Diss. ETH No. 25193

Data-driven Processing of Point-sampled Geometry

A dissertation submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by
Riccardo Roveri
MSc in Computer Science, ETH Zurich, Switzerland
Born on 09.10.1989
Citizen of Switzerland

accepted on the recommendation of
Prof. Dr. Markus Gross, examiner
Prof. Dr. Mario Botsch, co-examiner
Dr. Cengiz Öztireli, co-examiner

2018

Abstract

Point samples are an important representation for 3D geometry. Their common acquisition with the always more available scanners, as well as the generality of their meshless nature make them a natural choice for representing real world structures in Computer Graphics. Sets of points (point clouds) can, at the same time, model patterns of elements (such as the distribution of trees in a forest), or approximate the underlying surface of more continuous structures (such as a human face). A main challenge in using point clouds in practice is the accurate processing of very complex patterns and surfaces, due to the existing noise and sparseness in the data as well as to the lack of proper methods designed to handle and preserve details in geometry. In this thesis, we approach analysis and synthesis of point clouds in a data-driven fashion, in order to learn priors from existing data that can lead to processing of more detailed and complex geometry. First, we exploit the repetitions existing in many real world complex structures and patterns to propose example-based sampling methods, where the input sample data is reproduced in the output domain in a natural looking and statistical sound manner. Then, in order to process more general structures which are not necessarily repetitive, we resort to larger datasets of point clouds composed of numerous examples, and develop corresponding deep learning architectures. We apply the latter to the problems of point cloud consolidation and classification.

More in details, we start by proposing an example-based synthesis method for repetitive structures represented with point samples. The algorithm relies on expressing the input example and the synthesized output with continuous functions, and performing the synthesis through the smooth minimization of a patch-based similarity matching measure. The novel continuous formulation allows us to produce proper sampling, and to generally handle complex sets of discrete elements, continuous structures and their mixtures within the same framework.

Subsequently, we extend the synthesis method to rely on multiple input examples instead of a single one, thus handling more advanced patterns with varying arrangements of the points (correlations). We therefore introduce the concept of adaptive correlations, with a framework for analysis and synthesis of elaborated patterns based on point processes statistics.

From synthesizing new point clouds, we then focus on improving (consolidating)

a given one. We achieve proper preservation of geometric details in general structures by building large datasets composed of patches of points similar to the processed ones, and by designing a novel patch-based neural network architecture to learn accurate priors from the data and output consolidated patches. The architecture is based on a key projection component which transforms the 3D points into 2D images, allowing us to exploit the strengths of deep learning on 2D rasterized data.

Finally, some applications require point clouds to be processed in a global fashion. We thus extend our novel neural network projection component to handle more complex global point clouds, instead of simpler local patches, and apply it to the problem of point cloud classification. Our method automatically generates detailed 2D images representing the full point clouds and recognizes to which class of objects they belong to.

Sommario

Le nuvole di punti sono una rappresentazione importante per geometria in 3D. La loro frequente acquisizione con i sempre più accessibili scanner, come anche la generalità dovuta alla loro natura senza connettività li rendono una scelta logica per rappresentare strutture del mondo reale con la grafica virtuale. Insiemi di punti possono, allo stesso tempo, modellare distribuzioni di elementi (come gli alberi in una foresta), o approssimare la superficie sottostante di strutture più continue (come una faccia umana). Un'importante sfida nell'uso delle nuvole di punti è poter processare distribuzioni e superfici molto complesse, date le imperfezioni e la sparsità dei dati e la mancanza di metodi sviluppati apposta per sintetizzare e analizzare complesse strutture del genere. In questa tesi, approcciamo l'analisi e la sintesi di nuvole di punti con metodi basati sui dati, con lo scopo di sfruttare e imparare informazioni dai dati esistenti per poi processare meglio la geometria dettagliata. Innanzitutto, sfruttiamo le ripetizioni esistenti in molte strutture e distribuzioni del mondo reale per proporre metodi per la generazione di punti basati su un esempio, dove i dati di input sono riprodotti nel campo dell'output in modo naturale. Poi, per processare strutture che non sono necessariamente ripetitive, ci appoggiamo a insiemi di nuvole di punti più grandi composti da numerosi esempi, e sviluppiamo metodi di intelligenza artificiale (apprendimento profondo) basati su di essi. In particolare, presentiamo architetture di apprendimento profondo per la consolidazione e classificazione di nuvole di punti.

Più in dettaglio, iniziamo proponendo un metodo basato su un esempio per la sintesi di strutture ripetitive generali, rappresentate da punti. L'algoritmo dipende dall'espressione dell'esempio di input e dell'output sintetizzato con funzioni continue, e dall'eseguire la sintesi attraverso la minimizzazione "liscia" (smooth) di una misura di similarità di coppia. La nuova formulazione continua ci permette di produrre un numero di punti appropriato, e di gestire insiemi di elementi discreti, strutture continue così come delle combinazioni dei due, usando lo stesso metodo.

Successivamente, estendiamo il nostro metodo di sintesi per funzionare con molteplici esempi di input, al posto che un singolo esempio, in modo da gestire distribuzioni più complesse con una disposizione (correlazione) dei punti variabile. Introduciamo il concetto di correlazioni adattabili, con un metodo per l'analisi e la sintesi di distribuzioni di punti basato sulla statistica dei processi di punti (point processes).

Dal sintetizzare nuove nuvole di punti, ci concentriamo poi sul migliorare (consolidare) una data nuvola di punti. Otteniamo una propria preservazione dei dettagli geometrici costruendo dei grandi insiemi di dati composti da gruppi di punti simili a quelli processati, e sviluppando una nuova architettura di apprendimento profondo basata su piccole vicinanze di punti, per imparare informazioni accurate dai dati. L'architettura dipende un componente chiave di proiezione, che trasforma i punti 3D in immagini 2D.

Per finire, estendiamo la nostra nuova architettura di apprendimento profondo per gestire nuvole di punti globali, invece che piccole vicinanze locali. La applichiamo al problema della classificazione di nuvole di punti, generando automaticamente immagini 2D che rappresentano le nuvole di punti intere e riconoscendo a quale classe di oggetti appartengono.

Acknowledgments

First of all, I would like to express my deep gratitude to my advisor Prof. Markus Gross, who allowed me to pursue a Ph.D. at the Computer Graphics Laboratory. His expertise, vision and excitement for the topic were fundamental in motivating myself along this journey. In addition to directly guiding my research paths, he made sure to provide help through an optimal atmosphere in the laboratory and the involvement of excellent supervisors.

I will always be grateful to Dr. Cengiz Öztireli, who believed in me since my Master Thesis and guided me through my Ph.D. He has been an amazing source of inspiration in terms of learning the scientific approach to problems and handling the management of projects, while constantly showing great humanity and kindness. I will always be happy to work with him in the future.

I am also very thankful to Prof. Mario Botsch, who accepted to be a member of my examination committee.

In addition, I want to warmly thank all my other collaborators: Dr. Sebastian Martin, Dr. Barbara Solenthaler, Dr. Tobias Günther, Ioana Pandele and Lukas Rahmann. The obtained results would have not be possible without their help, and I consider myself lucky to have been able to learn something from each of them.

A big thank you goes to all my colleagues at CGL, CVG, IGL and Disney Research, who I am lucky to call friends. It has been amazing to spend these years with them in the lab, sharing curiosity, creativity and, especially, a lot of fun. A special mention goes to Vittorio, Fabio and Endri, what a ride it has been!

Last but not least, I am mostly grateful to my family and friends outside the lab who supported me during this journey. My parents and my brothers, who always inspired me with their example to become a better person, inside and outside the office. They never failed to share their enthusiasm towards my choices, which was crucial for succeeding in pursuing them. And of course you, Yanina, for always supporting me with your unlimited love, for sharing great ideas, for making my life so much fun and for your dimples.

Contents

CHAPTER 1

Introduction

A fundamental problem in Computer Graphics is to model the structures found in the real world with geometry representations. Real world elements vary greatly in shapes, characteristics and level of details. For example, the hill in Figure 1.1 (left) resembles a single smooth structure, contrary to the multiple stones elements in Figure 1.1 (center), and presents significant less details than the face in Figure 1.1 (right). Geometry representations should approximate as best as possible the shape of any real structure, while being computationally inexpensive in order to allow for fast processing.

A powerful and general model used in Computer Graphics is point-sampled geometry. The structures are approximated solely with unordered sets of points (point clouds), with no connectivity information. The generality of this meshless approach allows to efficiently model structures with different characteristics: for discrete elements, like a pile of stones, each point describes one individual element and the set represents their pattern, while for continuous structures, like a face, multiple points approximate a surface fitted to them (which can finally be obtained by performing surface reconstruction on the set of points).

Point clouds are synthesized or acquired by scanners (the Microsoft Kinect being one of the most common ones), and often present problems such as noise, sparsity, redundancy and missing parts. Motivated by their utility for many applications, and with the goal of proposing solutions to these obstacles, many researchers have worked on processing point clouds. New proposed methods and available scanners have lead to an increasing number of exist-

Figure 1.1: *Various real world structures. A smooth hill (left), multiple stones elements (center) and a more detailed face (right).*[1]

ing point-sampled geometries and more accessible point clouds generation procedures. Both these effects drove our efforts towards approaching point cloud processing in a data-driven fashion, inspired by the successful results obtained with rasterized images in example-based synthesis and processing with machine learning. In particular, example-based methods have been well researched in Computer Graphics and have proven to be powerful tools for user-controlled artistic generation, while, with the recent interest in neural networks and the creation of massive datasets, deep learning architectures have started to show promising and inspiring results in many processing operations for graphical data. In this thesis, we present our novel contributions in analyzing and synthesizing point clouds, exploiting information and statistics extrapolated from single data examples or large datasets.

Specifically, we designed data-driven methods for sampling, consolidation and classification of point clouds. Sampling means synthesizing (often after analyzing an input example) an optimal configuration of points, either to represent a surface or a pattern. It is desired to avoid sparsity and redundancy in the synthesized result. Similar to sampling, consolidating means synthesizing a new, dense point cloud in order to improve a given input point cloud. Challenges are the sparsity, noise and missing parts existing in the input. Finally, classifying a point cloud means analyzing it and determining to which class of objects it belongs. All these fundamental problems can be considered as preprocessing steps for different important applications, such as point cloud reconstruction and visualization. For example, in order to successfully reconstruct a surface from an input point cloud, the latter should be first consolidated in order to improve the quality of the data, an optimal sampling can be computed in order to avoid redundancy and thus improve

[1] From left to right: 'Silbury Hill' by Andy Wright available at `https://flic.kr/p/57SzG4`, 'Stone wall' by Keita Kuroki available at `https://flic.kr/p/gGWo1j`, 'Face' by Tim Green available at `https://flic.kr/p/7dAJqB`. All under a Creative Commons Attribution 2.0. Full terms at `http://creativecommons.org/licenses/by/2.0`.

reconstruction efficiency without altering the surface, and information about the object class could be exploited to define priors in the reconstruction. Likewise, proper sampling and consolidation allow to better visualize the point clouds, and information about the class of the represented objects could provide, for example, additional insights on the best view directions for optimal visualization.

We start by describing an example-based point sampling method to synthesize general repetitive structures (Chapter 3), motivated by the observation that repetitions are a common phenomena in nature and, in general, in the real world. Natural elements such as waves in the ocean or trees on a mountain, as well as human-made objects like a wall of stones are composed of very similar structures which repeat themselves on a larger scale. When dealing with repetitive structures, example-based approaches are a logical choice, where the user provides a small structure as input and a larger, similar looking output is produced. Most of the previous methods in the field rasterize the repetitions into regularly sampled grids (pixels or voxels) and use neighborhood matching in a texture synthesis manner. This makes it possible to produce continuous looking repetitive structures (e.g., the waves of the ocean), but not to generate patterns of discrete elements (e.g., the stones in the wall), as the individual elements are not preserved but rather merged together. Other existing methods allow to synthesize patterns of discrete elements based on an example, but, in turn, cannot handle continuous structures. Our proposed sampling method exploits the generality of point samples, introducing a unified framework for structures and patterns which consist of mixtures of discrete and continuous elements. We achieve this by converting the points belonging to the input example and the ones synthesized in the output domain into continuous functional representations, and formulating the patch-based texture synthesis problem as a smooth minimization. Our robust minimization comes with a better neighborhood matching metric and allows us to include precise sampling control, which is crucial to obtain an optimal sampling. In addition, the meshless nature of our method makes it efficient and suitable for interactive synthesis.

A limitation of the proposed example-based method is that only a single pattern can be synthesized in the output, which is the pattern represented by the input example (e.g., a blue noise pattern or a regular grid pattern). The same drawback is shared by the previous methods in point sampling, which propose techniques built on statistics to synthesize point distributions that can have varying density, but always present the same arrangement of points (namely the correlation) as in the input pattern. In the real world, on the other hand, patterns with varying correlations can be often observed, as, for example, the configuration of trees in a forest that can change from a

uniform distribution to a clustered distribution, depending on the spreading of resources. In order to being able to represent such structures, we extend our previously explained sampling method to handle multiple input examples, and interpolate them when synthesizing the output patches. More generally, we introduce a framework based on the theory of stochastic point process, which leads to analysis and synthesis methods for interpolating statistics in the space and time, and, in particular, to the notion of adaptive correlations (Chapter 4). The framework revolves on locally interpolating, in the output domain, a statistical measure, the pair correlation function, which is the joint probability of having pairs of point samples at particular locations, using a dictionary of pair correlation functions extracted from the multiple input examples. In addition to synthesizing natural distributions with varying patterns, we show results in several applications such as surface sampling, image stippling and image reconstruction.

Although our described synthesis method allows us to learn and create sampling from multiple given examples, there exist operations that deal with data which is not repetitive and is too varied to be processed only with a small set of examples. Point cloud consolidation belongs to such operations, where the goal is to generate a new point set that accurately samples the underlying surface, starting from an input point cloud with highly varying local structures, noise and missing parts. A typical example is the consolidation of a point cloud acquired by scanning a human face with a consumer device. Most of the previous methods tackling point cloud consolidation rely on local priors such as locally piece-wise smooth surfaces with sharp features, which cannot preserve very detailed geometric features. In order to handle general structures and recover elaborate features, we propose a data driven solution making use of deep learning and a large dataset of geometric patches (Chapter 5), instead of limiting us to a small dictionary of example patches like in our previously described example-based methods. We propose a generative convolutional neural network architecture that inputs and outputs patches of point clouds, by learning their local parametrization and the locally fitted surfaces of their geometric features. The most important component is a network module that projects the unordered 3D points of the patches to 2D heightmaps, which can then be easily and efficiently processed. We present obtained results of consolidations and following reconstructions of structures with vary levels of details, such as detailed sculptures and smooth flags.

Finally, point clouds can be processed as global structures as well, besides patch-by-patch. Point cloud classification is an important operation that benefits from handling point clouds globally, as the spatial relationships between parts of the structure add important information about the class the

object belongs to. While deep learning has proven to produce state of the art results in point cloud classification, most of the existing methods rasterize the point samples into regular 3D voxel grids in order to utilize common grid-based network architectures. Due to the sparse nature of the voxel grids, these methods are not memory efficient and limit the resolution of the point clouds. We propose an extension of our described projection-based convolutional neural network architecture for unordered points processing, adapted to consume full point clouds and not only patches (Chapter 6). A series of 2D depth images representing the input point cloud from optimal view points are generated within the network by the projection module, and classified with common image classification pipelines. The large amount of existing 2D image data allows us to effectively train our classifier and obtain results competitive to the state of the art. Moreover, the intrinsically generated, optimal depth images represent a useful resource for other point clouds operations, such as their visualization.

1.1 Contributions

In this thesis we propose the following main contributions:

- An example-based method for synthesizing general repeated geometries represented by point samples, treating continuous and discrete structures within the same framework. The approach allows for interactive synthesis on general domains, and includes a point sampling strategy for initialization independent, optimal sampling.
- The notion of adaptive correlations, and an analysis and synthesis framework for general point patterns with adaptive density and correlations, offering full control over the distributional characteristics. Mulitple input example distributions can be merged on Euclidean domains and surfaces.
- The first deep learning method for local point cloud processing with a fully differentiable architecture, featuring a projection layer for converting patches of unordered points to regularly sampled height maps. We show how its application to point clouds consolidation leads to more accurate surface representations compared to the previous methods.
- An extension of our point cloud processing neural network to handle point clouds representing full objects and not only patches, and a novel architecture for point cloud classification that achieves results

competitive with the state of the art. Our architecture produces a set of informative depth images of the point cloud, by predicting meaningful view directions, which can be used for further applications.

1.2 Publications

During this thesis, the following peer-reviewed publications were made:

R. ROVERI, A. C. ÖZTIRELI, S. MARTIN, B. SOLENTHALER and M. GROSS. Example Based Repetitive Structure Synthesis, *Proceedings of Eurographics Symposium on Geometry Processing (Graz, Austria, July 6-8, 2015), Computer Graphics Forum*, vol. 34, no. 5, 2015, pp. 39–52.

R. ROVERI, A. C. ÖZTIRELI and M. GROSS. General Point Sampling with Adaptive Density and Correlations, *Proceedings of Eurographics (Lyon, France, April 24-28, 2017), Computer Graphics Forum*, vol. 36, no. 2, pp. 107-117.

R. ROVERI, A. C. ÖZTIRELI, I. PANDELE and M. GROSS. PointProNets: Consolidation of Point Clouds with Convolutional Neural Networks, *Proceedings of Eurographics (Delft, The Netherlands, April 16-20, 2018), Computer Graphics Forum*.

R. ROVERI, L. RAHMANN, A. C. ÖZTIRELI and M. GROSS. A Network Architecture for Point Cloud Classification via Automatic Depth Images Generation, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Salt Lake City, USA, June 18-22, 2018)*.

CHAPTER 2

Related Work

In this chapter, we describe related works in the fields of analysis and synthesis of geometry structures. In order to provide a general view of the state of the art, we review related methods for point sampled geometry as well as for other representations like rasterized geometry and meshes. In particular, we first study general geometry synthesis methods (Section 2.1), classifying them in approaches for generating sets of discrete elements or continuous structures, mostly in an example-based fashion. We then give a deeper insight on point distributions, where the represented structure is the pattern of the points itself (Section 2.2). More precisely, we review algorithms for analyzing and synthesizing point patterns, introducing works based on stochastic point processes. Successively, we provide an overview of deep learning architectures that have been designed for processing 3D point clouds and other sparse geometry representations (Section 2.3), exploiting large datasets. The unordered nature of point clouds made it non-trivial to adapt image-based network architectures to the case. Finally, we describe related works for the applications of point cloud consolidation and point cloud classification (Sections 2.4 and 2.5, respectively).

2.1 Geometry Synthesis

Geometry synthesis algorithms have been designed to generate continuous structures (by extending 2D texture synthesis to 3D volumes or by working with meshes and models), or to synthesize sets of discrete elements (by

matching neighborhoods of individual elements or their aggregated statistics). Contrary to the previous approaches, in Chapter 3 we present an example-based method to synthesize repetitive structures represented by point samples, which can synthesize continuous structures, discrete elements and their mixtures.

Raster-based Texture Synthesis A classical approach to synthesizing repetitive structures based on given examples is to rasterize the example and the output, usually referred to as texture synthesis. The structures are thus represented as continuous values stored at regularly distributed sample points. For synthesis, the stored values in the output image are altered while keeping the point locations fixed. A texture image of arbitrary size can be synthesized from a given example image that contains a patch of the repetition using this technique [Paget and Longsta, 1995; Efros and Leung, 1999]. Most texture synthesis methods work by matching local neighborhoods such that for each neighborhood in the output, there is a similar neighborhood in the example [Wei et al., 2009]. This idea has also been extended to colors on meshes and 3D volumes [Turk, 2001; Kopf et al., 2007; Wei et al., 2009]. However, if the pattern consists of discrete elements, the integrity of individual elements can be lost with this method since they are rasterized [Ma et al., 2011]. Furthermore, the points are assumed to stay fixed on a regular structure, and most commonly on a 2D image. These limit the application of raster-based texture synthesis to general structures.

Geometric Texture Synthesis Instead of rasterizing discrete structures, they can be directly represented with discrete elements. The goal is then to synthesize a distribution of these elements that resembles a given example [Barla et al., 2006; Ijiri et al., 2008; Hurtut et al., 2009; Alves dos Passos et al., 2010; Ma et al., 2011; Ma et al., 2013; AlMeraj et al., 2013; Landes et al., 2013; Du et al., 2013; Huang et al., 2014]. The methods mainly differ in the representation of the structures and the definition and matching of element neighborhoods. For simpler example patterns, it is sufficient to match aggregated neighborhood statistics for all elements [Öztireli and Gross, 2012; Zhou et al., 2012; Heck et al., 2013], on which we focus in Section 2.2. However, for more complex examples, individual neighborhoods of elements need to be matched [Ma et al., 2011; AlMeraj et al., 2013], unless more elaborate distance metrics based on element shapes are utilized [Landes et al., 2013].

Most of these methods are designed for particular applications and hence operate under certain assumptions on the space, shapes, and arrangements of the elements. A notable exception is the work by Ma et al. [2011] that can synthesize element distributions in 2D, 3D, or on surfaces with variable element shapes arranged in arbitrary configurations, thanks to their

point-based representation of element shapes. Our general synthesis method also works with point samples for arbitrary domains and shapes of discrete elements. However, unlike geometric texture synthesis methods, it can synthesize continuous, as well as mixtures of discrete and continuous structures interactively. This is due to a new formulation of the neighborhood matching problem in a meshless framework that allows careful sampling control and better convergence behavior.

Most general geometric texture synthesis methods are originally designed for off-line texturing of defined regions. Recent efforts utilize and extend texture synthesis methods for generating repeated patterns in paintings [Kazi et al., 2012a; Lu et al., 2013; Lukáč et al., 2013; Kazi et al., 2014; Lu et al., 2014; Xing et al., 2014]. However, these techniques are designed for particular interaction scenarios and output. In contrast, our progressive synthesis method can be used to generate general structures interactively.

Continuous Geometry Synthesis Since the previously mentioned methods cannot be used to synthesize continuous structures when the domain is not a 2D or 3D volume, several other methods have been developed to synthesize continuous geometry for terrains [Zhou et al., 2007], mesh-based geometry [Zhou et al., 2006; Lai et al., 2005] or 3D models [Merrell and Manocha, 2008]. In addition, structured 2D pattern synthesis along curves has been recently proposed in [Lu et al., 2014] and [Zhou et al., 2013], and later made suitable for fabrication in [Zhou et al., 2014]. The continuity assumption comes at the expense of the possibility of losing shapes of individual elements, similar to raster-based texture synthesis methods. Furthermore, the structures should have application specific representations for these methods to work. Our method can handle continuous synthesis in general domains in combination with discrete elements, and for all structure representations once they are converted into point samples.

2.2 Point Patterns

In Chapter 4, we introduce a framework for analysis and synthesis of general point distributions with adaptive density and correlations, developed with the theory of stochastic point processes. While several previous methods can handle point patterns with adaptive density, they all assume a constant correlation.

Analysis of point patterns. Determining characteristics of point patterns is essential for many applications in computer graphics such as stippling and halftoning [Schmaltz et al., 2010; Fattal, 2011], anti-aliasing [Mitchell, 1987;

Lagae and Dutré, 2008; Heck et al., 2013], object placement [Wei, 2010], integration [Pharr and Humphreys, 2010; Subr and Kautz, 2013; Pilleboue et al., 2015], or geometry sampling [Öztireli et al., 2010]. A widely used analysis tool is the power spectrum, a 2D diagram computed by averaging the periodograms of point distributions that are instances of a certain point pattern. When the point pattern is translation invariant, many important characteristics such as anti-aliasing properties or anisotropy of the generated distributions can be inferred from the power spectrum [Lagae and Dutré, 2008; Ulichney, 1988]. Other analysis methods rely on spatial measures such as minimum distance between points [Lagae and Dutré, 2008], discrepancy [Shirley, 1991], distributions of difference vectors [Wei and Wang, 2011], or distances [Öztireli and Gross, 2012; Heck et al., 2013] between sample point locations. These methods can also be extended to non-Euclidean domains or point patterns with adaptive density or anisotropy [Bowers et al., 2010; Li et al., 2010; Wei and Wang, 2011]. For all cases, the underlying correlation model is assumed to be constant and translation invariant, and the adaptivity in density or space the points reside on is buried into the difference or distance measures used to compute the statistics. In contrast, we present a general analysis method that can handle point patterns with adaptive density and correlations. We prove that the proposed measures converge to provably discriminative statistics from stochastic point processes.

Synthesis of point patterns. Synthesis methods can generate point distributions with certain characteristics controlled by the construction of the synthesis algorithm or via explicitly provided statistics. Most techniques in computer graphics focus on blue noise distributions, where there is a minimum distance between pairs of points, and they are distributed randomly otherwise [Ulichney, 1988; Heck et al., 2013]. Variations can be generated by altering the distances between points and introducing randomness via adding, removing, or moving the sampling points [Lloyd, 1982; McCool and Fiume, 1992; Balzer et al., 2009; Fattal, 2011; de Goes et al., 2012; Jiang et al., 2015], or tiling [Ostromoukhov, 2007] methods. Such distributions are very important for their anti-aliasing properties [Ulichney, 1988; Heck et al., 2013], and can be combined with adaptive density methods [Li et al., 2010; de Goes et al., 2012; Chen et al., 2013], or generated on surfaces [Jiang et al., 2015] for further applications. However, they cannot be utilized to model more complex patterns where this correlation model does not hold. To synthesize more general patterns with controlled characteristics, recent techniques rely on matching the statistics of output distributions with given target statistics [Zhou et al., 2012; Öztireli and Gross, 2012; Wachtel et al., 2014; Ahmed et al., 2015]. These methods can also handle adaptive density, but are not trivial to extend to non-Euclidean domains such as surfaces [Jiang

et al., 2015]. Paralleling analysis methods, all synthesis algorithms so far assume a given constant pair-wise correlation model, and locally alter the density or anisotropy of points distributions. A notable exception is the work by Ju et al. [2010], which, however, only models group motions and crowd behaviour based on a qualitative analysis. We present the first technique that can synthesize general point distributions with adaptive pair-wise correlations and density. It has been observed in rendering [Durand, 2011; Subr and Kautz, 2013; Subr et al., 2014] that adapting both simultaneously can reduce the error in numerical integration. Utilizing our synthesis method, we show that such adaptivity can also significantly improve image and geometry sampling. Our algorithm offers full control over the spectrum of points on surfaces, in contrast to the previous surface sampling methods.

Stochastic point processes. The discipline of stochastic point processes [Møller and Waagepetersen, 2004; Illian et al., 2008] provides a principled mathematical treatment of general point patterns by characterizing generating processes that underlie point distributions. Hence, each distribution is considered a realization of a stochastic point process. A point process can be defined by setting a random variable at each point in space, and analyzing the correlations among these random variables. Equivalently, we can consider correlations among point locations over different realizations of a point process. Intuitively, first order correlations describe density, and second order pair-wise correlations determine the arrangement of points. Recent works in computer graphics explore utilizing statistics from point processes to analyze and synthesize point distributions [Wei and Wang, 2011; Öztireli and Gross, 2012; Heck et al., 2013]. The main assumption of these works is that the underlying point process is stationary, i.e. the generated distributions are translation invariant up to density differences. Adaptive density can then be obtained by altering the distance or difference metric utilized. However, many important distributions from classical jittering patterns [Mitchell, 1996] to complex distributions found in nature [Illian et al., 2008] cannot be modelled with these assumptions. In Chapter 4, we abandon the assumption of an underlying stationary correlation model, propose a comprehensive analysis framework for understanding a more general set of point processes, and develop the associated synthesis algorithms.

2.3 Geometry in Deep Learning

In Chapter 5, we introduce a novel neural network architecture for processing unordered pointsets, based on a differentiable projection component that creates 2D heightmaps from the 3D points within the network. We utilize the

network by inputting patches of points, and show results in the application of point cloud consolidation. In Chapter 6, we extend the architecture to produce proper depth images of full point clouds and classify them. In this section, we present the existing architectures designed for processing geometry and especially unordered point clounds in deep learning. We also list the related works that transform the 3D data to 2D images in network architectures. For a review of related deep learning methods specific to the applications of consolidation and classification, we refer to Section 2.4 and Section 2.5, respectively.

Architectures for Point Clouds and Sparse Representations for Geometry There are several ideas in the deep learning literature to handle 3D geometries efficiently via exploiting the sparsity of the data, for example by applying convolutional neural networks to images depicting multiple views of a 3D object [Su et al., 2015; Qi et al., 2016b], extracting features in a pre-processing step [Fang et al., 2015; Guo et al., 2015; Dibra et al., 2017], or representing shapes in a spectral domain [Bruna et al., 2013; Masci et al., 2015]. The most common architectures for geometry, though, require the input geometry to be transformed to a rasterized grid. Accordingly, even for point clouds the most straightforward approach is to convert the points to a uniform voxel grid and use CNN based methods for volumetric representations, as it is done in the methods presented in [Qi et al., 2016b; Wu et al., 2015b; Maturana and Scherer, 2015; Brock et al., 2016; Wu et al., 2016]. While transforming the point cloud to a voxel grid allows to feed regularly structured data to the network, the main disadvantage of these techniques is that they are computationally expensive, limiting the resolution of the point cloud. Some attempts have been proposed in order to overcome the voxel grid resolution issue and handle 3D geometries efficiently, for example by using an octree structure [Riegler et al., 2017; Wang et al., 2017], employing field probing filters [Li et al., 2016], or exploiting the sparseness of the problem via voting schemes [Wang and Posner, 2015] or with sparse convolutions [Graham, 2014; Graham, 2015; Engelcke et al., 2017]. However, the low resolution nature of voxel grids still constraints the size of the processed point cloud.

Instead of converting the point cloud to a voxel grid, some more recent works have presented methods to directly process unordered pointsets, usually showing results in descriptive tasks such as classification and segmentation. The authors of PointNet [Qi et al., 2016a] propose a network architecture to respect properties such as invariance to permutations and tranformations of the input points. In the recently published work [Simonovsky and Ko-

modakis, 2017], CNNs are generalized from grids to general graphs using edge-dependant filters. Similarly, two concurrent works to our publications in the field propose special architectures for unordered pointsets: in Kd-Networks [Klokov and Lempitsky, 2017] kd-trees are used as underlying graphs to simulate CNNs, and PointNet++ [Qi et al., 2017b] improves the original PointNet by applying the network recursively on a nested partitioning of the input point set. Like these last set of approaches, our neural network architectures take an unordered point cloud as input, which can also be of high resolution.

Rendering Depth Images In Neural Networks Rendering 2D images from 3D geometry within a neural network is an interesting task that could have impact in many computer vision applications. Spatial Transformer Networks [Jaderberg et al., 2015] presents a differentiable module for applying transformations to a feature map. By applying a 3D affine matrix and flattening the result, their method can produce a 2D projection of the 3D voxel grid input. In [Qi et al., 2016b], the authors also propose a differentiable module based on anisotropic kernels to generate 2D images using voxel grids as input. In OpenDR [Loper and Black, 2014], a differentiable renderer for triangle-based geometry is presented. While the simple 3D to 2D projection of Spatial Transformer Networks [Jaderberg et al., 2015] does not deal with rendering, [Qi et al., 2016b] requires a convertion from point clouds to a low resolution representation and OpenDR [Loper and Black, 2014] works on triangles, in our architectures we aim at generating heigthmaps and depth images from unordered point clouds.

2.4 Geometry Consolidation

Consolidation typically involves denoising, resampling, and surface normal estimation, as well as outlier removal and missing data completion. This is then followed by surface reconstruction to get the final surface. Many reconstruction methods can also be used for resampling, and methods that output dense point sets render the reconstruction problem trivial. Hence, our consolidation technique described in Chapter 5 is related to both classes of methods. We review the most relevant techniques below.

Consolidation with Smoothness Priors Consolidation and the subsequent task of reconstruction are ill-posed problems and hence further assumptions are required to generate reconstructed surfaces. A very versatile assump-

tion is local smoothness [Alexa et al., 2003]. Smooth reconstructions or resampled point sets can be obtained with radial basis functions [Carr et al., 2001], solving a Poisson equation in 3D [Kazhdan et al., 2006], parametrization-free projections [Lipman et al., 2007; Huang et al., 2009; Preiner et al., 2014], or moving least squares based local approximations [Alexa et al., 2003; Shen et al., 2004; Guennebaud and Gross, 2007]. The smoothness assumption breaks, however, for certain classes of real-world surfaces that contain sharp features. Many other methods thus focus on preserving such features by utilizing sparsity inducing norms [Avron et al., 2010; Sun et al., 2015], dictionary learning [Xiong et al., 2014], positional constraints [Kazhdan and Hoppe, 2013], dedicated sampling of edges [Huang et al., 2013], or robust statistics [Öztireli et al., 2009; Öztireli et al., 2010], leading to significant improvements especially for man-made objects. All these techniques rely on an input point set only, and hence cannot resolve surface shapes if the input point cloud contains a prohibitive amount of imperfection that makes inferring the underlying surface infeasible. In our technique, we solve this problem by guiding local fits with priors extracted from existing point cloud data of geometries with similar local structures. This learning based approach resolves ambiguities and steers the reconstructions towards accurate local structures.

Data-driven Geometry Completion and Reconstruction When large portions of geometry are missing, several methods use data-driven priors to complete and reconstruct surfaces from point clouds. This can be achieved by retrieving models [Shao et al., 2012; Kim et al., 2013; Li et al., 2015] or model parts [Gal et al., 2007; Shen et al., 2012; Sung et al., 2015] from a database that can also be deformed to match the input point clouds [Pauly et al., 2005; Nan et al., 2012; Kim et al., 2012]. Although such methods excel at global shape completion, resolving geometric features and details can be challenging due to the limited range of compatible objects or parts in the database, wrong matches, and misalignments [Han et al., 2017]. In contrast, in our consolidation technique we do not require close matches or alignments between training and test geometries, and focus on learning to recover geometric details. Other data-driven methods regress to parameters of a constructed model, which is typically used for e.g. human body shapes [Anguelov et al., 2005; Weise et al., 2011]. However, such parametric models lack the geometric details we target, and are only designed for when the test geometries belong to the specific parametric model constructed.

Geometry Generation and Completion with Deep Learning We propose a new neural network based deep architecture for the consolidation problem. The exceptional performance of deep neural networks on image denoising and inpainting tasks has led to various previous efforts on extending their power to 3D surfaces. Several approaches extend the 2D grids used for image processing to 3D voxels grids for geometry generation [Wu et al., 2015b; Sharma et al., 2016; Varley et al., 2016]. This allows a direct extension of many successful architectures to 3D. However, as covered in Section 2.3, these only work for relatively low resolution of grids (typically up to 32^3) due to the increased memory and computational requirements in 3D. Even with the state-of-the-art approaches that fuse global and local patches [Han et al., 2017], or utilize octrees [Riegler et al., 2017], the resolution is limited to 256^3. It has thus been so far not possible to directly recover geometric details with the current deep learning architectures [Dai et al., 2017]. We specifically target such geometric features and details and propose a new dedicated deep architecture for the consolidation problem.

While we do not aim at recovering large missing parts of point clouds, the concurrent work to our publication by Han et al. [Han et al., 2017] targets completion of 3D shapes. In addition to a global structure inference network, their deep learning architecture includes a patch-based local geometry refinement network. The latter is built with voxel grids and 3D CNN-s, while we propose a network component to project unordered points to 2D heightmaps. This makes our method memory efficient and suitable to preserve fine details. Even if the general application is different, we leave the comparison with the local geometry refinement network of [Han et al., 2017] for future work.

2.5 Geometry Classification

We review geometry classification works related to our point cloud classification method described in Chapter 6. In our technique, we extract and classify a set of informative depth images representing the point cloud. Therefore, we briefly introduce approaches which exploit multiple views of 3D data, as well as CNN architectures for 2D image classification.

Classification of Point Clouds with Deep Learning Most of the deep learning methods designed for unordered point clouds reviewed in Section 2.3 tackle the classification problem [Qi et al., 2016a; Simonovsky and Komodakis, 2017; Klokov and Lempitsky, 2017; Qi et al., 2017b], achieving state of the art results comparable to methods which classify volumetric

objects [Maturana and Scherer, 2015; Wu et al., 2015b; Qi et al., 2016b] and meshes [Kazhdan et al., 2003]. Like those set of approaches, our method takes an unordered point cloud as input, and, contrary to them, instead of tackling classification directly on the point cloud, we first extract a set of 2D depth images, and then exploit well studied CNN based image classification methods to classify point clouds.

Exploiting Multiple Views on 3D Data Many deep learning methods utilize multiple 2D views of 3D data in order to learn more complex features. For example, in [Dibra et al., 2017] and [Dibra et al., 2016] the authors show how adding additional views of the human body produces better results in estimating their shape. In MVCNN [Su et al., 2015], a 3D shape model is rendered with different virtual cameras from fixed view points, and the resulting images are combined with a view pooling operation and classified with a CNN based architecture. In the recent [Kalogerakis et al., 2017], views of 3D meshes are rendered from selected viewpoints in an initial step, and fed to a network architecture which segments the meshes using projective CNNs to project images onto the shape surface representation. These approaches require a preprocessing step where the input meshes are rendered from a set of views, using standard mesh rendering pipelines. Contrary to these works, we introduce a differentiable module for rendering point cloud data from different views on-the-fly from the input, which allows the network to automatically learn the most useful view directions.

Image recognition using CNNs Our method is related to image based CNN architectures, as we classify point clouds by first automatically extracting 2D images. CNNs have produced state of the art results in image recognition and related tasks e.g. [Cimpoi et al., 2014; Donahue et al., 2014; Girshick et al., 2014; He et al., 2015]. In particular, large image datasets available [Deng et al., 2009] allow CNNs to learn features that are general and suitable for different operations. For 3D data, such large datasets are not available and harder to obtain, which lies behind our idea of extracting 2D features from 3D data. In our proposed technique, we classify our extracted 2D views with ResNet [He et al., 2015], and utilize ImageNet [Deng et al., 2009] as a dataset for pre-training.

CHAPTER 3

Example Based Repetitive Structure Synthesis

In this chapter, we present an example based geometry synthesis approach for generating general repetitive structures. Our model is based on a meshless representation, unifying and extending previous synthesis methods. Structures in the example and output are converted into a functional representation, where the functions are defined by point locations and attributes. We then formulate synthesis as a minimization problem where patches from the output function are matched to those of the example. As compared to existing repetitive structure synthesis methods, the new algorithm offers several advantages. It handles general discrete and continuous structures, and their mixtures in the same framework, as shown in Figure 3.1. The smooth formulation leads to employing robust optimization procedures in the algorithm. Equipped with an accurate patch similarity measure and dedicated sampling control, the algorithm preserves local structures accurately, regardless of the initial distribution of output points. It can also progressively synthesize output structures in given subspaces, allowing users to interactively control and guide the synthesis in real-time. We present various results for continuous/discrete structures and their mixtures, residing on curves, submanifolds, volumes, and general subspaces, some of which are generated interactively.

Part of this chapter is based on the work presented in [Roveri, 2014].

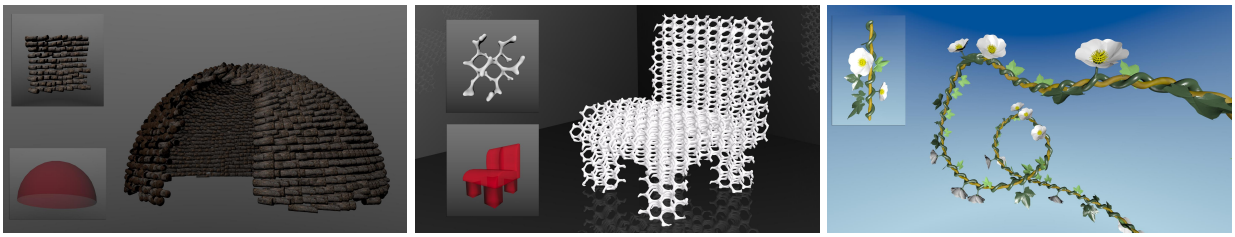

Figure 3.1: *Our example-based structure synthesis method can be used to generate structures with discrete elements (left), continuous geometries (middle), and their mixtures (right); on different domains such as surfaces, bounding volumes, or curves.*

3.1 Introduction

Repetition is an integral part of nature. Modeling repetitive structures is thus essential but also challenging. A common approach is controlling the large scale structure of an object by direct modeling, and letting an algorithm automatically add the details based on an example from the repetitive structures [Ma et al., 2011]. This has led to many algorithms tailored to particular applications with certain assumptions on the structures to be synthesized. Each of these algorithms thus come with application dependent constraints, which has been hindering content creation with general repetitive structures.

A classical approach for synthesizing repetitions is rasterizing them into regularly sampled images and using neighborhood matching based texture synthesis methods [Wei et al., 2009] to compute colors for each output pixel/voxel. This idea has also been extended to geometry synthesis for certain geometry representations [Zhou et al., 2006]. However, this raster based representation can only model a limited set of structures. Indeed, many repetitions in nature consist of individual elements, which should be kept intact. This has led to using geometric texture synthesis methods with discrete element textures, where individual elements and their interactions are utilized to describe the repetitive structure [Ma et al., 2011; Landes et al., 2013]. The discrete elements also allow resolution independent synthesis with object instancing.

However, there remain important challenges for discrete element based texture synthesis. 1) Preservation of element shapes comes at the expense of losing the ability of synthesizing continuous structures. With the current techniques, it is not possible to handle mixtures of continuous and discrete structures in general domains. 2) Representing textures with points makes the distinction between structure and sampling ambiguous, un-

like raster based textures. This translates into critical dependence on initial distribution of point samples, and non-trivial neighborhood definition and matching methods, which can result in unsatisfactory synthesis especially when continuous structures are desired. 3) Content creation with repetitive structures remains to be a challenge since the current methods are designed for off-line texture synthesis and do not support interactive artistic control for general domains [Ma et al., 2011; Landes et al., 2013; Xing et al., 2014].

We address these challenges by proposing a new method for progressive synthesis of general repetitive structures, unifying and extending previous texture synthesis techniques. The structures can consist of mixtures of discrete and continuous elements with arbitrary distributions and attributes, and reside in general domains including curved submanifolds. This allows us to extend the space of synthesizable structures, and robustly handle many structures in a unified framework. The synthesis can be intuitively and interactively guided by orientation and scaling fields defined along curves, surfaces, or volumes in 2D or 3D. The output is then synthesized on the fly automatically, allowing interactive texture brushing of general repetitive structures.

This is made possible by adopting a meshless, point-based representation and optimization framework, inspired by similar general approaches in geometry reconstruction [Alexa et al., 2001] and physically-based simulation [Martin et al., 2010]. The structures in the example and the output are first converted into a functional representation. The functions are represented by point samples with attributes extracted from the structures. The texture synthesis problem is then formulated as a smooth minimization that matches patches from the output function to those of the example function.

The generality of the meshless method allows us to seamlessly handle general structures. The functional representation results in a better neighborhood matching metric that correlates well with the visual quality of the synthesized structures. The smoothness of the optimization problem leads to robust minimization procedures with precise sampling control, which are essential to avoid bad configurations leading to incomplete or distorted structures. We show a variety of examples that range from classical texture synthesis to mixtures of continuous and discrete elements synthesized on user controlled curved domains, to illustrate the utility of our method in practice.

3.2 Overview

Our method is based on geometries represented by point samples with associated attributes (Section 3.3.1). In order to robustly compare the input exemplar and the synthesized structure, point data is converted into a functional representation encoding both the spatial configuration and attributes in the form of a sum of Gaussians (Section 3.3.3). A similarity measure between two smooth functions is then constructed (Sections 3.3.2 and 3.3.3). As in previous neighborhood-based texture synthesis approaches, our optimization alternates a matching step for local neighborhoods, and a merging step where point locations and attributes are updated according to the matching (Section 3.4.1). Our smooth formulation allows the computation of analytic gradients, thus robust methods such as the gradient descent can be applied. A multi-scale approach is used to optimize first large scale structures and then fine details, and a dynamic sampling control strategy based on the presented similarity measure guarantees the generation of a proper number of output samples (Section 3.4.1). Several large-scale control possibilities are presented in Section 3.5.1.

Our geometry representation based on points and attributes allows to model different kinds of structures in the same framework. Continuous structures (Figure 3.1 center, Figure 3.13) can be generated by storing a scalar or vector as attribute at each point location (for example, the surface normal vector). Discrete elements can be synthesized by either representing them with multiple points or with single samples (Figure 3.1 left). With the same optimization procedure, point data representing mixtures of discrete elements and continuous structures can be processed (Figure 3.1 right, Figure 3.18).

3.3 Measuring Structure Similarity

In this section we first present a flexible and powerful geometric representation that allows us to define a robust similarity measure between the generated and the example geometries that serves as the basis for synthesis. We strive to design a measure that accurately describes general structures and their similarities, accepts an adaptive and efficient discrete representation, and is smooth for utilizing robust and efficient optimization procedures.

3.3.1 Geometry Representation

In order to derive a general method for synthesizing new geometry from a set of examples, we need a general way of representing arbitrary geometries. We allow that they can have different dimensionality (1D/2D/3D) with possibly non-manifold connections. One obvious option would be to choose simplicial complexes (line segments, triangles, tetrahedra) as the underlying discrete representation [Landes et al., 2013; Zhu et al., 2014]. However as we do not want to deal with explicit connectivity between vertices, we choose to follow the general point-based representation of Ma et al.[2011] and treat material connectivity implicitly. That is, we represent all geometries by a set of tuples

$$\{(\mathbf{x}_i, \mathbf{a}_i), i = 1..n\} \tag{3.1}$$

where $\mathbf{x}_i \in \mathbb{R}^d$ are the point locations and $\mathbf{a}_i \in \mathbb{R}^{d'}$ is a vector of associated continuous attributes and encode additional geometric or appearance information. Choosing such a general representation allows us to cover a large variety of different applications as we will see later.

However, unlike Ma et al. [2011], we do not use the point samples themselves as the representation for the neighborhoods. Instead, we construct auxiliary smooth functions defined in terms of these samples, and compute a matching measure based on these functions, as we explain next.

3.3.2 Continuous Similarity Measure

In order to construct a robust similarity measure for our point-based geometry representation, we first study the problem of measuring similarity between two continuous functions and will then show how to perform a meshless discretization in order to derive the actual numerical scheme.

Similarity Error Density Let us first consider the problem of measuring *local* similarity between an output function $\mathbf{f}(\mathbf{x})$ and the example function

$\mathbf{e}(\mathbf{x})$. For this purpose, we define a window function $w(.)$ to delimit a local neighborhood (Figure 3.2, a) and a discontinuous mapping function $\mathbf{m}(\mathbf{x})$ that matches the output domain point $\mathbf{x}$ to a matching point within the example function. Using these definitions, we can define the *similarity error density* for the location $\mathbf{x}$ and a current matching $\mathbf{m}(\mathbf{x})$ as

$$S(\mathbf{f}(\mathbf{x}), \mathbf{e}(\mathbf{m}(\mathbf{x}))) = \int_{\mathbb{R}^n} |\mathbf{f}(\mathbf{x}+\mathbf{s}) - \mathbf{e}(\mathbf{m}(\mathbf{x})+\mathbf{s})|^2 \, w(\mathbf{s}) \mathbf{ds}. \tag{3.2}$$

The size and shape of the window function w characterize the actual matching. Large support sizes demand for large scale structures to match well, while smaller sizes only require small scale details to match.

Similarity Error The *total similarity error* is then given as

$$T = \int_{\Omega} S(\mathbf{f}(\mathbf{x}), \mathbf{e}(\mathbf{m}(\mathbf{x}))) \mathbf{dx}, \tag{3.3}$$

where Ω is a subspace in the output. The idea is now to minimize this total error alternatively for $\mathbf{m}(\mathbf{x})$ by finding the best matching neighborhoods for the current solution $\mathbf{f}(\mathbf{x})$, and the output function $\mathbf{f}(\mathbf{x})$ by finding the most similar output to the example, given the matching. Hence, we follow the common idea of neighborhood matching based texture synthesis methods, but reformulate it in a general way to handle different types of structures in the same framework. Once the structures in the example and output are converted into their smooth functional representations, the measure T gives a robust two-way matching.

3.3.3 Discrete Similarity Measure

Similarity Error Density In order to turn this continuous minimization problem into a numerically treatable form, we need to discretize the functions as well as the integrals involved. We achieve this by a general geometry representation, and a combination of analytic and numerical integration with meshless methods. Both output and example geometries, $\{(\mathbf{x}_i, \mathbf{a}_i), i = 1..n\}$ and $\{(\mathbf{e}_i, \mathbf{b}_i), i = 1..m\}$ respectively, are transformed into corresponding continuous functions as

$$\begin{aligned} \mathbf{f}(\mathbf{x}) &= \sum_i \mathbf{a}_i g(\mathbf{x} - \mathbf{x}_i, \sigma) \\ \mathbf{e}(\mathbf{x}) &= \sum_i \mathbf{b}_i g(\mathbf{x} - \mathbf{e}_i, \sigma), \end{aligned} \tag{3.4}$$

that is, we place Gaussians $g(\mathbf{x}, \sigma) = e^{-|\mathbf{x}|^2/\sigma^2}$ at all point locations that 'smear' the point attributes into their neighborhood (Figure 3.2, b). Switching

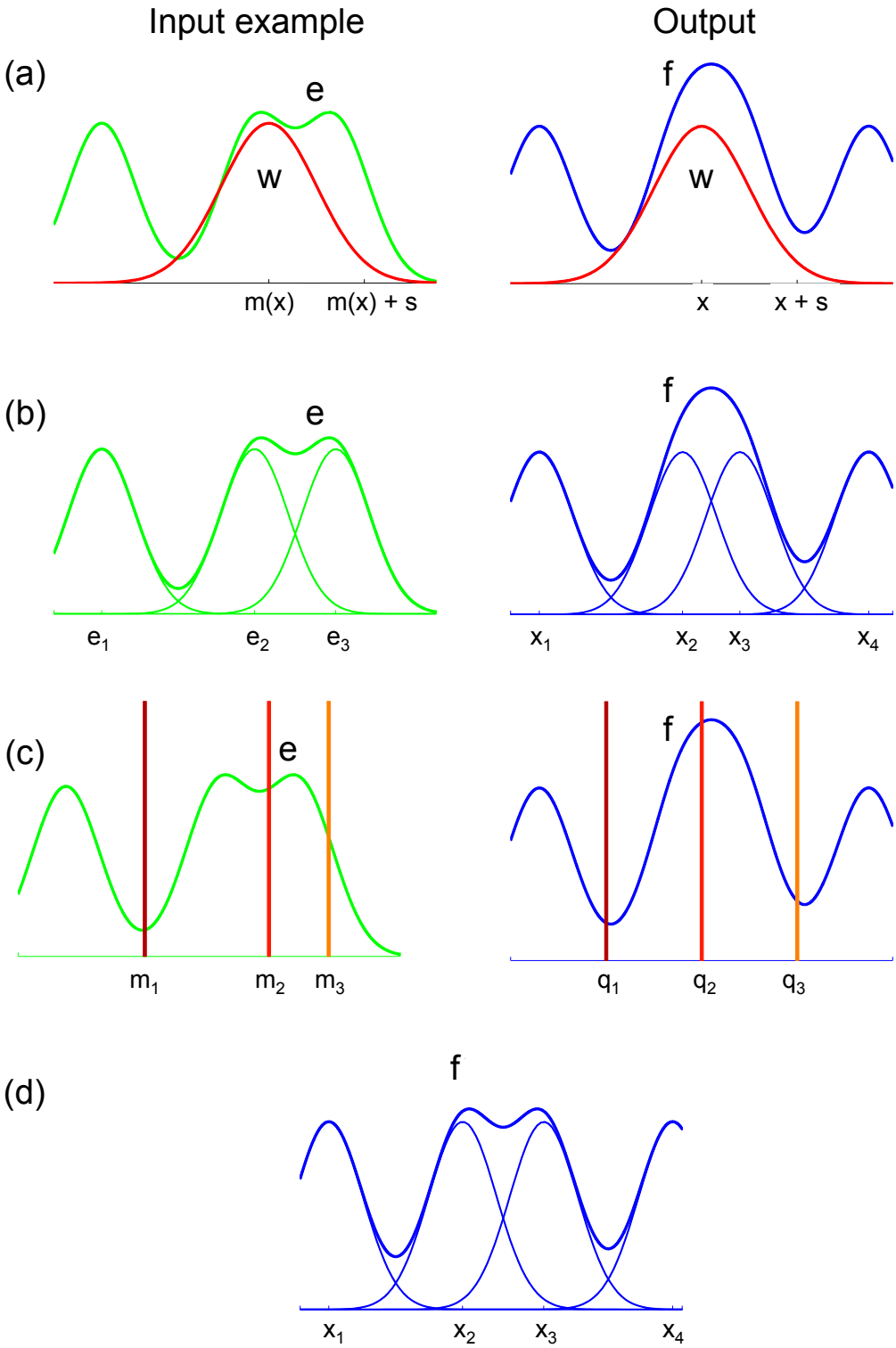

Figure 3.2: *(a) The example and output functions, (b) their discrete representation as a sum of Gaussians, (c) the quadrature points* $\mathbf{q}_k$ *and their matching points* $\mathbf{m}_k$*, (d) the final matched output function, which now has the same shape as the example function* **e** *in the region defined by the window function* **w**.

to a sum-of-Gaussians representation for the point data allows us to encode both their spatial configuration and attributes into the shape of continuous functions and to use the presented similarity measures for continuous functions. As shown in Appendix A.1, the similarity error density (3.2) can be analytically evaluated by representing the window function $\mathbf{w}$ with a Gaussian or a box function. If the latter is chosen, the evaluation results in the *discrete error density measure*

$$\begin{aligned} S(\mathbf{f}(\mathbf{x}), \mathbf{e}(\mathbf{m}(\mathbf{x}))) = \\ \textstyle\sum_{ij}(\mathbf{a}_i \cdot \mathbf{a}_j) g(\mathbf{x}_i - \mathbf{x}_j, \sqrt{2}\sigma) \\ -2\textstyle\sum_{ij}(\mathbf{a}_i \cdot \mathbf{b}_j) g((\mathbf{x}_i - \mathbf{x}) - (\mathbf{e}_j - \mathbf{m}(\mathbf{x})), \sqrt{2}\sigma) \\ + \textstyle\sum_{ij}(\mathbf{b}_i \cdot \mathbf{b}_j) g(\mathbf{e}_i - \mathbf{e}_j, \sqrt{2}\sigma), \end{aligned} \tag{3.5}$$

where the points $\mathbf{e}_i$ and $\mathbf{x}_i$ are in the neighborhood defined by the window function, for the example and output domains, respectively, and the equality is up to a constant. Hence, the functions $\mathbf{f}$ and $\mathbf{e}$ are replaced by their representations with the sets of point locations and attributes. The resulting discrete similarity measure for the case when a Gaussian of width δ is chosen for representing the window function, i.e., $w(\mathbf{s}) = g(\mathbf{s}, \delta)$, is shown in Appendix A.1. For numerical efficiency, we usually truncate Gaussians below a given threshold value, making similarity density measure local. Thus, we only consider the points that are 3δ distance apart from $\mathbf{x}$ in the output, and $\mathbf{m}(\mathbf{x})$ in the input example.

Similarity Error This discrete similarity density can now be used to measure the total similarity T between a synthesized point set and the example point set as defined in Equation 3.3. This requires computing another integral over all points in the output domain. Note that the first and last terms in Equation 3.5 do not depend on $\mathbf{x}$. However, the second term involves the mapping function $\mathbf{m}(\mathbf{x})$ that assigns each point $\mathbf{x} \in \Omega$ to the point in the input example domain with the matching neighborhood. This makes it impossible to take the final integral analytically. Hence, we resort to a numerical scheme where the output domain Ω is sampled with a regular grid. Since the integrand is a sum of shifted Gaussians of standard deviation σ, it has a fixed effective bandwidth, allowing us to use an optimal spacing between the grid points.

Let $\mathbf{q}_k$ denote the background integration points, all having a constant integration domain associated (Figure 3.2, c, right). Furthermore, let $\mathbf{m}_k = \mathbf{m}(\mathbf{q}_k)$ the associated best matching location for the quadrature points in the example domain (Figure 3.2, c, left). This then leads to the *discrete similarity*

error

$$T = \sum_k S(\mathbf{f}(\mathbf{q}_k), \mathbf{e}(\mathbf{m}_k)), \tag{3.6}$$

which compares the geometric neighborhood structure in the output around each quadrature point $\mathbf{q}_k$ to the best matching corresponding neighborhood in the example set.

3.3.4 Discussion

The proposed similarity error T measures how well each neighborhood in the output matches to its neighborhood in the input example. Thus, it follows the patch based texture synthesis approaches that rely on a Markov Random Field model [Kwatra et al., 2005]. Similar to these raster based approaches, it conceptually compares functions defined in the whole domain. However, the free parameters are no longer only the functional values, but also the point locations themselves. Similar to meshless surface reconstruction methods, the functions are adaptively represented such that only relevant structures are sampled. Representing structures with point samples gives us full flexibility in handling structures of different kinds and their combinations. This is contrast with methods that rely on certain assumptions on what the point samples represent and design the matching metrics accordingly [Landes et al., 2013; Lu et al., 2014; Xing et al., 2014].

Neighborhood matching metric We thus share the generality of point sampling based discrete element texture synthesis methods [Ma et al., 2011; Ma et al., 2013]. In contrast to these methods, however, we do not directly match difference vectors in the neighborhoods. Defining matching metrics directly on points is challenging, as the matching scores critically depend on the sampling in that case. Indeed, previous works [Ma et al., 2011; Ma et al., 2013] have a one-way matching score, where each vector $\mathbf{x}_j - \mathbf{x}_i$ for $\mathbf{x}_j$ in the neighborhood of $\mathbf{x}_i$ in the output is matched to the most similar vector in the input example. This implies that for output point sets where the points do not completely represent the desired structures, the matching energy will still be low, as illustrated in Figure 3.3. In contrast, our energy utilizes a two-way matching, leading to lower energy values only when the neighborhoods in the output and input are structurally similar (Figure 3.3).

The sampling dependency of previous methods [Ma et al., 2011; Ma et al., 2013] also means that the neighborhoods of only sample points are matched. Thus, depending on the sampling, some neighborhoods might be left unmatched, if they are covered with less samples. In contrast, we separate the

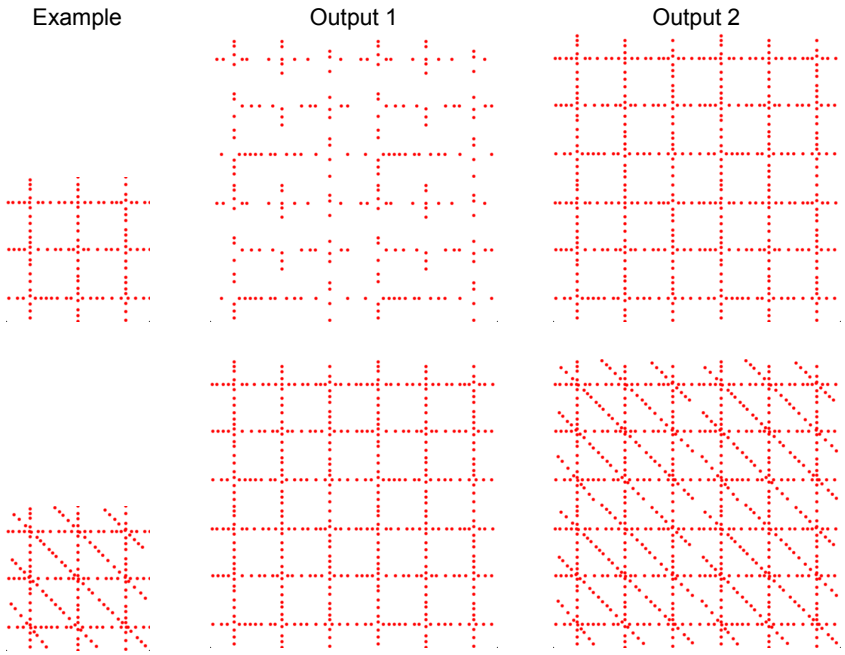

Figure 3.3: *We show two different outputs for each input example. For both outputs* 1 *and* 2*, the neighborhood matching energy defined by Ma et al. [2011; 2013] stays the same due to the one-way matching the pairs difference vectors from the output to those in the input example. In contrast, our method has 2 times higher energy for the wrong output* 1 *for each example.*

structure representation (sample points $\mathbf{x}_i$ and $\mathbf{e}_i$), and neighborhood matching (quadrature points $\mathbf{q}_k$), enforcing that all neighborhoods are matched equally well, regardless of the sampling. With the previous methods, the $\mathbf{q}_k$ and $\mathbf{m}_k$ are constrained to be among $\mathbf{x}_i$ and $\mathbf{e}_i$, respectively. We will see in Section 3.6 that this results in better preservation of local structures, which is very important especially when continuous structures in general domains are to be synthesized.

Attributes The representation of output and example functions in Equation 3.4 is different from previous attribute representations, where they are treated as additional point locations. Treating attributes as scaling factors for the Gaussians significantly reduces the dimensionality of the matching problem, especially when the attributed $\mathbf{a}_i$ live in high dimensional spaces. Since we regularly sample the output domain, this is an important consideration. Furthermore, we do not need to consider relative scaling between the spatial domain and the attributes, as in the previous works [Ma et al., 2011; Ma et al., 2013]. If the attributes are discrete, we snap to the closest discrete value after each optimization step, as described in Section 3.4. Equation 3.4 implies that putting two points at the sample location, or doubling an attribute

(assuming it is scalar) at that location results in the same representation. As we elaborate in the next section, we avoid such cases by carefully controlling the density of the points, and optimizing for the attributes and the point location separately.

Smooth approximation The smooth representation with sums of Gaussians (Equation 3.4) allows us to compute analytic integrals and derivatives, which are essential for efficient and accurate synthesis via well-established optimization methods, as we will illustrate in the next sections.

3.4 Structure Synthesis

The smooth representation of the structures allows us to formulate a robust optimization procedure with standard optimization methods. Our optimization follows the same basic iterative steps of previous neighborhood-based texture synthesis methods: a matching step for the neighborhoods, and a merging step that computes the positions of the sample points based on the matched neighborhoods. However, unlike previous methods, the matching measure is decoupled from the sampling points that represent the structures, i.e., we have the quadrature points $\mathbf{q}_k$ to measure how well the neighborhoods match. This distinction allows us to robustly handle the matching step, regardless of how the structures are sampled or represented.

Algorithm 1: Multi-scale Local-Global Solver

```
initialize x_i
initialize window size δ
loop multiScalelIterations times
    loop samplingControlIterations times
        loop localGlobalIterations times
            matching: {m_i} ← arg min_{m_i} T
            merging: {x_i}, {a_i} ← arg min_{x_i},{a_i} T
        end
        sampling control: add/remove x_i based on P(·)
    end
    δ ← α · δ where α ∈ [0, 1]
end
```

3.4.1 Multi-scale Local-Global Solver

To synthesize the output point set, we minimize the total error T defined in Equation 3.6 with respect to the point locations $\mathbf{x}_i$ and attributes $\mathbf{a}_i$. We perform this optimization with an alternating approach as outlined in Algorithm 1, where each step is guaranteed to decrease the similarity error T.

Local Step: Matching We first compute the best match for each quadrature point neighborhood by finding the matching $\mathbf{m}_k$ for each quadrature point $\mathbf{q}_k$. A notable property of this approach is that the matching point $\mathbf{m}_k$ for a given $\mathbf{q}_k$ can be optimized independently of the other quadrature points since $\partial T/\partial \mathbf{m}_k = \partial S/\partial \mathbf{m}_k$. This makes this local step of the problem highly

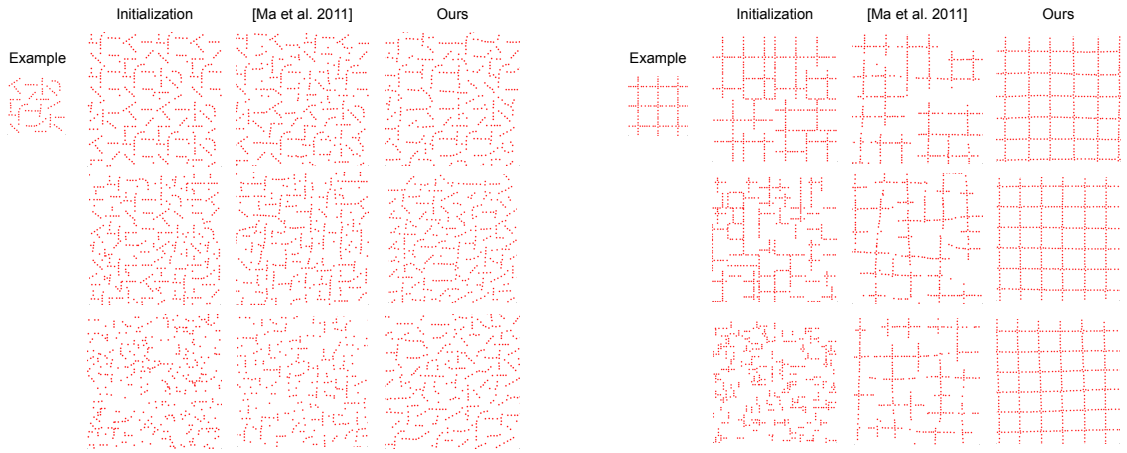

Figure 3.4: *Different initializations computed by copying patches of different sizes from the example to the output are used for each synthesis result. For all initializations, our method can generate accurate discrete (left) as well as continuous (right) structures. The previous methods require copying of considerably larger patches to initialize the synthesis, resulting in less randomness and a patchy look as illustrated in the left-topmost synthesis result. When continuous structures are present (right), even such large patches are not sufficient to generate accurate structures.*

parallelizable. We employ a simple gradient descent procedure to find the best matching locations. The necessary gradients $\partial S/\partial \mathbf{m}_k$ are given in Appendix A.1. This optimization is prone to get stuck in a local minimum and not finding the best match. Therefore we run this optimization with five different random initial positions for $\mathbf{m}_k$ and choose the best match. The repetitive pattern of the input exemplar guarantees the existence of many good local minima, thus we found five seeds to be sufficient for the optimization to succeed.

Global Step: Merging After having found best matches for all quadrature points, we adapt the output point set structure to be as similar as possible to these local neighborhoods as demanded by the similarity metric, i.e., we minimize T for all output points $\mathbf{x}_i$ and attributes $\mathbf{a}_i$. This is a globally coupled nonlinear optimization problem. We again employ a gradient descent procedure where we sequentially optimize the points and attributes, one after the other, in a Gauss-Seidel manner. Optimizing first for points $\mathbf{x}_i$, combined with a sampling control stage as explained below, prevents points from clustering to compensate for the difference in attributes. The required gradients $\partial T/\partial \mathbf{x}_i$ and $\partial T/\partial \mathbf{a}_i$ are provided in Appendix A.1.

Choosing such a simple optimization scheme allows us to parallelize this step

due to the local support of the window function w, and to have a consistent decrease in the energy at each step. The non-convexity of our similarity error function leads to the presence of multiple local minima. For the goal of geometry synthesis this is expected, as there are multiple configurations exhibiting the desired structures. It is in fact an advantage: having different regions ending up in different local minima increases the diversity among generated configurations.

Multi-scale Optimization Once the optimization converges sufficiently for a given window size, the best concense has been found for this feature size. However, this compromise can result in geometries where local features are not close to example features. In order to fix the smaller scale structures, we apply a multi-scale optimization where we decrease the window size starting from the initially provided window size, such that the algorithm can continue the descent to improve the local small scale details. By going from larger to smaller scales, we make sure that the larger structures, which are harder to reproduce, are matched first. The optimization then continues with refinements in smaller scales.

Sampling Control So far we assumed that the regions in the output domain always contain the optimal amount of points regarding the similarity measure, i.e., that there is no mismatch in the number of points and that errors only

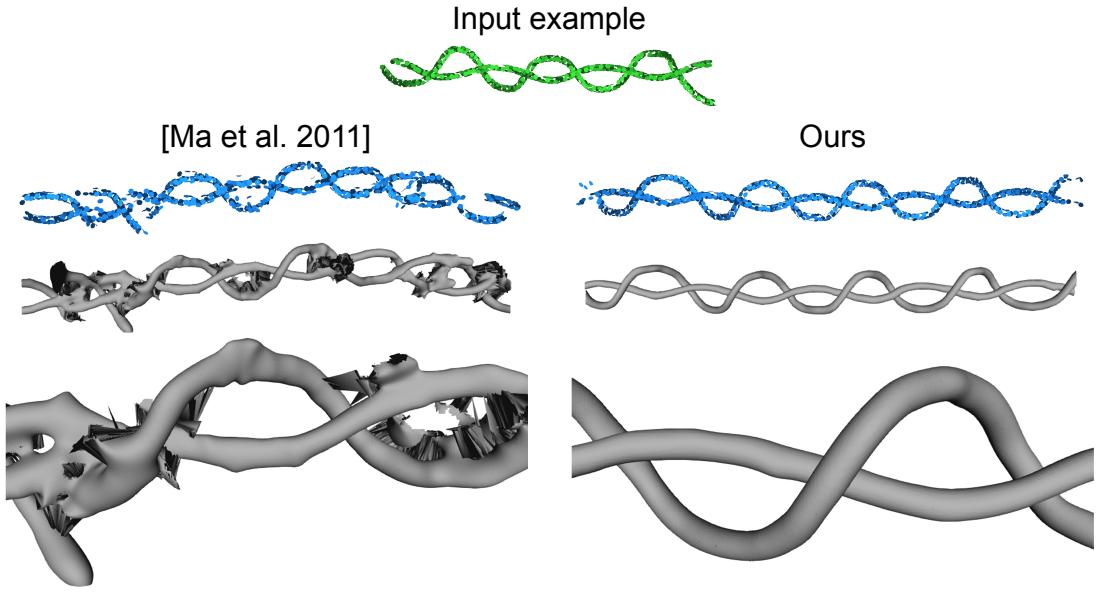

Figure 3.5: *Given the input example with points and normals (top), outputs point sets are synthesized (second row) and reconstructed (third and fourth row) using moving least squares surfaces [Öztireli et al., 2009]. Our representation and synthesis algorithm accurately handles such continuous structures.*

occur from non-optimal point locations and attributes. However, too many or too few points in a region impair our similarity measure, as for all point sample based texture synthesis methods. If there are not enough points, the structures in the example will be partially reproduced. Conversely, too many points will result in excess points that force the optimization to destroy structures to accommodate the extra points. In order to prevent this, we employ a sampling control strategy during the optimization, based on the same principle of our similarity error density measure.

In order to detect the deviation in the sampling density around a quadrature point we define a *sampling error density* function

$$P(\mathbf{x}, \mathbf{m}(\mathbf{x})) = \int \left(f(\mathbf{x}+\mathbf{s}) - e(\mathbf{m}(\mathbf{x})+\mathbf{s})\right) w(\mathbf{s}) \mathbf{ds} \tag{3.7}$$

that measures the *signed* difference between the output and example functions. However, here $f(\cdot)$ and $e(\cdot)$ are special instances of the two functions where $\mathbf{a}_i = \mathbf{b}_i = 1$ such that solely point locations are taken into account. If $P(\mathbf{x}, \mathbf{m}(\mathbf{x})) < 0$ for a given point $\mathbf{x}$, the function e in the example domain is on average larger than the output f, implying removal of points. Similarly, $P(\mathbf{x}, \mathbf{m}(\mathbf{x})) > 0$ calls for adding a point to match the functions f and e. For robustness, we introduce a threshold $\epsilon = 10^{-6}$ and add a new point in vicinity of the quadrature point if $P < -\epsilon$, or remove an unnecessary point if $P > \epsilon$. Once a new random point is added, we optimize for its location and attribute, and decide to keep the optimized point or not by checking whether the energy increases or decreases. The same check is performed for the case of removing a point.

3.4.2 Discussion

Robustness to initialization Our two-way neighborhood matching, as discussed in Section 3.3.4, combined with the optimization algorithm presented in the last section makes our method robust to initializations, as compared to the previous methods that critically depend on the initial distribution of points in the output [Ma et al., 2011; Ma et al., 2013]. We illustrate the robustness to initialization in Figure 3.4. It is especially important for synthesizing continuous structures (Figure 3.4, right), where even a single neighborhood mismatch can lead to visually disturbing reconstructions (Figure 3.5).

Control sampling Dynamically controlling the sampling by adding and removing points in the optimization is a key component for interactive synthesis, where the user continuously extends the region he wants to texture by brushing (Section 3.5.2). This is in contrast with the previous offline discrete

element texture synthesis methods. Adding/removing points for interactive synthesis has recently been explored for 2D drawing applications [Xing et al., 2014], and proposed as an optimization method for shape processes via MCMC [Landes et al., 2013]. In contrast to these approaches, our technique offers a general unified adding/removing strategy that is interleaved with the optimization steps that move the points for accurate placement. Such accurate placement takes considerably more time to obtain by merely adding entities [Landes et al., 2013].

3.5 Controlling Structures

We control the synthesized structures in a two-scale approach, where the small scale geometry is given by the example, and the large scale behavior is controlled by the user, similar to the existing methods [Ma et al., 2011; Ma et al., 2013]. The main strength of our structure definition and synthesis method is that we can accurately handle arbitrary structures represented in various forms. This turns our method into a powerful tool for users to create complex general output structures interactively, in contrast to the previous methods offline [Ma et al., 2011; Ma et al., 2013], or interactive [Kazi et al., 2012a; Xing et al., 2014] methods. In this section, we first show how the generic definition can be leveraged to generate and control various types of output representations by simply changing what the point locations and attributes represent. We then show how orientation and scaling fields can be used to steer the structure synthesis by specifying geometries of different dimensions.

3.5.1 Structure Representations

Our point samples can represent a variety of structures sparsely, ranging from discrete to continuous. This allows us to handle different structures, and their mixtures, in the same framework.

Continuous Structures Continuous geometry can be achieved by storing scalar and/or vectorial point attributes. A scalar value, such as a point color or radius, can be assigned and used for texturing and rendering, i.e., to extract an isosurface. Equipping each particle with a vector-valued surface normal attribute further allows to employ recent meshless surface reconstruction techniques [Öztireli et al., 2009] within our framework to synthesize high-quality surfaces.

Discrete Elements A discrete element can be represented by one or more samples in our framework. Representation with a single sample works well for cases where collisions are not a problem, such as the leaves in Figure 3.1. However, for more complex structures, we should have a sufficient sampling [Ma et al., 2011; Landes et al., 2013]. We represent such discrete elements with multiple points, often sampled on the surface of the element, and sometimes equipped with normals. Points belonging to a single element are added, removed, and moved together in the optimization, by treating them as a single point, i.e. by setting $\mathbf{x}_i = \mathbf{x}_0$ for all points $\mathbf{x}_i$ in the discrete structure, and optimizing with respect to $\mathbf{x}_0$. Note that the optimization procedure stays exactly the same as for the continuous structures.

Mixtures of Structures The common definition for continuous and discrete structures allows us to handle their mixtures seamlessly, since both are represented and optimized in the same way. An example synthesis results with structure mixtures containing discrete elements such as the organic structure and the discrete gems is illustrated in Figure 3.17. We can also use mixtures of discrete and continuous attributes if the structures require such a representation. As an example, we used a grouping attribute for the points to synthesize the structure in Figure 3.12, such that each bean is assigned a different grouping number. Note that this does not prevent the beans to exhibit random variations, as they are still represented as continuous structures with surface points and normals.

3.5.2 Large-scale Control

The large scale geometry is intuitively controlled by the user as orientation and scaling fields are defined along curves, triangular surfaces, or bounded volumes. We first discuss how our system can be extended to handle rotation and scale, and then explain how these guiding geometries are defined.

Rotation and Scale We extend the optimization to account for orientation and scale of the output geometry by adding a rotation $\mathbf{R}(\mathbf{x})$ and scaling factor $s(\mathbf{x})$ to the similarity error density shown in Equation 3.2, resulting in

$$S = \int |\mathbf{f}(\mathbf{x}+\mathbf{s}) - \mathbf{e}(s(\mathbf{x})\mathbf{R}(\mathbf{x})(\mathbf{m}(\mathbf{x})+\mathbf{s}))|^2 \, w(\mathbf{s})\mathbf{ds}. \tag{3.8}$$

Note that since $\mathbf{R}$ and s define fixed fields independent of the synthesized function $\mathbf{f}$, no modification to the derivatives and hence the optimization procedure is required. As implied by the expression, in practice, we implement this optimization by rotating and scaling the example $\mathbf{e}$ for each point $\mathbf{x}$. Note that in the discretized energy, $\mathbf{x}$ is represented by the quadrature points, and hence these fields are also stored at the quadrature points. Once they are stored, we can then run the same optimization. The fields $\mathbf{R}$ and s depend on the design metaphor utilized as discussed next. For scaling, the standard deviation σ of the Gaussians used to define the input and output functions in Equation 3.4 needs to be adapted according to the scaling function s as well.

Brush strokes With our system we can generate new geometries by drawing lines in 3D space. Quadrature points are automatically generated along and near the drawn lines. The drawing direction can be exploited to automatically set the rotation $\mathbf{R}$ of each quadrature point, such that the direction of the input example is aligned with the stroke tangent. The scaling factor s is manually set by the user as she/he draws the strokes. The user can also control the

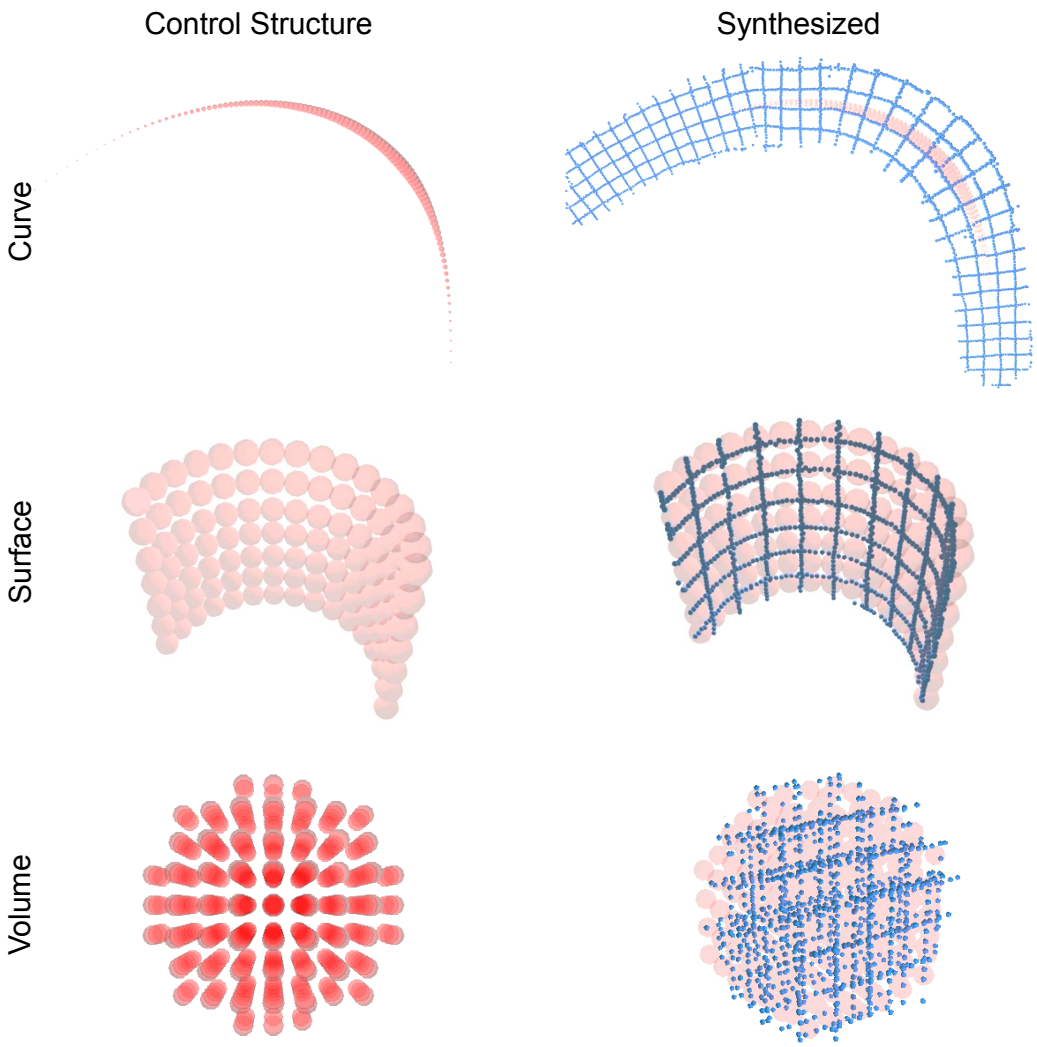

Figure 3.6: *The guiding fields can be defined along curves, surfaces, or volumes to control the large-scale geometry.*

brush size, which determines the region to be textured. Examples of this type of control is given in Figures 3.6 and top, 3.7. The brush direction defines the orientation, and the varying scaling factors smoothly change the pitch.

Triangular surfaces The guiding fields can also be defined on surfaces. As an example, we used the principle curvature directions as an orientation field in Figure 3.6, middle. Hence, we can add fine-scale details to existing surface geometries with our technique, as we further illustrate in the next section.

Volumetric synthesis This naturally extends to 2D and 3D volumes, where the fields are defined throughout the ambient space. The user can also utilize a 3D surface to shape the output structure as shown in Figure 3.6, bottom, and

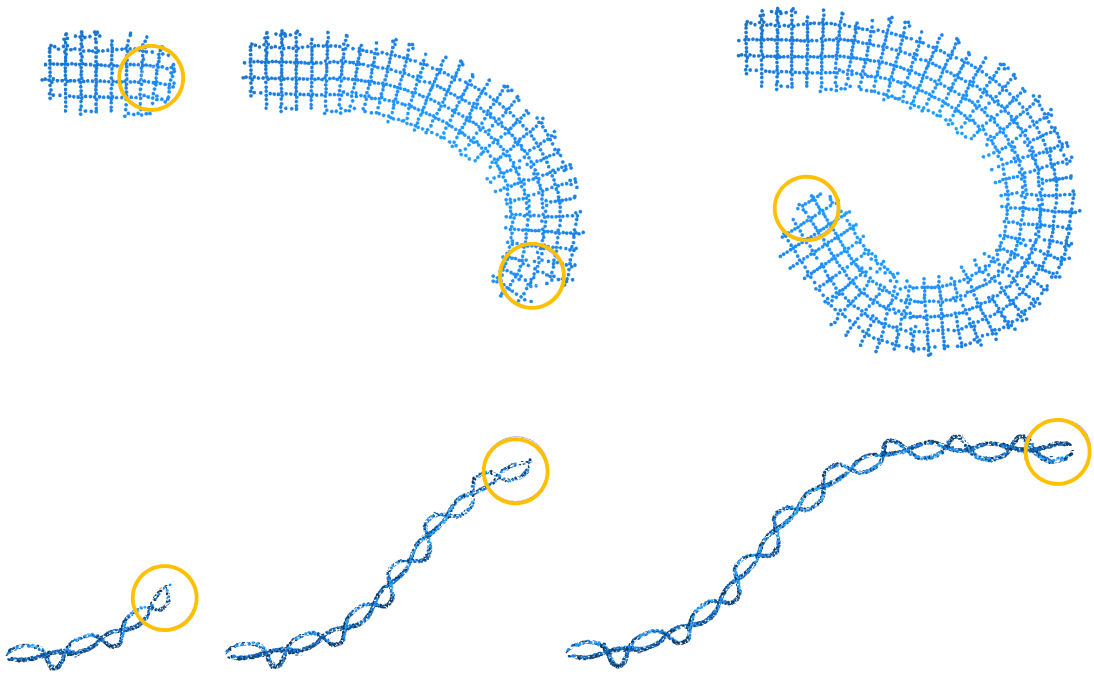

Figure 3.7: *The structures grow in real-time as the user interactively brushes (orange circle indicate brushing region).*

the chair example in Figure 3.1. The quadrature points are sampled inside the provided volume to avoid unnecessary computations.

3.6 Results

3.6.1 Implementation and Parameters

For our representation, σ gives the standard deviation of the Gaussians used in defining the functions $\mathbf{f}$. The parameter σ determines the smoothness of the matched functions, which in turn depends on the sampling of the structures. If the structures are sampled densely in the input example, we can set it to lower values to capture smaller scale details. Thus, in our implementation, it is set to the average spacing between the points $\mathbf{e}_i$ in the example. Once σ is set, the optimum spacing of the quadrature points $\mathbf{q}_k$ in the output can be analytically determined as the Gaussians are band-limited. The parameter δ determines the spatial extent of the window function w. The neighborhood size and hence the δ is provided by the user based on the expected scale of repetitions.

In the synthesis algorithm (Algorithm 1), the optimization by gradient descent is run till the average movement of the points $\mathbf{x}_i$ is below a threshold $\epsilon = 10^{-6}$ or the maximum number of iterations (we used 30) is reached. The step size for the gradient descent is set to 0.03. After this optimization, each quadrature point is checked for the condition on P as defined in Section 3.4.1, and a point is removed or added accordingly. These steps are repeated for a new shrunken neighborhood size by setting $\alpha = 0.9$. Once the new neighborhood size is half of the original size, the algorithm is stopped. We use a kd-tree to speed up neighborhood queries.

3.6.2 Analysis and Comparisons

Our new matching metric combined with the synthesis algorithm with sampling control leads to accurate reproduction of repeated patterns, independent of the initializations and complexity of the structures (please see Sections 3.3.4 and 3.4.2 for discussions on these properties). We illustrate robustness to initial conditions in Figures 3.4 and 3.5. As illustrated in Figure 3.4, top row, utilizing the initialization strategy proposed by Ma et al. [2011] leads to preservation of structures when continuity is not essential, although the resulting synthesized point sets look quite similar to the initializations with visible patches from the input example. However, continuous structures cannot be accurately synthesized with this technique [Ma et al., 2011], as shown in Figure 3.4, top row, right. Furthermore, even small changes in the initial point sets can lead to convergence to bad local minima, resulting in distorted structures with the method of Ma et al. [2011] (Figure 3.4, second

Example	#input points	#output points	Run time Us	Run time Ma et al.
helix	600	100	4 s	12 s
sand	1600	7200	2 min	9 min
chair	1100	23000	6 min	40 min

Table 3.1: *Run times for three different results.*

and third rows). In contrast, our algorithm is not affected by the initialization. This property is especially important if the resulting point sets are used for reconstructing smooth surfaces, as shown in Figure 3.5.

Contrary to Ma et al. [2011], our smooth formulation does not require solving a linear system at each iteration. We can thus achieve better performance which also allows for interactive results, and use input examples with a large number of points (usually more than 1000), which is essential for representing continuous structures. Figure 3.8 shows the convergence behavior for a point set of 1000 points for the input examples shown in Figure 3.7, top (2D), and Figure 3.18, bottom (3D). Table 3.1 shows the run time for the results in Figures 3.7 (bottom), 3.18 (bottom) and 3.1 (center), achieved with our method and with the technique of Ma et al. [2011]. An interactive brush was used for the helix example in Figure 3.7, thus the presented run time refers to the points within the brush (around 100). We run our method until convergence, and the one by Ma et al. [2011] for 10 iterations, as in the mentioned work. The neighborhood size in these three examples was about one fourth of the size of the input. Moreover, in Figure 3.9 we present the time required to run one iteration for the two methods, with different input and output sizes. One can notice that our method scales better with respect to the input and output size, due to our simpler gradient descent based optimization. The results were tested on a PC with an Intel i7-3770K CPU.

The neighborhood size, provided by the user, offers control over the scale of expected repetitions in the input and output. We illustrate its effect on the synthesis results in Figure 3.10. A bigger size can be used to preserve large scale structures, while a smaller size leaves room for more randomness.

Our technique allows for interactive brushing in general domains, and synthesizing mixtures of continuous and discrete structures. We provide examples of such results in the next sections.

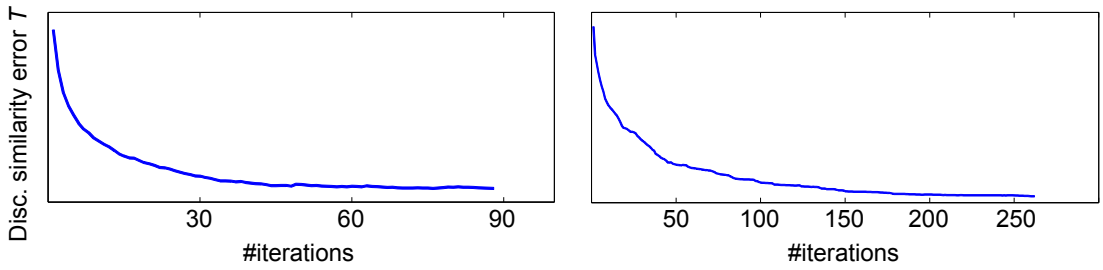

Figure 3.8: *Discrete similarity error T as a function of the number of iterations for the optimization of* 1000 *points in the output. On the left, the 2D grid input in Figure 3.7, on the right, the 3D beach input in Figure 3.18, bottom is analyzed.*

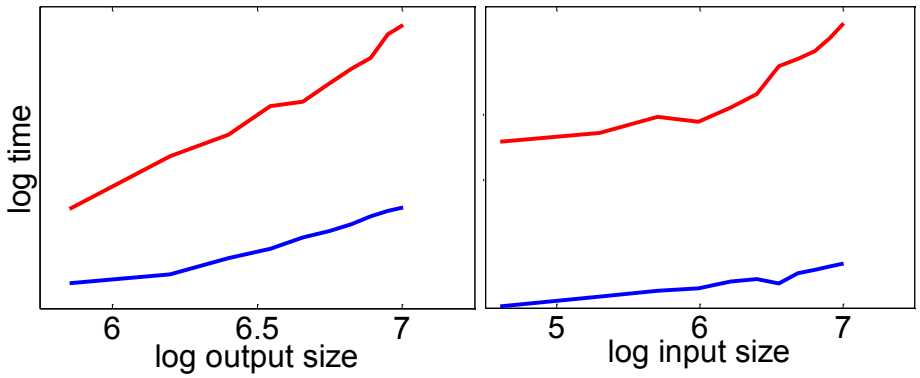

Figure 3.9: *Run time for one iteration for our method (blue) and Ma et al. [2011] (red). On the left image, the input exemplar contains 100 points, and the output structure varies from 300 to 3000 points. On the right, the output structure contains 1000 points, and the input exemplar varies from 100 to 1000.*

3.6.3 Synthesis Examples

We applied the technique to synthesize various structures ranging from discrete to continuous in different domains including curves, surfaces, and volumes. Some of the controlling domains are sketched interactively. For all the continuous structures in 3D, except for the one in Figure 3.13, we used points with normals sampled on the surface in the examples, synthesized the output point cloud with normals, and reconstructed the results with a moving least squares based surface reconstruction method [Öztireli et al., 2009]. For the example shown in Figure 3.13, we reconstructed the isosurface by placing a metaball (blob) at each point location. Locations and sizes of

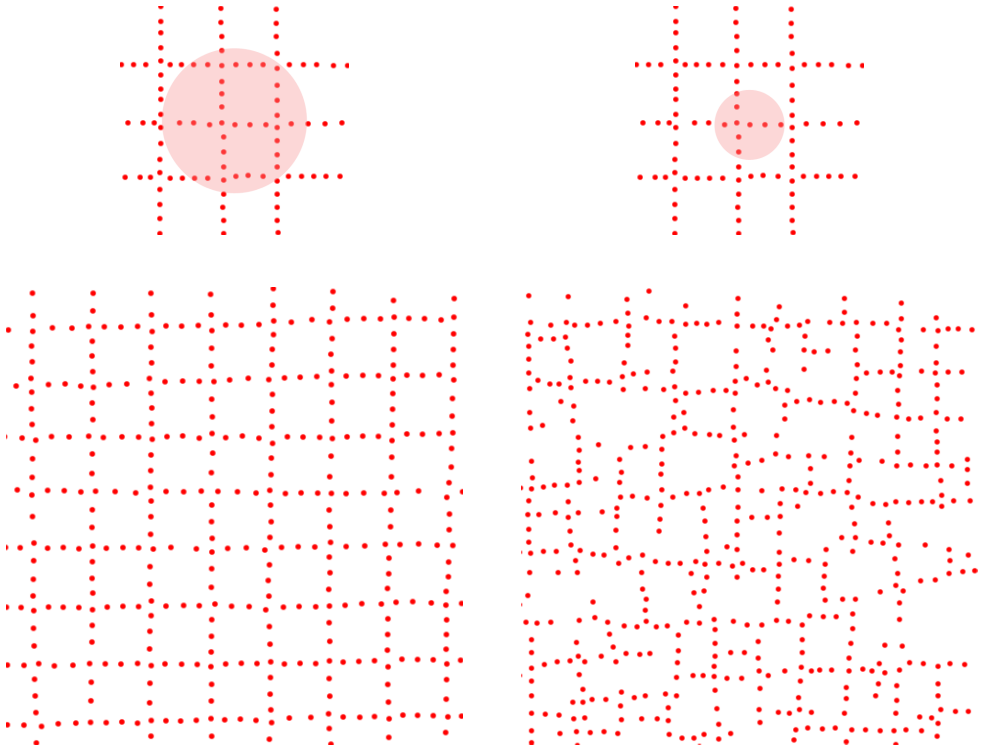

Figure 3.10: *Influence of the neighborhood size in the input example (top, red circle) on the synthesized result (bottom).*

the metaball are optimized according to the input. For discrete structures in 3D, we simply kept the original geometry representation, e.g. if the points are vertices of a mesh in the input example, the same mesh also appears in the output. For some of the results, we also included colors as attributes, as depicted in the corresponding figures.

In Figure 3.1, models of wooden sticks (discrete elements) have been placed on the hut surface shown in red.

The output structures representing the intertwined helix in Figure 3.11 and the chain in Figure 3.12 have been interactively generated along curves. The user draws a curve and specifies orientation and scaling fields along it, and the points are then synthesized accordingly. For smaller input examples (less than about 500 points), the output can be synthesized in real-time with a brushing interface, as in Figure 3.7 for the 2D grid and the 3D helix. The continuous surfaces have been reconstructed offline using moving least squares. Synthesizing continuous surfaces onto a triangular surface is shown in Figure 3.14. The input surface, shown on the left in red, defines the global shape of the structure, i.e., the deep-water waves. With our method we can synthe-

size small-scale capillary waves onto the surface, improving the realism of the water surface. In the same way, bumps have been generated on the stone structure in Figure 3.15. An example of using bounded volumes is presented in Figure 3.1. A 3D model of a chair is included, as shown on the right in red, defining the global shape. The chair is sampled by quadrature points during initialization, and then the volume is synthesized by an organic structure. In a post-processing step, the borders of the reconstructed output structure are cut using the same model of the chair. The structure in Figure 3.13 has been generated using metaballs of different sizes. Branches at the extremities are thinner, thus generated from smaller metaballs than the ones close to the main branch. Like in the case for moving least squares, our control sampling strategy is essential to reconstruct continuous, connected structures from the point clouds.

As our method generalizes previous approaches, we can simply combine discrete and continuous attributes. We present several examples to illustrate such mixed structures. The ivy example presented in Figure 3.1 shows a continuous ivy structure that is synthesized along a curve, similarly to the intertwined helix (Figure 3.11). The leaves and flowers are discrete elements represented as proxy samples in the input point set. Other examples using mixed structures are shown for a stone wall, a decorative object with gems, and a beach with shells in Figures 3.16, 3.17 and 3.18, respectively. For the first two, around 20 points have been sampled on the surface of each discrete element.

3.6.4 Limitations

The structure representation and thus the synthesis algorithm depends on a smooth approximation with Gaussians. As described in Section 3.6.1, we determine a global standard deviation given by σ for the Gaussians based on point spacing, to accurately capture the structures. This was sufficient for reproducing a variety of structures as presented. However, an adaptive σ can further improve the results, especially when there are far apart tight clusters of points in the input example.

Similar to the previous methods, we utilize a neighborhood size to capture the repetitions in the input examples. As shown in Figure 3.10, this size is an example-dependent parameter and thus should be provided by the user for artistic control, reflecting his/her view of the scale of repetitions in the example. However, if there are repetitions of different scales in the same example, setting a single size can be detrimental for the preservation of the repetitive structures at the other scales. In particular, we encountered this

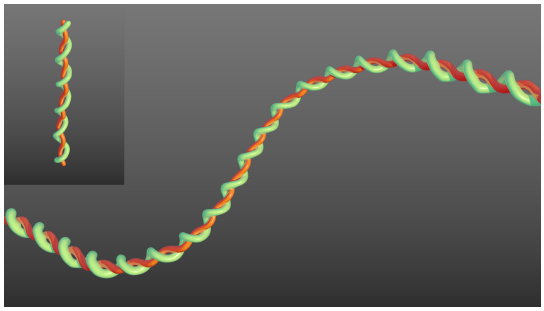

Figure 3.11: *The spiral is synthesized by specifying orientation and scaling fields along a curve.*

Figure 3.12: *The chain is synthesized with surface normals as attributes, and reconstructed.*

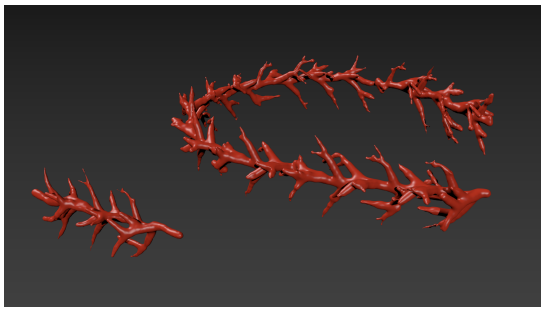

Figure 3.13: *Metaballs of different sizes are used to reconstruct a continuous surface along a curve.*

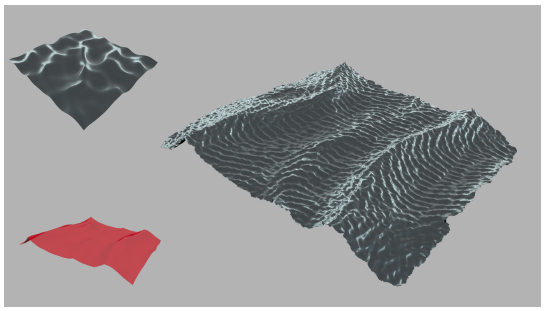

Figure 3.14: *Capillary waves synthesized onto coarse mesh defining the deep water waves.*

problem only in some of the discrete-continuous mixtures we experimented with, where samples representing discrete elements have a considerably larger scale of repetitions than the ones composing continuous structures. As illustrated in Figure 3.18, top, the arrangement of the sea shells (each of which is regarded as a discrete element) is not fully preserved, since the scale of the repetition of that pattern is much larger than that of the continuous background. On the other hand, purely continuous or discrete exemplars are not affected by this limitation, as these structures usually exhibit repetitions of similar scales, allowing us to use our approach for further applications, such as painting [Kazi et al., 2012b].

Moreover, like in the other neighborhood-based texture synthesis methods, our exemplars are required to have a repetitive enough pattern. This allows our matching optimization with multiple initial positions to avoid bad local minima, which would distort the synthesized structures.

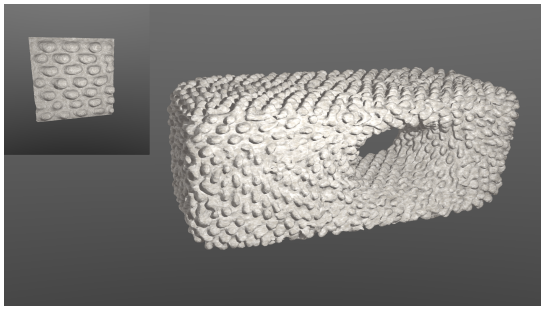

Figure 3.15: *Another example of synthesizing continuous structures (bumps) on a large scale surface.*

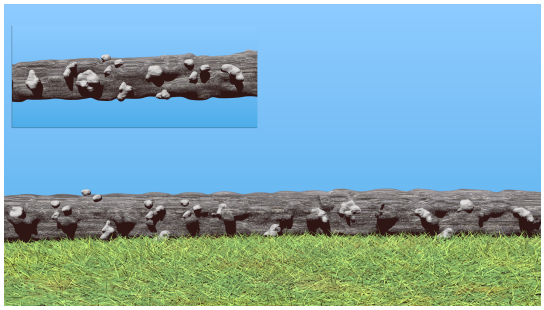

Figure 3.16: *The synthesized mixture of discrete elements of stones of lighter color and continuous background wall.*

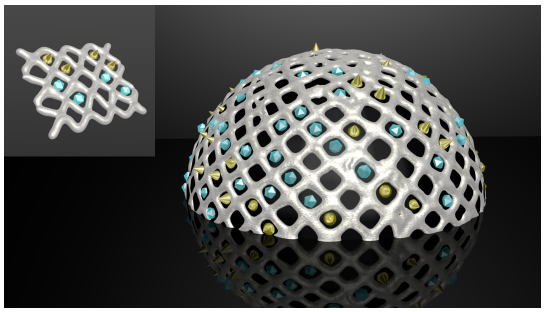

Figure 3.17: *Another example of synthesizing mixtures of discrete elements (gems) with continuous structures.*

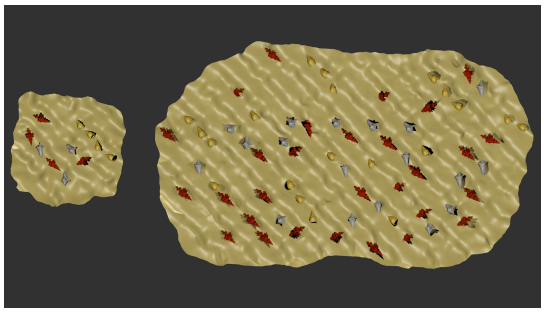

Figure 3.18: *A mixture of discrete elements (shells) are placed on a continuous structure (sand).*

3.7 Discussion

We presented a new point-based method for synthesizing general repeated structures given by an input example. In contrast to previous geometric texture synthesis methods, our technique offers treating continuous and discrete structures residing on general domains in the same framework, exhibits better and initialization independent preservation of structures, and allows for interactive repetitive synthesis of general structures.

Future work

We presented only a few applications of our method. It can be utilized in all applications where repetitive structures are utilized for synthesizing missing or augmenting existing structures. Some immediate applications are surface and image reconstruction, completion, consolidation, inpainting, or superresolution. The technique can be combined with physical or fabrication constraints for simulations or fabrication oriented design. By adding a time dimension, similar to previous methods [Ma et al., 2013], it can also be used for animation generation, and in combination with physical simulations.

The technique is well-suited for interactive authoring of such difficult structures. We utilized a simple stroke drawing interface for real-time results. It can be extended with more sophisticated and possibly application dependent metaphors for interactive synthesis.

CHAPTER 4

General Point Sampling with Adaptive Density and Correlations

As observed in Chapter 3, many geometry structures are composed by discrete elements and represented by their patterns. Common examples are the distribution of trees in a forest or stones in a wall. Analyzing and generating sampling patterns are fundamental problems for many applications in computer graphics. Ideally, point patterns should conform to the problem at hand with spatially adaptive density and correlations. Although there exist algorithms that can generate point distributions with spatially adaptive density or anisotropy, including our method in Chapter 3, the pair-wise correlation model, blue noise being the most common, is assumed to be constant throughout the space. Analogously, by relying on possibly modulated pair-wise difference vectors, the analysis methods are designed to study only such spatially constant correlations. In this chapter, we present the first techniques to analyze and synthesize point patterns with adaptive density and correlations. This provides a comprehensive framework for understanding and utilizing general point sampling. Starting from fundamental measures from stochastic point processes, we propose an analysis framework for general distributions, and a novel synthesis algorithm that can generate point distributions with spatio-temporally adaptive density and correlations based on a locally stationary point process model. In addition to a more general synthesis method matching neighborhood statistics, we extend our discrete texture synthesis algorithm presented in Chapter 3 to handle distributions with adaptive correlations, by providing multiple input example distribu-

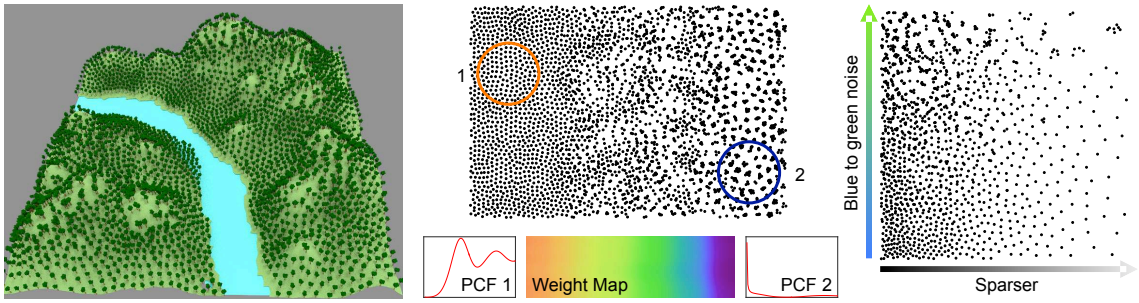

Figure 4.1: *Understanding natural or synthetic complex distributions such as the natural distribution of trees based on the altitude on the left is a difficult problem if the arrangement of the entities, resulting from pair-wise interactions, is spatially-adaptive, as shown for a canonical example in the middle. Our analysis technique provides an informative and comprehensive summary of such distributions. The correlations in our framework are represented with a set of extracted basis pair correlation functions (PCF 1 and 2, from local patches 1 and 2 in the example in the middle), and the corresponding weight maps illustrating how they are interpolated in space. Our synthesis algorithm utilizes these measures to synthesize distributions with adaptive density and correlations on Euclidean domains (right) or surfaces (left).*

tions instead of a single one. Our techniques also adapt to general metric spaces. We illustrate the utility of the new techniques on the analysis and synthesis of real-world distributions, image reconstruction, spatio-temporal stippling, and geometry sampling.

4.1 Introduction

Sampling patterns lie at the heart of many important applications in computer graphics such as representing and integrating functions, anti-aliasing, image and geometry sampling, physically-based simulation, non-photorealistic rendering, object and texture placement, and modeling natural distributions. It is thus essential to understand and control characteristics of sampling patterns.

Point distributions are generally characterized by the *density* and the arrangement of the sampling points given by the *pair-wise correlations* among point locations, as higher order correlations are not needed to study most patterns [Illian et al., 2008; Öztireli and Gross, 2012]. The most studied correlation model is based on variations of blue noise patterns where the points

are randomly distributed with a minimum distance between pairs [Ulichney, 1988]. Adaptive density, locally anisotropic distributions, or patterns on surfaces can be obtained by adapting the metrics used accordingly, while keeping the correlation model fixed [Li et al., 2010; de Goes et al., 2012; Jiang et al., 2015]. Recent works further explore matching a given target correlation model [Zhou et al., 2012; Öztireli and Gross, 2012; Wachtel et al., 2014; Ahmed et al., 2015], resulting in an explicitly controlled arrangement of points. Another class of methods [Ma et al., 2011], among them our approach in Chapter 3, is not based on matching an example statistics but rather individual neighborhoods of elements in the example. However, for all the synthesis methods, the correlation model is assumed to be constant throughout the space, resulting in translation invariant patterns up to local density variations. The analysis methods are thus also designed for studying such translation invariant correlations via statistics based on distributions of modulated difference vectors or distances between pairs of points and associated spectral measures [Bowers et al., 2010; Wei and Wang, 2011; Öztireli and Gross, 2012; Heck et al., 2013]. Hence, point patterns where the correlations are spatially varying cannot be handled with the current common analysis and synthesis techniques. In Figure 4.1, we show an example point distribution with spatially-adaptive correlations, where the pattern transitions from blue to green noise from left to right. Utilizing such patterns can lead to significantly improved results for many applications in computer graphics.

In this chapter, we present novel analysis and synthesis techniques for point patterns with adaptive density and *adaptive correlations*. Starting from the theory of stochastic point processes, we propose a novel analysis method for general point patterns, and a novel, general synthesis algorithm capable of generating distributions with spatially adaptive density and correlations. The presented statistics converge to provably discriminative measures from point processes, and provide a comprehensive framework for point patterns. Additionally, we extend our discrete texture synthesis approach from Chapter 3 to produce distributions with adaptive correlations which match our proposed statistics. We illustrate how such patterns can improve image and geometry sampling with various examples. In summary, our main contributions in this chapter are the following:

- The notion of adaptive correlations, and a comprehensive analysis framework for general point patterns. The proposed measures are based on well-known statistics form stochastic point processes, and reduce to previous analysis tools for the special case of translation invariant patterns.
- Synthesis algorithms for point patterns with adaptive density and

correlations on general domains. We apply the algorithm to generate distributions on Euclidean domains 4.1 (right) as well as surfaces 4.1 (left). In contrast to the previous works, the algorithm offers full control over the distributional characteristics.

4.2 Analysis of General Sampling Patterns

In this section, we introduce a theoretical framework to study general point patterns, and a set of tools and diagrams that allow for qualitative and quantitative understanding of point distributions exhibiting spatially adaptive density and correlations. We will illustrate that applying existing analysis techniques does not yield meaningful statistics for this general case. We start with the most general case where first and second order correlations are considered, and move on to a locally stationary model that describes a wide range of natural and synthetic point patterns.

4.2.1 Stochastic Point Processes

The field of stochastic point processes provides a general mathematical framework to study point patterns. Intuitively, a point process is a generating algorithm or mechanism for a set of distributions that share common characteristics. We utilize this theory as a basis to understand and analyze general point patterns with adaptive correlations. We present a brief introduction to point processes, we refer the readers to the excellent books [Møller and Waagepetersen, 2004; Illian et al., 2008] for a more in-depth discussion.

The main construct to define a point process is assigning a random variable $X(B)$ to every Borel set $B \in \mathcal{D}$ for a given domain $\mathcal{D}$. Hence, a point process is described by infinitely many random variables. If we fix some sets B_i, we can stack all random variables for these sets to have the random vector $X = [X(B_1), \cdots, X(B_n)]^T$. The point process can then be fully defined by the joint probability $\mathbb{P}(X(B_1) \leq b_1, \cdots, X(B_n) \leq b_n)$ of the random variables at B_i for all n and all different sets B_i. A familiar example of such a random variable is the random number of points $N(B)$ in B. We can then study the joint probability of the random variables $N(B_1), \cdots, N(B_n)$ to characterize a point process. For the simplicity of the exposition, we will consider point processes with $\mathcal{D} \in \mathbb{R}^d$ in the discussion below, but the concepts also extend to general measure spaces.

General Gaussian processes. Gaussian processes cover almost all types of point processes encountered in applications [Illian et al., 2008]. These

processes are characterized by having a Gaussian distribution for the random vectors X. Hence, the mean and covariance of the vectors are sufficient to describe Gaussian processes. The importance of such processes is highlighted by the term *second order dogma* in physics [Illian et al., 2008], as they very accurately model all distributions found in nature. For these processes, the first and second order moment measures and the associated product densities are sufficient for a complete specification.

Product densities. All Gaussian processes can be described by the first and second order product densities. The first order product density $\lambda(\mathbf{x})$ is called the intensity of the point process, and intuitively measures the average density of points at $\mathbf{x}$ over different distributions generated by the point process. It is proportional to the probability of finding a point in an infinitesimal volume $d\mathbf{x}$ around $\mathbf{x}$ such that $\lambda(\mathbf{x}) = p(\mathbf{x})d\mathbf{x}$. The expected number of points in a set B is given by the integral $\mathbb{E}_{\mathbf{X}}N(B) = \int_B \lambda(\mathbf{x})$, where the expectation is over different distributions $\mathbf{X} = [\mathbf{x}_i, \cdots]$, $\mathbf{x}_i \in \mathcal{D}$ generated by the point process. The second order product density $\varrho(\mathbf{x}, \mathbf{y})$ is proportional to the joint probability of finding a pair of points in $d\mathbf{x}$ and $d\mathbf{y}$, $\varrho(\mathbf{x}, \mathbf{y})d\mathbf{x}d\mathbf{y} = p(\mathbf{x}, \mathbf{y})$. This statistic determines the pair-wise correlation model, which can be spatially varying for general point processes.

Stationary and isotropic point processes. Stationarity and isotropy are common intrinsic assumptions in the literature. For stationary point processes, the generated distributions are translation invariant. Hence, $\lambda(\mathbf{x})$ is a constant, and $\varrho(\mathbf{x}, \mathbf{y})$ turns into $\varrho(\mathbf{x} - \mathbf{y})$, a function of the difference vector between two points in space. If we further assume rotation invariance, then $\varrho(\|\mathbf{x} - \mathbf{y}\|)$ is a 1D function. For stationary distributions, the second order product density is often expressed in terms of the normalized pair correlation function (PCF) $g(\mathbf{h}) = \varrho(\mathbf{h})/\lambda^2$, where we defined $\mathbf{h} = \mathbf{x} - \mathbf{y}$. Estimators of PCF are used for analysis and synthesis of stationary distributions in recent works [Wei and Wang, 2011; Öztireli and Gross, 2012; Heck et al., 2013]. It is also closely related to the more commonly used periodograms with a Fourier transform [Heck et al., 2013]. Utilizing this simplified form of pair-wise correlations lies at the heart of the main limitation of previous analysis and synthesis methods. Instead, we will utilize the general ϱ, and a model with local stationarity for our techniques.

4.2.2 Locally Stationary Processes

Analysis with general first and second order statistics is in general difficult as the resulting measures are not intuitive, and hard to estimate unless many

instances of the same pattern are available, due to the expectations $\mathbb{E}_{\mathbf{X}}$ involved (we present a formal derivation of the estimators of product densities, and related measures in Appendix A.2). For many cases, however, it can be assumed that these measures exhibit a certain degree of smoothness in space. We can then decompose the pattern into *locally stationary* patterns around each point in space.

Within the neighborhood $\mathcal{N}_{\mathbf{x}}$ of a point $\mathbf{x}$, the pattern is then fully described by a constant intensity λ_x, and a PCF $g_x(\mathbf{h})$. For a stationary distribution, there can only be a global anisotropy due to the translation invariance of the generated point distributions [Illian et al., 2008], such that the PCF can also be written as $g_x(\|\mathbf{h}\|)$ once the neighborhood is reshaped with a matrix $\mathbf{M}_x$ by applying it to all $\mathbf{h}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ for $\mathbf{x}_i, \mathbf{x}_j \in \mathcal{N}_{\mathbf{x}}$ to cancel this global anisotropy (we elaborate on how $\mathbf{M}_x$ can be computed in Appendix A.3).

Given a distribution with the set of points $\{\mathbf{x}_i, \cdots\}$ generated by an underlying point process, the local intensity λ_x and PCF g_x can be estimated. The intensity has a natural estimator as we derive in Appendix A.2 with $\hat{\lambda}_x = \sum_{\mathbf{x}_i \in \mathcal{N}_{\mathbf{x}}} k(\mathbf{x}, \mathbf{x}_i)$ for a normalized kernel such as the Gaussian $k(\mathbf{x}, \mathbf{x}_i) = e^{-||\mathbf{x}-\mathbf{x}_i||^2/\sigma^2}/(\sqrt{\pi}\sigma)^d$. The smoothness of the estimation controlled by σ can be set to reflect the assumed density variation in the distribution.

The estimated PCF can be computed by utilizing an existing smooth estimator for isotropic processes [Illian et al., 2008; Öztireli and Gross, 2012]:

$$\hat{g}_x(\|\mathbf{h}\|) = \frac{1}{\hat{\lambda}^2 |\partial V_d| \|\mathbf{h}\|^{d-1}} \sum_{\mathbf{x}_i \neq \mathbf{x}_j \in \mathcal{N}_{\mathbf{x}}} \frac{k(\|\mathbf{h}\| - \|\mathbf{h}_{ij}\|)}{a_{\Pi_{\mathcal{N}_{\mathbf{x}}}}(\mathbf{h}_{ij})}, \tag{4.1}$$

where $|\partial V_d|$ is the volume of a unit sphere in d dimensions, $a_{\Pi_{\mathcal{N}_{\mathbf{x}}}}$ is the autocorrelation function of the indicator function for $\mathcal{N}_{\mathbf{x}}$, i.e. $\Pi_{\mathcal{N}_{\mathbf{x}}}(\mathbf{y}) = 1$ for $\mathbf{y} \in \mathcal{N}_{\mathbf{x}}$ and zero otherwise, and k is a 1D normalized kernel (we use the Gaussian).

The intensity in the whole domain can be simply set as $\hat{\lambda}(\mathbf{x}) = \hat{\lambda}_x$. Similarly, the tensor field given by $\mathbf{M}_x$ can be interpolated or visualized as part of the analysis. However, setting a different PCF for each point in space adds many degrees of freedom, and hence makes qualitative and quantitative analysis, as well as synthesis difficult. Hence, we would like to compress the space of PCF's present in a given distribution.

4.2.3 Spatially Varying Correlations

It has been observed in previous works [Öztireli and Gross, 2012; Heck et al., 2013] that the space of possible PCF's is rather limited, as valid PCF's lie

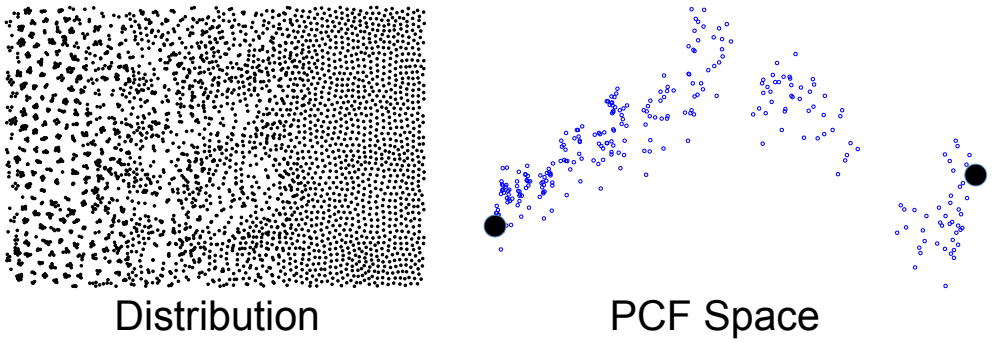

Figure 4.2: *A distribution with adaptive correlations (left), and the space of extracted PCF's (right) projected into a two dimensional subspace via PCA. Our algorithm detects the two indicated PCF's as the dominant ones, and represents all others as linear combinations.*

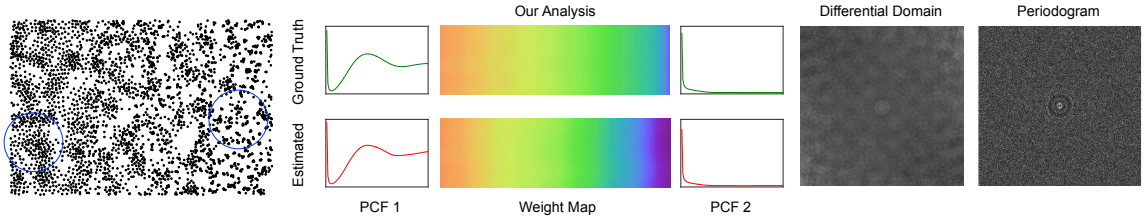

Figure 4.3: *An input distribution (left) and its analysis via our method, differential domain analysis [Wei and Wang, 2011], and periodogram. The patches that correspond to PCF 1 and 2 are marked on the point distribution. The weight map shown is w_1 for PCF 1. We do not show w_2 as it is equal to $1 - w_1$. This is a synthetic example, with ground truth PCF's and weight maps provided. Our analysis successfully recovers the dominant PCF's and weights. Since the previous analysis methods mix different correlations, they cannot provide an informative summary.*

on a subspace of low dimensionality (effectively 2 or 3 dimension). Due to this low dimensionality, all PCF's can be represented as linear combinations of a few basis PCF's. Although Öztireli and Gross [2012] have shown that the subspace of the PCF's is approximately linear, our only assumption for analysis is its low dimensionality such that PCF's can be represented as linear combinations of a few basis PCF's (for synthesis, we will need the stronger linearity assumption, as elaborated on how we do synthesis). We thus would like to summarize the variability in $\mathcal{N}_{\mathbf{x}}$ with a PCF dictionary, and express the rest of the PCF's as linear combinations of the dictionary elements.

In practice, we will have a finite number of neighborhoods $\mathcal{N}_{\mathbf{x}}$ around certain points $\mathbf{c}_k$ in space. These points can be regarded as measurement points for

the computed statistics. The corresponding PCF's $\hat{g}_k(\|\mathbf{h}\|)$ at $\mathbf{c}_k$'s are sampled and stored as vectors $\mathbf{g}_k$.

Given these $\mathbf{g}_k$, we would like to compute L basis PCF vectors $\mathbf{g}_l$ such that all others can be represented as a linear combination of these PCF's. Note that we would like to have a compact representation with an as small as possible L, while tolerating a certain error. The $\mathbf{g}_l$'s should also provide a meaningful summary of the adaptive correlations. We thus do not utilize sparseness based dictionary learning algorithms that generate an over-complete representation. Since there can be a variety of PCF's appearing in a given distribution, a simple clustering algorithm such as k-means clustering will also not give meaningful summaries, as can be observed in Figure 4.2.

Instead, we adopt an approach that is motivated by the structure of the PCF space. Öztireli and Gross [2012] have observed that the main variation in the PCF space is due to the degree of irregularity in the generated distributions, and most variance can be captured with a few components. We thus first perform a PCA on the vectors $\mathbf{g}_k$ and reduce the number of dimensions such that we retain 99 percent of the variance. This typically results in three components, the first capturing most of the variance. We then choose the PCF's that are at the two ends of the line segment formed by the first component as shown in Figure 4.2. If these are already very close, we can conclude that the distribution is stationary. The weights w_{kl} are then computed as described below such that $\mathbf{g}_k = \sum_l w_{kl}\mathbf{g}_l + \mathbf{e}_k$. If $\max_k \|\mathbf{e}_k\| > e_g$ for a threshold e_g, the PCF that is furthest away from the already added PCF's is added. These two steps are performed alternately till the maximum error becomes sufficiently small. For each $\mathbf{g}_k$, the weights are computed by solving the following optimization problem for w_{kl} with quadratic programming:

$$\min \left\| \mathbf{g}_k - \sum_{l=1}^{L} w_{kl}\mathbf{g}_l \right\|^2 \qquad \sum_{l=1}^{L} w_{kl} = 1, \quad w_{kl} \geq 0. \tag{4.2}$$

We illustrate an example of the chosen PCF's, corresponding patches, and weight maps for those PCF's in Figure 4.3. This is a synthetic example generated by our synthesis method we describe in the next section and thus comes with known ground truth PCF's and weight maps. Our analysis accurately figures out the two main PCF's and their interpolation weights.

4.2.4 The Analysis Framework

Our framework thus estimates spatially adaptive density, the basis PCF's $\mathbf{g}_l$, and the corresponding interpolation weights in space. We plot some of

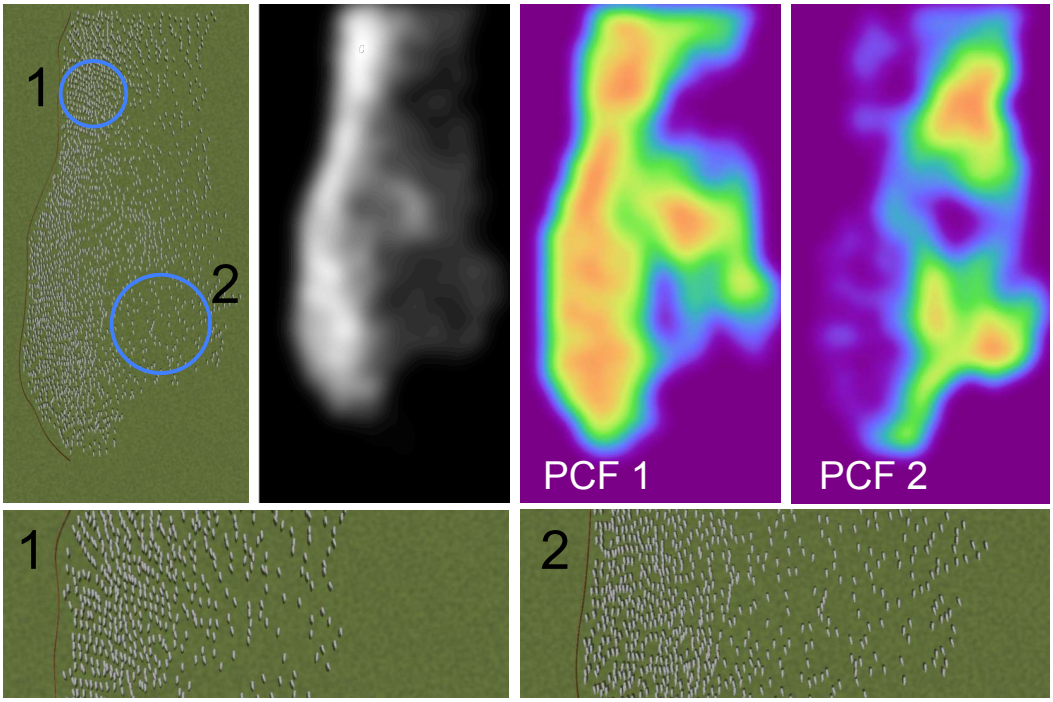

Figure 4.4: *(Top) From left to right: a real-world distribution of sheep, intensity map, and weight maps for the two marked patches. (Bottom) Zoomed regions around the marked patches.*

these diagrams for synthetic examples in Figures 4.1 and 4.3, and for a real example with distributions of locations of sheep acquired from a real scene in Figure 4.4. Note that before computing the PCF's, local anisotropy can be canceled as explained in Appendix A.3, and the PCF's $\mathbf{g}_k$ are normalized with respect to density (Equation 4.1). Thus, we get the same PCF regardless of these degrees of freedom.

Discussion. The density estimate we compute is closely related to previous works that utilize such local modulations of difference vectors or distances [Li et al., 2010; Wei and Wang, 2011]. The mentioned methods compute these directly from the spaces or functions to be estimated, e.g. replace Euclidean distance with geodesic distances, or compute the anisotropy tensors from an image to warp the difference vectors. After local normalization with density and anisotropy, previous analysis methods compute global statistics such as PCF's or periodograms on the whole dataset, leading to blending of different correlations, as discussed above and shown in Figure 4.3. In contrast to these approaches, our analysis technique can separate the important components of correlation models and density apart, and have spatially-adaptive correlations explicitly built in for handling complex point distributions.

Especially for clustering distributions, there is an inherent ambiguity on whether the intensity or PCF is causing the fluctuations in the density of points [Illian et al., 2008]. Hence, given a distribution such as in Figure 4.3, it is hard to disentangle these two different statistics. Our strategy is letting the user assume a certain degree of smoothness for intensity, such as the one we show in Figure 4.4.

In practice, in this work we assume local isotropy, as we work with small neighborhoods and we did not encounter locally strongly anisotropic distributions in practical cases. In Appendix A.3, we describe how strong local anisotropy can be handled with standard methods from point processes.

Parameters. For computing the PCF's we use the same parameters as in a previous work [Öztireli and Gross, 2012]. All parameters are relative to r_{max}, the minimum distance between pairs of points for the maximum packing of points in a domain [Lagae and Dutré, 2008]. We then set $\sigma = 0.25$ for the Gaussian kernel in Equation 4.1, the lower and upper limits for the PCF to $r_a = 0.01\sigma$ and $r_b = 2.5$, respectively, and use a regular sampling of $\|\mathbf{h}\|$ with 100 samples to convert $g_k(\|\mathbf{h}\|)$ to the vectors $\mathbf{g}_k$. The smoothness of the intensity $\lambda(\mathbf{x})$ is a user given parameter, to disambiguate the intensity-correlation duality as described above.

4.3 Synthesis of General Sampling Patterns

We present a general synthesis algorithm following the same model of local stationarity as elaborated on in the last section. We assume that the intensity λ_k and PCF $g_k(r) = g_k(\|\mathbf{h}\|)$ for each neighborhood $\mathcal{N}_k$ are provided or estimated from one or more example distributions with the proposed analysis framework. Then, the main idea of the synthesis algorithm is to generate a new distribution with statistics matching these target statistics.

4.3.1 The Synthesis Algorithm

We assume that we are given a domain and n points to be distributed to match the target characteristics. The intensity function $\lambda(\mathbf{x})$ can be scaled with a constant factor such that its integral is equal to n over the domain. For each neighborhood $\mathcal{N}_k$, the fitting error can then be computed as:

$$E_k(\mathcal{X}_k) = \int_{r_a}^{r_b} \left(g_k^E(r) - g_k(r) \right)^2. \tag{4.3}$$

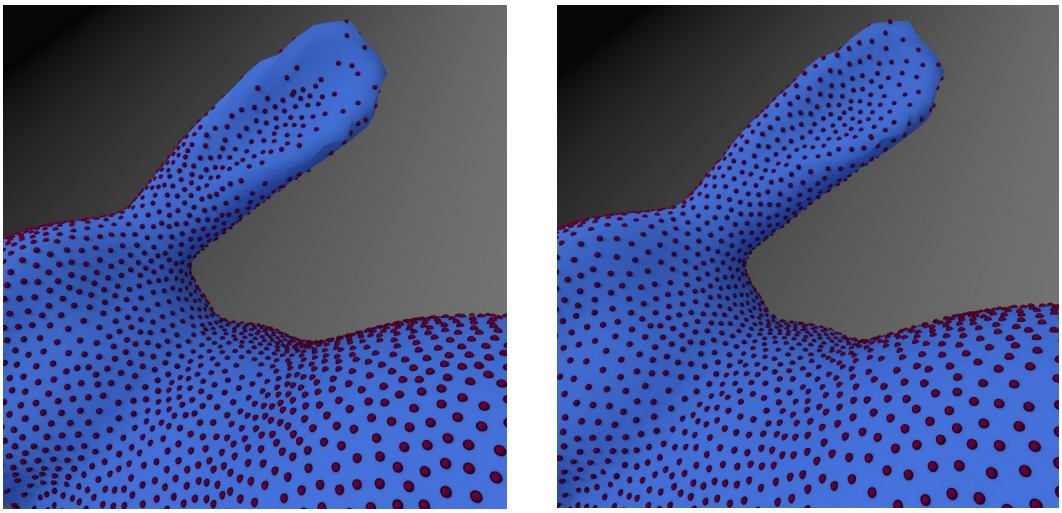

Figure 4.5: *Adaptive neighborhoods (right) can provide more accurate matching of target sample statistics than isotropic ones (left).*

Here, $\mathcal{X}_k$ denotes the set of sampling points within $\mathcal{N}_k$, $g_k^E(r)$ is the estimated PCF from these points with the estimator in Equation 4.1, and $g_k(r)$ is the target PCF to be matched to. The gradient $\frac{\partial}{\partial \mathbf{x}_i} E_k$ is then computed and summed over all neighborhoods to get the final gradient $\Delta_i = \sum_k \frac{\partial}{\partial \mathbf{x}_i} E_k$ for point $\mathbf{x}_i$. We then perform a gradient descent $\mathbf{x}_i = \mathbf{x}_i - \lambda \Delta_i$, where we choose λ with a line search at each iteration. Similar to the previous section, we discretize the PCF's such that the integral in Equation 4.3 turns into a sum (we explain how this can be adapted to handle strong local anisotropy in Appendix A.3).

Size and distribution of neighborhoods We assume that the domain is divided into overlapping spherical neighborhoods $\mathcal{N}_k$ around center points $\mathbf{c}_k$. The volume $|\mathcal{N}_k|$ of each neighborhood is computed such that there are the same number of points in each $\mathcal{N}_k$, by setting $|\mathcal{N}_k| = \alpha n / \lambda_k$ for a constant factor α and the total number of points n. Since we assume that the point process is locally stationary in $\mathcal{N}_k$, the expected total number of points in $\mathcal{N}_k$ is then given by [Illian et al., 2008] $\int_{|\mathcal{N}_k|} \lambda(\mathbf{x}) = |\mathcal{N}_k| \lambda_k = \alpha n$. Note that by fixing the number of points in each neighborhood, we also fix the r_{max} (Section 4.2.4, parameters) when estimating the PCF's.

To get correctly blended characteristics, each neighborhood should also see points belonging to the others. For a neighborhood of radius R_k, we thus retrieve all sample points that fall into a hypersphere of radius $R_k + r_b$ when computing the gradients $\frac{\partial}{\partial \mathbf{x}_i} E_k$ for all $\mathbf{x}_i \in \mathcal{N}_k$. For constant $R_k = R$, the neighborhood centers $\mathbf{c}_k$ lie on a regular grid. The spacing of the grid is set as $T = R/2$.

Adaptive neighborhoods In the case of adaptive λ_k, the neighborhood size and thus R_k changes. Decreasing the spacing, i.e. having neighborhoods such that $T \leq R/2$, does not alter the synthesized distributions, but degrades the performance. Hence, we utilize a conservative greedy non-uniform sampling of $\mathbf{c}_k$ for efficiency. We start from a sparse grid such that $T = \min_k R_k$. Each grid point $\mathbf{c}_k$ is then subdivided, starting from $\mathbf{c}_k$ with the largest λ_k, such that for each one-ring neighbor $T \leq R_k/2$.

Discontinuities in the density function violate our assumption of local smoothness in isotropic neighborhoods. For these cases, it is important to have an adaptive neighborhood that aligns itself along the discontinuity and thus avoids it. Typical examples of such cases are stippling images when the intensity changes abruptly, or geometry sampling when there are sharp features. For such discontinuities, we utilize adaptive neighborhoods computed by confining the neighborhood to one part of the discontinuity. For images, a bilateral filter on intensities, and for geometry on surface normals is first applied to cluster similar pixels/geometry points. Then, any neighborhood that contains different clusters is subdivided along the discontinuity. An example where we apply isotropic and adaptive neighborhoods for geometry sampling with blue noise is shown in Figure 4.5.

Initialization We use a simple initialization strategy with random sampling. Around a randomly chosen neighborhood center $\mathbf{c}_k$, we iteratively pick a random point, and keep this point if all neighborhoods containing this point have not reached the desired number of points. We then discard the ones that already contain enough points, and continue with random sampling around the remaining neighborhood centers.

Non-ergodic processes So far we have considered ergodic distributions where the statistics of the underlying point process can be estimated by observing a single distribution. There exist stationary point processes that are non-ergodic [Illian et al., 2008]. An important example that we encounter in practice is the locally regular distribution, where the points lie on a regular grid with fixed orientation but random global translation. Such a distribution is referred to as uniform or isotropic jittering in the literature [Ramamoorthi et al., 2012; Öztireli, 2016]. The statistics for this case cannot be extracted or matched to by considering just a single distribution, as it will have a constant global translation. In other words, expected values computed over many distributions are not equal to those over a single larger distribution. This is problematic for our synthesis algorihm as the output statistics are extracted and matched for a single distribution, the synthesized distribution. Thus, for locally non-ergodic stationary processes, we need a different synthesis

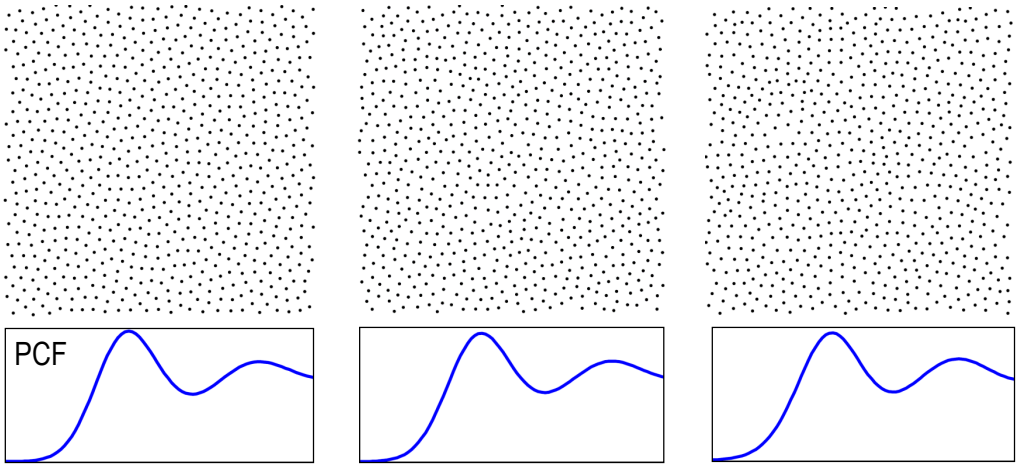

Figure 4.6: *A blue noise pattern [Balzer et al., 2009] (left) is used as the input for our texture synthesis approach (center) and for our PCF based synthesis algorithm (right).*

approach. For these cases, we propose an extension of our discrete texture synthesis algorithm from Chapter 3, as we elaborate in the next section.

4.3.2 Extension of the Discrete Texture Synthesis Algorithm

We propose to extend our discrete texture synthesis algorithm from Chapter 3. For ergodic distributions, this method provides equivalent results to those generated by the synthesis algorithm presented in Section 4.3.1, as the same statistical measures are assumed for both cases. An example can be seen in Figure 4.6. For all non-ergodic synthesis results in this chapter, i.e. where locally regular distributions need to be generated, we employ this method.

Although our original discrete texture synthesis algorithm can synthesize output point sets with general repeated patterns, these patterns are determined by a single example. Hence, it can be used to synthesize a stationary distribution with fixed pair-wise correlations, while the density can vary in the output space by scaling the example. Having an example distribution with spatially-adaptive correlations will not generate adaptive correlations in the output since the neighborhoods from the example are assumed to be repeated throughout the output space. Hence, the main challenge of adaptive synthesis is revising the technique to allow for multiple examples and their combinations.

Adaptive Correlations In order to synthesize distributions with adaptive correlations, we thus extend the *similarity error density* from Equation 3.2 to

consider multiple input functions and weight their importance depending on the location in the output domain. In this case, instead of having a single example function e, we have multiple functions e_p.

For an arbitrary point $\mathbf{x}$ in the output domain, there is a weight $\mathbf{w}_p(\mathbf{x})$ that describes how much the example e_p is influencing that point. Our *weighted similarity error density* then computes a weighted average of the local similarities between the output function and multiple input functions, as

$$S_w(\mathbf{x}) = \sum_p \mathbf{w}_p(\mathbf{x}) S(\mathbf{f}(\mathbf{x}), \mathbf{e}_p(\mathbf{m}_p(\mathbf{x}))) \tag{4.4}$$

where $S(\mathbf{f}(\mathbf{x}), \mathbf{e}_p(\mathbf{m}_p(\mathbf{x})))$ is the similarity between the output function f, and the p-th input function e_p, and $\mathbf{m}_p$ is the matching function from the output domain to the domain of e_p.

In the general synthesis algorithm in Section 4.3.1, we define a PCF at each point $\mathbf{x}$ as a linear combination of basis PCF's. The example distributions that define e_p encode these basis PCF's, and the weights w_p correspond to the computed or given weights for linearly combining the basis PCF's. Hence, instead of having a PCF at each point in the output domain as a linear combination of the basis PCF's, we have the corresponding basis example distributions and their linear combinations given in Equation 4.4.

Adaptive Density and Orientation The original discrete synthesis method accounts for adaptive density and orientation by introducing a scale factor $\mathbf{s}(\mathbf{x})$ and a rotation matrix $\mathbf{R}(\mathbf{x})$ defined at every location in the output domain, which scale and rotate the example domain before matching. After computing the sizes of the neighborhoods $\mathcal{N}_k$ according to the given local density λ_k as described in this chapter, we adjust the scaling field s such that, at each neighborhood, we have approximately the required number of points αn. The rotation for each neighborhood is used to rotate the corresponding examples for that neighborhood before synthesis. We use the same scheme as in Section 4.3.1 for placing the neighborhood center points $\mathbf{c}_k$.

Geometry Sampling and Local Anisotropy The original method allows to define guiding fields in 3D to accordingly orientate the input example and synthesize a distribution on a curved suface.

For small anisotropy factors, the original method allows to synthesize anisotropic distribution by simply scaling the input example. For strong anisotropy, the method can be extended to utilize anisotropic Gaussians.

Limitation The main limitation of this discrete texture synthesis based algorithm results from using actual example distributions rather than extracted

statistics as we do in Section 4.3.1. A well-known limitation of such neighborhood based texture syntehsis approaches is that they cannot handle repetitions at multiple scales, as observed in Chapter 3. For the case of synthesizing point distributions, this means that clustered distributions, where the points in each cluster follow a certain distribution, and the clusters themselves follow another distribution, may not be reliably synthesized. In practice, we have observed that we still get visually accurate results for these cases. However, the statistics deviate from those of the examples slightly.

4.4 Results

We test our analysis and synthesis algorithms for a variety of distributions on Euclidean domains and curved spaces, and illustrate a series of applications for these generalized sets of distributions.

4.4.1 Analysis and Synthesis of Complex Distributions

We illustrate several examples with adaptive density and correlations in Figures 4.1, 4.3, 4.4. The distributions in Figures 4.1 and 4.3 are synthesized with known characteristics, i.e. PCF's and weight maps. Our analysis recovers the ground truth parameters, and our synthesis reproduces local PCF's very accurately. Notice that, while linear transitions in PCF's may not translate into visually linear transitions of distributions, our algorithm synthesizes new distributions with the same visual transitions of the given example distribution. In Figure 4.4, the characteristics of a sheep distribution are extracted from a real-world distribution. It can be observed that they form more regular structures near the fences on the left. We then take these weights and warp the one for PCF 1 to simulate a circular fence in Figure 4.7, and synthesize a distribution. The result accurately reproduces the distributional characteristics of the sheep in accordance with the environment.

A similar analysis and synthesis result is shown in Figure 4.7, second and third rows, for the distribution of trees in a forest. Instead of extracting from existing natural distributions, the characteristics can also be specified by pre-determined rules. In Figure 4.1, the trees are forced to form more regular distributions at lower altitudes, in accordance with real-world observations.

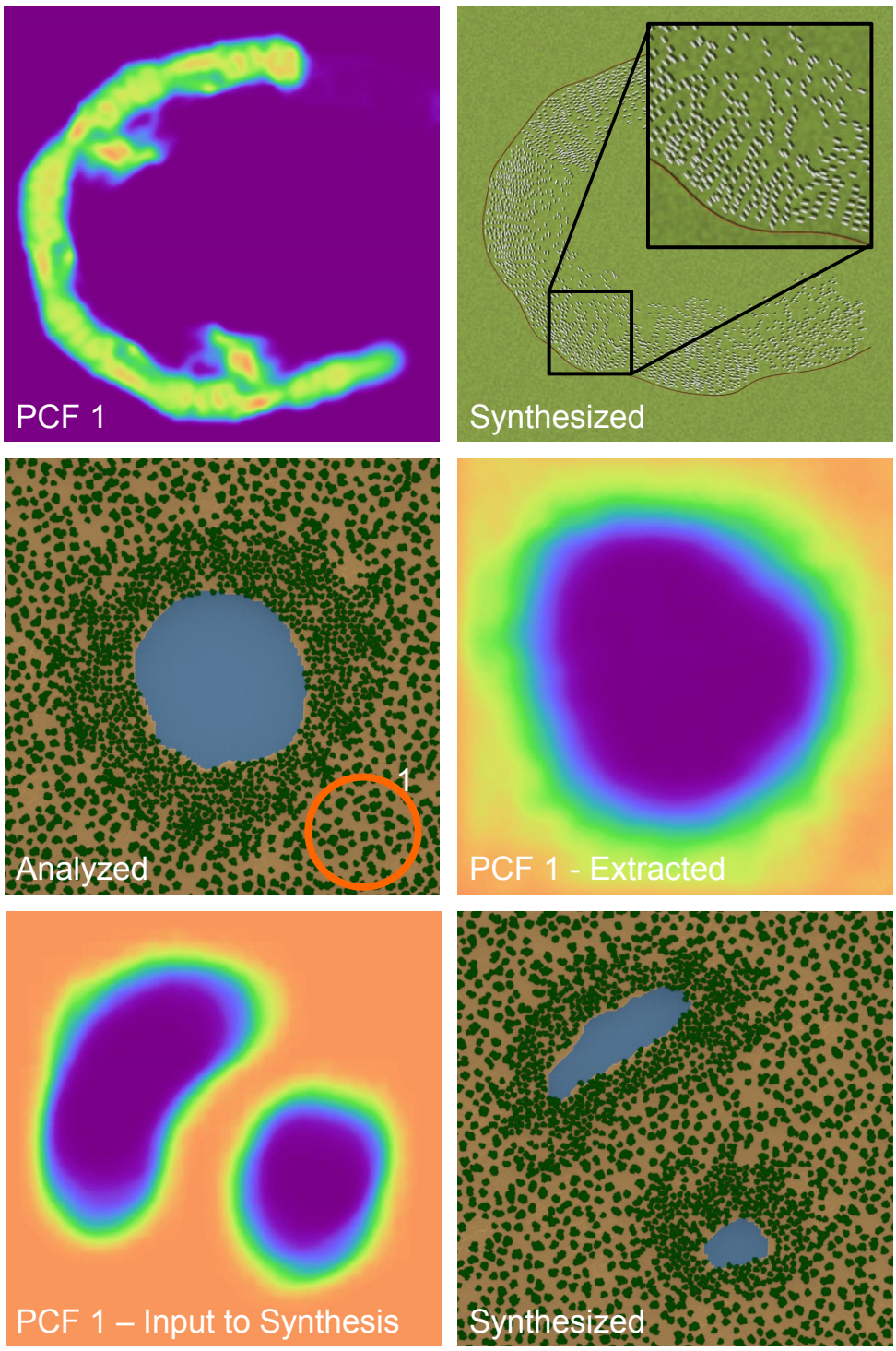

Figure 4.7: *(Top) The extracted PCF's and weight maps from a real-world distribution of sheep (Figure 4.4) are used to synthesize a new distribution with a new drawn weight map for PCF 1. (Bottom) Distribution of trees in a forest is analyzed, and a new forest with a custom weight map for PCF 1 is synthesized.*

4.4.2 Image Sampling and Reconstruction

Being able to control local characteristics of point distributions opens new ways of improving irregular sampling for function reconstruction and representation. We illustrate an example to image plane sampling and reconstruction. Instead of using patterns with anti-aliasing properties such as blue noise, or merely adjusting density or metrics based on image content, we propose to steer both the density and correlations in order to obtain better image reconstructions from irregular samplings.

The main idea is that by distributing sampling points regularly along main image edges, and ensuring that we do not run into aliasing artifacts by a smooth transition to blue noise, we can get sharper and artifact-free reconstructions. We illustrate examples where we compare to a blue noise pattern and the bilateral blue noise sampling of Chen et al. [Chen et al., 2013] in Figure 4.8. We first distribute samples with different methods. For our synthesis algorithm, we extract the edges in an image with Canny edge detector, and smooth them to obtain the weighting map (Figure 4.8, left, insets) for the regular distribution. This is used to interpolate between the regular and blue noise distributions. We align the regular distributions with the edges by computing the local orientation of the closest edge.

Each sample carries a color value. These samples are used to reconstruct an image. For the reconstruction, we use isotropic Gaussian kernels (Figure 4.8, top) and iterative bilateral filtering (Figure 4.8, bottom), when comparing to blue noise and Chen et al. [Chen et al., 2013], respectively. Combination of regular and blue noise sampling leads to significantly improved results, especially around the edges, for both cases.

4.4.3 Image and Video Stippling

Such rules can also be defined for image stippling to generate alternative stippling styles. In Figure 4.9, we illustrate how combinations of blue noise and regular sampling can be utilized to generate stippled images with a novel style. For these images, the intensity at a pixel determines the density of the points as in previous works (e.g. [Fattal, 2011; Zhou et al., 2012]), and a smoothed edge map is used as the weight map for interpolating between a blue noise and regular distribution. Similarly to our image sampling results, we first detect the main edges of the image, and set the weight associated with the regular distribution for a neighborhood $\mathcal{N}_k$ to be inversely proportional to the distance between $\mathbf{c}_k$ and the closest edge, such that as we move away from the edges, we get a more blue noise type distribution. The smooth transition

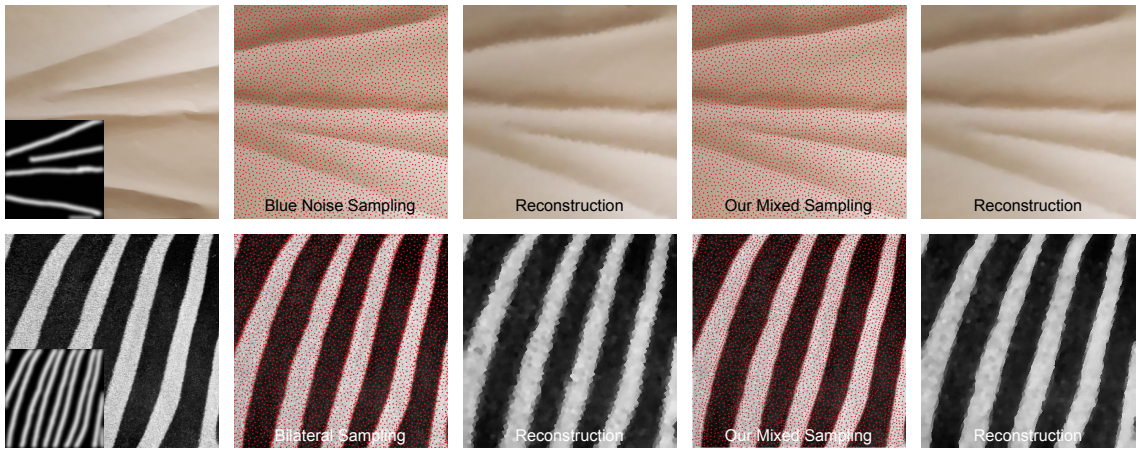

Figure 4.8: *Input images (with extracted and smoothed edges used for determining the weighting for regular sampling in our synthesis shown in the insets), sampling results with blue noise, bilateral blue noise [Chen et al., 2013], and our technique, and the corresponding recontructions with isotropic Gaussian kernels (top) and iterative bilateral filtering (bottom).*

between the two correlation models result in artifact-free distributions. This is in contrast with distributions generated by simply defining separate regions for blue noise and regular sampling as illustrated in Figure 4.9, middle. For this case, additional structures appear around the edges, leading to visually unpleasant results.

We can develop and extend other stippling styles as well thanks to the generality of the distributions we can handle. In Figure 4.10, we show a different stippling style, which is closer to the one proposed by Kim et al. [Kim et al., 2008]. In this style, we extract stylistic smooth edge maps [Kang et al., 2007] (Figure 4.10 top, middle), and compute the weight map for regular distribution (Figure 4.10 top, right) by dilating and smoothing them with a Gaussian. To achieve a more regular looking style, we generate larger regular regions compared to the previous results. We use the PCF of a blue noise pattern [Balzer et al., 2009] as the second PCF, and blend between this and the regular distribution using the computed weight map. The edge tangent flow [Kang et al., 2007] (a smooth vector flow describing the salient edge tangent direction in the image) is used to orient the regular distribution along the extracted edges. The density is constant and the sizes of the dots are changed depending on the intensity of the image, as in the work by Kim et al. [Kim et al., 2008]. Finally, the extracted lines are shown together with the points.

Interpolating between oriented regular distribution close to the edges and blue noise elsewhere allows us to avoid artifacts in regions were multiple

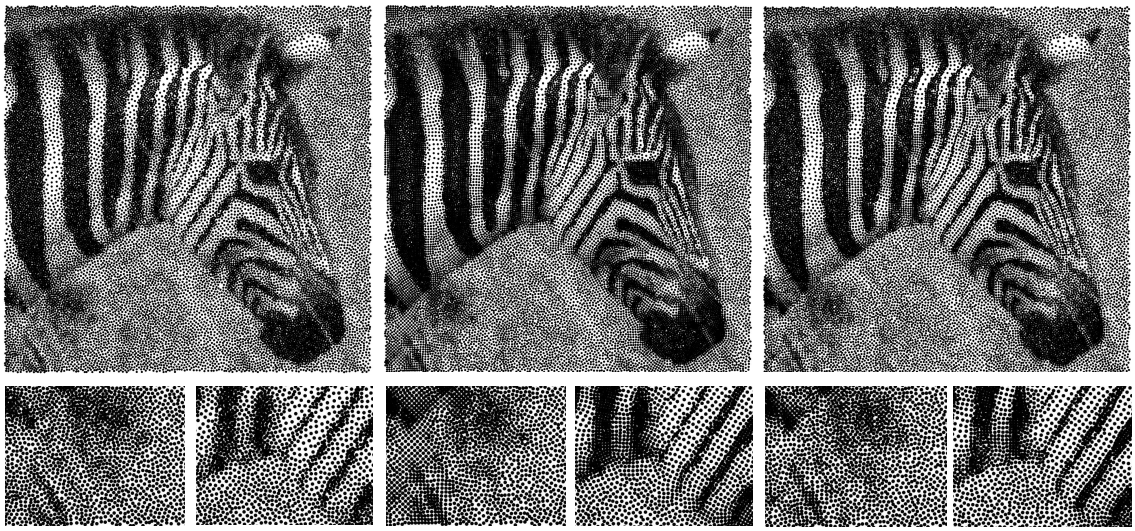

Figure 4.9: *From left to right: stippling with blue noise sampling, regular and blue noise sampling with a sharp transition, and regular and blue noise sampling with smooth transitions along the edges. The combination of blue noise and regular sampling with smooth transitions avoid artifacts, while providing a novel sharper stippling style, as illustrated in the insets below. This figure is best viewed on a computer screen, please zoom-in to see the details clearly.*

lines with different orientations intersect. Even though Kim et al. [Kim et al., 2008] propose additional controls to handle such cases, their method, purely based on regular distributions, can result in structured artifacts, as illustrated in Figure 4.11, left. By placing blue noise at the intersections, our method replaces these structures with blue noise instead (Figure 4.11, right).

Video stippling The neighborhoods in our synthesis algorithm can also be extended in time to get spatio-temporally smooth sampling. For example, in addition to blending a regular and a blue noise distribution, we set the PCF of a neighborhood at frame t to be a combination of its PCF and that of the same neighborhood in the previous frame. The weight assigned to the previous frame determines the trade-off between temporal smoothness and fidelity to the current frame. We found that a weighted average of PCF's where the PCF of the previous frame is given a weight of 0.25 works well in practice, i.e. $g = 0.25 g_{t-1} + 0.75 g_t$.

4.4.4 Geometry Sampling

Our synthesis algorithm extends to curved surfaces. Although recent works [Jiang et al., 2015] explore adapting the spectrum of blue noise distri-

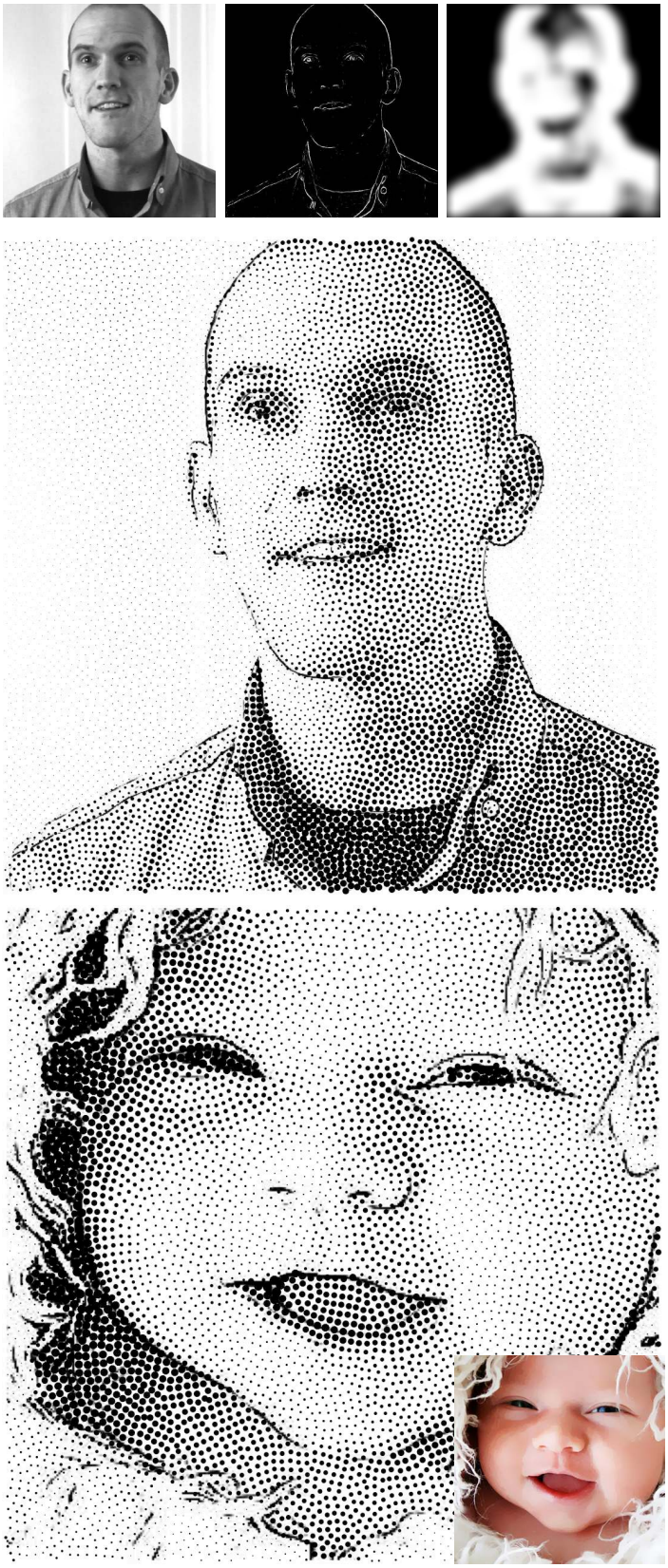

Figure 4.10: *Stippling with a different style. On the top row, we also show the extracted edges, and the corresponding smoothed map for the regular distribution.*

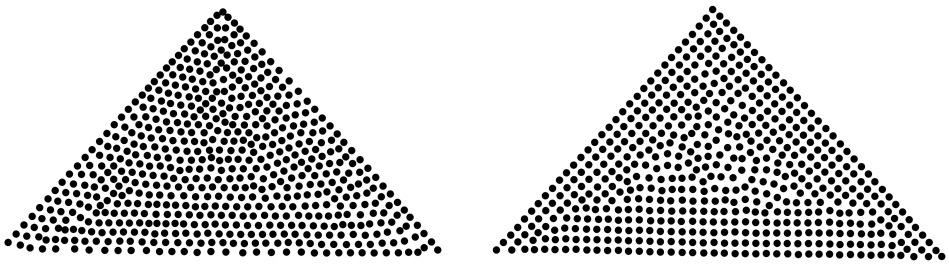

Figure 4.11: *Stippling a triangle using the method of Kim et al. [Kim et al., 2008] (left), and our style (right).*

butions on surfaces, to the best of our knowledge, there does not exist any algorithm to generate distributions with general characteristics on curved domains. We illustrate different noise patterns on surfaces in Figure 4.12. Similar to the planar case, our technique allows to generate smooth transitions between different patterns on surfaces as well, as shown for a combination of blue and green noise in Figures 4.12 and 4.1. We use a simple approximation of surface geodesics [Bowers et al., 2010] in the synthesis algorithm. For every local patch, the distances are computed on the tangent plane of the surface, and the points are projected back to the surface after being moved, similar to previous techniques for blue noise sampling on surfaces [Öztireli et al., 2010].

4.4.5 Performance

Most of the shown 2D distributions contain about 5000 samples and were generated with about 200 neighborhood centers $\mathbf{c}_k$. The stippling images with adaptive density (zebra) contain about 25000 samples and were generated with 10000 neighborhoods. The geometry sampling examples have up to 15000 samples and were generated with 2000 neighborhoods. To compute the gradient of a sample, the PCF's of all the neighborhoods in which the sample is included are taken into consideration. Thus, the performance is mostly influenced by the radii R_k and the spacing of neighborhood centers $\mathbf{c}_k$. On average, 25 iterations were needed until convergence. Our unoptimized, single core implementation takes up to one minute to complete one iteration on a PC with an Intel i7-3770K CPU. The bottleneck of the algorithm is the update of the PCF's of all the neighborhoods containing a point, after moving it, which could be optimized by computing some of the gaussians only once to update multiple PCF's. Furthermore, our synthesis algorithm can be significantly speeded up by moving multiple samples simultaneously, as they influence only their local neighborhood.

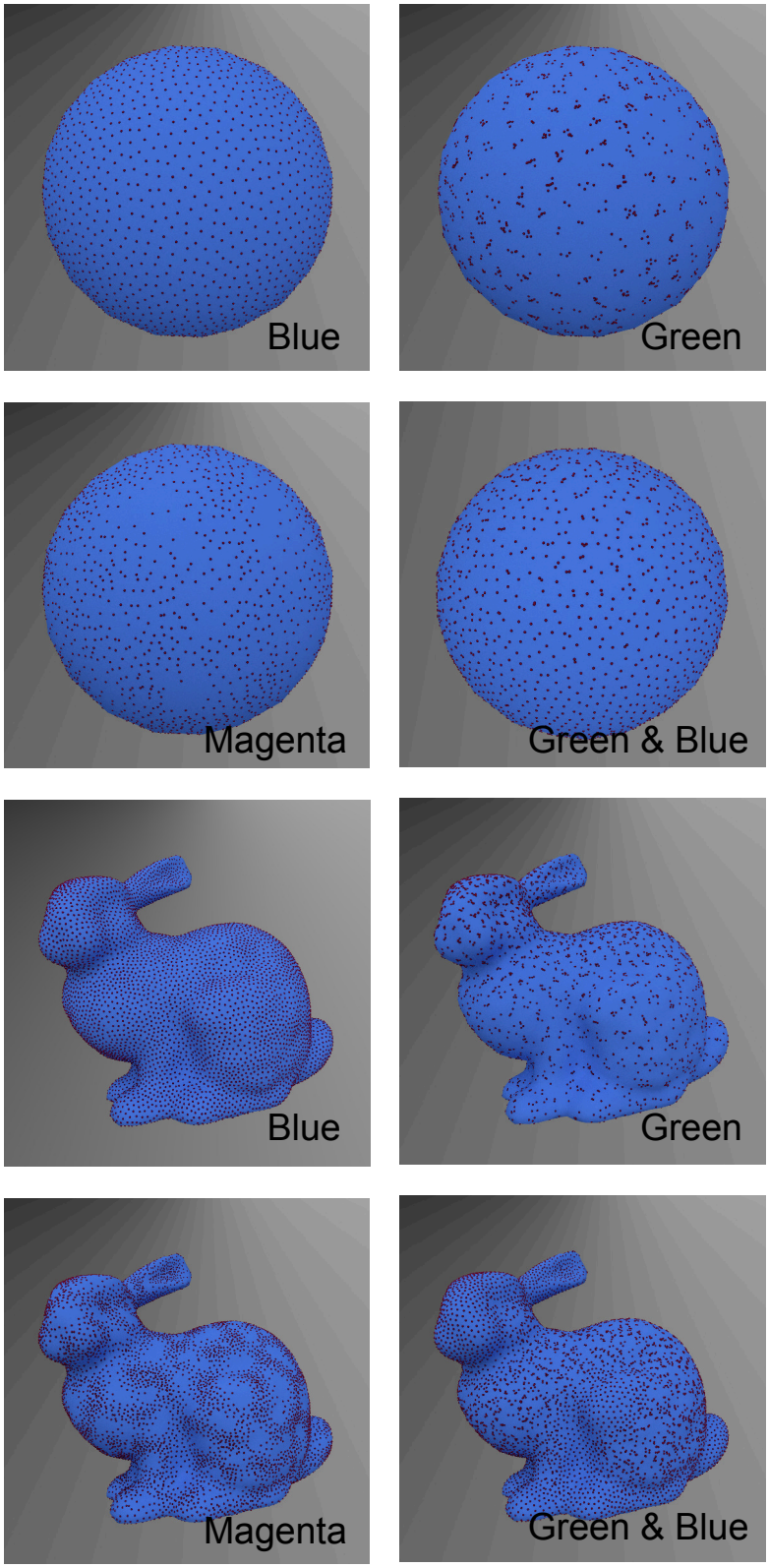

Figure 4.12: *Our synthesis algorithm can generate patterns with controllable characteristics and transitions on surfaces.*

4.4.6 Limitations

Theoretically, it is possible that there exist point patterns that violate the local stationarity assumption, with dense discontinuities of the first and second order correlations throughout the space, although we have not encountered such distributions in practice. As we elaborate in Appendix A.2, a more general analysis is possible, but will lead to major difficulties since it requires multiple instances of the same distribution.

When analyzing a given distribution, it is in general an ill-posed problem to determine the intensity, anisotropy, and second order correlations without certain assumptions. Our choices of a smooth intensity, a simple anisotropy model, and interpolated PCF's provide a set of such priors. Note, however, that the forward problem of synthesis does not suffer from this limitation, and we do not assume smooth density in that case.

Finally, our synthesis algorithm shares some of the limitations of previous PCF based fitting algorithms [Öztireli and Gross, 2012], as it simplifies to those for the case of a globally constant PCF. In particular, we cannot guarantee a minimum distance between point locations when synthesizing blue noise distributions, as the contribution of a pair of points can be blurred out by many others in the PCF. This can lead to small fitting errors.

4.5 Discussion

We introduced novel analysis and synthesis techniques for point distributions with adaptive density and correlations. The analysis framework provides an informative view of complex distributions with extracted maps capturing different distributional characteristics. Based on the same characteristics, the synthesis algorithm combines adaptive density and correlations and extends to general domains.

Sampling and anti-aliasing The proposed general framework offers new possibilities for accurate representation or anti-aliasing of images, instead of the generic patterns with blue noise properties. Exploiting the redundancy of image patches, measures based on the local content such as edges, textures, and other structures can be computed and utilized to steer the synthesis algorithm for general images.

Rendering Rendering involves computing integrals of complex functions. Traditionally, density adaptation of the samples via importance sampling has been the norm to improve the rendering quality by reducing noise

while avoiding artifacts due to aliasing. Recent works [Durand, 2011; Ramamoorthi et al., 2012; Subr and Kautz, 2013; Subr et al., 2014] have shown that adapting density and correlations simultaneously can significantly improve the rendering results. However, the researchers have been limited by the analysis techniques [Subr and Kautz, 2013] and pattern generation algorithms. Our techniques can be instrumental in developing new sampling methods and understanding existing patterns utilized in rendering such as for distributed ray tracing. As an example, combining adaptive correlations with the recent works that explore correlation models for rendering, e.g. [Öztireli, 2016], can be an exciting future direction.

Understanding natural phenomena Distributions and patterns in nature are inherently spatially adaptive due to environmental factors. We have illustrated that (e.g. Figure 4.4), it might not be possible to explain natural distributions by simply adaptive density. Our framework can be utilized to understand a spectrum of distributions ranging from surface details [Yan et al., 2014], facial features [Beeler et al., 2012], fluid particles [Öztireli and Gross, 2012], to crowds [Ju et al., 2010].

Geometry reconstruction and remeshing Reconstruction of geometry from point samples, and remeshing surfaces for rendering or simulations critically depend on the quality of the sampling patterns, given by both the density and the pair-wise correlations [Öztireli et al., 2010; Jiang et al., 2015]. Our synthesis algorithm extends full correlation control to surfaces, unlocking a significantly extended set of sampling patterns for geometry sampling. Similarly to our image reconstruction application, sharp features in surface reconstruction could be better preserved by aligning regular samples along them. For remeshing, a regular sampling could be ideal to generate quads, and a blue noise distribution could be adopted for transitions.

Marked and space-time processes We have illustrated a simple application of spatio-temporal sampling for stippling videos with a trivial extension. A more in-depth analysis with space-time processes can be used to principally extend our techniques. Similarly, extensions of the framework to multi-class sampling can be developed with the theory of marked processes [Illian et al., 2008].

CHAPTER 5

Consolidation of Point Clouds with Convolutional Neural Networks

While the sampling applications shown in Chapter 3 and Chapter 4 consist of generating new distinct point clouds to represent the geometry, consolidating means improving an existing raw point cloud, in order to better represent its underlying structure. With the widespread use of 3D acquisition devices, there is an increasing need of consolidating captured noisy and sparse point cloud data for accurate representation. There are numerous algorithms that rely on a variety of assumptions such as local smoothness to tackle this ill-posed problem. However, such priors lead to loss of important features and geometric detail. Instead, in this chapter we propose a novel data-driven approach for point cloud consolidation. Rather than relying on a small set of examples to drive the generated sampling (as in the methods in the previous chapters), we train a convolutional neural network based technique on a large dataset of geometry patches, to adapt the output to the varying structures in the input raw data. Our method takes a sparse and noisy point cloud as input, and produces a dense point cloud accurately representing the underlying surface by resolving ambiguities in geometry. The resulting point set can then be used to reconstruct accurate manifold surfaces and estimate surface properties. To achieve this, we propose a generative neural network architecture that can input and output point clouds, unlocking a powerful set of tools from the deep learning literature. We use this architecture to apply convolutional neural networks to local patches of geometry for high quality and efficient point cloud consolidation. This results in significantly

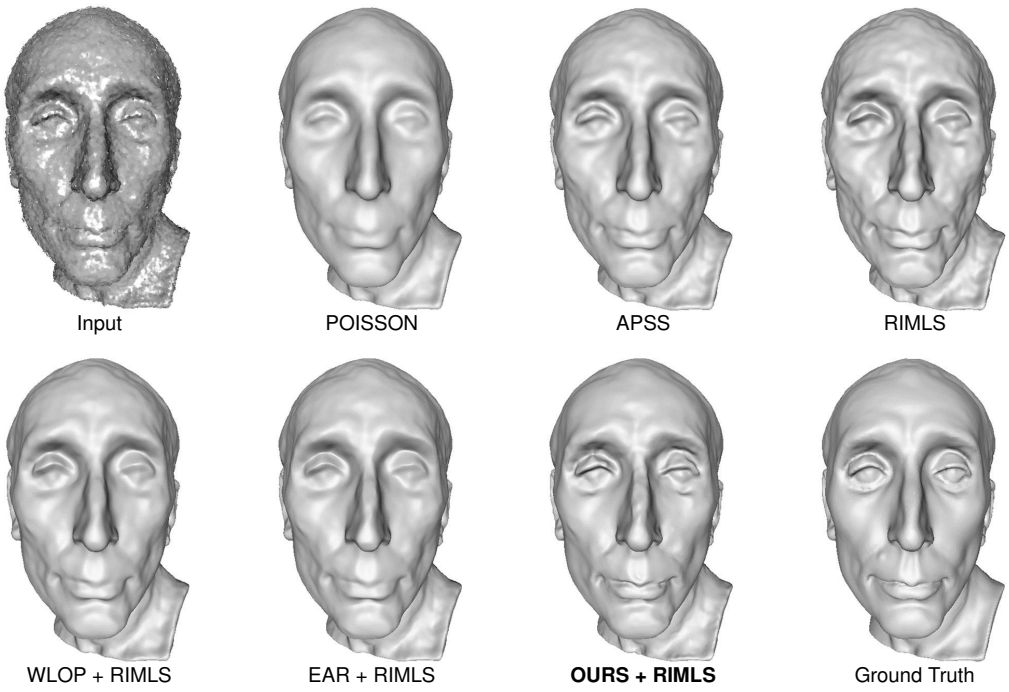

Figure 5.1: *Surface reconstruction from a noisy and sparse point cloud is an ill-posed problem with infinitely many possible reconstructed surfaces. Our technique consolidates an input point cloud by learning local maps from input to output geometry patches to enhance reconstructions with accurate geometric features and details. This leads to significant improvements for the resulting surfaces.*

more accurate surfaces, as we illustrate with a diversity of examples and comparisons to the state-of-the-art.

5.1 Introduction

Capturing 3D geometries is becoming commonplace thanks to the abundance of affordable and lightweight sensors and advancing algorithms. The captured geometries can then be used for various applications ranging from 3D printing to photography. A main challenge for 3D capture systems, however, is that noise and sparseness in point cloud data typically obscure important geometric features and details. Recovering those details can be very difficult or impossible for many cases.

Given a noisy and sparse point cloud depicting an object boundary, i.e. surface, it is an ill-posed problem to recover such geometric details: there can be infinitely many different geometries that would result in the same sparse and noisy set of sample points. To regularize the problem, we thus need

prior beliefs on the global or local structure of the geometry to be reconstructed [Berger et al., 2017]. For resolving fine features and details, most methods rely on local priors such as locally piece-wise smooth surfaces with sharp features [Öztireli et al., 2009; Huang et al., 2013]. This has led to many successful *consolidation*, i.e. synthesizing a new point set that accurately samples the underlying surface, and surface reconstruction algorithms.

Although these techniques generate plausible surfaces, they cannot recover elaborate geometric features if the artifacts in point cloud data become substantial or the priors do not hold. It is in general a very challenging problem to resolve geometric features and up-sample a point cloud especially when only the raw point cloud without further attributes such as surface normals are provided [Wu et al., 2015a].

In this chapter, we propose a data-driven approach to recover surface features and details by building on the recent very successful class of convolutional neural network (CNN) based deep learning methods. CNN-s have shown exceptional performance for many image processing problems, and are more and more used also for generative tasks where an input image is transformed into a new image with desired properties [Xu et al., 2015; Isola et al., 2016; Yan et al., 2016; Gharbi et al., 2017]. However, modern CNN based architectures require a regular sampling of data, and thus extending these techniques to unorganized point clouds is non-trivial [Qi et al., 2017a], and so far could only be used for coarse shape completion on voxel grids of relatively low resolution [Han et al., 2017].

We tackle this by exploiting the structure of our problem: the geometric features we target are encoded in local regions, which can be individually parametrized. Our method jointly learns local parametrizations and the locally fitted surfaces. We achieve this by developing a new neural network based generative architecture that can consume and output point clouds. This architecture provides an end-to-end approach where an input raw point cloud is used to generate a new very dense point cloud that accurately samples the underlying surface. We show that this leads to substantial improvements in terms of accuracy of the final surface representations (Figure 5.1). In summary, in this chapter we have the following main contributions:

- The first deep learning method for local point cloud processing with a fully differentiable architecture that we call PointProNets. A key component in this architecture is a differentiable points projection layer for converting unordered points to regularly sampled height maps. Although we use the architecture for consolidation, it can also be used for revising further point cloud processing tasks

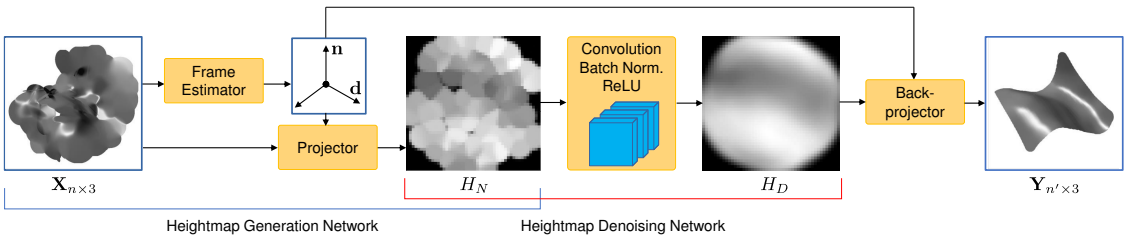

Figure 5.2: *The network architecture. Each patch* $\mathbf{X}$ *of an input point cloud is processed with this architecture to generate the consolidated point set stored in* $\mathbf{Y}$*. Each component is differentiable and hence allows for end-to-end training.*

- An end-to-end data-driven algorithm for consolidation of unorganized point clouds that leads to very accurate surface representations, with significant quantitative and visual improvements over the previous methods.

5.2 Algorithm Overview and Training Data Generation

5.2.1 Overview

The main idea of our technique is to learn a local mapping that transforms each set of points extracted from a local patch of the input point cloud to its consolidated version, where the output points sample the underlying surface very accurately and densely. The union of all such local output sets give us the final output point cloud. We define a patch as the set of points included in a local neighborhood. In particular, we represent a patch of geometry around a point as an oriented 2D heightmap that stores distances to the sample points in the neighborhood along a given direction. This 2D representation of local patches makes it possible to exploit the strengths of deep learning architectures for image denoising and super-resolution, and extend them to the task of processing unstructured 3D points.

In order to learn the mapping from a noisy patch of points to its consolidated version, we designed a new neural network architecture composed of fully differentiable components, as shown in Figure 5.2. The first component, *Heightmap Generation Network (HGN)* receives the matrix $\mathbf{X}_{n\times 3}$ that stores the x, y, z-coordinates of n input noisy points in the patch, and generates a noisy heightmap image H_N of resolution $k \times k$. In particular, it first learns a local coordinate frame for projection, projects the points onto the correponding image plane with a projection module, and resamples the resulting heightmap to obtain the regularly sampled image H_N. The second component, *Heightmap*

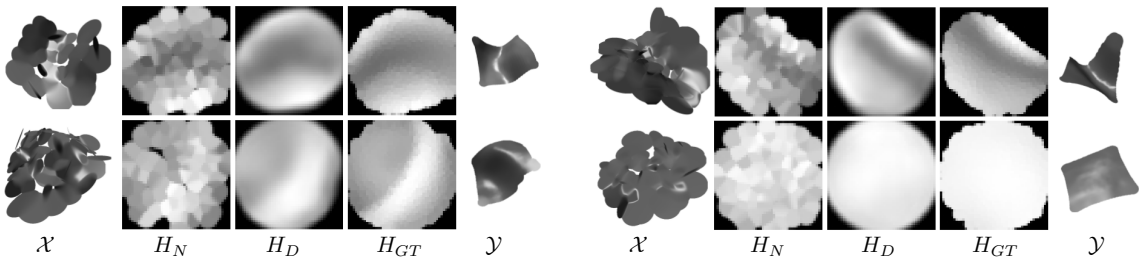

Figure 5.3: *Given an input point cloud patch $\mathcal{X}$ stored as raw point locations in the matrix $\mathbf{X}$, our network projects and resamples the geometry to convert it into the image H_N, and processes this image to produce H_D (here we also show ground truth H_{GT}'s for reference). This is finally back-projected to get the consolidated point set $\mathcal{Y}$.*

Denoising Network (HDN), uses image convolutions to transform the noisy heightmap H_N into a denoised version H_D. Finally, by transforming the pixel coordinates of H_D into point locations according to the learned image plane parameters and the stored distance values, the consolidated patch is generated and stored as a list of n' point coordinates $\mathbf{Y}_{n' \times 3}$, $n \leq n' \leq k^2$. In addition to learning the positions of the consolidated points, we propose a simple extension of our network architecture that allows us to learn consolidating their normals as well, if noisy normals of the input $\mathbf{X}$ are supplied or pre-computed.

5.2.2 Training Data Generation

We start with a set of pairs of an input patch, and the corresponding ground truth output patch. These are cut out from input and ground truth output point clouds in spherical neighborhoods of radius r. For each pair, we thus have a set $\mathcal{X}$ of noisy and sparse points, and the corresponding denser set $\mathcal{Y}_{GT}$ of consolidated, i.e. denoised and up-sampled, points. We then extract a ground truth heightmap H_{GT} from $\mathcal{Y}_{GT}$. At training time, the aim of the network is to produce a denoised heightmap H_D that is as similar as possible to H_{GT}, starting from the input set $\mathcal{X}$.

Ground Truth Heightmap Generation The consolidated patch $\mathcal{Y}_{GT}$ is not directly fed to the network, but transformed to a 2D representation: we aim at extracting a ground truth heightmap H_{GT} which best encodes, in a 2D image, the 3D geometry. We thus want to find a normalized vector $\mathbf{n}_{GT}$ of a proper image plane positioned at an offset $-r\mathbf{n}_{GT}$ (to avoid negative distances).

Since a heightmap can represent only one layer of geometry, we would like to have the least amount of points from different depth levels projected onto the same image pixel and thus averaged. In practice, we set the vector $\mathbf{n}_{GT}$ as the average of the normals of the points in the consolidated set $\mathcal{Y}_{GT}$. Due to the high density of $\mathcal{Y}_{GT}$, its uniform sampling, and lack of noise, we found this average is robust for capturing local geometries for the patch sizes we consider. Given the image frame defined by $\mathbf{n}_{GT}$, and an orthogonal vector $\mathbf{d}_{GT}$, the consolidated points are projected onto the plane orthogonal to $\mathbf{n}_{GT}$, and transformed into image coordinates. The distances between the original points and the projected ones are interpolated with gaussians to produce a resampled heightmap stored in H_{GT}. The same heightmap generation procedure is performed in a custom module within our network in the HGN component. We thus refer to Section 5.3.1 for a more detailed explanation of the operations.

Data Augmentation Note that the orientation of $\mathbf{n}_{GT}$ is ambiguous: both $\mathbf{n}_{GT}$ and $-\mathbf{n}_{GT}$ would produce a valid heightmap, even though the resulting images can be substantially different. We thus choose the sign of $\mathbf{n}_{GT}$ randomly for every patch, in order to ensure that we feed the network with varied data. The vector $\mathbf{d}_{GT}$ defines rotation of heightmaps on the image plane. We choose a random $\mathbf{d}_{GT}$ orthogonal to $\mathbf{n}_{GT}$ to make the learned representation invariant to this degree of freedom. When feeding data to the network during training, pairs $(\mathcal{X}, \mathcal{Y}_{GT})$ are randomly extracted from the training point clouds by positioning centers of the neighborhoods at random points in input point clouds. We thus get a dense coverage of each geometry in the database. Finally, we further augment the patch pairs dataset by random resampling of the input point clouds, getting an arbitrary sampling rate for each $\mathcal{X}$. The number of points can then be matched to n by random downsampling or replication of points in $\mathcal{X}$, as the network expects a fixed-size input matrix $\mathbf{X}$.

5.3 Network Architecture

Given the training data consisting of pairs $(\mathcal{X}, \mathcal{Y}_{GT})$ that are transformed into $(\mathbf{X}, H_{GT})$ as described above, we would like to design an architecture that can be trained with these pairs at training time, and produce consolidated output points $\mathcal{Y}$ for an arbitrary $\mathcal{X}$ at testing time. In this section, we elaborate on the main components of our network (Figure 5.2) in more detail, and explain how the output of the network (used in a feed forward manner) serves to produce the final consolidated point cloud.

5.3.1 Heightmap Generation Network

The goal of our first component, *Heightmap Generation Network (HGN)* in Figure 5.2, is to estimate an image plane orientation, and to produce a corresponding noisy heightmap by projecting the points stored in $\mathbf{X}$. The component is thus divided in two parts: first, a vector $\mathbf{n}$ and an orthogonal direction $\mathbf{d}$ are estimated from the input points in $\mathcal{X}$, then, the input points are projected to the image plane, generating a noisy heightmap image H_N.

Frame Estimator In order to estimate $\mathbf{n}$, the component *Frame Estimator* (Figure 5.2) needs to deal with the unordered structure of the input point set given in $\mathbf{X}$. To tackle this problem, we utilize the idea of using a symmetric function with respect to ordering of points $\mathbf{X}$ with a single max pooling layer from a very recent work [Qi et al., 2017a], and use it to predict $\mathbf{n}$. In the original work, the points are fed as input, points features are produced with fully connected layers and aggregated by max pooling, and a final fully connected layer produces a global descriptor, which is then used for classification or segmentation. In our method, we adopt the same architecture but modify the output of the final fully connected layers to produce the 3D vector $\mathbf{n}$. Additionally, it would be beneficial that the learned representation is invariant to translations and rotations of $\mathbf{X}$. We achieve this by centering the points in $\mathbf{X}$ by subtracting the patch center, and by feeding patches with random rotations at training time.

As elaborated on in Section 5.2.2, due to the ambiguity of the sign of $\mathbf{n}_{GT}$, our dataset contains patches of either orientation. Even without this augmentation, we found out that there can be many similar patches with similar $\mathbf{n}_{GT}$ but with opposite signs. The frame estimator then typically learns to estimate an average, which significantly distorts the learned $\mathbf{n}$ and thus heightmaps. To avoid this averaging, at training time, we snap the orientation of $\mathbf{n}$ to that of $\mathbf{n}_{GT}$ by setting $\mathbf{n} \leftarrow \mathbf{n}(\mathbf{n}^T\mathbf{n}_{GT})$ and normalizing. This ensures that the network is forced to learn the direction of $\mathbf{n}$, and choose either of the two orientations, and not their average. Note that this snapping component is not present at testing time, where the orientation of $\mathbf{n}$ is irrelevant for generating the final consolidated patch. Similarly, at training time, the direction vector $\mathbf{d}$, is kept as close as possible to $\mathbf{d}_{GT}$ and orthogonal to $\mathbf{n}$ at each iteration by setting $\mathbf{d} \leftarrow \mathbf{d}_{GT} - (\mathbf{d}_{GT}^T\mathbf{n})\mathbf{n}$ and normalizing in the component, to ensure rotations on the plane are not averaged. At testing, a random $\mathbf{d}$ orthogonal to $\mathbf{n}$ is sufficient as the learned representation is invariant to rotations on the plane.

Projector The second part of HGN takes the vectors $\mathbf{n}$, $\mathbf{d}$, and the input point set in the form of the matrix $\mathbf{X}$, and renders a 2D heightmap H_N regularly sampled at pixel coordinates. The projector component first projects the 3D points onto the image plane given by the vectors $\mathbf{n}$ and $\mathbf{d}$, and positioned at an offset of $-r$, to avoid negative distances. Hence, for each point $\mathbf{x} \in \mathcal{X}$ (i.e. each row of $\mathbf{X}$), we define

$$\mathbf{p} = \mathbf{x} - (\mathbf{x}^T\mathbf{n} + r)\mathbf{n}, \tag{5.1}$$

as the projected position of $\mathbf{x}$. For each projected point $\mathbf{p}$, we also store the distance $\|\mathbf{x} - \mathbf{p}\|$. The projected points are then transformed into image coordinates as

$$\mathbf{i} = \frac{k}{2r}\begin{bmatrix}\mathbf{p}^T\mathbf{d} + r & \mathbf{p}^T(\mathbf{n}\times\mathbf{d})/\|(\mathbf{n}\times\mathbf{d})\| + r\end{bmatrix}^T, \tag{5.2}$$

where H_N is a $k \times k$ image. Notice that r is the same radius of the neighborhood defined previously, and is added to avoid having negative values in the image coordinates. We thus get the image coordinates $\mathbf{i}$ and the corresponding distance values $D(\mathbf{i}) = \|\mathbf{x} - \mathbf{p}\|$. The heightmap image H_N is then generated by interpolating the distances $D(\mathbf{i})$ on the image plane with Gaussian interpolation at pixel centers.

We use a Gaussian with a cutoff such that for a given pixel center $\mathbf{c}$ in image coordinates, only the points $\mathbf{i}$ given by $\mathcal{N} = \{\mathbf{i} \mid \|\mathbf{c} - \mathbf{i}\| < \delta\}$ need to be considered. The value at $\mathbf{c}$ is then given by

$$H_N(\mathbf{c}) = \begin{cases}\frac{1}{W(\mathbf{c})}\sum_{\mathbf{i}\in\mathcal{N}(\mathbf{c})} g(\mathbf{c},\mathbf{i})D(\mathbf{i}), & \mathcal{N}(\mathbf{c}) \neq \emptyset \\ 0, & \mathcal{N}(\mathbf{c}) = \emptyset,\end{cases} \tag{5.3}$$

where $W(\mathbf{c}) = \sum_{\mathbf{i}\in\mathcal{N}(\mathbf{c})} g(\mathbf{c},\mathbf{i})$, and $g(\mathbf{c},\mathbf{i}) = e^{-\frac{\|\mathbf{c}-\mathbf{i}\|^2}{\sigma^2}}$. We show some examples of generated H_N at testing time in Figure 5.3. All operations of the projection module are differentiable with respect to the inputs, thus the gradients can be back-propagated through the network.

5.3.2 Heightmap Denoising Network

Our second network component, *Heightmap Denoising Network (HDN)* in Figure 5.2, takes the noisy heightmap H_N as input, and generates a denoised version H_D as its output. As this is a mapping between regular images, many previous methods from the image processing literature can be utilized. CNN-based architectures have been successfully adopted for image denoising and

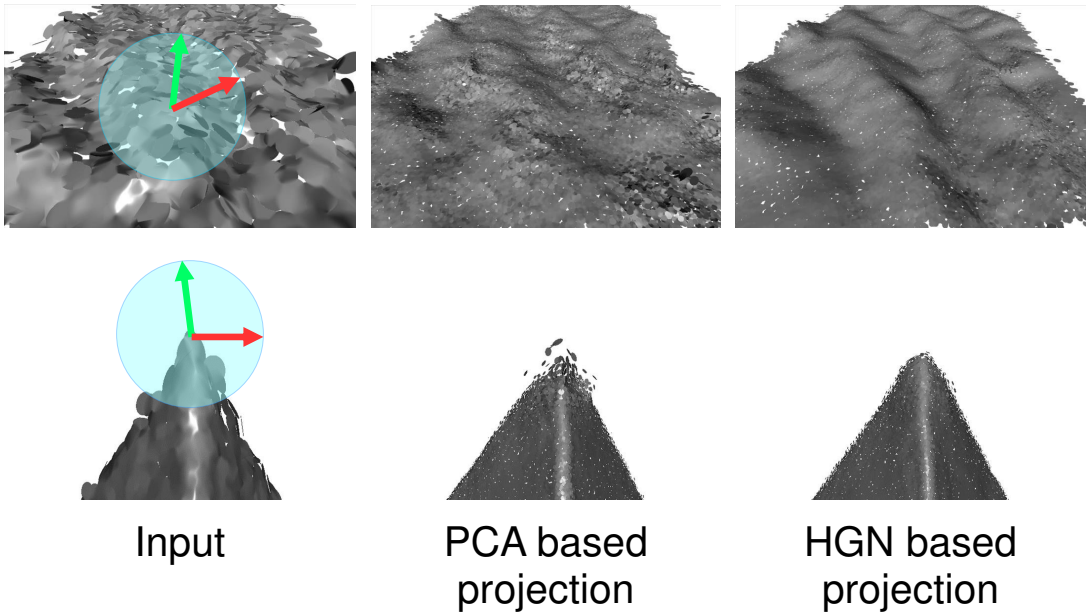

Figure 5.4: *Estimating consistent local directions for projection that are robust to noise and result in heigtmaps that capture local structure well is difficult with geometric methods such as PCA (shown in red), whereas our architecture generates a robust and propoer direction for heightmap generation (green). This is essential for HDN to generate accurate and consistent results as we show for consolidated point sets with projections estimated by PCA and HGN (middle and right).*

super-resolution [Kim et al., 2015; Zhang et al., 2016], obtaining state-of-the-art results. We thus also adopt a deep CNN for this step. HDN is inspired by a recent network architecture [Kim et al., 2015], consisting of a sequence of 10 convolutional layers with depth 64, and filters of size 7×7. After each convolution, batch normalization and a rectified linear unit layer (RELU) are applied.

Examples of noisy H_N and corresponding denoised H_D heightmaps obtained at testing time are shown in Figure 5.3. The network learns a very accurate mapping, leading to H_D very close to the ground truth patch images.

5.3.3 Training Procedure and Analysis

Training and Loss We first train HDN by using the ground truth plane parameters, thus by substituting $\mathbf{n}$ with $\mathbf{n}_{GT}$ in HGN, and minimizing the loss $\|H_D - H_{GT}\|_2$. This allows us to train the convolutional layers of HDN on patch pairs with perfect projection and resampling. Once the weights of HDN are trained, we fix them and train HGN with the same loss as before.

By imposing the same loss, we force HGN to learn the best projection such that the projected heightmap, once denoised, becomes as similar as possible to the ground truth image H_{GT}.

Robustness to Noise and Sampling Adopting a learning based approach to estimate projection and denoising simultaneously makes our local fits robust to noise and sparse data, and consistent with the local geometric features. This is very hard for purely geometric algorithms, such as fitting local planes with PCA. Such methods result in parameters that are overfitted to noise or biased with respect to the patch structure, depending on the size of the neighborhood, noise level, and local geometry.

Figure 5.4 (top, left) shows an example for a noisy patch (blue circle), where a PCA-based estimation of the normal vector at the patch center is given in red, and the **n** estimated by our network in green. The former is obtained by averaging the normals of the points in the patch, all estimated with a small neigborhood size (one third of the patch size) with PCA. Our estimated **n** generates a better heightmap that is consistent for noisy patches, as it captures the underlying local geometry well. This is clear also in the final consolidated point clouds, as we show for PCA-based projections followed by HDN in Figure 5.4 (top, center), and our full architecture HGN + HDN in Figure 5.4 (top, right). Utilizing our full architecture results in a much smoother geometry while preserving important features.

Such local fits with geometric techniques are also problematic when the size of neighborhood considered is large with respect to local geometric structures, which is the case for all our patches, as we need to capture local structure within our networks. In Figure 5.4 (bottom), the same comparison as in (top) is shown for a sharp feature, but in this case the PCA normals are computed with a size as large as that of the patch. The **n** estimated by our network (green) allows HGN to generate a proper heightmap around the peak, which can then be effectively denoised by HDN (bottom, right). PCA, on the other hand, estimates a vector (red) that is perpendicular to **n**. This results in a heightmap where distances of points are averaged, leading to artifacts in output consolidated point sets (bottom, center).

5.3.4 Processing Point Clouds at Testing Time

Processing a Single Patch At testing time, given an input point cloud, a spherical patch of radius r around a point is extracted. If the normals of the input point cloud are provided or pre-computed, the patch can be further

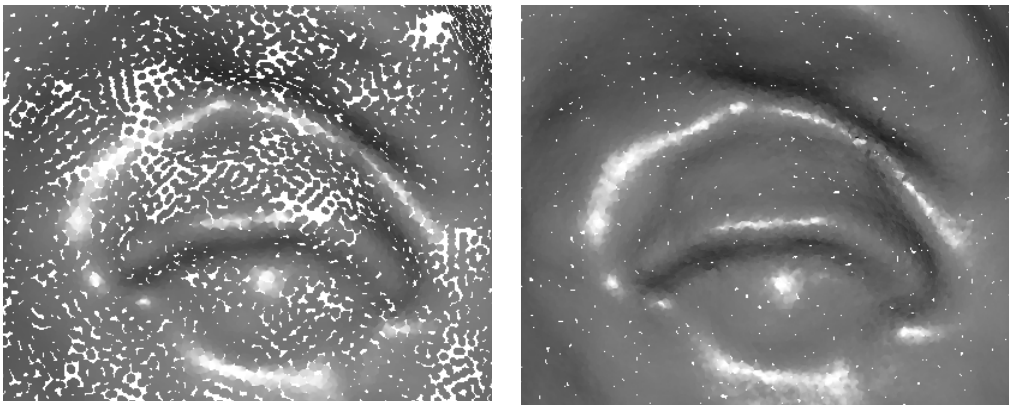

Figure 5.5: *The output consolidated point cloud, obtained by evaluating only patches around one quarter of the input noisy points (left), and around every input noisy point (right).*

refined by considering location-wise and normal-wise close points. The patch is first resampled to obtain n points as in data generation for training (Section 5.2.2), and centered at the origin. It is then fed to the *Frame Estimator* component of HGN (Figure 5.2) to estimate the normal direction $\mathbf{n}$ for the plane over which the heightmap H_N is defined. A random direction vector $\mathbf{d}$ orthogonal to $\mathbf{n}$ is then computed, and the noisy height map H_N is generated with *Projector*. The H_N is then denoised by HDN to produce H_D, which is finally converted into a point cloud by *Back-projector* with the same frame that *Projector* uses. Each pixel center with the corresponding depth given by H_D projects into a 3D point. Pixels with zero values are not projected, as they do not represent geometry (the resulting positions fall out of the patch sphere due to the offset we use as explained in Section 5.3.1), but are rather placeholders for no geometry. The resulting consolidated set of points is then translated to the original position of the input patch. We show examples of consolidated sets $\mathbf{Y}$ in Figure 5.3. As compared to the input noisy and sparse set $\mathcal{X}$, we get a denoised and much denser output set $\mathcal{Y}$.

Reprojection Density The above process is repeated independently for patches around every point of the input point cloud. The generated point sets are all retained in a final set representing the consolidated point cloud, without any further processing such as averaging of point locations.

In order to introduce overlaps between patches and hence produce a dense output, we evaluate a patch around each input noisy point. Less overlaps can be introduced for efficiency, at the cost of quality due to sparseness of the

output. An example of output point cloud with less overlapping is shown in Figure 5.5 (left), where patches were extracted only around one quarter of the input noisy points. Compared to the denser version in Figure 5.5 (right), it is smoother and contains several small holes.

The number of new points n' sampled on H_D further defines the density of the final consolidated point cloud. For example, reprojecting a single point corresponding to the central pixel of H_D would produce a denoised point cloud with the same number of points as the input, thus possibly losing the ability to preserve fine details. On the other hand, reprojecting a point for every pixel of H_D could lead to artifacts at the borders of the patch, nearby the zero value pixels which are placeholders for no geometry. The convolutions, indeed, may introduce smooth transitions between the zero values pixels and the ones representing geometry. We found projecting the pixels in a central part of H_D for each patch produces best results. After H_D is computed, we thus reproject only the pixels that fall into a square of size m around the patch center.

5.3.5 Extension for Point Normals

As we get a dense and denoised point set as the consolidated output, surface normals at the output points can simply be computed with existing geometric methods such as PCA. However, for cases where there are sharp features to be preserved, we might still not get the exact expected sharpness for normals as we are limited by the resolution of the intermediate image-based representation H_D. For such cases, we thus propose an additional network component for denoising point normals. The idea is to denoise, instead of a single-channel noisy heightmap H_N as before, a three-channel noisy normal map N_N, generated from the input normals. The architecture is the same as before with a few modifications. First, HDN processes images N_N of three channels, each representing a component of normal vectors. Second, each channel j of the normal map N_N is generated as in Equation 5.3, by interpolating the j^{th} component of the surface normals denoted by $N(\mathbf{i}, j)$ for projected points $\mathbf{i}$ as

$$N(\mathbf{c}, j) = \frac{1}{W(\mathbf{c})} \sum_{\mathbf{i} \in \mathcal{N}(\mathbf{c})} g(\mathbf{c}, \mathbf{i}) N(\mathbf{i}, j). \tag{5.4}$$

Here, $N(\mathbf{c}, j)$ is the j^{th} component of the normal vector stored at the pixel center $\mathbf{c}$, expressed in image plane coordinates as before.

At testing time, given an input patch $\mathcal{X}$, the maps H_N and N_N are generated with respect to the estimated frame, and their denoised versions H_D and N_D

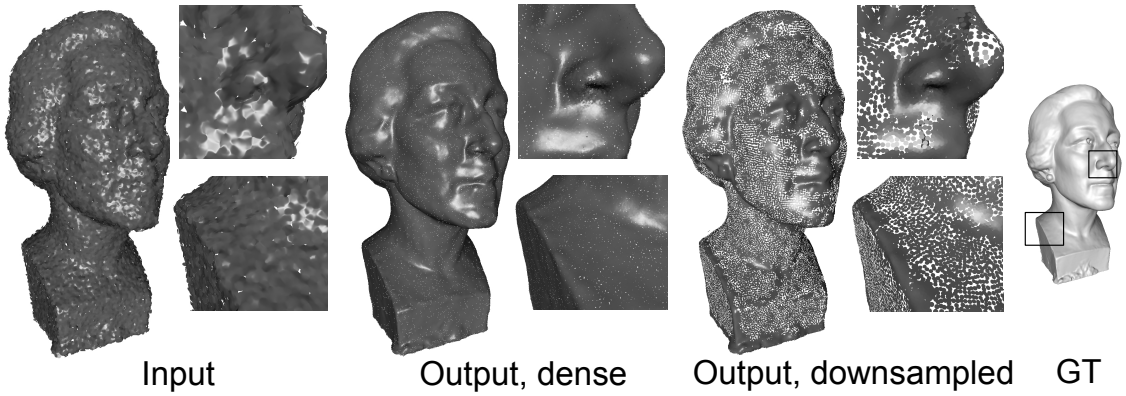

Figure 5.6: *From an input noisy point cloud (Input), our method produces a dense, consolidated version (Output, dense). Prior to surface reconstruction, this can be optionally adaptively downsampled (Output, downsampled) to speed up reconstruction algorithms. The ground truth mesh (GT) is shown as reference.*

are obtained via two separate HDN's. We can then obtain the point locations from H_D by back-projection as before, and normals from N_D by changing the coordinate system according to the same estimated frame.

5.4 Results

5.4.1 Network Implementation and Parameters

For all our experiments, we set the patch radius r to 5 times the average spacing between the input points, and resample the patches to $n = 100$ points (Section 5.2.2). We use images of size $k = 48$, and set $\sigma = 1/r$ for generating H_N, and $\sigma = 1/2r$ for generating the training dense heightmaps H_{GT}, with $\delta = 2.5\sigma$. We back-project points from the heightmap image H_G in a square of size $m = 24$. The whole architecture is trained with the Adam Optimizer with an initial learning rate of 0.0001 lowered by 10 times every $30k$ steps. We feed the patches in batches of size 8. For the PointNet components [Qi et al., 2017a], we use the default parameters and their basic code for handling unordered point sets as input. The network was implemented in TensorFlow.

5.4.2 Pipeline For Surface Reconstruction

A key application of point cloud consolidation is to serve as a preprocessing step to surface reconstruction algorithms. These algorithms are affected by

noise and sparseness of the input data. Thus, providing a consolidated, dense point cloud is critical for improving the reconstructed surfaces. We start by applying our method to a noisy input point cloud and generating a consolidated dense version. Since the resulting point cloud is very dense, we can easily downsample it in an adaptive fashion, keeping a high density of points in proximity of the features. This step considerably speeds up the reconstructions without loosing quality. In particular, we use a simple and efficient clustering algorithm [Pauly et al., 2002], where downsampling is obtained by grouping points in local clusters. In order to keep a denser sampling near the features, the size of the clusters is adapted to the local variation of the point set. We use 30 as a maximum cluster size and 0.02 as maximum surface variation (for the *flags* dataset, see below, 20 and 0.03).

Figure 5.6 illustrates an input point cloud (of about 50k points) from our *sculptures* dataset (see below), our dense consolidated point cloud (about 1.6M points), our subsampled point cloud (about 150k points), and the ground truth mesh from which the noisy input was sampled. The generated dense point cloud does not present noise and preserves detailed features such as the nostril and the edges of the base. Those features are also preserved in the downsampled point cloud, while reducing the overall sample count.

If not learned through our point normals network extension, we estimate the point normals of the consolidated point cloud using PCA of local neighborhoods and a Riemannian graph for their global orientation. Due to the high density and quality of the output consolidated point cloud, this simple approach already obtains high quality results. We compute point normals with PCA on 50 nearest neighbor points.

Finally, we reconstruct the underlying surface by extracting the iso-surface of the *RIMLS* [Öztireli et al., 2009] using the marching cubes algorithm. We refer to our surface reconstruction results as *OURS-R*, and our output dense point clouds as *OURS*. We use a spatial low pass filter of 7 to 10 times the local spacing of output points for RIMLS. The RIMLS sharpness parameter σ_n is set to 0.75.

5.4.3 Datasets

For our experiments, we built three datasets: two with synthetic data and different levels of noise (*sculptures* and *flags*), and one with real-data from a sensor (*Kinect v2*), each composed of objects with similar features and separated in a training and a testing subset. For testing our point normals network extension, we built an additional synthetic dataset containing multiple geo-

metric sharp features (*geometric shapes*). The test point clouds only contain point locations, without point normals or any additional attributes.

For training, we have three ground truth models for the *sculptures*, three for *flags*, four for *geometric shapes*, and four for *Kinect v2*, from which many training patches are generated. The ground truth point clouds of *sculptures*, *geometric shapes* and the noisy and ground truth point clouds of *Kinect v2* were extracted from the models also used in recent works [Wang et al., 2016]. For the Kinect models represented as meshes, we remove the connectivity information and just retain the vertex locations. The meshes of the *flags* dataset were generated by animating a waving flag mesh and randomly selecting frames. While the *flags* dataset is very specialized (each model has wrinkles of similar shapes and sizes), the other datasets are more general. The ground truth models of our datasets are shown in Figure 5.7.

For each model in the training sets, ground truth point sets are twice as dense as the input point sets, and are generated by Poisson disk sampling for equal distribution of points. Synthetic Gaussian noise was dynamically added to the input points at training time, as the training patches were generated. For the *sculptures* dataset, three training sessions were performed, each with noise of a different standard deviation ($\sigma_1 = 0.037r$, $\sigma_2 = 0.075r$ and $\sigma_3 = 0.15r$), while for the *flags* and the *geometric shapes* datasets, $\sigma = 0.075r$.

Our testing sets contains six models for *sculptures*, ten for *flags*, two for *geometric shapes*, and 14 scans of three models for *Kinect v2*. For each model, an input noisy and sparse point cloud was sampled (except for *Kinect v2*, where we already have noisy scans) from the ground truth model with the same conditions as in the corresponding training dataset. The input point clouds of the *sculptures* dataset consist of about 70k points on average, while the other datasets come with around 15k points for testing models.

5.4.4 Comparisons

We compare numerically (reconstructions) and visually (point clouds and reconstructions) to five common and state-of-the-art methods for point consolidation and surface reconstruction: Poisson Surface Reconstruction [Kazhdan et al., 2006], APSS [Guennebaud and Gross, 2007], RIMLS [Öztireli et al., 2009], WLOP [Huang et al., 2009], and EAR [Huang et al., 2013]. While Poisson, APSS, and RIMLS directly produce an iso-surface, WLOP and EAR generate a resampled point cloud. For comparing our surface reconstruction results, we thus apply RIMLS to the output of WLOP and EAR, and utilize marching cubes to extract the final surface for all methods. We refer to these combinations as WLOP-R and EAR-R.

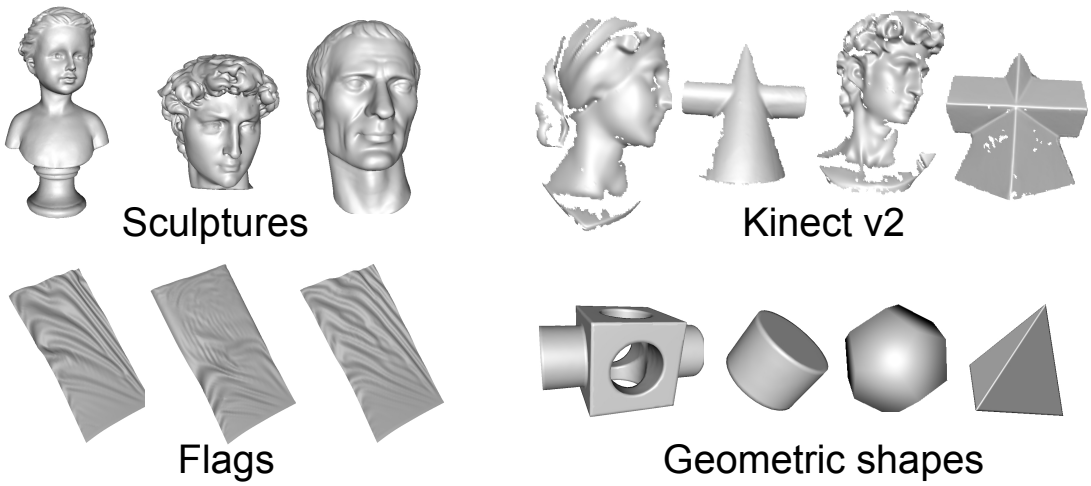

Figure 5.7: *The ground truth meshes for our four training datasets.*

In order to numerically compare the mesh reconstruction results, we adopt the Hausdorff distance between the reconstructed meshes and the ground truth ones. As we have models that are not closed, we used the one-sided Hausdorff distance from a ground truth mesh to the reconstructed one, in order to avoid including errors due to extra surface parts around the boundaries in the reconstructed mesh. The Hausdorff distance is normalized with respect to the diagonal of the bounding box of the mesh, and multiplied by 10^4. For every dataset, we compute the average Hausdorff distance for all testing models.

We exhaustively search for the best parameters for the other methods by running an extensive test for each model. For APSS, RIMLS, WLOP-R and EAR-R, the spatial low pass filter parameter is tuned separately for each dataset and each method, by testing a set of values varying between three to ten times the local point spacing, and choosing the best result. The RIMLS sharpness parameter σ_n is set to 0.75 for all methods. For Poisson Surface Reconstruction, an octree depth parameter of 14 is used. The WLOP and EAR neighborhood radius parameter is set to 8 times the average spacing of the input point set, and the EAR sharpness parameters to an angle of 30 with edge sensitivity 0.05. For methods that require surface normals, we estimate them with PCA again by using optimized values for each case.

5.4.5 Experiments

For all experiments, we observed a significant visual and numerical improvement over the existing methods when using our technique. We show example input and consolidated output point clouds using our technique as well as

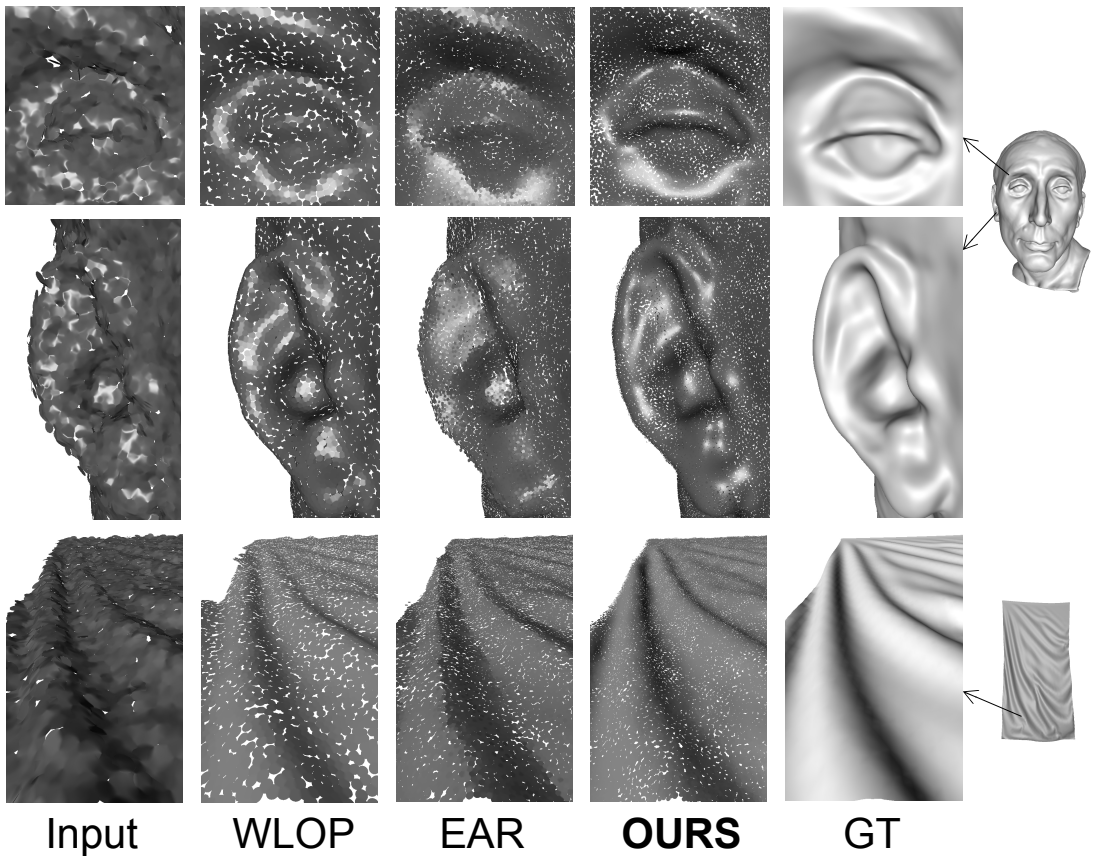

Figure 5.8: *Consolidated point clouds on the noisy input Nicolo (from the sculptures dataset with σ_2), and on a model from the flags dataset. Our method captures local structures of the ground truth (GT) model accurately.*

others in Figure 5.8. While WLOP oversmoothes the details of the model Nicolo and the wrinkles of the flag, the dense point clouds of EAR deform the geometry by creating extra sharp edges that are not present in the original models, e.g. at the border of the ear or at the peak of the flag wrinkle. Our dense point clouds reproduce the ground truth local structures more faithfully, e.g. we get a realistically rounded eyelid without turning it into a sharp edge.

We show visual comparisons of reconstructions in Figures 5.1, 5.9, 5.10, and 5.11. We observe that Poisson, APSS and WLOP-R tend to generate oversmoothed surfaces, as can be seen for many surface features, e.g. for the eyes of Nicolo in Figure 5.1, the ear and hair of Bimba in Figure 5.10, or the flag in Figure 5.11. The oversmoothing effect is confirmed by the last row of Figure 5.9, displaying the distances from the ground truth to the closest point on the reconstructed meshes. In the detailed ear and hair regions, these methods have high errors. On the other hand, these methods can also produce

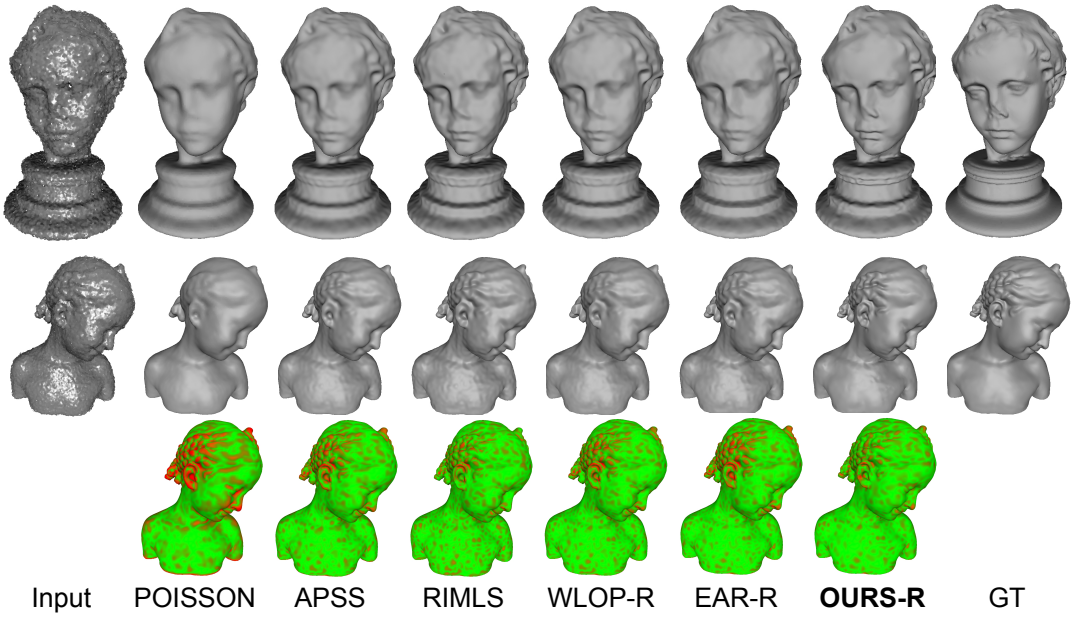

Figure 5.9: *Surface reconstructions for the test models Eros (top) and Bimba (bottom) from the sculptures dataset with σ_2, with ground truth meshes (GT) for reference. For Bimba, the distances from the ground truth mesh to the reconstructed meshes are also displayed (red encodes large values).*

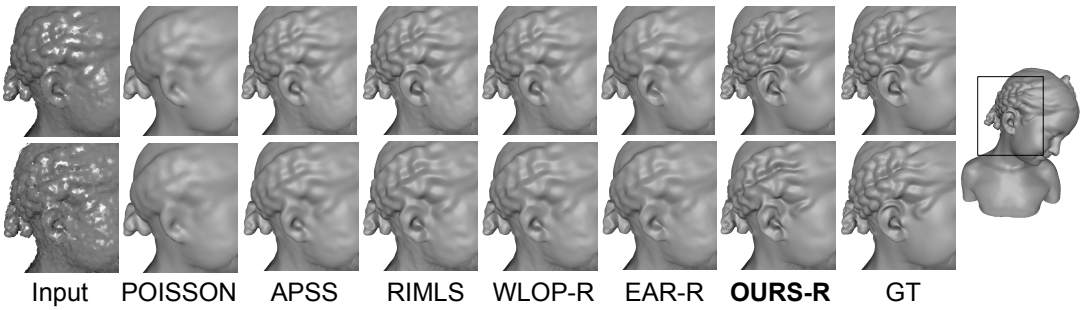

Figure 5.10: *Close-ups of the reconstruction of Bimba, with two different levels of noise σ_1 and σ_2. In both cases, our method (OURS-R) produces the most accurate ear and hair features, while keeping the cheek smooth as in the ground truth (GT).*

noisy results depending on the input, as for the buste and neck of the Boy model in Figure 5.11. Our technique produces faithful reconstructions for all cases, avoiding over- or under-smoothing of local structures.

RIMLS and EAR are designed to preserve sharp features. Indeed, EAR-R produces accurate results in geometric shapes with clear edges such as the base of Eros in Figure 5.9. However, it fails to correctly preserve more organic, detailed features such as the face of the same model or the ear and hair

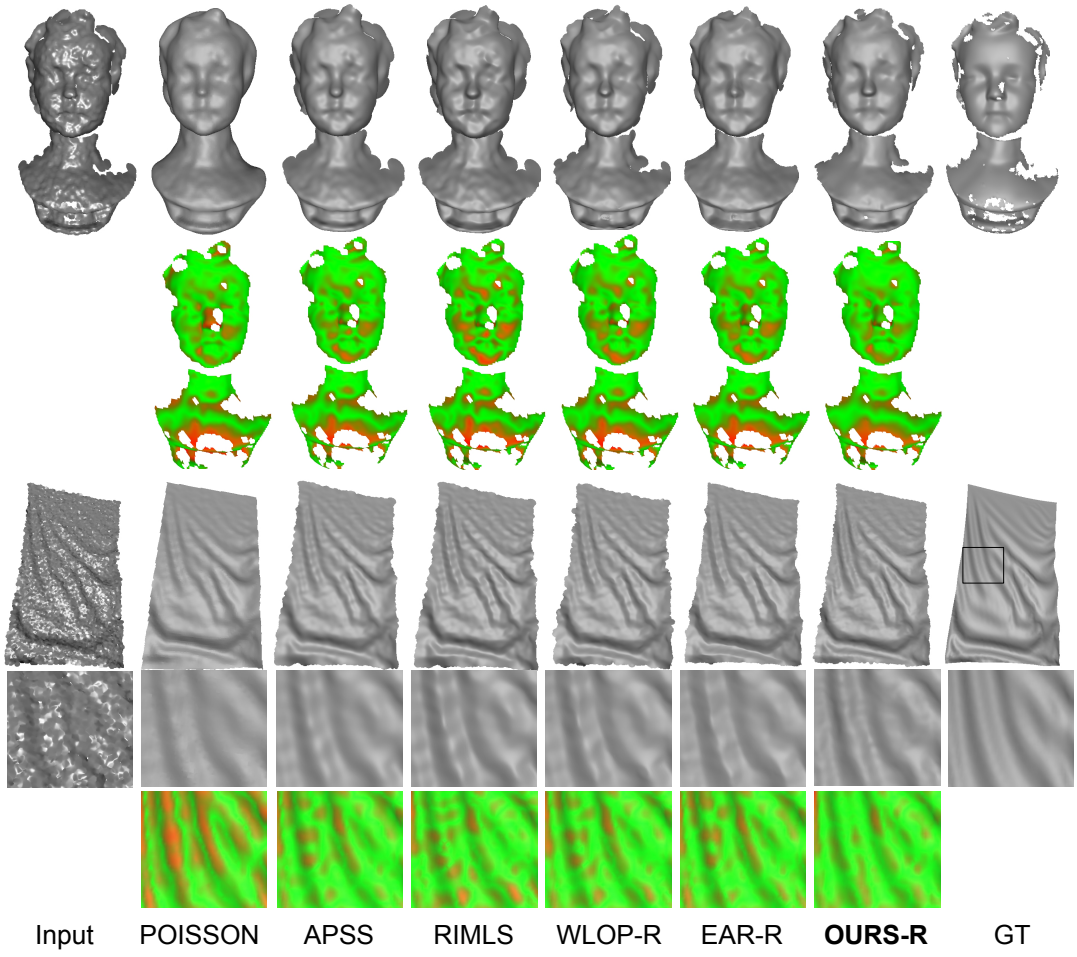

Figure 5.11: *Reconstructed surfaces for the testing models Boy (top) and a flag (bottom) from the Kinect v2 and flags datasets, respectively. The distances from the ground truth (GT) mesh to the reconstructed meshes are also plotted for the flag.*

of Bimba, as shown in Figure 5.10 and in the distance maps in Figure 5.9. Similarly, RIMLS performs better on sharp features, but produces bumpy results in smooth regions, such as the buste of Bimba, and overall cannot capture delicate structures such as the eyes of Nicolo in Figure 5.1, or the wrinkle profile in Figure 5.11. Instead of sharpening details, our method outputs high quality structures that more faithfully reproduce the underlying geometry, thanks to the learned representation.

These visual results are confirmed by quantitative comparisons in Table 5.1. The best two results for every dataset are highlighted in bold. Our method obtains the smallest Hausdorff distance for every dataset.

Dataset	APSS	RIMLS	WLOP-R	EAR-R	**OURS-R**
Scu. σ_1	2.85	**2.62**	2.99	3.6	**2.57**
Scu. σ_2	**4.11**	4.27	4.26	4.56	**3.70**
Scu. σ_3	**6.28**	7.37	6.54	6.74	**6.14**
Flags	**5.69**	6.03	**5.69**	5.93	**5.55**
Kin.v2	17.58	19.40	**17.24**	17.83	**17.10**

Table 5.1: *The Hausdorff distances averaged over the testing models in each dataset for each method. The best two performing methods are highlighted for each dataset. Our method (OURS-R) outperforms the others in every dataset.*

Cross-Training In Figure 5.12, we analyze the importance of training datasets for accurate structure recovery. We show consolidated point clouds of the Nicolo model by using the ground truth plane normal $\mathbf{n}_{GT}$ and direction $\mathbf{d}_{GT}$ for each patch, and varying the heightmap denoising procedure. In particular, in the first column, the H_D's are generated by simply smoothing the H_N with Gaussian interpolation using a small σ, in the second column the same smoothing is applied but with a larger σ, in the third column we use HDN to generate the H_D but using the weights trained on the *flags* dataset, and in the fourth column we use the weights trained on the *sculptures* dataset. The last column is the ground truth, and every column contains the generated point cloud and four example H_D's. Simpler image denoising techniques produce considerably worse results, and by specializing the training with models of the same class as the testing models, we can obtain substantially better results than with more general datasets.

Point Normals While our dense output point clouds allow for accurate estimation of surface normals with PCA, we found our extension for denoising normals to be useful for preservation of sharp features for geometric objects, as elaborated on in Section 5.3.5. In this case, the input normals to generate N_N were estimated from the noisy point clouds with PCA, and oriented with a Riemannian graph. In Figure 5.13, we show reconstruction results on the *geometric shapes* testing dataset (Fandisk and Icosahedron). The normals on the consolidated point clouds, color-coded in this figure, are estimated with PCA, or learned with the normal estimation network for comparison. The close-ups of the point clouds (top), and the reconstructed surfaces (bottom) illustrate that the learned normals better preserve the sharp features. In Figure 5.14, we further illustrate some noisy normal maps N_N, denoised versions N_D, and the ground truth N_{GT} for the Fandisk model. We can observe that our N_D contains very sharp edges.

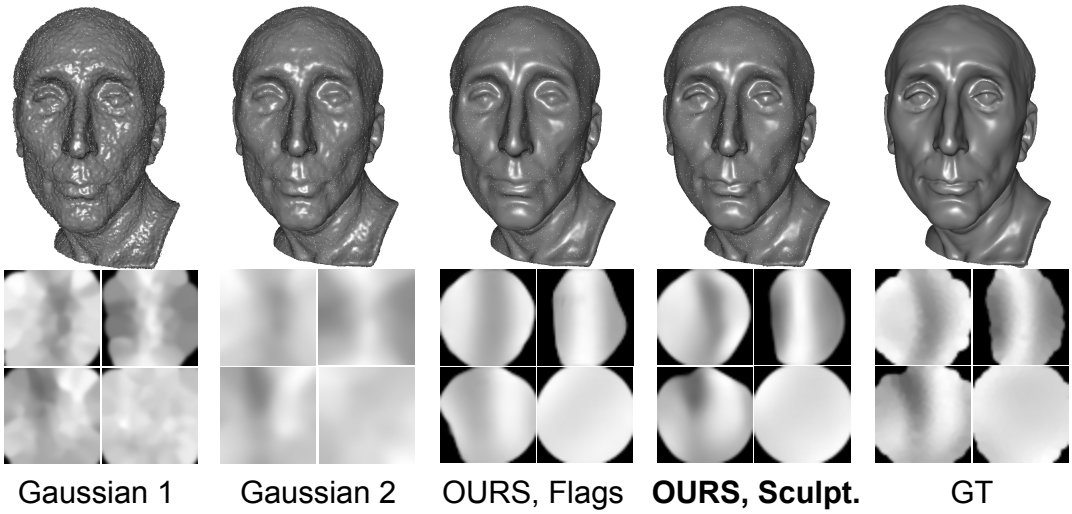

Figure 5.12: *Different denoising variations. The noisy heightmaps H_N of a noisy Nicolo model are denoised with: Gaussian smoothing with a small σ, Gaussian smoothing with a large σ, using our network trained on the flags dataset, using our network trained on sculptures. The resulting consolidated point clouds are shown (top) with four example denoised heightmaps (bottom). The ground truth (GT) data is shown for reference.*

Timing All trainings were performed for 200k steps, lasting on average 5 hours. The total time required to denoise an input patch is about 0.013 seconds, out of which about 44% is for preprocessing the input, 15% for estimating H_N, 33% to denoise it to H_D, and 7% to reproject the points. For the bimba model (of about 60k points), the total processing time was about 90 seconds on a GTX 970 and a i5-3570 CPU, 3.40GHz. As each patch is local and processed independently, our technique thus allows for real-time patch-wise consolidation, and is trivially parallelizable.

Limitations In this chapter, we target the typical problems of noise and sparse sampling in input point clouds. However, when the input point cloud contains relatively large holes, our current scheme is not able to fill them with a sampling as dense as in the other parts of the point cloud. This is because there is less overlap of projected patches near the holes, since we generate patches of consolidated point clouds only around existing input points. As a consequence, the RIMLS reconstruction might lead to deformed surface parts in those regions. This can be seen for the base of the Boy model in Figure 5.11. This could be alleviated by having a denser sampling of patches around the holes, starting from the boundaries and progressively closing the holes, similar to texture synthesis. In case of missing parts considerably

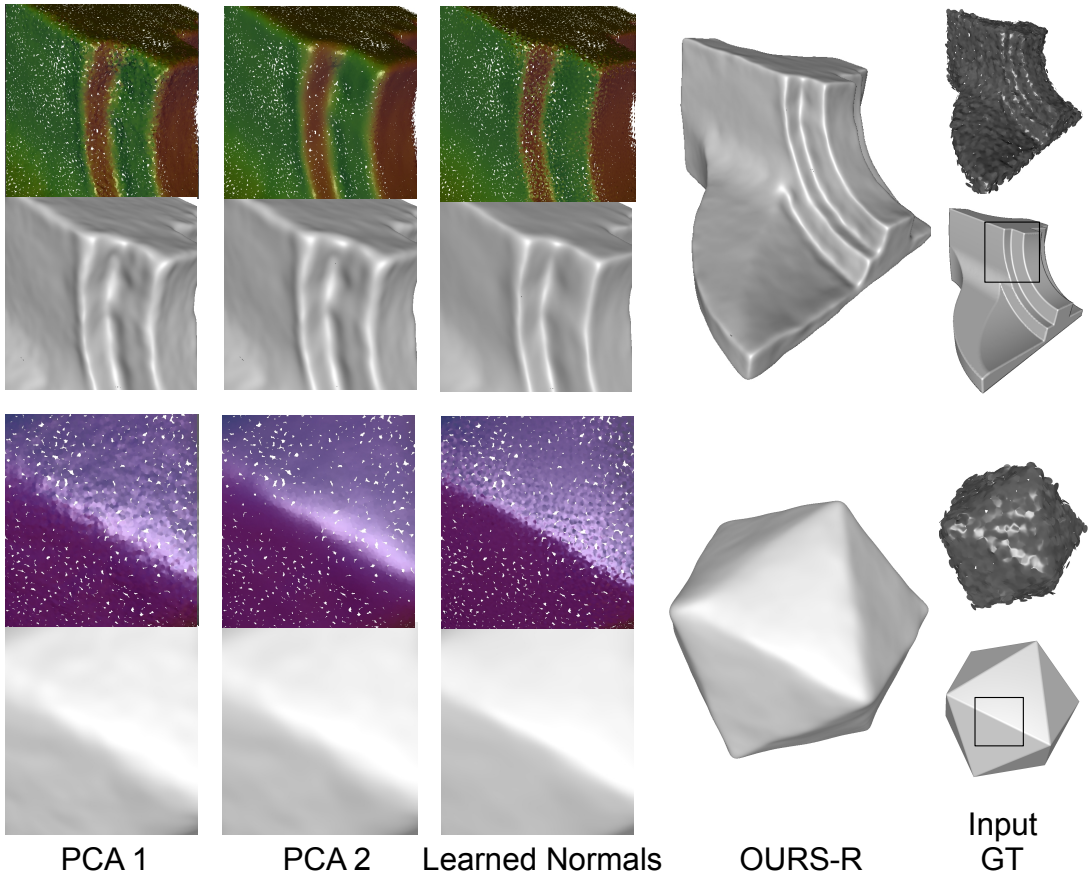

Figure 5.13: *Consolidated point clouds with color-coded normals and RIMLS reconstructions of a noisy input Fandisk (above) and Icosahedron (below) from the geometric shapes dataset. The normals are: estimated with PCA with a small radius (PCA 1), PCA with a large radius (PCA 2), or learned with our normal estimation network (Learned Normals). The latter better preserves sharp edges.*

larger than the patch size, a global filling approach would be required. Our method is designed to capture local structures for manifold surfaces, as many previous techniques including MLS based approaches. This allows us to use local heightmaps as an intermediate representation. However, such a representation comes with well-known limitations for non-manifold structures and large surface parts that cannot be represented with such a parametrization. Possible solutions are utilizing multi-depth maps and more complex differentiable parametrizations that can be efficiently trained. A typical example is when two surface sheets fall into the same patch, where our current method would average their locations as shown in Figure 5.15 (left). If the input normals are provided, this problem can be solved by extracting patches considering location-wise and normal-wise close points,

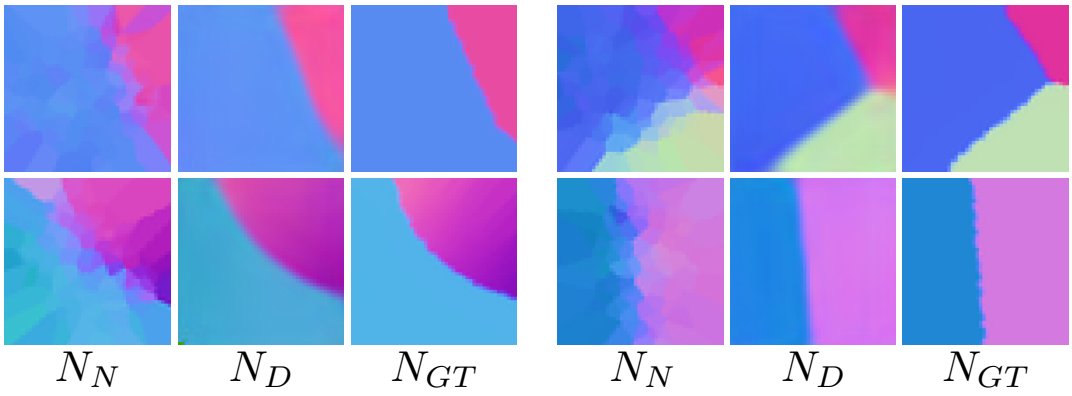

Figure 5.14: *Example normal map denoising results from the Fandisk model. N_N is the input noisy, and N_D is the denoised normal map. The ground truth normal map N_{GT} is shown for reference.*

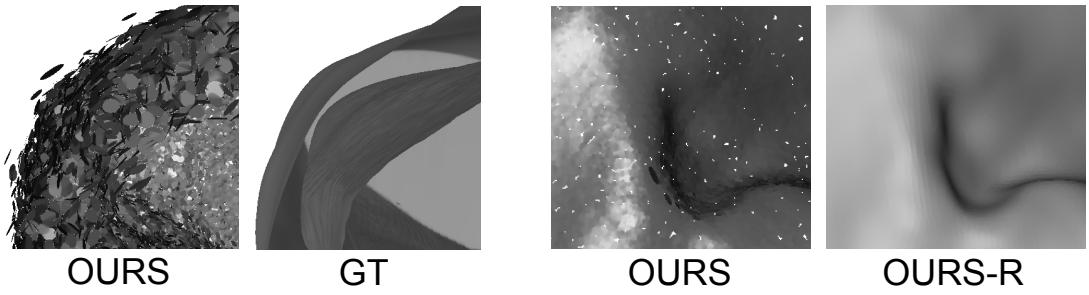

Figure 5.15: *Limitations of our method. In case of two surface sheets falling into the same patch (left, GT) , our method averages the positions of the points creating a noisy result (left, OURS). Occasional frames can be badly estimated in the presence of complex patches unseen during training, and thus some bad points can be generated (right, OURS). Due to the high density of our results, the final reconstructed surfaces are not affected (right, OURS-R).*

as mentioned in Section 5.3.4. We generate a patch around each point in an input point cloud at testing time. This means that for high levels of outliers, we might end up with extra output points that are far away from the surface. For these cases, a new training dataset and procedure need to be designed to set all depths values of H_D for an outlier to zero. Similarly, in cases of patches very different from the ones in the training set (e.g., partial patches at the borders), or badly performed training (e.g., too short training) occasional badly estimated outlier frames may occur. A badly estimated frame would, in most cases, produce a bad set of points, as can be seen in our output point cloud in Figure 5.15 (right). Since our produced point cloud is very dense and mainly made of consistent re-projections across patches, even in the presence of few bad points the final reconstruction is not negatively affected,

as shown in the reconstructed surface in Figure 5.15 (right). Finally, our H_D images are sometimes slightly smoother than H_{GT}. This is a property of the used convolutions, and the behaviour could be improved by utilizing more advanced image network architectures such as Generative Adversarial Networks [Goodfellow et al., 2014].

5.5 Discussion

In this chapter, we presented PointProNets, a fully differentiable, CNN based deep learning architecture to process point clouds. The input unordered points are internally converted to regularly sampled height maps, which are suitable to be processed by modern and well-performing CNN architectures. We demonstrated the potential of this architecture by developing an end-to-end algorithm to consolidate raw point clouds, where local parametrizations and fitted surfaces are learned jointly, to achieve superior reconstructions where delicate features and details of surfaces are preserved.

Although we have focused on point cloud consolidation in the scope of this work, the proposed architecture has the potential to be used for many other points based geometry processing tasks. Moreover, as in our additional component for point normals denoising, the architecture could be easily extended to points with attributes, such as colors for joint depth-color data processing.

CHAPTER 6

A Network Architecture for Point Cloud Classification via Automatic Depth Images Generation

In this chapter, we show how our patch-based neural network architecture from Chapter 5 can be extended for global operations where the full point cloud is processed. In particular, we propose a novel neural network architecture for point cloud classification. Our key idea is to automatically transform the 3D unordered input data into a set of useful 2D depth images, and classify them by exploiting well performing image classification CNNs. We present new differentiable module designs to generate depth images from a point cloud, by adapting the ones of Chapter 5. These modules can be combined with any network architecture for processing point clouds. We utilize them in combination with state-of-the-art classification networks, and get results competitive with the state of the art in point cloud classification. Furthermore, our architecture automatically produces informative images representing the input point cloud, which could be used for further applications such as point cloud visualization.

6.1 Introduction

After tackling local problems in the previous chapters, we now consider an application where the point clouds are processed in a global fashion: point cloud classification. Being able to automatically classify point clouds is a

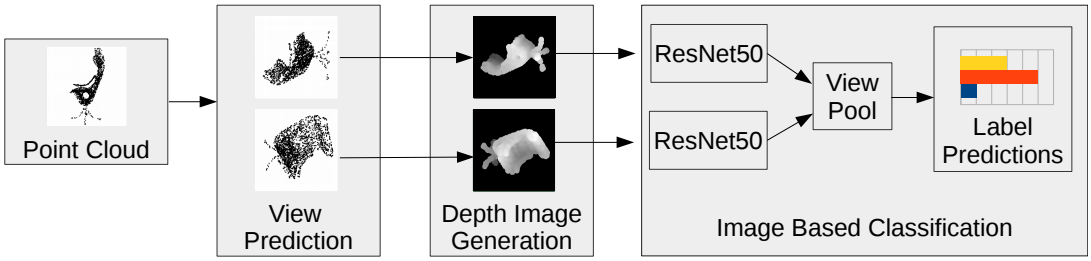

Figure 6.1: *An overview of our network architecture. Given an input point cloud of a chair, two informative views are predicted, the corresponding depth images are generated and fed to the image classification module.*

challenging task that can have impact in many other problems in Computer Vision and Graphics, such as scene understanding and surface reconstruction.

Even though 3D scanners are becoming cheaper and more available, 2D images still represent the majority of our graphical information. Thanks to significantly large image datasets [Deng et al., 2009], and a growing interest in the research community, Convolutional Neural Networks (CNNs) for image classification have been well studied and achieved state of the art results. Inspired by their high quality results, we build a novel neural network architecture which allows us to exploit the strengths of 2D image based CNNs for classifying 3D point clouds. Unlike some very recent deep learning methods [Qi et al., 2016a; Klokov and Lempitsky, 2017; Simonovsky and Komodakis, 2017], which handle unordered 3D data by directly processing and classifying them, our idea is to design a set of trainable network components that automatically transform the 3D input to informative 2D images, which are then input to image classification networks. Contrary to previous works [Su et al., 2015] that classify 3D meshes by exploiting rendered images, in our method, the images are not generated in a pre-processing step, but rather learned within the network.

In particular, our completely differentiable architecture first intrinsically predicts one or multiple views, which are informative about the shape and features of the input point cloud. Secondly, another differentiable module generates the corresponding depth images of the point cloud rendered from those views. These depth images are produced by extending the work presented in Chapter 5 to handle point clouds with multiple layers of depth, occlusions and overlapping structures. Finally, a third component combines the images and uses an image classification CNN [He et al., 2015] to classify them. Thanks to the generated depth images and high performance of CNNs for image classification, we obtain competitive classification results to the recent methods in the field. Furthermore, the views intrinsically generated

by our network can be extracted as an additional output at testing time, and used for point cloud visualization.

To summarize, the contributions presented in this chapter are the following:

- We propose a novel neural network architecture for point cloud classification that achieves results competitive with the state of the art. The key idea is to automatically transform the unordered 3D points to informative 2D images and exploit the well studied image based classification network architectures (and their pre-trained weights on large image datasets).
- Our architecture produces one or a set of informative depth images of the point cloud, by predicting meaningful view directions. We illustrate that the learned view directions and the corresponding depth images can be used for other applications, such as point cloud visualization.
- We propose a fully differentiable module for generating depth images of point clouds representing full 3D objects with occluded points, by integrating a point cloud culling strategy, extending the methods described in Chapter 5. This module can be used in further tasks and architectures that work with point clouds.

6.2 Network Architecture

6.2.1 Overview

Instead of directly classifying a point cloud, we designed a network architecture which automatically transforms the 3D input into a set of informative 2D depth images, and then solves the problem of classifying them. The view directions for generating the depth images are learned in an unsupervised manner, thus predicted with the goal of maximizing the classification accuracy. The main advantage of this approach, compared to directly processing the 3D points as in [Qi et al., 2016a], is that it allows us to exploit the well studied deep learning architectures for image classification, which have been proven to achieve state-of-the-art results. Moreover, in addition to outputting a class label prediction for the input point cloud, our network intrinsically learns to predict one or a set of informative view directions and generate the corresponding 2D depth images, which could be used for other applications e.g. for 3D object recognition or point cloud visualization.

Given an input point cloud P and a desired number of views K, our pipeline is to first choose K views directions, generate the correspondent depth images and then utilize them to classify the point cloud. Our network architecture is thus composed of three modules: the first one takes the input point cloud coordinates as input and predicts K direction vectors (Section 6.2.2); the second module receives the input point cloud coordinates and the K directions and generates the depth images accordingly (Section 6.2.3); finally, the third module combines the K depth images and produces a vector representing the prediction labels for classes (Section 6.2.4).

We train the three modules jointly within a single architecture, using a softmax cross entropy loss on the class labels, provided as ground truth. Notice that we do not include a loss on the view directions nor on the generated depth images. Figure 6.1 shows an overview of the network architecture, for the case where $K = 2$.

6.2.2 View Prediction

The first module of our architecture receives the point cloud P as input and produces K view directions, where K is a parameter chosen by the user. In particular, we want to estimate the K camera-pose matrices which represent 3D rotations to transform the point cloud in order to perform an orthogonal projection. Like in Chapter 5, we start from the recent method PointNet [Qi et al., 2016a], which proposes a network architecture that allows for processing unordered 3D point sets, like our input. The prediction of the views should respect crucial properties such as invariance to permutations of the input data and invariance under transformations. PointNet achieves input permutations invariance through a max pooling layer that approximates a symmetric function, and transformation invariance by predicting an affine matrix applied to the input. Finally, a fully connected layer creates a global descriptor used for classification.

In our approach, we utilize a separate PointNet architecture for each of our K views, modifying the final fully connected layer to produce a 6D vector, representing the camera view vector v_k and the up-axis u_k of the kth camera-pose matrix. We build C_k, the camera-pose matrix for the kth view, by setting $w_k = v_k \times u_k$ and $C_k = \left[w_i^T; u_i^T; v_i^T\right]$.

We finally multiply the input point cloud sequentially with every camera-pose matrix, producing K rotated point clouds. After the transformation, the p_x and p_y coordinates of a point $p \in P$ represent its image coordinates, and the p_z coordinate is its depth.

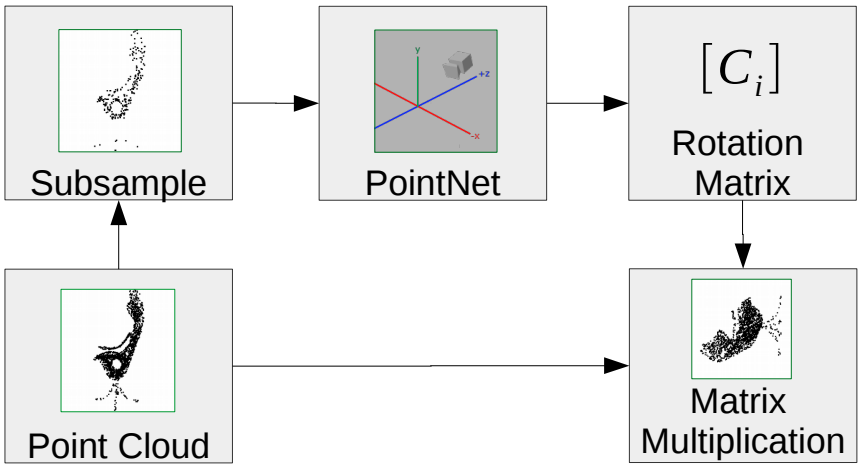

Figure 6.2: *Our view prediction module. The camera-pose matrix parameters are estimated from a downsampled version of the input, and the original points are rotated accordingly.*

Note that, due to memory limitations, we utilize a subsampled version of the input point cloud (keeping 12.5% of the original points) to estimate the camera-pose matrices. See Section 6.3.1 for more details on the implementation. Figure 6.2 shows a diagram of the view prediction module.

6.2.3 Depth Image Generation

In this network component, the goal is to generate a depth image for an input point cloud using differentiable operations. In Chapter 5, we present a differentiable layer to create a distance field image from a point cloud by interpolating the depth values of the points on the image plane using Gaussian interpolation. Since we project every point of the point cloud to the image plane and interpolate their distances, the approach is mostly suitable for generating depth images of small patches of points lying on a single sheet without overlapping structures, but does not produce useful depth images when the point cloud contains structures on different depth layers. In that case, indeed, points with a large difference in the depth coordinate may be projected to closeby pixels on the image, and their depth values will be averaged together, leading to skewed geometry representations as shown in Figure 6.3 (top).

In this chapter, we propose an extension that is suitable for point clouds representing full objects, with points lying also on occluded multiple layers. We thus aim at producing depth images which properly approximate a rendering of a surface passing through the points, which can be more reliably

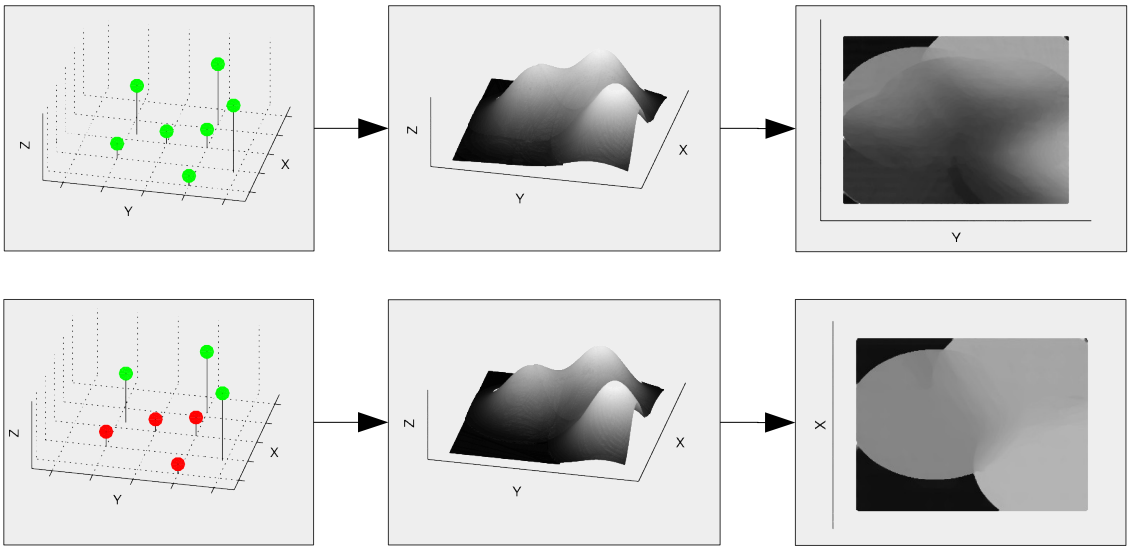

Figure 6.3: *(Top) The Gaussian interpolation of Chapter 5, where all the points are interpolated together. (Bottom) Our extended Gaussian interpolation, where the red points are filtered out because their depth is too different than the max_D values of the pixels. The green points are interpolated.*

classified by our final image classification component. In particular, a depth image should present clear edges where the depth changes abruptly, and continuous values in smooth parts of the object.

The main idea of our depth image generation method is to apply a bilateral-filering-like interpolation to obtain point cloud culling. In practice, instead of considering all the points in the point cloud, we segment the points belonging to the farthest surface layers from the image plane, and interpolate only their depths. We get the final image f by first computing a maximum depth value max_D for every pixel c, representing the maximum depth of the points $p \in P$ which are close enough to c when projected on the image plane. We define the subset $P'(c)$ as all the points $p \in P$ close enough to c on the image plane, given a threshold δ_1 :

$$P'(c) = \{p \in P | \ \|(c_x, c_y) - (p_x, p_y)\| < \delta_1\}. \tag{6.1}$$

It follows that:

$$max_D(c) = \max\{p_z | \ p \in P'(c)\}. \tag{6.2}$$

If $P'(c)$ is empty, no points will be projected closeby c, so we set the final value $f(c)$ for the pixel to zero. This step implicitly introduces a cutoff distance

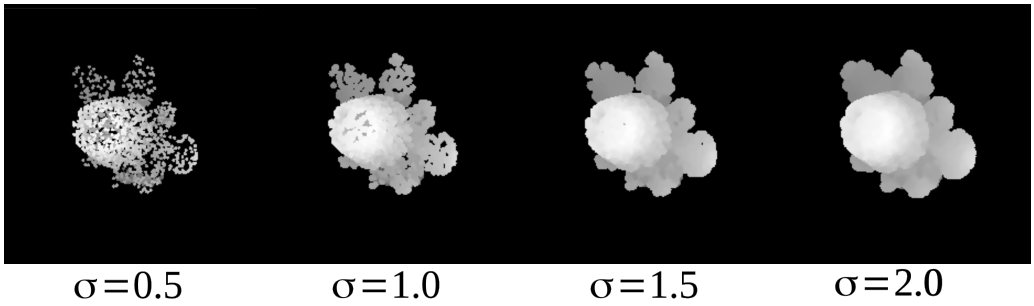

Figure 6.4: *Depth images of a flower pot generated with different values for σ.*

of δ_1 to the final Gaussian interpolation of our image generation procedure, allowing us to produce depth images with clear hard edges at the border of the objects. In case $P'(c)$ is not empty, in order to compute the final value $f(c)$ for a pixel, we consider only the points which have a depth value close enough to $max_D(c)$, and interpolate their depths (Figure 6.3, bottom). This ensures that our generated image has hard edges where the depth changes abruptly and smoother variations elsewhere. For a chosen threshold δ_2, we define a new subset of points $P''(c)$:

$$P''(c) = \{p \in P| \ |max_D(c) - p_z| < \delta_2\}. \tag{6.3}$$

For a pixel c, we apply Gaussian interpolation on the depth values of the points in $P''(c)$ as follows:

$$f(c) = \frac{1}{W} \sum_{p \in P''(c)} g((c_x, c_y), (p_x, p_y)) p_z, \tag{6.4}$$

with a normalization term W:

$$W = \sum_{p \in P''(c)} g((c_x, c_y), (p_x, p_y)), \tag{6.5}$$

and a Gaussian function g:

$$g((x, y), (x', y')) = e^{\frac{-(x-x')^2-(y-y')^2}{2\sigma^2}}, \tag{6.6}$$

where σ influences the smoothness of the generated depth images, as can be seen in Figure 6.4.

We generate a depth image for each of the K point clouds passed from the previous module. Notice that since we define max_D with the maximum

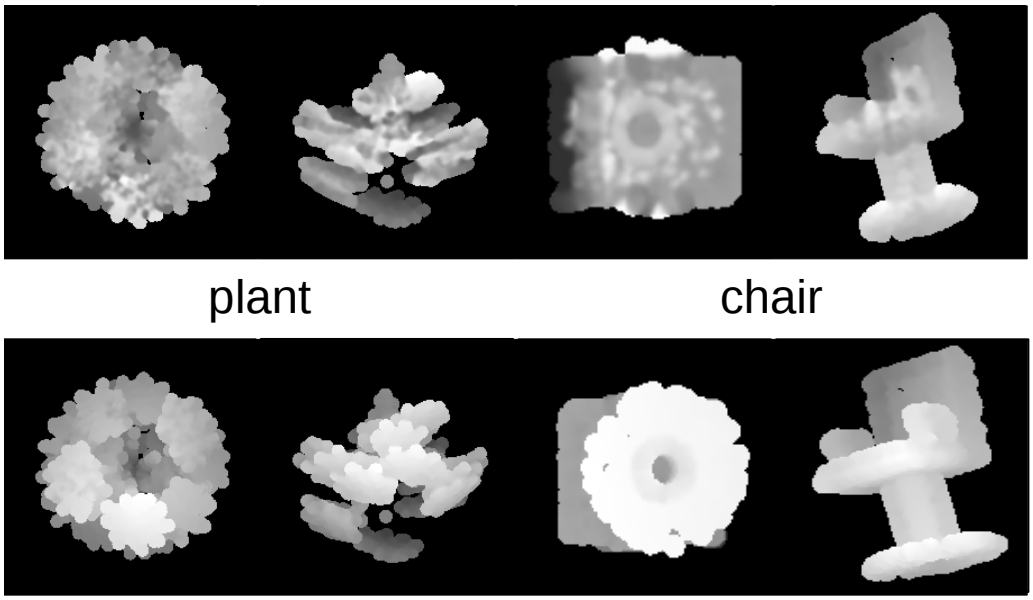

Figure 6.5: *Depth images of a flower pot and a chair from two different views each, generated using the simple projection and Gaussian interpolation from Chapter 5 (above) and using our new depth image generator (below).*

depth, the resulting views will be rendered for a camera placed at $[0, 0, 1]$ and pointing at $[0, 0, 0]$, assuming the point cloud lies in $[-1, 1]^3$. This module was implemented as a custom layer in TensorFlow and CUDA, see Section 6.3.1 for implementation details. The gradients of this custom module are shown in Appendix A.4.

Figure 6.5 shows various depth images produced with our module and with the simple projection and Gaussian interpolation from Chapter 5. The results obtained with the extended interpolation better represent the depth of the object, especially featuring sharper edges at discontinuities in the depth field.

6.2.4 Image Based Classification

Our final module takes the K depth images generated by the previous layer and implements a classification network for images. In particular, we utilize K ResNet50 [He et al., 2015] architectures that share variables. Similar to MVCNN [Su et al., 2015], which deals with classifying images from different fixed views for 3D objects, we include a max pooling operation before regressing to a denser layer of classification logits. This allows the network to share variables between the ResNet50 architectures and thus learn features which require multiple images. We place the dense classification layer after the max pooling operation. For the ResNet50 architectures, we make use of

pre-trained weights as initialization, trained on ImageNet [Deng et al., 2009] dataset.

6.3 Results

6.3.1 Implementation, Parameters and Timing

In the view prediction component, we subsample original point clouds consisting of 2048 points to 256 points. All the batch normalization layers in PointNet were trained using a batch normalization decay of 0.9. In the depth image generation component, we set $\sigma = 2.0$, $\delta_1 = 1.4\sigma$, and $\delta_2 = I/12$, where I is the image size and is 229. The depth image generation layer was implemented as two native ops (gradient and forward pass) in TensorFlow using the CUDA backend. For the image based classification component, we used the preset values from the TensorFlow Slim library.

In order to train the network, we used the Adam optimizer with an initial learning rate of 0.0001 lowered every 50000 steps by a percentage of 5% (if the number of views $K = 1$), 10% (if $K = 2$), and 20% (if $K = 4$). The batch size varied between the models (128 for one view, 64 for two views and 32 for four views), due to memory reasons.

In our implementation tested on a GeForce GTX 1080 Ti graphics card, the forward-pass and backward-pass through the depth image generation layer take around 2 seconds (1.323 seconds for the forward-pass, 0.752 seconds for the backward-pass), using a batch size of 512, and projecting 2048 points to images of 229x229 pixels. In the complete graph, this corresponds to 10% of the computation time for the forward pass and 7% for the backward pass. The resulting training time of our single view architecture on randomly rotated point clouds is comparable to PointNet (about 8 hours). The training time increases with a sublinear dependency on the number of views, as the convergence is faster with multiple views.

6.3.2 Point Cloud Classification

We evaluate both PointNet and our network variations on the ModelNet40 [Wu et al., 2015b] benchmark for shape classification, composed of CAD models labelled in 40 classes and separated between training (9843 models) and testing (2468). For our method, we generated the point clouds from the shape dataset by uniformly sampling 2048 points on each model. For PointNet [Qi et al., 2016a], we used the most recent version of the original

	3D overall	y-Axis overall	3D class	y-Axis class
Ours, 1 View	0.854	0.873	0.815	0.828
Ours, 2 Views	0.869	0.884	0.829	0.851
Ours, 4 Views	**0.872**	0.885	**0.830**	0.856
PointNet	0.855	**0.892**	0.805	**0.862**

Table 6.1: *Classification results of our architecture with 1, 2 and 4 views, and the PointNet [Qi et al., 2016a] method, on a dataset augmented with random rotations (3D) and augmented only with random rotations around the vertical axis (y-Axis). Both the instance-based accuracy (overall) and the class average (class) are shown.*

implementation by the authors with the default settings (1024 points), as using 2048 points did not improve the results. Both for our method and for PointNet, we augment the training and testing models in the dataset by applying two different strategies: rotating the point clouds randomly in every direction, and rotating the point clouds randomly only around the vertical axis. Similarly to PointNet, in our training data we augment the point clouds by adding random Gaussian of noise of $\sigma = 0.001$. We let our method train until convergence, and present the average of the testing results from 5 evaluations.

In Table 6.1, we show the results of our classification using 1, 2 and 4 views and compare them to the state of the art PointNet [Qi et al., 2016a] point cloud classifier, for both datasets augmented with random rotations in every direction (3D) and the dataset augmented with random rotations only around the vertical axis (y-Axis). Both the instance-based accuracy (overall) and the class average accuracy (class) are presented. The best value for each column is highlighted.

The first three rows show how our architecture profits from estimating multiple depth images, thanks to our image based classification network that combines features from multiple views to optimize the outcome. Our best results are obtained using 4 views, for both the datasets.

While our 4 views architecture obtains slightly worse results than PointNet in the dataset with objects aligned along the vertical axis, our results vary less between the two datasets. In particular, in the more difficult dataset with randomly rotated objects, we outperform PointNet with our 2 and 4 views architectures, and obtain comparable results with a single view. Our intuition is that our generated depth images are more informative and result in less

	Learned	PCA
1 View	0.854	0.844
2/3 Views (Learned/PCA)	0.869	0.850

Table 6.2: *Instance-based classification results of our simpler PCA alternative with 1 and 3 views (PCA), compared to our original architecture results for 1 and 2 views on the dataset augmented with random rotations (Learned).*

source of confusion while learning. Image classification is a better studied problem than 3D point cloud classification, thus utilizing state of the art image classification networks (and their pre-trained weights on large datasets) on properly estimated depth images allows us to obtain better results, starting from the same raw input.

In our experiments, adding more views than 4 did not help improving the results. We believe that this limit is due to the sparsity of the point clouds, which do not contain very detailed features. Thus, our 4 views can already include the majority of the structure of the object representations in the dataset.

Note that all the recently proposed or concurrent techniques for classifying 3D points [Qi et al., 2016a; Qi et al., 2017b; Qi et al., 2016b; Simonovsky and Komodakis, 2017; Klokov and Lempitsky, 2017] present results obtained on the easier dataset, where the objects are aligned with the y-axis. The concurrent work PointNet++ [Qi et al., 2017b] obtains better classification results than the original PointNet by about 2%, while the volumetric variant of the recent work [Qi et al., 2016b] obtains results comparable to PointNet. As future work, it would be interesting to test if they generalize to the dataset with random rotations. Finally, there exist works which use 3D meshes instead of 3D point clouds and obtain better classification results. In [Su et al., 2015], the best results are obtained by rendering 80 views of 3D shape models; their views, though, are fixed and not learned, and their input 3D meshes are more detailed than our point clouds. Generating meshes from our sparse and noisy point clouds with triangulation or surface reconstruction methods would lead to meshes of significantly lower quality, leading to inferior results for mesh rendering based methods in this case. We refer to Appendix A.4 for more results.

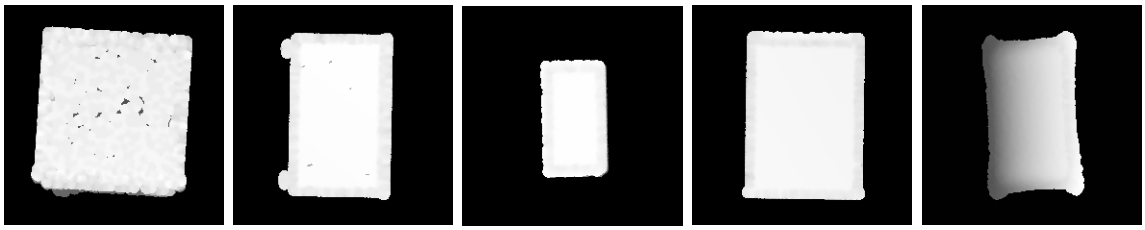

Figure 6.6: *Examples of images generated by our network with PCA.*

	3D Learned	y-Axis Learned	Random
1 View	0.854	0.873	0.849
2 Views	0.869	0.884	0.859
4 Views	0.872	0.885	0.868

Table 6.3: *Instance-based classification results of our simpler random views alternative (Random), compared to our original architecture results on the dataset augmented with random rotations (3D, Learned) and the dataset augmented with random rotations only around the y-Axis (y-Axis, Learned).*

6.3.3 Comparisons to Simpler Alternatives

In order to quantitativaly evaluate the impact of our automatic view estimation, we trained our architecture by substituting the view prediction module with a selection based on the PCA axes. We performed this experiment with a single view (projecting onto the plane spanned by the directions of the two largest PCA components), and 3 views (projecting on each plane spanned by the PCA directions). Notice that the obtained PCA views, thus the final accuracy, are equivalent in both the dataset augmented with random rotations and the dataset augmented with random rotations only around the y-Axis. In Table 6.2 we compare the obtained instance-based accuracy results (PCA) to the ones of our 1 and 2 views original architectures on the more difficult dataset with random rotations (Learned). One can notice how, already in the more difficult dataset, the PCA-based average accuracies are lower than our original 1 and 2 views results. In Figure 6.6, we show examples of PCA views of objects from different classes produced by the network, demonstrating how they are often ambiguous. Important features of the objects can indeed be hidden by their large surfaces, and, in case of isotropic point clouds, the PCA views are equivalent to random views (see Appendix A.4 for more examples).

Furthermore, we trained our architecture by substituting the view prediction

# Train.	87	104	173	163	149	64	124	197	240	88	231	138	167	680	124	475	128	200	149	392	200	200	90	103	106
± Acc.	-0.03	-0.02	-0.02	-0.02	-0.01	-0.01	-0.01	-0.01	-0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.04	0.05
	wardrobe	radio	bench	tent	laptop	bowl	stairs	car	plant	person	piano	curtain	cone	sofa	lamp	vase	sink	night stand	flower pot	table	desk	dresser	stool	xbox	bathtub

Table 6.4: *Difference of obtained accuracy between our original method and the random views alternative (± Acc.) and number of training examples (# Train.), per class.*

module with a random view selection, for 1, 2 and 4 views. In Table 6.3, we compare the obtained instance-based results (Random) with the ones of our original architectures in both datasets (3D and y-Axis, Learned). Notice that the results of the random views alternative are equivalent in both datasets, as the views are randomly sampled in the 3D space. The results obtained by our original architectures are better than the ones of random views for any number of views, especially for the dataset augmented with rotations around the y-Axis. For the more difficult dataset augmented with random rotations, the improvement given by our learned views is smaller. We believe that the gap could be larger if the ModelNet40 dataset would not present some commonly known limitations [Su et al., 2015; Arvind et al., 2017] such as ambiguities between pairs of classes (especially for low resolution inputs like ours) and classes with little training examples. In Table 6.4, we present the difference of obtained accuracy between our original method and the random views alternative (± Acc.), averaged between the 1, 2 and 4 views results, for the classes where the absolute difference was at least 1%. For each class, we additionally show the number of training examples (# Train.). The classes where the random views alternative performed better (i.e., where the values are smaller than zero) are often the ambiguous ones such as plant (confused with flower pot and vase), wardrobe (confused with bookshelf) and radio (confused with regular objects such as glass box), and have in general small number of training examples per class (on average 145, while the classes where our original method works better have on average 273 examples). We thus expect our method, based on learned views, to work best with a large number of training examples per class. We refer to Appendix A.4 for the individual accuracy results for 1, 2 and 4 views used for Table 6.4.

6.3.4 View Selection and Visualization

Our network outputs a set of views and corresponding depth images as additional information. The depth images generated represent informative 2D views of the 3D input, and can be utilized for applications such as visualizing

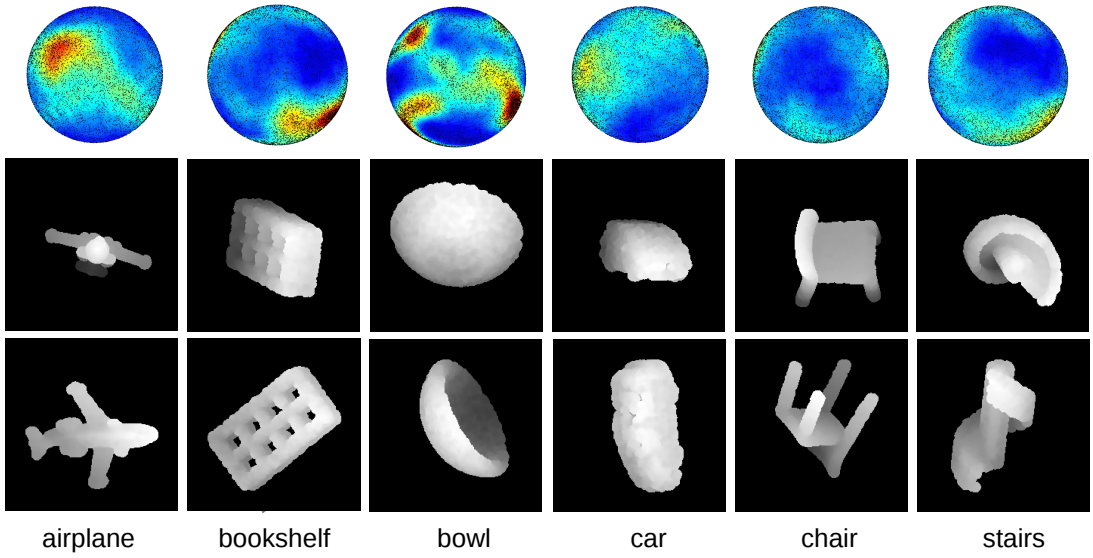

Figure 6.7: *Learned view density functions (top), depth images generated by our network corresponding to the least likely learned view (center), and the most likely learned view (bottom), for six testing point clouds.*

a point cloud. In order to visually evaluate the quality of our learned views, we feed a set of test point clouds to our single view architecture multiple times, always with a different random rotation, and plot the distribution of the learned views for each point cloud. Each point cloud is fed 10000 times to the network, and the resulting learned views, with respect to the original orientation of the point cloud, are sampled on a sphere.

In Figure 6.7 (top) the density of these learned views for six testing models are presented (red: high, blue:low). The density functions contain peaks and further regions with very low values. Hence, our views are optimized for different objects, and not random or constant. For most objects, multiple views can be considered appropriate for classification. Thus, we do not expect our density functions to present only a single sharp peak, but rather smoother regions of high values.

For each test point cloud, we sample the view corresponding to the highest value of the views density (i.e. the most likely view estimated by our network), and show the depth image generated by our network for that view in Figure 6.7 (bottom). Similarly, Figure 6.7 (center) presents the depth images corresponding to the view with the lowest probability. Our view estimation procedure outputs depth images that clearly expose distinguishing features, making the classes easily recognizable. For example, the legs of the chair, the

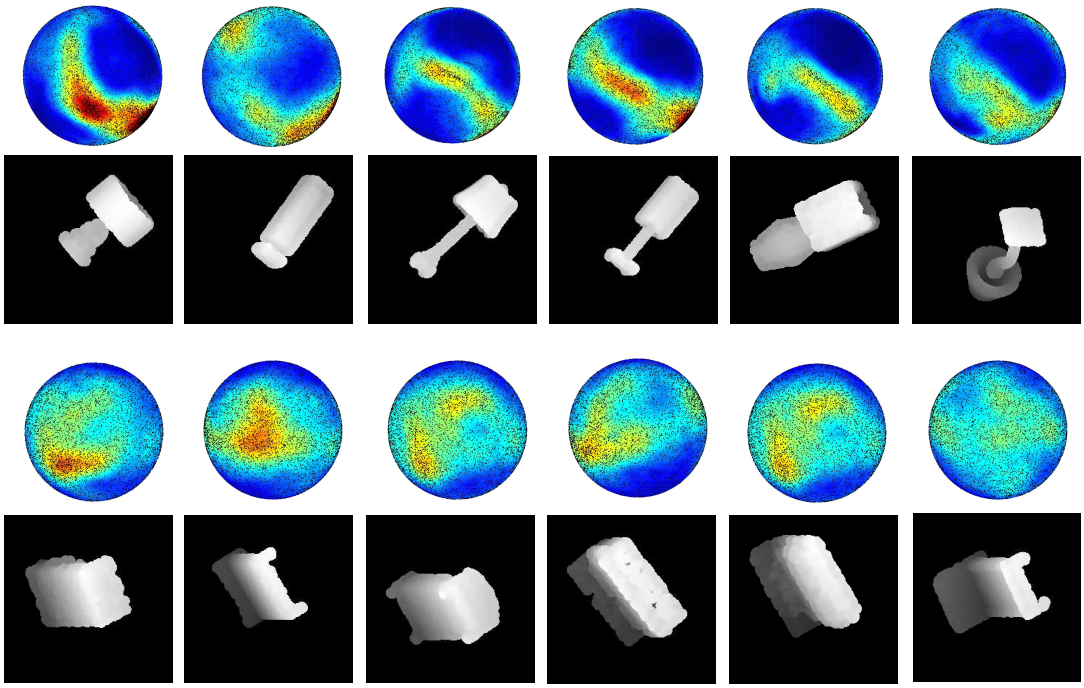

Figure 6.8: *Learned view density functions and the depth images corresponding to the view with highest probability for two classes of objects (lamps and beds).*

tail of the airplane, the internal structure of the bookshelf, and the borders of the bowl are visible from the learned views, while hidden for a view with low probability of selection. Similarly, the shapes of the stairs and the car are fully visible for the view with high probability of selection by our architecture, while only partially for a low probability view.

In Figure 6.8, further view density functions and depth images for the highest probability views are presented, for two classes of objects (lamps and beds). It is interesting to see how the learned views are similar for objects of the same class, but different for objects of different classes. This shows that the network specializes the views according to the classes.

We refer to Appendix A.4 for similar visualization results for the multiple views case.

Figure 6.9 shows examples of depth images generated by our network with two views, for different classes of objects. To generate these results, the test point clouds were fed to the network 10 times with random rotations, and the run which produced the highest single softmax prediction was considered. In most cases, our network predicts two views which complement each other, providing an even clearer overview of the point cloud compared to the views

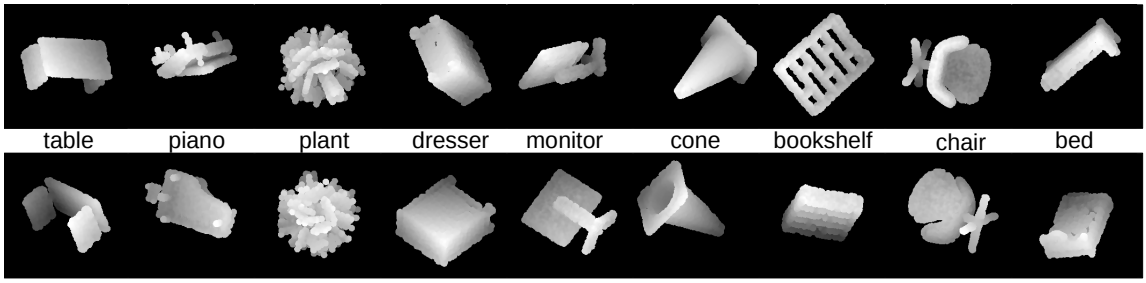

Figure 6.9: *Examples of generated depth images from the two views architecture. The two images (top and bottom) complement each other, providing a more informative overview of the object, as compared to a single view.*

given by our single view architecture. This is a consequence of the view pooling operation in our network, which combines features from different views. For example, the two depth images of the bed, piano, and bookshelf are a side view and a top/down view, the images of the chair show its front and back, and those of the cone contain the bottom hole and the pick at the top.

6.4 Discussion

We propose a novel neural network architecture for point cloud classification, which obtains results competitive with the state of the art for raw point clouds as the input. Our key idea is to automatically transform the input point cloud to one or more depth images, which can be combined and classified by a CNN classification module. The high performance of image based CNNs, and the large availability of data to train them, makes classification on our images better than considering only the 3D data.

In the future, we would like to explore further applications of this view learning and depth images generation approach. First of all, more experiments on point cloud visualization could be performed. Properly visualizing a point cloud is not a trivial task due to occlusions, lack of detailed features and sparse data, and we believe that our depth image generation method can be a useful representation. Another possible interesting extension would be to use our depth image generation layer as an autoencoder or for unsupervised learning. Finally, it can be adapted to generative adversarial networks (GANs) e.g. for point cloud segmentation.

CHAPTER 7

Conclusion

In this thesis, we presented novel data-driven approaches for tackling various tasks in point cloud processing. Namely, we worked on point cloud sampling for the generation of new geometry structures (represented by patterns or surfaces), on point cloud consolidation for the improvement of a given geometry, and on point cloud classification for analyzing the given geometry. While the synthesis methods were based on local patches, the last analysis classification problem considered global point clouds. We approached the tasks by relying on existing data in two different fashions: for repetitive structures, we designed the methods to exploit a matching between the geometry and one or a small set of input examples, while for more general and varying structures we resorted to large datasets to learn flexible outputs. The priors extracted from the utilized data allowed us to deal with more complex geometries than the previous methods, for example the continuous and discrete elements in Chapter 3, the patterns with adaptive correlations in Chapter 4, the very detailed consolidated structures in Chapter 5 and the multi-image representation for the geometries in Chapter 6.

First, in Chapter 3, we introduced an example-based sampling method to synthesize real-world, general repetitive structures represented by point samples. Instead of defining matching neighborhoods measures on the points themselves, we obtain smooth functions describing the input and output sampling, and formulate the matching as a smooth optimization problem. This permits us to produce an optimal sampling through an accurate control of the generation of new samples, as well as to handle, within the same

framework, complex structures of different nature: patterns of discrete elements, continuous surfaces as well as their mixtures. In addition to allow to synthesize general structures, the meshless representation makes our method suitable for interactive synthesis. We believe that generality and interactivity are two fundamental properties that make our approach suitable to the task of designing complex geometric structures, for modeling or artistic purposes.

In order to be able to deal with a class of more complex, natural-looking patterns, we extended the synthesis method to rely on multiple input examples, introducing the concept of adaptive correlation. We coupled our sampling approach with a framework for analyzing and synthesizing point distributions with locally varying correlations, which is based on the statistics of point processes. With this work, we unlocked the processing of a class of patterns that could not be handled by previous methods in the literature. Several applications of adaptive correlations are shown in Chapter 4, and we believe that they can serve as proofs of concept for the adoption of these locally varying patterns in problems spanning different areas of Computer Graphics, such as rendering and image compression.

In Chapter 5, we focused on the operation of consolidating a given input point cloud. Contrary to the geometries processed in the previous chapters, acquired geometry does not necessarily present repetitive structures, but rather varying general structures. We thus approached the problem by relying on large datasets composed of geometries with similar local neighborhoods, and developing a novel convolutional neural network architecture for local processing of point clouds. The key component of our deep learning method is a fully differentiable projector, which transforms the input points into a 2D image, allowing to exploit the strengths of CNNs for 2D image processing problems, and extending them to 3D unordered point clouds. For this reason, we believe that this component has potential to be adopted in many point clouds operations besides consolidation.

A further application of our deep learning projection technique is shown in Chapter 6. The module is extended to handle point clouds representing full objects, instead of projecting only local patches of points. A set of informative depth images of the input object is automatically generated, and used to classify the point cloud. Analysis operations of point clouds, such as their classification, are often to be performed in a global fashion, in order to extrapolate high level informations that the single patches could individually not provide. With this example application, we have shown that our deep learning technique is extendable to different scales of geometric structures.

7.1 Extensions of Our Techniques

We tackled the described problems by processing either local patches of points (Chapters 3, 4, and 5), or the global point clouds (Chapter 6). We believe that all our techniques could benefit from a mixed approach, where patches of different sizes are considered in combination. The larger patches would allow us to exploit more global information about the geometry, while the smaller ones would optimize for the details. In particular, our example-based synthesis technique presented in Chapters 3 and 4 could be directly extended to support two (or more) neighborhood sizes. The matching similarity between the output geometry and the input example would be expressed in terms of both the neighborhood sizes, and a total energy accounting for both the neighborhood sizes would be minimized. In this way, example structures with details of different scales of repetition (Figure 3.1, right) could be better reproduced, by assigning the points belonging to the large scale repetitions to the large neighborhood and the ones responsible for the small scale repetitions to the smaller neighborhood. Correspondingly, our deep learning architectures presented in Chapters 5 and 6 could be adapted to multi-size patches, by producing multiple images of the geometry to process details of different levels. The images representing the larger patches would infer semantic information about the objects being processed, while the images representing the smaller patches would process their details. For consolidation, an input patch could be transformed by considering both a large patch and a small one around it. The oriented heightmap resulting from the large patch would allow to produce a coarse output defining the global shape of the object (solving existing challenges such as covering holes), while the heightmap resulting from the small patch would add the details to the geometry. For classification of point clouds, in addition to learn a rotation for every view, a learnable zooming factor could be introduced, to allow the generation of depth images of details of the objects, which could help classifying them.

Another possible future direction to explore is the processing of non-manifold structures and other more complex geometries like multiple close-by sheets. Many complex geometries from the real world and existing 3D data (like the CAD models) include such patches, thus it would be beneficial to extend our methods to handle these situations. While some of the results showed in Chapter 3 present details with intersecting structures (such as the leaves and the stem in Figure 3.1, right), and our classification network architecture is designed to handle structures composed of multiple layers of depth, our patch-based synthesis and consolidation approaches rely on the assumption that the underlying large-scale geometry lay on manifolds. By first segment-

ing the points in a patch depending on which manifold structure they belong to, and then processing each group of points separately (through minimization of the similarity function or denoising of the generated heightmap), both methods could be extended to handle patches with multiple intersecting or close-by manifolds.

Finally, we believe that there are interesting potential improvements in the way we rely on data in our techniques. Our example-based synthesis methods have resulted to be logical approaches for dealing with repetitive structures, by exploiting the repetitions existing in the input sample. It is not completely clear, though, what output to expect in case the input sample does not contain enough repetitions, as well as what the ideal size of the input is. We believe that a necessary future step is to provide the users with methods to design proper inputs, such as validation measures or the generation of fast, interactive previews of the output corresponding to the given input. As for our techniques based on deep learning, we regard the improvement and tuning of the intrinsically generated 2D images representing the patches as a main future research path. More detailed images of the geometry would certainly lead to better results in the tackled operations. A promising approach could be the generation of more accurate consolidated heightmaps, and the adoption of generative adversarial networks (GANs), which have demonstrated the ability to produce better details-preserving images in many image processing tasks. Their unsupervised training strategies are an additional attractive property, which would ease the data collection process.

A.1 Discrete Similarity Measure

A.1.1 Deriving the Discrete Similarity Measure

For the derivations in this section, we denote dot product of two vectors with $\mathbf{xy}$, and the squared norm of a vector with $\mathbf{x}^2$ for brevity. If we substitute the sums of Gaussians in Equation 3.2, we obtain:

$$
\begin{aligned}
&S(\mathbf{f}(\mathbf{x}),\mathbf{e}(\mathbf{m}(\mathbf{x}))) = \\
&= \int \left| \sum_i \mathbf{a}_i g(\mathbf{x}+\mathbf{s}-\mathbf{x}_i,\sigma) - \sum_i \mathbf{b}_i g(\mathbf{m}(\mathbf{x})+\mathbf{s}-\mathbf{e}_i,\sigma) \right|^2 g(\mathbf{s},\delta)\mathbf{ds} \qquad \text{(A.1)} \\
&= \int \sum_{ij} (\mathbf{a}_i\mathbf{a}_j) g(\mathbf{x}+\mathbf{s}-\mathbf{x}_i,\sigma) g(\mathbf{x}+\mathbf{s}-\mathbf{x}_j,\sigma) g(\mathbf{s},\delta) \\
&\quad - 2\sum_{ij} (\mathbf{a}_i\mathbf{b}_j) g(\mathbf{x}+\mathbf{s}-\mathbf{x}_i,\sigma) g(\mathbf{m}(\mathbf{x})+\mathbf{s}-\mathbf{e}_j,\sigma) g(\mathbf{s},\delta) \\
&\quad + \sum_{ij} (\mathbf{b}_i\mathbf{b}_j) g(\mathbf{m}(\mathbf{x})+\mathbf{s}-\mathbf{e}_i,\sigma) g(\mathbf{m}(\mathbf{x})+\mathbf{s}-\mathbf{e}_j,\sigma) g(\mathbf{s},\delta)\mathbf{ds} \qquad \text{(A.2)}
\end{aligned}
$$

The products of Gaussians lead to the following form:

$$
\begin{aligned}
&S(\mathbf{f}(\mathbf{x}),\mathbf{e}(\mathbf{m}(\mathbf{x}))) = \\
&= \int \sum_{ij} (\mathbf{a}_i\mathbf{a}_j) \exp\left(\frac{-2-\frac{\sigma}{\delta}^2}{\sigma^2}\mathbf{s}^2 + \frac{-4\mathbf{x}+2\mathbf{x_i}+2\mathbf{x_j}}{\sigma^2}\mathbf{s} + \frac{-2\mathbf{x}^2-\mathbf{x_i}^2-\mathbf{x_j}^2+2\mathbf{x}\mathbf{x_i}+2\mathbf{x}\mathbf{x_j}}{\sigma^2} \right) \\
&\quad - 2\sum_{ij} (\mathbf{a}_i\mathbf{b}_j) \exp\left(\frac{-2-\frac{\sigma}{\delta}^2}{\sigma^2}\mathbf{s}^2 + \frac{-2\mathbf{x}-2\mathbf{m}(\mathbf{x})+2\mathbf{x_i}+2\mathbf{e_j}}{\sigma^2}\mathbf{s} + \frac{-\mathbf{x}^2-\mathbf{m}(\mathbf{x})^2-\mathbf{x_i}^2-\mathbf{e_j}^2+2\mathbf{x}\mathbf{x_i}+2\mathbf{x}\mathbf{e_j}}{\sigma^2} \right) \\
&\quad + \sum_{ij} (\mathbf{b}_i\mathbf{b}_j) \exp\left(\frac{-2-\frac{\sigma}{\delta}^2}{\sigma^2}\mathbf{s}^2 + \frac{-4\mathbf{m}(\mathbf{x})+2\mathbf{e_i}+2\mathbf{e_j}}{\sigma^2}\mathbf{s} + \frac{-2\mathbf{m}(\mathbf{x})^2-\mathbf{e_i}^2-\mathbf{e_j}^2+2\mathbf{m}(\mathbf{x})\mathbf{e_i}+2\mathbf{m}(\mathbf{x})\mathbf{e_j}}{\sigma^2} \right)\mathbf{ds}
\end{aligned} \qquad \text{(A.3)}
$$

We now use the integral form of a Gaussian function:

$$
\int k\exp(-f\mathbf{x}^2+g\mathbf{x}+h)d\mathbf{x} = k\sqrt{\frac{\pi}{f}}\exp(\frac{g^2}{4f}+h) \qquad \text{(A.4)}
$$

Thus:

$$
\begin{aligned}
&S(\mathbf{f}(\mathbf{x}),\mathbf{e}(\mathbf{m}(\mathbf{x}))) = \\
&= c'' \Big[\sum_{ij} (\mathbf{a}_i\mathbf{a}_j) \exp\left(\frac{-(\mathbf{x_i}-\mathbf{x_j})^2 - (\frac{\sigma}{\delta})^2(\mathbf{x}-\mathbf{x_i})^2 - (\frac{\sigma}{\delta})^2(\mathbf{x}-\mathbf{x_j})^2}{2\sigma^2+\frac{\sigma^4}{\delta^2}} \right) \\
&\quad - 2\sum_{ij} (\mathbf{a}_i\mathbf{b}_j) \exp\left(\frac{-((\mathbf{m}(\mathbf{x})-\mathbf{e_i})-(\mathbf{x}-\mathbf{x_j}))^2 - (\frac{\sigma}{\delta})^2(\mathbf{m}(\mathbf{x})-\mathbf{e_i})^2 - (\frac{\sigma}{\delta})^2(\mathbf{x}-\mathbf{x_j})^2}{2\sigma^2+\frac{\sigma^4}{\delta^2}} \right) \\
&\quad + \sum_{ij} (\mathbf{b}_i\mathbf{b}_j) \exp\left(\frac{-(\mathbf{e_i}-\mathbf{e_j})^2 - (\frac{\sigma}{\delta})^2(\mathbf{m}(\mathbf{x})-\mathbf{e_i})^2 - (\frac{\sigma}{\delta})^2(\mathbf{m}(\mathbf{x})-\mathbf{e_j})^2}{2\sigma^2+\frac{\sigma^4}{\delta^2}} \right) \Big]
\end{aligned} \qquad \text{(A.5)}
$$

where $c'' = \sqrt{\frac{\pi\sigma^2}{2+(\frac{\sigma}{\delta})^2}}$

Finally, the following form is reached:

$$\begin{aligned} & S(\mathbf{f}(\mathbf{x}), \mathbf{e}(\mathbf{m}(\mathbf{x}))) = \\ =& c' \Big(\sum_{ij} (\mathbf{a_i}\mathbf{a_j}) g(\mathbf{x_i} - \mathbf{x_j}, \sigma c) g(\mathbf{x} - \mathbf{x_i}, \delta c) g(\mathbf{x} - \mathbf{x_j}, \delta c) \\ & - 2 \sum_{ij} (\mathbf{a_i}\mathbf{b_j}) g((\mathbf{m}(\mathbf{x}) - \mathbf{e_i}) - (\mathbf{x} - \mathbf{x_j}), \sigma c) g(\mathbf{m}(\mathbf{x}) - \mathbf{e_i}, \delta c) g(\mathbf{x} - \mathbf{x_j}, \delta c) \\ & + \sum_{ij} (\mathbf{b_i}\mathbf{b_j}) g(\mathbf{e_i} - \mathbf{e_j}, \sigma c) g(\mathbf{m}(\mathbf{x}) - \mathbf{e_i}, \delta c) g(\mathbf{m}(\mathbf{x}) - \mathbf{e_j}, \delta c) \Big) \end{aligned} \tag{A.6}$$

where $c = \sqrt{2 + (\frac{\sigma}{\delta})^2}$ and $c' = \frac{\sqrt{\pi\sigma^2}}{c}$.

Equation A.6 can be used as the similarity measure and its gradients can be computed. In practice, we propose a slight adaptation which allows us to formulate the discrete similarity measure in a more compact form, and results in a negligible change in the similarity and, thus, does not influence the results. Instead of using a Gaussian for the window function, one can use a box function with size δ. Then, Equation A.1 can be approximated with:

$$S(\mathbf{f}(\mathbf{x}), \mathbf{e}(\mathbf{m}(\mathbf{x}))) = \int \left| \sum_i \mathbf{a}_i g(\mathbf{x} + \mathbf{s} - \mathbf{x}_i, \sigma) - \sum_i \mathbf{b}_i g(\mathbf{m}(\mathbf{x}) + \mathbf{s} - \mathbf{e}_i, \sigma) \right|^2 \mathbf{ds}, \tag{A.7}$$

where the contributions of some points are canceled: only the $\mathbf{x_i}$ closer than δ to $\mathbf{x}$ and the $\mathbf{e_i}$ closer than δ to $\mathbf{m}(\mathbf{x})$ are considered. Following the steps above, the following final form is reached:

$$S(\mathbf{f}(\mathbf{x}), \mathbf{e}(\mathbf{m}(\mathbf{x}))) = \frac{\sqrt{\pi\sigma^2}}{2} \Big(\sum_{ij} (\mathbf{a_i}\mathbf{a_j}) g(\mathbf{x_i} - \mathbf{x_j}, \sqrt{2}\sigma) - 2 \sum_{ij} (\mathbf{a_i}\mathbf{b_j}) g((\mathbf{m}(\mathbf{x}) - \mathbf{e_i}) - (\mathbf{x} - \mathbf{x_j}), \sqrt{2}\sigma) + \sum_{ij} (\mathbf{b_i}\mathbf{b_j}) g(\mathbf{e_i} - \mathbf{e_j}, \sqrt{2}\sigma) \Big) \tag{A.8}$$

Notice that the integral form of a Gaussian function can still be utilized to reach the final form, as only the window function is transformed into a box function, but not the Gaussian functions placed at the point locations.

A.1.2 Computing the Gradients

We show the gradients for the more compact variant which uses a box function.

$$\frac{\partial T}{\partial \mathbf{m_k}} = -\frac{\sqrt{2\pi}}{\sigma} \sum_{ij} (\mathbf{a_i}\mathbf{b_j}) g((\mathbf{m_k} - \mathbf{e_i}) - (\mathbf{q_k} - \mathbf{x_j}), \sqrt{2}\sigma)((\mathbf{m_k} - \mathbf{e_i}) - (\mathbf{q_k} - \mathbf{x_j})) \tag{A.9}$$

$$\frac{\partial T}{\partial \mathbf{x_i}} = -\frac{\sqrt{2\pi}}{\sigma} \sum_k \left(\sum_j (\mathbf{a}_i \mathbf{b}_j) g((\mathbf{m_k} - \mathbf{e_j}) - (\mathbf{q_k} - \mathbf{x_i}), \sqrt{2}\sigma)((\mathbf{m_k} - \mathbf{e_j}) - (\mathbf{q_k} - \mathbf{x_i})) + \sum_j (\mathbf{a}_i \mathbf{a}_j) g(\mathbf{x_j} - \mathbf{x_i}, \sqrt{2}\sigma)(\mathbf{x_j} - \mathbf{x_i}) \right) \tag{A.10}$$

$$\frac{\partial T}{\partial \mathbf{a_i}} = \frac{\sqrt{\pi\sigma^2}}{2} \sum_k \left(\sum_j \mathbf{a}_j g(\mathbf{x_i} - \mathbf{x_j}, \sqrt{2}\sigma) - 2 \sum_j \mathbf{b}_j g((\mathbf{m_k} - \mathbf{e_i}) - (\mathbf{x_k} - \mathbf{x_j}), \sqrt{2}\sigma) \right) \tag{A.11}$$

where only input points $\mathbf{e_i}$ and output points $\mathbf{x_i}$ close enough to $\mathbf{m_k}$ and $\mathbf{q_k}$ for the current k are considered.

A.2 Analysis of General Sampling Patterns

In this appendix, we present a theory of general point distributions in terms of the product densities of underlying point processes. We show how these can be estimated from example distributions and interpreted. We establish how this theory relates to and extends existing techniques that assume a stationary point process, and propose a measure of stationarity.

A.2.1 Campbell's theorem

Campbell's theorem relates sums of values of functions at the point locations generated by a point process, to the integrals of those functions and the product densities. For our methods, the following two special cases of the theorem will be important,

$$\mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i \in \mathbf{X}} f(\mathbf{x}_i) = \int_{\mathbb{R}^d} f(\mathbf{x}) \lambda(\mathbf{x}) d\mathbf{x}, \tag{A.12}$$

$$\mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}}^{\neq} f(\mathbf{x}_i, \mathbf{x}_j) = \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} f(\mathbf{x}, \mathbf{y}) \rho(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}, \tag{A.13}$$

for any positive function f, where $\sum^{\neq}_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}}$ considers only different positions. We will start from these expressions to derive our estimators in the next section. For brevity, we drop the integration domains where clear.

A.2.2 Estimating Product Densities

Our analysis relies on estimating the statistics $\lambda(\mathbf{x})$ and $\rho(\mathbf{x}, \mathbf{y})$ as defined in the previous section. These two statistics define *all* point patterns under the common assumption of Gaussianity. The density measure $\lambda(\mathbf{x})$ can be easily estimated using standard techniques such as kernel density estimation. The estimation of $\rho(\mathbf{x}, \mathbf{y})$ has so far not been considered except for the case of stationary or isotropic correlation models. We derive a reliable estimator that converges to ρ, and show that it can be interpreted as a density estimator in the higher dimensional space of aggregated point coordinates.

Given multiple distributions generated by a point process, we would like to design unbiased and low variance estimators for the product densities. We start with deriving an estimator for the intensity based on the Campbell's theorem and show how it naturally extends to second order product density ρ. We assume a monotonically decreasing positive kernel function k for the estimators. In this work, we assume a Gaussian kernel with

$k_i(\mathbf{x}) := k(\mathbf{x}, \mathbf{x}_i) := \frac{1}{(\sqrt{\pi}\sigma)^d} e^{-||\mathbf{x}-\mathbf{x}_i||^2/\sigma^2}$. If we plug this into the expression into Equation A.12, we get

$$\mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i \in \mathbf{X}} k(\mathbf{x}, \mathbf{x}_i) = \int k(\mathbf{x}, \mathbf{y}) \lambda(\mathbf{y}) d\mathbf{y}. \tag{A.14}$$

The convolution on the right-hand side converges to $\lambda(\mathbf{x})$ as $\sigma \to 0$. The left-hand side is the classical non-parametric density estimator. This estimator will get better as we get more instances of the point pattern.

Utilizing Campbell's theorem for higher order statistics, we can generalize this result. Specifically, for any two distinct point locations $\mathbf{x}_i$ and $\mathbf{x}_j$, if we plug $k(\mathbf{x}, \mathbf{x}_i)k(\mathbf{y}, \mathbf{x}_j)$ into Equation A.13, we obtain

$$\mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}}^{\neq} k_i(\mathbf{x}) k_j(\mathbf{y}) = \int \int k(\mathbf{x}, \mathbf{z}) k(\mathbf{y}, \mathbf{t}) \rho(\mathbf{z}, \mathbf{t}) d\mathbf{z} d\mathbf{t}. \tag{A.15}$$

Analogous to the estimator for $\lambda(\mathbf{x})$, the right-hand side converges to $\rho(\mathbf{x}, \mathbf{y})$ as $\sigma \to 0$, and the left-hand side gives us the estimator for ρ. Hence, we define the estimator for ρ as

$$\boxed{\hat{\rho}(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}}^{\neq} k_i(\mathbf{x}) k_j(\mathbf{y}).} \tag{A.16}$$

This estimator can be computed for each $\mathbf{x}$ and $\mathbf{y}$ by averaging over different distributions. Note that this is a $2d$-dimensional statistic, to fully capture the rich correlation structure of the underlying point process.

Interpretation as a density estimator Since we assume Gaussian kernels, we can combine the two kernels in Equation A.16 to obtain a single Gaussian that depends on the distance between the $2d$-dimensional vectors $[\mathbf{x}^T \mathbf{y}^T]^T$, and $[\mathbf{x}_i^T \mathbf{x}_j^T]^T$. Hence, this estimator can be considered as a non-parametric density estimator in the product space $\mathbb{R} \times \mathbb{R}$ for the points generated by aggregating all combinations of different points in the original point pattern.

Figure A.1 shows plots of ρ for different 1D distributions. For the regular grid, we get a regular grid again since the density in the product space is regular. For an infinite regular grid with random translation, we get lines parallel to the $x = y$ line as shown in Figure A.1 (b). This is due to the correlated translation of sampling points, i.e. for each realization of this point process, the point locations are $x + t$ and $y + t$ for a random translation t. If this translation is independent for each sampling point, we get a jittered distribution [Mitchell, 1996], and the lines start to become blurred due to

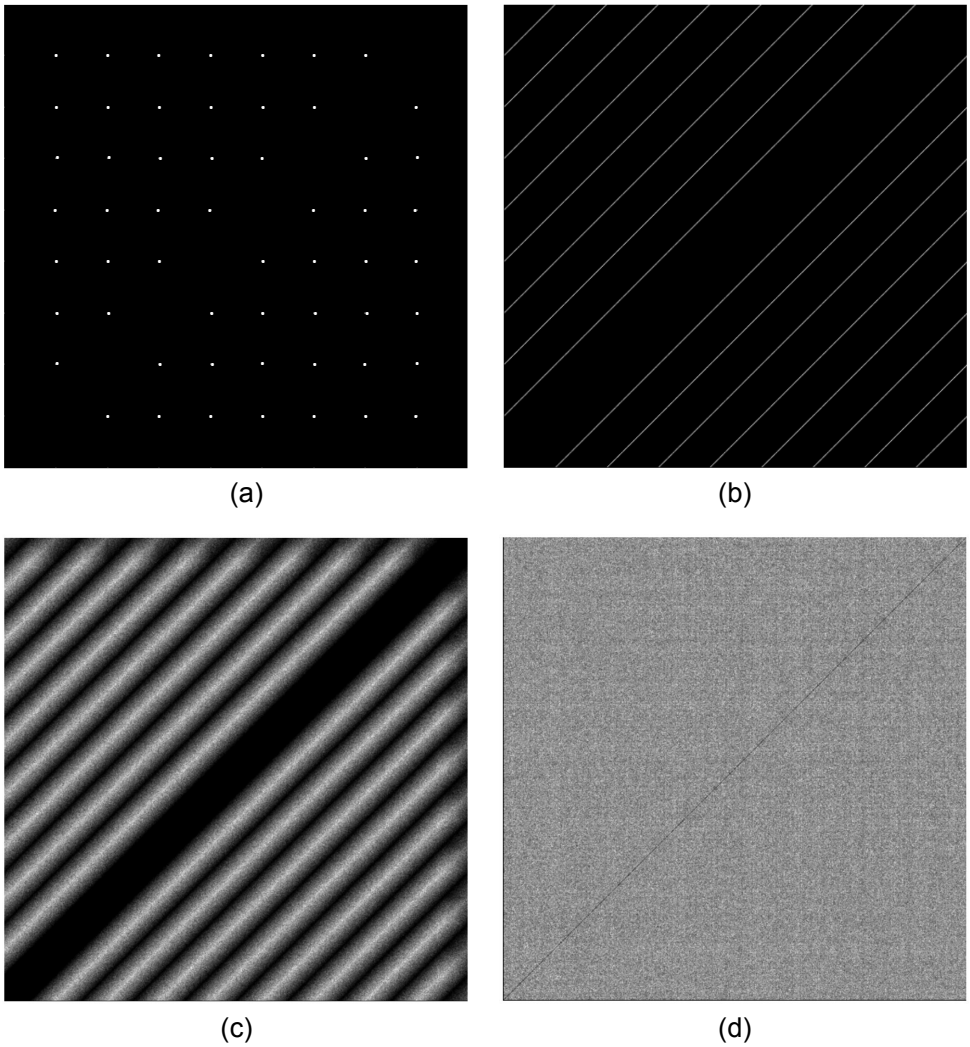

Figure A.1: *Plot of ρ for different 1D distributions: regular grid (a), infinite regular grid with random translation (b), jittered distribution (c), random distribution (d).*

the loss of correlations (Figure A.1 (c). Finally, for a completely random distribution, we get a constant ρ (Figure A.1, d).

Relation to previous analysis techniques The ρ reduces to previously used statistics for the special case of stationary distributions. For these distributions, $\rho(\mathbf{x}, \mathbf{y}) = \rho(\mathbf{x} - \mathbf{y})$ is constant for all $\mathbf{x}$ and $\mathbf{y}$ with $\mathbf{y} = \mathbf{x} + \mathbf{h}$ for a constant $\mathbf{h}$. Then, we can integrate over all such $\mathbf{x}$ and $\mathbf{y}$ pairs to get an

estimator of $\rho(\mathbf{h})$ for stationary distributions as follows

$$\begin{aligned}
&\int \mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}}^{\neq} k_i(\mathbf{x}) k_j(\mathbf{x}+\mathbf{h}) d\mathbf{x} d\mathbf{y} \\
=&\mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}}^{\neq} \int k(\mathbf{x}-\mathbf{x}_i) k(\mathbf{x}-\mathbf{x}_j+\mathbf{h}) d\mathbf{x} d\mathbf{y} \\
=&\mathbb{E}_{\mathbf{X}} \sum_{\mathbf{x}_i,\mathbf{x}_j \in \mathbf{X}}^{\neq} k'(\mathbf{h}-\mathbf{h}_{ij}),
\end{aligned} \tag{A.17}$$

with $\mathbf{h}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ and k' is a Gaussian with standard deviation 2σ. This results in the estimator for stationary distributions proposed by Wei et al. [Wei and Wang, 2011]. Similarly, if we integrate for constant $||\mathbf{h}||$, we can recover the estimator for isotropic point processes in the work by Öztireli et al. [Öztireli and Gross, 2012]. Since the ρ is related to periodogram with a Fourier transform [Heck et al., 2013], spectral analysis can be similarly treated in the same framework for stationary distributions. Note that for the distributions with adaptive density, previous works alter the difference or distance metric used, but still assume that the underlying correlation model is fixed by a single translation invariant ρ. The only method that does not operate under the invariance assumption is proposed by Subr and Kautz [Subr and Kautz, 2013], where they utilize the Fourier spectrum instead of the phaseless power spectrum. However, spatially adaptive correlations can be mixed into different frequencies in the spectral domain, and thus it is not clear how to interpret and utilize the resulting diagrams for adaptive correlations.

A measure of stationarity The above observation can be utilized to devise a test of stationarity for general distributions. To normalize with respect to density, we work with the pair correlation function (PCF) $g(\mathbf{x},\mathbf{y}) := \rho(\mathbf{x},\mathbf{y})/\lambda(\mathbf{x})\lambda(\mathbf{y})$. We expect $g(\mathbf{x},\mathbf{x}+\mathbf{h})$ to be constant for a given $\mathbf{h}$. Hence, we can define the variance of g at this $\mathbf{h}$ as a measure of stationarity.

$$\boxed{\hat{s}(\mathbf{h}) = \frac{1}{|V|}\int_V \hat{g}^2(\mathbf{x},\mathbf{x}+\mathbf{h}) d\mathbf{x} - \left[\frac{1}{|V|}\int_V \hat{g}(\mathbf{x},\mathbf{x}+\mathbf{h}) d\mathbf{x}\right]^2,} \tag{A.18}$$

for a given domain V and its volume $|V|$. In the product space, this means that we are measuring variance of the values of g along lines parallel to the $\mathbf{x} = \mathbf{y}$ line. For stationary distributions such as the uniformly translated grid in Figure A.1 (b) and random distribution in Figure A.1 (d), we get low variance along these lines. We further plot variance graphs for some 2D distributions in Figure A.2. As expected, stationary distributions result in much less variance.

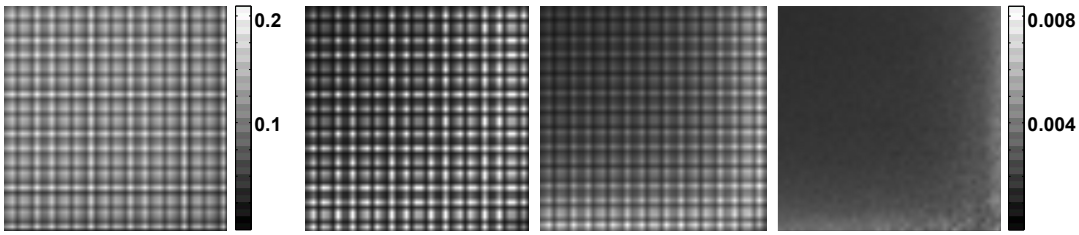

Figure A.2: *Variance graph for some 2D distributions: regular grid (a), infinite regular grid with random translation (b), jittered distribution (c), random distribution (d).*

A.3 Analysis and Synthesis with Local Anisotropy

A.3.1 Analysis

The matrix $\mathbf{M}_x$ is a measure of anisotropy. We utilize 2D anisotropy measures from the spatial point processes literature [Illian et al., 2008] to compute $\mathbf{M}_x$. This method performs a simple density estimation on the difference vectors $\mathbf{h}_{ij}$. First, a radial histogram of the vectors $\mathbf{h}_{ij}$ is computed. Each bin thus corresponds to a direction on the unit hypersphere and a distance from its center. The dominant anisotropy is then extracted by picking the directions with the smallest and largest bin sums. We then form a covariance matrix $\mathbf{C}_x$ with two eigenvectors set as these directions, and the eigenvalues as the corresponding bin values. Finally, the matrix $\mathbf{M}_x$ is set as the whitening transform $\mathbf{M}_x = \mathbf{C}_x^{-1/2}$ such that the resulting difference vectors $\mathbf{h}_{ij}$ are distributed isotropically. Note that when we apply the transform $\mathbf{M}_x$, we assume that the volume $|\mathcal{N}_x|$ also scales so as to keep the intensity at λ_x. This step is designed for point distributions on two dimensional domains such as the 2D plane or two-manifold surfaces in 3D, or with limited number of anisotropy directions in higher dimensional domains.

A.3.2 Synthesis

Before feeding into the estimator, the difference vectors $\mathbf{h}_{ij}$ are first whitened with $\mathbf{M}_k\mathbf{h}_{ij}$. The gradient $\frac{\partial}{\partial \mathbf{x}_i}E_k$ is then computed, multiplied with $\mathbf{M}_k^{-1}$, and summed over all neighborhoods to get the final gradient $\Delta_i = \sum_k \mathbf{M}_k \frac{\partial}{\partial \mathbf{x}_i}E_k$ for point $\mathbf{x}_i$.

A.4 View Selection, Comparisons and Gradients

A.4.1 View Selection for Our 2 Views Architecture

Similar to Figure 6.7, we show in Figure A.3 (top) the density of the learned views for six testing models, this time for the case of our 2 views architecture. The first row is the density of the first view, and the second row the one of the second view. Again, the density functions contain peaks and further regions with very low values, optimized for different objects. Most interestingly, one can also notice how the density functions of the two views are complementary to each other: the regions of high values in the first view usually have low values in the second view, and the other way around. This demonstrates how our network chooses different view points to combine their features and optimize the results.

For each test point cloud, we sample the views corresponding to the highest value of the views densities (i.e. the most likely views estimated by our network), and show the depth images generated by our network for those views in Figure A.3 (bottom, first row for the first view and second row for the second view). It is noticeable how the views complement each other, by showing different features of the objects. For example, one can see the side of the cone and its bottom part, the side of the cup and its inside, the top of the bench and its bottom part.

Like in Figure 6.8, in Figure A.4 we show further view density functions and depth images for the highest probability views, for a class of objects (lamps) using our 2 views architecture. Again, the pairs of learned views are similar for objects of the same class, demonstrating how our network specializes to the different objects. It is also clear how the views are complementary to each other. In particular, the first view tends to show the lamps from the side, and the second view often adds a shifted view point.

All the view density functions are spheres rendered from a fixed arbitrary point view.

A.4.2 Comparison with PCA

In Figure A.5 we show a comparison of our selected views to views selected by exploiting the PCA components. The three objects are indistinguishable from the PCA views, while easily recognizable with our method. Both the generated depth images and the point clouds rendered from the view points are shown.

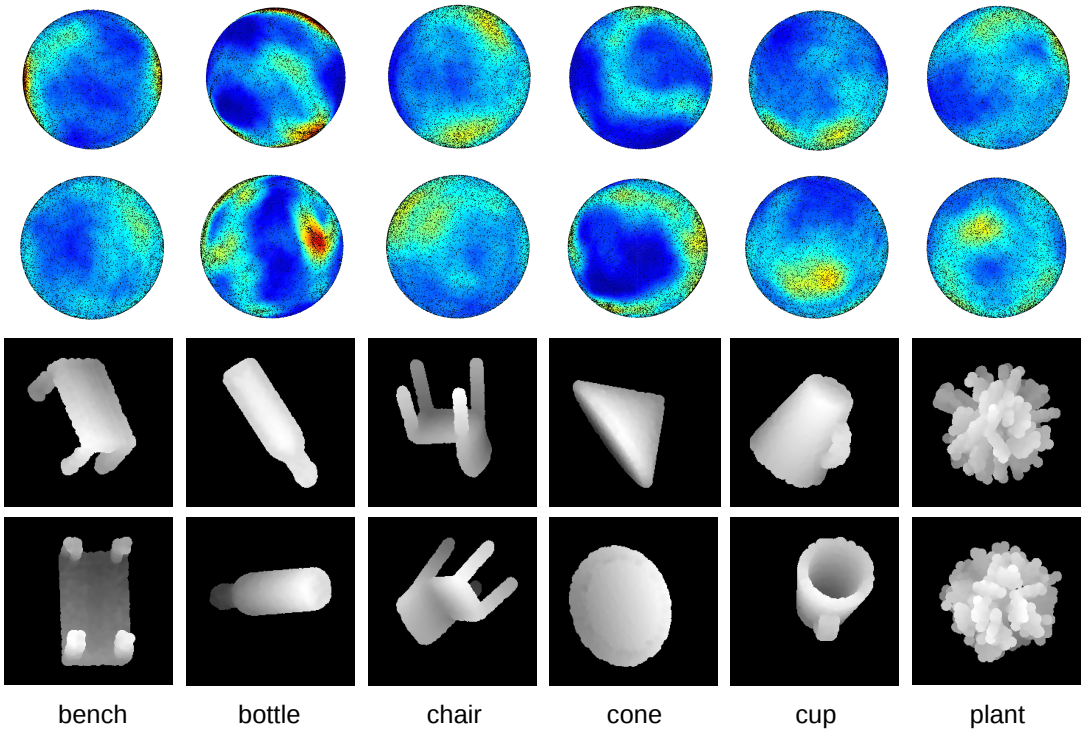

Figure A.3: *Learned view density functions (top, first row for the first view and second row for the second view), depth images generated by our network corresponding to the most likely learned views (bottom, first row for the first view and second row for the second view), for our 2 views architecture.*

A.4.3 Failure Cases and Comparison with Meshes

Due to the low resolution of the input point cloud and the ambiguity of some models in the dataset, there exists a subset of classes which are hard to classify. In Figure A.6, three objects from these classes are shown. Like PointNet and the other most recent papers that handle point clouds, our method fails to properly classify them, as the correspondent generated depth images are ambiguous. The very detailed meshes used in MVCNN [Su et al., 2015] include more features (because of their high resolution), and we believe this is the reason for their higher accuracy numbers compared to point-based methods. In Figure A.7, we compare the original high resolution meshes of two models with the meshes obtained by reconstructing a point cloud of the objects sampled with the same number of points that we use (2048), using the commonly adopted Poisson reconstruction method with optimal parameters. As one can see, the reconstructed meshes (on the right) contain way less details (XBox), and, sometimes, the reconstruction even fails to

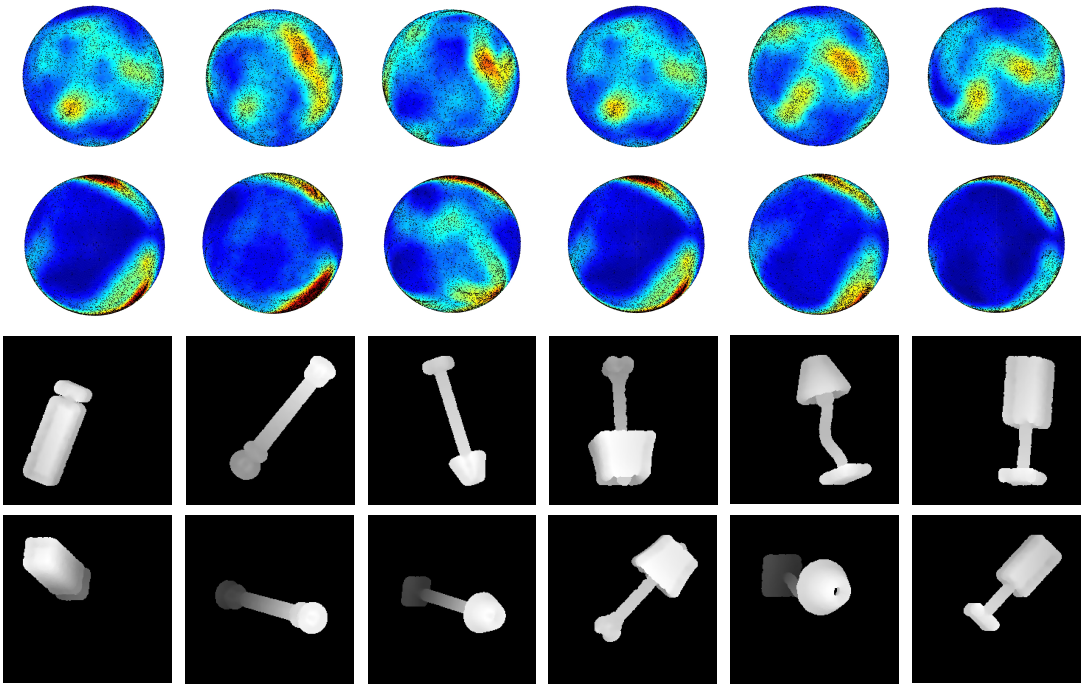

Figure A.4: *Learned view density functions and the depth images corresponding to the views with highest probability for a class of objects (lamps), for our 2 views architecture. The first row of the density functions and of the depth images corresponds to the first view, and the second row to the second view.*

correctly represent the shape of the object (plant). This shows how generating meshes from our sparse point clouds would not facilitate the problem.

A.4.4 Comparison with Random Views Alternative

In Table A.1, Table A.2, and Table A.3, we present the difference of obtained accuracy between our original method and the random views alternative (± Acc.), for 1, 2 and 4 views respectively, for the classes where the absolute difference was at least 1%. The numbers from 5 evaluations were averaged for

± Acc.	-0.06	-0.03	-0.03	-0.02	-0.02	-0.02	-0.02	-0.02	-0.01	-0.01	-0.01	-0.01	-0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.03	0.03	0.03	0.04	0.05
	bowl	car	person	**bench**	**radio**	range hood	laptop	stairs	cone	sink	cup	curtain	glass box	**piano**	**vase**	toilet	sofa	bookshelf	bed	tv stand	lamp	**desk**	door	**table**	plant	keyboard	**bathtub**	**dresser**	**night stand**	stool	flower pot

Table A.1: *Difference of obtained accuracy between our original method and the random views alternative (± Acc.), for 1 view, per class.*

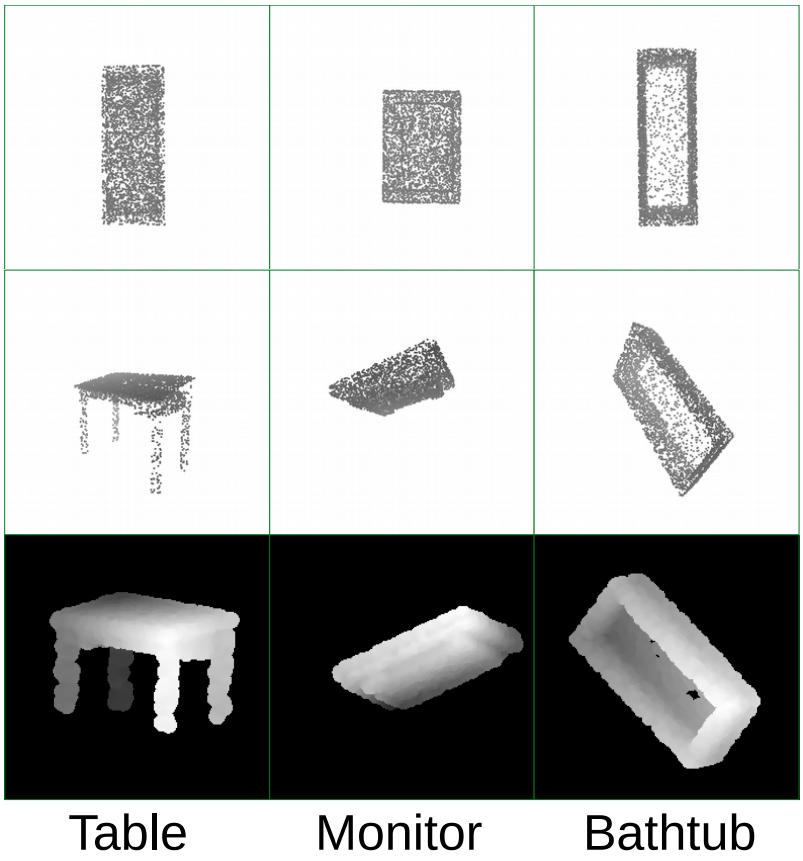

Figure A.5: *The views selected by PCA (top: point clouds), and by our method (middle: point clouds, bottom: depth images).*

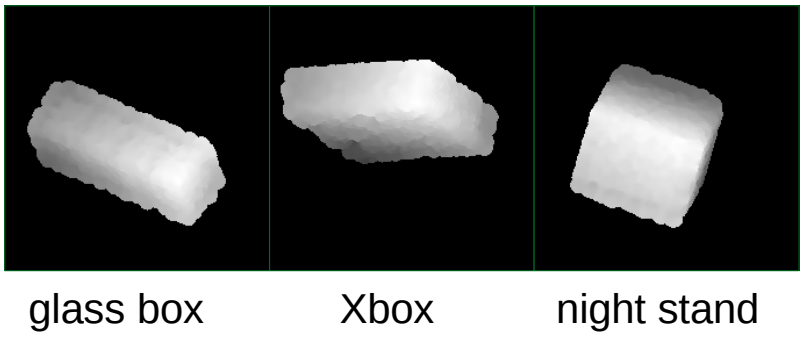

Figure A.6: *Three classes where our generated views are ambiguous.*

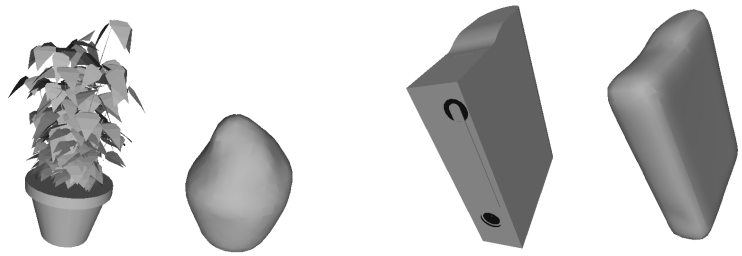

Figure A.7: *The original high resolution mesh (left) and the reconstructed mesh from a point cloud of 2048 points (right), for two models (plant and XBox).*

each case. The classes in bold are the ones which were always better classified with the random alternative (bench and radio), or always better classified with our original architecture (piano, vase, desk, table, baththub, dresser and night stand). It is noticeable how our original architecture performs consistently better in more classes than the random alternative, due to its specialized learned views that adapt to classes. We believe that the random alternative can sometimes perform better than our method due to hard, ambiguous classes present in the dataset (e.g., radio that can be confused with other regular objects).

A.4.5 Gradients for Depth Image Generation

In this section, we present the gradients of our custom module for depth images generation.

We name the nominator and denominator of the function $f(c)$ of Equation 6.4 as follows:

$$f(c) = \frac{\sum_{p \in P''(c)} g(c,p) p_z}{\sum_{p \in P''(c)} g(c,p)} = \frac{f_c^1(P)}{f_c^2(P)} \tag{A.19}$$

The gradients with respect to the position of a point $p \in P$ for f_1 can be

± Acc.	-0.05	-0.04	-0.04	-0.02	-0.01	-0.01	-0.01	-0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.04	0.04	0.05	0.06	0.06	0.07	0.07
	wardrobe	stairs	**radio**	laptop	flower pot	chair	plant	**bench**	glass box	bed	mantel	**piano**	tent	**night stand**	tv stand	**dresser**	range hood	**vase**	sofa	cone	door	person	cup	sink	bowl	curtain	**table**	stool	**desk**	xbox	**baththub**

Table A.2: *Difference of obtained accuracy between our original method and the random views alternative (± Acc.), for 2 views, per class.*

± Acc.	-0.06	-0.04	-0.03	-0.03	-0.02	-0.02	-0.02	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.03	0.03	0.04	0.04	0.04	0.04
	tent	wardrobe	door	**bench**	plant	cup	tv stand	curtain	keyboard	bed	bowl	stool	glass box	bottle	monitor	**radio**	car	**table**	**piano**	**desk**	chair	mantel	**vase**	**night stand**	cone	lamp	person	sink	stairs	flower pot	xbox	**dresser**	**bathtub**

Table A.3: *Difference of obtained accuracy between our original method and the random views alternative (± Acc.), for 4 views, per class.*

defined as:

$$\frac{\partial f_c^1(P)}{\partial p_z} = \begin{cases} 0 & if \quad p \notin P''(c) \\ g((c_x, c_y), (p_x, p_y)) & otherwise \end{cases} \tag{A.20}$$

$$\frac{\partial f_c^1(P)}{\partial p_y} = \begin{cases} 0 & if \quad p \notin P''(c) \\ \frac{(c_y - p_y)}{\sigma^2} g((c_x, c_y), (p_x, p_y)) p_z & otherwise \end{cases} \tag{A.21}$$

The gradients for f_2 are defined as follow:

$$\frac{\partial f_c^2(P)}{\partial p_z} = 0 \tag{A.22}$$

$$\frac{\partial f_c^2(P)}{\partial p_y} = \begin{cases} 0 & if \quad p \notin P''(c) \\ \frac{(c_y - p_y)}{\sigma^2} g((c_x, c_y), (p_x, p_y)) & otherwise \end{cases} \tag{A.23}$$

Similarly, the gradients with respect to p_x can be computed.

The final derivative for $f(c)$ is then defined by utilizing the product rule.

Notice that the presented derivatives assume that the subset P'' stays constant for a pixel c. Hence, the dependence on the sample positions $p \in P$ is neglected. In practice we found this approximation to not influence the results, as the subset P'' generally changes smoothly for varying view directions. This is likely due to the high quality dataset, where the models are made of continuous surfaces, are sampled uniformely and have no missing parts. As a future work, it would be interesting to investigate the effect of this approximation for more difficult cases that can be found in practice, where the subset P'' could change abruptly.

References

[Ahmed et al., 2015] Abdalla G. M. Ahmed, Hui Huang, and Oliver Deussen. Aa patterns for point sets with controlled spectral properties. *ACM Trans. Graph.*, 34(6):212:1–212:8, October 2015.

[Alexa et al., 2001] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *Proceedings of Visualization*, VIS '01, pages 21–28, 2001.

[Alexa et al., 2003] 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*, 9(1):3–15, Jan 2003.

[AlMeraj et al., 2013] Zainab AlMeraj, Craig S. Kaplan, and Paul Asente. Patch-based geometric texture synthesis. In *Proceedings of the Symposium on Computational Aesthetics*, CAE '13, pages 15–19, 2013.

[Alves dos Passos et al., 2010] V. Alves dos Passos, M. Walter, and M.C. Sousa. Sample-based synthesis of illustrative patterns. In *Computer Graphics and Applications*, PG '10, pages 109–116, 2010.

[Anguelov et al., 2005] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: Shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, July 2005.

[Arvind et al., 2017] Varun Arvind, Anthony Costa, Marcus Badgeley, Samuel Cho, and Eric Oermann. Wide and deep volumetric residual networks for volumetric image classification. *CoRR*, abs/1710.01217, 2017.

[Avron et al., 2010] Haim Avron, Andrei Sharf, Chen Greif, and Daniel Cohen-Or. L1sparse reconstruction of sharp point set surfaces. *ACM Trans. Graph.*, 29(5):135:1–135:12, November 2010.

[Balzer et al., 2009] Michael Balzer, Thomas Schlömer, and Oliver Deussen. Capacity-constrained point distributions: A variant of lloyd's method. *ACM Trans. Graph.*, 28(3):86:1–86:8, July 2009.

[Barla et al., 2006] Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. Eurographics)*, volume 25, pages 663–671, 2006.

[Beeler et al., 2012] Thabo Beeler, Bernd Bickel, Gioacchino Noris, Steve Marschner, Paul Beardsley, Robert W. Sumner, and Markus Gross. Coupled 3d reconstruction of sparse facial hair and skin. *ACM Trans. Graph.*, 31:117:1–117:10, August 2012.

[Berger et al., 2017] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 36(1):301–329, 2017.

[Bowers et al., 2010] John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph.*, 29:166:1–166:10, December 2010.

[Brock et al., 2016] André Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *CoRR*, abs/1608.04236, 2016.

[Bruna et al., 2013] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *ArXiv e-prints*, December 2013.

[Carr et al., 2001] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[Chen et al., 2013] Jiating Chen, Xiaoyin Ge, Li-Yi Wei, Bin Wang, Yusu Wang, Huamin Wang, Yun Fei, Kang-Lai Qian, Jun-Hai Yong, and Wenping Wang. Bilateral blue noise sampling. *ACM Trans. Graph.*, 32(6):216:1–216:11, November 2013.

[Cimpoi et al., 2014] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[Dai et al., 2017] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[de Goes et al., 2012] Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. Blue noise through optimal transport. *ACM Trans. Graph.*, 31(6):171:1–171:11, November 2012.

[Deng et al., 2009] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[Dibra et al., 2016] Endri Dibra, Himanshu Jain, A. Cengiz Öztireli, Remo Ziegler, and Markus H. Gross. Hs-nets: Estimating human body shape from silhouettes with convolutional neural networks. In *Fourth International Conference on 3D Vision, 3DV 2016, Stanford, CA, USA, October 25-28, 2016*, pages 108–117, 2016.

[Dibra et al., 2017] Endri Dibra, Himanshu Jain, A. Cengiz Öztireli, Remo Ziegler, and Markus H. Gross. Human shape from silhouettes using generative hks descriptors and cross-modal neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, July 21-26, 2017*, 2017.

[Donahue et al., 2014] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*, 2014.

[Du et al., 2013] Song-Pei Du, Shi-Min Hu, and Ralph R. Martin. Semiregular solid texturing from 2d image exemplars. *IEEE Trans. Vis. Comput. Graph.*, 19(3):460–469, 2013.

[Durand, 2011] Fredo Durand. A frequency analysis of monte-carlo and other numerical integration schemes. Technical Report MIT-CSAILTR-2011-052, CSAIL, MIT,, MA, February 2011.

[Efros and Leung, 1999] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision*, volume 2 of *ICCV '99*, pages 1033–1038, 1999.

[Engelcke et al., 2017] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d

point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 1355–1361, 2017.

[Fang et al., 2015] Y. Fang, Jin Xie, Guoxian Dai, Meng Wang, Fan Zhu, Tiantian Xu, and E. Wong. 3d deep shape descriptor. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2319–2328, June 2015.

[Fattal, 2011] Raanan Fattal. Blue-noise point sampling using kernel density model. *ACM Trans. Graph.*, 30(4):48:1–48:12, July 2011.

[Gal et al., 2007] Ran Gal, Ariel Shamir, Tal Hassner, Mark Pauly, and Daniel Cohen-Or. Surface reconstruction using local shape priors. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 253–262, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[Gharbi et al., 2017] Michaël Gharbi, Jiawen Chen, Jonathan T. Barron, Samuel W. Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *ACM Trans. Graph.*, 36(4):118:1–118:12, July 2017.

[Girshick et al., 2014] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA, 2014. IEEE Computer Society.

[Goodfellow et al., 2014] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[Graham, 2014] B. Graham. Spatially-sparse convolutional neural networks. *ArXiv e-prints*, September 2014.

[Graham, 2015] Ben Graham. Sparse 3d convolutional neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 150.1–150.9. BMVA Press, September 2015.

[Guennebaud and Gross, 2007] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. *ACM Trans. Graph.*, 26(3), July 2007.

[Guo et al., 2015] Kan Guo, Dongqing Zou, and Xiaowu Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Trans. Graph.*, 35(1):3:1–3:12, December 2015.

[Han et al., 2017] X. Han, Z. Li, H. Huang, E. Kalogerakis, and Y. Yu. High-resolution shape completion using deep neural networks for global structure and local geometry inference. In *IEEE International Conference on Computer Vision (ICCV)*, October 2017.

[He et al., 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[Heck et al., 2013] Daniel Heck, Thomas Schlömer, and Oliver Deussen. Blue noise sampling with controlled aliasing. *ACM Transactions on Graphics*, 32(3):25:1–25:12, 2013.

[Huang et al., 2009] Hui Huang, Dan Li, Hao Zhang, Uri Ascher, and Daniel Cohen-Or. Consolidation of unorganized point clouds for surface reconstruction. *ACM Trans. Graph.*, 28(5):176:1–176:7, December 2009.

[Huang et al., 2013] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao (Richard) Zhang. Edge-aware point set resampling. *ACM Trans. Graph.*, 32(1):9:1–9:12, February 2013.

[Huang et al., 2014] Zhe Huang, Jiang Wang, Hongbo Fu, and Rynson W. H. Lau. Structured mechanical collage. *IEEE Trans. Vis. Comput. Graph.*, 20(7):1076–1082, 2014.

[Hurtut et al., 2009] T. Hurtut, P.-E. Landes, J. Thollot, Y. Gousseau, R. Drouillhet, and J.-F. Coeurjolly. Appearance-guided synthesis of element arrangements by example. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '09, pages 51–60, 2009.

[Ijiri et al., 2008] Takashi Ijiri, Radomír Mech, Takeo Igarashi, and Gavin S. P. Miller. An example-based procedural system for element arrangement. *Computer Graphics Forum*, 27(2):429–436, 2008.

[Illian et al., 2008] Janine Illian, Antti Penttinen, Helga Stoyan, and Dietrich Stoyan, editors. *Statistical Analysis and Modelling of Spatial Point Patterns*. John Wiley and Sons, Ltd., 2008.

[Isola et al., 2016] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[Jaderberg et al., 2015] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015.

[Jiang et al., 2015] Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and

Jian Jun Zhang. Blue noise sampling using an sph-based method. *ACM Trans. Graph.*, 34(6):211:1–211:11, October 2015.

[Ju et al., 2010] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable crowds. *ACM Trans. Graph.*, 29(6):140:1–140:10, December 2010.

[Kalogerakis et al., 2017] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 3D shape segmentation with projective convolutional networks. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017.

[Kang et al., 2007] Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. In *ACM Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 43–50, August 2007.

[Kazhdan and Hoppe, 2013] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, July 2013.

[Kazhdan et al., 2003] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '03, pages 156–164, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Kazhdan et al., 2006] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[Kazi et al., 2012a] Rubaiat Habib Kazi, Takeo Igarashi, Shengdong Zhao, and Richard Davis. Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1727–1736, New York, NY, USA, 2012. ACM.

[Kazi et al., 2012b] Rubaiat Habib Kazi, Takeo Igarashi, Shengdong Zhao, and Richard Davis. Vignette: Interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1727–1736, New York, NY, USA, 2012. ACM.

[Kazi et al., 2014] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. Draco: Bringing life to illustrations with kinetic textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 351–360, New York, NY, USA, 2014. ACM.

[Kim et al., 2008] Dongyeon Kim, Minjung Son, Yunjin Lee, Henry Kang, and Seungyong Lee. Feature-guided image stippling. *Computer Graphics Forum*, 27(4):1209–1216, 2008.

[Kim et al., 2012] Young Min Kim, Niloy J. Mitra, Dong-Ming Yan, and Leonidas Guibas. Acquiring 3d indoor environments with variability and repetition. *ACM Trans. Graph.*, 31(6):138:1–138:11, November 2012.

[Kim et al., 2013] Young Min Kim, Niloy J. Mitra, Qi-Xing Huang, and Leonidas J. Guibas. Guided real-time scanning of indoor objects. *Comput. Graph. Forum*, 32(7):177–186, 2013.

[Kim et al., 2015] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. *CoRR*, abs/1511.04587, 2015.

[Klokov and Lempitsky, 2017] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[Kopf et al., 2007] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 26(3):2:1–2:9, 2007.

[Kwatra et al., 2005] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24(3):795–802, July 2005.

[Lagae and Dutré, 2008] Ares Lagae and Philip Dutré. A comparison of methods for generating Poisson disk distributions. *Comput. Graph. Forum*, 27(1):114–129, March 2008.

[Lai et al., 2005] Y.-K. Lai, S.-M. Hu, D. X. Gu, and R. R. Martin. Geometric texture synthesis and transfer via geometry images. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, SPM '05, pages 15–26, New York, NY, USA, 2005. ACM.

[Landes et al., 2013] Pierre-Edouard Landes, Bruno Galerne, and Thomas Hurtut. A shape-aware model for discrete texture synthesis. *Computer Graphics Forum*, 32(4):67–76, 2013.

[Li et al., 2010] Hongwei Li, Li-Yi Wei, Pedro V. Sander, and Chi-Wing Fu. Anisotropic blue noise sampling. *ACM Trans. Graph.*, 29(6):167:1–167:12, December 2010.

[Li et al., 2015] Yangyan Li, Angela Dai, Leonidas Guibas, and Matthias Nießner.

Database-assisted object retrieval for real-time 3d reconstruction. *Computer Graphics Forum*, 34(2), 2015.

[Li et al., 2016] Yangyan Li, Sören Pirk, Hao Su, Charles Ruizhongtai Qi, and Leonidas J. Guibas. FPNN: field probing neural networks for 3d data. *CoRR*, abs/1605.06240, 2016.

[Lipman et al., 2007] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph.*, 26(3), July 2007.

[Lloyd, 1982] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982.

[Loper and Black, 2014] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *Computer Vision – ECCV 2014*, volume 8695 of *Lecture Notes in Computer Science*, pages 154–169. Springer International Publishing, September 2014.

[Lu et al., 2013] Jingwan Lu, Connelly Barnes, Stephen DiVerdi, and Adam Finkelstein. Realbrush: Painting with examples of physical media. *ACM Trans. Graph.*, 32(4):117:1–117:12, July 2013.

[Lu et al., 2014] Jingwan Lu, Connelly Barnes, Connie Wan, Paul Asente, Radomir Mech, and Adam Finkelstein. DecoBrush: Drawing structured decorative patterns by example. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, August 2014.

[Lukáč et al., 2013] Michal Lukáč, Jakub Fišer, Jean-Charles Bazin, Ondřej Jamriška, Alexander Sorkine-Hornung, and Daniel Sýkora. Painting by feature: Texture boundaries for example-based image creation. *ACM Trans. Graph.*, 32(4):116:1–116:8, July 2013.

[Ma et al., 2011] Chongyang Ma, Li-Yi Wei, and Xin Tong. Discrete element textures. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, pages 62:1–62:10, 2011.

[Ma et al., 2013] Chongyang Ma, Li-Yi Wei, Sylvain Lefebvre, and Xin Tong. Dynamic element textures. *ACM Transactions on Graphics*, 32(4):90:1–90:10, 2013.

[Martin et al., 2010] Sebastian Martin, Peter Kaufmann, Mario Botsch, Eitan Grinspun, and Markus Gross. Unified simulation of elastic rods, shells, and solids. *ACM Transaction on Graphics (Proc. SIGGRAPH)*, 29(3):39:1–39:10, 2010.

[Masci et al., 2015] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *2015 IEEE*

International Conference on Computer Vision Workshop (ICCVW), pages 832–840, Dec 2015.

[Maturana and Scherer, 2015] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, September 2015.

[McCool and Fiume, 1992] Michael McCool and Eugene Fiume. Hierarchical poisson disk sampling distributions. In *Proceedings of the Conference on Graphics Interface '92*, pages 94–105, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[Merrell and Manocha, 2008] Paul Merrell and Dinesh Manocha. Continuous model synthesis. *ACM Transaction on Graphics*, 27(5), 2008.

[Mitchell, 1987] Don P. Mitchell. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph.*, 21(4):65–72, August 1987.

[Mitchell, 1996] Don P. Mitchell. Consequences of stratified sampling in graphics. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 277–280, New York, NY, USA, 1996. ACM.

[Møller and Waagepetersen, 2004] Jesper Møller and Rasmus Plenge Waagepetersen. *Statistical inference and simulation for spatial point processes*. Chapman & Hall/CRC, 2003, Boca Raton (Fl.), London, New York, 2004.

[Nan et al., 2012] Liangliang Nan, Ke Xie, and Andrei Sharf. A search-classify approach for cluttered indoor scene understanding. *ACM Trans. Graph.*, 31(6):137:1–137:10, November 2012.

[Ostromoukhov, 2007] Victor Ostromoukhov. Sampling with polyominoes. *ACM Trans. Graph.*, 26(3), July 2007.

[Öztireli and Gross, 2012] A. Cengiz Öztireli and Markus Gross. Analysis and synthesis of point distributions based on pair correlation. *ACM Transaction on Graphics*, 31(6), 2012.

[Öztireli et al., 2009] A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2):493–501, 2009.

[Öztireli et al., 2010] A. Cengiz Öztireli, Marc Alexa, and Markus Gross. Spectral sampling of manifolds. *ACM Trans. Graph.*, 29(6):168:1–168:8, December 2010.

[Öztireli, 2016] A. Cengiz Öztireli. Integration with stochastic point processes. *ACM Trans. Graph.*, 35(5):160:1–160:16, August 2016.

[Paget and Longsta, 1995] Rupert Paget and Dennis Longsta. Texture synthesis via a non-parametric markov random field. In *Proceedings of Digital Image Computing: Techniques and Applications*, volume 1, pages pp. 547–552, 1995.

[Pauly et al., 2002] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the Conference on Visualization '02*, VIS '02, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

[Pauly et al., 2005] Mark Pauly, Niloy J. Mitra, Joachim Giesen, Markus Gross, and Leonidas J. Guibas. Example-based 3d scan completion. In *Proceedings of the Third Eurographics Symposium on Geometry Processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[Pharr and Humphreys, 2010] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[Pilleboue et al., 2015] Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. Variance analysis for monte carlo integration. *ACM Trans. Graph.*, 34(4):124:1–124:14, July 2015.

[Preiner et al., 2014] Reinhold Preiner, Oliver Mattausch, Murat Arikan, Renato Pajarola, and Michael Wimmer. Continuous projection for fast l1 reconstruction. *ACM Trans. Graph.*, 33(4):47:1–47:13, July 2014.

[Qi et al., 2016a] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[Qi et al., 2016b] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5648–5656, 2016.

[Qi et al., 2017a] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[Qi et al., 2017b] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

[Ramamoorthi et al., 2012] Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. A theory of monte carlo visibility sampling. *ACM Trans. Graph.*, 31(5):121:1–121:16, September 2012.

[Riegler et al., 2017] Gernot Riegler, Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[Roveri, 2014] Riccardo Roveri. Example-based geometry brush. *Master Thesis, ETH*, 2014.

[Schmaltz et al., 2010] Christian Schmaltz, Pascal Gwosdek, Andrés Bruhn, and Joachim Weickert. Electrostatic halftoning. *Comput. Graph. Forum*, 29(8):2313–2327, 2010.

[Shao et al., 2012] Tianjia Shao, Weiwei Xu, Kun Zhou, Jingdong Wang, Dongping Li, and Baining Guo. An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Trans. Graph.*, 31(6):136:1–136:11, November 2012.

[Sharma et al., 2016] Abhishek Sharma, Oliver Grau, and Mario Fritz. Vconv-dae: Deep volumetric shape learning without object labels. *Computer Vision – ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*, pages 236–250, 2016.

[Shen et al., 2004] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. *ACM Trans. Graph.*, 23(3):896–904, August 2004.

[Shen et al., 2012] Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Trans. Graph.*, 31(6):180:1–180:11, November 2012.

[Shirley, 1991] Peter Shirley. Discrepancy as a quality measure for sample distributions. In *In Eurographics '91*, pages 183–194. Elsevier Science Publishers, 1991.

[Simonovsky and Komodakis, 2017] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[Su et al., 2015] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

[Subr and Kautz, 2013] Kartic Subr and Jan Kautz. Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration. *ACM Trans. Graph.*, 32(4):128:1–128:12, July 2013.

[Subr et al., 2014] Kartic Subr, Derek Nowrouzezahrai, Wojciech Jarosz, Jan Kautz, and Kenny Mitchell. Error analysis of estimators that use combinations of stochastic sampling strategies for direct illumination. *Comput. Graph. Forum*, 33(4):93–102, 2014.

[Sun et al., 2015] Yujing Sun, Scott Schaefer, and Wenping Wang. Denoising point sets via l 0 minimization. *Comput. Aided Geom. Des.*, 35(C):2–15, May 2015.

[Sung et al., 2015] Minhyuk Sung, Vladimir G. Kim, Roland Angst, and Leonidas Guibas. Data-driven structural priors for shape completion. *ACM Trans. Graph.*, 34(6), October 2015.

[Turk, 2001] Greg Turk. Texture synthesis on surfaces. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 347–354, 2001.

[Ulichney, 1988] R.A. Ulichney. Dithering with blue noise. *Proceedings of the IEEE*, 76(1):56–79, Jan 1988.

[Varley et al., 2016] J. Varley, C. DeChant, A. Richardson, J. Ruales, and P. Allen. Shape Completion Enabled Robotic Grasping. *ArXiv e-prints*, September 2016.

[Wachtel et al., 2014] Florent Wachtel, Adrien Pilleboue, David Coeurjolly, Katherine Breeden, Gurprit Singh, Gaël Cathelin, Fernando de Goes, Mathieu Desbrun, and Victor Ostromoukhov. Fast tile-based adaptive sampling with user-specified fourier spectra. *ACM Trans. Graph.*, 33(4):56:1–56:11, July 2014.

[Wang and Posner, 2015] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[Wang et al., 2016] Peng-Shuai Wang, Yang Liu, and Xin Tong. Mesh denoising via cascaded normal regression. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 35(6), 2016.

[Wang et al., 2017] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Trans. Graph.*, 36(4):72:1–72:11, July 2017.

[Wei and Wang, 2011] Li-Yi Wei and Rui Wang. Differential domain analysis for non-uniform sampling. *ACM Trans. Graph.*, 30(4):50:1–50:10, July 2011.

[Wei et al., 2009] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. In *Eurographics '09 State of the Art Reports (STARs)*, March 2009.

[Wei, 2010] Li-Yi Wei. Multi-class blue noise sampling. *ACM Trans. Graph.*, 29(4):79:1–79:8, July 2010.

[Weise et al., 2011] Thibaut Weise, Sofien Bouaziz, Hao Li, and Mark Pauly. Real-time performance-based facial animation. *ACM Trans. Graph.*, 30(4):77:1–77:10, July 2011.

[Wu et al., 2015a] Shihao Wu, Hui Huang, Minglun Gong, Matthias Zwicker, and Daniel Cohen-Or. Deep points consolidation. *ACM Trans. Graph.*, 34(6):176:1–176:13, October 2015.

[Wu et al., 2015b] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, pages 1912–1920. IEEE Computer Society, 2015.

[Wu et al., 2016] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[Xing et al., 2014] Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. Autocomplete painting repetitions. *ACM Trans. Graph.*, 33(6):172:1–172:11, November 2014.

[Xiong et al., 2014] Shiyao Xiong, Juyong Zhang, Jianmin Zheng, Jianfei Cai, and Ligang Liu. Robust surface reconstruction via dictionary learning. *ACM Transactions on Graphics (Proc. SIGGRAPH Aisa)*, 33, 2014.

[Xu et al., 2015] Li Xu, Jimmy S. J. Ren, Qiong Yan, Renjie Liao, and Jiaya Jia. Deep edge-aware filters. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1669–1678. JMLR.org, 2015.

[Yan et al., 2014] Ling-Qi Yan, Miloš Hašan, Wenzel Jakob, Jason Lawrence, Steve Marschner, and Ravi Ramamoorthi. Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Trans. Graph.*, 33(4):116:1–116:9, July 2014.

[Yan et al., 2016] Zhicheng Yan, Hao Zhang, Baoyuan Wang, Sylvain Paris, and Yizhou Yu. Automatic photo adjustment using deep neural networks. *ACM Trans. Graph.*, 35(2):11:1–11:15, February 2016.

[Zhang et al., 2016] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising. *CoRR*, abs/1608.03981, 2016.

[Zhou et al., 2006] Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. Mesh quilting for geometric texture synthesis. *ACM Trans. Graph.*, 25(3):690–697, 2006.

[Zhou et al., 2007] Howard Zhou, Jie Sun, Greg Turk, and James M. Rehg. Terrain

synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, 2007.

[Zhou et al., 2012] Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. Point sampling with general noise spectrum. *ACM Transaction on Graphics*, 31(4):76:1–76:11, 2012.

[Zhou et al., 2013] Shizhe Zhou, Anass Lasram, and Sylvain Lefebvre. By-example synthesis of curvilinear structured patterns. *Comput. Graph. Forum*, 32(2):355–360, 2013.

[Zhou et al., 2014] Shizhe Zhou, Changyun Jiang, and Sylvain Lefebvre. Topology-constrained synthesis of vector patterns. *ACM Trans. Graph.*, 33(6):215:1–215:11, November 2014.

[Zhu et al., 2014] Bo Zhu, Ed Quigley, Matthew Cong, Justin Solomon, and Ronald Fedkiw. Codimensional surface tension flow on simplicial complexes. *ACM Transaction on Graphics (Proc. SIGGRAPH)*, 33(4):111:1–111:11, 2014.