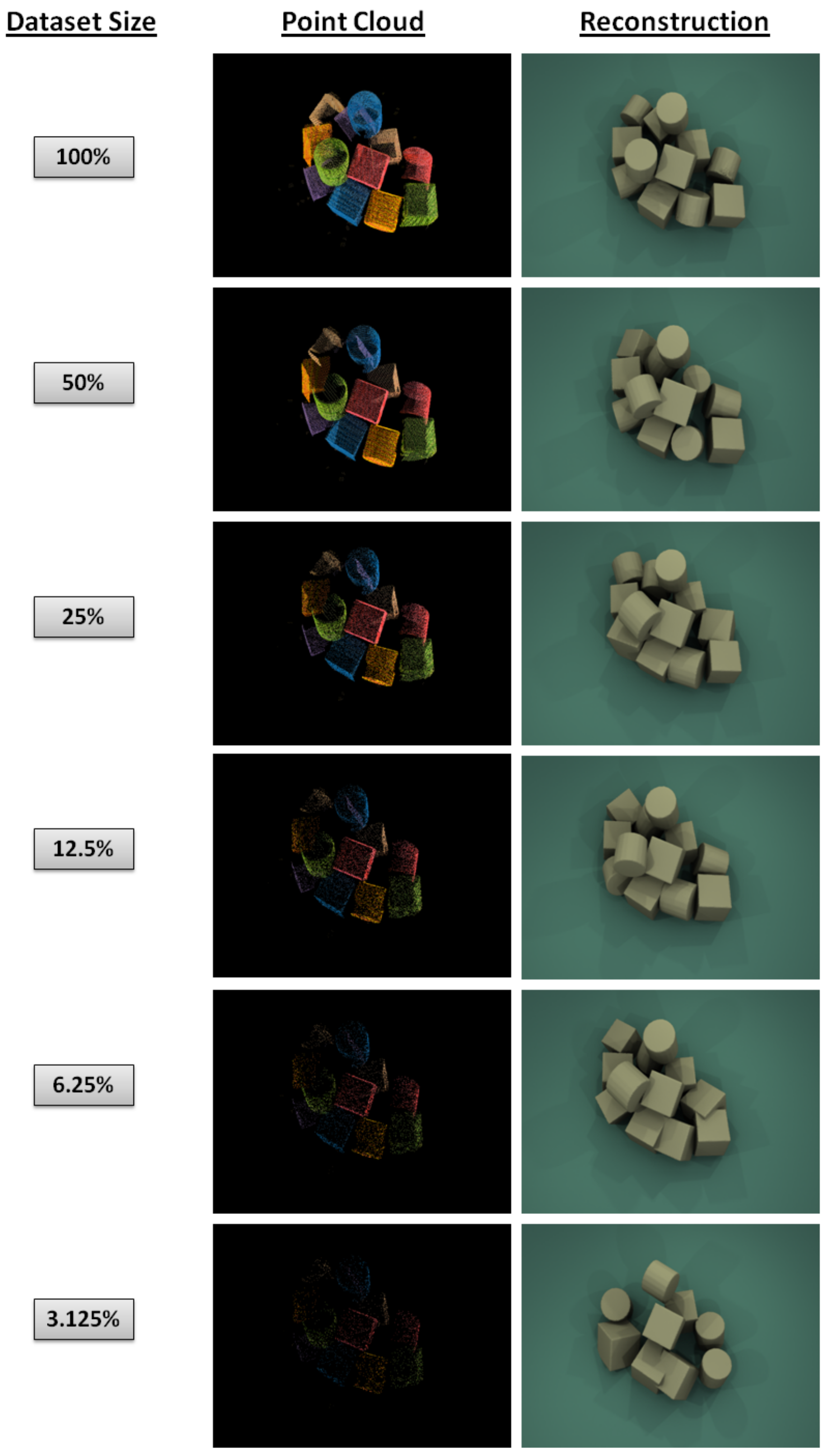


Fig. 36. A series of tests were performed on reduced data to evaluate the robustness of our overall process with respect to dataset resolution.

values). If the matching tolerances are low when matching a root element, it may also cause incorrect iterative fitting of surrounding patches as well. These two behaviors can result in misidentification of objects, fitting of objects in a wrong orientation, and various other errors such as object/object intersections. When reconstructing an individual object we currently do not take into account the location of other already constructed objects, so we are unable to analyze whether an object intersects the boundary of another. At 25%, 12.5%, and 6.25%, similar behaviors occur and the errors are with many of the same objects. At 3.125%, the recognition process begins to break down completely. We are only able to correctly identify and reconstruct three objects, and the overall process cannot match and identify many of other objects within an acceptable tolerance range. Thus, they are never found.

Our approach heavily depends on having adequate samples to correctly identify an object as our method does not address sparse recognition. In addition, clutter does affect the data collection process by occluding objects, and those objects that are less visible will have fewer samples describing them. The more visible objects tend to have many more samples. This is apparent in our reduced datasets as those objects more hidden are the first to encounter difficulty, and the more visible objects are more easily identified. Thus, it is critical that an adequate data collection process is used to fully capture and sample the environment to ensure proper reconstruction.

D. Discussion

The closest methods to our proposed approach are those described by Johnson and Hebert [92], Jenke et al. [15], Schnabel et al. [47], and Gal et al. [45]. The recognition approach described by Johnson and Hebert [92] also uses a local matching procedure for identifying objects in clutter, but they perform a global fitting of the object in space (i.e., using a rigid transformation). Our approach avoids this global fitting process by performing a reconstruction of locally fitted patches into a globally complete object. Their local matching and global fitting process can run into trouble with objects such as those shown in Figure 34, unless a carefully selected sampling size is used to ensure the two similar objects can be adequately distinguished. We are also able to easily handle objects across different scales, an aspect they do not address. Finally, our approach incorporates a weighting scheme to avoid false positive matches and ensures a better starting location for the fitting process.

The patch-graph approach described by Jenke et al. [15] allows a feature-preserving reconstruction of the visible surfaces (i.e., only those represented with point samples) of an object. Their approach is similar to our work in that it uses a graph of locally constructed surface patches as part of the overall reconstruction process. However, their patches are constructed using a locally defined reconstruction and a subsequent subdivision approach (in a feature-aware fashion), and is limited to using only the information provided from the point samples. The authors do not address reconstructing occluded areas as in our work, but simply focus on performing a reconstruction in an efficient and robust manner.

Schnabel et al. [47] provide an approach for filling large occluded surfaces on a single object using simple primitives. Their approach does not require a prior model as input, and they demonstrate that it works well for very regular geometric objects. Through the use of the prior models, our approach allows for a wider range of reconstructions including more irregular and free-form objects.

Our work builds on the augmented reconstruction algorithm presented by Gal et al. [45]. Through our graph-based inference approach, the matching and fitting of prior patches across the model can be performed very efficiently without having to fully search, match, and fit each patch on an individual basis. In addition, the authors demonstrate the filling of missing areas using shape priors, but this was limited to smaller areas near a fitted prior. Through the use of the prior graph, our approach is capable of inferring and reconstructing major portions of the structure of even a highly occluded object. The focus of Gal et al. was on augmenting the reconstruction process for a single object, whereas our work goes beyond this concept to identify and reconstruct objects within a larger scene.

Finally, the proposed matching algorithm described in Section 2 provides one alternative for identification of a potential object within the scene. However, other alternatives also exist. The 4PCS algorithm allows fast and robust pairwise alignment of 3D point sets and is resilient to noise [95]. An extension of this approach could be used for identifying a potential object (using the priors) and establishing an initial alignment. ICP could then be used to refine this alignment, and the remaining recognition and reconstruction stages of our algorithm used to iteratively fit the priors and construct a solid model representation.

E. Limitations and Future Work

This current work has several ways in which it could be improved. One significant issue is that a user must provide as input a database of specific objects to be recognized and reconstructed. This requirement could be eased by allowing the user to define objects with annotated behaviors (e.g., optional parts, repetitive parts, variable articulation, etc.) such as through a shape grammar, thereby allowing a single object to match a larger variety of scene elements. This approach would increase the difficulty and runtime for searching and recognizing objects within the point cloud, as many different configurations must now be robustly matched. However, such an approach would also require fewer models to be provided as input by the user.

We also require a minimal set of point cloud samples to recognize an object. Reconstruction will be difficult if the object is almost completely hidden during the capture process. One possible solution would be to allow a sketch-based interface where the user could illustrate the likely location of these hidden objects, and the automated algorithm could then take this information as prior knowledge and try to fit a corresponding object based on the minimal data.

If noise becomes too excessive, the matching algorithm cannot adequately identify the object and an invalid fitting may take place. We have shown that our approach works fine with the low levels of noise in our real-world datasets, but if noise becomes too extreme, it would create problems. Allowing user intervention and guidance as previously mentioned, would help in these cases also.

Finally, our algorithm does not analyze the physics of a reconstructed environment. As a result, two objects in real-world contact may be reconstructed slightly separated or possibly intersecting. This is due to the variance in the patch fitting process by noise, removal of "incorrect" samples during the iterative object construction process, and other factors. A simple follow-on post-processing that incorporates a simple relaxation process would easily address these issues, resulting in a more accurately constructed environment.

There are several additional ways in which our work might be extended. Much of the approach could be parallelized relatively straightforwardly. Both the recognition and reconstruction phases work around data stored in a regularized grid (i.e., the octree and voxel grid) which could be used to decompose the problem space into smaller parallelized sub-problems. Another interesting direction would be to dynamically extend the rule set based on observed behaviors in the dataset, allowing the reconstruction of objects based on observed examples. As new parts and behaviors are found, the graph could be dynamically updated with data captured from the point cloud. However, in some cases this data may not be as high quality as the prior patches so this must be taken into account during the matching process. Along with the ideas mentioned above (shape grammars, user guidance, incorporating physics), these extensions would allow for handling a wider range of objects allowing this technique to be extended to larger and more complex environments.

CHAPTER IV

INFERENCE-BASED GENERATIVE MODELING

A. Overview

In many situations, a point cloud capture may not be available during the modeling process. In addition, the environment to be constructed may be quite complex and contain a large number of very diverse objects. Figure 1 provides a good real-world example of such an environment. Manually developing an extensive model library can be a very expensive and tedious task. However, reproducing identical objects repeatedly throughout a scene can decrease the underlying realism of the environment, thereby reducing overall user immersion. Thus, there is a cost versus realism trade-off that must be evaluated when modeling such an environment.

The cluttered scene in Figure 1 contains a large number of objects composed of similar features, yet each object consists of different underlying composition. Trying to model all of these objects manually would be a painstakingly complex process. One alternative to alleviate this dilemma is to use more automated methods capable of constructing different variations from rules or templates. Such methods have been used for generating foundational elements such as buildings and cities [49], and vegetation and terrain [54].

The focus of the work we present in this chapter is on constructing variations of individual object models such that they can be easily incorporated into any simulation-based environment. Thus, using a technique such as the generative approach we propose allows a wide range of objects to be developed very quickly with only minimal effort by the user, who simply provides an object template that is used to guide the underlying construction process.

In this chapter, we present a novel generative modeling technique centered on an inference-based construction algorithm for developing diverse models from a set of object templates. Our approach extracts surface patches from a template model, and then fits these patches together in a consistent fashion to fully define the boundary of an object. A parameterization serves as a "road map" for object construction, and patches are incrementally fit around it to define an object. Different behaviors can be dynamically incorporated into the construction process, which allows a wider variety of object configurations to be developed. As a result, this approach is capable of generating a rich collection of different solid model representations. Our work provides three main contributions:

- We provide an efficient algorithm that locally fits patches around a defined parameterization in a globally consistent fashion, and is capable of generating a solid model representation of the object.
- We provide a means in which this process can function in both a semi-automated and a fully automated fashion using a series of techniques for obtaining the underlying parameterization around which the object is constructed.
- We provide several extensions of our basic algorithm that allow for more complex object definitions through the use of articulation, repetition of parts, and interchangeable parts.

B. Algorithm Details

The overall methodology we describe extends our inference-based reconstruction algorithm previously presented. Our process for constructing objects and incorporating them into a virtual environment is broken down into five stages, and is shown in Figure 37.

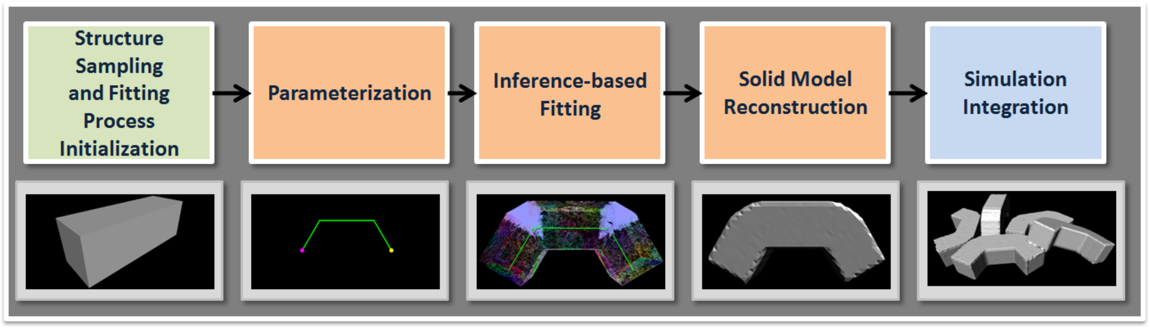

Structure Sampling and Fitting Process Initialization
Parameterization
Inference-based Fitting
Solid Model Reconstruction
Simulation Integration

Fig. 37. Overview of the generative construction process.

The first stage takes as input a set of object templates (i.e., polygonal meshed models), and samples and extracts a set of representative surface patches and a structure description from each. It then generates and initializes the data structures used by the patch fitting process. The second stage obtains a defined parameterization of the object to be constructed using one of several different automated/semi-automated methods. The third stage then takes the sampled patches, the fitting process data structures, and the defined parameterization, and uses an inference-based fitting process to construct an object based on these defining elements. In the fourth stage, the fitted patches are used to reconstruct a solid model representation of the final object. Finally, the last stage integrates the final constructed object into the environment being constructed.

1. Structure Sampling and Annotation

Our approach takes as input one or more annotated polygonal meshed models that represent templates for the objects to be constructed. The first stage in our process takes these input models and samples them, constructing *surface patches* that capture the local surface properties of each object. These patches will serve as the underly-

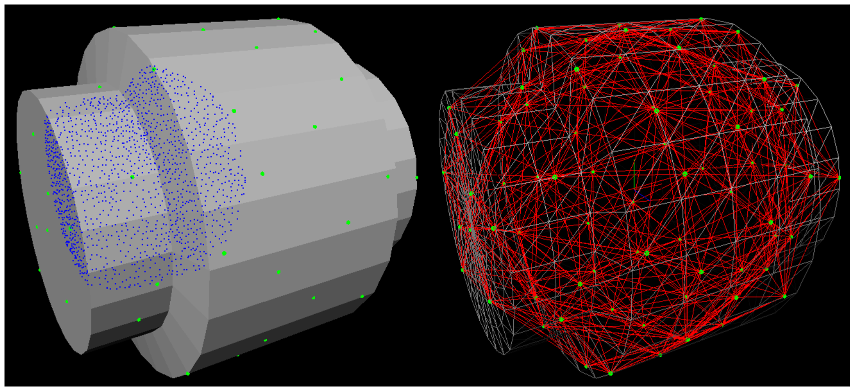


Fig. 38. Example model sampling (green points), patch sub-sampling (blue points), and neighbor graph (nodes in green and edges in red).

ing fitting elements used during the later construction process. A neighbor graph that defines the interconnectivity of these patches, thereby capturing the underlying structure of the object, is then generated. Finally, the data structures used for the fitting process are constructed.

Structure sampling is performed using a similar process as was defined in Chapter III. Each model is first uniformly sampled (described in detail in Appendix A) using a random area weighted scheme, followed by a relaxation procedure that ensures an even distribution of samples across the model's surface [100]. For each sample, the faces and vertices that intersect the volume of a support sphere (i.e., of a fixed user-defined size, centered at that sample, and containing a normal which resides inside a defined support angle around the sample's normal) are captured and stored as the defining elements for the patch. An example sampling of a model is shown in Figure 38 where the samples are shown in green.

Each identified patch is then independently sub-sampled (again using the process as defined by Turk [100]) with the additional constraint that the sub-samples must reside inside the support sphere. Figure 38 shows an example sub-sampling of a patch where the sub-samples are shown in blue. These sub-samples are then normalized (using the process defined by Elad et al. [101]) and stored. These patches provide a characterization of the local behavior of an object's surface.

A neighbor graph is then constructed from the patches. Nodes in this graph correspond to the patch samples, and two nodes are connected with an edge in the graph when their respective support spheres overlap in space. An example of such a graph is shown in Figure 38. Here nodes are represented by the green points and an edge between nodes with a red line. This graph serves as a definition of relationships between different patches, and will be utilized during the later inference-based fitting process to guide the creation of the underlying data structures used during fitting.

Our algorithm makes the assumption that each template model provided is axis-aligned and that its primary features are defined along a particular axis dimension. A one-dimensional *parameterization* is then inherently defined along each axis line. A parameterization is simply defined as the specification of a curve which maps the structure of an object from 0.0 to 1.0 along that particular axis line. This parameterization provides a simple topological skeleton of the object along that axis direction, but does not incorporate branching as with the definition of a medial axis. This approach works well for objects that are very regularly defined along an axis (e.g., a block, pipe, etc.) and may not work well for other objects that are more irregular/organically defined (e.g, a rock, telephone, etc.). This parameterization will be used for determining correct and consistent placement of patches within the mapping process described later in this chapter.

Finally, for each model a user can annotate special regions used during the fitting process. Figure 39 shows an example of an annotated model. The region shown in yellow corresponds to a part that can be repeated iteratively along a parameterization, and those in blue are regions that can be interchanged. Any *annotated sample* that

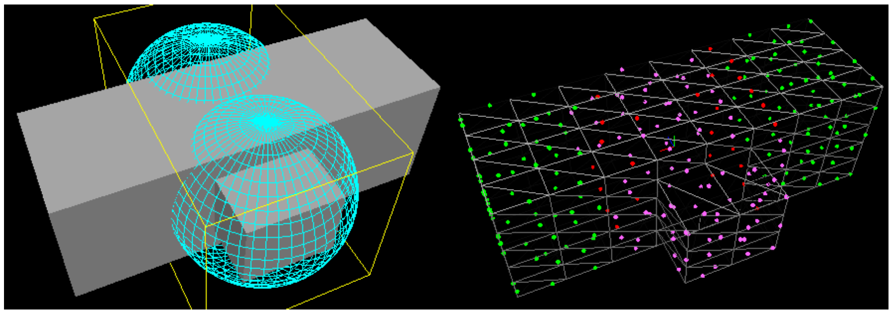


Fig. 39. Example annotated model (left) and corresponding annotated samples (right).

resides inside one of these regions, and any graph edges that cross its boundaries, are identified and marked for later use. In some cases, these regions can also be used for filtering patch sub-samples external to their boundary to provide more accurate constructions. Any sample not residing inside one of these regions is referred to as an *anchor sample*. Anchor samples serve as base elements to begin the construction process with, and the remaining *annotated samples* are then used to dynamically build the object around an arbitrarily defined parameterization. The details behind how these items are used within the construction process will be described in the next two sections.

2. Fitting Process Initialization

The goal of our approach is to define the boundary of a new object by consistently fitting a set of extracted patches from the template model in an alternative configuration. This patch fitting process is driven by our inference-based algorithm which leverages a *colored Petri net* data structure. This process allows patches to be locally fit in a stepwise fashion while ensuring consistency between adjacent items. It also allows the incorporation of different extensions for the generation of a wider variety of different constructed objects. The next step in our process generates and initializes

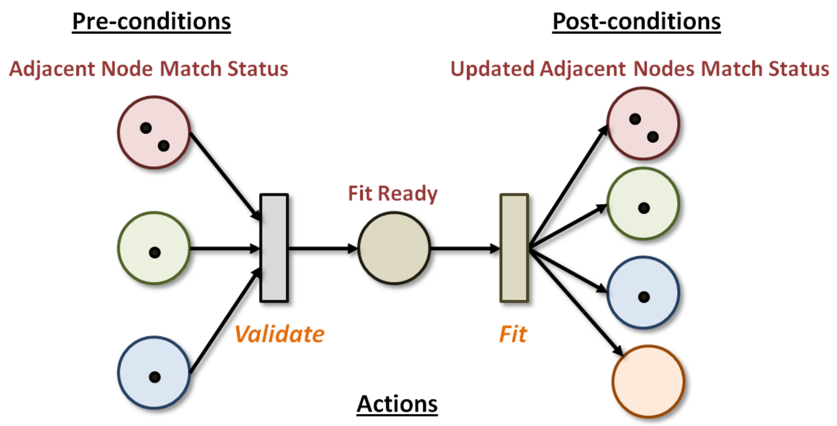

Pre-conditions
Adjacent Node Match Status
Post-conditions
Updated Adjacent Nodes Match Status
Fit Ready
Validate
Fit
Actions

Fig. 40. Colored Petri net structure defining the process for fitting a given node in the neighbor graph. Places are illustrated as circles and transitions as rectangles.

the Petri net data structure used during the fitting process.

Petri nets are powerful data structures because they provide a natural methodology for modeling stepwise processes that include action, choice, iteration, parallelism, synchronization, and dependency [111]. Each of these properties is very applicable to our construction process because our algorithm attempts to logically fit together pieces of an object in a stepwise, yet consistent fashion. A Petri net is a directed graph containing two types of nodes, places and transitions (an example Petri net is shown in Figure 40). A colored Petri net allows the storage of values in the *tokens* that are passed through the network, which can be very beneficial for tracking statuses. Appendix C provides a detailed overview of Petri nets and how they function.

When modeling a system/process with a Petri net, *places* (shown as circles) correspond to the states of the system and *transitions* (shown as rectangles) correspond to actions that the system can perform. In order for the Petri net to define the construction process for a given template model, a set of states and transitions must be defined and joined accordingly to describe the steps necessary for constructing the object from the set of patches.

Thus, we begin by taking the previously described samples, patches, and neighbor graph developed during structure sampling, and use these items to construct a Petri net structure. The objective of this structure is to help define, guide, and track the distributed fitting process, and ensure that it is done in a correct fashion during the later fitting process. For each node in the neighbor graph a *matching status place*, a *fit ready place*, a *validation transition*, and a *fitting transition* are constructed. A structure similar to that shown in Figure 40 is then built for each node in the neighbor graph using these elements. This structure defines the steps necessary for fitting a patch. These locally developed structures, when combined, form a global network that fully defines the process for constructing the template object.

For each node the *pre-conditions* necessary for, and the *post-conditions* that result from fitting a patch are modeled as places (i.e., using the matching status places). In addition, two transitions are added. The first, the validation transition, determines when a patch is capable of being fit (i.e., if all of the adjacent nodes have been marked as possible fits) and after firing, moves the item into a fitting state. The second, the fitting transition, prompts the fitting process to start, and updates all adjacent nodes after the actual fitting has been performed. As a node is fit, its adjacent nodes are alerted to this fact. As enough information about the neighbors of an adjacent node becomes available, it is in turn fit.

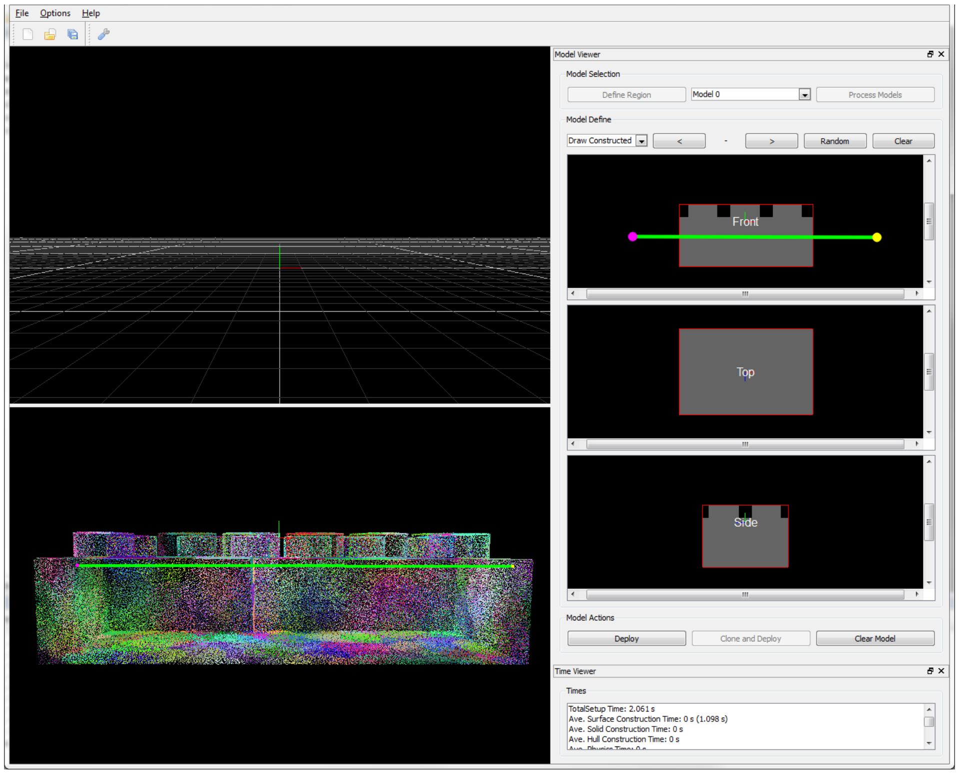
3. Parameterization

The construction process begins by obtaining a parameterization that defines the basic configuration of the object to be constructed. This parameterization is defined with a piecewise smooth curve, and serves as a “road map” for the patch fitting process. It is somewhat similar to an object’s skeleton, but it is a piecewise linear curve only and based on its definition, different behaviors extracted from the template model can be dynamically incorporated into the constructed object. Patches are fit around this curve, but fit relative to each other in a consistent fashion based on their original definition in the template model. This allows a reconfigurable definition of an object that still fully defines a complete boundary. We have experimented with four different methods for obtaining this parameterization, including methods that function in both fully automated and semi-automated fashions.

The first approach, a sketch-based interface, allows the user to define a configuration using a simple 2D drawing interface. The user simply sketches a curve along one of the three fixed axis-aligned dimensions. The resulting two-dimensional curve can then be easily transformed into the third dimension to guide the construction process. Figure 41 shows an example of our user interface, and an example of a sketch-based curve is shown in the top left of Figure 42. This user-defined approach allows the generation of a wide variety of customized curves.

The second approach, a random walk, is fully automated and generates a completely random curve. In this approach minimum and maximum numbers of vertices are defined, and a random size within this range selected. Vertices are then randomly placed in space, and the respective edges are defined based on the order of vertex creation. This approach is fast and can be used to generate an unbounded number of curves. However, it lacks intelligent placement of vertices and often results in overlapping line segments in the final curve which may be bad for object construction. An example of a random walk curve is shown in the top right of Figure 42.

The third approach, a turtle graphic [112], uses a slightly more sophisticated method. This approach begins by finding a uniform sampling of the defined space for the object, and then picks a random starting point within this sampling. The pen (i.e., the turtle) has a position and orientation, and for each step it automatically



```
TotalSetup Time: 2.061 s
Ave. Surface Construction Time: 0 s (1.098 s)
Ave. Solid Construction Time: 0 s
Ave. Hull Construction Time: 0 s
```

Fig. 41. Our sketch-based interface for defining a parameterization.

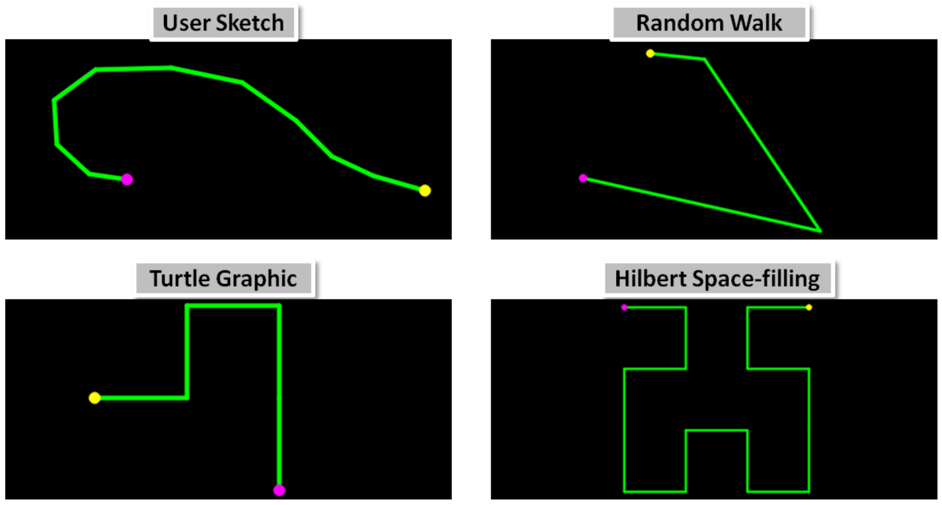


Fig. 42. Example parameterizations we have experimented with.

picks a random adjacent sample to move to. It is careful to not use a sample more than once, it moves only a predefined number of steps, and for each movement a line is drawn between the samples. This approach can generate nice regular curves that are nonintersecting, but the method is limited in its creativity because for each step the pen can only move to an adjacent point. An example of a generated curve using this method is shown in the bottom left of Figure 42.

The fourth approach, the Hilbert space-filling curve [113], uses a more sophisticated method and generates fractal-based curves that fill the defined space for the object. This approach subdivides the space into a set of cells, and recursively constructs the curve using a defined mathematical function and an extension of the previously described turtle graphic approach. Defining higher order curves can produce larger and more detailed parameterizations. This approach can produce some very interesting results, but there are many other space filling curves [113] that could work equally well. An example of a generated curve using this approach is shown in the bottom right of Figure 42.

All four methods provide an easy means for quickly defining the general configuration of the object to be constructed, and this set of methods can be used to generate a wide collection of very different results. The defined parameterization obtained from this step is passed to the next stage to guide the construction process.

4. Inference-Based Fitting

The inference-based construction process incrementally fits surface patches extracted from the user provided template, around the defined parameterization. This fitting must be done in a consistent fashion to ensure that an accurate and complete object definition is generated (i.e., adjacent patches must have consistent overlapping areas to define a smooth and complete surface when joined; dynamic behaviors such as repetition and interchanging of parts makes this fitting process more complex).

Once the Petri net structure has been generated and a parameterization defined, then the iterative fitting process can begin. The fitting process then iteratively propagates over the surface of the object, fitting patches along the way. The general flow of this process is described in Algorithm 5, and the details behind each step of the process will be explained in the remainder of this section.

Algorithm 5

```
1.  $node = FindStartingNode();$
2.  $AddStartingTokenToNode(node);$
3.
4.  repeat
5.      $FirePetriNet();$
6.      $PerformPostFireCleanup();$
7.      $anyFittingsReady = CheckPetriForFittings();$
8.
9.      if $anyFittingsReady$
10.         then $HandlePetriFittings();$
11.         else $SearchNewPetriFitting();$
12. until $CheckIfFittingProcessFinished() ==$ true;
```

a. Initialization and Stepwise Fitting

The first step in this process must determine the starting sample/patch in which to begin the fitting. This item is found by analyzing the samples with respect to the parameterizations defined in the original model space (i.e., referred to as *template space*), and identifying the sample with a minimum value along the parameterization of the longest primary axis (i.e., if it is defined in the XY plane, and has a general horizontal orientation, then the X range is evaluated). The selected sample must also be an anchor sample to ensure a correct start to the overall fitting process.

Once the starting element has been identified, an initial matching token is placed into the fit ready place for that respective item. This will trigger the fitting process to begin on the next firing of the Petri net. Since we use a colored Petri net, the added token is capable of storing values. For each token, the current state (i.e., match, trial, or invalid), the location of the node the token is set for, the current repetition iteration and/or interchangeable part index, and the current offset (i.e., as a result of repetition of regions in the template) are stored. Each of these values will be used during later steps of the fitting procedure.

The iterative process begins by first firing the Petri net. For a validation or fitting transition to fire, all of the incoming places must contain a match or trial token that matches the respective inputs for the current item. In addition, a check is performed to ensure that the patch has not already been fit to that particular location in construction space. If these conditions hold and a validation transition fires, then the matching tokens are removed and a token is placed into the item's fitting place. This triggers the start of the fitting process for the patch. Upon the next fire of the network, the patch will be fit and the post-conditions are handled. As a result, all adjacent elements (including the item being fit itself) are given a new matching token. These new tokens imply that the adjacent patches may be ready for fitting based on the information obtained from previously handled items.

For each iteration in the fitting process, the network is fired, some post-firing cleanup is performed (this will be explained later), and a check is performed to see if any patches are ready to be fit (i.e., by checking if any of the fitting transitions fired). If it is determined that there is not adequate information to fit a patch, a new item must be selected. This selection process is performed by iterating over all of the validation transitions, and for each item, analyzing the tokens that exist within the item's incoming places. The transition that contains the highest number of incoming places (i.e., has the most supporting information) is selected as the next "best" node to fit. In this case, a trial token is added to each element's matching status place that does not currently have a valid matching token. This allows a test fitting to be performed upon the next fire of the Petri net.

The overall fitting process is performed incrementally, fitting patches across the surface of the object based on the progressive fitting of neighboring patches. This process is performed until all items in the original template object have been used, and there are no longer any items left to be handled. The propagation across the parameterization, fitting patches along the way, ensures that all elements of the constructed object are handled and a complete object defined.

b. Patch Mapping

Once a patch is selected for fitting, it must be mapped from the original template space to the space in which object is being constructed (i.e., referred to as *construction space*). The fitting process begins with the selected starting patch that resides along the minimum of the determined primary axis, and this patch is fit to the starting point of the constructed model parameterization (i.e., at $t = 0$ of the parameterization). To fit the patch in construction space, a transformation must be applied to all of its elements (i.e., its sample, sub-samples, etc.). Algorithm 6 provides an overview of this process.

Algorithm 6

```
1.  $param = GetParameterization();$
2.  for $x \in samples \bigcup subsamples$
3.          $t = FindRelativeTValue(x);$
4.          $TransformPositionAlongParameterization(x, t);$
```

For each element in the patch, a respective t value is found within model space and used to determine the correct position along the parameterization in construction space. Each element is then translated to the position such that it maintains the features of the item in template space, but is fit relative to the defined curve in construction space. Finally, the element is rotated such that its three principal axes

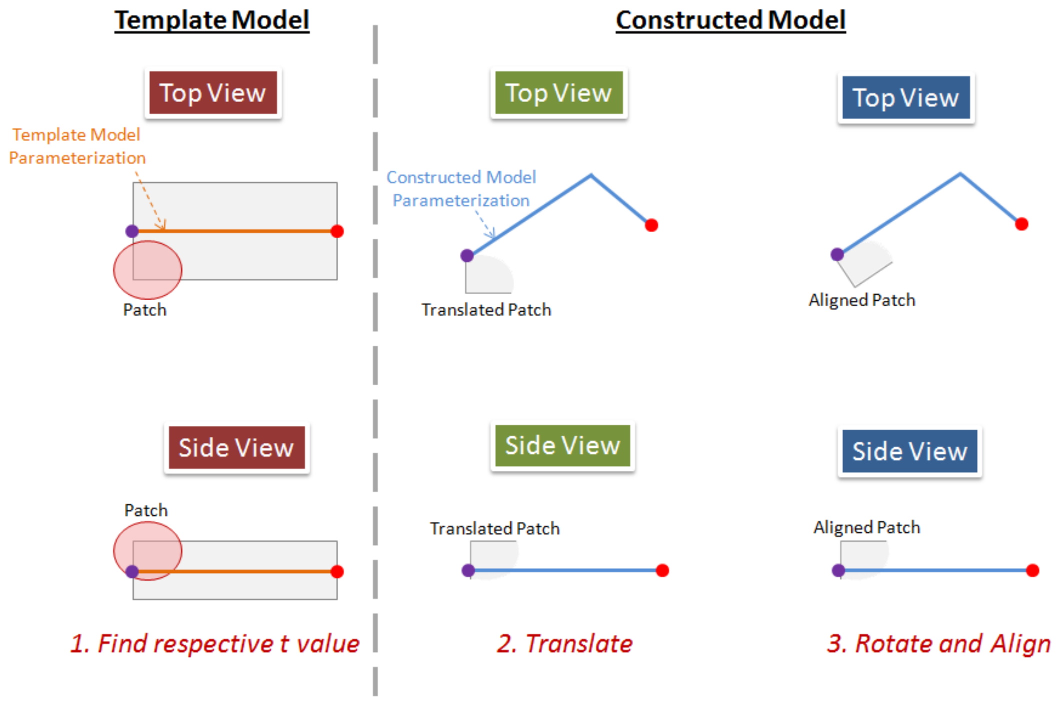


Fig. 43. An illustration of the patch mapping process used to fit a patch from the template model to its correct position and orientation along the defined parameterization. The extracted patch is translated along the parameterization to the determined t value, and then rotated such that its principal axes are aligned with the sketched curve.

are aligned with that of the sketched curve. The result of this fitting process is a sample, and set of sub-samples, that encompass the features of the template object but are fit to the parameterization. Figure 43 provides a two-dimensional illustration of this mapping process.

After the initial patch is fit, and as new adjacent patches are selected, they must be fit relative to the initial item to ensure a proper surface definition. This operation is performed by again finding a t value for each item taken from template space, but this value must be relative to the initial fitted patch in construction space.

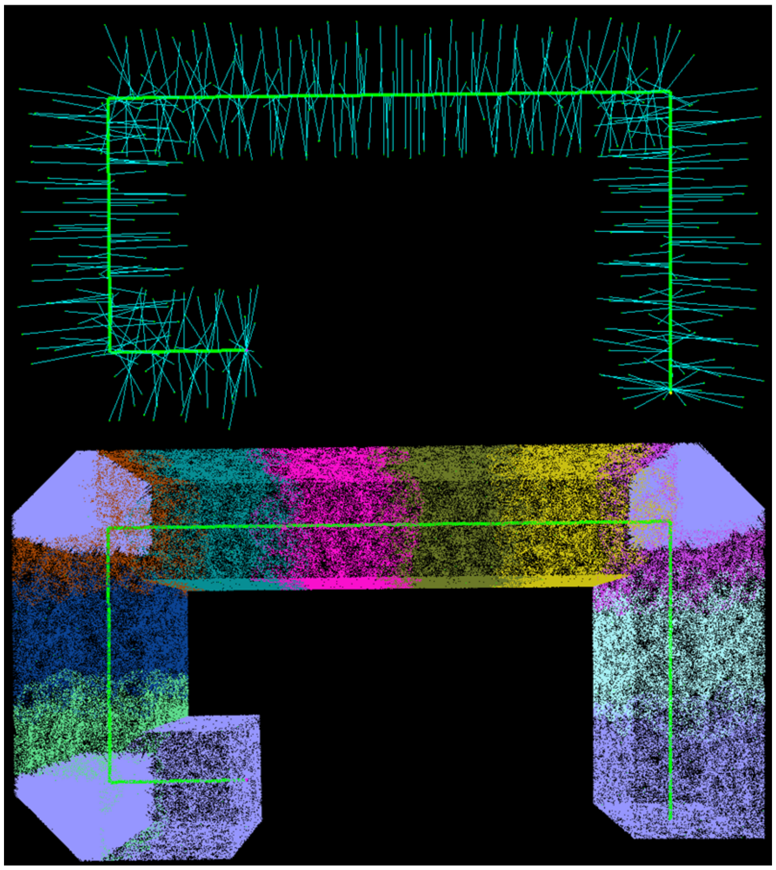


Fig. 44. Mapping of samples and sub-samples along a defined parameterization.

Thus, this process fits the patches exactly as they exist in template space, but simply transformed around the new parameterization. Figure 44 shows an example of the results from this mapping process. The blue lines illustrate the t mapping for each sample to the underlying parameterization.

c. Handling Articulation

The parameterized curves allow the definition of articulated joints. As a result, the fitting patches must be properly mapped around these regions. For patches that are split across a boundary defined by a joint, gaps are created in the samples above

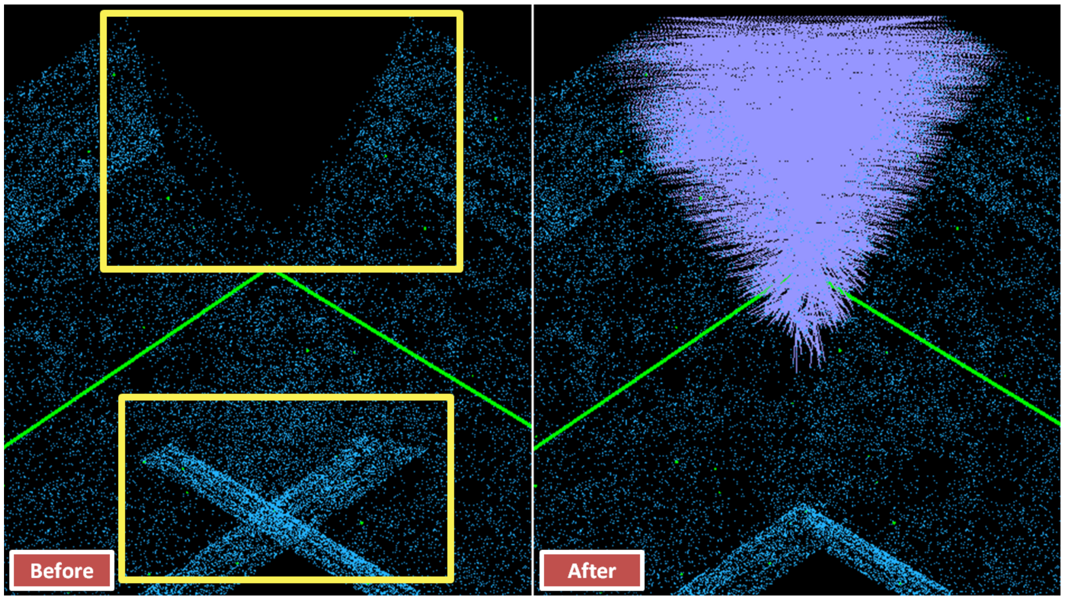


Fig. 45. The before/after states for cleanly mapping patches across articulated joints.

the curve and excess points will appear below the curve (i.e., from an inter-surface intersection) due to the discontinuity in the mapping process. The left side of Figure 45 illustrates an example of such a case. In order to ensure a solid model is constructed during the later stages, these gaps must be adequately filled with samples and the excess points inside the surface removed. Algorithm 7 provides an overview of this process.

To fill a gap caused by a break in a patch, we begin with the original patch as defined in template space, and identify the crease in which the patch is being broken (i.e., using the chosen sketch plane and the parameterization). A region of a fixed width on either side, and along the full length of this crease is found (e.g., using a fixed percentage of the defined support distance), and the samples residing within each region on either side are identified. The nearest neighbor for each sample residing in the opposite region is then found, and a line between the two samples is created and sub-sampled, and transformed into construction space. The collection of these sub-sampled lines "stretches" the patch across the gap and provides the necessary filler to ensure proper reconstruction. Figure 46 illustrates this process and Figure 45 shows an example of its results.

Algorithm 7

```
1.  for $x \in patchesEffectedByArticulation$
2.      $crease = FindCreaseDetails(x)$;
3.      $FindRegionsOnBothSidesOfCrease(x, crease, leftRegion, rightRegion)$;
4.      $leftSamples = FindSamplesInsideRegion(leftRegion)$;
5.      $rightSamples = FindSamplesInsideRegion(rightRegion)$;
6.      for $l \in leftSamples$
7.          $adjNeighbors = FindAdjacentNeighbor(l, rightSamples)$;
8.          $lineSamples = ConnectWithLineAndSubsample(l, adjNeighbors)$;
9.          $TransformAndStoreSubsamples(lineSamples)$;
10.
11.     for $r \in rightSamples$
12.         $adjNeighbors = FindAdjacentNeighbor(r, leftSamples)$;
13.         $lineSamples = ConnectWithLineAndSubsample(r, adjNeighbors)$;
14.         $TransformAndStoreSubsamples(lineSamples)$;
15.
16.     $plane = FindBisectionPlane(crease)$;
17.     for $s \in samples \bigcup subsamples$
18.         if $ResidesOnOppositeSideOfBisectionPlane(s, plane)$
19.           then $RemovePoint(s)$;
20.           else  $KeepPoint(s)$;
```

To remove the excess points, the bisection plane for the joint is found and a simple test performed. All points that are determined to belong to one side of the split in template space, yet reside on the opposite side of the bisection plane in construction space are removed. Figure 47 illustrates this process and Figure 45 shows an example of its results.

d. Handling Repetition

Another aspect of our approach is the ability to fit an object to a parameterization that may be much longer than the originally defined template object. The example

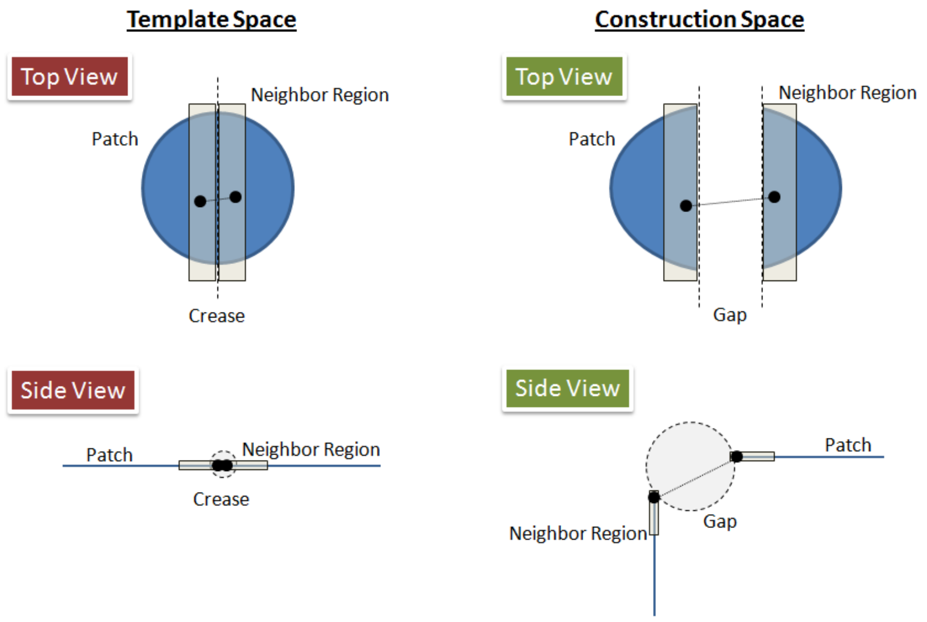


Fig. 46. Gaps in the surface definition caused by articulation can be filled by "stretching" a patch across the void. This is performed by identifying a fixed width region on either side of the crease in template space, locating the samples in each region and their corresponding nearest neighbor in the opposite region, and then sub-sampling the line connecting each pair of samples.

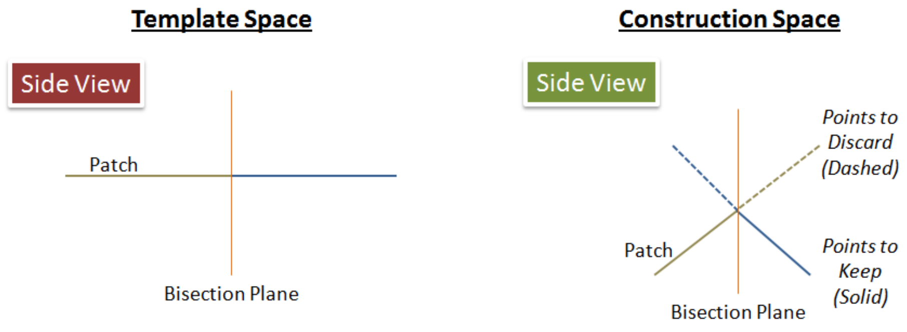


Fig. 47. Points from an inter-surface intersection caused by articulation can be removed by finding the bisection plane at the joint, and performing a simple spatial test to identify the points that lie on one side of this plane in template space and the opposite side in construction space.

shown in Figure 48 illustrates such behavior. In this example, a simple rectangular block was used to construct a much larger and more complex object. In order to adequately handle these cases, repetition of the previously described user annotated regions is used. Through storing additional values within the tokens in the Petri net, and using post-firing manipulation of these tokens, this behavior can be easily handled. Algorithm 8 provides an overview of this process.

Algorithm 8

```
1.  for $tok \in tokensCrossingRepetitionBoundary$
2.          $selReg = GetRegionCrossed(tok);$
3.          if $DidTokenEnterRegion(tok, selReg)$
4.            then
5.                $enterStatus = IsFirstTimeRepRegionEntered(tok, selReg);$
6.                if $enterStatus == true$
7.                  then
8.                      $IncreaseRepetitionCounter(tok.repCounter);$
9.                      if $enterStatus == true$
10.                       then $markedFirstNode = GetNodeFromToken(tok);$

12.         if $DidTokenExitRegion(tok, region)$
13.           then
14.               if $IsAnotherRepetitionNeeded()$
15.                 then
16.                     // update/move token to the first marked node of repetition
17.                     $RemoveTokenFromNode(tok);$
18.                     $IncreaseRepetitionCounter(tok.repCounter);$
19.                     $UpdateOffset(tok.offset);$
20.                     $AddTokenToNode(tok, markedFirstNode);$
21.                 else
22.                     // leave the token where it is, but invalidate it
23.                     $InvalidateRepetitionCounter(tok.repCounter);$
24.                     $UpdateOffset(tok.offset);$
```

As tokens are moved through the network from the places corresponding to the anchor samples, across the edges intersecting the boundary of a repetition region and into those places corresponding to repetition samples, the values of the token must be adjusted to represent the current iteration of repetition. If a token is entering into a region for the first time, then it must be marked with a valid repetition iteration

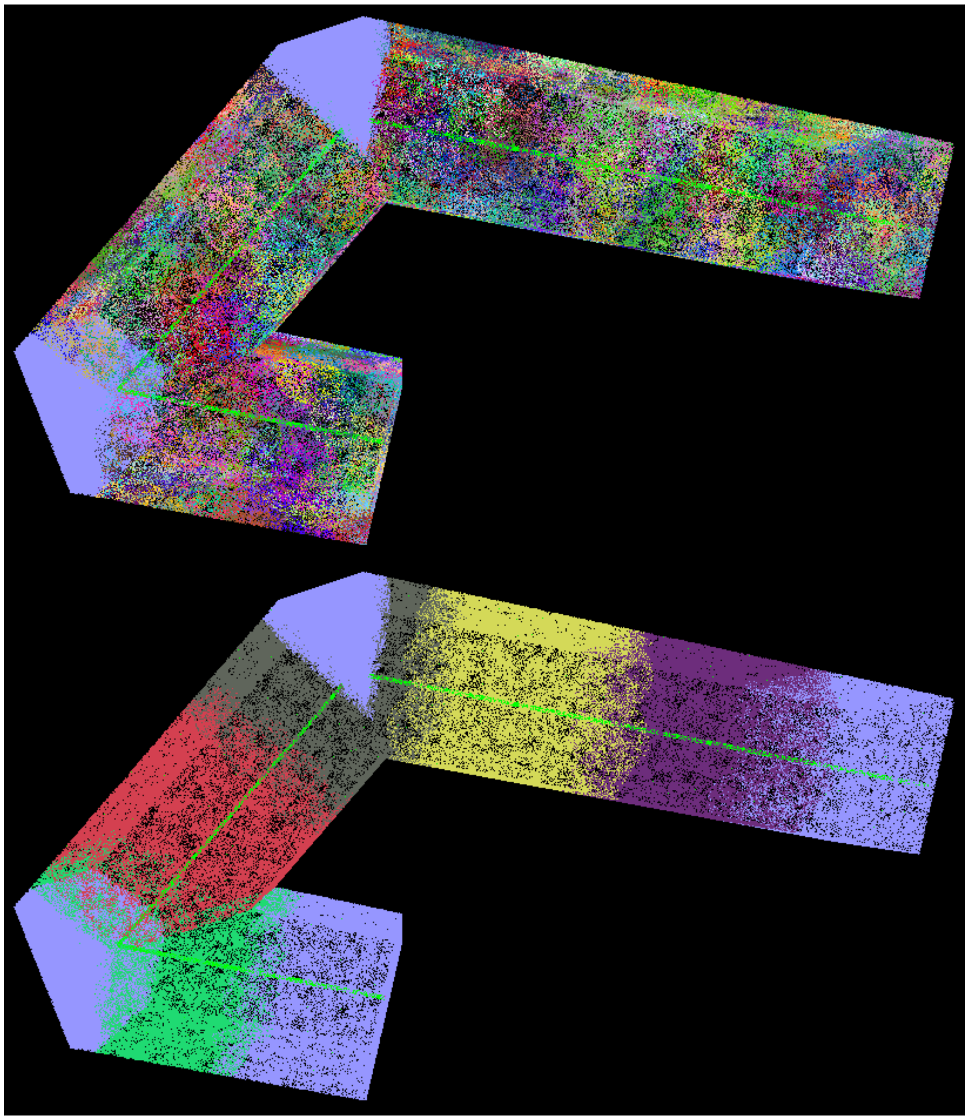


Fig. 48. Results of the patch fitting process using articulation and repetition of parts. The top image shows the fitted patches and the bottom shows the repeated parts.

(i.e., the repetition counter is set to zero). This value is a simple counter that tracks which chosen repetition the token belongs to, and helps determine the necessary offset transformation for correct fitting. Note that a place may contain tokens from several different iterations. Thus, this value must be checked during transition firing, and for a place to fire, all tokens must be of the same iteration. Finally, if this is the first repetition place to receive a token, it is recorded as the initial starting place for the repetition region.

If a token is leaving one of these repetition regions and going back into an anchor region, then the token must be captured and an analysis performed to see if another repetition is needed to fully define an object for the parameterized curve. If it is determined that another iteration needs to be started, then the token is removed from the outgoing place it currently resides in, and it is reinserted at the previously captured starting place for the repetition with an incremented iteration value. If it is determined that another repetition is not needed, then the token is passed on to the adjacent node but only after marking its iteration value as invalid.

Finally, as tokens move out of a repetition region and into an anchor region, the offset value stored in the token must be updated. For anchor regions that appear after a repetition region with respect to the defined parameterization, the offset value stored in the token is kept. For regions that appear before the repetition region, the value is set back to zero. This is an important step as this offset is used to determine the correct final fitting location of the patches in construction space (i.e., by finding an offset t value based on the repetition part size to correctly fit a patch).

Figure 48 shows the result of the patch fitting process when using repetition of parts. In this figure, the top image shows the individual fitted patches and the bottom image shows the actual repeated parts (i.e. each repetition is assigned a unique color, and repetition is performed until the parameterization has been satisfied). The areas in purple along each joint in the parameterization are the samples added to fill the surface disconnects due to articulation. These samples cap the holes created when splitting the patches. The full set of these samples defines the boundary of the object, and can be passed on for integration into a solid model representation.

e. Handling Interchanging

The final aspect of our approach is the fitting of interchangeable parts obtained from the input template. Figure 49 shows an example of this behavior using the annotations from Figure 39. In this example, the user marked two different parts as being interchangeable (i.e., the two blue circles). One part corresponds to a peg, and the other to a flat region. Interchanging these two parts randomly, along repeated regions, allows the creation of many different configurations of the object (one such example is shown in Figure 49). Similar to the repetition of parts, through analyzing and manipulating the tokens moving through the Petri net, this behavior can be easily handled. Algorithm 9 provides an overview of this process.

Algorithm 9

```
1.  for $tok \in tokensCrossingInterchangeBoundary$
2.        $selReg = GetRegionCrossed(tok)$;
3.        if $DidTokenEnterRegion(tok, selReg)$
4.          then
5.               if $IsFirstTimeInterchangeRegionEntered(tok, selReg)$
6.                 then
7.                      $annlist = GetListOfInterchangeAnnotations(selReg)$;
8.                      $randReg = ChooseRandomRegion(annlist)$;
9.                      $fitParams = GetTransformParams(selReg, randReg)$;
10.                     $offset = GetRelativeOffset(tok)$;
11.                     $node = FindClosestNodeInRegion(randReg, offset)$;
12.                     $tokenCreated = InstantiateNodeForPartFitting(node)$;
13.                     $SetupRepetitionDetails(tokenCreated)$;
14.                else
15.                     $DiscardToken(tok)$; // part already started
16.       if $DidTokenExitRegion(tok, region)$
17.         then $DiscardToken(tok)$; // do nothing on an exit but discard token
```

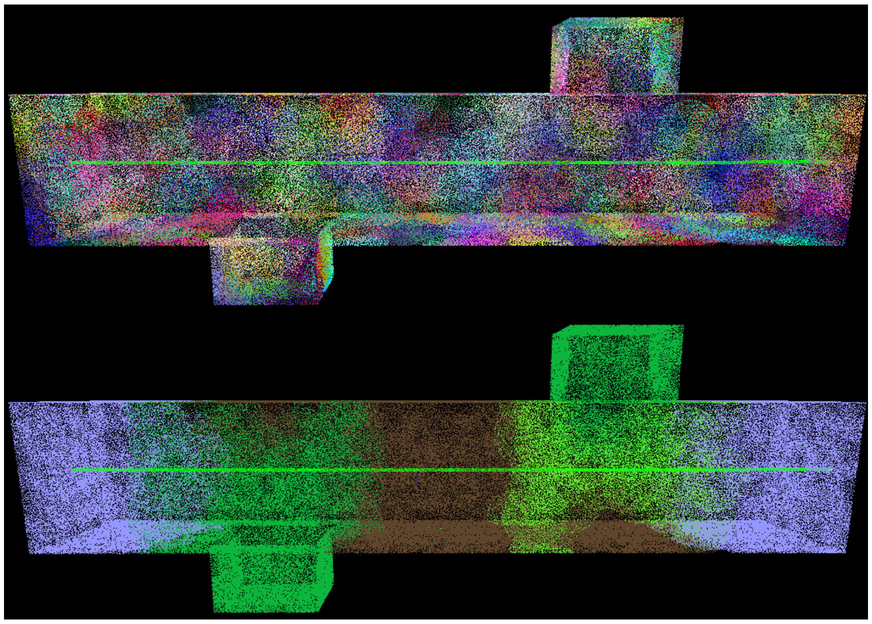


Fig. 49. Results of the patch fitting process using interchangeable parts. The top image shows the fitted patches and the bottom shows the interchanged parts.

As a token moves through the network from a non-interchangeable region, into a region defining an interchangeable part, a check must be performed to see if this is the first time the region and repetition has been entered. If it is the first time, then a new part must be selected for fitting. If it is not the first time, then the token is discarded as the region has already been handled and a fitting is underway.

The process for selecting a new part involves several steps. First, a random selection is made from the list of available parts as defined by the user annotations. During the annotation process, the user identifies which elements are swappable. This step simply chooses a random item from this corresponding list. Once the selection has been made, then information to support the fitting process for the selected part must be obtained.

In order to fit a randomly selected part to the correct location along a parameterization, several steps must be performed. This process begins by finding the parameters needed to transform the randomly selected part to the selected part being fit to the parameterization (i.e., a transformation between each of these items as they are defined in template space). Next, the relative offset of the selected interchangeable sample to the center of the interchangeable region is found. Using the transformation, this offset is then used to find the closest corresponding sample within the randomly selected part. This sample will serve as the starting point for fitting the remainder of this part.

Now that the starting element has been found, a token can be instantiated and initialized with a starting set of values (e.g., current interchange index and repetition, randomly selected interchange index, and location). Finally, the token is added to the corresponding place for this starting element to begin the fitting process for the part. On subsequent iterations of fitting, the patches making up the selected region will be fit until all have been handled and the part is fully defined.

Note that the interchange case must be handled in conjunction with the repetition case, as repeated regions can encompass different interchangeable parts. The image in Figure 49 shows an example of this behavior. With each repetition, a different set of randomly selected parts is chosen. Using a combination of these two behaviors allows easy generation of a wide variety of different model configurations.

5. Solid Model Reconstruction and Integration

Once the inference-based patch fitting process has completed, the next stage in the construction process is to integrate the patches and their corresponding samples into a solid model. We use a process similar to that defined in Chapter III for this overall operation.

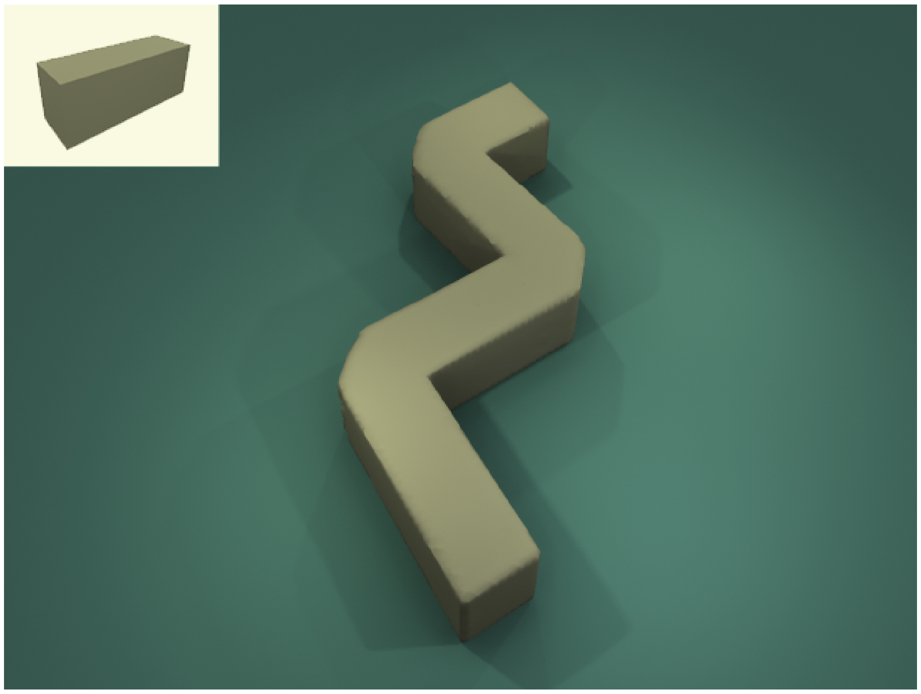


Fig. 50. An example constructed object (center) from a block template (top left).

The reconstruction process begins by taking the point samples from all of the fitted patches, and those that were added to fill the gaps, and combines them to form a single point cloud representation of the object (i.e., a set of samples that fully defines the boundary of the object). A projection-based Moving Least Squares (MLS) approach [4] is then used to obtain a smooth and continuous surface definition. A detailed description of the MLS process we use is provided in Appendix B. A Marching Cubes contouring algorithm [5] is then used to generate a polygonal representation from the implicit surface. As discussed in Chapter III, Marching Cubes is a simple and efficient technique, but many other alternative contouring algorithms could also be used. An example constructed object is shown in Figure 50, and the underlying template object used is shown in the top left corner.

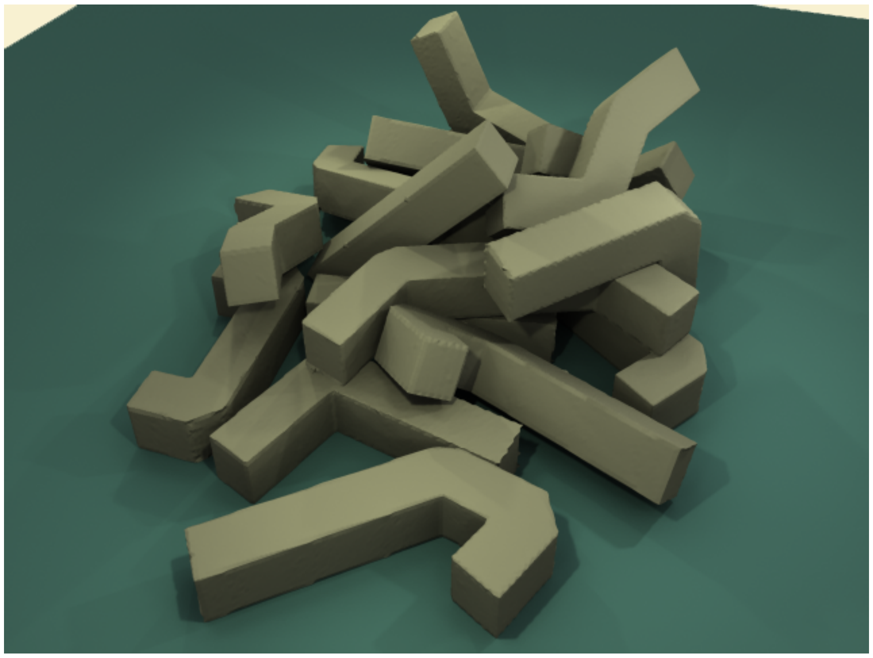


Fig. 51. Example environment composed of several constructed objects.

The final stage of our process takes the constructed object and integrates it into an overall environment. Integration is an important step, as the object must be inserted into the environment in a believable position and pose such that it looks natural to the user. Since our work is focused on creating very cluttered environments such as the real-world example shown in Figure 1, we use a simple method for insertion. Each object starts at a predefined height above the pile and is simply dropped into place. A physically-based simulation then moves the object and determines its final resting place. The result provides an adequate re-creation of such environments. Figure 51 provides an example of a generated environment using this approach.

Table 3. Details of the Generative Modeling Examples

Dataset	**Num. Objects**	**Total Sampling**	**Ave. Fitting**	**Ave. Solid**	**Total Runtime**
Example 1	10	2.2s	0.9s	12.7s	218.3s
Example 2	8	2.0s	1.1s	9.0s	189.5s
Example 3	6	5.8s	2.7s	23.7s	237.4s
Example 4	8	3.6s	0.9s	15.9s	217.9s
Example 5	30	25.2s	6.6s	21.3s	1,470.1s

C. Results

In order to demonstrate the application of our approach, we provide several results along with the details of their execution. All results were captured using an Intel Xeon 2.67 GHz CPU with 4GB of memory. Currently, our algorithm implementation performs all computations on the CPU and uses the GPU only for rendering. The dataset details and respective execution times for each stage (i.e., structure sampling, patch fitting, and solid model construction) of the provided examples are shown in Table 3. The total runtime summarizes the complete construction time and the amount of time for integrating the objects into a simulation-based environment. This integration process will be described in more detail later in Chapter V.

Figures 52, 53, 54, 55, and 56 show some results generated by our approach. In these examples, we show how different types of objects can be easily constructed. Through the use of articulation, repetition, and interchanging of parts, many different object configurations can be quickly developed. The template object used for each

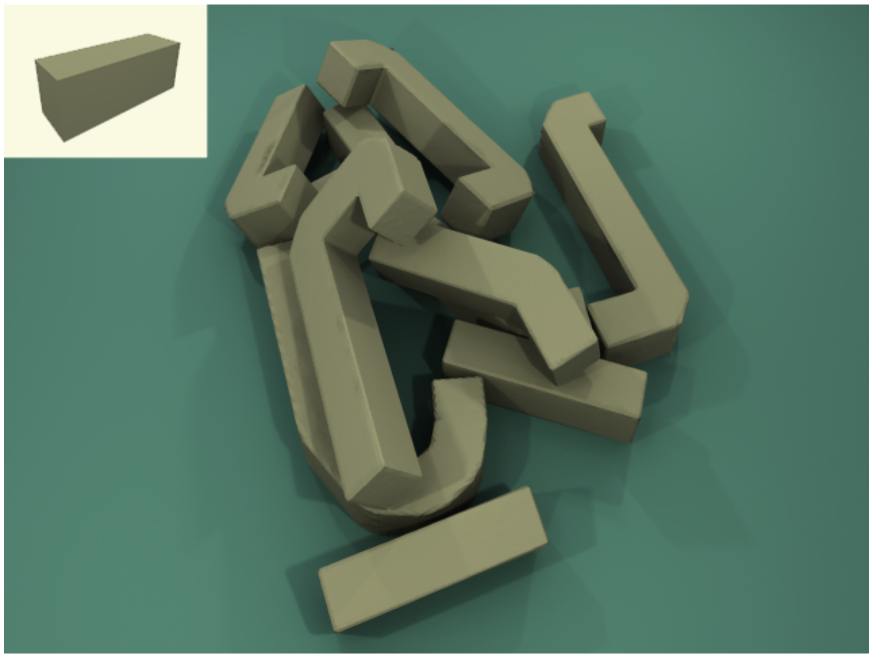


Fig. 52. Constructed blocks using articulation and part repetition (Example 1).

example is shown in the top left corner of each figure.

D. Discussion

The closest methods to our proposed approach are the model synthesis technique described by Merrell et al. [72][73][74], shape grammars (e.g., [63]), and parts-based and example-based modeling methods (e.g., [78][79][45][82]).

The work by Merrell et al. [72][73][74] has a common theme to ours, but takes a different underlying approach. Similar to our method, they allow arbitrary generation of random models by extracting regions from a provided template. However, their approach centers around the satisfaction of different types of constraints (e.g., adjacency, algebraic, incidence, connectivity, and large-scale). These constraints govern the dynamic growing/creation of a model using model synthesis. Our approach

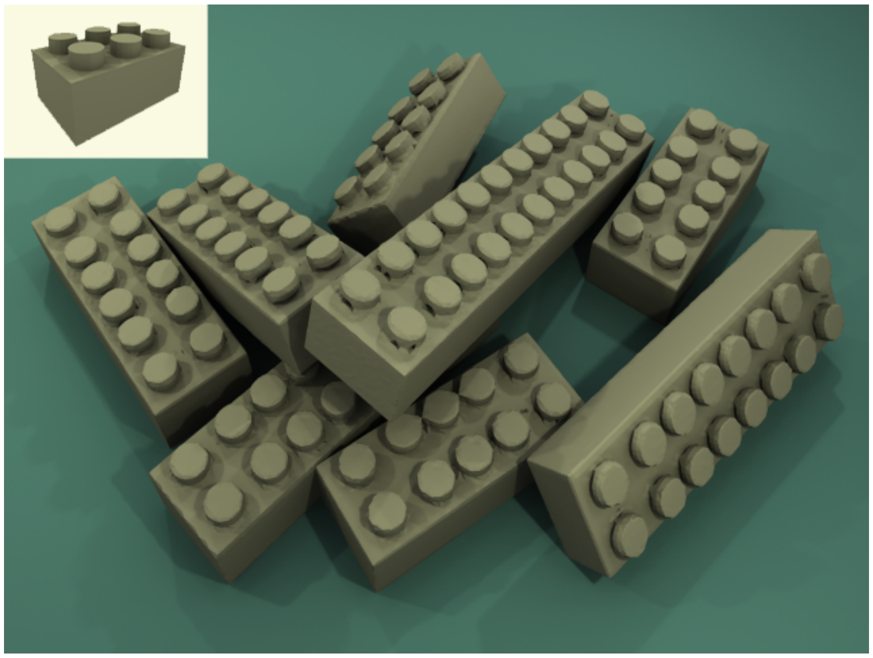


Fig. 53. Constructed bricks using only part repetition (Example 2).

is different and is focused on fitting the extracted patches around an already defined parameterization. While their work allows the construction of arbitrary models, their models are generally regularly defined and self similar. Our approach allows very different models to be constructed from the template through the use of arbitrary articulation, as well as random interchanging of parts. Finally, the authors state that the time and memory requirements involved in their construction heavily depends on the number of vertices generated during the synthesis process. As a result, their approach does not work well with curved or highly tessellated models. With the exception of the grid size used during contouring, our approach does not have these same challenges.

The general approach of shape grammars has some similarity to our work as well. These methods center around taking one or more primitive shapes, along with

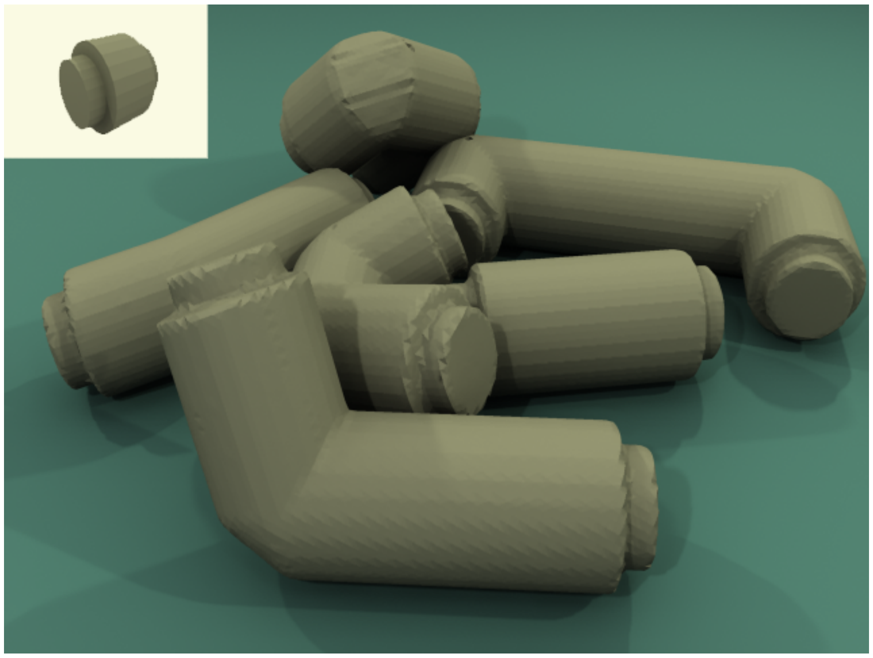


Fig. 54. Constructed pipes using articulation and part repetition (Example 3).

a set of defined rules for transforming these parts, to develop arbitrary geometric models. These approaches are commonly used for defining architecture (e.g., [63]). The biggest difference between our work and many of these approaches is that we use the template model as a means of automatically developing the rules for how to properly fit patches together in a consistent fashion. The shape grammar approaches typically rely on user-defined rule sets, along with primitive shapes for construction. Our approach avoids this process of manually defining rules typically required of these approaches (e.g., as in the case of constructing buildings [63]).

The parts-based and example-based modeling approaches also have some similarities to our work. There are several different variations of these techniques. Some methods extract patches and parts from a model provided for modeling by example [78][79] or improved reconstruction [45]. Other methods identify structural regu-

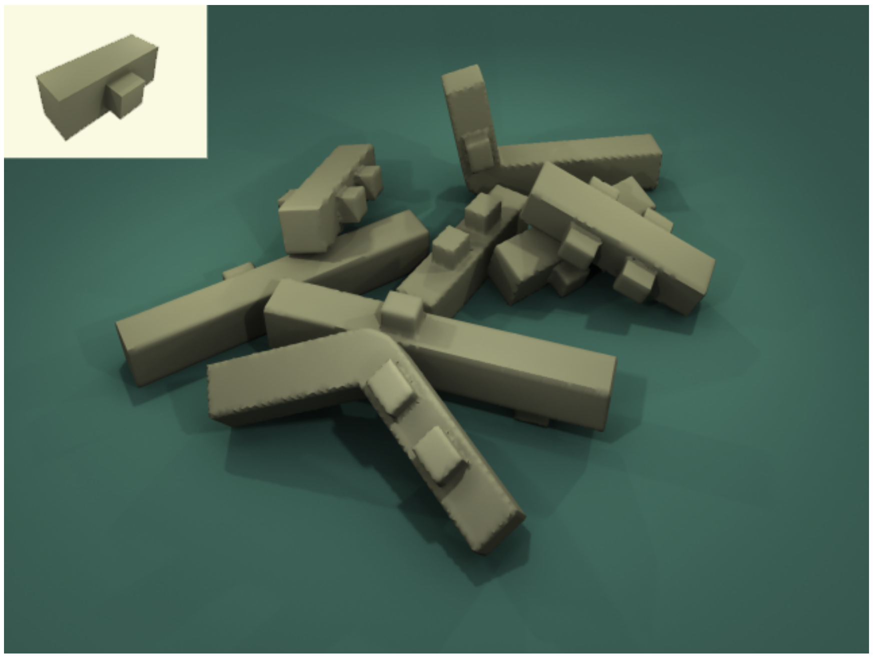


Fig. 55. Constructed blocks using articulation, and repetition and interchanging parts (Example 4). A peg is interchangeable with a flat region, and these elements can be repeatedly swapped across the model.

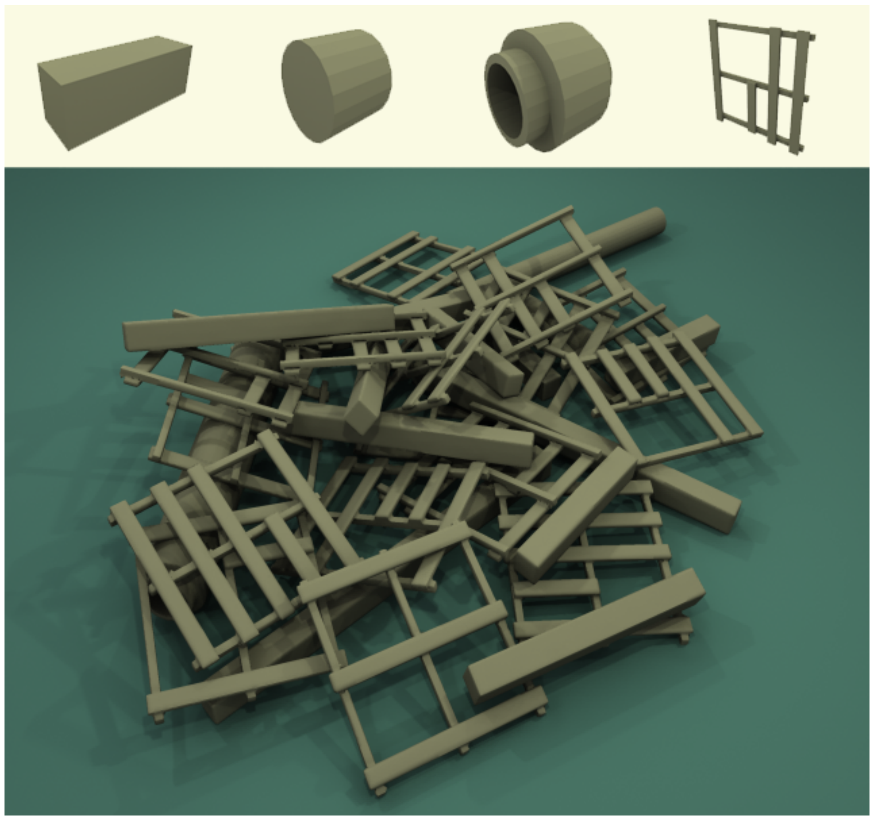


Fig. 56. Constructed environment similar to the provided real-world example that inspired this approach (Example 5). The different configurations of pallets were generated using random interchanging and repetition of board configurations.

larity within a given model to analyze scene detail for cleanup, compression, etc. [82][83][84]. While these approaches all extract patches/parts from a provided template, their underlying purpose is for very different reasons. These methods all focus on model repair, higher quality reconstructions, and geometry synthesis of existing models/datasets as their underlying rationale. Our approach is different as we use the patches as underlying primitives in a construction process, for fully defining the boundary of an object.

E. Limitations and Future Work

Our generative approach allows the definition of many different object configurations, but currently has several limitations that hinder its creativeness. First, it only allows fitting of patches from within a single provided model. It also requires two adjacent patches being fit to have smooth and similar transitions in overlapping areas (i.e., to correctly join the patches together). MLS can smooth small discontinuities, but large gaps between patches will result in poorly constructed models. Finally, the parameterizations proposed allow for only limited control over the object being constructed.

In order for this approach to be more effective, patches/parts need to be obtained and analyzed from a database of models, rather than just from a single object. Deformation/manipulation of patches to ensure proper fitting between adjacent patches would also have to be performed (e.g., possibly through an approach similar to that described by Pauly et al. [43]). Better and more expressive parameterizations also need to be developed. Each of these ideas allows our approach to be more creative, as well as other improvements that can be made to address construction quality and performance.

The “stretching” approach we use for filling the voids caused by articulation simply extends the patch across the gap, and may not generate ideal results. In some cases a sharp corner may look more natural, and in other cases a curved transition may look best. Allowing various void filling methods based on user preference, or identified behaviors in the template model, could be incorporated to provide better quality constructions. Another alternative is to use deformation of other nearby similar patches across the void. These alternative methods would simply replace our existing technique, as each of these methods would simply generate additional “filler” points in a similar fashion as the existing technique.

If a parameterization contains a high number of joints (e.g., as in the case of a very irregular free form sketch), the construction process will be much more expensive as many smaller voids must be filled for the larger set of defined joints. Our current approach will take much longer as each joint has to be addressed. Improving the efficiency of the void filling algorithm through alternative methods (e.g., replacing the adjacent neighbor identification, linking, and sub-sampling process) or parallelization of existing methods (e.g., handling joints is a localized problem that could be addressed independently of others) would allow this process to be more efficient and scale to larger parameterizations.

Finally, there are several additional ways in which our work might be extended. First, the sketch-based approach can be extended beyond the simple definition we use, to a more expressive system that allows the user to draw symbols and better defined illustrations for annotating desired behaviors. Second, by alleviating the user from annotating the input models (for marking specialized regions), and replacing this step with a more automated approach, the overall construction process becomes much simpler. Performing a formal analysis and verification of the generated Petri nets could ensure sound structure, prove there are no deadlocks, and allow for an interesting analysis of underlying properties of these generated networks and the objects they construct. The incorporation of these different ideas, combined with addressing this approach's current limitations, would allow a more flexible approach that could generate a wider variety of models.

CHAPTER V

EXTENSION TO SIMULATION

A. Overview

The overall goal of our work is to develop a virtual environment consisting of one or more individual solid model representations that can be easily incorporated into any application. In particular, we are very interested in how these objects can support simulation-based applications. In order to demonstrate the potential capabilities of our approach to this domain, we provide examples of several constructed environments using our approaches into a physically-based simulation. We also describe the general process used for performing this incorporation, as well as alternative methods necessary for possible migration to other applications.

1. Convex Hull Generation

Many simulations and games cannot provide real-time performance using a collection of fully detailed models. Thus, most applications use the fully detailed model for rendering, but use a simpler convex hull representation for computational purposes. This allows real-time physics calculations where the convex hull is used to determine object/object interactions, with the resulting effects. Since our goal is to integrate these objects into a simulation, a convex representation must be generated for each constructed object. We use an existing convex decomposition method that can quickly generate a simple approximation of the input object [114]. Figure 57 shows an example of a constructed model along with its generated convex hull. Once the hull has been generated, the final step in our process integrates the object into the simulation.

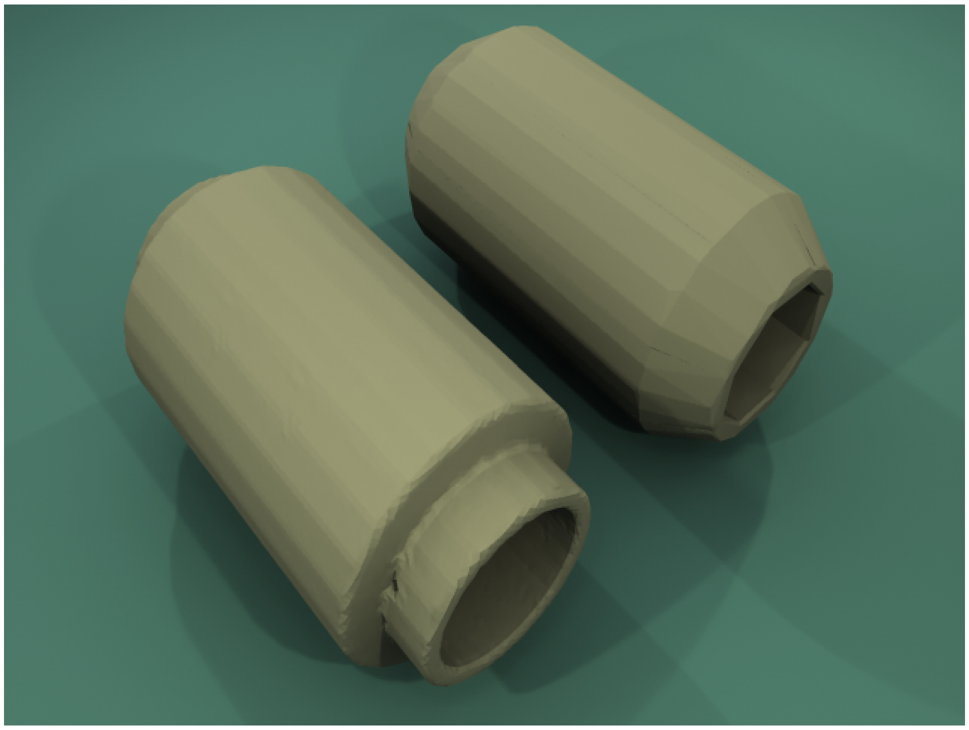


Fig. 57. Constructed object with its corresponding convex hull.

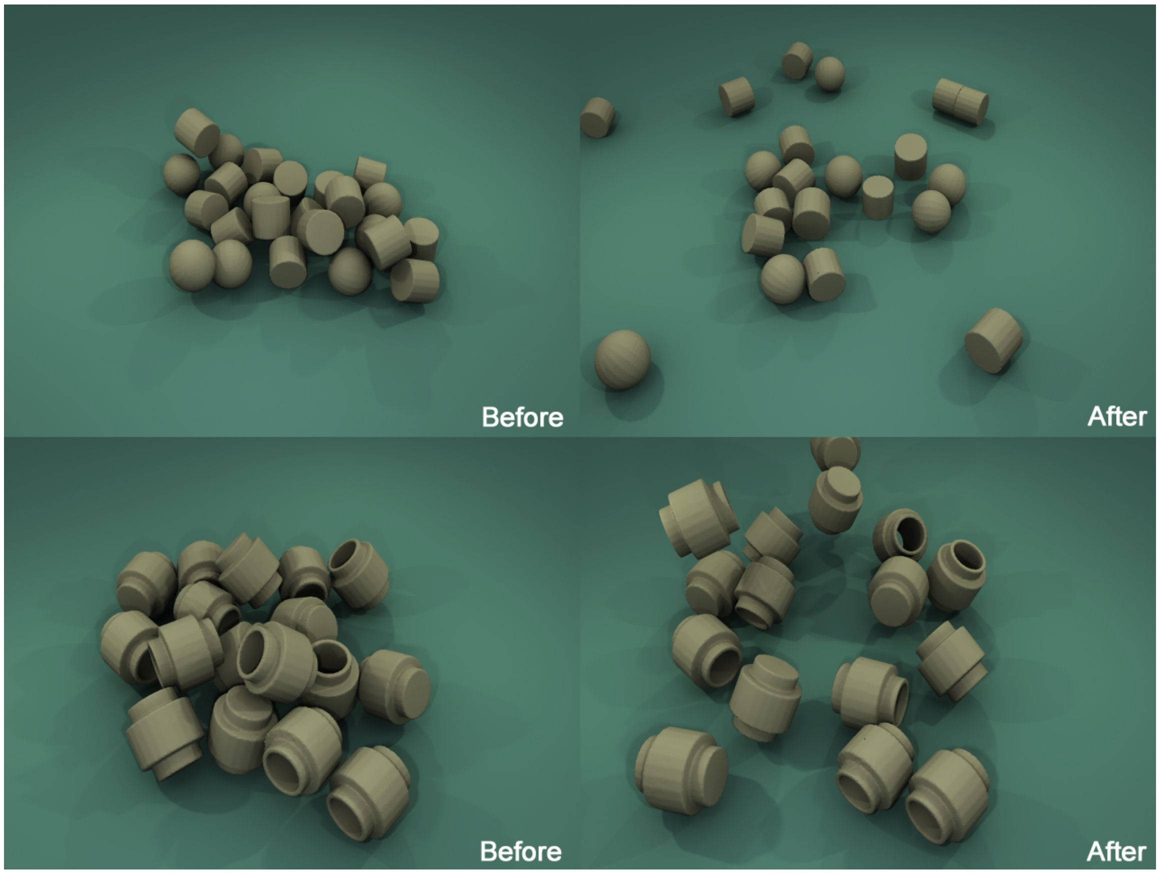


Fig. 58. Results from two different synthetic datasets (Examples 1 and 2) reconstructed and incorporated into a physically-based simulation.

B. Point Cloud Reconstruction Examples

Figures 58, 59, and 60 show the integration of several reconstructed environments into a physically-based simulation (i.e., a synthetic, a scan-based, and a photo-based dataset). Note that in these examples, each reconstructed object is a complete solid meshed model. As a result, each is capable of interacting and influencing others within the simulation. These figures show the point cloud reconstruction, and the resulting effects after a force has been applied to the object pile.

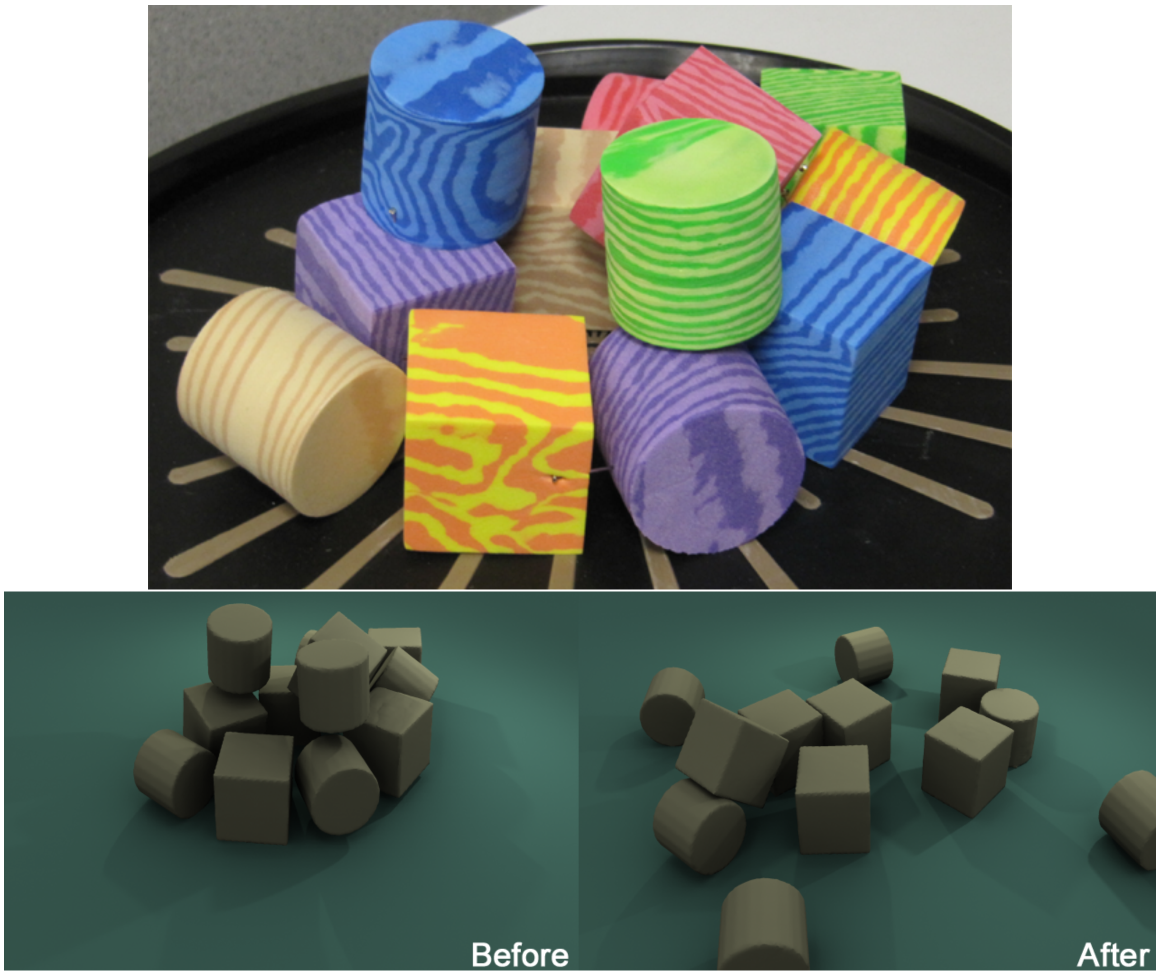


Fig. 59. Results from a scan-based dataset (Example 7) reconstructed and incorporated into a physically-based simulation.

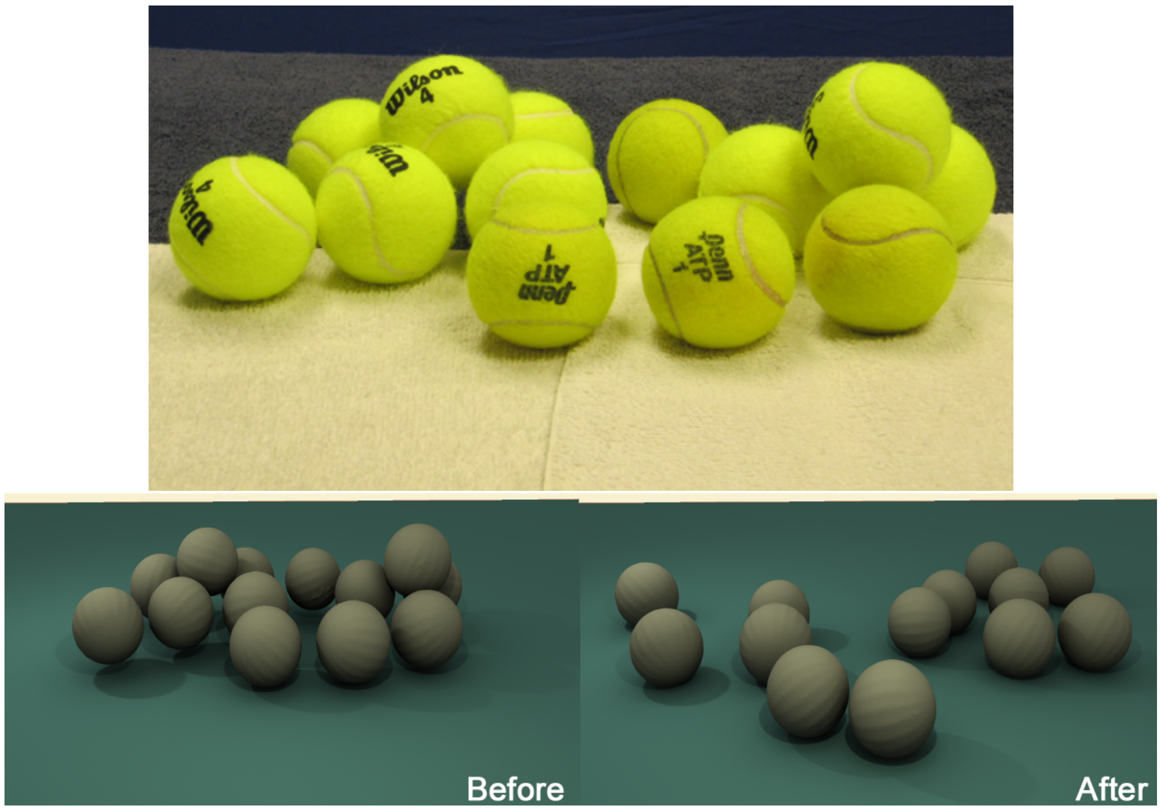


Fig. 60. Results from a photo-based dataset (Example 9) reconstructed and incorporated into a physically-based simulation.

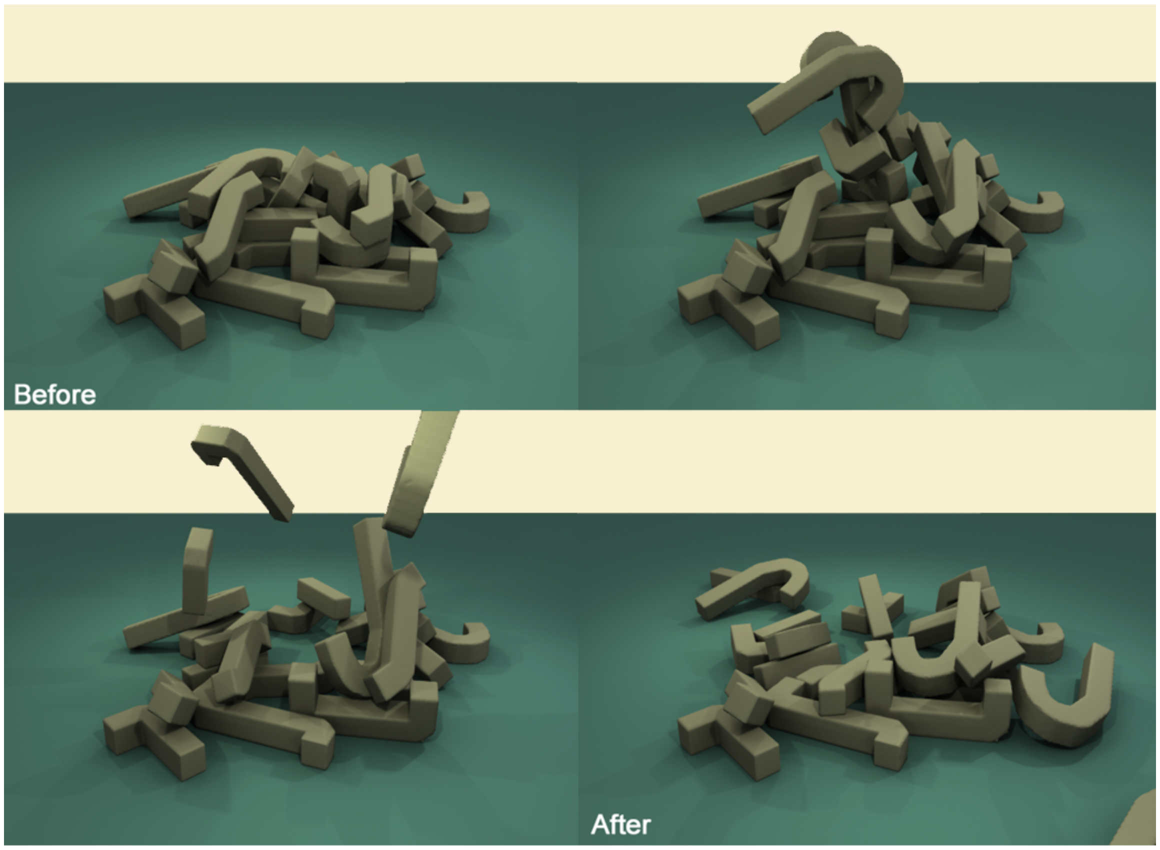


Fig. 61. Results of a generative modeling-based environment incorporated into a physically-based simulation.

C. Generative Modeling Examples

Figure 61 shows the integration of a generative modeled environment into a simulation. In this example we show the before, during, and after images of an upwards force being applied to the center of the pile of objects. As illustrated, once the force is applied the underlying dynamics cause a shift in the set of objects. Thus, these generated solid models can be easily integrated and used in many different simulation-based applications.

1. External Simulation Integration

Finally, each of the previous examples shows integrating a constructed environment directly into a simulation. These environments are also easily incorporated into other commercial systems by simply exporting the environment into a portable format.

There are many available formats that provide transportable representations for interchanging models between different applications. For this example we chose the COLLADA (COLLAborative Design Activity) file format, an XML-based representation for easily transporting three-dimensional models [115]. COLLADA provides a flexible and powerful format, and has been used by many companies including Autodesk, Google, and Sony. The process begins by first converting the set of constructed objects into a COLLADA formatted file. This file describes the detailed structure of the modeled objects (i.e., the vertex and face details). It can also describe other elements of the objects such as the visual attributes (i.e., color, texture, etc.) and physical attributes (i.e., mass, friction, etc.) if they are available. This representation contains all of the information required for the set of constructed objects to be imported into another system.

To demonstrate the integration of a reconstructed environment within an external simulation, we chose to incorporate the results from a simple reconstruction into the Unreal Engine [116]. The Unreal Engine provides a very flexible environment for incorporating external content. It also supports the COLLADA format for importing models. Thus, using the Unreal Editor [117], the generated models can be easily integrated.

To construct a simulated environment using these imported objects, a secondary tool was developed to export an environment in the Unreal T3D format, a text formatted file that characterizes the details of the environment including lighting details,

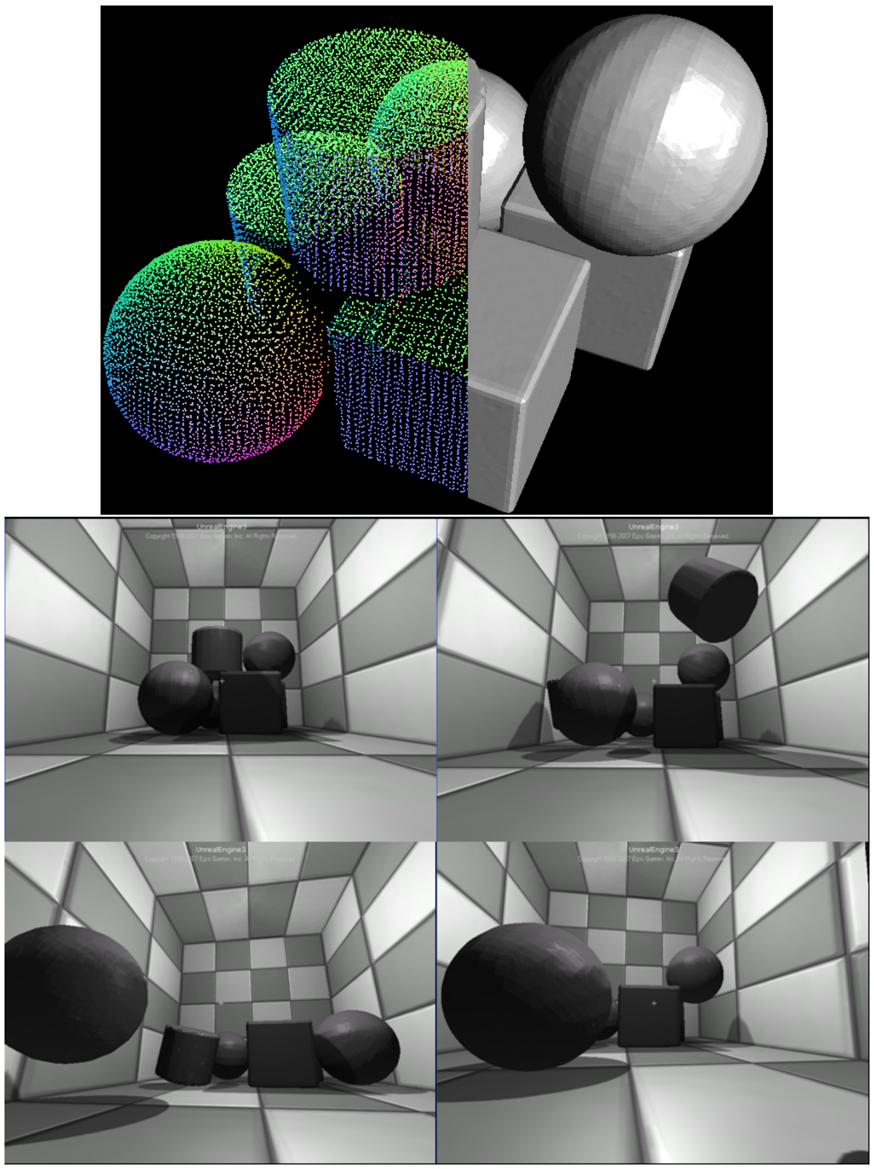


Fig. 62. A reconstructed environment incorporated into a commercial game engine.

position and orientation of the defined objects, etc. Our process automatically exports this file from the reconstructed results directly into the Unreal Editor. The combination of the COLLADA and T3D files fully describe the environment needed for simulation. Once imported, this environment can then be compiled and loaded as a game environment where the user can manipulate objects in whatever fashion required. There are many potential applications for such a constructed world.

Figure 62 shows an example of this environment in action. Note that while we chose to integrate this example into the Unreal Engine, the presented approach could easily extend to other applications equally well. For example, it could be used to integrate reconstructed environments into systems such as OLIVE [118], Second Life [119], RealWorld [120] or other similar virtual environments.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

A. Conclusions

In this dissertation, we have introduced two inference-based geometric modeling algorithms for developing solid models under different conditions. We have shown that these methods provide different alternatives for re-creating complex virtual environments in two different fashions. We have described the background and technical details behind these approaches, and evaluated them on several different types of datasets. Finally, we have demonstrated the application of our results within the simulation domain to illustrate our underlying motivation.

We believe that our inference-based surface reconstruction algorithm provides an effective means of reconstructing point cloud representations of cluttered environments. We have shown that this approach is capable of recognizing and reconstructing solid models, even amongst heavy clutter and object occlusion. We also believe that our generative modeling approach provides an efficient means for re-creating various models based on a user provided template. We have shown that this process can function in both a fully automated and semi-automated fashion, and can robustly construct solid model definitions that include articulation, and repetition and interchanging of parts obtained from a template model.

While each of these methods provide a means for modeling objects under different conditions, the broader objective of our work is to develop high-fidelity virtual environments for anything from games and entertainment, to various simulation-based applications. In each of these application areas, having a solid representation of each object for simulation purposes is critical. Figures 58, 59, 60, and 61 provide a hint of how such an application could work. Figures 58, 59 and 60 show examples where an initial environment capture is reconstructed and ultimately used to support a simulation. Figure 61 shows an example where an environment is automatically generated, and also incorporated into a simulation. Overall, our techniques help to address the requirements of many different application areas by supporting the automated construction of more complex environments in a quicker and easier fashion.

B. Future Work

There are many future extensions to this work and we have discussed several specific ideas already. However, we envision several broader directions this work could take to lead to future research problems. We briefly discuss a few such ideas.

The approaches we have presented have all been focused on using automated techniques. While automation is important because it reduces the burden on the user, it also limits overall effectiveness and robustness of an approach when working with more complex data and/or environments. The first significant extension of this work is to analyze and better incorporate user interaction into the overall construction process. Previously we proposed sketch-based interaction to allow user intervention to help resolve ambiguity or uncertainty during reconstruction. However, allowing sketch-based interfaces where a user could very quickly annotate their desired behavior, followed by an automated construction algorithm that understands how to digest this information and construct the corresponding scene elements, would be an incredibly powerful capability for modeling. Our generative modeling approach attempts to take an initial step in this direction, but much more complex gestures/behaviors need to be incorporated that allow for better definition of a scene and the individual objects within it. This underlying idea could be applied to both the reconstruction and generative approaches presented.

The second extension we propose is to incorporate the notion of physics into our overall approach. We previously mentioned incorporating physics as a relaxation step during the reconstruction algorithm to obtain more natural looking results. However, future work could also address "cause and effect" relationships within a constructed environment. What if the scene in Figure 1 represented the "after state" of a collapse? Obtaining a reconstruction of this "after state" and then analyzing the state of objects, how they are resting, their physical properties, etc., combined with a physically-based model, could provide a means of stepping backwards to determine the original state of an environment. This could allow a better understanding of the cause of a collapse, and a glimpse of the before and after states.

Finally, we have presented the two methods in a separate fashion. However, these methods could be combined to form an integrated system that allows the development of a wider range of environments. For example, trying to reconstruct solid models from a real-world capture of the environment shown in Figure 1 would be a very challenging problem. While there are many different similar objects within it, they all have substantial differences. There are also many objects that are incomplete and in pieces. Using a combination of the two techniques could allow general identification of objects that are visible within the point cloud data, followed by constructing general approximations using generative modeling. Once all of the visible objects have been constructed, the generative approach could then be used to develop representative objects that reside lower in the pile and not necessarily visible within a capture. Such an approach would have to analyze potential voids and incorporate the physical nature of the environment to determine correct placement of objects within a pile. Combining these two techniques would allow a more complete re-creation of the real-world environment, while avoiding the manual modeling of such a complex scene.

REFERENCES

[1] H. Müller, "Surface Reconstruction - An Introduction," *Scientific Visualization Conference*, pp. 239–242, IEEE Computer Society Press, 1997.

[2] L. Kobbelt and M. Botsch, "A Survey of Point-Based Techniques in Computer Graphics," *Computers & Graphics*, vol. 28, no. 6, pp. 801–814, 2004.

[3] M. Sainz and R. Pajarola, "Point-Based Rendering Techniques," *Computers & Graphics*, vol. 28, no. 6, pp. 869–879, 2004.

[4] N. Amenta and Y. J. Kil, "Defining Point-Set Surfaces," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 264–270, 2004.

[5] W. E. Lorensen and H. E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 163–169, ACM, 1987.

[6] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, "The Digital Michelangelo Project: 3D Scanning of Large Statues," *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 131–144, ACM Press/Addison-Wesley Publishing Co., 2000.

[7] P. Allen, I. Stamos, A. Troccoli, B. Smith, M. Leordeanu, and S. Murray, "New Methods for Digital Modeling of Historic Sites Using Range and Image Data," *IEEE Computer Graphics and Applications*, vol. 23, no. 6, pp. 32–41, 2003.

[8] U. of Washington Computer Science, "Bundler: Structure from Motion for Unordered Image Collections." http://phototour.cs.washington.edu/bundler/, February 2011.

[9] U. of Washington Computer Science, "Clustering Views for Multi-view Stereo (CMVS)." http://grail.cs.washington.edu/software/cmvs/, February 2011.

[10] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[11] S. Rusinkiewicz and M. Levoy, "Efficient Variants of the ICP Algorithm," *Proceedings of the Third International Conference on 3D Digital Imaging and Modeling*, pp. 145–152, IEEE Computer Society, 2001.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Points," *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 71–78, ACM, 1992.

[13] S. Rusinkiewicz and M. Levoy, "QSplat: A Multiresolution Point Rendering System for Large Meshes," *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 343–352, ACM Press/Addison-Wesley Publishing Co., 2000.

[14] T. Boubekeur, P. Reuter, and C. Schlick, "Local Reconstruction and Visualization of Point-Based Surfaces Using Subdivision Surfaces," *Computer Graphics & Geometry*, vol. 8, no. 1, pp. 22–40, 2006.

[15] P. Jenke, M. Wand, and W. Straßer, "Patch-Graph Reconstruction for Piecewise Smooth Surfaces," *Proceedings of Vision, Modeling and Visualization*, pp. 3–12, Academic Press AKA, 2008.

[16] H. Edelsbrunner and E. P. Mücke, "Three-dimensional Alpha Shapes," *ACM Transactions on Graphics*, vol. 13, no. 1, pp. 43–72, 1994.

[17] C. L. Bajaj, F. Bernardini, and G. Xu, "Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans," *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 109–118, ACM, 1995.

[18] N. Amenta, M. Bern, and M. Kamvysselis, "A New Voronoi-Based Surface Reconstruction Algorithm," *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 415–421, ACM, 1998.

[19] N. Amenta, S. Choi, T. Dey, and N. Leekha, "A Simple Algorithm for Homeomorphic Surface Reconstruction," *Proceedings of the 16th Annual Symposium on Computational Geometry*, pp. 213–222, ACM, 2000.

[20] T. K. Dey, J. Giesen, and J. Hudson, "Delaunay Based Shape Reconstruction from Large Data," *Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics*, pp. 19–27, IEEE Computer Society, 2001.

[21] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 303–312, ACM, 1996.

[22] J. Carr, R. Beatson, J. Cherrie, T. Mitchell, W. Fright, B. McCallum, and T. Evans, "Reconstruction and Representation of 3D Objects with Radial Basis Functions," *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 67–76, ACM, 2001.

[23] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and Rendering Point Set Surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 1, pp. 3–15, 2003.

[24] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson Surface Reconstruction," *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, pp. 61–70, Eurographics Association, 2006.

[25] L. P. Kobbelt and M. Botsch, "An Interactive Approach to Point Cloud Triangulation," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 19, no. 3, pp. 479–487, 2000.

[26] R. Mencl, *Reconstruction of Surfaces from Unorganized 3D Points Clouds.* PhD thesis, Dortmund University, North Rhine-Westphalia, Germany, 2001.

[27] T. Várady, R. R. Martin, and J. Cox, "Reverse Engineering of Geometric Models - An Introduction," *Computer-Aided Design*, vol. 29, no. 4, pp. 255–268, 1997.

[28] G. Turk and M. Levoy, "Zippered Polygon Meshes from Range Images," *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pp. 311–318, ACM, 1994.

[29] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The Ball-Pivoting Algorithm for Surface Reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[30] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel, "Multi-level Partition of Unity Implicits," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 463–470, 2003.

[31] I. Tobor, P. Reuter, and C. Schlick, "Efficient Reconstruction of Large Scattered Geometric Datasets Using the Partition of Unity and Radial Basis Functions," *Journal of WSCG 2004*, vol. 12, pp. 467–474, 2004.

[32] T. Boubekeur, W. Heidrich, X. Granier, and C. Schlick, "Volume-Surface

Trees," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 25, no. 3, pp. 399–406, 2006.

[33] P. J. Crossno and E. S. Angel, "Spiraling Edge: Fast Surface Reconstruction from Partially Organized Sample Points," *Proceedings of the 10th IEEE Visualization Conference*, pp. 317–324, IEEE Computer Society, 1999.

[34] M. Gopi, S. Krishnan, and C. Silva, "Surface Reconstruction Based on Lower Dimensional Localized Delaunay Triangulation," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 19, no. 3, pp. 363–371, 2000.

[35] C.-C. Kuo and H.-T. Yau, "A Delaunay-based Region-growing Approach to Surface Reconstruction from Unorganized Points," *Computer-Aided Design*, vol. 37, no. 8, pp. 825–835, 2005.

[36] I. Ivrissimtzis, W.-K. Jeong, and H.-P. Seidel, "Using Growing Cell Structures for Surface Reconstruction," *Proceedings of the Shape Modeling International 2003*, pp. 78–86, IEEE Computer Society, 2003.

[37] H. Hoppe, "Progressive Meshes," *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 99–108, ACM, 1996.

[38] R. Mencl and H. Müller, "Graph-Based Surface Reconstruction Using Structures in Scattered Point Sets," *Proceedings of the Computer Graphics International 1998*, pp. 298–311, IEEE Computer Society, 1998.

[39] S. Gumhold, X. Wang, and R. Macleod, "Feature Extraction from Point Clouds," *Proceedings of the 10th International Meshing Roundtable*, pp. 293–305, Sandia National Laboratories, 2001.

[40] S. Fleishman, D. Cohen-Or, and C. T. Silva, "Robust Moving Least-squares

Fitting with Sharp Features," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 544–552, 2005.

[41] J. Davis, S. Marschner, M. Garr, and M. Levoy, "Filling Holes in Complex Surfaces Using Volumetric Diffusion," *Proceedings of the First International Symposium on 3D Data Processing, Visualization, and Transmission*, pp. 428–438, IEEE Computer Society, 2002.

[42] A. Sharf, M. Alexa, and D. Cohen-Or, "Context-based Surface Completion," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 878–887, 2004.

[43] M. Pauly, N. J. Mitra, J. Giesen, L. J. Guibas, and M. Gross, "Example-based 3D Scan Completion," *Proceedings of the Third Eurographics Symposium on Geometry Processing*, pp. 23–32, Eurographics Association, 2005.

[44] V. Kraevoy and A. Sheffer, "Template-Based Mesh Completion," *Proceedings of the Third Eurographics Symposium on Geometry Processing*, pp. 13–22, Eurographics Association, 2005.

[45] R. Gal, A. Shamir, T. Hassner, M. Pauly, and D. Cohen-Or, "Surface Reconstruction Using Local Shape Priors," *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, pp. 253–262, Eurographics Association, 2007.

[46] S. Shalom, A. Shamir, H. Zhang, and D. Cohen-Or, "Cone Carving for Surface Reconstruction," *ACM Transactions on Graphics*, vol. 29, no. 6, pp. 1–10, 2010.

[47] R. Schnabel, P. Degener, and R. Klein, "Completion and Reconstruction with Primitive Shapes," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 28, no. 2, pp. 503–512, 2009.

[48] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing and Modeling: A Procedural Approach, 3rd Edition.* San Francisco, CA: Morgan Kaufmann Publishers Inc., 2002.

[49] G. Kelly and H. McCabe, "A Survey of Procedural Techniques for City Generation," *Institute of Technology Blanchardstown Journal*, vol. 14, pp. 87–130, 2006.

[50] B. Watson, P. Müller, P. Wonka, C. Sexton, O. Veryovka, and A. Fuller, "Procedural Urban Modeling in Practice," *IEEE Computer Graphics and Applications*, vol. 28, no. 3, pp. 18–26, 2008.

[51] I. Procedural, "CityEngine: 3D Modeling Software for Urban Environments." http://www.procedural.com/, February 2011.

[52] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants.* New York, NY: Springer-Verlag, Inc., 1990.

[53] I. D. V. Inc., "SpeedTree." http://www.speedtree.com/, February 2011.

[54] R. M. Smelik, K. J. de Kraker, S. A. Groenewegen, T. Tutenel, and R. Bidarra, "A Survey of Procedural Methods for Terrain Modeling," *Proceedings of the CASA Workshop on 3D Advanced Media in Gaming and Simulation* (A. Egges, W. Hürst, and R. C. Veltkamp, eds.), pp. 25–34, 2009.

[55] P. Software, "Planetside." http://www.planetside.co.uk/, February 2011.

[56] P. Inc., "MojoWorld." http://www.mojoworld.org/, February 2011.

[57] M. Barnsley, *Fractals Everywhere, 2nd Edition.* San Diego, CA: Academic Press Professional, Inc., 1993.

[58] K. Perlin, "An Image Synthesizer," *SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[59] A. Lindenmayer, "Mathematical Models for Cellular Interactions in Development I and II," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 280–315, 1968.

[60] S. Lefebvre and F. Neyret, "Pattern Based Procedural Textures," *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, pp. 203–212, ACM, 2003.

[61] G. N. Stiny, *Pictorial and Formal Aspects of Shape and Shape Grammars and Aesthetic Systems.* PhD thesis, University of California, Los Angeles, CA, 1975.

[62] Y. I. H. Parish and P. Müller, "Procedural Modeling of Cities," *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 301–308, ACM, 2001.

[63] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural Modeling of Buildings," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 614–623, 2006.

[64] E. Whiting, J. Ochsendorf, and F. Durand, "Procedural Modeling of Structurally-Sound Masonry Buildings," *ACM Transactions on Graphics*, vol. 28, no. 5, pp. 1–9, 2009.

[65] P. Merrell, E. Schkufza, and V. Koltun, "Computer-Generated Residential Building Layouts," *ACM Transactions on Graphics*, vol. 29, no. 6, pp. 1–12, 2010.

[66] R. Měch and P. Prusinkiewicz, "Visual Models of Plants Interacting with Their

Environment," *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 397–410, ACM, 1996.

[67] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane, "The Use of Positional Information in the Modeling of Plants," *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 289–300, ACM, 2001.

[68] J. Weber and J. Penn, "Creation and Rendering of Realistic Trees," *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 119–128, ACM, 1995.

[69] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic Modeling and Rendering of Plant Ecosystems," *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 275–286, ACM, 1998.

[70] J. Lluch, E. Camahort, and R. Vivó, "Procedural Multiresolution for Plant and Tree Rendering," *Proceedings of the Second International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pp. 31–38, ACM, 2003.

[71] R. Geiss, *GPU Gems 3*, ch. 1 - Generating Complex Procedural Terrains Using the GPU. New York, NY: Addison-Wesley Publishing Co., 2007.

[72] P. Merrell, "Example-Based Model Synthesis," *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pp. 105–112, ACM, 2007.

[73] P. Merrell and D. Manocha, "Continuous Model Synthesis," *ACM Transactions on Graphics*, vol. 27, no. 5, pp. 1–7, 2008.

[74] P. Merrell and D. Manocha, "Constraint-based Model Synthesis," *Proceedings of the 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pp. 101–111, ACM, 2009.

[75] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for Point-Cloud Shape Detection," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 26, no. 2, pp. 214–226, 2007.

[76] B. Jenke, P. Krückeberg and W. Straßer, "Surface Reconstruction from Fitted Shape Primitives," *Proceedings of Vision, Modeling and Visualization 2008*, pp. 31–40, IOS Press, 2008.

[77] R. Schnabel, R. Wessel, R. Wahl, and R. Klein, "Shape Recognition in 3D Point-Clouds," *Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pp. 1–8, UNION Agency-Science Press, 2008.

[78] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin, "Modeling by Example," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 652–663, 2004.

[79] V. Kraevoy, D. Julius, and A. Sheffer, "Model Composition from Interchangeable Components," *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pp. 129–138, IEEE Computer Society, 2007.

[80] R. Gal and D. Cohen-Or, "Salient Geometric Features for Partial Shape Matching and Similarity," *ACM Transactions on Graphics*, vol. 25, no. 1, pp. 130–150, 2006.

[81] S. Shalom, L. Shapira, A. Shamir, and D. Cohen-Or, "Part Analogies in Sets

of Objects," *Proceedings of Eurographics Symposium on 3D Object Retrieval*, pp. 33–40, Eurographics Association, 2008.

[82] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas, "Discovering Structural Regularity in 3D Geometry," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 1–11, 2008.

[83] A. Berner, M. Bokeloh, M. Wand, A. Schilling, and H.-P. Seidel, "A Graph-Based Approach to Symmetry Detection," *Proceedings of the IEEE/EG International Symposium on Volume and Point-Based Graphics*, pp. 1–8, Eurographics Association, 2008.

[84] M. Bokeloh, A. Berner, M. Wand, H.-P. Seidel, and A. Schilling, "Symmetry Detection Using Feature Lines," *Computer Graphics Forum (Proceedings of Eurographics)*, vol. 28, no. 2, pp. 697–706, 2009.

[85] J. Han, M. Pratt, and W. C. Regli, "Manufacturing Feature Recognition from Solid Models: A Status Report," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 6, pp. 782–796, 2000.

[86] J. J. Shah, D. Anderson, Y. Se Kim, and S. Joshi, "A Discourse on Geometric Feature Recognition from CAD Models," *Journal of Computing and Information Science in Engineering*, vol. 1, no. 1, pp. 41–51, 2001.

[87] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision, 2nd Edition.* Pacific Grove, CA: International Thomson Publishing, Inc., 1998.

[88] R. J. Campbell and P. J. Flynn, "A Survey of Free-Form Object Representation and Recognition Techniques," *Computer Vision and Image Understanding*, vol. 81, no. 2, pp. 166–210, 2001.

[89] E. Borenstein and J. Malik, “Shape Guided Object Segmentation,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 969–976, IEEE Computer Society, 2006.

[90] T. Cour and J. Shi, “Recognizing Objects By Piecing Together the Segmentation Puzzle,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE Computer Society, 2007.

[91] Y. Lamdan and H. Wolfson, “Geometric Hashing: A General and Efficient Model-based Recognition Scheme,” *Second International Conference on Computer Vision*, pp. 238–249, IEEE Computer Society Press, 1988.

[92] A. Johnson and M. Hebert, “Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433–449, 1999.

[93] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik, “Recognizing Objects in Range Data Using Regional Point Descriptors,” *European Conference on Computer Vision*, pp. 224–237, Springer-Verlag, Inc., 2004.

[94] D. Huber, A. Kapuria, R. R. Donamukkala, and M. Hebert, “Parts-based 3D Object Classification,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 82–89, IEEE Computer Society, 2004.

[95] D. Aiger, N. J. Mitra, and D. Cohen-Or, “4-Points Congruent Sets for Robust Surface Registration,” *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 1–10, 2008.

[96] A. N. Vasile and R. M. Marino, “Pose-Independent Automatic Target Detection

and Recognition Using 3D Laser Radar Imagery," *Lincoln Laboratory Journal*, vol. 15, no. 1, pp. 61–78, 2005.

[97] J. W. Tangelder and R. C. Veltkamp, "A Survey of Content Based 3D Shape Retrieval Methods," *Proceedings of the Shape Modeling International*, pp. 145–156, IEEE Computer Society, 2004.

[98] B. Bustos, D. A. Keim, D. Saupe, T. Schreck, and D. V. Vranić, "Feature-based Similarity Search in 3D Object Databases," *ACM Computing Surveys*, vol. 37, no. 4, pp. 345–387, 2005.

[99] T. Funkhouser, M. Kazhdan, P. Min, and P. Shilane, "Shape-based Retrieval and Analysis of 3D Models," *Communications of the ACM*, vol. 48, no. 6, pp. 58–64, 2005.

[100] G. Turk, "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion," *Computer Graphics*, vol. 25, no. 4, pp. 289–298, 1991.

[101] M. Elad, A. Tal, and S. Ar, "Content Based Retrieval of VRML Objects: An Iterative and Interactive Approach," *Proceedings of the Sixth Eurographics Workshop on Multimedia*, pp. 107–118, Springer-Verlag, Inc., 2002.

[102] 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," *Proceedings of the IEEE/EG Symposium on Volume and Point-Based Graphics*, pp. 1–15, Eurographics Association, 2008.

[103] G. M. Nielson and B. Hamann, "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes," *Proceedings of the Second Conference on Visualization*, pp. 83–91, IEEE Computer Society Press, 1991.

[104] C. Montani, R. Scateni, and R. Scopigno, "A Modified Look-up Table for Implicit Disambiguation of Marching Cubes," *The Visual Computer*, vol. 10, no. 6, pp. 353–355, 1994.

[105] C. Montani, R. Scateni, and R. Scopigno, "Discretized Marching Cubes," *Proceedings of the Conference on Visualization*, pp. 281–287, IEEE Computer Society Press, 1994.

[106] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature Sensitive Surface Extraction from Volume Data," *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 57–66, ACM, 2001.

[107] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual Contouring of Hermite Data," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 339–346, 2002.

[108] S. Schaefer, T. Ju, and J. Warren, "Manifold Dual Contouring," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 610–619, 2007.

[109] S. Schaefer and J. Warren, "Dual Marching Cubes: Primal Contouring of Dual Grids," *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, pp. 70–76, IEEE Computer Society, 2004.

[110] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe, "Unconstrained Isosurface Extraction on Arbitrary Octrees," *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, pp. 125–133, Eurographics Association, 2007.

[111] J. L. Peterson, "Petri Nets," *ACM Computing Surveys*, vol. 9, no. 3, pp. 223–252, 1977.

[112] H. Abelson and A. diSessa, *Turtle Geometry: The Computer As a Medium for Exploring Mathematics.* Cambridge, MA: MIT Press, 1986.

[113] H. Sagan, *Space-Filling Curves.* New York, NY: Springer-Verlag, Inc., 1994.

[114] J. W. Ratcliff, "Convex Decomposition: A Convex Decomposition Library." http://code.google.com/p/convexdecomposition/, February 2011.

[115] K. Group, "COLLADA Overview." http://www.khronos.org/collada/, February 2011.

[116] E. Games, "Unreal Technology." http://www.unrealtechnology.com/, February 2011.

[117] J. Busby, Z. Parrish, and J. VanEenwyk, *Mastering Unreal Technology: The Art of Level Design.* Indianapolis, IN: Sams Publishing, 2005.

[118] SAIC, "OLIVE." http://www.forterrainc.com/, February 2011.

[119] L. R. Inc., "Second Life." http://secondlife.com/, February 2011.

[120] T. I. S. Inc., "RealWorld Platform." http://www.realworld-sim.com/, February 2011.

[121] G. Turk, *Texturing Surfaces Using Reaction-Diffusion.* PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, 1992.

[122] M. Gross and H. Pfister, *Point-Based Graphics (The Morgan Kaufmann Series in Computer Graphics).* San Francisco, CA: Morgan Kaufmann Publishers Inc., 2007.

[123] S. Schaefer, T. McPhail, and J. Warren, "Image Deformation Using Moving Least Squares," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 533–540,

2006.

[124] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa, "Point Based Animation of Elastic, Plastic and Melting Objects," *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 141–151, Eurographics Association, 2004.

[125] D. Levin, "The Approximation Power of Moving Least-Squares," *Mathematics Computing*, vol. 67, no. 224, pp. 1517–1531, 1998.

[126] D. Levin, *Geometric Modeling for Scientific Visualization*, ch. 1 - Mesh-independent Surface Interpolation, pp. 37–49. New York, NY: Springer-Verlag, Inc., 2003.

[127] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition.* New York, NY: Cambridge University Press, 1992.

[128] C. Petri, *Kommunikation mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

APPENDIX A

UNIFORM SAMPLING OF A MESHED MODEL

The process of uniformly sampling a meshed surface was discussed as part of both modeling approaches previously presented. Uniform sampling of a mesh can be accomplished using the method defined by Turk [100], and in the remainder of this appendix, we explain his algorithm in more detail.

This uniform sampling process is broken down into two main stages. A random surface sampling is performed first, followed by a relaxation procedure. This algorithm outputs a set of samples that resides on the surface of a model that is (approximately) uniformly distributed across its surface. Figure 63 shows an example of both the initial random sampling of a model, and the uniform sampling once the relaxation has been performed. The input to this process is a triangulated meshed model M and a defined sample size n. The user can select a desired value for n, or this value can be automatically chosen based on the total surface area of the model or some other heuristic method.

The sampling process begins by first distributing the n samples randomly on the surface of M (i.e., choosing from the total set of model faces F obtained from M). The position of each sample μ is selected such that it lies on a polygonal face. A face $f \in F$ in which to place μ is chosen by using a random area-weighted selection criteria. A list of all face areas is calculated and sorted during a precomputation step. A random selection from this list can then be found efficiently using a binary search through the list of partial sums calculated from the areas stored in this list. Once a face has been selected, μ is positioned within f using randomly chosen barycentric coordinates.

Once all of the samples have been randomly placed on the surface of M, the second stage of this process performs a relaxation procedure that allows for a more regular distribution of the samples across the surface. This procedure is performed by iteratively repelling the nearby neighbors of each sample, thereby evenly spacing the samples on the surface. In order for two samples to repel each other, they must be within a repulsive radius r of each other (defined in Equation A.1):

$$r = 2\sqrt{a/n} \tag{A.1}$$

where a is the total area of the surface, and n is the number of samples to place on the surface. If two samples are greater than r distance away, they will not affect each other and the repulsive force falls off linearly with distance.

Algorithm 10

```
1.   for $i \leftarrow 1$ to $k$
2.
3.        for $\mu \in samples$
4.             $A = GetTheFaceSampleLiesOn(\mu, F)$;
5.             $near\_points = DetermineNearbySamples(\mu, samples)$;
6.             for $\gamma \in near\_points$
7.                  $MapNearbyPointOntoPlane(\gamma, A)$;
8.             $V = ComputeRepulsiveForce(\mu, near\_points)$;
9.             $StoreRepulsiveForce(\mu, V, forces)$;
10.
11.       for $\mu \in samples$
12.            $\mu_{new} = ComputeNewPositionFromForces(\mu, forces)$;
13.            $MapNearbyPointBackOntoSurface(\mu_{new}, F)$;
```

The relaxation process is described in Algorithm 10 and is performed over k iterations. The process begins by first finding the nearby samples of μ by identifying all those samples that reside within a repulsive radius r of μ. The total force applied to μ from these nearby samples is then determined. The total force applied to a sample μ that resides on a face A is calculated by mapping all of the nearby samples

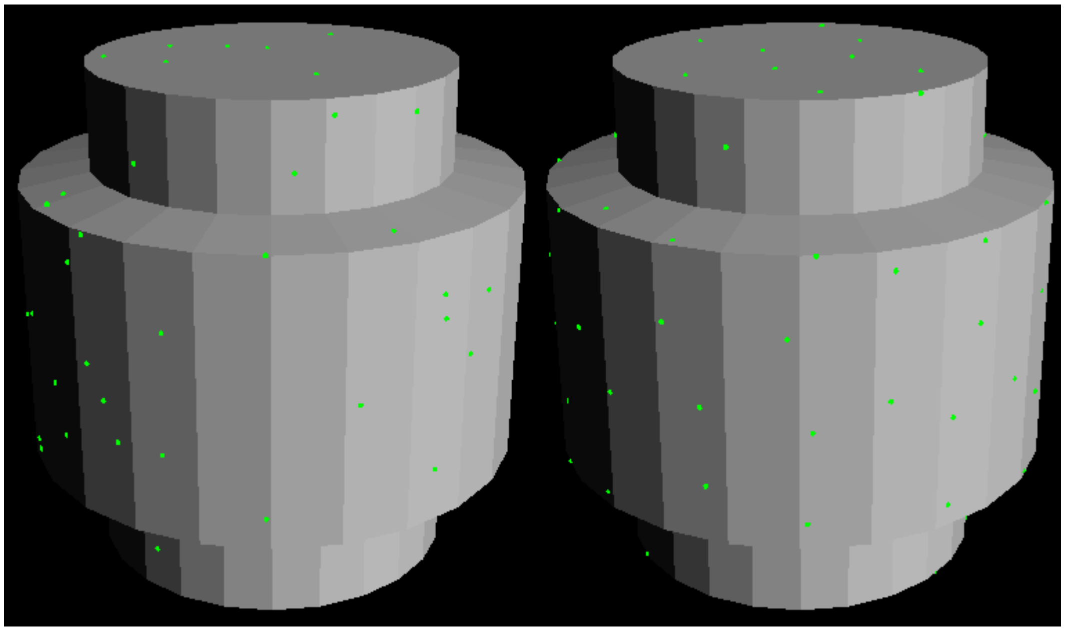


Fig. 63. An initial random sampling (left) and uniform sampling after relaxation (right).

onto the plane defined by A. A nearby sample is mapped onto this plane according to one of the following rules:

- A sample γ that already lies on A, remains where it is defined.
- A sample γ that lies on a face B that shares an edge with A, is rotated about the shared edge between A and B until γ lies on the plane defined by A.
- A sample γ that is on a face B that is not adjacent to A (i.e., it is *remote*) is first rotated about the nearest edge of A, and then projected onto the plane defined by A.

Figure 64 provides an illustration (similar to that provided by Turk [121]) of how this mapping process functions for both an adjacent face, as well as a remote face.

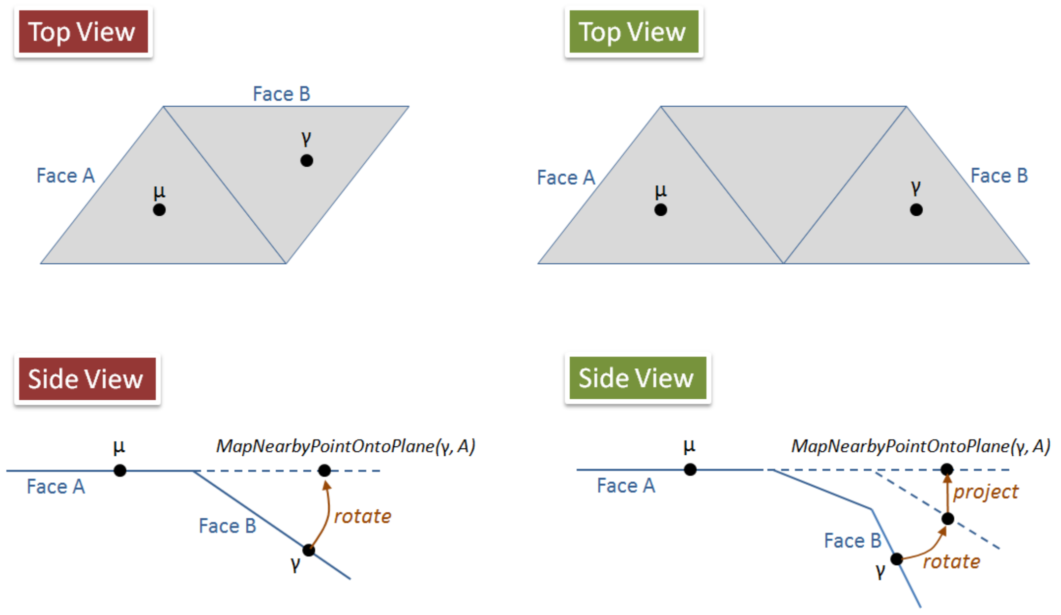


Fig. 64. An illustration of the mapping of samples onto a common plane for both adjacent faces (left) and remote faces (right).

Once the samples have been mapped onto a common plane, then a vector V can be found for μ that stores the sum of all repelling forces from the nearby samples. A corresponding V is found for each sample, then a second pass is made where the forces are applied to each sample and a new position μ_{new} is determined for each (as defined in Equation A.2):

$$\mu_{new} = \mu + kV \tag{A.2}$$

where μ is the original position of the sample, V is the vector storing the summed forces, and k is a small scaling factor that can help control the magnitude of sample displacement during each iteration.

If μ_{new} still lies within the bounds of the face A, then the process moves on to the next sample. However, if μ_{new} is not within the bounds of the face A, then it is no longer on the surface of M and must be mapped back to its appropriate position on the surface. The edge of A that μ was pushed across must first be identified, and the adjacent face B that shares this edge is found. Next, the sample μ_{new} is rotated about the shared edge between A and B such that it will lie on the plane defined by the face B. This process is performed iteratively until μ_{new} lies on the surface of M (or within a defined tolerance). Each step in this process gets μ_{new} closer to the surface, and after several iterations, μ_{new} should lie on the surface. Most faces should share an edge with another face, but if a sample is moved across an edge with no defined adjacent face, then it is simply moved back to the nearest position such that it lies on a face of M.

This sampling process can be used for both the full surface of an object, as well as a localized region (e.g., for the sub-sampling step described in Chapters III and IV). In order to sub-sample a localized region on the surface of an object, the usable surface region must first be found. Given a surface sample, this region is found using the surface intersection approach described in Chapter III. To subsample the patch found, an extension of the process described for full surface sampling can be used. The only difference with sub-sampling is that a subsample must remain inside the bounding sphere that defines the underlying surface patch (even if the face it lies on extends outside of this boundary). An extra check is performed to ensure that the subsamples do not move outside this bounding sphere, and if they do, they are snapped back to the nearest position on a face within the boundary.

This process provides a quick and easy way to sample most any meshed model. We use this approach to sample a triangulated meshed object, but it easily extends to an polygonal object. Turk also provides other extensions [100] that we do not incorporate within this work (e.g., adjusting the sampling along regions of heavy curvature).

APPENDIX B

MOVING LEAST SQUARES SURFACE FITTING

Least Squares (LS) approximation is a mathematical technique commonly used for fitting a continuous function (i.e., a polynomial of degree M and spatial dimension D) to a scattered set of data points [122]. LS centers on minimizing the sum of the squared error (i.e., the residual difference between the observed and modeled values) over the set of points. Figure 65 shows a simple two-dimensional example of a LS fitting for a random set of points.

The objective of a LS fitting is to find a globally defined function $f(x)$ that approximates a given set of scalar values f_i at points x_i, with a minimal amount of error/residual. The standard approach (shown in Equation B.1) is to formulate this problem as a minimization over $\prod_M^D$, the set of polynomials of degree M and spatial dimension D.

$$argmin_{f \in \prod_M^D} \sum_i \|f(x_i) - f_i\|^2 \tag{B.1}$$

Moving Least Squares (MLS) is an extension of standard LS fitting that also allows for reconstruction of a continuous function from a set of unorganized points, but does so using a series of locally weighted fittings to define the curve [122]. For each location r where the surface should be evaluated, a polynomial is computed using a weighted least squares fitting. The influence of data points to this fitting is based on a weighting function that typically incorporates both the distance of each neighboring point to r, and the sampling density of the neighborhood around r. This weighting biases the fitting to localized regions immediately around r, and penalizes the influence of points further away. The value of this local approximation provides

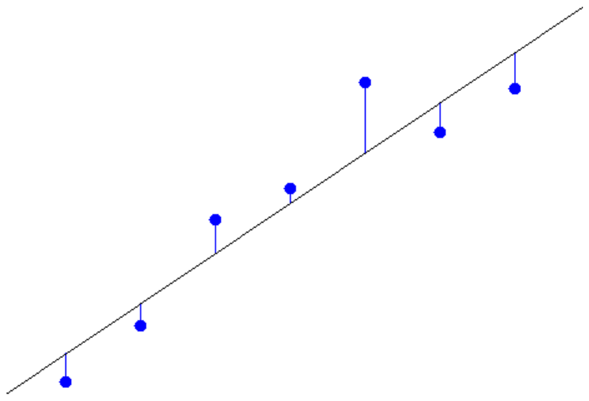


Fig. 65. A simple least squares fitting for a random set of points.

the definition of the curve at r, and the set of these locally approximated values defines the implicit surface. Equation B.2 provides the standard definition of MLS:

$$argmin_{f\in\prod_M^D}\sum_i \|f(x_i)-f_i\|^2\theta(\|x-x_i\|) \tag{B.2}$$

where $\theta(d)$ serves as a non-negative decreasing weighting function. A Gaussian function (shown in Equation B.3 where d is the distance and h is a fixed parameter describing point density) is a commonly used function for $\theta(d)$. Figure 66 shows a simple two-dimensional example of a MLS fitting. In this example the color of the samples depict their weights, and the collection of these localized fittings define the overall approximated surface.

$$\theta(d) = e^{\frac{d^2}{h^2}} \tag{B.3}$$

Moving Least Squares has become a very popular technique within the field of Computer Graphics. It has been used in areas such as surface reconstruction [23], image deformation [123], and physically-based animation [124].

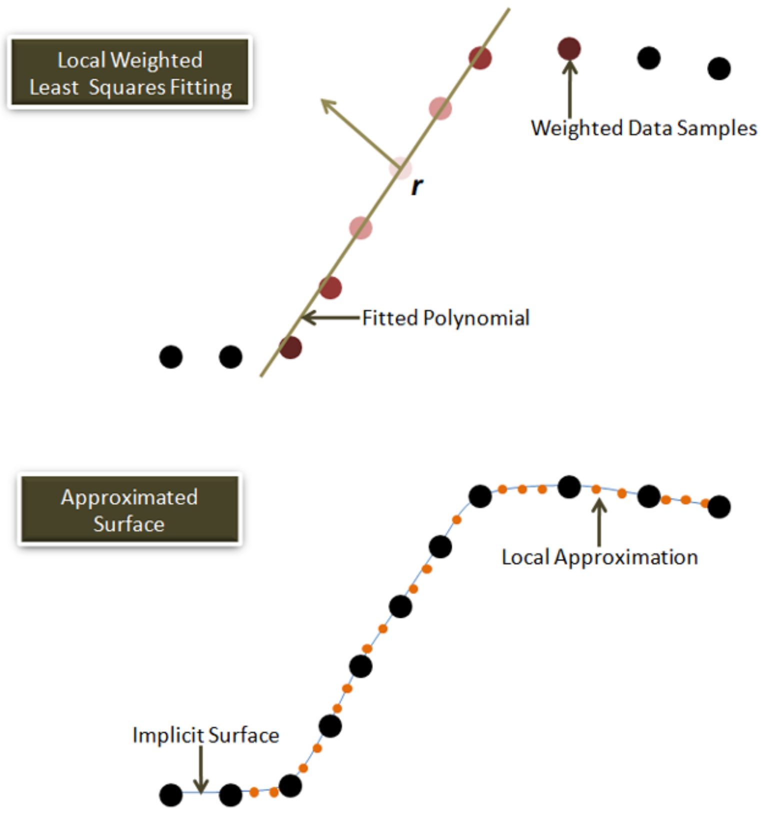


Fig. 66. A Moving Least Squares fitting for a random set of points. The top illustration shows the local fitting process, and the bottom illustration shows how the set of these local fittings form the approximated surface.

Both modeling approaches presented within this dissertation incorporate MLS as part of their construction process. Cheng et al. [102] describe two main classifications of MLS-based approaches, projection-based methods (i.e., those that employ a stationary projection in their definition) and implicit-based methods (i.e., those that employ a scalar field in their definition). As previously presented, our algorithms both use a projection-based MLS method for reconstructing a continuous surface definition from a point cloud representation. The remainder of this appendix will describe in detail the projection-based process employed by our work.

Projection-based MLS was first introduced by Levin [125][126] as a means of determining a smooth approximation of unstructured surface data. This idea was later extended by Alexa et al. and shown to be an efficient tool for reconstructing high quality surfaces and could scale to large datasets [23]. Many different alternatives to these two approaches have been proposed as well [102]. We chose the approach defined by Amenta and Kil [4] to implement within our work, but many of these alternative solutions could be used in place of our chosen method.

The MLS surface reconstruction algorithm takes as input an oriented set of points $P = \{p_1, ..., p_n\} \in \mathcal{R}^3$ assumed to sample an unknown surface. It can be assumed that these samples may contain noise and the points are irregularly located in space. The objective of projection-based MLS is to obtain a surface approximation $\mathcal{S}$ that smoothly and continuously approximates these point samples by projecting them onto the implicitly defined surface. For a given location r where the surface should be evaluated, the projection-based procedure iteratively transforms r until it lies on the surface defined by $\mathcal{S}$.

The projection procedure by Amenta and Kil gives an explicit definition of MLS in terms of the critical points of an energy function $e(x, a)$ on lines determined by a vector field $n(x)$ [4]. The energy function (defined in Equation B.4) takes a point x

and an un-oriented direction vector a as input, and measures the quality of fit (using a weighted distance measure) of a plane through x with normal a to the points in P.

$$e_{MLS}(x, a) = \sum_{p_i \in \mathcal{P}} (\langle a, p_i \rangle - \langle a, x \rangle)^2 \theta(x, p_i) \tag{B.4}$$

The un-oriented vector field (defined in Equation B.5) assigns a direction vector at x by finding the normal of the plane through x that is the best fit to that local region of the point cloud. This normal is determined based on finding the smallest Eigenvalue (and corresponding Eigenvector) from the minimization of the matrix of weighted covariances [23].

$$n_{MLS}(x) = argmin_a e_{MLS}(x, a) \tag{B.5}$$

The MLS surface $\mathcal{S}$ is then described as the set of points described using a minimization of $e(x, a)$ along a line found from $n(x)$ (defined in Equation B.6).

$$\mathcal{S}_{MLS} = \{x | x \in arglocalmin_{y \in l_{x,n(x)}} e(y, n(x))\} \tag{B.6}$$

The projection-based MLS surface is then found as follows. For each iteration, a direction vector $n(x_i)$ which defines a line $l_{x_i,n(x_i)}$ through x_i is found from the vector field. The local minimum of the energy function $e(x_i, n(x_i))$ along this line is then found. This new location is defined as x_{i+1}, and this minimization process can be performed iteratively. For each step taken, the total energy should decrease. As a result, this process will eventually converge to the point x_n which lies on the surface $\mathcal{S}$. Figure 67 provides an illustration (similar to that provided by Amenta and Kil [4]) of this iterative projection process.

Amenta and Kil follow up this formal definition of MLS with a second more detailed definition that is used for implementation. If oriented normals are provided

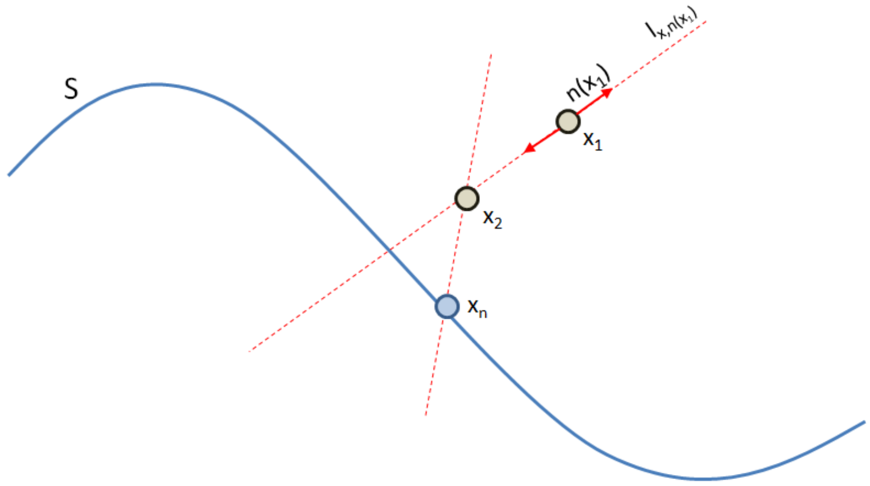


Fig. 67. An illustration of the iterative convergence process for projecting a point onto the surface. After n iterations, the point x will reside on the surface $\mathcal{S}$.

with the input P, then a vector average can be used to find $n(x)$ (defined in Equation B.7).

$$n(x) = \sum_i a_i \theta_N(x, p_i) \tag{B.7}$$

A weighted average is computed using a normalized Gaussian weighting function (defined in Equation B.8). This allows the energy to be greater than zero even when a sample is far from the surface.

$$\theta_N(x, p_i) = \frac{e^{-d^2(x,p_i)/h^2}}{\sum_j e^{-d^2(x,p_j)/h^2}} \tag{B.8}$$

A Mahalanobis distance (similar to Euclidean distance, but using an elliptical unit ball rather than spherical; defined in Equation B.9) is used to define the distance of x to a point p_i.

$$d_M(p_i, a_i, x) = \langle (x - p_i), a_i \rangle^2 + c\|(x - p_i) - \langle (x - p_i), a_i \rangle a_i\|^2 \tag{B.9}$$

The parameter c is a scaling factor that controls the effects of the distance metric. Amenta and Kil provide several examples of surfaces generated using different values for c [4]. The energy function $e(x, a)$ is defined in Equation B.10 as:

$$e(x, a) = \sum_i d_M(p_i, a_i, x)\theta_N(x, p_i) \tag{B.10}$$

Our implementation uses Equations B.7 and B.10 to define the surface $\mathcal{S}$, and follows the details provided by Amenta and Kil [4]. We use a *kd*-tree to efficiently find nearby samples within P. The neighbor size used varied by dataset, and the density of samples each dataset contained. The energy minimization was performed using an implementation of Brent's method for one-dimensional non-linear optimization provided in *Numerical Recipes in C* [127]. This procedure was used to find the minimum of $e(x, n(x))$ along the line $l_{x,n(x)}$ by identifying a bounded region (described by a lower and upper bound), and then reducing the intervals until a minimum is found (within a defined tolerance).

MLS provides a powerful and efficient tool for generating a continuous surface approximation from a scattered set of data points. Once the surface has been defined, then a contouring algorithm can be used to quickly generate a polygonal representation of the implicit surface. The approach we have described provides one alternative for reconstruction, but there are many alternative MLS-based approaches that offer a variety of improvements (e.g., sharp features, more robust fittings with noisy data, etc. [102]).

APPENDIX C

OVERVIEW OF PETRI NETS

A Petri net is a formal modeling language capable of describing concurrent and distributed systems [111]. Petri nets were first introduced in the dissertation of Carl Adam Petri [128]. These structures were presented as a model for information flow in distributed systems, where asynchronous and concurrent operations of different parts are necessary. Petri nets allow these different parts to work together through the use of a graph/network structure. They provide a natural means for modeling processes that include action, choice, iteration, parallelism, synchronization, and dependency [111].

Since Petri nets were first introduced, they have since been used in many domains as a means of modeling information flow, as generators of formal languages, and for managing resource allocation/distribution [111]. The remainder of this appendix will describe both the main concepts behind the basic Petri net structure, and the colored Petri net extension used within our research. This description follows that provided by Peterson [111].

A Petri net is represented as a directed bipartite graph composed of two types of elements.

- A set of nodes, referred to as *places*, that correspond to states/conditions within the modeled system.
- A set of bars, referred to as *transitions*, that correspond to events within the modeled system.

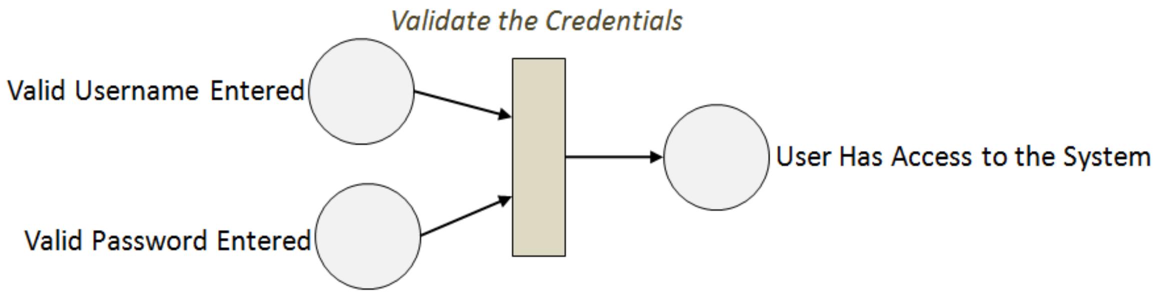


Fig. 68. A simple Petri net composed of three places and a single transition for validating user access to a system.

Places and transitions are connected together with directed arcs, defined only from places to transitions and transitions back to places, to describe relationships between the elements. Figure 68 shows a simple example of a Petri net used to model a user validation system. This example shows that once a user has entered a valid username and password combination, the transition will fire and validate the user, and grant them access to a system as a result.

A Petri net is executed by passing tokens between places based on transition *firing*, where a transition will only enable and fire if all of its incoming places contain valid tokens. When a transition fires, tokens are removed from the incoming places and new tokens added to all of the output places. This token passing process may result in further firings of other adjacent transitions in future iterations. Figure 69 shows an example of this token passing scheme. Notice that the place holding a token at the bottom left of each Petri net instance does not fire (and release its token) until both incoming places into the respective transition hold a token.

The distribution of tokens in a Petri net defines the current state of the system, and is referred to as a *marking*. The firing of a Petri net is atomic and performed as a single non-interruptible step. After each firing, this marking may change based

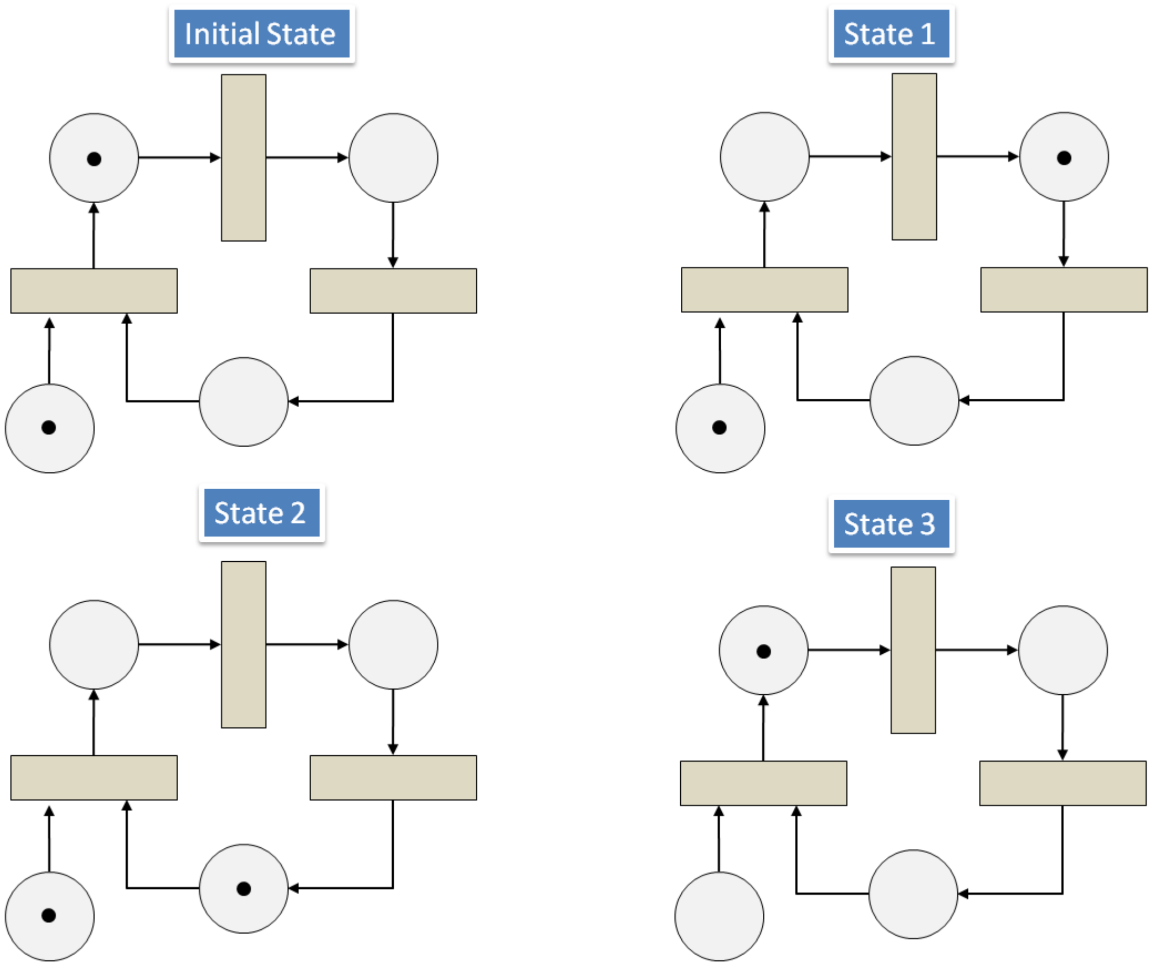


Fig. 69. An example of the iterative token passing process used within a Petri net. Places are illustrated as circles and transitions as rectangles.

on the underlying firing of transitions. Petri net firing is done in an iterative fashion. For each firing, the full set of transitions is analyzed using the current marking, and all those that are in a valid state are fired. In some cases two overlapping transitions may both be in a valid state and capable of firing, but by firing one transition it disables the other (and vice versa). These two transitions are said to be *in conflict*. Two potential fixes to address this situation are to allow for a unique prioritization of each transition that defines which transition takes precedence over another, or using a randomized selection criteria. Once the firing has executed, tokens are moved, and a new marking is generated. This overall process can then be repeated with the new state.

Petri nets allow for modeling a system composed of a series of discrete events whose order of occurrence is governed based on the defined state of the system. This allows for asynchronous execution of dependent events, and once one event has been chosen, subsequent events are decided based on established conditions in the network. Petri nets also allow for concurrent execution of sets of events. Events can occur simultaneously or sequentially depending on their definition. By using the Petri net as the guiding definition of a system, very complex processes can be performed in an overall globally consistent fashion. Petri nets have the capability of being hierarchical, where a single abstract place in one model may be replaced with another entire sub-model. Thus, Petri nets can be used for both bottom-up and top-down modeling of systems.

Petri nets have certain underlying mathematical properties that make them very interesting. Given a network and a marking, different properties can be analyzed such as reachability, liveness, and boundedness. Analyzing these properties allows for a more formal understating of the model, and identification of potential issues. Petri nets have also been used as a means for studying formal languages and automata.

In the generative modeling approach previously presented, we used a colored Petri net. Colored Petri nets are an extension to the standard approach that allows for tokens to store one or more values. These values can be as simple as a single integer, or complex structures with many fields. As the tokens are passed through the Petri net, these values can be checked and used to determine whether to fire a transition or not, as well as adjusted during the firing process.

Petri nets provide a simple data structure that allows for handling asynchronous and concurrent behavior within a distributed system. In this dissertation we have shown one application, but many others exist. There are also many other specific extensions to the Petri net structure not incorporated into our approach. Peterson provides a very good overview of the many capabilities and extensions of this powerful technique [111].

VITA

Name:	Keith Edward Biggers
Email Address:	biggers@tamu.edu
Web-site:	http://students.cs.tamu.edu/kbiggers/
Address:	Texas Center for Applied Technology,
	Texas A&M University
	3407 TAMU
	College Station, Texas 77843-3407
Education:	B.S., Computer Science, Texas Wesleyan University, 1999
	M.S., Computer Science, Texas A&M University, 2001
	Ph.D., Computer Science, Texas A&M University, 2011