Fine Feature Reconstruction in Point Set Surfaces Using Deep Learning

Prashant Raina

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy (Computer Science)

Concordia University

Montréal, Québec, Canada

October 2019

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Prashant Raina

Entitled: Fine Feature Reconstruction in Point Set Surfaces Using Deep Learning

and submitted in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. William Lynch

_____ External Examiner
Dr. David Mould

_____ External to Program
Dr. Hassan Rivaz

_____ Examiner
Dr. Thomas Gordon Fevens

_____ Examiner
Dr. Adam Krzyzak

_____ Thesis Co-Supervisor
Dr. Sudhir P. Mudur

_____ Thesis Co-Supervisor
Dr. Tiberiu Popa

Approved by _____
Dr. Leila Kosseim, Graduate Program Director

January 9, 2020 _____
Dr. Amir Asif, Dean
Gina Cody School of Engineering & Computer Science

Abstract

Fine Feature Reconstruction in Point Set Surfaces Using Deep Learning

Prashant Raina, Ph.D.

Concordia University, 2019

Point clouds are typically captured from surfaces of real-world objects using scanning devices. This scanning process invariably results in the loss of sharp edges as well as other geometric features of the original surface, which we collectively refer to as "fine features". This thesis explores leveraging recent advances in deep learning in order to recover fine features of surfaces which were lost when acquiring the point cloud. We first focus on reconstructing sharp edges of the original surface. We define a new concept — a *sharpness field* — over the underlying surface of a point cloud, whose ridges give the locations of sharp edges even at points not originally sampled from the surface. We then demonstrate that with appropriate training data, deep neural networks can be trained to compute the sharpness field for a point cloud. We evaluate several different local neighborhood representations and deep learning models to improve the accuracy of sharpness field computation across different neighborhood scales. Some applications of the sharpness field are then described. The most important such application is *feature-aware smoothing*: using the computed sharpness field to preserve sharp edges while removing noise from point clouds. Our novel smoothing algorithm shows superior performance in reconstructing sharp edges and corners compared to the state-of-the-art RIMLS algorithm, while also yielding points lying on sharp edges. Other applications of the sharpness field are also presented: generating a graphical representation of sharp edges and segementing a point cloud into smooth surface patches. We then expand the scope of the problem from sharp edges to undersampled fine features in general. We tackle this by developing a unique deep learning approach to point cloud super-resolution using an innovative application of generative adversarial networks (GANs). The novelty of our super-resolution method lies in framing point cloud super-resolution as a domain translation task between heightmaps obtained from point clouds and heightmaps obtained from triangular meshes. By using recent developments in domain translation using GANs, we obtain results qualitatively and quantitatively superior to state-of-the-art point cloud super-resolution methods, all while using a radically different deep learning approach which

is also more computationally efficient.

The main contributions of this thesis are:

1. Establishing *sharpness field* computation as a novel method for localizing sharp edges in 3D point clouds.

2. Several new data-driven methods to compute the sharpness field for a point cloud using different local neighborhood representations and machine learning models.

3. A novel feature-aware smoothing algorithm for denoising point clouds while preserving sharp edges, using the aforementioned sharpness field. This method produces denoising results superior to the state-of-the-art RIMLS smoothing method.

4. An innovative application of GAN-based domain translation, in order to transform sparse heightmaps obtained from point cloud neighborhoods into dense heightmaps obtained from triangular meshes.

5. A unique method for reconstrucing fine features in point clouds, by using heightmap domain translation to perform point cloud super-resolution. The reconstructed surfaces are qualitatively and quantitatively superior to those produced by state-of-the-art point cloud super-resolution methods.

Other contributions include:

1. An algorithm for extracting an explicit graphical representation of the sharp edges of a point cloud, using the sharpness field.

2. An algorithm for segmenting a point cloud into smooth patches, using the sharpness field.

3. Feature-aware smoothing algorithms which incorporate the aforementioned edge graph and patch segmentation.

Acknowledgments

My PhD journey was made possible by the Concordia University Full Tuition Recruitment Award from the School of Graduate Studies at Concordia University. This tuition waiver made it financially feasible for me to devote myself full-time to academic research work during these five years.

I would like to extend my sincere thanks to my two PhD supervisors, Dr. Tiberiu Popa and Dr. Sudhir Mudur, for initiating me into computer graphics research, and for ensuring that I was adequately funded and equipped during my time as a PhD student. Despite their busy schedules, they always made themselves available for discussions related to my research. Their mentorship inspired me to persevere in research and explore new ideas.

Dr. Popa's infectious energy, enthusiasm and passion for computer graphics convinced me to take the plunge into a PhD program in this field, despite the fact that my previous research work was in natural language processing. Over the years, I have grown to see him not only as a guide and collaborator, but also as a friend.

Dr. Mudur was a pillar of support to me during the difficult periods of my research life. He was always willing to share his wisdom gained from decades of experience in conducting research and guiding graduate students. I thank him for the meticulous attention he showed when editing my research papers and thesis.

It was a risky career decision for me to give up my comfortable job in Singapore and start a new life in Canada, pursuing research in a field that was new to me. During this time, my parents' support and encouragement have never wavered. They have always believed in my ability and potential to perform well in my endeavors.

Most importantly, I would like to record my deepest gratitude to Master Li Hongzhi, founder of the spiritual practice of Falun Dafa. His teachings, being well-suited to a modern lifestyle, have enabled me to maintain a calm mind and a healthy body through all the ups and downs of life. I will continue to be guided by these teachings in the future.

Contribution of Authors

Sharp Edge Reconstruction (Chapters 3 & 4)

Prashant Raina	implementation, writing, editing and proofing
Tiberiu Popa	research supervisor, funding, editing and proofing
Sudhir Mudur	research supervisor, funding, editing and proofing

Point Cloud Super-Resolution (Chapters 5 & 6)

Prashant Raina	implementation, writing, editing and proofing
Tiberiu Popa	research supervisor, editing and proofing
Sudhir Mudur	research supervisor, editing and proofing

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This thesis investigates reconstructing fine features in 3D point clouds using deep learning. By "fine features", we refer to geometric features of the underlying surface of the point cloud whose radius of curvature is comparable to or smaller than the gaps between sampled points. This includes sharp edges (corresponding to radius of curvature 0), as well as other fine details, such as creases and darts, that are lost due to the limited sampling resolution of the point cloud. Preserving sharp edges and increasing the resolution of point clouds are classic problems in 3D geometry, which have traditionally been tackled using procedural methods. In constrast to these procedural methods, we are motivated to seek data-driven solutions, in order to explore the viability of applying recent advances in machine learning to classic geometric problems. In particular, to the best of our knowledge, ours is the first attempt at sharp feature detection in point clouds using machine learning [7, 10]. We have also developed an innovative application of generative adversarial networks (GANs) to point cloud super-resolution [12].

Recent years have seen a veritable revolution in applying machine learning to complex tasks. The availability of large datasets and widespread data collection from the real world have created new frontiers for applying machine learning. Tasks which were previously handled by procedural, expert-designed algorithms are now being revisited with new data-driven approaches. This has been made possible by the advent of *deep learning* [13]. Deep learning generally refers to training artificial neural networks with many hidden layers. These hidden layers learn progressively higher-level representations of low-level input data such as audio samples or pixel intensities in images. This hierarchical representation learning approach enables fully exploiting the potential of large quantities of raw, low-level data.

Most of the developments in deep learning have centered around convolutional neural networks (CNNs), which have proven to be very effective in the domain of image processing [14]. This is because CNNs exploit the spatial relationships provided by the regular grid structure of discretized data from Euclidean domains, such as 2D images. Bringing the benefits of deep learning to 3D geometry is more challenging, since 3D geometric data is represented in many different forms. 3D discrete convolution can still be applied to voxel grids, because they represent volumetric geometric information. However, in the case of 3D surface representations such as point clouds and meshes, it is impossible to directly apply discrete convolution directly to these representations, because they do not have a regular grid structure. Furthermore, the data is sampled from 3D surfaces, which have non-Euclidean geometry. This necessitates creative strategies to apply deep learning to 3D geometric data, which makes for an exciting and active area of research.

The work in this thesis focuses on 3D point clouds, a 3D surface data representation which is unstructured and lacks any topological information. Point clouds are important in the fields of computer graphics and computer vision, not because they are convenient to work with, but because they are the primary form in which 3D data is captured from the real world. It has also become commonplace in consumer electronics to have hardware which captures 3D images in the form of point clouds.

The broad theme of the research presented here is to leverage deep learning to "go deeper" into 3D point cloud data, and extract fine-grained geometric information in the gaps between the sampled points. We shall see that it is possible to recover sharp edges as well as other fine features of the underlying surface of a point cloud.

For recovering sharp edges from point clouds, we define a novel *sharpness field* over the underlying surface and demonstrate that a convolutional neural network can be trained to compute this sharpness field. In brief, the sharpness field can be viewed as a truncated distance field which takes the value 1 at edges and tapers to 0 at a short distance from edges. In contrast to previous work that classified individual points of the point cloud as belonging to edges, our sharpness field is defined even at points that do not belong to the input point cloud. It can therefore be used to identify the locations of sharp edges even if the edges lie between points sampled from the original surface.

Feeding an entire point cloud as input to a feedforward neural network would require that every point cloud have exactly the same number of points. This is, of course, an unrealistic requirement for most applications. It is more practical to perform computations on a per-point basis, by obtaining a representation of the local neighborhood of each point to be processed. We therefore begin our

attempts at sharpness field computation by developing an algorithm for unfolding a radial grid around a point lying on a point set surface. This radial grid is used to generate a Euclidean representation of the neighborhood of a point in a point cloud, which can be fed to a CNN as input. With appropriate training data that we have designed, this is sufficient to train a CNN to compute a sharpness field even for shapes that are more complex than those found in the training data.

In order to improve the performance of sharpness field computation on curved edges, we systematically experiment with simpler local neighborhood representations, while increasing the complexity of the neural network to handle these new input representations. We show that heightmaps enable better sharpness field computation, provided that they are fed to special neural network models that provide rotation invariance, such as harmonic networks [9] or spatial transformer networks [15]. We also show that even the raw coordinates of neighboring points can be used to train a neural network to compute the sharpness field, by using the recently developed PointNet [1] neural network architecture and a multi-scale variant called PCPNet [2]. We have also performed a thorough evaluation of these different neighborhood representations and neural network architectures for computing the sharpness field, comparing their performance across different neighborhood sizes for the same point cloud.

Once computed, the sharpness field can be used to recover sharp edges of the underlying surface of a point cloud. We demonstrate this by developing an algorithm which removes noise from a point cloud while using the sharpness field to preserve sharp edges. This denoising (smoothing) method is quite effective. The resulting surface has distinctly visible sharp edges, something which is not possible with the state-of-the-art RIMLS [16] denoising method. Patches of the point cloud enclosed by edges can be smoothed to a great extent with a large smoothing radius parameter, without the risk of losing sharp edges. In addition, we have developed algorithms for extracting a graphical representation of the sharp edges from the sharpness field, as well as for segmenting a point cloud into smooth patches using the sharpness field.

Since fine features are not limited to sharp edges, we have also developed a general-purpose method for super-resolution of point clouds using deep learning. We adapt recent work on image domain translation using generative adversarial networks (GANs) [17]. GANs have shown an astonishing ability to generate realistic-looking images that appear to originate from a particular domain of data. This capability has been used to produce realistic "translations" of images from one domain to another, such as from summer landscapes to winter landscapes or from sketches of handbags

to photographs of handbags. Unlike these artistic or toy applications, we develop a surprising application of this method to transforming sparse heightmaps from point cloud neighborhoods into dense heightmaps from mesh neighborhoods. These dense heightmaps can be used to increase the resolution of the point cloud at the local level. Furthermore, normals can be trivially computed at the newly generated points as a byproduct of our method. We show that our super-resolution method is superior both qualitatively and quantitatively to far more complicated state of the art methods for point cloud super-resolution. We also show that our method can easily be extended to interpolate the sharpness field over points that are newly generated during super-resolution. Provided the sharpness field was previously computed for the low-resolution point cloud, this is more computationally efficient than recomputing the sharpness field.

1.1 Contributions

The main contributions of this thesis are:

1. Establishing *sharpness field* computation as a novel method for localizing sharp edges in 3D point clouds.

2. Several new data-driven methods to compute the sharpness field for a point cloud using different local neighborhood representations and machine learning models.

3. A novel feature-aware smoothing algorithm for denoising point clouds while preserving sharp edges, using the aforementioned sharpness field. This method produces denoising results superior to the state-of-the-art RIMLS smoothing method.

4. An innovative application of GAN-based domain translation, in order to transform sparse heightmaps obtained from point cloud neighborhoods into dense heightmaps obtained from triangular meshes.

5. A unique method for reconstrucing fine features in point clouds, by using heightmap domain translation to perform point cloud super-resolution. The reconstructed surfaces are qualitatively and quantitatively superior to those produced by state-of-the-art point cloud super-resolution methods.

Other contributions include:

1. An algorithm for extracting an explicit graphical representation of the sharp edges of a point cloud, using the sharpness field.

2. An algorithm for segmenting a point cloud into smooth patches, using the sharpness field.

3. Feature-aware smoothing algorithms which incorporate the aforementioned edge graph and patch segmentation.

1.2 Thesis structure

Chapter 2 provides the background on related work in surface reconstruction from point clouds and point cloud super-resolution. It also summarizes the developments in the field of deep learning which are most important for 3D geometry.

Chapters 3 and 4 deal with the reconstruction of sharp edges in point clouds. We describe all the methods in Chapter 3 and discuss the results obtained in Chapter 4.

We define a smooth scalar field called the *sharpness field*, defined over the underlying surface of a point cloud, which allows us to localize sharp edges even at points which were not sampled from the original surface. Chapter 3 describes several machine learning approaches to sharpness field computation. It also details applications of the sharpness field for feature-aware smoothing, edge graph reconstruction and patch segmentation. In Chapter 4, we present the set of results we obtained using the different methods described in Chapter 3. The chapter includes a quantitative comparison of all proposed methods for sharpness field computation, including their performance for different neighborhood sizes.

In Chapters 5 and 6, we explore the recovery of fine features in point clouds as a super-resolution problem. Again Chapter 5 deals with the methods and Chapter 6 presents the results.

We leverage recent work on deep learning for domain translation using generative adversarial networks (GANs), in order to frame point cloud super-resolution as a problem of domain translation between point cloud neighborhoods and polygonal mesh neighborhoods. This method is described in detail in Chapter 5. Our local neighborhood-based deep learning approach allows us to obtain super-resolution results superior to the state-of-the-art point cloud super-resolution methods that adopt completely different deep learning approaches. Incidentally, our domain translation approach to point cloud super-resolution also enables super-resolution of a precomputed scalar field, such as the sharpness field described in Chapter 3. Details on our results and comparison with state-of-the-art

methods are provided in Chapter 6.

Chapter 7 concludes the thesis and provides directions for future work for possible extensions of the methods presented here.

Chapter 2

Background and Related Work

In the context of computer vision and graphics, 3D surface geometry is first acquired as a set of points from the real world, using LiDAR, laser scanners or other photogrammetric hardware (this excludes the special case of medical imaging, where volumetric data is obtained). These points are sampled from a 3D surface, but cannot be directly resampled or processed as a surface. A 3D surface must therefore be reconstructed from these points, and this surface may or may not be represented in an explicit form. The reconstructed surface is then used to build a polygonal mesh, if a mesh is desired for the target application. Following this, the resulting surface representation undergoes further processing, such as uniform resampling, simplification or analytical fitting. The geometry is then consumed by the target application. The work in this thesis can best be understood as taking place at or slightly before the surface reconstruction stage in the geometry pipeline.

Reconstructing surfaces from 3D point clouds has been studied extensively long before the widespread adoption of deep learning. In Section 2.1, we first provide some general background on the subjects of point clouds and deep learning. We then summarize the relevant past work in surface reconstruction from point clouds in Section 2.2.

2.1 Background

We first provide some general background on the use of point clouds as a 3D geometry representation in Section 2.1.1. Section 2.1.2 then gives a summary of the most important developments in the field of deep learning which are relevant to 3D geometry. Three recently published deep learning models related to point clouds are described in more detail. These models have been incorporated

into surface reconstruction and super-resolution methods described in this thesis.

2.1.1 Point clouds

3D geometric data is most easily acquired from the real world in the form of collections of 3D points lacking any associated topological information. This is in contrast to human-designed 3D data, which often have topological information built into them as a consequence of the design process (e.g. parametric surfaces or polygonal meshes). Unstructured collections of 3D points are referred to as *point clouds*.

Point clouds can be considered as irregular samplings of continuous 2D surfaces embedded in $\mathbb{R}^3$. Reconstructing the underlying surfaces of point clouds is a challenging problem for a variety of reasons, including noisy measurements and limited sampling resolution (the sampling density may not even be constant across the sampled surface). In particular, sharp features of surfaces are difficult to reconstruct because reconstruction methods often contain assumptions of smoothness. Surfaces reconstructed from point clouds are referred to as *point set surfaces*.

During the time period spanning approximately the years 2000 to 2010, it was unclear whether point clouds or polygonal meshes would become the dominant representation of 3D geometry in video games and other entertainment applications. Triangle meshes eventually emerged as the winner for a number of reasons, including the ability to compactly represent large surfaces with small numbers of triangles, planarity of triangles, easy differentiation of front-facing and back-facing surfaces, continuity and easy interpolation of vertex properties over a triangle. The most significant factor was, of course, the very flexible programmable rendering pipeline for triangle meshes incorporated into consumer graphics processing units (GPUs).

In recent years, point clouds are seeing a resurgence in interest in the fields of computer graphics and computer vision due to the growing availability of 3D depth sensors on consumer hardware, as well as the increasing usage of LiDAR sensors for autonomous driving. It is therefore worth revisiting past work from the early 2000's on point set surfaces to investigate whether new developments in the field of computer science can be used to improve upon earlier work.

2.1.2 Deep learning for 3D geometry

Deep learning [13] is the most successful form of *representation learning*. Representation learning is a field of machine learning in which machine learning models automatically learn representations of

low-level or raw input data which enable them to make inferences. This is as opposed to being given high-level representations called *features* as input (not related to geometric features). Such high-level features are typically designed by humans through *feature engineering*. Deep learning is characterized by the hierarchical learning of progressively more abstract representations from low-level data such as pixel intensity values or audio samples. The learnable functions which transform the low-level data into these higher-level representations are typically implemented as layers of an artificial neural network (referred to as a "deep" neural network when there are more than two hidden layers). Artificial neural networks are differentiable universal function approximators which are themselves composed of many simple functions. It has been proven [18] that when training deep neural networks, the global optimum can be reached using stochastic gradient descent [19]. Consequently, the memory cost of large training datasets can be controlled by sampling small batches called *minibatches* from the training dataset at each optimization step. Training these neural network models is very efficient nowadays thanks to numerous programming frameworks which parallelize the optimization process on the graphical processing unit (GPU).

The popularity of deep learning is largely due to the success of convolutional neural networks (CNNs) in the fields of image processing, computer vision, speech and medical imaging. CNNs learn kernels for performing convolutions on Euclidean data, such as 1D audio, 2D images or 3D voxel grids. Their success in 2D image applications is due in large part to "deep image prior" induced by the structure of CNNs [20]. Over the past years, several best practices have emerged for training CNNs, including Glorot weight initialization [21], batch normalization [22], as well as a variety of activation functions [23]. Most CNNs are also structured to have translation invariance, using either max pooling or strided convolutions [24]. In recent years, spatial transformer networks [15] and harmonic networks [9] have been proposed to provide other kinds of affine invariance.

After CNNs, the next major innovation in the field of deep learning is the generative adversarial network (GAN) [25]. A GAN consists of a pair of neural networks which are trained in an adversarial manner in order to obtain a generative model which generates realistic data samples. While GANs have seen many variants used for many different applications [26], they have been most extensively used for image-to-image translation. Image-to-image translation using deep learning has gained a lot of attention since Isola *et al* published their seminal work [17] on using conditional adversarial networks for translating between image domains given a training set of paired examples. This quickly led to more work on unpaired image translation [27], as well as image translation between more than two domains [28]. There have also been attempts to bring GAN-based domain translation methods

to other domains such as natural language text [29], audio [30] and voxel-based 3D data [31].

Most of the attempts to apply deep learning to 3D data have either used 3D voxel grids [32, 33, 34] or 2D views of a 3D shape [35, 33, 36]. Point clouds have also been converted into Euclidean representations through either voxelization [37] or multi-view rendering [38, 39, 40].

Some attempts have been made to extend the concept of a convolutional neural network layer to Riemannian manifolds [41, 42, 43, 44, 45]. While most of these methods are primarily intended for triangle meshes, some of them have been applied to point clouds as well, by creating nearest-neighbor graphs connecting the points [44, 45].

An interesting approach was proposed by Qi *et al*, called PointNet [1]. PointNet, along with its variants [3, 2], takes a fixed number of raw coordinates as input, while using a max operation and shared weights to obtain permutation invariance. Other works have used a PointNet-like architecture as part of a larger deep learning architecture [39, 46]. Guerrero *et al* used a multiresolution PointNet-based architecture called PCPNet [2] to estimate local geometric features in point clouds, such as normals and curvature. Achlioptas *et al* were the first to build a GAN based on PointNet [47]. Some recent work on point clouds such as FoldingNet [48] and AtlasNet [46] can also, in some sense, be regarded as precursors to domain translation between point clouds and explicit surfaces.

We now describe PointNet, PCPNet and PointNet++ [3] in further detail. This is to provide the necessary background for our experiments with sharpness field computation using PointNet and PCPNet. PointNet++ is the basis for all the recent point cloud super-resolution methods described in Section 2.2.4, which we later compare with our method in Chapter 6. It is therefore important to review it here before describing those methods.

2.1.2.1 PointNet

PointNet allows feeding a fixed number of unordered 3D points to a neural network. There are two 3D spatial transformer networks inside PointNet, which output affine transformations that transform their inputs to a canonical orientation. The PointNet architecture is naturally invariant to permutations of these points, since the weights of its convolutional layers are shared across all input points. After several convolutional layers, the activations of the neural network are aggregated in a permutation-invariant manner by taking the maximum activations across all input points. This produces a permutation-invariant global encoding of the point cloud. This global encoding, called a "feature vector", can be passed to subsequent layers to perform global classification of the

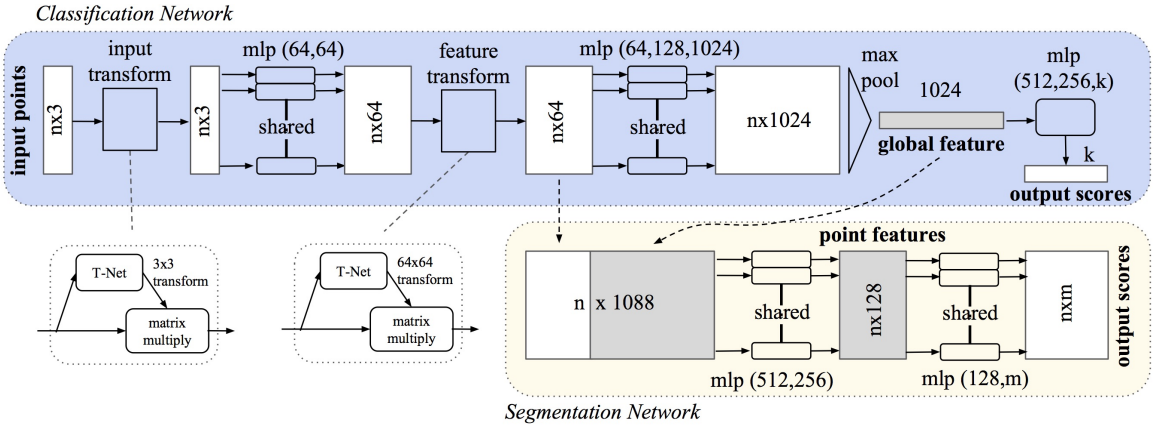

Figure 1: PointNet neural network architectures for classification and segmentation (image by Qi *et al* [1]).

point cloud. It can also be concatenated with the activations of layers preceding the permutation-invariant aggregation, thus forming a feature vector that provides both local and global context. This feature vector can then be passed on to subsequent convolutional layers that perform global semantic segmentation of the point cloud. The complete network architecture for classification and segmentation is shown in Figure 1.

Note that in the context of PointNet and various methods derived from it, "features" and "feature vectors" refer to the results of intermediate computations in the neural network. A feature vector associated with a point in a point cloud is often high-dimensional, and contains implicit geometric information about the neighborhood of the point. This high-dimensional vector cannot be directly interpreted, but it nevertheless contributes to the final result of the neural network.

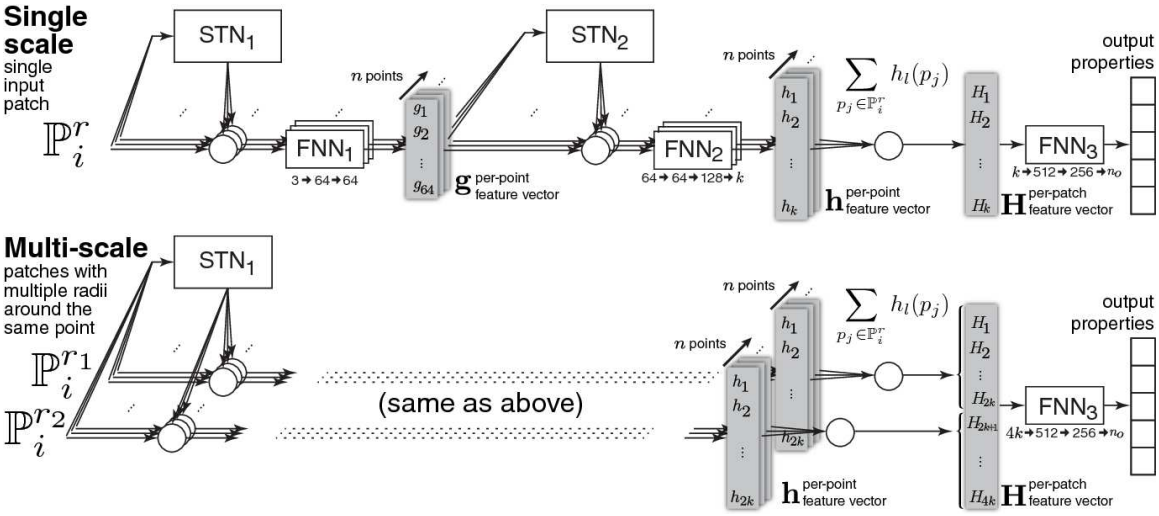

Figure 2: PCPNet neural network architecture (image by Guerrero *et al* [2]).

2.1.2.2 PCPNet

A PointNet variant called PCPNet [2] proposes a procedure for training PointNet to estimate geometric properties in point clouds, such as oriented normals or curvatures. Unlike PointNet, which operates on the entire point cloud at once, PCPNet takes the neighborhood of a single point as input. This neighborhood is represented as the normalized and centered 3D coordinates of a fixed number of nearest neighbors of the input point. Their single-resolution network architecture is essentially the same as the PointNet architecture used for global classification of point clouds. However, the first spatial transformer outputs a quaternion instead of a 3×3 matrix. When estimating normals, the output of the network must be rotated using the inverse of the quaternion suggested by the first spatial transformer. They have also presented a new multi-resolution architecture which operates on multiple point neighborhood sizes simultaneously. The network architecture is shown in Figure 2.

2.1.2.3 PointNet++

We have previously described PointNet in Section 2.1.2.1. PointNet++ [3] attempts to improve on the performance of PointNet for point cloud classification and segmentation by learning hierarchical multi-scale deep features from point clouds. The full network architecture is shown in Figure 3. In each layer of the hierarchy (called a "set abstraction layer"), a number of points are chosen as "patch centers" using farthest-point sampling. K nearest neighbors are found around each patch center. This results in a 3D tensor of fixed size which can then be reshaped as a 2D matrix and passed as

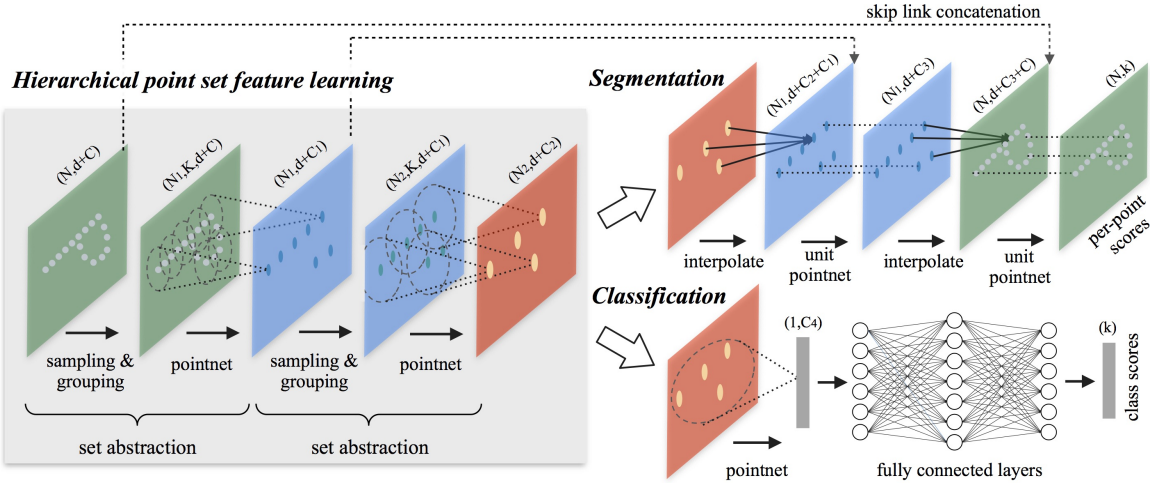

Figure 3: PointNet++ neural network architecture (image by Qi *et al* [3]).

input to a PointNet (having the architecture for point cloud segmentation) which outputs a matrix containing feature vectors for all patch centers. The number of patch centers naturally decreases over several set abstraction layers, after which we have a matrix containing a small fixed number of patch center feature vectors.

If one wishes to perform global classification of the point cloud, these feature vectors can be passed to a couple of fully connected neural network layers to predict the class of the point cloud. If the intended application is point cloud segmentation, the procedure is more complicated, since the output must have the same number of points as the original input point cloud. In this case, the output of the set abstraction layers is passed to several "feature propagation layers" which progressively restore the number of points to the original number (the number of feature propagation layers is equal to the number of set abstraction layers). In each feature propagation layer, the features of the input patch centers are interpolated based on Euclidean distance to the patch centers obtained from an earlier set abstraction layer (this concatatenation with previous layers is referred to as a "skip connection"). These points and their feature vectors are passed through multiple 1×1 convolutions to obtain a desired number of features. The final feature propagation layer has a skip connection to the original input, therefore it outputs a feature vector containing segmentation information for every input point.

2.2 Related Work

This section summarizes important work related to our methods for fine feature reconstruction in point clouds.

We recap the most relevant point set surface reconstruction literature in Section 2.2.1. We then focus exclusively on sharp features and how they are detected and reconstructed in Section 2.2.2. Section 2.2.3 then gives a brief introduction to generative adversarial networks and their applications. Section 2.2.4 finally summarizes the relevant work in point cloud super-resolution, which has recently been done almost exclusively using deep learning.

2.2.1 Point set surfaces

A comprehensive survey of methods to reconstruct surfaces from point clouds has been compiled by Berger *et al* [49]. In this thesis, we are mainly interested in reconstructing fine features, as opposed to the coarse shape of the original scanned object. Therefore, we do not consider methods that require generating coarse explicit surface representations, such as a triangular mesh [50]. Methods that generate implicit representations of surfaces are more interesting to us, because these implicit representations can potentially be enhanced to include fine features of the surface.

Implicit methods for surface reconstruction may either try to reconstruct the entire shape globally, or they may reconstruct the surface at the local level. One early global method for surface reconstruction was the attempt by Carr *et al* [51] to obtain radial basis functions (RBFs) which interpolate points in a point cloud. As an interpolating method, it is naturally sensitive to noise in the point cloud. RBF-based methods have also generally fallen out of favor due to their high computational cost. Kazhdan *et al* [52, 53] had more success when they showed that given oriented points on a watertight surface, a global indicator function can be obtained by integrating a Poisson equation. The surface is then reconstructed as an isosurface of the indicator function. It is worth noting that surfaces reconstructed in this manner are generally smooth and lack sharp features [54].

We are ultimately more interested in local implicit methods for surface reconstruction, since these methods scale better to large point sets, and have more scope for adding fine details because of their local nature. The most popular family of local implicit methods is based on Moving Least-Squares (MLS) surfaces, which were pioneered by Alexa *et al* [55]. They showed that it is possible to reconstruct surfaces as the set of stationary points of an MLS projection operator. This projection operator projects points onto a Moving Least-Squares approximation of the underlying surface. This

builds upon earlier work by Levin [56] showing that Moving Least-Squares can be used to locally approximate functions given scattered point data. In the case of MLS surface reconstruction, the domain of the function being approximated is a plane fitted to the local neighborhood of a point near the point cloud. MLS surfaces are known for being very efficient to compute, while also smoothing out noise because they are infinitely differentiable. Alexa and others followed up with additional work on utilizing MLS surfaces as a general technique for rendering point clouds as surfaces [57], as well as variations of the numerical methods used to compute MLS surfaces [58, 59]. Some of the theoretical properties valued in MLS surfaces are not preserved by the numerical methods used to approximate MLS surfaces. Kolluri [60] proposed a numerical method for MLS surface computation which does not compromise on these theoretical properties of MLS surface. Cheng *et al* [61] published a survey of MLS surface reconstruction methods as of 2008.

Various methods extend MLS surface reconstruction while better preserving high-frequency features. One approach has been to vary the shape of the local domain used for function approximation. While the original MLS method fits planes, Guennebaud and Gross [62] fit spherical surfaces. Lipman *et al* [63] went even further, by fitting polynomial spline surfaces to local neighborhoods.

Another approach has been to change the manner in which neighboring points are handled when computing MLS projections.

Fleishman *et al* [64] use an anistropic Gaussian kernel in MLS projection, inspired by earlier work [65] on mesh smoothing using anisotropic diffusion and robust statistics [66]. Fleishman *et al* perform iterative refitting of smooth patches to the neighborhood of a point in a point cloud, using Least-Median-of-Squares fitting, in order to classify neighboring points as inliers or outliers. Outliers are discarded from MLS projection computations. This classification is often inconsistent along an edge, which leads to jagged edges in the smoothed result. This method also attempts to synthesize new points lying on sharp edges by finding the intersections of pairs of smooth patches and synthesizing points along these intersections.

The robust statistics approach to extending MLS surfaces culminated with Robust Implicit Moving Least-Squares (RIMLS) by Öztireli *et al* [16]. This method frames MLS as a form of Local Kernel Regression (LKR). This formulation is then used to define a new type of implicit surface which extends Kolluri's implicit MLS surface formulation [60], while naturally incorporating robust statistics [66] into the weighting of point neighbors. The resulting surface is differentiable everywhere, and therefore cannot completely recover sharp edges. Nevertheless, the quality of the feature preservation and computational efficiency have made RIMLS the state of the art in local implicit

methods for surface reconstruction. It has also been made widely available as part of the software Meshlab [67].

Our feature-aware smoothing method (Section 3.3.2) also adopts the approach of discarding points across edges, similar to Fleishman *et al* [64]. Adopting this approach allows us to completely reconstruct sharp edges, unlike RIMLS. Rather than using robust statistics, we identify outliers using an explicitly computed *sharpness field*. In our method, as long as the sharpness field is computed accurately, the problem of jagged edges does not arise. We also place points on sharp edges, but without the assumption that there are exactly two smooth patches intersecting. This is because we push points near edges or corners towards the local maximum of the sharpness field. This allows us to obtain points lying on not only edges, but also corners.

2.2.2 Sharp features in 3D meshes and point clouds

Most of the initial work on handling sharp features in 3D surface data involved adapting techniques for feature-preserving noise removal in 2D images to the 3D domain. One of the early attempts in this direction was made by Tasdizen *et al* [68], who adapted anisotropic diffusion [69] to 3D surfaces represented as level sets. Fleishman *et al* [70] later presented a method for denoising 3D surfaces represented as meshes, by adapting bilateral filtering [71], to operate on the tangent planes of triangular meshes. Jones *et al* had a concurrent work [65], which also aimed to move away from diffusion-based mesh smoothing by applying robust statistics to the tangent planes of meshes. Sun *et al* [72] later refreshed the anisotropic diffusion approach to mesh smoothing by introducing a simpler and more computationally efficient algorithm for filtering vertex normals.

More recently, Zheng *et al* [73] have presented a new adaptation of bilateral filtering which filters vertex normals and has a carefully designed weighting scheme to preserve sharp features. In addition to the traditional local iterative implementation of bilateral smoothing, their work includes a global noniterative implementation which improves preservation of sharp features. Another recent global method by He *et al* [74] adapts L_0 minimization on 2D images [75] to 3D meshes, by replacing the color gradient operator in 2D images with the discrete Laplacian operator [76].

While the methods mentioned thus far have focused largely on triangular meshes, there has also been plenty of work involving point clouds. Gumhold *et al* [77] attempted to identify sharp features in point clouds by classifying individual points as belonging to creases. This classification was done based on hand-crafted rules which consider the shape of the *correlation ellipsoid* of the neighborhood

of each point in the point cloud (in other words, principal component analysis — PCA). Pauly *et al* [78] later implemented a multi-scale extension of this approach, in which the classification of each point considers neighborhoods of different sizes. This method requires a significant amount of parameter tuning.

Weber *et al* [8] depart from the PCA-based approach to classifying points, and instead construct a Gauss map for each point. Sharp feature points are then identified based on the number of clusters found in the Gauss map. The presence of two clusters implies an edge, while the presence of three or more clusters implies a corner. Naturally, this approach is highly dependent on the normal estimation method used prior to the creation of the Gauss map. They also attempt to reduce the number of user-specified parameters by locally adapting some of the parameters used for clustering.

Cao *et al* [79] observe that when smoothing a point cloud by weighted averaging, points lying near sharp features tend to be displaced by a greater amount than other points. They therefore use a threshold parameter for this displacement to classify points as belonging to a sharp edge. This method tends to perform poorly in the presence of noise, although it is nevertheless an improvement over PCA-based approaches. Tran *et al* [80] adopt a similar method, but they use Otsu's Method [81] to automatically select the displacement threshold. Nie [82] also uses a similar method to create a "smooth shrink index", which is used to identify and connect points belonging to sharp edges.

Bazazian *et al* [83] build upon the method of Weber *et al* [8], by replacing the Gauss map clustering with PCA-based clustering of neighboring points. They report improved performance for edges with small dihedral angles. Fu and Wu [84] recently demonstrated a method to classify points as belonging to edges by first eliminating points which are unlikely to belong to edges, and then thresholding the remaining points based on a "linear intercept ratio".

It is worth noting that all of the above methods only perform binary classification of points in the point cloud. Our *sharpness field*, by contrast, is defined over the entire underlying surface, and does not require thresholding (see Section 3.1.1).

Lipman *et al* [63] were among the first to implement feature-aware smoothing for point clouds. They first detect sharp edges in a neighborhood based on the theoretical error of MLS projection. This is similar to many of the aforementioned sharp feature detection methods, which are based on the displacement caused by smoothing. They then fit a polynomial spline to the neighborhood so that its singularities are aligned with the sharp features. This spline then serves as the local domain for function approximation in MLS computations.

Weber *et al* have a follow-up work [85] to their sharp feature detection method which modifies

MLS surface reconstruction using sharp feature points detected using their method [8]. This preserves sharp features better in the reconstructed surface. They accomplish this by fitting a cubic Bezier curve to the detected feature points in each local neighborhood. This curve is used to cluster neighboring points. All neighbors belonging to clusters other than that of the central point are removed for the purpose of MLS computations. We have taken a similar approach in some of our methods for feature-aware smoothing using barrier planes (Section 3.3.1.1). However, our methods allow us to handle a larger variety of edge configurations in local neighborhoods, since we represent edges as multiple polylines instead of as a single parametric curve.

2.2.3 GANs and their applications

Generative adversarial networks (GANs) are a class of neural network generative models introduced by Goodfellow *et al* in 2014 [25]. Previous neural generative models such as variational autoencoders (VAEs) sought to minimize ℓ_1 or ℓ_2 loss when modeling the probability distribution of a set of data. GANs, on the other hand, seek to minimize an *adversarial loss*, which represents how "realistic" a generated sample appears.

GANs are composed of two sub-networks:

- a *generator*, $G(z) : \mathbb{R}^m \rightarrow \mathbb{R}^n$, which attempts to generate realistic samples that appear to be sampled from the distribution of the n-dimensional training data. Different samples are generated for different noise vectors $z \in \mathbb{R}^m$. Typically, $m << n$.
- a *discriminator*, $D(y) : \mathbb{R}^n \rightarrow [0, 1]$, which attempts to classify data samples as real or fake. $D(y)$ is the probability that y is genuinely sampled from the training dataset, and not artificially generated by the generator.

The objective of the GAN training is given by the following adversarial loss function:

$$\mathcal{L}_{GAN} = \mathbb{E}_y[\log D(y)] + \mathbb{E}_z[\log(1 - D(G(z))] \tag{1}$$

Minimizing this loss is equivalent to minimizing the Jensen-Shannon divergence between the probability distribution of the training data and the implied distribution of the generator [25].

The generator and discriminator effectively play a minimax game, which ideally converges to a Nash equilibrium during training. At this point, the trained generator is able to generate data samples which are sufficiently realistic to fool the trained discriminator. The trained generator is

given by:

$$G^* = \arg \min_G \max_D \mathcal{L}_{GAN}(G, D) \tag{2}$$

The GAN formulation described above provides a generative model which generates data samples which realistically appear to be sampled from a particular domain of data. This has been used, for instance, to randomly generate realistic-looking photos of non-existent celebrities, after training on a dataset of photos of celebrities. However, it does not provide any way to control the generated data samples. This limits the utility of the basic GAN architecture.

GANs become truly useful when they are extended to include some form of supervision or control over their learning process. The most popular GAN application is in *domain translation*, that is, translating a data sample from one domain to another structurally similar domain. This began with the seminal work Isola *et al* [17] on using conditional adversarial networks for translating between image domains given a training set of paired examples. For example, their work is able to translate images of street maps into plausible satellite images of the same terrain, as well as between sketches of shoes and plausible photorealistic images of shoes corresponding to the sketches. This GAN application with paired training examples is popularly known as *pix2pix*.

2.2.3.1 pix2pix

The basic idea behind paired image-to-image translation using GANs is to modify the generator and discriminator to be conditioned on an input image, x. This leads to a modified GAN objective:

$$\mathcal{L}_{GAN} = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z))] \tag{3}$$

In addition, it is typical to add a simple ℓ_1 loss to the objective, in order to improve the quality of the low-frequency information in the resulting image:

$$\mathcal{L}_{\ell_1} = \mathbb{E}_{x,y,z} \, | \, y - G(x, z) \, |_1 \tag{4}$$

The trained generator is given by:

$$G^* = \arg \min_G \max_D \mathcal{L}_{GAN}(G, D) + \lambda \mathcal{L}_{\ell_1}(G) \tag{5}$$

where λ is a weighting coefficient for the ℓ_1 loss.

In practice, the noise vector z is introduced implicitly by random dropout of neurons with 50% probability. Furthermore, the discriminator used for image-to-image translation does not attempt to classify the entire input image as real or fake, but rather classifies individual patches (referred to as a *PatchGAN* architecture [17]).

2.2.3.2 CycleGAN

In a follow-up work, Zhu *et al* [27] have presented a method for eliminating the requirement of paired translation examples during training. They have accomplished this by simultaneously training two generators (F and G) and two discriminators (D_X and D_Y), which learn both a forward mapping and a backward mapping between two image domains (X and Y). This architecture is named *CycleGAN*.

The most distinctive feature of training CycleGAN is the *cycle consistency loss* ($\mathcal{L}_{cyc}$), which constrains both generators to produce mappings that are inverses of one another:

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \in X}[|F(G(x)) - x|_1] + \mathbb{E}_{y \in Y}[|G(F(y)) - y|_1] \tag{6}$$

During training, x and y are randomly chosen from the training datasets. The loss function to be minimized is the sum of the cycle consistency loss and the standard GAN losses for the two generator-discriminator pairs.

2.2.3.3 Improving GAN training and image quality

Training GANs is notoriously unstable [86], particularly when attempting to generate high-resolution images (128×128 pixels or larger). The reason for this is that the probability distribution learned by the generator is the union of disjoint low-dimensional manifolds [86] that do not necessarily overlap with the underlying low-dimensional manifold of real-world data [87]. This makes it too easy for the discriminator to learn to differentiate between the two distributions, upon which the generator may be unable to learn anything further. The problem is particularly acute when generating high-resolution images, because the higher resolution allows easier differentiation between generated and training images [88, 89]. In addition, several approaches have been proposed to enable GANs to generate high-resolution images.

Zhang *et al* have described a two-step approach to training a GAN to generate high-resolution images from textual descriptions, which they call StackGAN [90]. In their method, they train

two conditional GANs, referred to as Stage-I GAN and Stage-II GAN. Stage-I GAN learns to generate low-resolution images (64×64 pixels), conditioned on the embedding vector of a piece of text describing the image contents. This embedding vector is obtained by passing the description text through a pre-trained encoder [91]. Stage-II GAN learns to generate high-resolution images (256×256 pixels), conditioned on both the input text as well as the low-resolution output of the Stage-I GAN. The key insight of this method is that by conditioning the Stage-II GAN on a roughly aligned low-resolution image, there is a greater chance of the support of the implied distribution of the GAN intersecting with the support of the distribution of training images.

Karras *et al* were the first to generate detailed 1024×1024 images, by using a unique approach to training GANs. They train their GAN to generate progressively larger images, starting from size 4×4 and doubling the size after each period of training. Convolutional layers are progressively added to the end of the generator and the beginning of the discriminator each time the image size is doubled. The weights of the previously added layers continue to be fine-tuned even after new layers are added. In order to gradually introduce the new layers and avoid drastically changing the old layers, the output of each new layer is linearly interpolated with the upsampled/downsampled output of the most recently added old layer. The interpolation factor is gradually increased from 0 (only output of old layer) to 1 (only output of new layer) over the course of training for a given image size.

Some works have also shown that changing the loss functions for training GANs has benefits to training stability. Arjovsky *et al* showed that minimizing Wasserstein distance instead of the Jensen-Shannon divergence improves the stability of GAN training [92]. Isola *et al* used an ℓ_1 loss to capture low-frequency information in their image translation method. Zhang *et al* recently presented a method [93] for removing rain from photographs using a conditional GAN, in which the loss function includes a perceptual loss based on the feature loss at layer `relu2_2` in a pretrained VGG-16 model [94].

These methods give a broad background of GAN applications. Since we can easily compute paired examples of local heightmaps for point clouds and triangular meshes, our domain translation method for point cloud super-resolution is based on the paired image translation method of Isola *et al* [17]. All of our heightmaps have 64×64 pixels, therefore it is unnecessary to apply the aforementioned techniques for generating large images. As we shall see in Chapter 5, our heightmaps are already sufficiently large to obtain 256x super-resolution of point clouds (as in the case of our *superdense* mode).

2.2.4 Point cloud super-resolution

Point cloud super-resolution is fundamentally a different problem than depth image super-resolution [95, 96], which mainly deals with 2D grids instead of non-Euclidean data. Furthermore, depth images are almost invariably accompanied by color or intensity images, which provide vital information that is exploited in past work. Point cloud super-resolution is related to the older problem of point cloud *consolidation*, where the goal is to obtain a point cloud representation from which an accurate 3D mesh can be reconstructed [97]. A variety of procedural point cloud consolidation methods have been proposed, including LOP [98], WLOP [97], EAR [99] and deep points consolidation [100]. All of these methods involve fitting local geometry to point clouds, and WLOP and EAR have been incorporated into popular geometry processing libraries.

Recent years have seen a wave of interest in applying deep learning to point clouds, sparked by the success of PointNet [1] and its multi-scale variant, PointNet++ [3], in the field of point cloud classification and semantic segmentation. This has led to point cloud consolidation [39] and super-resolution [101] approaches based on PointNet, as well as a family of point cloud upsampling techniques based on PointNet++ that includes PU-Net [4], EC-Net [5] and patch-based progressive upsampling [6]. Our super-resolution method in Chapter 5 uses local heightmaps, much like Roveri *et al* [39]. However, their work focuses on consolidation of noisy point clouds with 50 thousand points, while we are mainly interested in super-resolution of much sparser point clouds (5000 points or less). Moreover, they do not use domain translation as we do, since their ground truth data is also obtained from point clouds.

In the remainder of this section, we summarize the state-of-the-art point cloud super-resolution approaches, to provide context for the comparisons with our method in Chapter 6.

The current state-of-the-art methods for point cloud super-resolution and upsampling are all deep learning approaches based on the PointNet [1] architecture. These include the work by Zhang *et al* [101] on training a PointNet-based neural network to upsample an entire point cloud in a single step. Our super-resolution approach is not really comparable to this method because we upsample each local neighborhood independently. More relevant to our work is a family of recent neural network architectures [4, 5, 6] which upsample local patches of points from point clouds. These neural network architectures are all based on the multi-scale extension of PointNet, called PointNet++ [3].

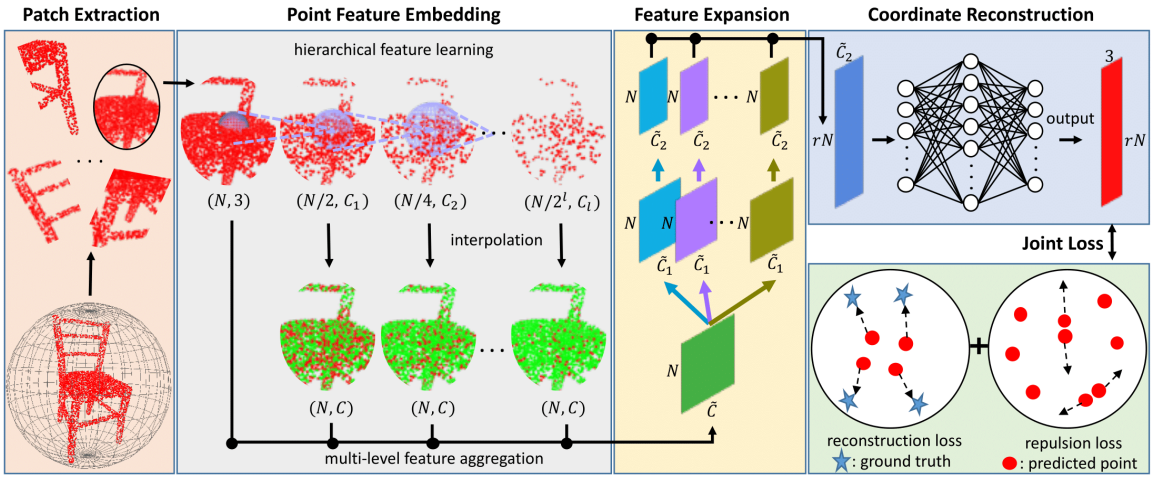

Figure 4: PU-Net neural network architecture (image by Yu *et al* [4]).

The comparisons of our super-resolution method in Chapter 6 mainly focus on the 3PU architecture proposed by Wang *et al* [6], since it is the most recent and best-performing state-of-the-art point cloud super-resolution method. Some comparisons with EC-Net [5] are also provided, since it is the most recent method before 3PU.

Since PointNet++ is integral to all 3 state-of-the-art point cloud super-resolution methods listed in this section, it is worth reviewing its workings in Section 2.1.2.3 before proceeding. Also, note that in the context of PointNet++-based methods, "features" and "feature vectors" refer to the results of intermediate computations in the neural network. These terms have no relation to geometric features such as edges or creases. A feature vector associated with a point in a point cloud is often high-dimensional, and contains implicit geometric information about the neighborhood of the point. This high-dimensional vector cannot be directly interpreted, but it nevertheless contributes to the final result of the neural network.

2.2.4.1 PU-Net

PU-Net [4] is the first deep learning-based point cloud super-resolution method. The authors show that the point cloud segmentation architectures for both PointNet and PointNet++ can be naïvely adapted to point cloud super-resolution, by treating the coordinates of 4 new points for every original point as segmentation classes predicted by the network. However, they greatly improve super-resolution results by using the PU-Net architecture shown in Figure 4. PU-Net mainly involves extending PointNet++ with additional properties that aid super-resolution. An input point cloud is first divided into several patches which are processed separately. The set abstraction and feature

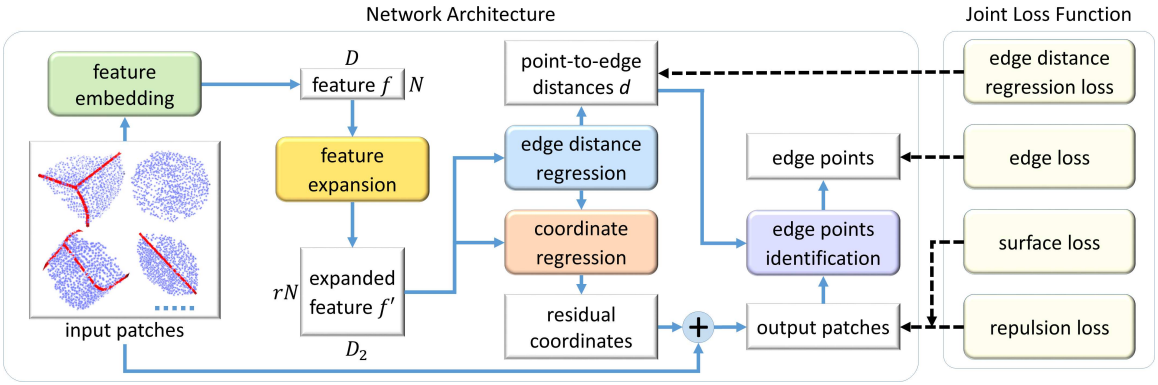

Figure 5: EC-Net neural network architecture (image by Yu *et al* [5]).

propagation layers from PointNet++ are used to compute a hierarchical set of feature vectors for all points of the patch, which are then concatenated along with the point positions to form a single matrix of feature vectors for the points. This matrix is duplicated into 4 copies which undergo separate 1×1 convolutions to obtain a new matrix with a new set of feature vectors for 4 times the original number of points. These feature vectors are then passed to additional 1×1 convolutional layers (erroneously referred to as "fully connected layers" in [4]), which convert the feature vectors into point coordinates. This gives a new point cloud patch with 4 times the original number of points.

There are two loss functions which are combined to form the optimization objective during training. The first is the Earth Mover's Distance [102], which is the sum of all distances between corresponding points of two sets of points, for the bijective mapping that minimizes this sum (this requires creating ground truth data with the same number of points, so that one can have a bijection). The Earth Mover's Distance is not differentiable, which is a requirement for a neural network loss function. However, Fan *et al* [103] have provided code for a differentiable approximation of the Earth Mover's Distance along with the implementation of their method.

Besides the Earth Mover's Distance, the neural network also needs to minimize a *repulsion loss*. This acts as a regularizer to prevent newly generated points from clumping together and instead encourage them to spread out evenly. For an output point and one of its nearest neighbors, the repulision loss for the pair of points is $-re^{-r^2/h^2}$, where r is the distance between the two points and $h = 0.03$ is a parameter for controling the decay of the repulsion with distance.

2.2.4.2 EC-Net

EC-Net [5] is a minor extension of PU-Net which attempts to improve point cloud upsampling near sharp edges. The authors manually annotated sharp edges onto 3D meshes from the ShapeNet [104] dataset, as well as other undisclosed sources. These edges are then used to train the network to estimate the distance of generated points to sharp edges. An additional loss function is added to encourage points near edges to lie closer to sharp edges, by minimizing their estimated distance to edges. (In practice, we were not able to reproduce results with sharp edges such as those in the paper, since the paper does not provide enough details on the steps to reproduce their result figures from the output point clouds.) The complete network architecture is shown in Figure 5, with the edge-related components being the main differences from PU-Net.

2.2.4.3 3PU

3PU [6] is the most recent state-of-the-art method for point cloud super-resolution. The authors report performance superior to PU-Net, EC-Net and the older Edge-Aware Resampling (EAR) [99] technqiue, which does not use machine learning. The idea behind 3PU is to progressively train a neural network to upsample a point cloud up to 16 times the original resolution, with a neural network containing 4 "upsampling units" that double the number of points. Each upsampling unit focuses on upsampling one particular level of detail, i.e. either the input point cloud itself or the output of one of the previous upsampling stages. The higher-resolution upsampling units are initially kept frozen during training, and then progressively activated one by one. Figure 6 shows the structure of one upsampling unit. It extracts a hierarchical set of features from an input point cloud patch and duplicates them to perform super-resolution, much like PU-Net. However, the details of the implementation are not identical to PU-Net and PointNet++.

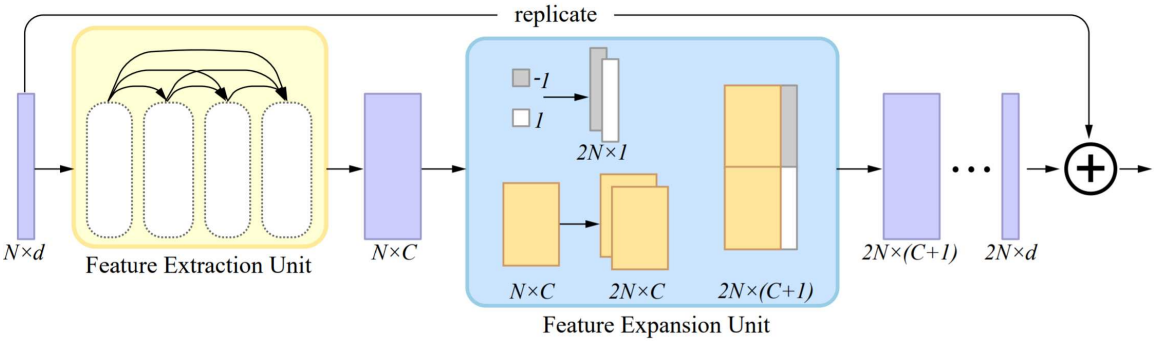

Figure 6: Illustration of an upsampling network unit in 3PU (image by Wang *et al* [6]).

As we have seen in this chapter, machine learning has never been used for sharp feature detection or recovery in 3D point clouds. Our methods presented in the following chapter (Chapter 3) are therefore the first to attempt sharp feature detection using machine learning. The state-of-the-art point cloud super-resolution techniques do use deep learning, but they share certain inherent limitations (see Section 6.1.1). In Chapter 5, we propose a completely new method which is free of these limitations, while also providing more accurate results and better computational efficiency.

Chapter 3

Methods for Sharp Edge Reconstruction

This chapter describes the methods we have developed for sharp edge reconstruction in 3D point clouds [7, 10]. Section 3.1 describes how machine learning can be used to compute a *sharpness field* for a point set surface. Since we find that certain edges are not fully captured by this method, we evaluate different neighborhood representations and neural network architectures to improve sharpness field computation in Section 3.2. Finally, we describe feature-aware smoothing algorithms and other applications of the sharpness field in Section 3.3.

3.1 Sharpness Field Computation

The reconstruction of sharp edges in scanned objects presents an inherent challenge. Sharp edges are discontinuities in the normal field of the surface of an object. This means that electromagnetic rays received from the scanned object are extremely unlikely to have reflected off a sharp edge. Therefore, we can be certain that in a point cloud of a scanned 3D object, *none* of the points lies exactly on a sharp edge. This means that edges of the object cannot be accurately reconstructed by fitting lines or curves to edge points. Nevertheless, there have been previous attempts to detect edge points by posing the problem as one of binary classification of points into points near edges and points away from edges (see related work in Section 2.2.2). These have the obvious limitation that they cannot localize sharp features that lie between sampled points in the point cloud.

We propose an alternative approach to detecting sharp edges in point clouds. Rather than classifying individual points in the point cloud, we define a scalar field called the *sharpness field* over the underlying surface defined by the point cloud. The goal of the sharpness field is to have an explicit sharp feature representation which is defined over a continuous domain, allowing one to localize sharp features at any point on the underlying surface of a point cloud. As we shall see in Section 3.3, computing a sharpness field has a number of applications which stem from the ability to find the locations of sharp edges accurately. Section 3.1.1 defines the sharpness field and provides further details. In order to provide a continuous domain for the sharpness field, we typically use a *proxy surface*, which is an MLS approximation of the underlying surface of the point cloud. Since MLS projections can be accomplished by many methods, and they are heavily used in this chapter, Section 3.1.2 explains the method we use for all MLS projections.

The sharpness field for a given point set can be computed by feeding local neighborhoods of the point set to a machine learning model, which outputs scalar values of the sharpness field at the center of each neighborhood. This amounts to an implicit representation of the sharpness field, i.e. the field is represented by a combination of the point cloud and the trained machine learning model. Section 3.1.3 goes into further detail on how to frame sharpness field computation as a machine learning problem. Section 3.1.4 describes a method for passing point cloud neighborhood information to a machine learning model, which we found to work best with convolutional neural networks (CNNs). The results and evaluation of other machine learning models are given in Section 4.1.

3.1.1 Definition of the sharpness field

In order to define the sharpness field, we must first clarify the scope of the “sharp edges" considered in our work. We consider the sharp edges of a 3D shape to be discontinuities in the normal field of a surface. These discontinuities form a graph whose edges are smooth curves joining corners of the shape. The sharpness field is a scalar field defined over the surface of the shape. It takes the value 1 at edge points or corners, and quickly tapers linearly to 0 at a certain scale-dependent distance from the the nearest point with value 1. Edge and corner points are thus local maxima and ridges of the sharpness field.

Since we are dealing with point clouds, we do not have an explicit representation of the shape, but instead a set of points sampled by a scanning device. This scanning device may have introduced some noise into the measurements. If the sampling of the points is roughly uniform, we can use

the sampled points to obtain a smooth Moving Least-Squares (MLS) surface approximation of the underlying surface, which we call the *proxy surface* of the point cloud. The proxy surface serves as an approximate continuous domain for the sharpness field in the absence of the ground truth continuous surface.

The proxy surface is smooth, unlike the original surface, which has sharp edges. However, we shall see that its role as the domain of the sharpness field still enables the localization of sharp features. This is because every point on the proxy surface can be regarded as the MLS projection of a point on the original noiseless shape. We can therefore identify the locations of the original edges by locating points that project to local maxima or ridges of the sharpness field computed on the proxy surface of an input point cloud.

In our experiments, we choose to have the sharpness field taper to 0 at a distance of 4δ from the nearest local maximum, where δ is the median distance between a point and its nearest neighbor in the point cloud. δ gives us an approximate measure of the sampling density of the point cloud.

3.1.2 Moving Least-Squares projection

Algorithm 1 Algorithm for Moving Least-Squares Projection

procedure $\overline{M}_r(\tilde{\mathbf{p}}, \tilde{\mathbf{N}}, \tilde{P})$ ▷ Input: center point, normal, point cloud with normals
  $\bar{\mathbf{p}} \leftarrow \tilde{\mathbf{p}}$ ▷ Projected point position
  $\bar{\mathbf{N}} \leftarrow \tilde{\mathbf{N}}$ ▷ Projected point normal
  for $i \leftarrow 1$ **to** 5 **do** ▷ Optimization iterations
    $\mathbf{x}_{avg} \leftarrow 0$ ▷ Weighted average of neighbors of $\bar{\mathbf{p}}$
    $\mathbf{N}_{avg} \leftarrow 0$ ▷ Weighted average of normals of neighbors of $\bar{\mathbf{p}}$
    $w_{sum} \leftarrow 0$ ▷ Sum of weights of neighboring points
    for $\mathbf{x}, \mathbf{N}$ **in** $\tilde{P}$ **where** $\mathbf{x} \in B(\bar{\mathbf{p}}, r)$ **do** ▷ Loop over neighbors of $\bar{\mathbf{p}}$
      $w \leftarrow \exp\left(-\frac{\|\mathbf{x}-\bar{\mathbf{p}}\|^2}{2(r/3)^2}\right)$
      $w_{sum} \leftarrow w_{sum} + w$
      $\mathbf{x}_{avg} \leftarrow \mathbf{x}_{avg} + w\mathbf{x}$
      $\mathbf{N}_{avg} \leftarrow \mathbf{N}_{avg} + w\mathbf{N}$
    end for
    $\mathbf{x}_{avg} \leftarrow \frac{\mathbf{x}_{avg}}{w_{sum}}$
    $\mathbf{N}_{avg} \leftarrow \frac{\mathbf{N}_{avg}}{\|\mathbf{N}_{avg}\|}$
    $\bar{\mathbf{p}} \leftarrow \bar{\mathbf{p}} + [(\mathbf{x}_{avg} - \bar{\mathbf{p}}) \cdot \mathbf{N}_{avg}]\mathbf{N}_{avg}$
    $\bar{\mathbf{N}} \leftarrow \mathbf{N}_{avg}$
  end for
  return $\bar{\mathbf{p}}$, $\bar{\mathbf{N}}$
end procedure

We compute the projection of a point onto the proxy surface using Algorithm 1, which is adapted from Section 4.2 of the list of methods for computing MLS projections compiled by Alexa *et al* [58].

This algorithm requires an approximate oriented normal to be known at every point of the input point cloud. This is not a major constraint, since unoriented normals can easily be estimated using principal components analysis or deep learning approaches [105, 2], after which they can be oriented using the well-established method proposed by Hoppe *et al* [106].

The parameter r is variously called the *smoothing radius*, *kernel radius* or *search radius*. The standard deviation of the Gaussian function used to weight neighboring points is given by $r/3$. Since the Gaussian function is already negligibly small at three standard deviations from the mean, we do not consider neighboring points outside this radius when computing weighted averages. A large smoothing radius implies a large standard deviation for the Gaussian function, which will lead to a very coarse and smooth approximation of the underlying surface.

We use a small kernel radius $r = 3\delta$ for MLS projections onto the proxy surface. The purpose of using MLS for the proxy surface is merely to provide a continuous domain, so we keep the kernel radius as small as possible to avoid losing important features due to smoothing. At the same time, the radius should be large enough that it includes at least one neighboring point for all points in the point cloud.

3.1.3 Machine learning approach to sharpness computation

Machine learning approaches to computing a sharpness field for a point cloud require the following components:

1. A dataset of shapes with ground truth sharpness fields, used for training a machine learning model.

2. An appropriate machine learning model which can learn from the available training data and generalize to previously unseen data.

3. A method of representing the local neighborhood of a point in a point cloud, which can be fed as input to the machine learning model.

We experimented with several different machine learning models for computing the sharpness field, including both "deep" models such as neural networks, as well as "shallow" models such as gradient boosting machines and random forests. Qualitative and quantitative comparisons for the different machine learning models are given in Section 4.1.2.

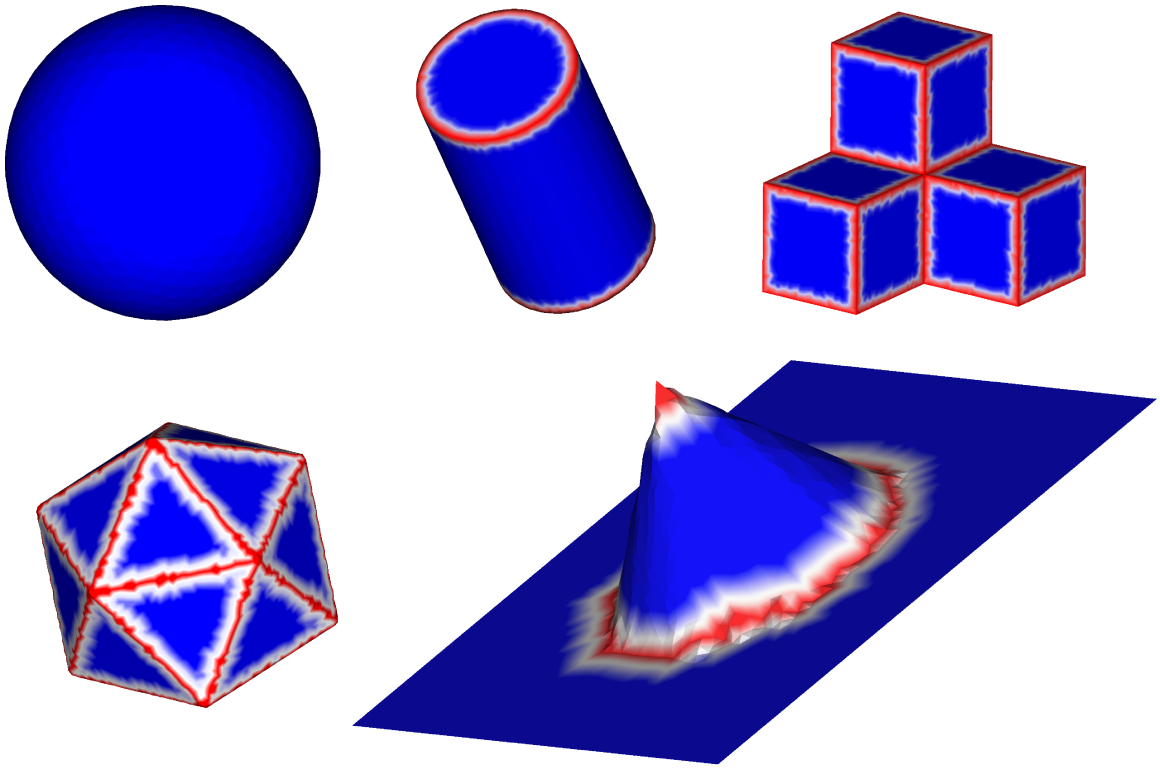

Figure 7: Our initial training set [7], rendered with ground truth sharpness fields.

3.1.3.1 Preparation of training data

In order to learn sharpness field computation, the machine learning model must be presented with a sufficiently diverse set of point neighborhoods during training. We have therefore devised a set of training shapes whose surfaces contain a variety of sharp edges, corners, as well as smooth regions (Figure 7). These shapes include:

1. A sphere (no sharp features).
2. A cylinder.
3. A cone whose base lies on a planar sheet (the tip of the cone and the points intersecting with the sheet are sharp features).
4. An icosahedron (all edges and corners of the icosahedron are sharp features).
5. The union of four cubes, arranged in such a way as to maximize the variety of concave and convex edges and corners.

These shapes are represented as meshes instead of point clouds, since this makes it easy to compute ground truth sharpness fields using dihedral angles. Every mesh is remeshed to have a large number of vertices, so that the set of vertices may be treated as a dense point cloud representation of the shape. Each mesh contains around 3,000 to 12,000 points, yielding a total of around 40,000 points. Note that the mesh connectivity does not play any role when computing the sharpness field at testing time. It is only used for generating training and validation data, and for rendering results. Each of the five models has its coordinates normalized to the range $[-1, 1]$. They are also remeshed to make all edge lengths approximately 0.04, while preserving sharp edges, using the remeshing algorithm of Surazhsky *et al* [107], as implemented in the software Graphite 2. This is how we ensure that our training data is a densely and uniformly sampled set of points.

To generate the ground truth sharpness field, we search for every edge in each mesh whose adjacent faces make an angle greater than 30 degrees, and mark the vertices of the edge with sharpness value 1. For the cone and the icosahedron, it is necessary to manually connect a few gaps in the sharpness field after the automated marking. We then extend the sharpness field to neighboring vertices while tapering the sharpness value based on the approximate geodesic distance, as mentioned in Section 3.1.1. The remaining vertices are assigned a sharpness value 0.

With the five aforementioned ground truth training models, we generate two additional sets of five models with two different levels of noise (random perturbation along the normal direction by 1.5δ or 3δ). This gives us a training set with a total of around 120,000 training examples, each containing a single point neighborhood and a single sharpness value.

As we shall see in later sections, the aforementioned set of training data is sufficient to train machine learning models for sharpness field computation for our *normal angles* neighborhood representations (Section 3.1.4).

3.1.4 Sharpness field computation using normals sampled on a radial grid

Although our input is a point cloud, the original 3D shape from which the points were sampled is assumed to be a closed 2D surface. Therefore it makes sense to define a local neighborhood representation which captures the local curvature of the neighborhood. A radial grid centered at the center of a given neighborhood would make for a useful neighborhood representation, especially since a grid-like structure effectively parametrizes the local neighborhood so that it can be treated as Euclidean data. Masci *et al* have used a similar radial grid in their proposed extension of convolution

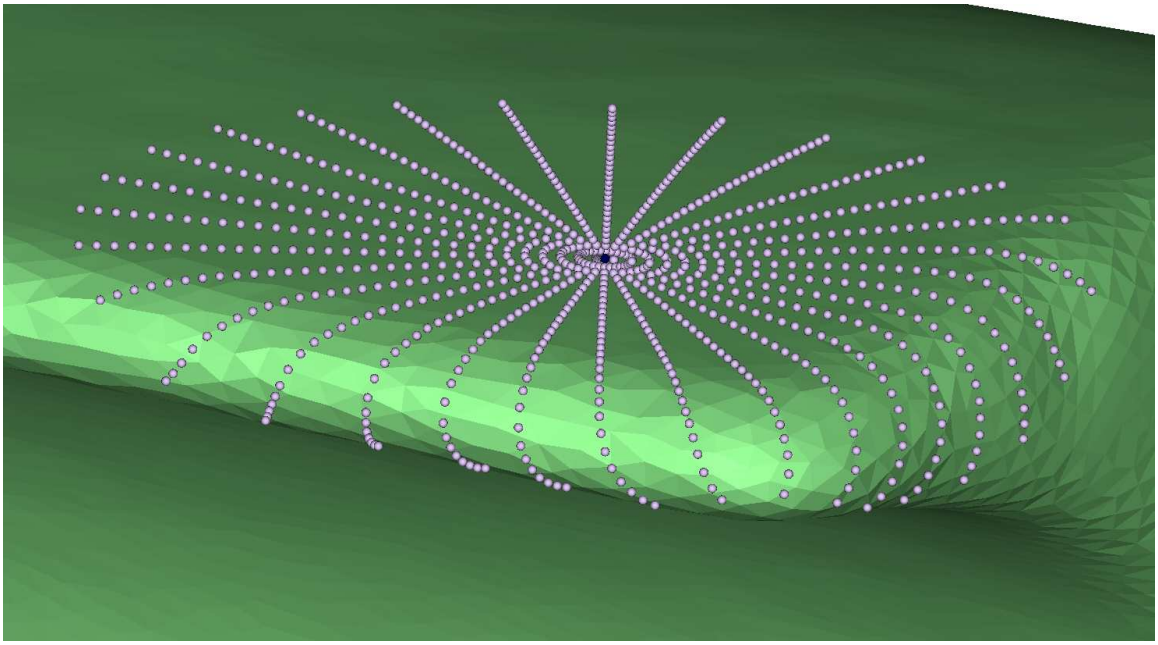

Figure 8: Example of points sampled in a 30×30 radial grid on a proxy MLS surface.

to manifolds [41]. However, the method they use for unfolding the grid is only applicable to meshes. We therefore define a novel unfolding algorithm which unfolds a radial grid consisting of 30×30 uniformly spaced points over an MLS surface. We can see an example of this radial grid in Figure 8. Since the points are sampled from an MLS surface, we can easily obtain the normal at each point as well. The details of this unfolding procedure are given in Section 3.1.4.1.

This radial grid allows us to define a neighborhood representation as follows: we compute the unsigned angle between each sampled normal and the normal at the center of the radial grid. This gives us an input representation consisting of 900 angles, which is invariant to global translation and global scaling of the point cloud, but not to global rotation. One approach to making the representation rotation-invariant is to sort the angles in each ring (row) of the radial grid in ascending order. This is the approach we used in [7]. Figure 9 shows a visualization of this representation for different kinds of neighborhoods. Another approach to rotation invariance is to shift the burden of rotation invariance onto the neural network by randomly performing cyclic permutations of the spokes (columns) of the radial grid during training. This effectively augments the training data.

We feed this collection of 900 angles to the convolutional neural network shown in Figure 10. This type of neural network architecture is often seen in 2D image processing. We are able to use it for our non-Euclidean local neighborhoods thanks to the parametrization provided by the radial

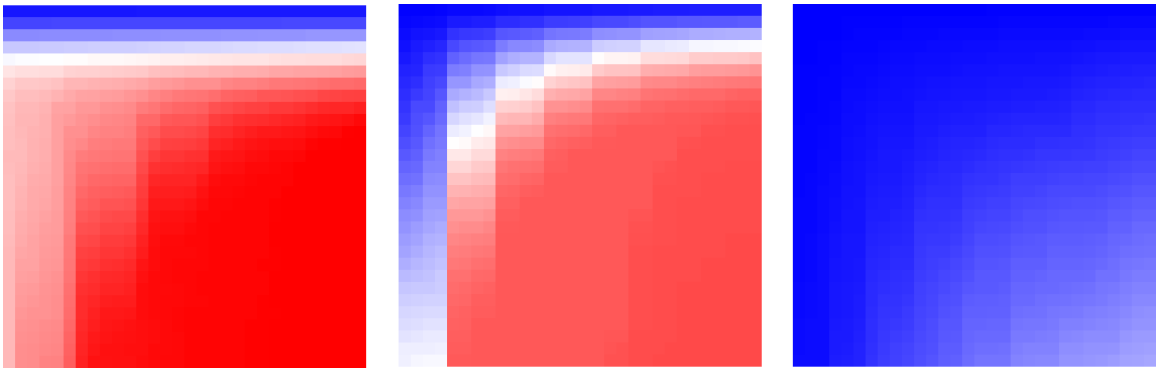

Figure 9: Local neighborhood representation of 30×30 sorted angles for three points selected from the *fandisk* point cloud: (L to R) a point on a corner, a point on a sharp edge and a point on a smooth patch. The top row of each image is the innermost ring of the radial grid. Each row is sorted in ascending order. For clearer visualization, we colorize the values with blue being the minimum, red being the maximum and white being halfway in between.

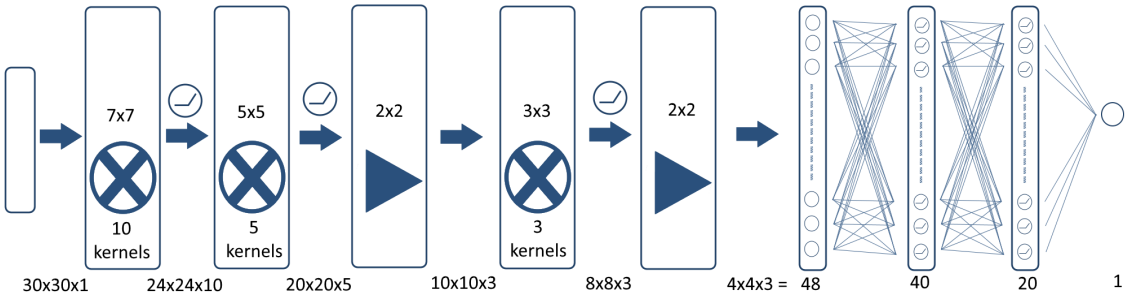

Figure 10: CNN architecture used in Section 3.1.4

grid. Although it seems reasonable to constrain the output of the network to the range $[0, 1]$, which is the range of values taken by the sharpness field, we found in practice that we obtained smoother sharpness fields by not applying a sigmoid activation function at the end to explicitly constrain the output to this range. We use the mean squared error between the predicted and target sharpness values as the loss function for training.

3.1.4.1 Radial grid unfolding algorithm

Algorithm 2 Radial grid unfolding algorithm

procedure UNFOLD($\tilde{p}, \widetilde{N}, R, n_i, n_j$) ▷ Input: center point, normal, radius of grid, number of radial divisions, number of angular divisions
  $\bar{p}, \bar{N} \leftarrow \overline{M}_r(\tilde{p})$
  $t_1, t_2 \leftarrow$ arbitrary tangent and bitangent for normal $\widetilde{N}$
  for $j \leftarrow 1$ **to** n_j **do**
    $\theta \leftarrow \frac{2\pi}{n_j}(j - 0.5)$
    $T_0 \leftarrow I$
    $T_1 \leftarrow I$
    $\bar{G}_{0j}, \bar{N}_{0j} \leftarrow \bar{p}, \bar{N}$
    for $i \leftarrow 1$ **to** n_i **do**
      $x \leftarrow \frac{R}{n_i}(i - 0.5)$
      $\widetilde{G}_{ij} \leftarrow \tilde{p} + x \cdot (\cos\theta \cdot t_1 + \sin\theta \cdot t_2)$
      $\bar{G}_{ij}, \bar{N}_{ij} \leftarrow \overline{M}_r(T_{i-1} \cdot \widetilde{G}_{ij})$
      if $i > 1$ **then**
        $T_i \leftarrow T_{i-1} \cdot$ (rotation by the angle between $\bar{N}_{i-2,j}$ and $\bar{N}_{i-1,j}$, about the axis defined by the point $\bar{G}_{i-2,j}$ and the vector $\bar{N}_{i-2,j} \times \bar{N}_{i-1,j}$)
      end if
    end for
  end for
  return all $\bar{G}_{ij}$ and $\bar{N}_{ij}$ ▷ Output: grid points $\bar{G}_{ij}$ and their normals $\bar{N}_{ij}$ on $\bar{S}_r$
end procedure

The algorithm shown above takes a point $\tilde{p}$ from the input point cloud and yields a set of $n_i \times n_j$ points on a radial grid of geodesic radius R, sampled on the proxy surface $\bar{S}_r$. The proxy surface is an implicit MLS surface, computed with a smoothing kernel of radius r. The proxy surface is defined by the projection operator $\overline{M}_r(\tilde{x})$, which takes a point $\tilde{x}$ near the proxy surface and maps it to a point and its corresponding normal on the proxy surface.

The basic idea behind the algorithm is to first generate radial grid points $\widetilde{G}_{ij}$ in the tangent plane of the center point $\tilde{p}$, and then move outward along each spoke j, deforming the spoke so that it lies on the proxy surface. This cannot be done simply by projecting each $\widetilde{G}_{ij}$ using $\overline{M}_r$, as the Euclidean distances between the points $\widetilde{G}_{ij}$ would not be preserved as geodesic distances after projection. Instead, at each spoke point i we compute a cumulative rigid transformation T_i, which

moves the following spoke point close to its desired location on $\bar{S}_r$ before applying $\overline{M}_r$ to make it snap to the proxy surface. This is done by approximating the deformation of the previous points on the spoke as a sequence of rotations centered on previous points (see Algorithm 2 for details).

In practice, we set $r = 3\delta$ and $R = 8\delta$. As mentioned in Section 3.1.1, δ is our measure of the scale of a point cloud, and is defined as the median distance between a point and its nearest neighbor. The smoothing radius r is meant to be small, so as to avoid smoothing important features. Its only purpose is to facilitate the resampling of the proxy surface performed in Algorithm 2. As shown in Figure 11, the algorithm is reliable for moderate levels of noise, where the proxy surface does not have too much high-frequency noise.

The resulting sharpness fields can be found in Section 4.1.1 (page 66).

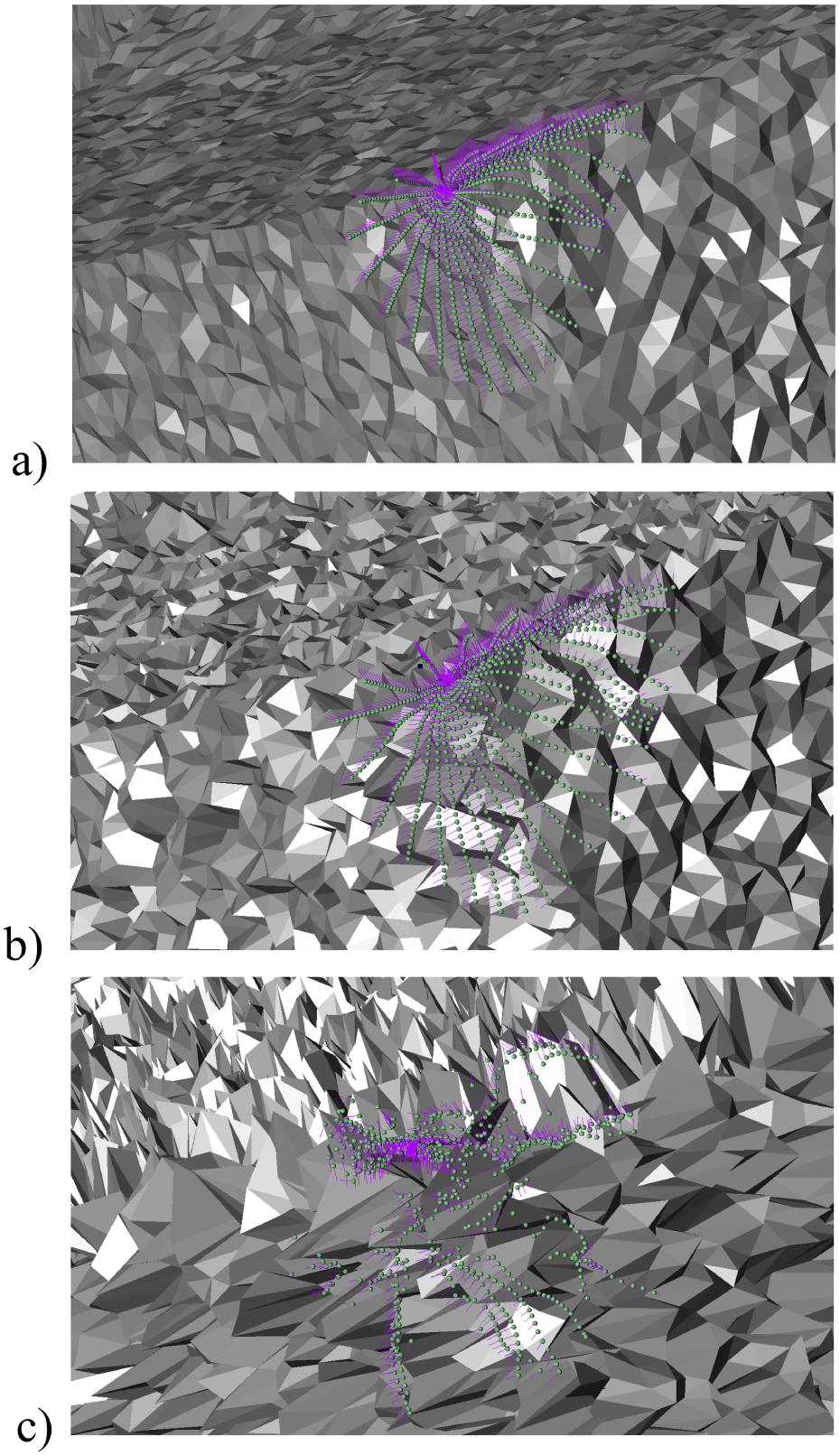

Figure 11: Unfolding breaks only for higher noise levels exceeding 15δ: (a) 1.5δ, (b)4.5δ, (c) 15δ.

3.2 Representations and Models for Sharpness Field Computation

Using the methods described in the previous section, we have obtained the results given in Section 4.1 (page 65). These results validate that our *normal angles* neighborhood representation, along with CNNs, can be used for sharpness field computation. However, this formulation is insufficient to compute the sharpness field in some cases, such as certain types of curved edges. This can be seen clearly for the *blade* model, as shown in Figure 32 (page 72). We believe that this is because important geometric information is lost over the course of the steps involved in computing the *normal angles* neighborhood representation. We can plainly see that there are several steps involved where geometric information can be lost:

1. Using neighboring points to compute projections onto an implicit Moving Least-Squares surface (the *proxy surface*).
2. Resampling the proxy surface to create a radial grid.
3. Discarding the positions of radial grid points in favor of the normals at these points.
4. Computing unsigned angle differences between these normals and the normal at the center of the radial grid.

In this section, we investigate other neighborhood representations that can be used for sharpness field computation. These neighborhood representations are devised by progressively rolling back the aforementioned steps, so as to preserve more geometric information:

1. In Section 3.2.1, we generate a heightmap from the radial grid instead of using the normals at grid points. We feed these heightmaps to a standard CNN. Since we are still using the radial grid, rotation invariance can be achieved in the same manner as for the unsorted normal angles mentioned in Section 3.1.4.
2. In Section 3.2.2, we leave aside the radial grid entirely, and instead generate a Cartesian heightmap from the neighboring points. This requires applying newer convolutional neural network architectures to achieve rotation invariance, including *harmonic networks* [9] and *spatial transformer networks* [15].

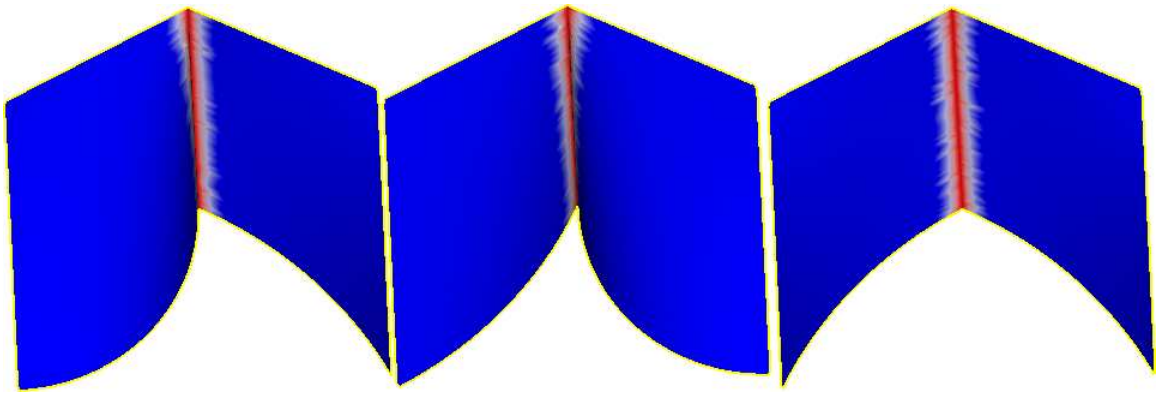

Figure 12: Wedge models with different sharp feature profiles for additional training examples, inspired by Figure 7 in [8].

3. In Section 3.2.3, we experiment with passing the raw coordinates of neighboring points to a neural network. Since this neighborhood representation is completely lacking in structure, it is incompatible with convolutional neural networks. Therefore we must use the new PointNet [1] architecture and its multi-scale variant, PCPNet [2]. Note that both of these architectures internally contain 3D spatial transformer networks in order to achieve invariance to affine transformations.

Having more geometric information in the local neighborhood representation necessitates a greater variety of training data. We therefore added the wedge shapes shown in Figure 12 to the training dataset. These shapes contain new types of edges at the junctions of concave and convex curved sheets, which do not exist in the original training dataset. It is worth noting that simply adding these shapes to the training data did not show any improvement in performance using the method described in Section 3.1.

3.2.1 Radial grid heightmaps

The radial grid described in Section 3.1.4 can be easily used to generate a heightmap by calculating the signed distance of each of the 900 points on the grid to the local tangent plane of the point at the center of the grid. Some examples of this type of heightmap are visualized in Figure 13. Unlike in a Cartesian heightmap, the radial axis represents geodesic distance and not Euclidean distance. This type of heightmap restores some of the information lost by using only the angles between normals, but it cannot distinguish between a smooth S-shaped curve and a sharp U-shaped bend.

For this type of neighborhood representation, we can use the same CNN architecture described in

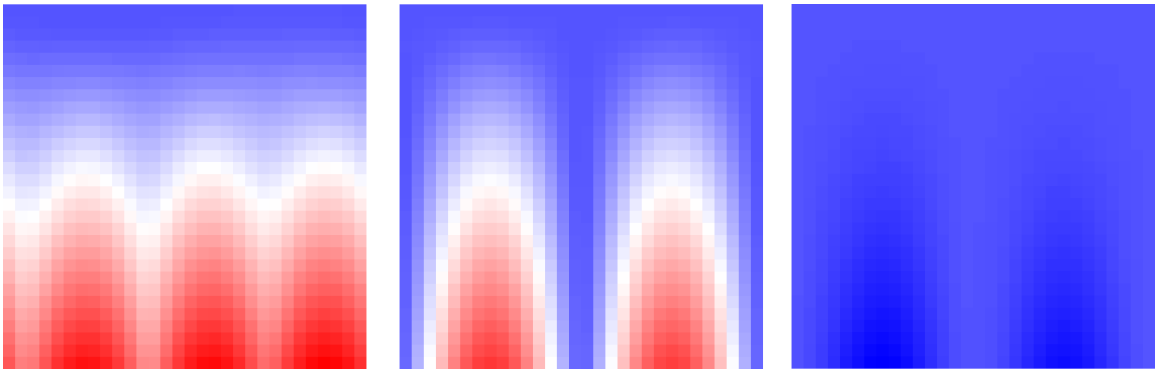

Figure 13: Visualization of radial heightmaps (with a blue-white-red color map). (L to R) A point on a corner, a point on a sharp edge and a point on a smooth patch.

Section 3.1.4. One difference is that each convolutional layer is followed by batch normalization. The main change required in training is to randomly perform cyclic permutations of the spokes of the grid during training in order to enable the network to learn rotation invariance. We also experimented with enforcing permutation invariance using average pooling, similar to how PointNet [1] achieves permutation invariance using max pooling. The resulting sharpness field showed zigzag patterns around edges, without the desired smoothness that we expect of the sharpness field.

3.2.2 Cartesian heightmaps

The high computational cost of performing 900 MLS projections per input point necessitates a batched GPU implementation of the radial grid computation. This reduces the flexibility of the training process and makes it harder to perform data augmentation. With these limitations in mind,

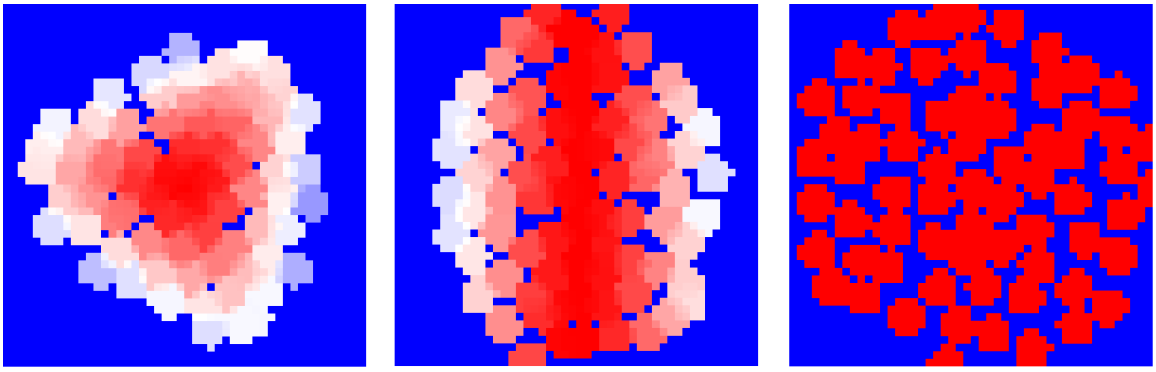

Figure 14: Visualization of the Cartesian heightmaps described in Section 3.2.2 (with a blue-white-red color map). (L to R) A point on a corner, a point on a sharp edge and a point on a smooth patch.

we adapted the heightmap generation algorithm from Roveri *et al* [39]. Compared to the radial grid, these heightmaps are more conventional because they use a Cartesian coordinate frame, and do not follow the curvature of the surface. Instead, all neighboring points in a spherical neighborhood of a point in the point cloud are projected onto the local tangent plane, and their signed distances are interpolated with a Gaussian kernel when discretizing the heightmap image. Unlike Roveri *et al*, we uniformly scale the spherical neighborhood by the reciprocal of the search radius. This affects the height values by reducing dependence on the size of the search radius. We also make use of the fact that we have oriented normals available at each input point, by culling back-facing neighbor points so that they do not contribute to the heightmap.

We now describe the heightmap generation method in detail. The heightmap is assigned a size of $k \times k$ pixels, where we select $k = 64$. The side of the heightmap corresponds to a length of $2r$ in the space of the input shape, where r is the *search radius* used to collect nearest neighbors from the point cloud. The local coordinate frame of the neighborhood is centered on a particular point $\mathbf{p}$ of the point cloud, which has a corresponding oriented normal $\mathbf{n}$. The horizontal and vertical directions of the local frame are given by an arbitrary tangent $\mathbf{t}$ and its corresponding bitangent $\mathbf{b} = \mathbf{n} \times \mathbf{t}$.

A random set of neighboring points (limited to 1024) is chosen from within the search radius r. Neighbors with back-facing normals are omitted. After centering the neighbors by subtracting $\mathbf{p}$, the neighborhood is then scaled by a factor of $1/r$. These points are then projected orthogonally onto the plane containing the point $\mathbf{p}$ with normal $\mathbf{n}$, which is also the plane of the heightmap image. For each pixel in the heightmap image, we compute the corresponding pixel center in the local coordinate frame of the neighborhood. We then compute the intensity of each pixel as the weighted average of the signed distances of nearby projected points from their original positions. When computing this weighted average, the unnormalized weights have a Gaussian falloff $w_i = exp(-\frac{d_i^2}{2\sigma^2})$, where d_i are the distances of the projected points from the pixel center in the image plane (we set $\sigma = 5r/k$). The weights are normalized by dividing them by $\sum w_i$. A constant value 1 is added to all projection distances, so that the value 0 is reserved for unoccupied pixels.

Deep learning frameworks typically rely on building directed acyclic graphs of computations that operate on tensors with compatible dimensions. Resorting to conventional programming techniques such as loops when defining neural network architectures often results in significantly downgraded performance, when compared to using tensor operations. These tensor operations are highly optimized to exploit parallel computing (e.g. SIMD instructions, multiple CPU cores and GPUs).

Therefore, in order to be able to compute neighborhood representations online during neural network training, it is important to have an algorithm which can be easily implemented using the tensor operations provided by deep learning frameworks. Unlike the radial grids described in previous sections, the Cartesian heightmaps that we use here satisfy this requirement. One caveat is that there must be a limit on the number of neighboring points passed to the heightmap generation algorithm (which is why we set a limit of 1024 neighbors). Otherwise, collections of neighboring points from multiple neighborhoods cannot be combined into a single tensor for parallel computation. Following Roveri *et al*, we make a random choice of the neighbors when the number of neighbors exceeds this limit. Figure 14 shows examples of these heightmaps.

Since these heightmaps use Cartesian coordinates instead of polar coordinates, a wider variety of deep learning techniques from the field of 2D image processing can be applied. We found that these heightmaps did not give better results when used with a conventional CNN. This could be due to the fact that, with Cartesian coordinates, rotation of the heightmap no longer corresponds to a simple cyclic permutation of the columns of the heightmap. We therefore experimented with two newer neural network architectures which are more suited to address the rotation issue: harmonic networks [9] and 2D spatial transformer networks [15].

3.2.2.1 Using harmonic networks

The first of these architectures is the harmonic network proposed by Worrall *et al* [9]. Harmonic networks are an analog of convolutional neural networks which constrain kernels to belong to a family of circular harmonic functions $R(r)e^{i(m\phi+\beta)}$, where r and ϕ are the spatial polar coordinates of feature maps, $m \in \mathbb{Z}$ is a user-specified parameter called the *rotation order* of the kernel, and $R(r)$ and β are learned parameters. When the input 2D image is rotated by an angle θ about its center, the convolution of the input with such a kernel maintains the same magnitude, while being rotated in the complex plane by $m\theta$. The m values of harmonic convolution layers are therefore chosen so that there is no net rotation of the input at the final harmonic convolution layer. This gives a rotation-equivariant convolutional neural network. Worrall *et al* also proposed equivalents for pooling, batch normalization and ReLU activation. We use the network architecture shown in Figure 15, which was used in [9] for classifying rotated MNIST handwritten digits. The differences in our case are (1) the network has a single output, and (2) it is trained with mean squared error loss.

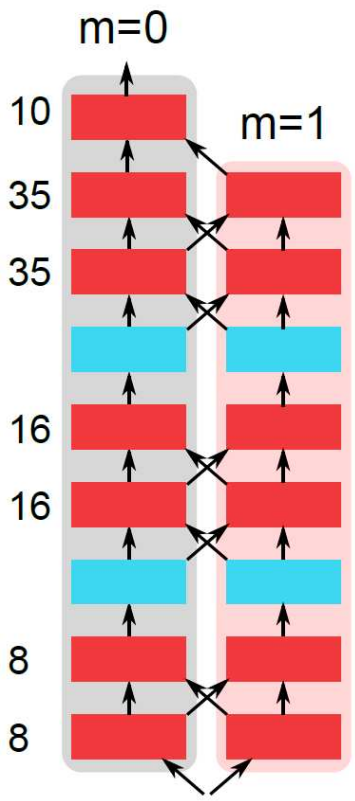

Figure 15: Harmonic network architecture used in Section 3.2.2 (reproduced from [9]). Red boxes are harmonic convolutions and blue boxes are average pooling. The two rotation order "streams" are combined by summing.

3.2.2.2 Using spatial transformer networks

The second architecture we experimented with is the addition of a spatial transformer network (STN) [15] to a regular CNN architecture. During training, the STN learns to transform the input so as to allow the subsequent layers to pay attention to the most important portion of the input. The STN contains a *localization network* which generates a set of 6 parameters defining a 2D affine transformation. This affine transformation is used to transform a sampling grid, which resamples the input image. The complete network architecture is shown in Figure 16. In Figure 17, we show examples of heightmaps before and after transformation by our trained STN. Note that, as in Section 3.2.1, every convolution layer is followed by batch normalization. We used strided convolutions instead of max pooling layers to improve performance, as recommended by Springenberg *et al* [24]. We also use the *swish* activation function [23] instead of ReLU. The structure of the localization network in the STN is the same as that of the outer network, but with 8 kernels per convolutional layer and 6 final outputs.

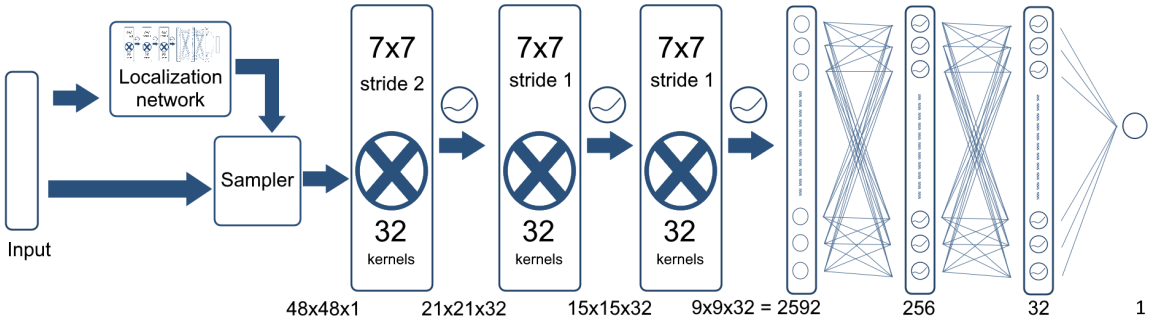

Figure 16: Network architecture with the STN, used in Section 3.2.2. The localization network has the same structure as the outer network.

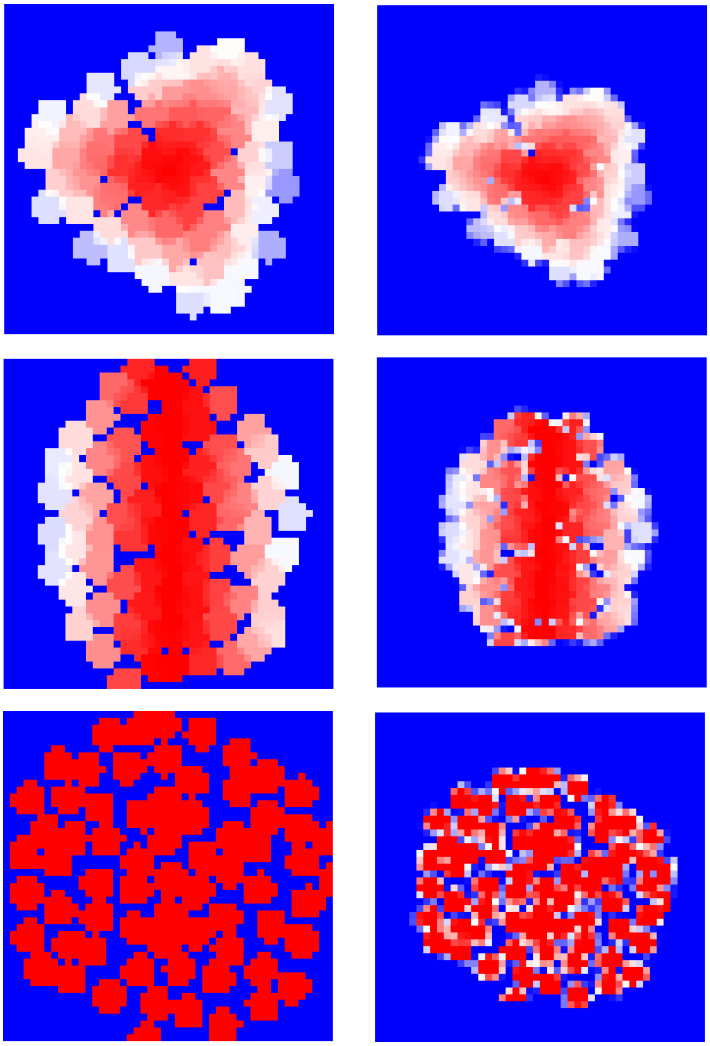

Figure 17: Visualization of Cartesian heightmaps before and after transformation by the spatial transformer network in Figure 16. The localization network learns to transform the input image to allow the subsequent layers to focus on the most salient features.

3.2.3 PointNet-based approaches

Rather than using heightmaps, we can feed machine learning models an even more basic form of a point cloud neighborhood — the raw 3D coordinates of the points in the neighborhood. This is an important alternative to devising Euclidean representations of 3D point clouds for usage with neural networks, which was shown to be feasible using the recently published PointNet [1] neural network architecture. An explanation of PointNet is given in Section 2.1.2.1 (page 10).

Since PointNet is meant to operate globally on point clouds with a fixed number of points, some additional work is necessary to adapt it to operate on local point cloud neighborhoods. This motivated the developement of PCPNet by Guerrero *et al* [2] (explained in Section 2.1.2.2 on page 12). We adapt the basic training procedure of Guerrero *et al* to use PointNet for sharpness field computation, while also adding a significant amount of data augmentation.

In our training procedure, we use the PointNet architecture for global classification, with one output scalar. Although this architecture was originally meant for binary classification, it can serve just as well for estimating a real-valued quantity if we train it to minimize mean-squared error instead of binary cross-entropy loss. The input we give the network is not the entire point cloud, but simply 128 neighboring points within a certain radius of a single input point. This allows us to compute the sharpness field at the local level, one input point at a time. The neighborhood presented to the network is first translated and scaled so that the original input point is at the origin, and the neighborhood radius is 1. If there are fewer than 128 points within the neighborhood radius, the remaining slots are filled with zeroes, while if there more than 128 then a random subset is chosen.

The training data for the PointNet and PCPNet experiments were the 8 shapes in Figure 7 (page 31) and Figure 12 (page 39). We also used the following data augmentation steps during training:

- The neighborhood radius is randomly chosen between 2δ and 9δ.
- The points are randomly perturbed by values in the range $[-0.3\delta, 0.3\delta]$ along their respective normals.
- The points are randomly rotated about the center of the neighborhood.
- The neighborhood is uniformly scaled again by $s = 2^{\gamma}$, where γ is a random number in $[-1, 1]$.
- The entire neighborhood is subject to a single random translation sampled from the cube $[-0.1s\delta, 0.1s\delta]^3$.

We also attempted to use the multi-resolution PCPNet for sharpness field computation. As input, it takes multiple sets of nearest neighbors, chosen from different neighborhood sizes. We did not use the single-resolution PCPNet because its sole innovation over PointNet, the quaternion spatial transformer network, is not relevant to directionless quantities such as the sharpness field. Note that it is not advisable to use the accompanying code of Guerrero *et al* [2] as-is, because the neighborhood radii are expressed as fractions of the diagonal of the axis-aligned bounding box. This leads to poor results for point clouds with bounding box shapes not seen during training. We therefore modified the code to use radii in terms of our δ value defined in Section 3.1.1. In our experiments, we used a multiresolution PCPNet with three patch radii, where the smallest and largest radii correspond to the smoothing kernel radii we typically use for computing the proxy surface and the feature-aware smoothing method respectively: 3δ, 5δ and 8δ.

The corresponding results and discussion can be found in Section 4.2.3 (page 78).

3.3 Applying the Sharpness Field

In this section, we describe how the sharpness field can be applied to *feature-aware smoothing* of noisy point clouds. This allows one to have a complete procedure for removing noise from point clouds, as shown in Figure 18. Feature-aware smoothing refers to noise removal that uses explicit knowledge of sharp feature locations to avoid losing these sharp features during the noise removal process. This is in contrast to implicit feature-preserving noise removal algorithms such as RIMLS [16]. Such methods seek to preserve sharp features with no explicit knowledge of these sharp features.

Sections 3.3.1 and 3.3.2 describe several methods that we have devised for feature-aware smoothing using a computed sharpness field. The most attention is given to the method presented in Section 3.3.2, where we propose a smoothing method that uses an anisotropic Gaussian kernel to preserve sharp edges. This is our most successful smoothing algorithm. It allows removing large amounts of noise with a large smoothing radius parameter, while still preserving sharp edges. It also produces points lying on sharp edges more accurately than implicit smoothing methods.

Some of the smoothing methods given in Section 3.3.1 rely on first obtaining explicit discrete representations of the shape using the sharpness field. Although these were not incorporated into our most successful smoothing algorithm, we have described them here because they are still of interest. Potential future applications include segmentation of surfaces in computer vision, or reverse-engineering of CAD models from point clouds of scanned objects. Section 3.3.3 describes an approach to segmenting the point cloud into smooth surface patches using the sharpness field. Section 3.3.4 describes methods that use the sharpness field to generate an explicit graphical representation of the sharp edges of a shape.

Most of our methods presented here have been published [7, 10].

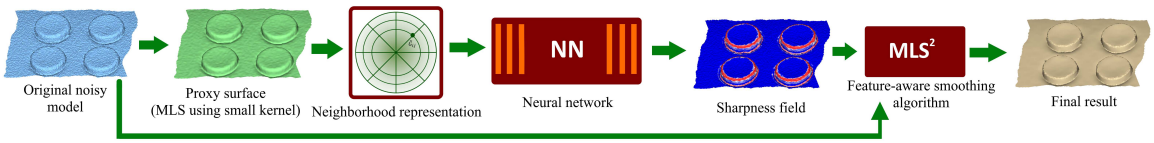

Figure 18: Procedure for noise removal from a point cloud using a sharpness field.

3.3.1 Feature-aware smoothing using barriers and patches

The methods described in Sections 3.1 and 3.2 give us a sharpness field defined over the proxy surface of an input point cloud. We describe here a method to accomplish feature-aware smoothing of the point cloud using the computed sharpness field, in order to remove noise while preserving sharp edges.

The loss of sharp features in regular MLS smoothing is due to performing weighted averages of all neighboring points and their normals within a certain search radius. This average includes neighbors which should not contribute to the average. For example, points near a sharp edge contribute to the weighted averages computed on both sides of the edge, which causes these points to be drawn inwards, roughly towards the centroid of the neighborhood. This leads to a smooth, curved edge and a slight reduction in the volume of the shape. Therefore, in order to maintain the sharpness of the edge, we should exclude neighbors that lie on the opposite sides of a sharp edge when computing these weighted averages.

Excluding neighbors which are on the opposite side of nearby sharp features will inevitably result in the projected point moving away from the sharp feature, which means that no points will be projected onto a sharp edge or crease. This is natural, since the underlying surface is not differentiable on its sharp edges and creases. Thus a polynomial approximation such as MLS will never capture these points where the surface is not differentiable. Instead, we must develop a post-processing method which will either add these points or move nearby points onto the edges.

We describe here several approaches to excluding unwanted neighboring points from the weighted averages, by making use of an already computed sharpness field.

3.3.1.1 Smoothing using local barrier planes

In this method, when projecting a point to the smoothed surface, we make a list of all of its *edge neighbors*, which are the neighbors whose sharpness field value is above a certain threshold. If a nearby edge exists within the smoothing radius, we then try to find a plane such that one of its half-spaces contains the neighbors which we wish to include in the weighted average. One possible approach to this is to take the plane which is parallel to both the average normal of the edge neighbors and the line which best fits the edge neighbors, and which contains the centroid of the edge neighbors. The best-fitting line for the edge neighbors can be found by computing the covariance matrix of the positions of the edge neighbors, centered at their centroid. The best-fitting line is then the line

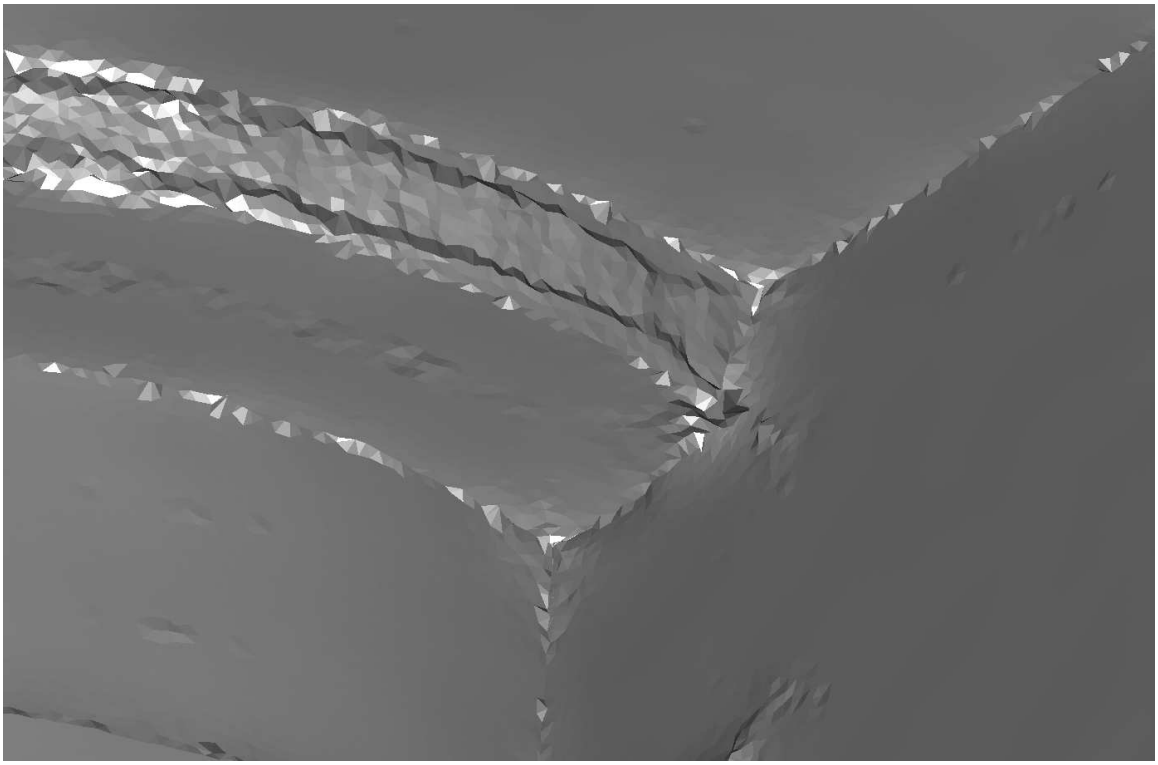

Figure 19: Smoothing result using local barrier planes (Section 3.3.1.1). Smoothing is performed with a smoothing radius of 8δ on the *fandisk* model with 3δ normal noise added. Note the artifacts in areas where there is more than one edge in the smoothing radius.

passing through the centroid and parallel to the eigenvector of the covariance matrix corresponding to the largest eigenvalue. As can be seen from Figure 19 (page 50), this method works quite well to smooth the surface around sharp edges. However, the presence of more than one sharp edge in the same neighborhood still presents some difficulties. This occurs near corners, as well as between parallel sharp edges which are close together. In such cases, one cannot fit just one line to the edge neighbors.

3.3.1.2 Smoothing using an edge graph with barrier planes

As mentioned earlier, even if neighboring points which are across an edge from the point to be projected are properly excluded from the weighted average, this will not help us to have points which actually lie on the edges. Instead of relying entirely on post-processing to solve this problem, we could simply tackle it directly by generating an explicit graphical representation of the sharp features from the sharpness field. The procedure for generating this *edge graph* is described in Section 3.3.4. The points on the edges of the edge graph should also possess normals. After applying

feature-aware smoothing, points near edges could simply be moved to the nearest edge of the edge graph. Alternatively, new points can be directly synthesized on edges of the edge graph.

The edge graph can be used to perform smoothing in a similar manner to the local barrier planes mentioned in Section 3.3.1.1. In that method, we have one barrier plane per input point for excluding unwanted neighbors from the weighted average. This has some limitations such as not being able to handle regions with multiple edges or corners. Having an edge graph provides us an easy way to overcome these limitations. The edge graph can have barrier planes associated with each vertex, or with each edge (we investigated both approaches). When projecting a given point to the smoothed surface, one can find the barrier planes of neighboring vertices or segments of the edge graph, and use these to check if neighboring points should be included in the weighted average.

Since we can have multiple segments of the edge graph inside the neighborhood of the input point, there can be multiple barrier planes available for each point to be projected. Therefore we have to choose how to handle weighted averages when there are multiple barrier planes in the neighborhood. A simple approach is to treat all barrier planes equally. We can either include a point if it is on the same side as the current point for all nearby barrier planes (similar to logical AND), or for at least one of them (similar to logical OR). Either option will result in some points being wrongly excluded or included if the current point is near a zigzagging portion of the edge graph, but we found in practice that the logical AND gave better results. At this point, the results are already better than those for the previous local barrier plane method in Section 3.3.1.1. However, there is still scope for improvement near corners.

The method described above performs satisfactorily in areas where the smoothing radius contains segments of two different edges of the edge graph, or in areas where the current point is surrounded on multiple sides by segments belonging to the same graph edge. However, at corners in the surface there are usually 3 edges of the edge graph which converge at one point. The edge at the far side of the corner will contain barrier planes that should not be considered, as they could cause legitimate neighbor points to be excluded. Therefore we need a way to distinguish between barrier planes belonging to different edges. This can be done by numbering each edge using a simple algorithm that traverses each segment of the graph between graph vertices. Now, when looking for the nearby points on the edge graph, we can choose to include only barrier planes belonging to the 2 nearest graph edges. This method improves results, but still has trouble excluding irrelevant edges when more than 3 edges are in the smoothing radius.

If the normals of the barrier planes are consistently oriented along each graph edge, one can also

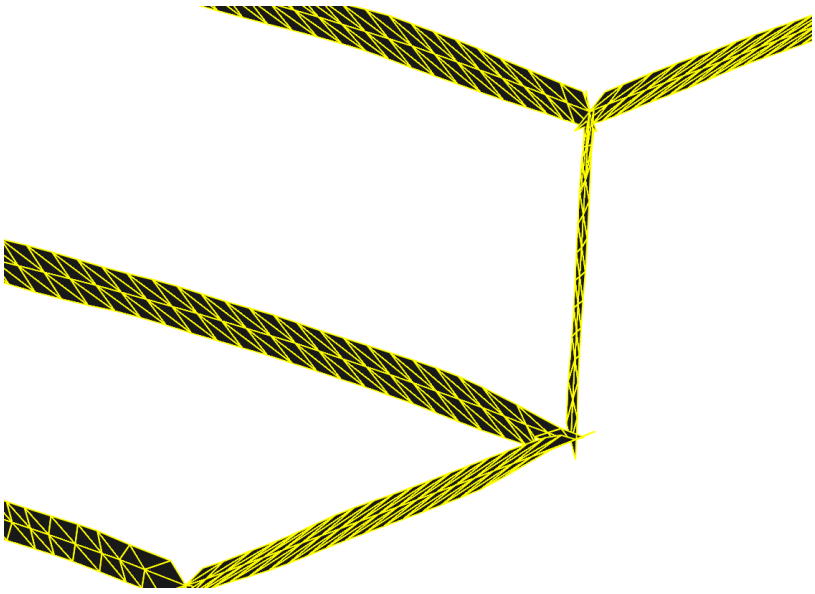

Figure 20: Example of a barrier mesh.

improve the classification of neighbor points for a given edge by changing the criterion for inclusion of a neighboring point to the following: for each nearby graph edge, the signed distance from the current point to the barrier plane of the nearest point on the edge should have the same sign as the signed distance of the neighboring point to the barrier plane of the nearest point to the neighbor point on the edge graph.

We can also generate a triangular mesh called the *barrier mesh* by extruding each segment of the edge graph in the direction of the normal and in the opposite direction of the normal. The projection operator will then find all triangles of the barrier mesh within the smoothing radius and only include neighbors which are on the same side of every neighboring triangle as the point being smoothed. This approach has the advantage of giving us more flexibility to prevent barrier planes from exerting an influence too far from the edge graph. Figure 20 shows an example of a barrier mesh.

Figure 21 (page 53) shows the result obtained using barrier planes obtained from an edge graph. It is a clear improvement over the results obtained in Section 3.3.1.1, especially between parallel sharp edges and convex corners. However, there are still some artifacts near concave corners.

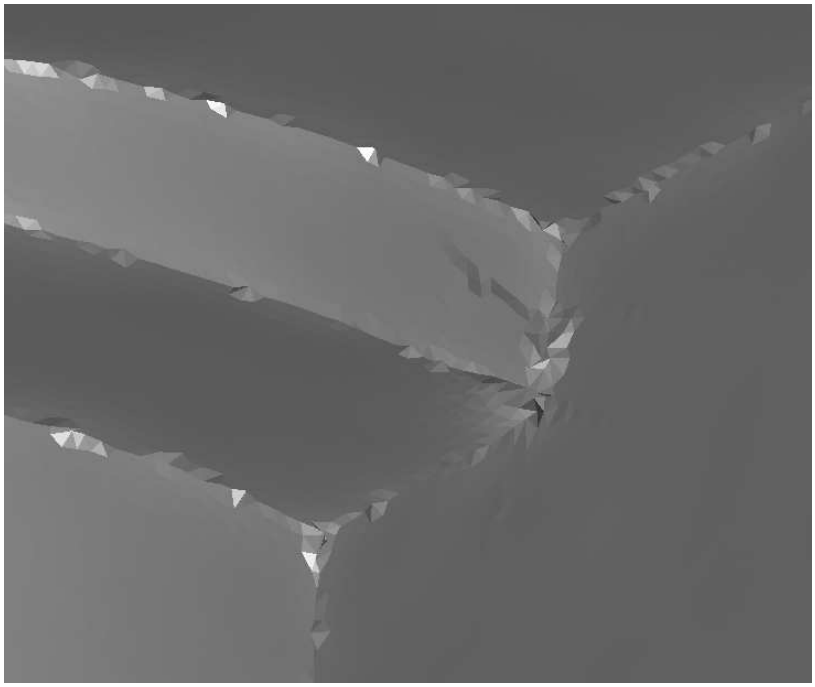

Figure 21: Smoothing result using barrier planes obtained from an edge graph (Section 3.3.1.2).

3.3.1.3 Smoothing using segmented patches

Segmenting the point cloud into patches, as described in Section 3.3.3, provides an easy method to exclude unwanted neighbors from our weighted average. In most cases, we can simply exclude neighbor points which do not belong to the same patch as the current point. This means that we usually no longer need an explicit representation of edges when trying to classify a neighbor point after patch segmentation. However, it is possible to have edges which are partially inside a patch, but do not divide it into two separate patches. The points on either side of such edges will belong to the same patch. These correspond to edges of the edge graph which do not belong to any loop of the graph. Neighbor points which are on the opposite side of such graph edges must therefore be excluded from the weighted average.

Figure 22, shows the smoothing result obtained on the *blade* model using the segmentation and edge graph shown in Figure 54 (page 95). The result is an improvement over the results of Section 3.3.1.2, because it does not show significant artifacts near concave corners (as seen in Figure 21 on page 53).

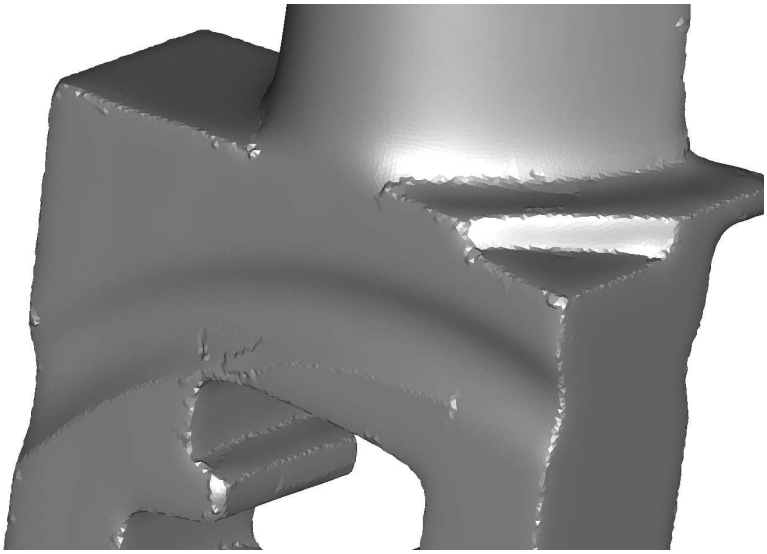

Figure 22: Smoothing result using patches obtained by flood-fill segmentation (Section 3.3.1.3).

3.3.2 Feature-aware smoothing using an anisotropic Gaussian kernel

This section describes an approach to feature-aware smoothing which is different from all the methods described in Section 3.3.1 because it eliminates the need to construct discrete representations of sharp edges such edge graphs or barrier planes. Instead, it makes better use of the smoothness of the sharpness field by constructing anisotropic Gaussian kernels that are shaped by the sharpness field. We also describe an application uniquely made possible by the sharpness field: a method to place points accurately on sharp edges after smoothing. The methods in this section have been published [7, 10].

3.3.2.1 Method overview

For the weighted average, instead of the usual isotropic 3D Gaussian kernel based on Euclidean distance, we define an anisotropic 2D smoothing kernel whose cross-sections are 1D Gaussian functions of the geodesic distance. The standard deviation of each of these 1D Gaussians is constrained so that the kernel reduces to a negligible value at a neighboring sharp edge, if any. In practice, we discretize the kernel by constructing the same radial grid described in Section 3.1.4. The algorithm for constructing the grid is identical to that described in 3.1.4.1, except that the previously computed sharpness values are also interpolated for each point on the grid.

Figure 23 shows an example of such a radial grid for the neighborhood of a point on the *fandisk* model. We will use this neighborhood to explain the creation of the anisotropic Gaussian kernel.

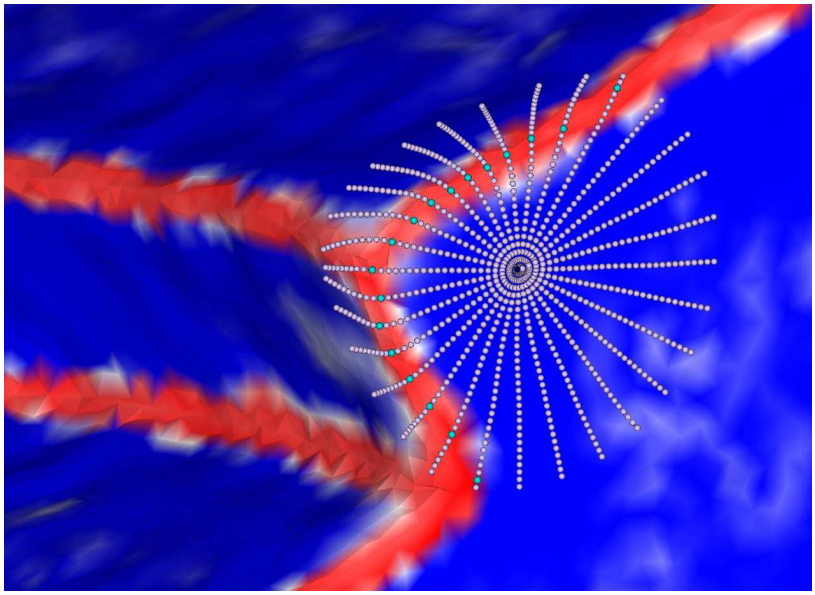

Figure 23: Example of a radial grid near multiple sharp edges on the *fandisk* model. The cyan-colored points are local minima of the sharpness field along each spoke of the grid, i.e. *barrier points*.

Note that the point shown in this example is near multiple sharp edges.

Each spoke of the grid corresponds to a 1D Gaussian kernel. By traveling outward along each spoke, we can determine where it crosses a local maximum of the sharpness field, if any. Figure 24 shows graphs of the sharpness field values along each spoke of the radial grid. The point before the maximum (or the last point, if there is no edge on the spoke) becomes the *barrier point* of the spoke. We can see the barrier points of each spoke in Figure 23.

The standard deviation of the Gaussian kernel of each spoke is adjusted to be one-third of the geodesic distance to its barrier point. At this point, we have effectively determined the shape of the anisotropic smoothing kernel, subject to errors due to discretization. Figure 25 shows an example of the shape of the anisotropic Gaussian for the same neighborhood used in Figure 23.

For our smoothing computation, we need to use the anisotropic kernel to compute a weight for each neighboring point in the original point cloud, based on its geodesic distance to the current input point. To make computation more efficient, we first compute weights for each point on the radial grid, since we already know both their geodesic distances as well as the shape of the anisotropic kernel along each spoke (shown in the left half of Figure 25). Weights of points beyond the barriers are set to 0. We then triangulate the radial grid and project each neighbor point to the nearest triangle of the radial grid. Barycentric interpolation can then be used to interpolate the weight of

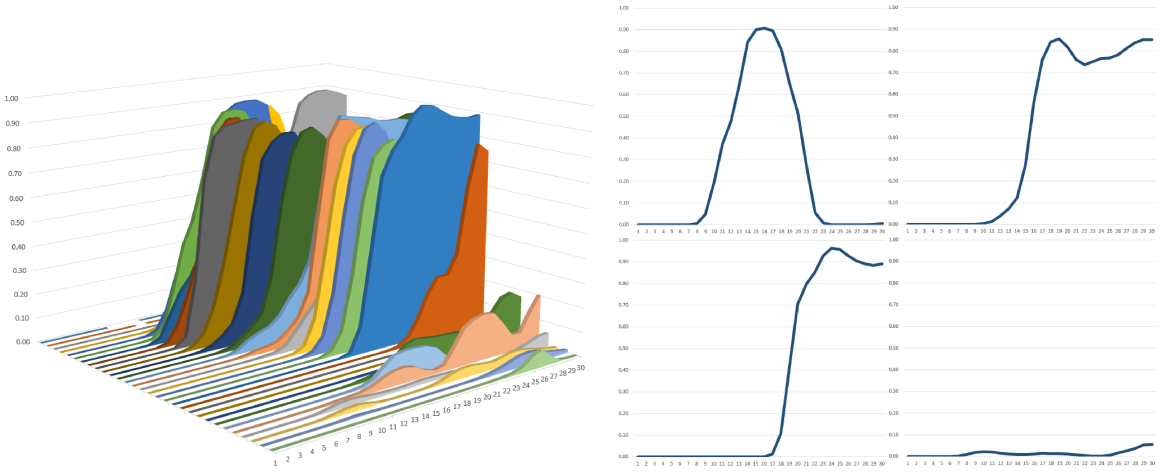

Figure 24: Graphs showing the sharpness field values along different spokes of the radial grid shown in Figure 23. Left: a collection of graphs for all spokes. Right: graphs for four selected spokes, showing different possible radial profiles of the sharpness field.

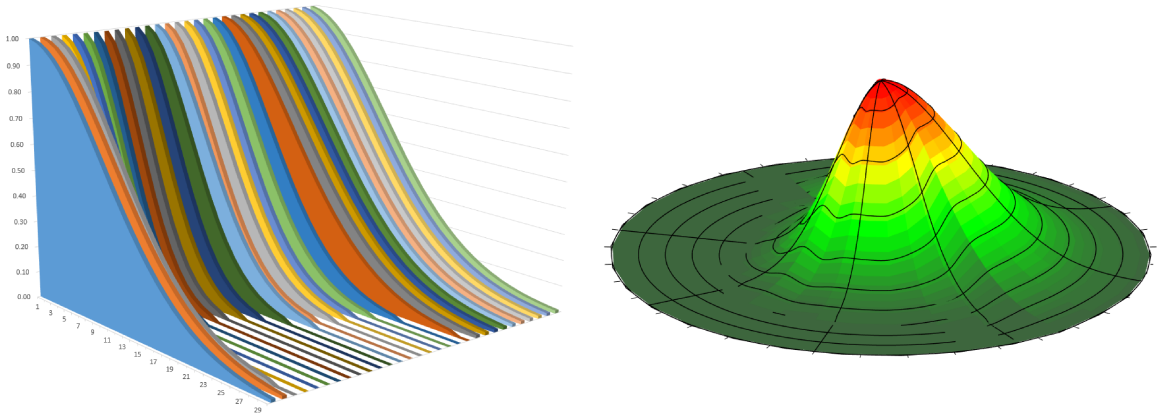

Figure 25: Anisotropic Gaussian kernel computed for the radial grid shown in Figure 23. Left: collection of graphs of weights along each spoke. Right: 3D plot showing the overall kernel shape.

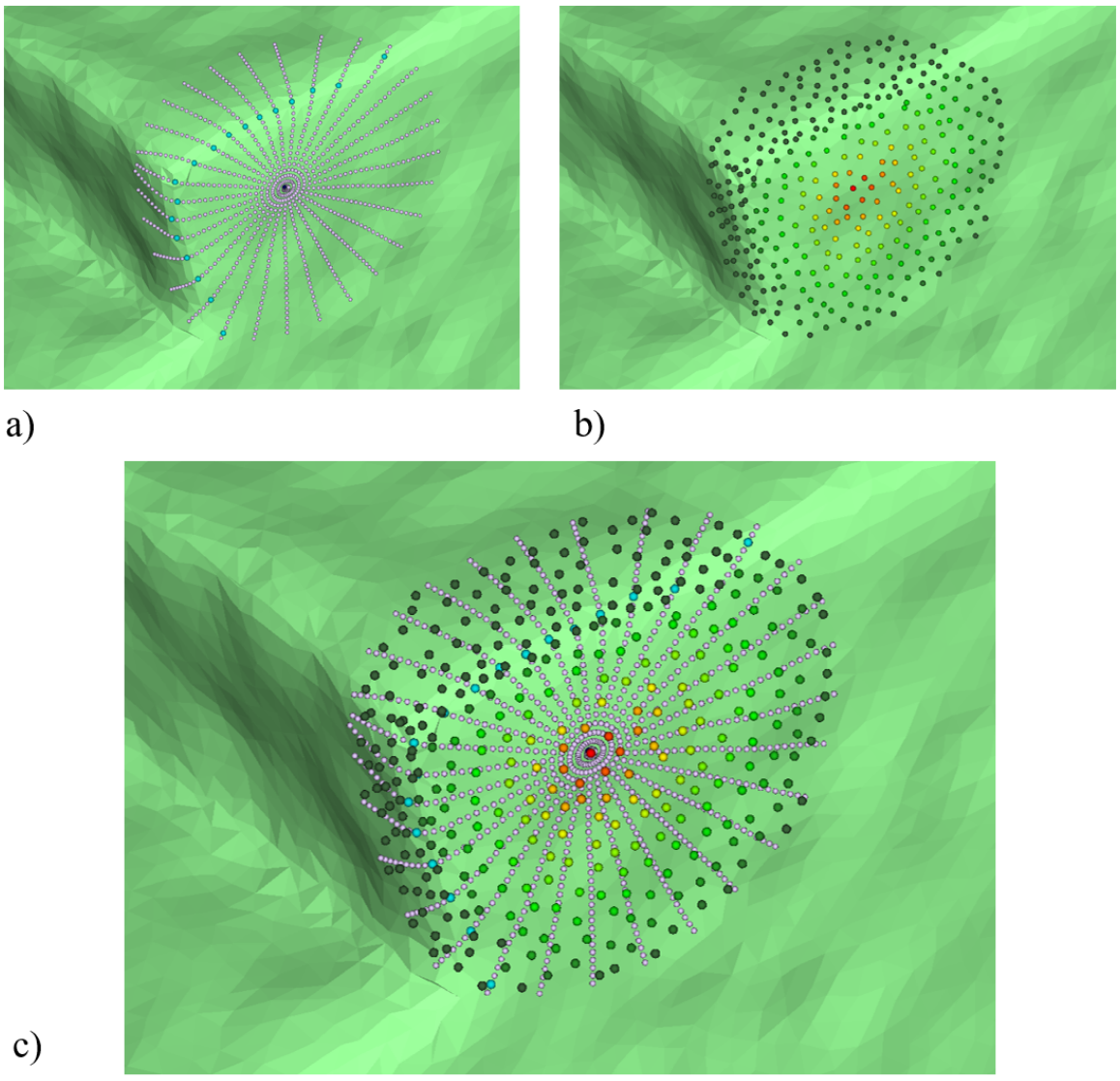

Figure 26: Illustration of smoothing weights: a) The input point and the radial grid, showing the barrier points rendered in cyan. b) The local neighborhood of the input point, with weights color-coded on a logarithmic scale: red corresponds to a weight of 1 and dark green corresponds to a weight of 0. Note that the points across edges have 0 weight. c) Superimposition of the points in a) and b).

each neighbor point. Figures 26 (b) and (c) show the neighbor weights of a given input point, with and without the radial grid superimposed (this is the same neighborhood shown in Figure 23, but the sharpness field is not shown to improve visibility).

The weights now obtained can be used to perform a weighted average of neighboring points and normals, which yields an accurate normal for points near edges. However, it does not automatically yield points which lie on the edges. To accomplish this, we can make use of the fact that the accurate normals give us the correct local tangent plane of points near edges. We clone points near edges, and iteratively move the clones in their local tangent planes by small steps, ascending the gradient of the sharpness field. Note that the sharpness field is defined on the proxy surface, and therefore at each step, every candidate destination point is projected onto the proxy surface to determine its sharpness value. We stop moving each point in the tangent plane when its projection on the proxy surface has reached a local maximum of the sharpness field.

The following section gives further details on the weight computation portion of the smoothing algorithm.

3.3.2.2 Smoothing weight computation algorithm

Our feature-aware smoothing algorithm performs a weighted average of the neighbors of a given point in the input point cloud, to obtain its new position and normal. This requires computing weights for all neighboring points within a specified radius R. In order to avoid the difficulty of tracing geodesic lines along the proxy surface $\bar{S}_r$ from the input point to every neighboring point, we reuse the radial grid computation algorithm described in Section 3.1.4.1. The only difference is that in this version, the MLS projection operator $\overline{M}_r$ also provides an interpolated sharpness value in addition to the point and normal on the proxy surface. This gives us a grid of $n_i \times n_j$ points $\bar{G}_{ij}$ and sharpness values $\bar{s}_{ij}$ sampled on the proxy surface. The radial grid structure means we can easily compute the approximate geodesic distance from the center to each grid point. This allows us to easily compute weights for every grid point, as shown in Algorithm 3. We typically set $s_{min} = 0.6$.

Once we have the weights w_{ij} for the grid points, we then triangulate the grid. For example, $\bar{p}$, $\bar{G}_{11}$ and $\bar{G}_{12}$ form one triangle, while $\bar{G}_{11}$, $\bar{G}_{21}$ and $\bar{G}_{22}$ form another. This gives us a set of $n_j[2(n_i - 1) + 1]$ triangles. For each neighbor point, we compute the barycentric coordinates $(\lambda_a, \lambda_b, \lambda_c)$ of its projection onto the plane of each triangle, along with the perpendicular distance. We then find the closest triangle, formed by $\bar{G}_{a_i a_j}$, $\bar{G}_{b_i b_j}$ and $\bar{G}_{c_i c_j}$, which contains the projection and is no further than 2δ from the neighbor point. This allows us to interpolate the weight of the

Algorithm 3 Weight computation on the radial grid

procedure ComputeWeights($\bar{G}_{ij}, \bar{s}_{ij}, \tilde{p}, n_i, n_j, s_{min}$) ▷ Input: grid points, sharpness values of grid points, center point, number of radial divisions, number of angular divisions, min sharpness value for local maximum

$\bar{p}, s_0 \leftarrow \overline{M}_r(\tilde{p})$

for $j \leftarrow 1$ **to** n_j **do**

$\bar{s}_{max} \leftarrow \bar{s}_0$ ▷ sharpness value at barrier

$i_{max} \leftarrow 0$ ▷ index of barrier point

for $i \leftarrow 1$ **to** n_i **do** ▷ first pass, to find the local maximum

if $\bar{s}_{ij} > max(s_{min}, \bar{s}_{max})$ **then**

$\bar{s}_{max} \leftarrow \bar{s}_{ij}$

$i_{max} \leftarrow i$

end if

end for

$\delta \leftarrow \frac{1}{3} d_{i_{max}}$

$d_0 \leftarrow 0$

$\bar{G}_{0j} \leftarrow \bar{p}$

for $i \leftarrow 1$ **to** n_i **do** ▷ second pass, to compute the weights

$d_i \leftarrow d_{i-1} + \left\| \bar{G}_{ij} - \bar{G}_{i-1,j} \right\|$ ▷ approximate geodesic distance from center

if $1 \leqslant i_{max} \leqslant i$ **then**

$w_{ij} \leftarrow 0$ ▷ at or beyond the barrier point

else

$w_{ij} \leftarrow exp(-\frac{d_i^2}{2\delta^2})$

end if

end for

end for

$w_{00} \leftarrow 1$

return all w_{ij} ▷ smoothing weights for grid points

end procedure

neighbor point as $\lambda_a w_{a_i a_j} + \lambda_b w_{b_i b_j} + \lambda_c w_{c_i c_j}$. However, if any of the three weights is 0, we set the weight of the neighbor to be 0 as well, since it is beyond the barrier.

Once the weights of each neighbor have been computed, they are normalized and used to perform weighted averages of the neighbor points and their normals. The input point is finally displaced by the projection of the average point onto the average normal.

3.3.3 Segmentation into patches

We can use the sharpness field to segment the surface into patches using a variant of the well-known flood-fill algorithm [108]. A kd-tree is constructed from the point cloud, to allow efficient nearest neighbor queries. We pick a random seed point and add its unvisited nearest neighbors within a certain radius to a queue. These points will all be assigned the same patch ID. Points in the queue are processed one at a time in the same manner, which effectively performs a breadth-first search that expands radially. However, the neighbors are not added to the queue if the sharpness field takes a value above a certain threshold at some of the neighbors. These points will have to be handled later. When the queue is empty, a new patch ID and a new seed point are chosen. This procedure is repeated until all points have either been assigned a patch, or they are too close to edges to be handled immediately. The complete algorithm is shown in Algorithm 4. Note that the point cloud P is treated as a set of pairs $(\mathbf{p}_i, s_i)$ of point positions and their computed sharpness values.

There can be several methods to assign patch IDs to points near edges. The simplest method is to perform nearest-neighbor interpolation of the patch IDs. Figure 53 on page 95 shows an example of this technique applied to the *fandisk* model. An alternative approach is to use the edge graph described in Section 3.3.4 to create a barrier mesh, which gives a precise boundary between patches. Rays can then be cast from each edge point to its nearest neighbors; the patch ID of the neighbor will only be copied if the ray does not intersect the barrier mesh. Figure 54 on page 95 shows an example of this technique applied to the *blade* model, along with the corresponding barrier mesh.

In practice, the method we found to work best was to reuse the neighbor weights calculated in Section 3.3.1 for feature-aware smoothing. A weight of zero for a neighbor point implies that it does not belong to the same side of all nearby edges. Therefore, a point and one of its neighbors can be reliably determined to belong to the same patch if their weights with respect to each other are both non-zero.

Algorithm 4 Algorithm for flood-filling a patch number

```
procedure floodfill(i, k, r, t, P)        ▷ Input: index of seed point, patch number, search radius, threshold, point cloud with sharpness values
    Q ← {i}                                                          ▷ Queue of points to visit
    Mark i as visited
    while Q is not empty do
        a ← dequeue Q_1 from Q                                       ▷ Set current point index
        set patch of a to k
        N = {(p_j, s_j) in P where p_j ∈ B(p_a, r)}                  ▷ Neighbors of p_a
        for (p_j, s_j) in N do                ▷ First pass, to make sure there are no points near edges
            if s_j > t then
                continue the while loop
            end if
        end for
        for (p_j, s_j) in N do                ▷ Second pass, to add unvisited neighbors to the queue
            if j is not visited then
                Mark j as visited
                Enqueue j in Q
            end if
        end for
    end while
end procedure
```

3.3.4 Edge graph extraction

In Section 3.1.1, we mentioned that we are interested in shapes whose sharp edges form a network of curves which intersect at corners. We have developed an algorithm for generating an explicit graphical representation of this curve network, which we call the *edge graph*. We can use a computed sharpness field to generate an edge graph by connecting the locations of ridges of the sharpness field. Figure 55 on page 96 shows an example of an edge graph generated from the sharpness field computed for the *fandisk* model (shown in Figure 27 on page 67).

The algorithm traverses the sharpness field in a manner similar to a depth-first search. The basic idea is to select a starting point and move to an unvisited neighboring point which is selected according to some heuristics. The newly selected point is connected to the previously selected point in the graph. This process is repeated again at the newly selected point until the selected point has no suitable unvisited neighbors. Then a new starting point is selected according to certain criteria, and the traversal resumes. The algorithm terminates when no suitable starting point can be found after reaching a dead end. Algorithm 5 shows the basic structure of the algorithm. We elaborate further on some parts of the algorithm below.

Algorithm 5 Algorithm for generating an edge graph

```
1: procedure traverse(P, r_n, r_v) ▷ Input: point cloud, neighborhood radius, radius for marking visited points
2:     G ← {}                                                                  ▷ Edge graph
3:     while true do      ▷ loops as long as suitable starting points exist for growing graph edges
4:         p ← starting point chosen from P                                    ▷ (see Sec. 3.3.4.1)
5:         if no suitable starting point then
6:             break
7:         end if
8:         p_0 ← p                                          ▷ Save the starting point for later
9:         while true do              ▷ loops as long as the current graph edge can be extended
10:            N ← set of unvisited points sampled around p within radius r_n   ▷ (see Sec. 3.3.4.2)
11:            if |N| = 0 then
12:                break
13:            end if
14:            n ← most suitable neighbor in N                                  ▷ (see Section 3.3.4.3)
15:            Add line segment l connecting (p, n) to the edge graph
16:            Mark points in P within radius r_v of line segment l as visited
17:            if n = p_0 or n is a corner point then
18:                break
19:            end if
20:        end while
21:    end while
22:    return the edge graph G
23: end procedure
```

3.3.4.1 Choice of starting point

Before beginning traversal of the sharpness field, we first find all corner points of the shape in the point cloud. To do this, we first identify all points in the point cloud which are near corners. This can be done by building a 3×3 covariance matrix for every point in the point cloud, using all neighboring points having a sharpness value above a threshold. Points near corners can then be identified as points for which the two largest eigenvalues of the covariance matrix are close in value (indicating that it is not easy to fit a line to neighboring points near edges). Points near corners are then clustered based on their positions, based on a specified maximum cluster radius. The centroid of each cluster is then taken to be the position of a corner.

When we need to choose a starting point for traversing edges, we choose a corner point with unvisited neighboring points whose sharpness value is above a threshold. Knowing the corner points in advance also has another advantange: we can stop extending the current edge when we encounter a corner, by extending the edge to connect to the corner point (line 17 of Algorithm 5). Of course, not all shapes with sharp edges have corners (e.g. cylinders). If no corners remain, we choose the unvisited point with the highest sharpness field value as the starting point.

3.3.4.2 Sampling candidate points for extending a graph edge

When sampling the neighborhood of the current graph point to obtain candidates for extending the graph edge, it would be easy to simply take unvisited points from the input point cloud within a certain radius. However, this causes the graph edge to zigzag if the point cloud is not sampled sufficiently densely. Ideally, we would like the edges of the graph to exactly follow the ridges of the sharpness field, without being restricted to points in the point cloud. This can be accomplished by sampling candidate edge points on the proxy surface. We do this by sampling points on a radial grid of geodesic radius r_n around the current edge point, using the same algorithm described in Section 3.1.4 on page 35.

We also interpolate the sharpness value and the visited state at each of the sampled points. Since the points we are sampling are not from the point cloud, it is harder to mark portions of the surface as visited. We get around this hurdle by treating the visited state as a number in $[0, 1]$ instead of a binary value. When marking regions as visited, we assign a visited value of 1.0 to points in the point cloud within a certain radius r_v of the path traveled between the current point and the previous point on the current graph edge (this is a capsule-shaped region). If the radius r_v is too large, parts of the sharpness field will never be visited, and if it is too small, the algorithm might retrace edges which it has previously covered. After interpolating the visited state for a point on the radial grid, we threshold it to return to a binary value.

3.3.4.3 Choosing the next point to add to the current edge

The next point to visit is chosen from among the unvisited points on the radial grid. The new point on the edge graph is also assigned the same normal as the chosen radial grid point. We typically choose the candidate point with the highest sharpness field value as the next point to add to the current graph edge. However, if one of the candidate points is a corner point or the starting point of the graph edge, this candidate takes priority, because it allows completion of the edge.

3.3.4.4 Post-processing the edge graph

Once the edge graph has been generated, some post-processing operations are needed to ensure that it can be used with the smoothing described in Section 3.3.1.2. Since most of the edge graph lies on the proxy surface, the edge graph points must first be backprojected so that they lie closer to the target surface. To do this, we find the offset of every point in the point cloud from its projection

on the proxy surface. We then interpolate these offsets for points of the edge graph, using Gaussian interpolation based on Euclidean distance. These offsets are then used to translate each edge point along their normals (the normals were obtained in Section 3.3.4.3). This brings the edge graph close to the target surface.

The edge graph also needs to be smoothed, since a smooth edge graph greatly improves the performance of barrier plane-based methods in Section 3.3.1.2. This is done using a smoothing algorithm similar to Algorithm 1 (page 29). The main difference is that on each iteration, we move the point in the average tangent plane instead of along the average normal. This effectively removes zigzag patterns from the edge graph, while minimizing shrinking.

In this chapter, we have seen a number of methods for computing the sharpness field for a given point cloud. We have also described applications of the sharpness field to feature-aware smoothing, edge graph generation and smooth patch segmentation. The following chapter presents the results we obtained using the methods described in this chapter, for both sharpness field computation as well as the applications of the sharpness field.

Chapter 4

Results and Discussion: Sharp Edge Reconstruction

This chapter details the results we obtained for sharpness field computation and sharpness field applications using the various methods described in Chapter 3. Section 4.1 discusses the results obtained using the sharpness field computation method described in Section 3.1. We also evaluate the performance of different machine learning models for sharpness field computation using the same input representation in Section 4.1.2. Section 4.2 describes the results obtained using the local neighborhood representations and neural network architectures described in Section 3.2, which were implemented for the purpose of improving upon the results in Section 4.1. Section 4.2.4 also includes a quantitative comparison of all methods for sharpness field computation described in Chapter 3. Section 4.3 describes the results obtained for the different sharpness field applications from Section 3.3, including feature-aware smoothing, smooth patch segmentation and edge graph extraction. The results shown here have been published [7, 10].

4.1 Sharpness Field Computation Using Normal Angles

Section 4.1.1 summarizes the results we obtained using the sharpness field computation method described in Section 3.1. We have also experimented with using the same normal angles local neighborhood representation with other machine learning models besides CNNs. Section 4.1.2 compares the performance of different machine learning models for sharpness field computation.

4.1.1 Sharpness field results

We have tested our trained neural networks on four shapes:

1. The well-known *fandisk* model, appropriately resampled. It is a synthetic CAD model of a mechanical part, which means that we can compute a ground truth sharpness field for evaluation purposes.

2. The *blade* model, which is a 3D scan of a mechanical part containing measurement noise.

3. Our 3D scan of a landline phone keypad, made using a high-resolution laser 3D scanner.

4. Our 3D scan of the facade of a wooden house model, also captured using the laser scanner.

With the basic CNN architecture from Figure 10 (page 34) and the normal angles neighborhood representation from Section 3.1.4, we are already able to obtain a usable sharpness field. In Figure 27, we can see the sharpness fields computed after adding different levels of synthetic noise in the normal direction to the *fandisk* model. Naturally, some subtle creases cannot be completely identified, depending on the level of noise. Figure 28 shows the computed sharpness fields for the two 3D scans — the *blade* model and our scanned phone keypad. We can see that most of the sharp edges are fully captured.

We have also experimented with avoiding the sorting step when generating the 900 angles, by shifting the burden of rotation invariance onto the neural network, as mentioned earlier. As can be seen from Figures 30 and 31, this produces results similar to Figures 27 and 29, although the results vary a bit between training runs.

We can see from Figure 32 that the curved edges in the middle of the *blade* model are not captured at all. This cannot be explained by noise in the point cloud, and points to a shortcoming of either the neighborhood representation or the neural network model. We shall see in Section 4.2 that other representations can enable us to capture these edges as well. Quantitative evaluation for sharpness field computation on curved edges is given in Section 4.2.4.

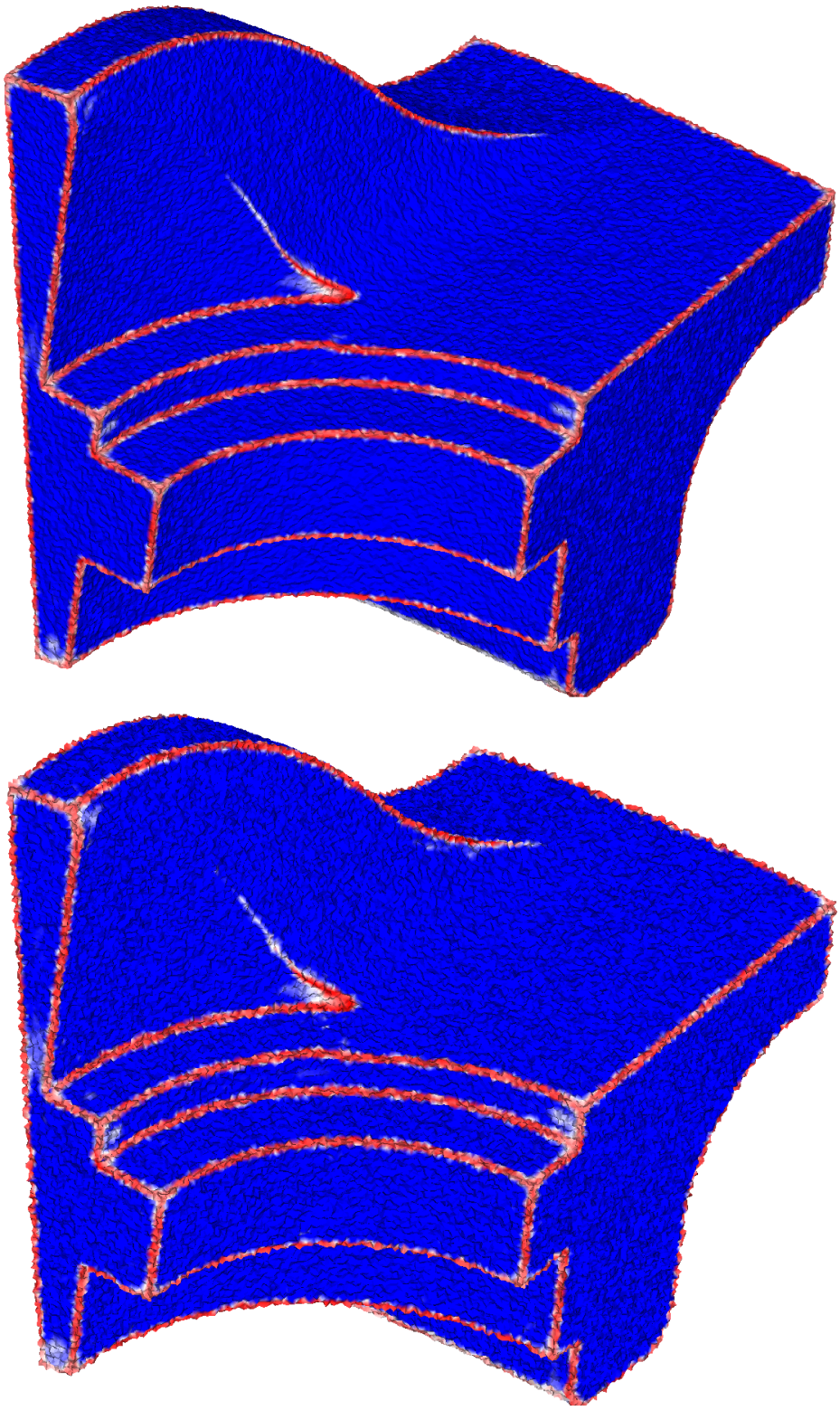

Figure 27: Sharpness field results obtained using 30×30 sorted normal angles. (T to B) The *fandisk* model with 1.5δ normal noise added, and with 3δ noise added,.

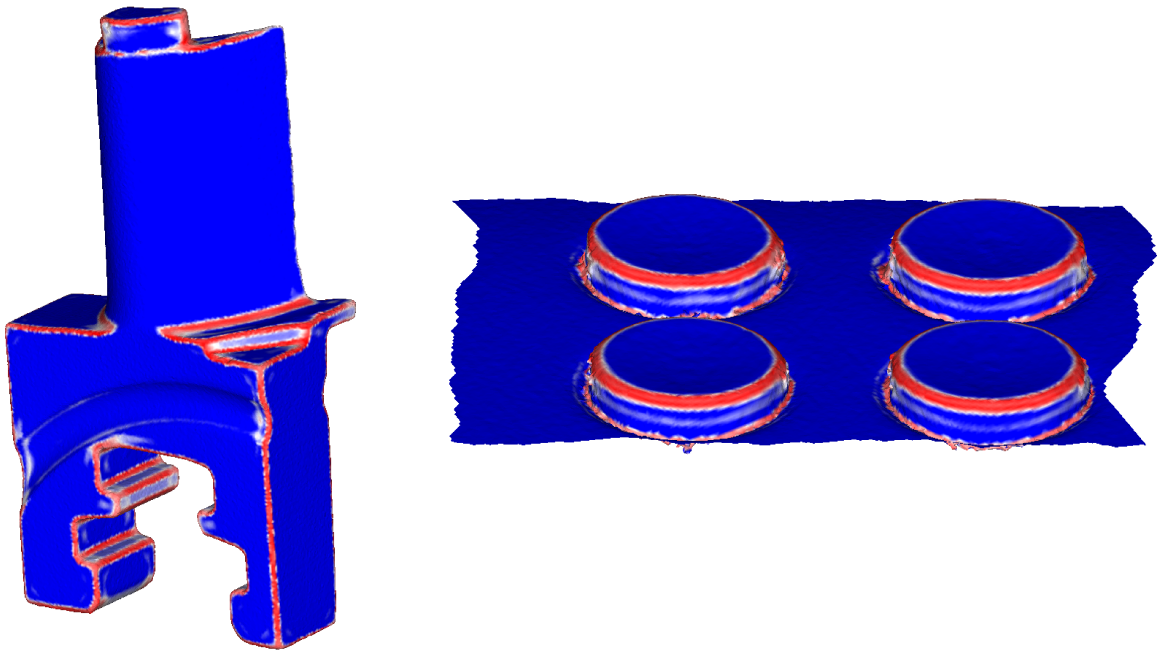

Figure 28: Sharpness field results obtained using 30×30 sorted normal angles. (L to R) The *blade* model and the phone keypad model.

Figure 29: Sharpness field results obtained using 30×30 sorted normal angles for the *house* model.

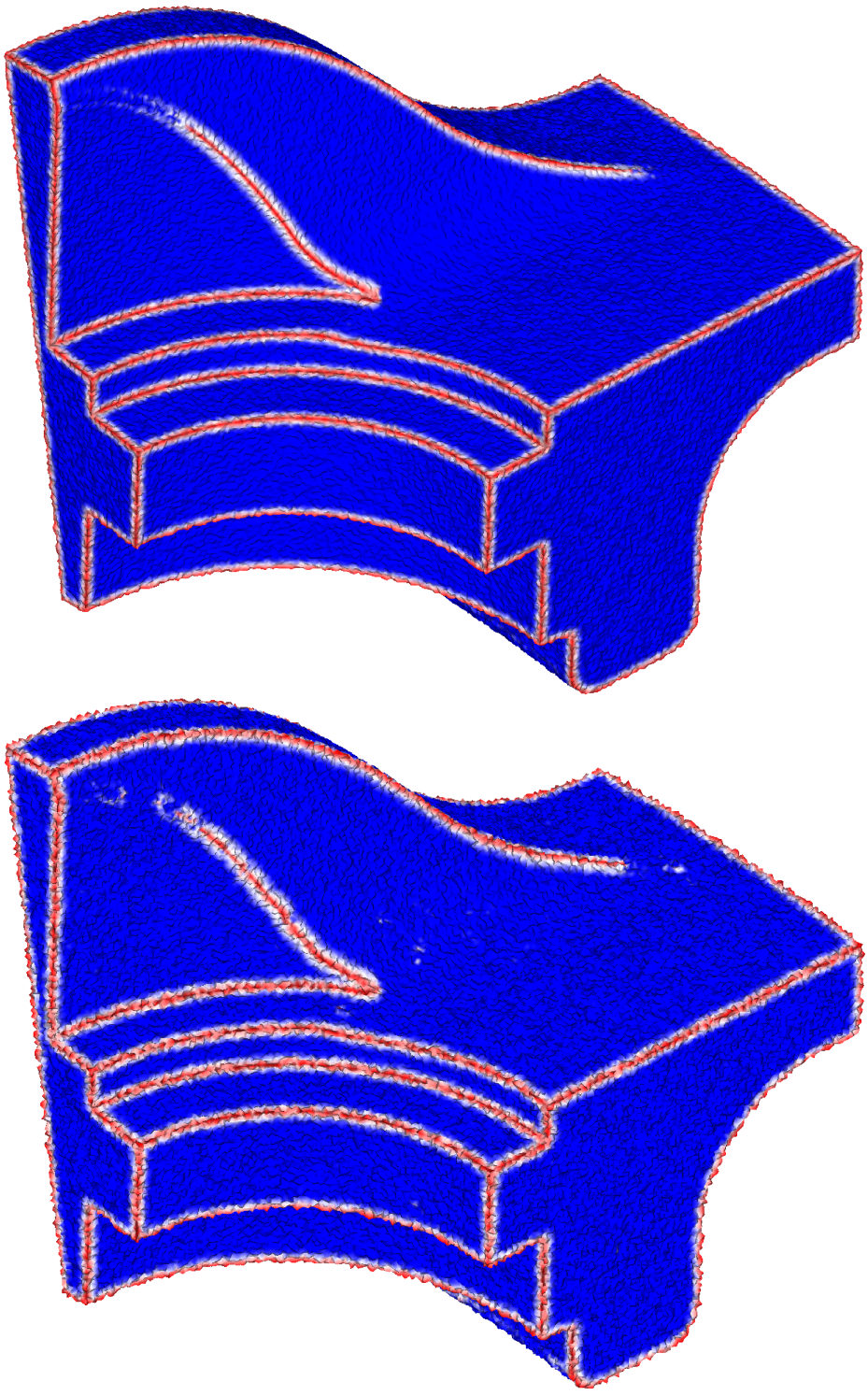

Figure 30: Sharpness field results obtained without sorting the 30×30 normal angles. (T to B) The *fandisk* model with 1.5δ normal noise added, and with 3δ noise added.

Figure 31: Sharpness field results obtained using 30×30 normal angles, without sorting, for the *house* model.

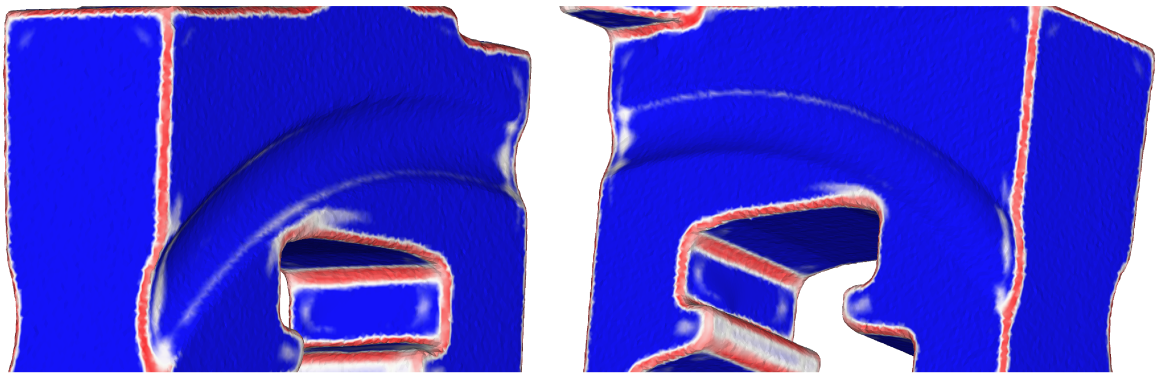

Figure 32: Sharpness field results obtained using 30×30 sorted normal angles on the curved edges of the *blade* model. We can see that the curved edges are not captured.

4.1.2 Comparison of machine learning models

We have performed a quantitative evaluation of our sharpness field computation method for different machine learning models. We use the sorted 900 angles neighborhood representation described in Section 3.1.4, since this representation is compatible with all the machine learning models. To create ground truth data for evaluation, we designed an ideal sharpness field for the *fandisk* model (Figure 33). This model has points clearly located on sharp edges, as well as a variety of different types of sharp features. The validation set also includes the 3 wedge shapes shown in Figure 12 (page 39). The wedge shapes were not part of the training set in our evaluation. We automatically marked edges with sharpness value 1 based on the angle between their adjacent faces, and then diffuse the sharpness value to nearby points, with the sharpness value diminishing linearly to 0 with distance. The quality metric we use for evaluating the sharpness field is the mean square error of the sharpness value at each point. Mean square error is a standard metric for real-valued quantities such as sharpness field values.

We see from Table 1 that convolutional neural networks (CNNs) have a slight advantage over other machine learning models in the quantitative comparison. We can also see from the results obtained on the *blade* model (Figure 34) that qualitatively, the CNN does a better job of capturing creases.

Model	Fandisk	Wedges
Linear Regression	0.015019	0.044839
Linear SVM	0.009747	0.017954
Random Forest	0.006219	0.005651
Gradient Boosting	0.010869	0.008251
CNN	**0.005976**	**0.005275**

Table 1: Comparison of error in the sharpness field for different machine learning models

Figure 33: Ground truth sharpness field of the *fandisk* model.

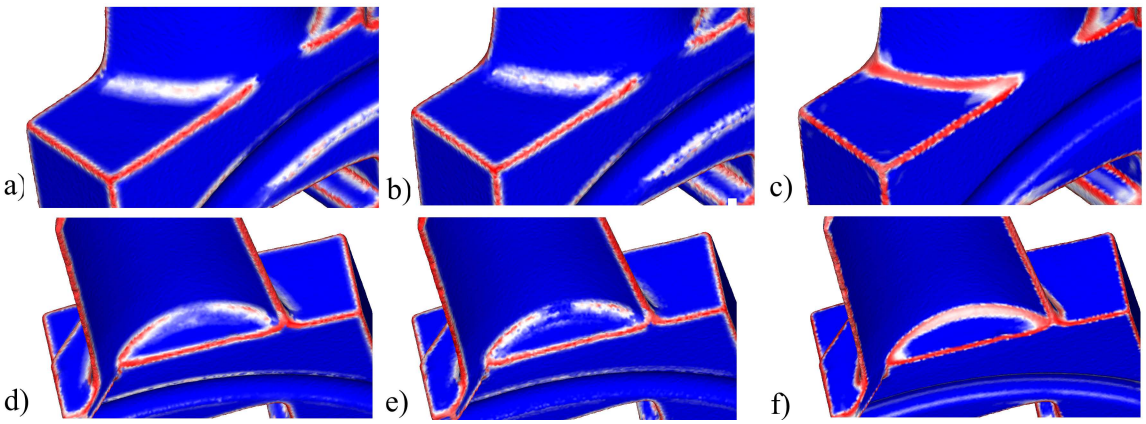

Figure 34: Comparison of gradient boosting, random forests and convolutional neural networks (CNNs). a,d) Gradient boosting results. b,e) Random forest results. c,f) CNN results.

4.2 Representations and Models for Sharpness Field Computation

Section 4.2.1 presents the sharpness field computation results using radial grid heightmaps and CNNs, as described in Section 3.2.1. Section 4.2.2 shows how we obtain improved results over those in Section 4.1 by using Cartesian heightmaps in combination with harmonic networks and spatial transformer networks, as described in Section 3.2.2. Section 4.2.3 contains the sharpness field results obtained using PointNet and PCPNet, as described in Section 3.2.3.

Section 4.2.4 presents a quantitative evaluation of all evaluated neighborhood representations and neural network models for the case of curved edges. This includes analysis of how the scale of the neighborhood affects the accuracy of the computed sharpness field. The results in this section have been published in [10].

4.2.1 Radial grid heightmaps

The training process initially showed instability from one epoch to the next, resulting in an output value range much larger than $[0, 1]$, which kept expanding over the course of training. We improved the stability of the training by adding a sigmoid activation layer at the end of the network, which essentially constrains the output range to $[0, 1]$. The results obtained were still not satisfactory because we did not obtain a smooth field (Figure 35). However, the network did seem to be generalizing to the curved creases of the *blade* model, which was an encouraging sign for the overall approach of using heightmaps.

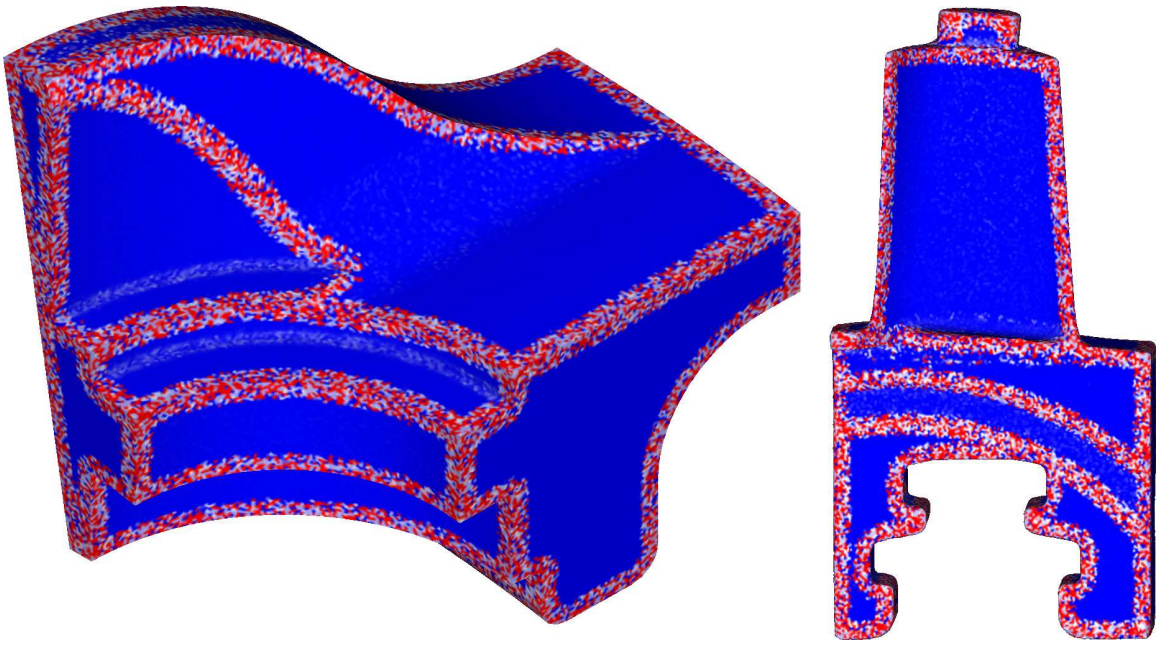

Figure 35: Sharpness fields obtained using a CNN trained on radial grid heightmaps, as described in Section 3.2.1.

4.2.2 Cartesian heightmaps

The two network architectures that used Cartesian heightmaps — the harmonic network and the CNN with a spatial transformer network — gave similar results to the CNN with the normal angles as input (Section 3.1.4) in most respects, except that they were also able to capture the curved edges in the *blade* model, as shown in Figure 36. These results were obtained with a neighborhood radius of 3δ. An additional result for the wooden house facade is shown in Figure 37.

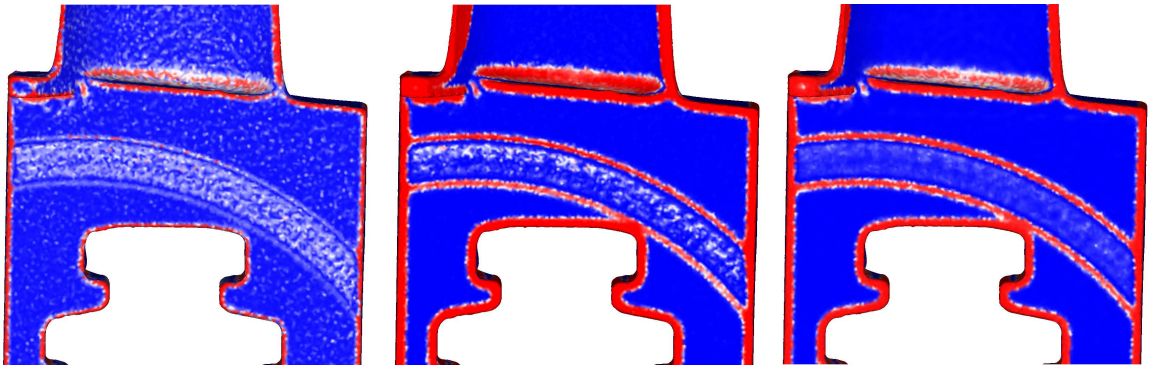

Figure 36: Sharpness fields computed for the *blade* model using Cartesian heightmaps. (L to R) Using an ordinary CNN, using a harmonic network, and using a CNN with a spatial transformer network.

Figure 37: Sharpness field computed for the *house* model using Cartesian heightmaps passed to a CNN with an STN.

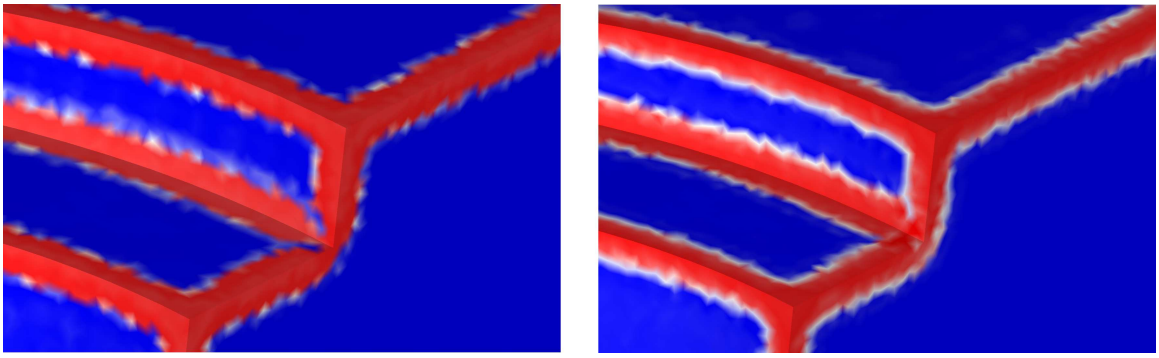

Figure 38: Sharpness field results obtained at concave corners, using a CNN with a spatial transformer network. L: after 12 epochs. R: after 2,000 epochs.

Figure 36 also illustrates that the curved edges of the *blade* model are not detected by an ordinary CNN, and that we need either harmonic convolutions or the STN to obtain an improvement over the normal angles representation.

For the *fandisk* model, we observed a gap in the sharpness field on concave edges leading towards corners (Figure 38). After over 2,000 epochs of training, the STN+CNN network eventually learned to fill this gap, albeit in a manner which does not maintain the continuity of the field. The Cartesian heightmap representation therefore appears to have disconnected subsets near concave corners.

4.2.3 PointNet-based approaches

Examples of sharpness fields obtained using PointNet are shown in Figure 39. We can see that in the case of PointNet, the computed sharpness field is comparable to the results we obtained with the normal angles (Figures 27 and 28 on page 67). The curved edges of the *blade* model are not distinctly captured, although the computed sharpness field value is around 0.5 along these edges.

The results we obtained using the multi-resolution PCPNet are shown in Figure 40. Surprisingly, the computed sharpness field is noisy around sharp edges, and visibly lacks the local monotonicity property that we seek. The curved edges of the *blade* model are also not captured. This is likely because the multi-resolution neighborhood representation requires a greater variety of training data, in which the data is diverse at multiple neighborhood scales. The lack of such variety in our training data causes PCPNet to overfit the training set.

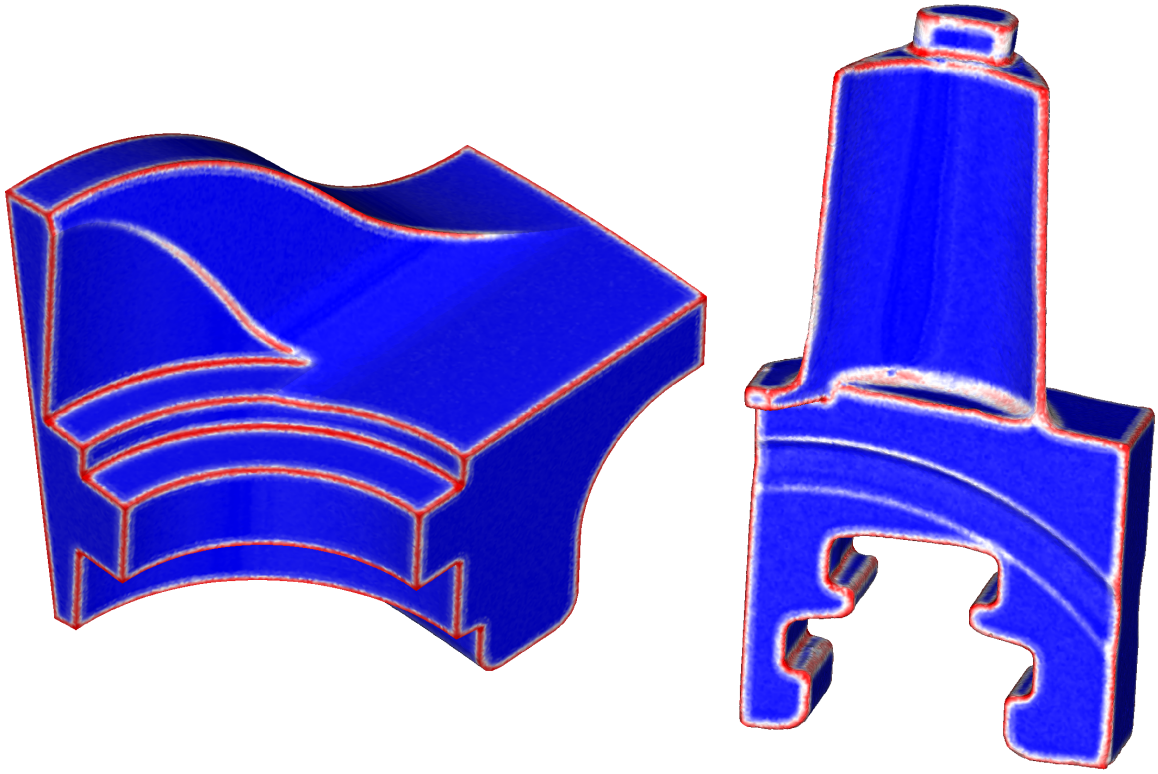

Figure 39: Sharpness fields for the *fandisk* and *blade* models, computed using PointNet.

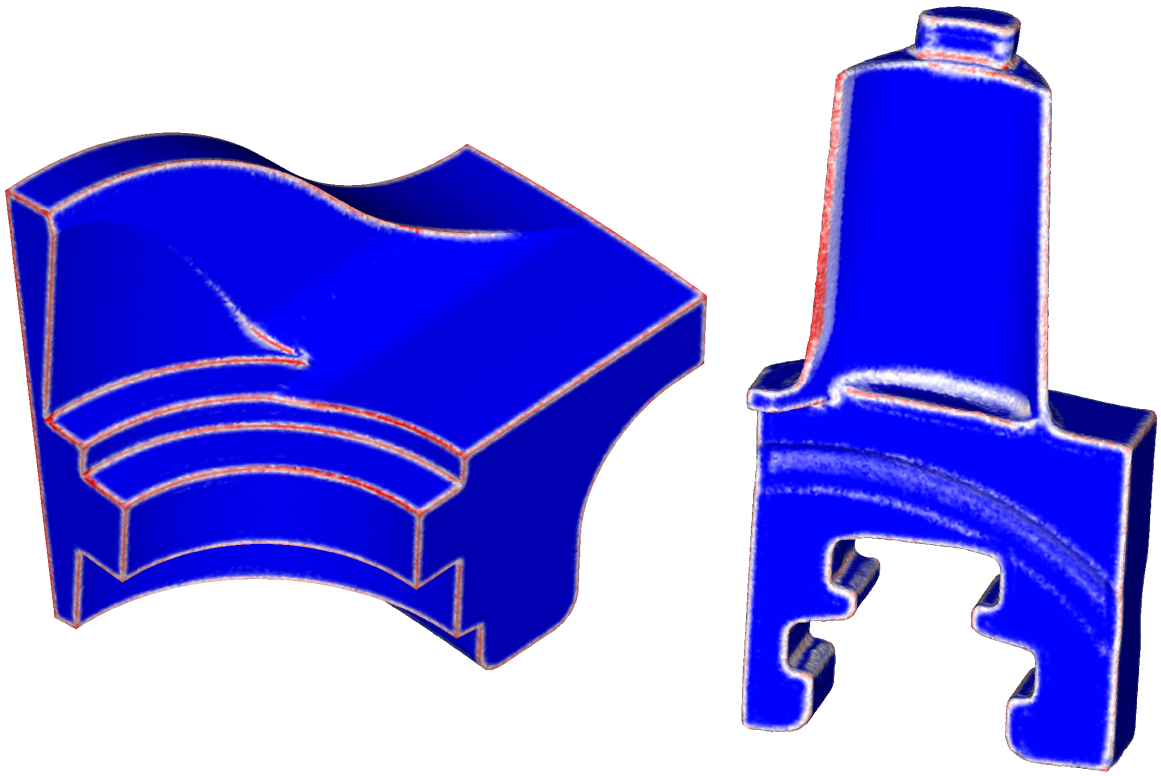

Figure 40: Sharpness fields for the *fandisk* and *blade* models, computed using a multi-resolution PCPNet.

4.2.4 The effect of scale on sharpness computation

In Chapter 3, we have listed a total of 7 approaches to computing the sharpness field of a point cloud. Our motivation for experimenting with different neighborhood representations and models in Section 3.2 was to improve computation of the sharpness field on the curved edges of the *blade* model. Therefore we have performed a numerical evaluation of all 7 approaches for computing the sharpness field on these curved edges. We also analyze how the scale of the neighborhood affects the performance of each method.

4.2.4.1 Half-pipe model

The *blade* model is a 3D scan of a real-world object, which naturally lacks points lying on sharp edges. Therefore, one cannot compute a ground-truth sharpness field for the *blade* model, even by using the mesh connectivity. In order to perform a fair comparison of the various methods on such a shape, we have designed the *half-pipe* model shown in Figure 41. The shape was constructed by extruding a circular arc along a larger circular arc. This model is meant to resemble the curved depression in the *blade* model, with the key difference being that it has points which lie on sharp edges. This allows us to compute a ground-truth sharpness field using dihedral angles, just as we did for the training data. We chose the name because of its resemblance to half-pipes in children's playgrounds.

4.2.4.2 Comparative evaluation on the half-pipe model

For each technique, we have computed the sharpness field on the *half-pipe* model using 16 neighborhood sizes in the range 2δ to 17δ. This enables us to assess the performance of the models at different scales. The error metric at each scale is the squared difference between the computed sharpness field and the ground truth sharpness field.

Model	Min	Mean	Std. Dev.
CNN, sorted angles	293.3	354.7	36.8
CNN, unsorted angles	111.0	270.3	122.7
CNN, heightmap	386.1	416.1	14.4
Harmonic Net, heightmap	189.0	296.4	66.1
CNN + STN, heightmap	151.3	**215.4**	42.9
PointNet (single-res)	321.8	390.3	53.2
PCPNet, multi-res	395.5	418.9	18.2

Table 2: Comparison of total squared error in the sharpness field for the *half-pipe* model, using different methods presented in this Sections 3.1 and 3.2.

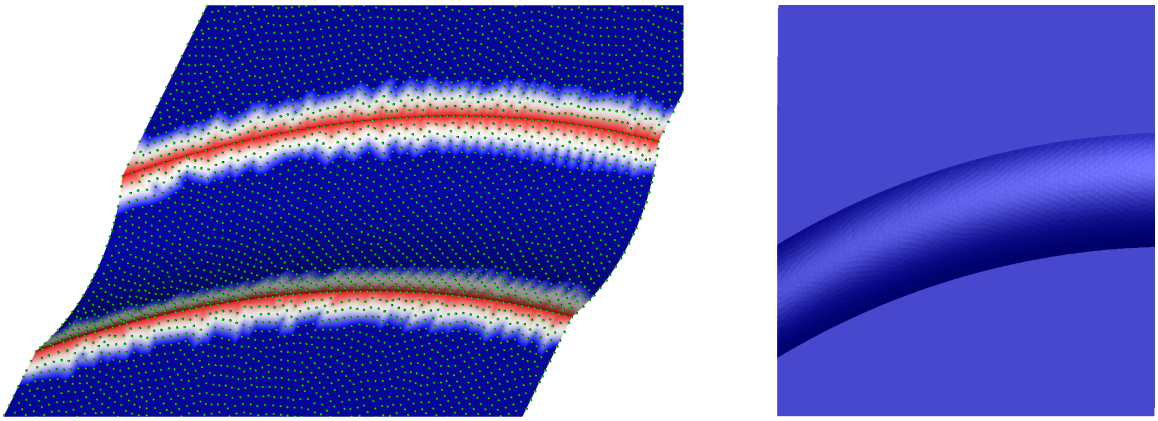

Figure 41: The *half-pipe* model used in our experiments. L: Ground truth sharpness field along with points. R: Top view, shaded.

Table 2 shows the minimum, mean and standard deviation of the squared error across the 16 neighborhood sizes. Note that a network which outputs 0 for all points would result in a squared error of 418. We can see that Cartesian heightmaps, when combined with STNs, perform the best. The low mean and standard deviation for Cartesian heightmaps with harmonic networks and STNs show that they perform consistently well across different scales. The unsorted normal angles also perform well at the larger scales. However, their performance is highly scale-dependent, as evidenced by the high standard deviation. PointNet and PCPNet were not able to generalize well to this model, and neither did the vanilla CNN (except for the unsorted angles at some scales).

Tables 3 and 4 show the complete numerical results for different models at different scales. Each cell shows the total squared difference between the sharpness field computed for the *half-pipe* model in Figure 41 and its ground-truth sharpness field. For the multi-resolution PCPNet, we used 3 neighborhood radii differing by 2δ, and each column corresponds to the middle radius. As a rule of thumb, squared error values greater than 400 imply that the network completely failed to capture the edges.

Model/Scale (times δ)	2	3	4	5	6	7	8	9
CNN, sorted angles	418	418	401	336	308	293	302	328
CNN, unsorted angles	418	418	415	413	405	379	328	265
CNN, heightmap	392	399	414	408	386	413	413	417
H-Net, heightmap	426	324	234	192	189	212	250	279
CNN+STN, heightmap	263	151	206	337	273	213	210	204
PointNet (single-res)	375	336	322	325	334	339	353	369
PCPNet, multi-res	—	—	423	406	397	395	400	406

Table 3: Comparison of total squared error in the sharpness field for the *half-pipe* model, using different methods presented in Chapter 3. Scales 2δ to 9δ.

Model/Scale (times δ)	10	11	12	13	14	15	16	17
CNN, sorted angles	351	357	355	365	369	356	360	359
CNN, unsorted angles	219	203	183	149	121	111	131	168
CNN, heightmap	417	423	425	429	437	429	433	422
H-Net, heightmap	294	308	319	328	336	342	350	357
CNN+STN, heightmap	199	188	190	193	195	197	211	216
PointNet (single-res)	386	405	426	441	455	459	462	458
PCPNet, multi-res	415	425	433	440	443	443	—	—

Table 4: Comparison of total squared error in the sharpness field for the *half-pipe* model, using different methods presented in Chapter 3. Scales 10δ to 17δ.

4.3 Applying the Sharpness Field

Section 4.3.1 shows the feature-aware smoothing results we obtained using the anisotropic Gaussian kernel described in Section 3.3.2. The results are compared with the state-of-the-art RIMLS [16] smoothing method. A quantitative comparison of our method with RIMLS is given in Section 4.3.2. Section 4.3.3 shows examples of shapes whose points have been segmented into patches using the method described in Section 3.3.3. Section 4.3.4 shows examples of edge graphs extracted using the method described in Section 3.3.4.

4.3.1 Feature-aware smoothing using an anisotropic Gaussian kernel

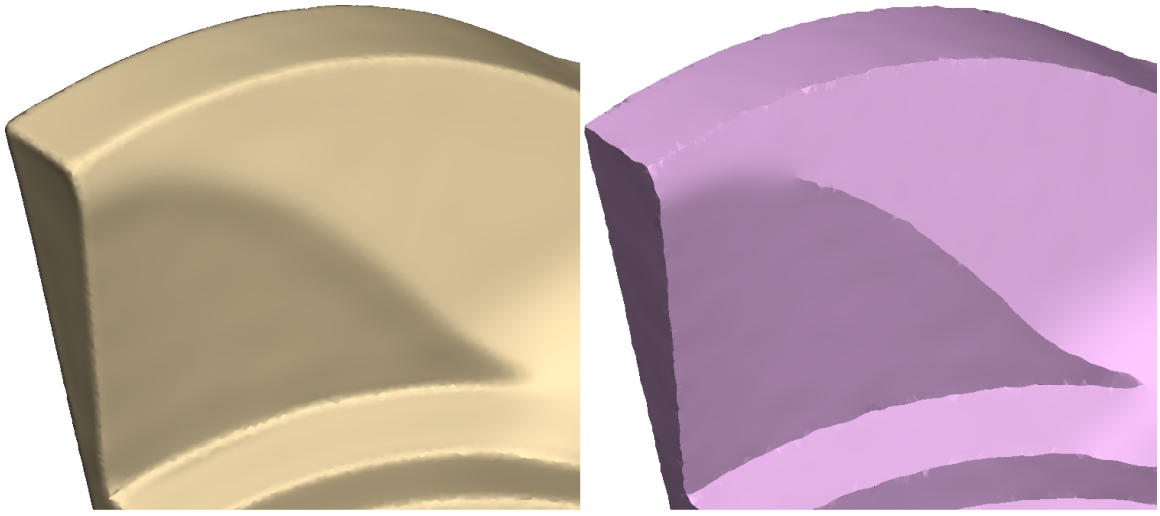

Figure 42: Zoomed-in smoothing results for the *fandisk* model, with 1.5δ normal noise added. Left: RIMLS result. Right: our result. Full comparison in Figure 45.

We have implemented the feature-aware smoothing algorithm described in Section 3.3.2 using C++ and CUDA 9.2. Our program was able to smooth a point cloud with 100K points in 4 minutes, with an Intel i7 quad-core CPU and an NVIDIA Geforce GTX Titan X GPU having 12 GB VRAM.

We tested our algorithm on the same four models listed in Section 4.1.1 and compared them to the results obtained from the implementation of RIMLS [16] available in Meshlab [67]. The same kernel size (8δ) was used for both smoothing methods. For RIMLS, the default sharpness parameter (0.75) was used.

As can be seen in the example shown in Figure 42 (zoomed-in version of results in Figure 45), our smoothing method does a better job of preserving sharp edges than RIMLS. In addition, the reason why the edges look sharp is because we are able to obtain points lying on the sharp edges,

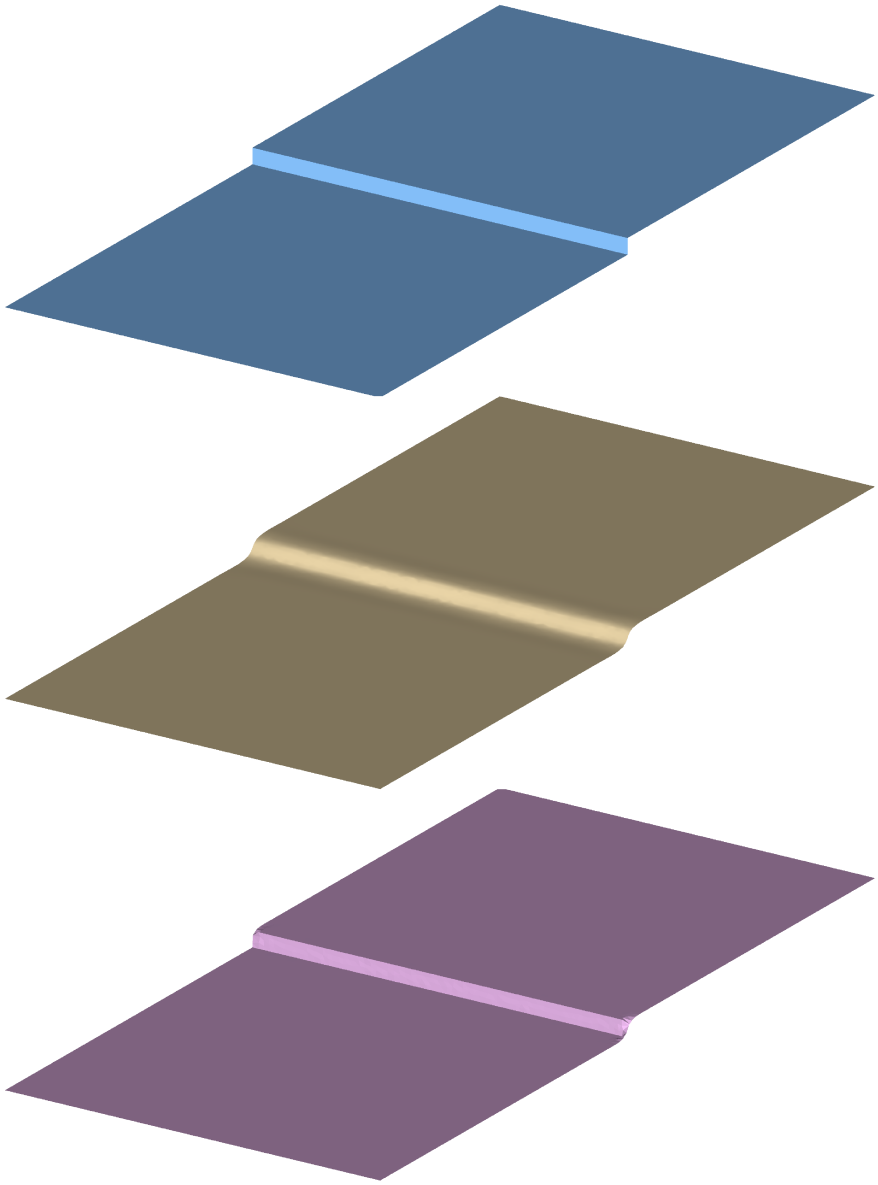

Figure 43: Smoothing results for a surface with a short step-like feature. Top: original. Middle: RIMLS result. Bottom: our result.

using the technique described previously in this section.

Figure 43 shows an example which illustrates the value of explicitly detecting sharp edges. RIMLS, a method which uses robust statistics to implicitly handle sharp edges as outliers, is unable to preserve a small step-like feature if the vertical patch is not tall enough. Our method, on the contrary, can easily avoid smoothing a step-like feature because it is explicitly detected by the neural network.

The complete feature-aware smoothing results are shown in Figures 45 to 50. Our method generally better preserves the sharp features, especially when these features are very small compared to the kernel radius. This can be seen in the house facade example (Figure 50), where the window and door frame are elevated by only a couple of millimeters.

Improved Results Using Heightmaps

The improved sharpness field obtained using heightmaps and the newer networks as described in Section 3.2.2 results in improved feature-aware smoothing of the *blade* model. Figure 44 shows a comparison of our original results in [7], and the improved results in which the sharpness of the curved edges is preserved [10].

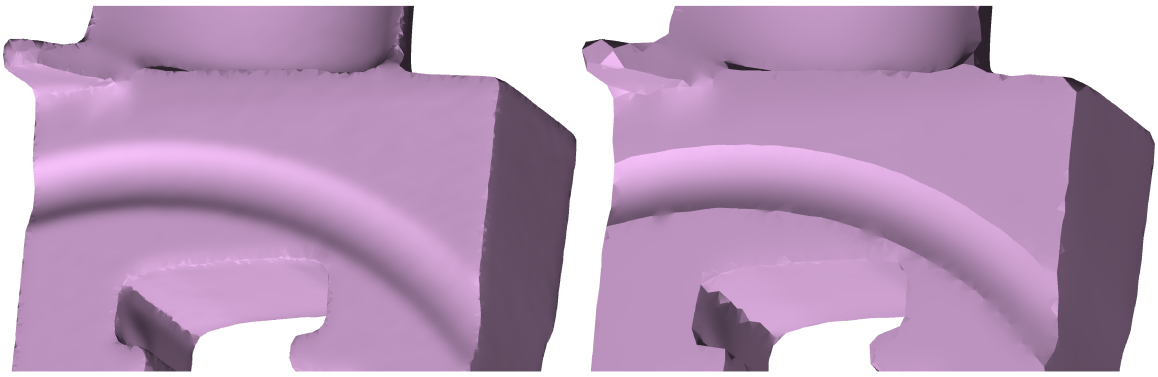

Figure 44: Improved feature-aware smoothing as a result of better sharpness field computation. L: smoothing result for the *blade* model obtained previously [7] for the sharpness field in Figure 28 (page 68), which was computed using the method described in Section 3.1.4. R: improved result [10] obtained for the sharpness field in Figure 36 (page 76), which was computed by using Cartesian heightmaps with the spatial transformer network as described in Section 3.2.2.

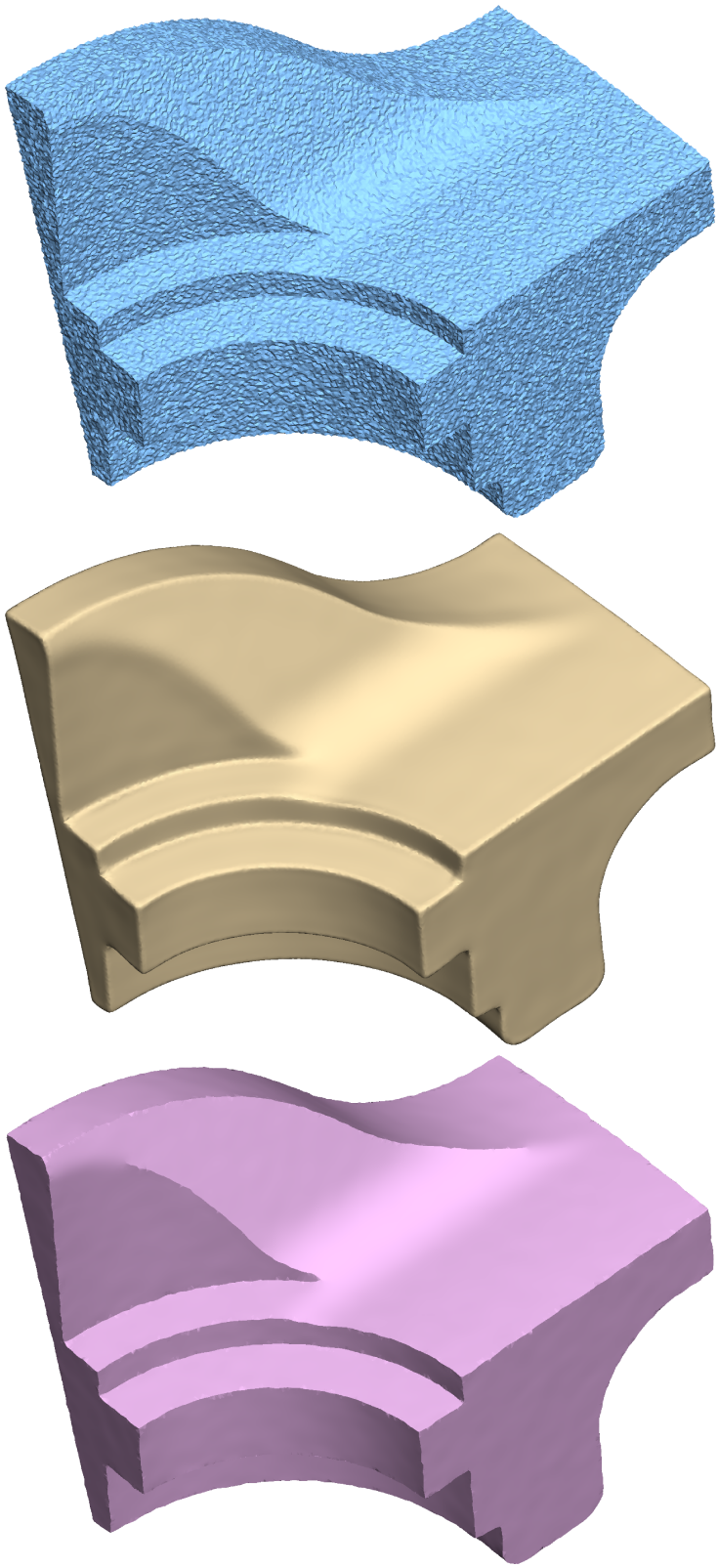

Figure 45: Smoothing results for the *fandisk* model, with 1.5δ normal noise added, using the computed sharpness field from Figure 27 (page 67). Top: original. Middle: RIMLS result. Bottom: our result.

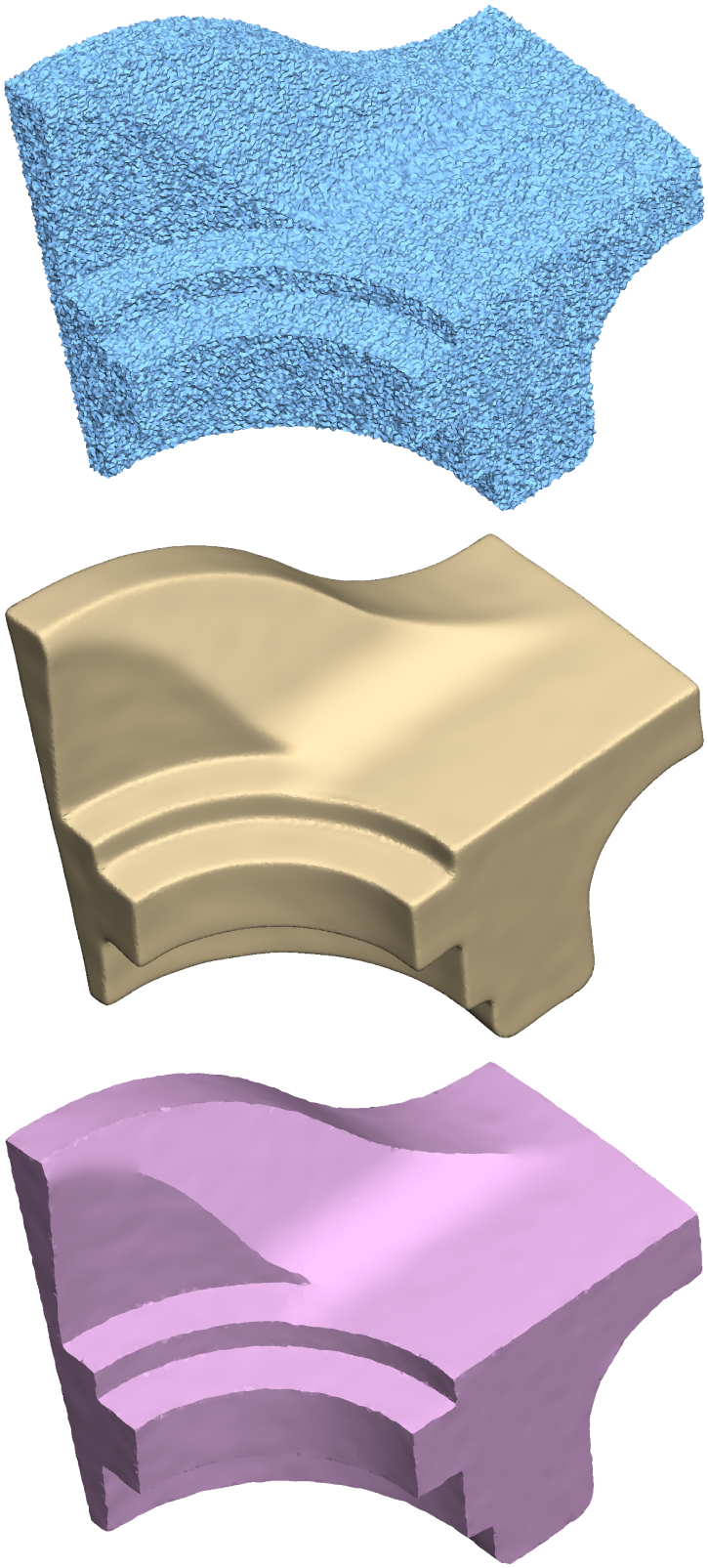

Figure 46: Smoothing results for the *fandisk* model, with 3δ normal noise added, using the computed sharpness field from Figure 27 (page 67). Top: original. Middle: RIMLS result. Bottom: our result.

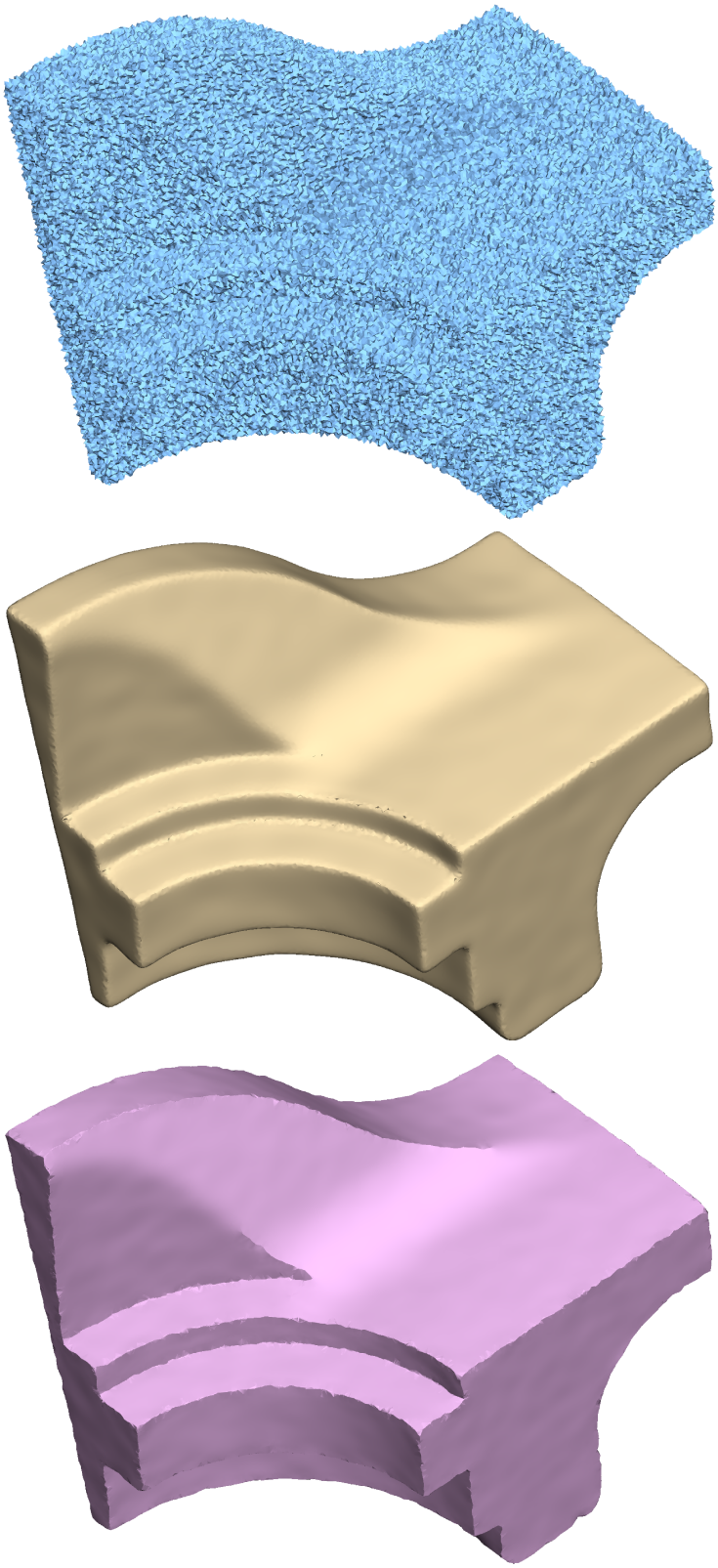

Figure 47: Smoothing results for the *fandisk* model, with 4.5δ normal noise added, using the computed sharpness field from Figure 27 (page 67). Top: original. Middle: RIMLS result. Bottom: our result.

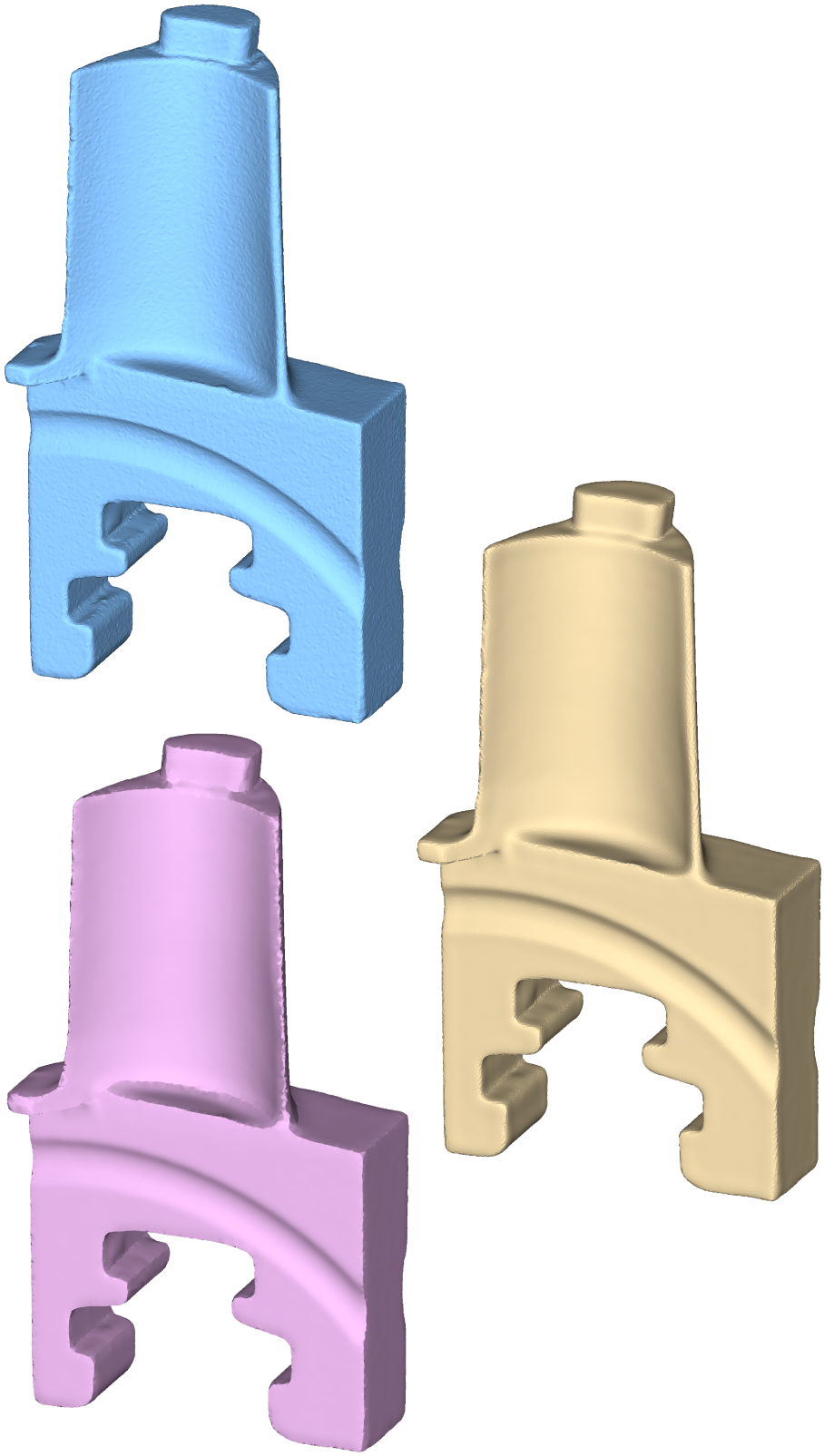

Figure 48: Smoothing results for the *blade* model, using the computed sharpness field from Figure 28 (page 68). Top: original. Middle: RIMLS result. Bottom: our result.

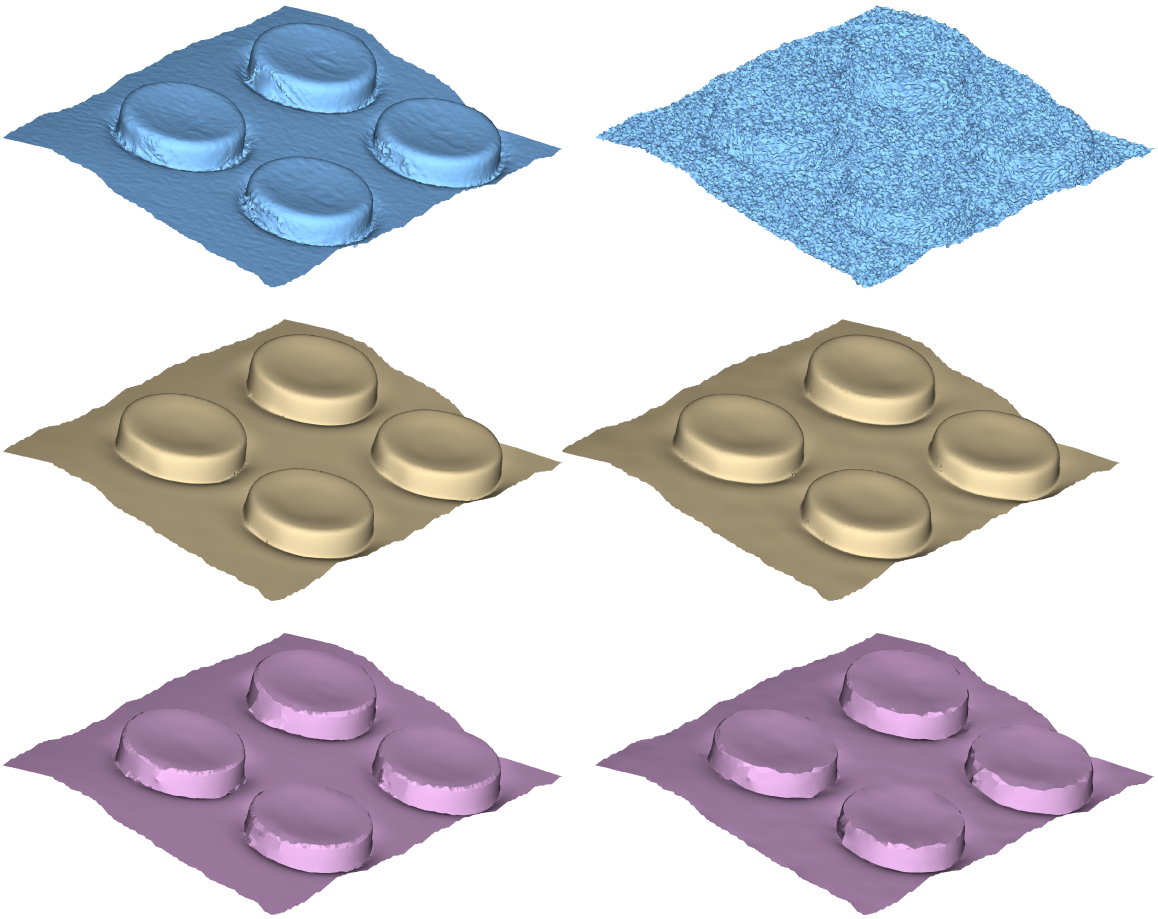

Figure 49: Smoothing results for our 3D scan of a phone keypad, using the computed sharpness field from Figure 28 (page 68). Left column: original scan. Right column: 3δ normal noise added. Top row: original. Middle row: RIMLS result. Bottom row: our result.

Figure 50: Smoothing results for our 3D scan of the facade of a wooden house, using the computed sharpness field from Figure 29 (page 69). Top: original. Middle: RIMLS result. Bottom: our result.

4.3.2 Quantitative evaluation of feature-aware smoothing

We have quantitatively evaluated our feature-aware smoothing method by comparing the smoothed *fandisk* point clouds to the ground truth *fandisk* mesh. The sharpness fields used for computing these results are the same as those given in Section 4.1. Table 5 shows the results obtained for three levels of noise: 1.5δ, 3δ and 4.5δ. We evaluate the smoothing results based on three metrics:

1. Distance of each point in the smoothed point cloud to the ground truth mesh (D2M).
2. Hausdorff distance (HD): the maximum distance between a point on the ground truth mesh and its nearest neighbor in the smoothed point cloud.
3. Chamfer distance (CD) [103]: the mean distance between a point on the ground truth mesh and its nearest neighbor in the smoothed point cloud.

The results obtained using our method are clearly superior to those obtained using the state-of-the-art RIMLS method, with the sole exception of the maximum distance to the ground truth mesh for large amounts of noise. We found this discrepancy to come from a small number of points with false positive sharp edges, due to the neural network model being thrown off by high noise levels in regions of large curvature. One notable point about our results is that the mean distance to the ground truth mesh and the Chamfer distance are superior to those of RIMLS. This suggests that in addition to preserving sharp edges, there is less change in the volume of the shape when using our method. This is in contrast to many traditional smoothing methods that shrink the shape, as well as RIMLS, which expands the shape in order to compensate for shrinkage.

The distance to the ground truth mesh is visualized in Figures 51 and 52. We can see from the heatmaps that the errors of our method are concentrated along sharp edges which were obscured by noise. On the other hand, the larger errors of RIMLS are found around all concave corners, some edges and even in some smooth regions between sharp edges.

Model	**D2M**				**HD**	**CD**
	Min	Max	Mean	Std. Dev		
1.5δ noise, Ours	0	**3.86E-03**	**2.23E-04**	2.82E-04	**3.86E-03**	**2.30E-04**
1.5δ noise, RIMLS	0	7.91E-03	6.39E-04	6.33E-04	7.91E-03	6.24E-04
3δ noise, Ours	0	11.06E-03	**3.17E-04**	3.46E-04	**5.37E-03**	**3.28E-04**
3δ noise, RIMLS	0	**7.09E-03**	6.50E-04	5.96E-04	5.78E-03	4.59E-04
4.5δ noise, Ours	0	12.98E-03	**4.37E-04**	4.92E-04	**8.80E-03**	**4.45E-04**
4.5δ noise, RIMLS	0	**7.42E-03**	7.20E-04	6.45E-04	9.99E-03	7.57E-04

Table 5: Quantitative comparison of our feature-aware smoothing method with RIMLS.

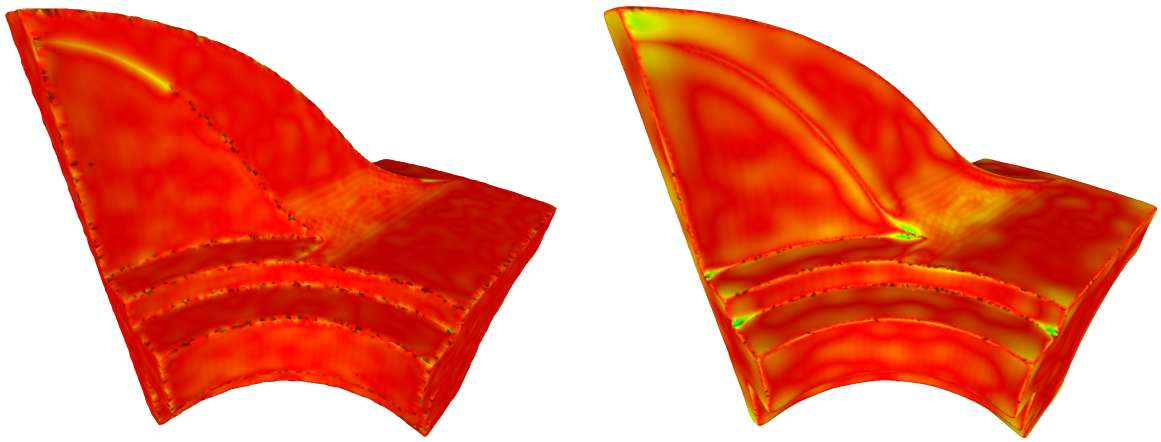

Figure 51: Visualization of smoothing errors after smoothing a *fandisk* with 3δ noise. Red represents minimum error and blue represents maximum error. Left: Ours. Right: RIMLS.

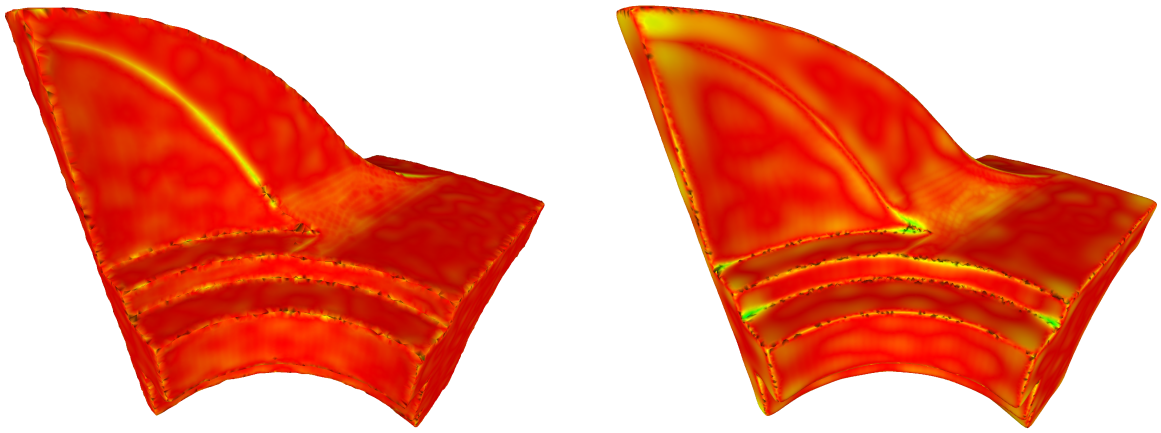

Figure 52: Visualization of smoothing errors after smoothing a *fandisk* with 4.5δ noise. Red represents minimum error and blue represents maximum error. Left: Ours. Right: RIMLS.

4.3.3 Segmentation into patches

The results shown here were explained in Section 3.3.3 (page 60).

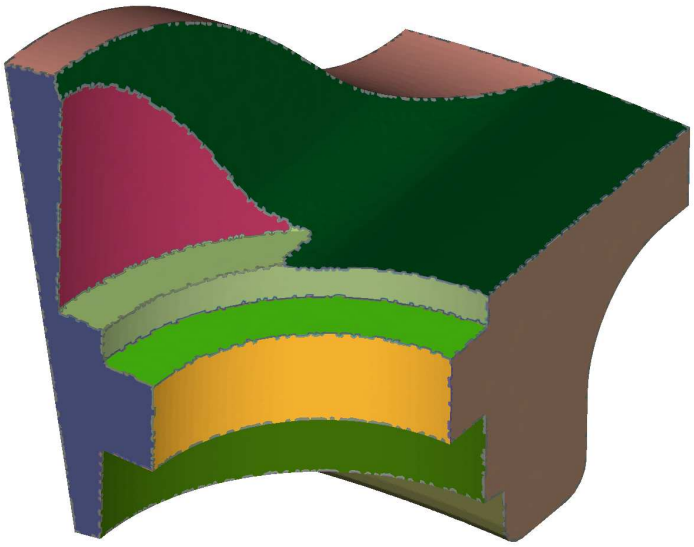

Figure 53: The *fandisk* model, after segmentation into patches using the ground truth sharpness field from Figure 33 (page 74). Points near edges were assigned to the nearest patch.

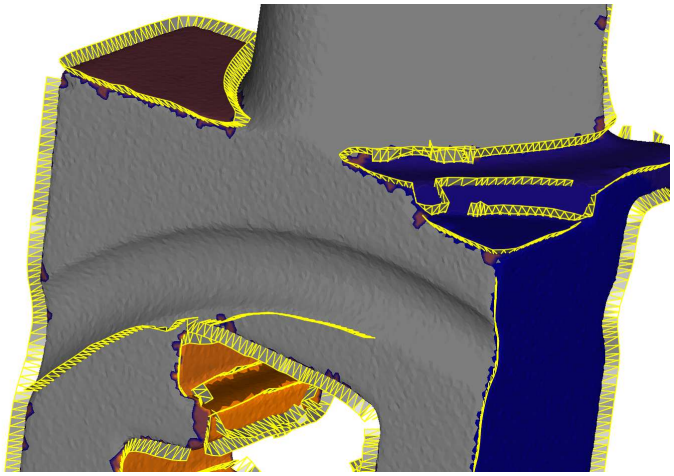

Figure 54: The *blade* model, after segmentation into patches using both the computed sharpness field from Figure 28 (page 68), as well as a barrier mesh generated from the edge graph (that was generated from the computed sharpness field).

4.3.4 Edge graph extraction

The results shown here were explained in Section 3.3.4 (page 61).

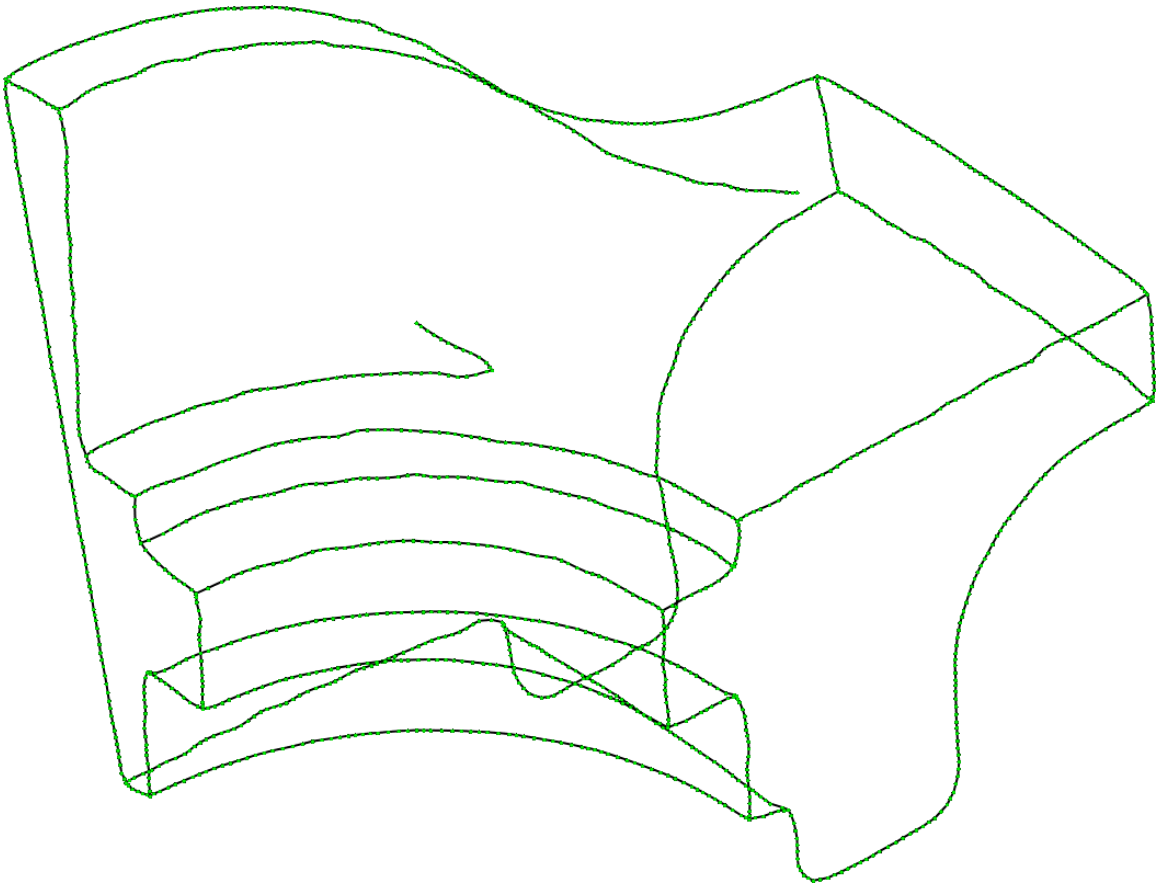

Figure 55: An edge graph generated for the *fandisk* model, using the computed sharpness field shown in Figure 27 on page 67.

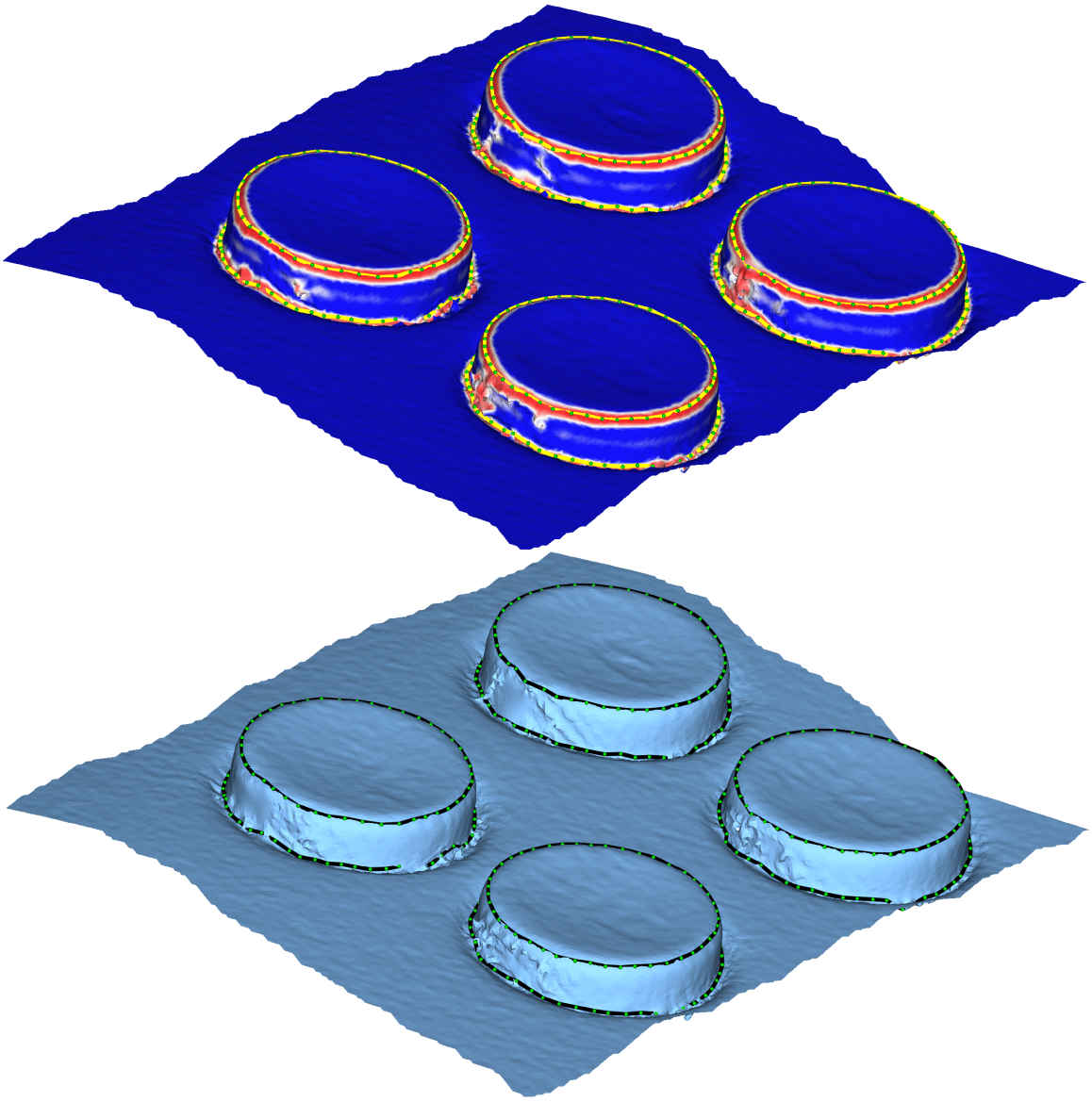

Figure 56: An edge graph generated for our scan of a phone keypad, shown along with the computed sharpness field from Figure 28 on page 68.

In this chapter, we have seen that deep learning is an effective tool for sharpness field computation. In particular, the sharpness field can be computed accurately for a large range of neighborhood scales by feeding Cartesian heightmaps to a CNN equipped with a spatial transformer network. We then showed that the sharpness field could be used to reconstruct sharp edges using our feature-aware smoothing algorithm. Sharp edges are a particularly challenging type of fine feature to reconstruct, as they arise from discontinuities in the normal field of a surface. This necessitates treating sharp edges as a special case of fine features, as we have done so far. In the following chapter, we treat reconstruction of fine features as a point cloud super-resolution problem. This a more general problem, which expands the scope of the fine features that we seek to reconstruct beyond merely sharp features.

Chapter 5

Feature Preservation Through Point Cloud Super-Resolution

The feature-aware smoothing technique described in Chapter 3 allows us to preserve sharp edges while smoothing a point cloud, including by moving smoothed points directly onto sharp edges. However, the fundamental reason why previous reconstruction methods fail to reconstruct fine features is due to undersampling of regions of the surface with fine details. Therefore, we need to consider another broad approach to reconstructing fine features, by performing super-resolution of a given point cloud, to obtain a denser and more detailed point cloud.

Ultimately, the goal is to reconstruct a more detailed surface representation (such as a mesh) from the high-resolution point cloud. Therefore, we are more concerned with the quality of the reconstructed surface than the properties of the high-resolution point cloud. For example, we need not be overly concerned about the sampling evenness of the high-resolution point cloud as long as it does not negatively affect surface reconstruction. After all, if a higher-resolution surface can be reconstructed, a point cloud with desirable properties such as evenness can be obtained by simply resampling the reconstructed surface.

In Section 2.2.4, we have already described related work in point cloud super-resolution. The state-of-the-art methods all perform super-resolution of subsets of input point clouds called "patches", using neural network architectures based on PointNet++. This approach has certain inherent limitations (see Section 6.1.1). Our method is altogether different from previous methods and avoids these limitations.

In this chapter, we describe our novel method for point-cloud super-resolution by *domain translation* of the local heightmaps described in Section 3.2.2. This domain translation allows us to obtain more densely sampled local neighborhoods for each input point, thereby obtaining a denser point cloud. Section 5.1 describes the current state-of-the-art technique for domain translation, which uses Generative Adversarial Networks (GANs). We apply this technique in our point cloud super-resolution method, which is described in Section 5.2.

For point clouds with an already-computed sharpness field, the same point cloud super-resolution technique of ours can be adapted to perform super-resolution of the computed sharpness field as well. This is described in Section 5.3.

The material in this chapter and Chapter 6 has been accepted for publication [12].

5.1 GANs for domain translation

Generative adversarial networks (GANs) are a class of neural network generative models introduced by Goodfellow *et al* in 2014 [25]. Previous neural generative models such as variational autoencoders (VAEs) sought to minimize ℓ_1 or ℓ_2 loss when modeling the probability distribution of a set of data. GANs, on the other hand, seek to minimize an *adversarial loss*, which represents how "realistic" a generated sample appears.

GANs are composed of two sub-networks:

- a *generator*, $G(z) : \mathbb{R}^m \to \mathbb{R}^n$, which attempts to generate realistic samples that appear to be sampled from the distribution of the n-dimensional training data. Different samples are generated for different noise vectors $z \in \mathbb{R}^m$. Typically, $m << n$.
- a *discriminator*, $D(y) : \mathbb{R}^n \to [0, 1]$, which attempts to classify data samples as real or fake. $D(y)$ is the probability that y is genuinely sampled from the training dataset, and not artificially generated by the generator.

The objective of the GAN training is given by the following adversarial loss function:

$$\mathcal{L}_{GAN} = \mathbb{E}_y[\log D(y)] + \mathbb{E}_z[\log(1 - D(G(z))] \tag{7}$$

The generator and discriminator effectively play a minimax game, which ideally converges to a Nash equilibrium during training. At this point, the trained generator is able to generate data

samples which are sufficiently realistic to fool the trained discriminator. The trained generator is given by:

$$G^* = \arg \min_G \max_D \mathcal{L}_{GAN}(G, D) \tag{8}$$

The GAN formulation described above provides a generative model which generates data samples which realistically appear to be sampled from a particular domain of data. This has been used, for instance, to randomly generate realistic-looking photos of non-existent celebrities, after training on a dataset of photos of celebrities. However, it does not provide any way to control the generated data samples. This limits the utility of the basic GAN architecture.

GANs become truly useful when they are extended to include some form of supervision or control over their learning process. The most popular GAN application is in *domain translation*, that is, translating a data sample from one domain to another structurally similar domain. This began with the seminal work Isola *et al* [17] on using conditional adversarial networks for translating between image domains given a training set of paired examples. For example, their work is able to translate images of street maps into plausible satellite images of the same terrain, as well as between sketches of shoes and plausible photorealistic images of shoes corresponding to the sketches.

This quickly led to more work on unpaired image translation [27], as well as image translation between more than two domains [28]. There have also been attempts to bring GAN-based domain translation methods to other domains such as natural language text [29], audio [30] and voxel-based 3D data [31].

The basic idea behind paired image-to-image translation using GANs is to modify the generator and discriminator to be conditioned on an input image, x. This leads to a modified GAN objective:

$$\mathcal{L}_{GAN} = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z))] \tag{9}$$

In addition, it is typical to add a simple ℓ_1 loss to the objective, in order to improve the quality of the low-frequency information in the resulting image:

$$\mathcal{L}_{\ell_1} = \mathbb{E}_{x,y,z} \mid y - G(x, z) \mid_1 \tag{10}$$

The trained generator is given by:

$$G^* = \arg \min_G \max_D \mathcal{L}_{GAN}(G, D) + \lambda \mathcal{L}_{\ell_1}(G) \tag{11}$$

where λ is a weighting coefficient for the ℓ_1 loss.

In practice, the noise vector z is introduced implicitly by random dropout of neurons with 50% probability. Furthermore, the discriminator used for image-to-image translation does not attempt to classify the entire input image as real or fake, but rather classifies individual patches (referred to as a *PatchGAN* architecture by Isola *et al* [17]).

Now that we have reviewed how GANs can be used for domain translation, we can see how this domain translation method can be used, not for image translation per se, but for point cloud super-resolution.

5.2 Super-resolution of point clouds using domain translation

The key idea behind our method is that super-resolution of point cloud neighborhoods can be reduced to an image-to-image translation problem between two kinds of local heightmaps (examples shown in Figure 57):

1. Sparse heightmaps, which are sampled from point clouds. By "sparse", we mean that not all pixels in the heightmap are occupied. This follows naturally from the fact that point clouds have gaps between points. In practice, we also set an upper limit on the number of points contributing to a sparse heightmap, which makes our method more robust while also speeding up computation of the sparse heightmaps. Therefore, they are also sparse in the conventional sense that there are $O(1)$ points represented in the heightmap.

2. Dense heightmaps, which are sampled from meshes. Since the mesh is typically defined everywhere in a neighborhood, we can obtain a heightmap by casting rays onto the mesh. It is worth noting that this raycasting approach can easily be modified to obtain other kinds of local "maps", such as scalar or vector fields defined over the mesh.

In this section, we first explain how these two types of heightmaps are generated (Section 5.2.1). We then show how our overall approach has additional benefits for estimating normals (Section 5.2.2).

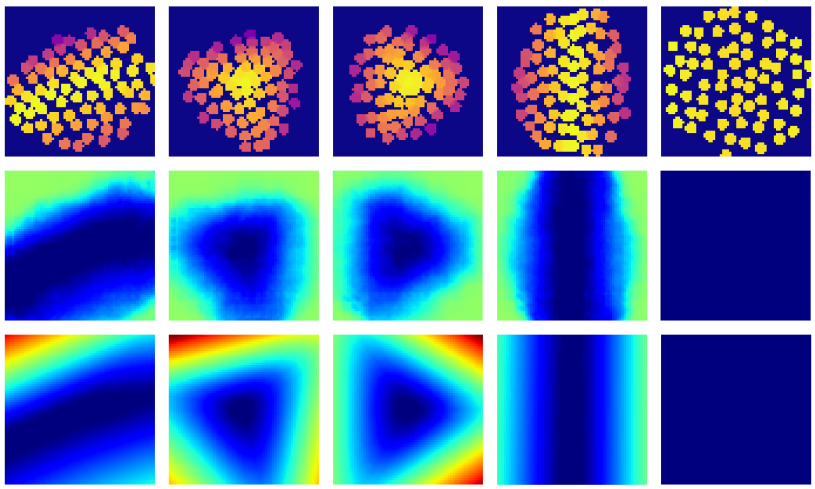

Figure 57: Heightmaps sampled from the *fandisk* model. The sparse and dense heightmaps have different color maps to reflect the fact that sparse heightmaps are not occupied at all pixels. Top row: sparse heightmaps obtained from the point cloud. Middle row: dense heightmaps predicted by our GAN. Bottom row: ground truth heightmaps obtained by casting rays onto the original mesh.

5.2.1 Local heightmap generation

Some aspects of heightmap computation are common to both the sparse and dense local heightmaps. The heightmap is assigned a size of $k \times k$ pixels, where we generally select $k = 64$. The side of the heightmap corresponds to a length of $2r$ in the space of the input shape, where r is the *search radius* used to collect nearest neighbors from the point cloud. The local coordinate frame of the neighborhood is centered on a particular point of the point cloud, which has a corresponding oriented normal $\mathbf{n}$. The horizontal and vertical directions of the local frame are given by an arbitrary tangent $\mathbf{t}$ and its corresponding bitangent $\mathbf{b} = \mathbf{n} \times \mathbf{t}$.

5.2.1.1 Sparse heightmap generation

For computing sparse heightmaps from point clouds, we use a similar heightmap representation to the one described in Section 3.2.2. A random set of neighboring points (limited to 100) is chosen from within the search radius r. Since our sampled points have consistent associated normals, we can omit neighbors with back-facing normals. The neighborhood is scaled by a factor of $1/r$ to reduce dependence on scale. These points are then projected orthogonally onto the local tangent plane, i.e. the plane of the heightmap image. For each pixel in the heightmap image, we can easily compute the corresponding pixel center in the local coordinate frame of the neighborhood. We can

then compute the intensity of each pixel as the weighted average of the signed distances of nearby projected points from their original positions. The unnormalized weights have a Gaussian falloff $w_i = exp(-\frac{d_i^2}{2\sigma^2})$, where d_i are the distances of the projected points from the pixel center in the image plane (we set $\sigma = 5r/k$). A constant value 1 is added to all projection heights, so that the value 0 is reserved for unoccupied pixels. As shown in Section 5.3, the same approach can also be used to generate a sparse map, not for height, but for a scalar field.

5.2.1.2 Dense heightmap generation

Given a point on a surface represented by a mesh (which need not be a vertex), and its corresponding normal, we can use raycasting to generate a dense heightmap. We first transform the center of each heightmap pixel to the coordinate space of the mesh. We can do this because we know the local tangent frame, as mentioned earlier. From each pixel center, we shoot two rays in opposite directions perpendicular to the tangent plane. If both rays intersect the mesh, we choose the nearer intersection point. The intensity of the pixel is the signed distance of the pixel center to the intersection point, or a fixed large value in the event that neither ray intersects the mesh. These raycasting operations are parallelized efficiently on the CPU using the Embree raycasting framework [109]. Embree also provides us with the intersected triangle ID and the Barycentric coordinates of the intersection point. This additional information can be used to interpolate scalar or vector fields previously computed on the vertices of the mesh, in order to generate other kinds of dense local maps.

5.2.2 Prediction of normals

The regular grid structure of a heightmap provides us the additional benefit that it allows easy computation of normals at each point on the heightmap. Given a dense heightmap, we can use backward differences to estimate gradients in the tangent and bitangent directions. This gives us the approximate normal at each point as the direction of the vector $(\frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}, \frac{2r}{k})$. This approximate normal map proved to be sufficient for our purposes, although it could also be refined using a neural network.

5.2.3 Point cloud super-resolution

We trained our GAN on a set of 90 meshes of statues obtained from SketchFab. These meshes were generated by 3D-scanning statues of people and animals (Figure 58). Training details such as

Figure 58: Examples of training models from SketchFab.

hyperparameters are given in Section 5.2.4. For evaluation, we have procured 16 additional meshes of statues. All ground truth meshes have several hundred thousand vertices.

For point cloud super-resolution, we randomly sample a fixed number of points from a test mesh using Poisson disk sampling (using the implementation available in the VCG library [110]). Figure 59 shows an example of sampling these points. Normals for these points are estimated using PCA, based on 30 nearest neighbors.

After generating sparse 64×64 heightmaps for these sampled points using the method described in Section 5.2.1.1, we use our trained model to suggest a plausible dense 64×64 heightmap. Given the search radius as well as the normal and tangent vectors used to obtain the sparse heightmap, we can easily transform pixels of the dense heightmap back into points in the space of the original point cloud. Note that since the conditional GAN is transforming heightmaps at the local level with no global information, we do not expect the extremities of the newly generated heightmap to be accurate. However, we can still obtain a high density point cloud simply by taking points from an 8×8 or 16×16 square in the middle of the generated heightmap.

We experimented with different combinations of heightmap search radius, size of the central square, and stride (a stride of 2 implies taking every alternate row and column). Note that δ is the median distance between a point and its nearest neighbor in the input point cloud, which we use as

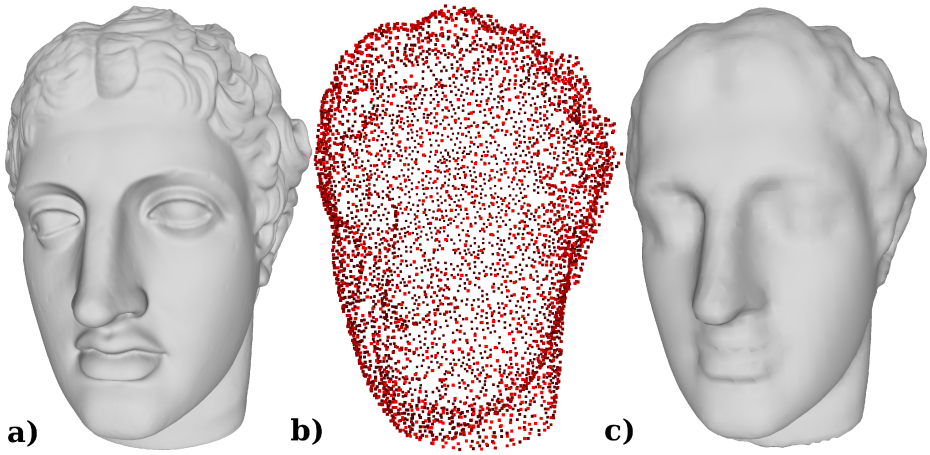

Figure 59: a) Ground truth *head* model. b) 5000 points sampled to act as testing input. c) Poisson reconstruction of the input point cloud, showing the loss of features due to undersampling.

a measure of scale. Out of these combinations, we found three to be interesting:

1. *smooth* mode: Radius 4δ, central 8×8, stride 2 (16x upsampling). This mode produces the smoothest looking mesh when screened Poisson reconstruction is applied.

2. *even* mode: Radius 8δ, central 8×8, stride 2 (16x upsampling). The high-resolution point cloud obtained with this mode appears to be the most evenly sampled.

3. *superdense* mode: Radius 4δ, central 16×16, stride 1 (256x upsampling). This mode produces an extremely dense point cloud, from which a reasonably accurate mesh can be reconstructed.

Figure 60 compares the results for the three modes, after processing the sampled points from Figure 59.b.

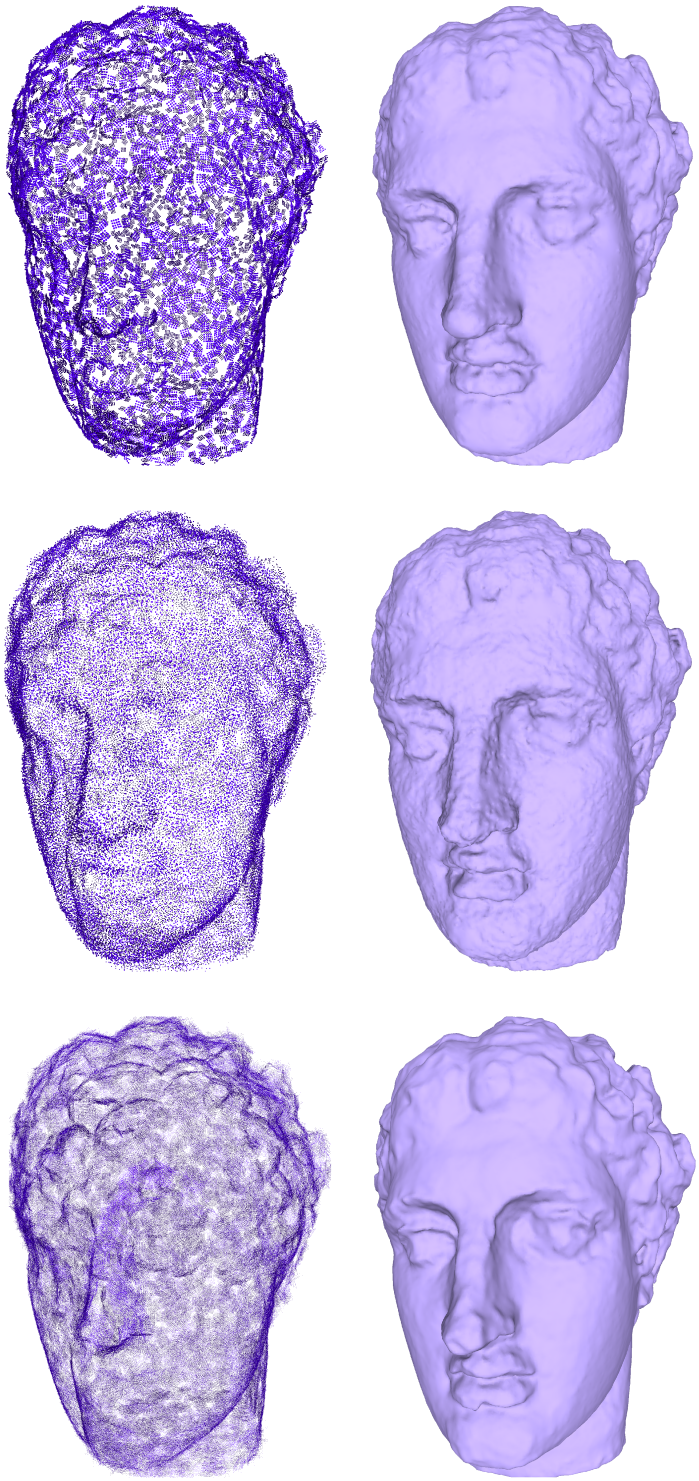

Figure 60: Our super-resolution results for the 5000 points in Figure 59.b, using different modes. Top: *smooth* mode. Middle: *even* mode. Bottom: *superdense* mode. The right column contains their respective reconstructions using screened Poisson surface reconstruction.

5.2.4 Training details

In this section, we provide additional details on training the GAN used in our experiments.

5.2.4.1 Generator architecture

The generator used in our GAN has a U-Net architecture [111]. This is an encoder-decoder architecture in which the image is first progressively downsampled and then upsampled, with skip connections between mirrored layers of the encoder and decoder. These skip connections allow features of the image at different scales (such as edges) to be more easily transferred to the output image during image translation [17]. Following Isola *et al* [17], our generator uses strided convolutions [24], leaky ReLU [112] activations in the encoder portion, as well as instance normalization [113] in all inner layers of the network and dropout regularization [114] in the innermost layers, in order to improve training performance. Our network contains the following layers:

1. 4×4 convolution, stride 2, 64 kernels, leaky ReLU(slope= 0.2)
2. 4×4 convolution, stride 2, 128 kernels, instance norm., leaky ReLU(slope= 0.2)
3. 4×4 convolution, stride 2, 256 kernels, instance norm., leaky ReLU(slope= 0.2)
4. 4×4 convolution, stride 2, 512 kernels, instance norm., leaky ReLU(slope= 0.2), 50% dropout
5. 4×4 convolution, stride 2, 512 kernels, instance norm., leaky ReLU(slope= 0.2), 50% dropout
6. 4×4 convolution, stride 2, 512 kernels, leaky ReLU(slope= 0.2), 50% dropout
7. 4×4 transposed convolution, stride 2, 512 kernels, instance norm., ReLU, 50% dropout applied to concatenated output of layers 5 and 6.
8. 4×4 transposed convolution, stride 2, 512 kernels, instance norm., ReLU, 50% dropout applied to concatenated output of layers 4 and 7.
9. 4×4 transposed convolution, stride 2, 256 kernels, instance norm., ReLU applied to concatenated output of layers 3 and 8.
10. 4×4 transposed convolution, stride 2, 128 kernels, instance norm., ReLU applied to concatenated output of layers 2 and 9.

11. 4×4 transposed convolution, stride 2, 64 kernels, instance norm., ReLU

    applied to concatenated output of layers 1 and 10.

12. 2x upsampling

13. 4×4 convolution with bias, stride 1 with zero padding, 1 kernel, Tanh

5.2.4.2 Discriminator architecture

Our discriminator uses the PatchGAN architecture proposed by Isola *et al* [17], which is itself based on the discriminator in the Markovian GAN architecture [115]. This architecture classifies individual patches of the input image as real or fake, instead of classifying the image as a whole. This effectively restricts the scope of the adversarial loss function to high-frequency features in small patches of the image, while relying on a separate ℓ_1 loss for low-frequency features. The network layers are as follows:

1. 4×4 convolution with bias, stride 2, 512 kernels, leaky ReLU(slope= 0.2)
2. 4×4 convolution with bias, stride 2, 1024 kernels, instance norm., leaky ReLU(slope= 0.2)
3. 4×4 convolution with bias, stride 2, 2048 kernels, instance norm., leaky ReLU(slope= 0.2)
4. 4×4 convolution, stride 1 with zero padding, 1 kernel, linear activation

We use a patch size of 8×8 for our heightmap domain translation experiments. For scalar field domain translation (Section 5.3), we obtained better results with 16×16 patches, therefore layer 3 is omitted.

5.2.4.3 Training hyperparameters

We have trained our GANs using the Adam [116] optimization algorithm. The following hyperparameters are identical for both the generator and the discriminator:

batch size = 16

learning rate = 3×10^{-4}

momentum: $\beta_1 = 0.5, \beta_2 = 0.999$

The discriminator loss is the mean squared error of classifying each patch as real or fake. The generator has to maximize the discriminator loss, while also minimizing the ℓ_1 error of the predicted image. The ℓ_1 error is given a weight of 10 for heightmap domain translation, and 0.5 in the case of scalar field domain translation.

5.3 Super-resolution of the sharpness field

Our method also has the potential for using the conditional GAN to predict values of a scalar or vector field corresponding to the dense heightmap at a given point. As an example, we show an application to the *sharpness field* defined in Chapter 3. The feature-aware smoothing method described in Chapter 3 is mainly concerned with the ridges of the field, which are scale-invariant. This makes it simple to apply our GAN to locally predict the value of the sharpness field of a point cloud, as shown in Figure 61 (page 111). If values of the sharpness field are pre-computed for the low-resolution point cloud, we can obtain a sparse image of the sharpness values simultaneously with the sparse heightmap (top row of Figure 61). By obtaining ground truth sharpness fields on meshes, we can then train a separate conditional GAN to predict the values of the sharpness field corresponding to all points of the dense heightmap (middle row of Figure 61).

In order to predict values of the sharpness field at the output points from our super-resolution method, we trained a separate GAN to predict sharpness field values at the output points, using the same training procedure and data augmentation as the heightmap super-resolution GAN. The only difference is that we obtained better results with a larger patch size of 16×16 for the discriminator. The training data is a set of meshes of simple geometric shapes, whose sharpness fields are computed using dihedral angles as described in Section 3.1.3.1. We then pre-computed the sharpness field of the *blade* point cloud (80K points) using the CNN with a spatial transformer, which was the best performing method across different scales in Section 4.2.4. We then perform 16x super-resolution using the *even* mode.

The results are presented in Section 6.3 (page 124).

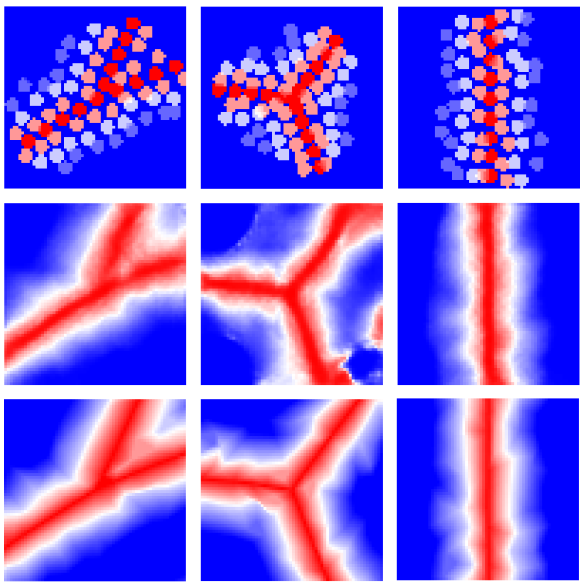

Figure 61: Sharpness field sampled from the *fandisk* model. Top row: sparse sampling of the sharpness field precomputed on the point cloud. Middle row: dense sharpness field predicted by our GAN. Bottom row: ground truth sharpness field obtained by casting rays onto the original mesh.

In this chapter, we have described a new point cloud super-resolution method based on domain translation of point cloud neighborhoods. In the following chapter, we present our super-resolution results, along with comparisons to state-of-the-art methods.

Chapter 6

Results and Discussion: Point Cloud Super-Resolution

In this chapter, we show our super-resolution results obtained using the method described in Section 5.2. These results are compared with the state-of-the-art methods EC-Net [5] and 3PU [6].

We refer the reader back to Section 2.2.4 (page 22) for a description of the state-of-the-art approaches to point cloud super-resolution. These methods are all based on deep learning but do not incorporate domain translation or heightmaps. Reviewing these methods is important for understanding how our method differs from the state of the art, and introduces the methods that we compare with in this chapter. Section 6.1.1 (page 121) also explains the disadvantages of these methods, resulting from the fact that they are based on PointNet++.

Section 6.1 gives a qualitative comparison of the results, while Section 6.2 has a quantitative comparison using standard metrics. Finally, Section 6.3 shows a simple example of sharpness field super-resolution, as described in Section 5.3.

The results in this chapter have been accepted for publication [12].

6.1 Comparison with recent work

Examples of our super-resolution results are shown in Figures 62 to 64. The meshes are reconstructed using screened Poisson surface reconstruction [53]. Note that there were no post-processing operations such as smoothing in any of our figures. In Figures 62 and 63, we can see that our

method is able to reconstruct fine details such as facial features from a sample of only 5000 points. Figure 64 shows an extreme example where we upsample a set of only 625 points, while still being able to reconstruct features such as the wings on the helmet. For comparison, we show the corresponding result using the method described in a concurrent work, informally called 3PU [6] (code is provided by the authors). This method has already been shown to be superior to state-of-the-art approaches such as EC-Net [5] and PU-Net [4]. All three of these methods are ultimately based on PointNet++ [3], therefore they do not make use of local heightmaps. Additional results are given in Figures 65 and 66.

During our experiments on extremely low resolution point clouds, we naturally found that different random samplings of 625 input points for the same testing mesh give slightly different resulting meshes. We have performed an experiment to assess the variation in results for different random samplings, whose results are shown in Figure 67. We find that our method produces consistent results across different random samplings. We can see from the same figure that 3PU produces less consistent results. Figure 67 omits examples where PCA failed to give good enough normals for the 3PU point cloud, resulting in failure of the screened Poisson reconstruction. Our method provides normals for output points, so no additional normal estimation step is required. Furthermore, our normals have never caused screened Poisson reconstruction to fail in our experiments.

Figure 68 shows additional 16x upsampling results on point clouds obtained from depth cameras. This includes both our own depth scans obtained using Kinect v2, as well as a point cloud of a kitchen obtained from the ScanNet [11] dataset. These results show that our GAN trained on statues can generalize to some extent to upsampling point clouds from depth cameras. However, it tends to treat signficant amounts of noise in depth images as features to be upsampled.

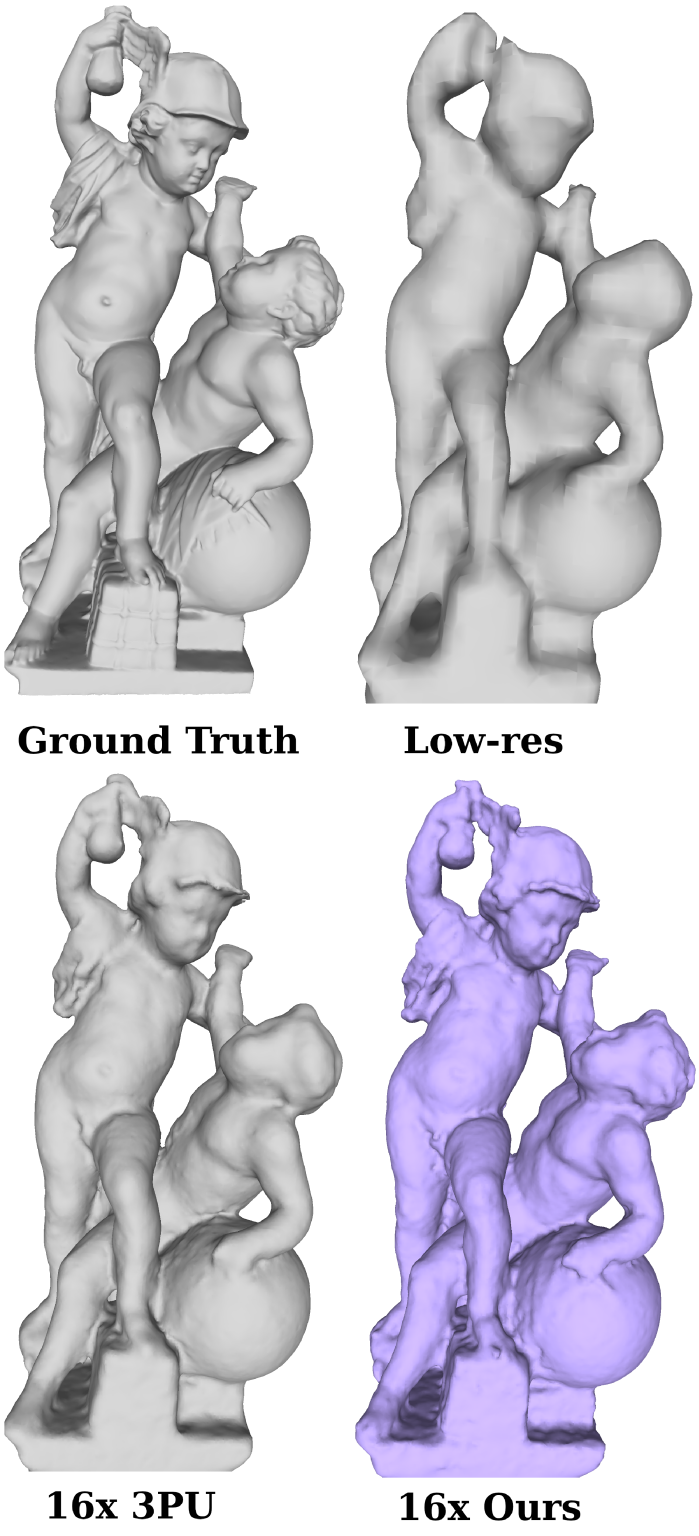

Figure 62: Results for the statue *Cupid Fighting*, reconstructed from a sample of 5000 points from the ground truth mesh.

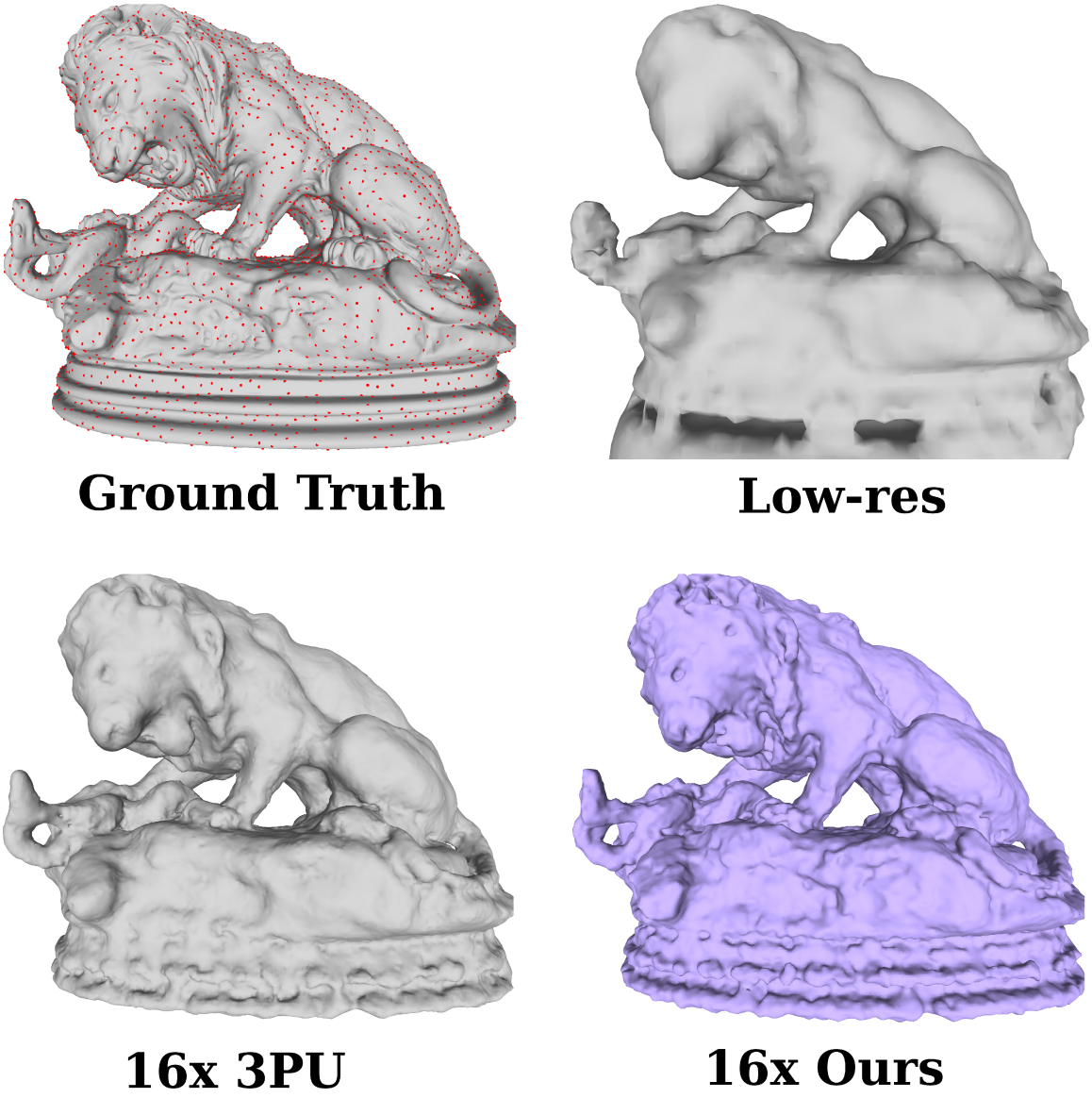

Figure 63: Results for the statue *Lion Étouffant Un Serpent*. The 5000 sampled points are overlaid on the ground truth mesh.

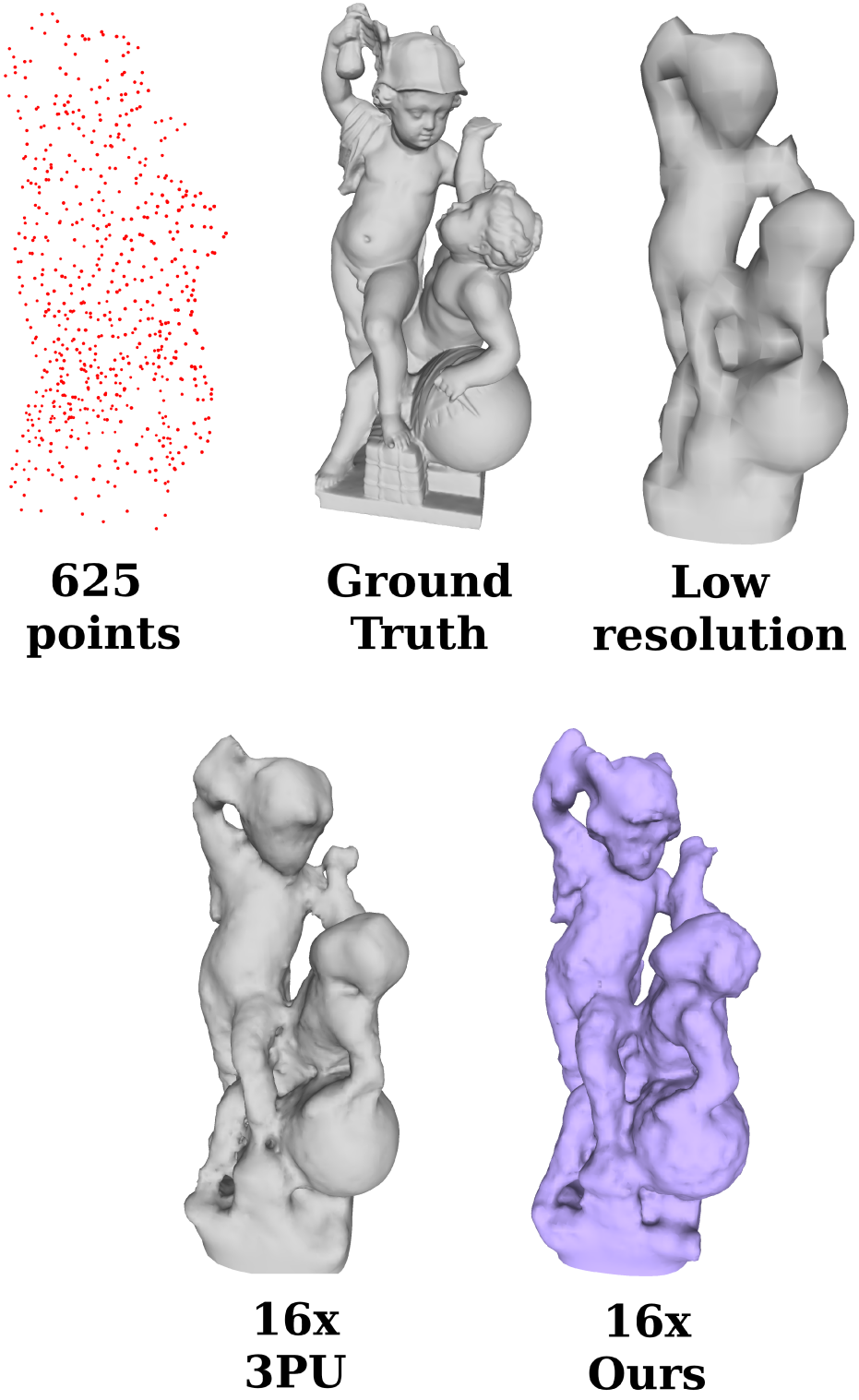

Figure 64: Results for an extremely low-resolution sample of 625 points (in red) from *Cupid Fighting*. We compare against the best results we were able to obtain for 3PU, across multiple random samplings of 625 points.

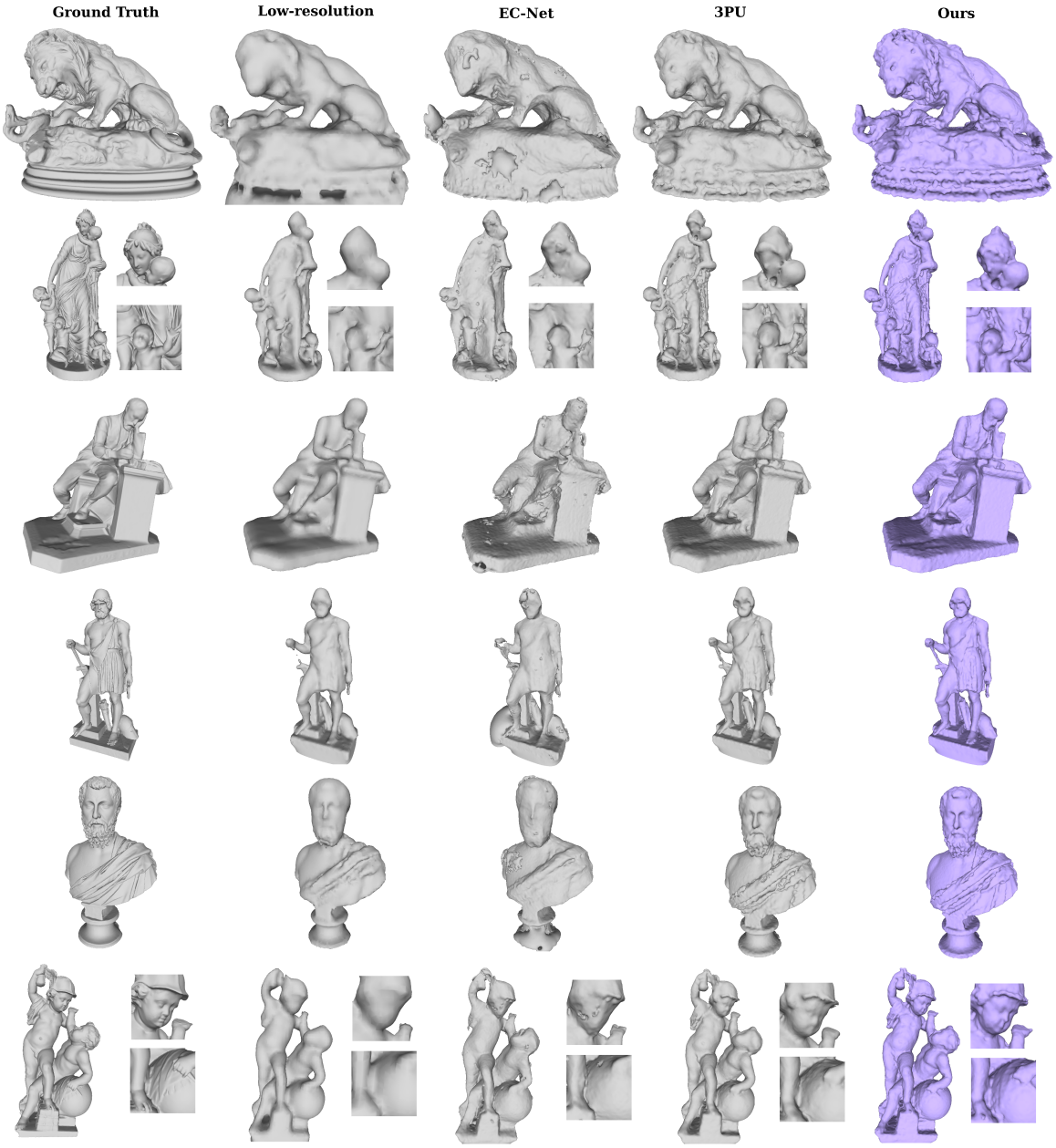

Figure 65: Additional results for 16x super-resolution using our *smooth* mode, compared with 3PU and EC-Net.

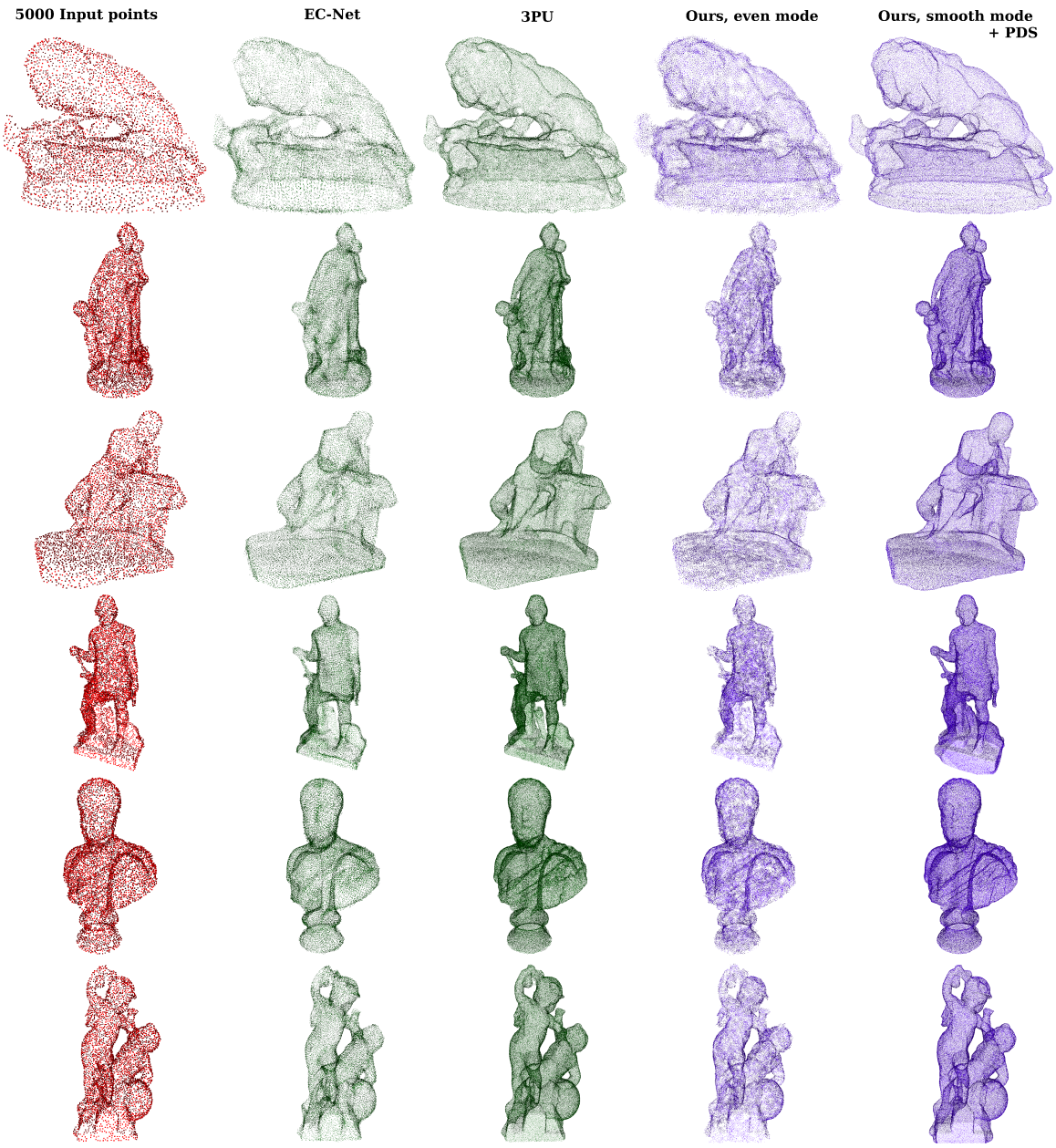

Figure 66: Obtaining point clouds using our *even* mode, compared with others. The last column shows results from Poisson disk sampling of our reconstructed mesh, using the *smooth* mode.

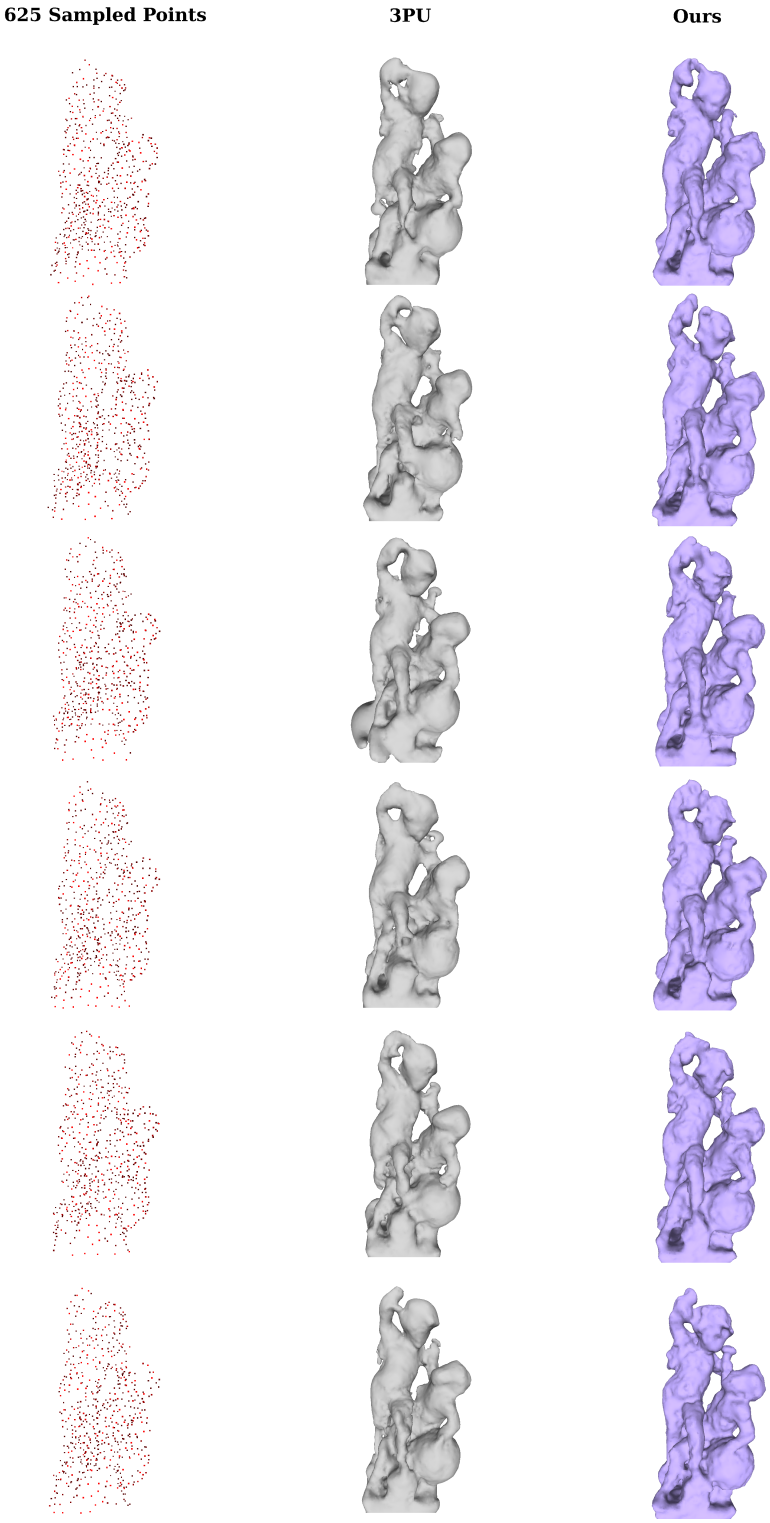

Figure 67: Comparison of results obtained from multiple random samplings of 625 points of the *Cupid Fighting* model.

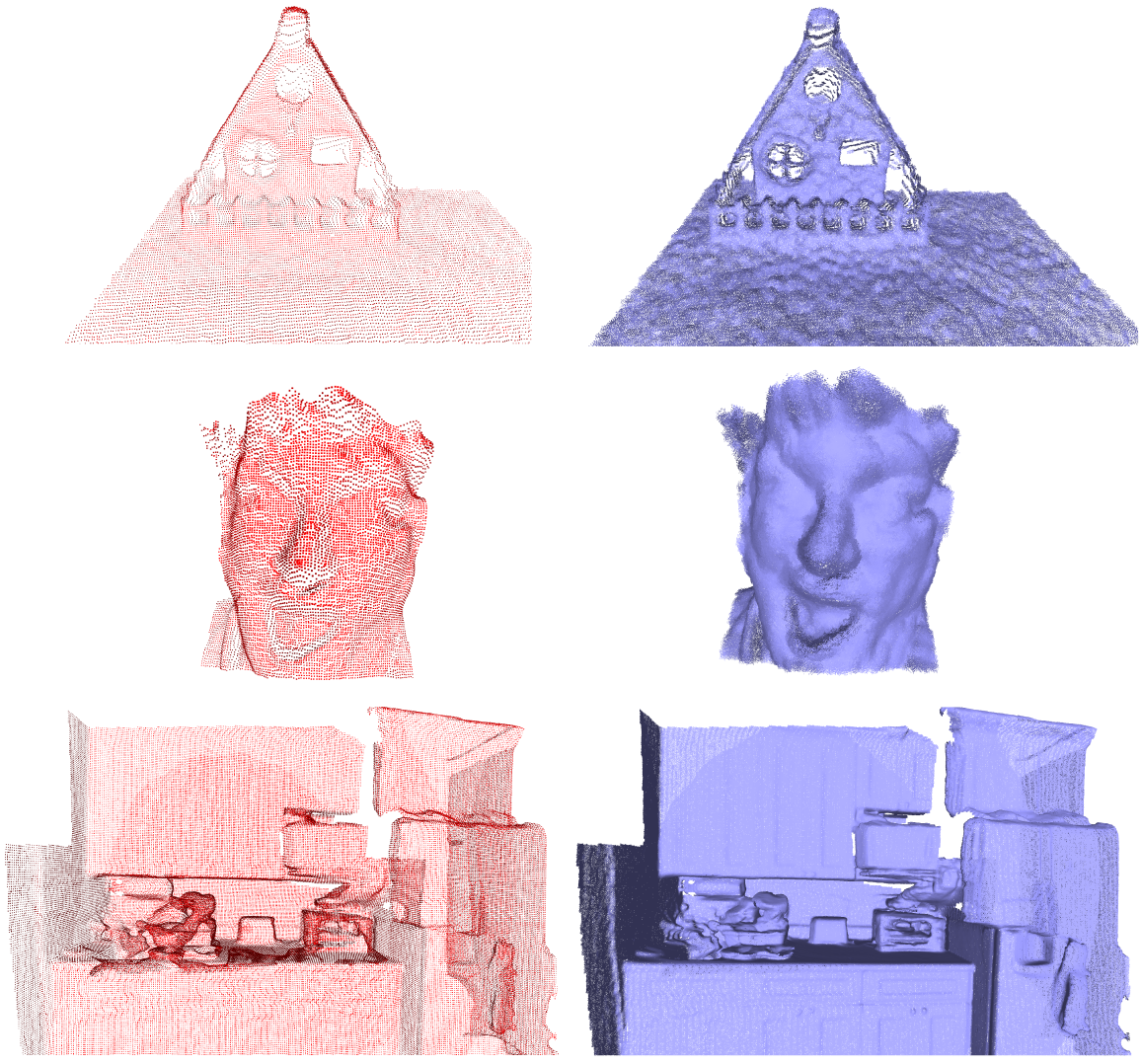

Figure 68: Top & middle rows: Upsampling of point clouds scanned by us using Kinect v2. Bottom row: Upsampling of a kitchen obtained from ScanNet [11]. Results are shown as points.

6.1.1 Disadvantages of PointNet++-based super-resolution

It is also worth noting that all super-resolution methods based on PointNet++ share certain weaknesses:

1. They are not invariant to permutations of the point cloud. This is because they all rely on farthest-point sampling as an initial step for their neural networks [117].

2. They rely on large and complicated neural network architectures, which affects the computational efficiency. For example, the ScanNet example in Figure 68 took only 3 minutes to upsample with our method, as opposed to 214 minutes for 3PU.

3. They do not perform well in regions with unusually dense sampling. This is because after the farthest-point sampling step, K-nearest neighbors are taken as representatives of the local neighborhood. Therefore, regions of high density will result in too many neighbors that are very close to the center point, thus giving a skewed picture of the local neighborhood.

6.2 Quantitative evaluation

We evaluate our super-resolution results quantitatively based on three metrics:

1. Distance of each point in the high-resolution point cloud to the ground truth mesh (D2M).

2. Hausdorff distance (HD): the maximum distance between a point on the reconstructed mesh and its nearest neighbor on the ground truth mesh:

$$\max\left(\max_{p\in P}\min_{q\in Q}\|p-q\|^2, \max_{q\in Q}\min_{p\in P}\|p-q\|^2\right) \tag{12}$$

3. Chamfer distance (CD) [103]: the mean distance between a point on the reconstructed mesh and its nearest neighbor on the ground truth mesh:

$$\frac{1}{2}\left(\frac{1}{|P|}\sum_{p\in P}\min_{q\in Q}\|p-q\|^2 + \frac{1}{|Q|}\sum_{q\in Q}\min_{p\in P}\|p-q\|^2\right) \tag{13}$$

We used Meshlab [67] to automate computation of all three metrics. The Hausdorff distance and Chamfer distance can be computed simultaneously by first randomly sampling a large number of points from the vertices, edges and faces of one mesh. The number of points sampled is equal to the

Mode	D2M	HD	CD
Smooth – 5000 points	**3.42E-04**	**3.24E-02**	**1.03E-03**
Even – 5000 points	3.96E-04	3.48E-02	1.42E-03
Superdense – 5000 points	5.89E-04	3.35E-02	1.37E-03
Smooth – 2500 points	**4.00E-04**	**3.66E-02**	**1.53E-03**
Even – 2500 points	4.38E-04	4.08E-02	2.11E-03
Superdense – 2500 points	8.39E-04	4.00E-02	2.25E-03
Smooth – 625 points	1.16E-03	5.20E-02	**3.86E-03**
Even – 625 points	**1.10E-03**	**5.03E-02**	5.25E-03
Superdense – 625 points	2.93E-03	5.81E-02	6.23E-03

Table 6: Quantitative comparison of our three modes for super-resolution of input point clouds with 5000, 2500 and 625 points. Note that the *superdense* mode produces 256 times the number of points.

Size of point clouds	D2M	HD	CD
>300K: 5000 points	**3.42E-04**	**3.24E-02**	**1.03E-03**
5K: 5000 points	8.04E-04	3.26E-02	1.30E-03
625: 5000 points	1.17E-03	3.38E-02	1.54E-03
>300K: 2500 points	**4.00E-04**	**3.66E-02**	**1.53E-03**
5K: 2500 points	9.63E-04	3.72E-02	1.90E-03
625: 2500 points	1.46E-03	3.96E-02	2.17E-03
>300K: 625 points	1.16E-03	5.20E-02	**3.86E-03**
5K: 625 points	**1.09E-03**	**5.04E-02**	4.32E-03
625: 625 points	1.97E-03	5.54E-02	4.58E-03

Table 7: Comparison of using different point cloud sizes when training: all mesh vertices (over 300K points), 5000 points sampled using Poisson disk sampling, or 625 points. The resulting metrics are compared for input point clouds with 5000, 2500 and 625 points.

number of vertices on the sampled mesh (several hundred thousand). We then find the mean and maximum distances between these points and the nearest vertices on the other mesh. Pairs of points are discarded if their separation is greater than 5% of the diagonal of the bounding box. The above procedure is then repeated in the opposite direction, giving us two maximums and two means. The Hausdorff distance is the maximum of the two maximums and the Chamfer distance is the mean of the two means (this differs slightly from the definition of Chamfer distance given by Fan *et al* [103], who take the sum of the means, since they are mainly interested in minimizing Chamfer distance as a loss function). Note that for performing screened Poisson reconstruction on the results of other methods, we estimate normals using PCA based on 30 nearest neighbors. This is not necessary for our method, since we obtain normals from the dense heightmap as mentioned in Section 5.2.2.

In Table 6, we compare the results for the three modes for sampling our dense heightmaps, as defined in Section 5.2.3. We first consider the *smooth* and *even* modes, which both produce 16 times the number of input points. Although the *smooth* mode produces the best reconstructed mesh,

Method	D2M	HD	CD
EC-Net – 5000 points	3.42E-04	6.30E-02	3.89E-03
3PU – 5000 points	**2.91E-04**	3.63E-02	1.32E-03
Ours – 5000 points	3.42E-04	**3.24E-02**	**1.03E-03**
EC-Net – 2500 points	6.56E-04	6.57E-02	5.80E-03
3PU – 2500 points	**3.16E-04**	4.86E-02	2.13E-03
Ours – 2500 points	4.00E-04	**3.66E-02**	**1.53E-03**
EC-Net – 625 points	1.85E-03	5.84E-02	8.12E-03
3PU – 625 points	1.31E-03	5.55E-02	4.96E-03
Ours – 625 points	**1.16E-03**	**5.20E-02**	**3.86E-03**

Table 8: Quantitative comparison of 16x super-resolution between EC-Net, 3PU and our *smooth* mode for point clouds with 5000, 2500 and 625 points.

it is noteworthy that the *even* mode is not far behind, and even surpasses the *smooth* mode for very low resolution point clouds. Those seeking to apply our method to obtain a point cloud, and not a mesh, have a choice between using our *even* mode, or alternatively performing Poisson disk sampling on the mesh produced using our *smooth* mode. The *superdense* mode, which produces 256 times the number of input points, suffers a bit when it comes to the distance to the ground truth mesh. However, it is not far behind the other modes when it comes to the Hausdorff and Chamfer distances.

A unique feature of our method is that we can obtain good results on low resolution point clouds, even after training on high-resolution point clouds. Our training meshes are very dense, and each has over 300 thousand vertices. They can also be randomly downsampled to a lower resolution such as 5000 or 625 points during training. We therefore investigate how different resolutions of the training point clouds affect the super-resolution results. After all, the resolution of the training point clouds does affect the variety of sparse heightmaps that will be seen during training. Table 7 shows that using the entire set of mesh vertices during training usually produces the best results. Even in the case where the testing point clouds have 625 points, it is a GAN trained on a higher resolution (5000 points) which gives the best results. This is likely due to a combination of two factors: i) we use raycasting to obtain our ground truth heightmaps, and ii) we randomly sample only 100 points from each local neigborhood to contribute to the sparse heightmap, thereby making our domain translation more robust.

Table 8 compares the surfaces reconstructed by applying screened Poisson reconstruction on the results of our domain translation approach, versus other recent methods. We choose the *smooth* mode for comparison, since it produces 16x super-resolution, and it produces the most accurate

reconstructed meshes. We compare our work with the results obtained using EC-Net [5], which is the most recently published point cloud upsampling work. We also compare with a concurrent work by Wang *et al* [6] (code released under the name "3PU"), which claims better results than previous work for 16x sampling. In order to compare results for 16x upsampling for EC-Net, we apply 4x upsampling with EC-Net twice in succession, as recommended by Yu *et al* in their correspondence with Wang *et al* [6]. We can see that our results are clearly superior to those of EC-Net. Even when compared to the very recent 3PU method, our results are superior in terms of Hausdorff distance and Chamfer distance, while still being competitive in terms of the D2M metric.

6.3 Super-resolution of the sharpness field

The results we obtained (Figure 69) show that our GAN gives a sharpness field with similar properties to a sharpness field computed from scratch on the point cloud after super-resolution. Furthermore, the entire super-resolution procedure along with the sharpness field estimation takes around 7 minutes, compared to 20 minutes if the sharpness field is recomputed from scratch after super-resolution. The combination of the dense heightmap, the normal map and the sharpness field together provide all the information necessary for downstream methods to accurately reconstruct the surface along with sharp features. We have included this example for its curiosity value. We believe that our domain translation approach to super-resolution has the potential for extension to other scalar fields and vector fields.

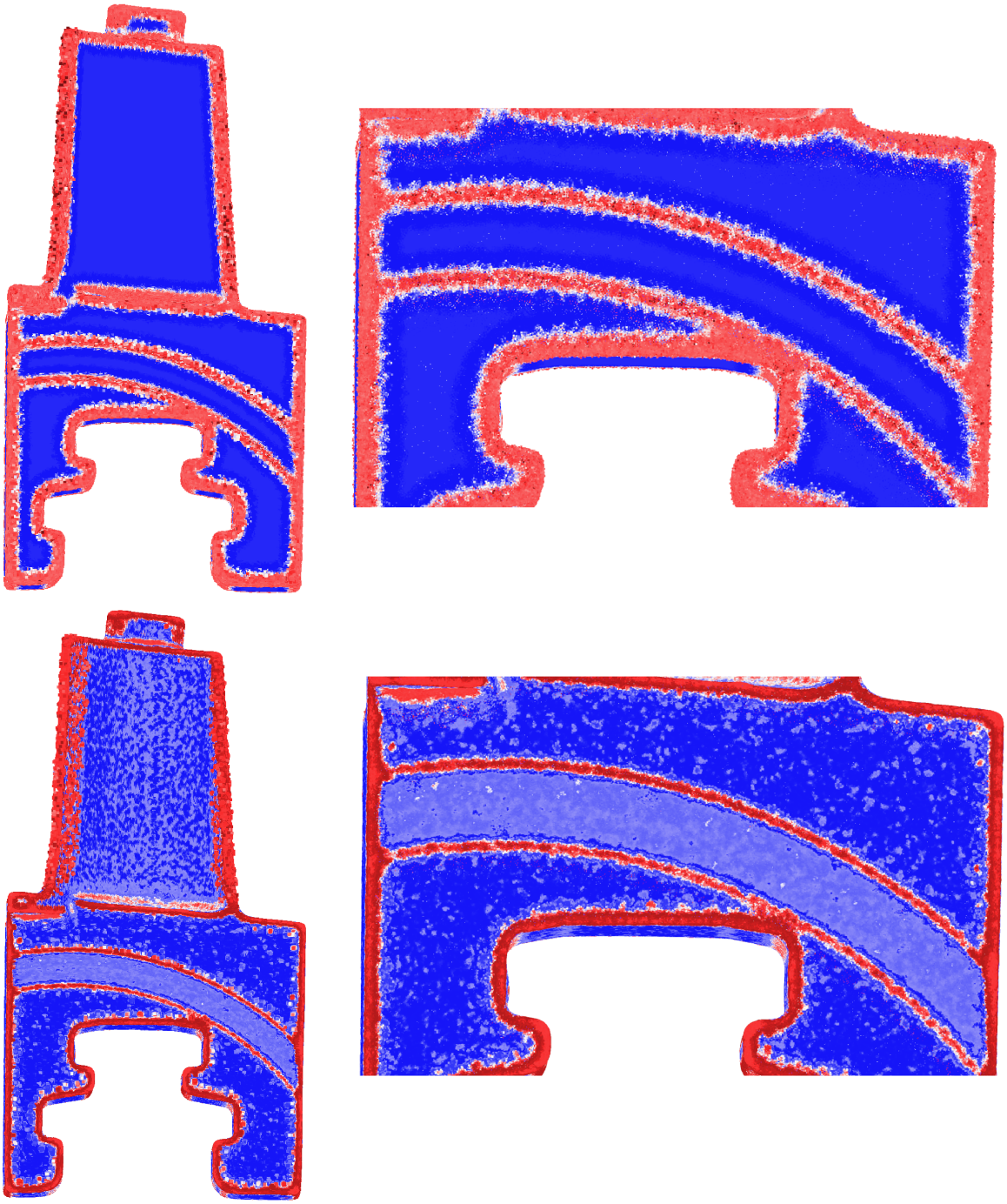

Figure 69: Super-resolution results for the *blade* model, along with its sharpness field. Top: using our GAN to estimate the sharpness field. Bottom: recomputing the sharpness field from scratch on the high-resolution point cloud, using the method from Section 3.2.2.

Chapter 7

Conclusion

This thesis is a compilation of our research on reconstructing fine features in 3D point clouds using deep learning. As previously mentioned, "fine features" refers to features whose scale is comparable to or smaller than the separation between points of the point cloud. For example, sharp edges can be considered as fine features, since they can never be captured by scanning devices that produce point clouds.

We have introduced the novel concept of a *sharpness field*, defined on the underlying surface of a point cloud, which enables us to localize sharp edges even in the gaps between points in the point cloud. The sharpness field can easily be computed on polygonal meshes using dihedral angles. This allows us to design ground truth data for training machine learning models to compute sharpness field values in point clouds. We have built a training dataset consisting of over 100K points, whose neighborhoods contain different levels of noise. These points were sampled from a carefully designed set of 8 shapes, which contain a sufficiently large variety of edges and corners for learning sharpness field computation. We also designed a representation for point cloud neighborhoods which enables feeding these neighborhoods to machine learning models. This representation consists of the angular deviations of normals sampled on a 30×30 geodesic radial grid centered at the input point. We have found that this machine learning approach allows us to compute a sharpness field using several different machine learning models, including gradient boosting machines, random forests and CNNs. Our evaluation in Section 4.1.2 shows that CNNs performed the best out of the evaluated machine learning models. The computed sharpness field is observed to be smooth and accurat for complex shapes not seen in the training data. However, certain curved edges, such as those in the *blade* model, require more work, going beyond the methods explored in Section 3.1.

In order to fully capture these curved edges, we systematically rolled back the processing steps involved in generating the local neighborhood representation, in order to preserve progressively greater amounts of geometric information in the neighborhood representation. These neighborhood representations are more "raw" than our original neighborhood representation, while also possessing important spatial relations within them. We therefore need to feed them to deep learning models, such as CNNs, harmonic networks, spatial transformer networks and PointNets. After extensive evaluation (Section 4.2.4), we have shown that curved edges can be captured reliably by feeding Cartesian heightmaps to a CNN with an added spatial transformer network.

This sharpness field can be used to denoise point clouds while preserving sharp edges, using our feature-aware smoothing algorithms. The most effective feature-aware smoothing algorithm that we have developed uses an anisotropic Gaussian kernel in a variant of MLS projection (Section 3.3.2). The shape of the kernel is adjusted so that it does not cross sharp edges (local maxima or ridges of the sharpness field). We have seen that point clouds denoised with our feature-aware method are as smooth as those denoised using the feature-agnostic RIMLS method [16], while having sharper edges. We have also presented algorithms that use the sharpness field to generate an explicit graphical representation of sharp edges (Section 3.3.4), as well as to segment a point cloud into smooth patches (Section 3.3.3).

Besides sharp edge reconstruction, our research also extends to general fine feature reconstruction in point clouds. This has been accomplished by our unique method for point cloud super-resolution that uses conditional generative adversarial networks (GANs) for domain translation of local neighborhoods. This is an innovative application of GAN-based image translation, and is a rare instance where image translation is applied with the aim of obtaining quantitatively accurate results. This is in stark contrast to typical image translation applications, such as translating between photos, sketches, satellite images and street maps. In our work, heightmaps sampled from low-resolution point clouds are "translated" into dense heightmaps sampled from triangle meshes. Pixels from the dense heightmap can then be transformed back into 3D points, giving a point cloud which is denser by a certain factor (e.g. 16 times or 256 times). This dense point cloud contains important fine details that greatly improve the accuracy of the surface reconstructed using screened Poisson surface reconstruction. This has allowed us to obtain super-resolution results far superior to the state-of-the-art method published in late 2018 [5]. Even when compared to the concurrently developed 3PU method [6], our super-resolution results can be seen to reconstruct more fine details from low-resolution point clouds. The superiority of our method is particularly noticeable for very sparse

point clouds with just 625 points. We have also shown that our super-resolution method produces more consistent results than 3PU, as our results do not vary as much for different random samplings of points from the same shape. Furthermore, our method is far more computationally efficient for large point clouds than 3PU, and is not subject to the inherent weaknesses of methods based on PointNet++ (Section 6.1.1). Another unique feature of this super-resolution method is that it can easily be modified to upsample a precomputed scalar field, such as the sharpness field. In the case of the sharpness field, this is more computationally efficient than recomputing the sharpness field from scratch for the upsampled point cloud.

In order to extend our current work, we would require a larger dataset of CAD models. This would provide two advantages: (1) ground truth data for large-scale quantitative evaluation of sharpness field computaion methods. (2) large amounts of training data for training deeper neural network models. These models should ideally be represented as parametric surfaces in which the borders between surface patches are sharp edges. Using parametric surfaces instead of triangular meshes for training data would enable dynamically resampling points for training. It would also remove any ambiguity in the ground truth locations of sharp edges stemming from dihedral angles. There is also more scope for involving deep learning in our feature-aware smoothing algorithm. In future work on point cloud super-resolution, we hope to expand our method for scalar field super-resolution to support scale-dependent scalar fields such as mean curvature, as well as vector fields. We also envisage training the super-resolution GAN to compute these fields directly, instead of using a precomputed field.

Bibliography

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, 2017.

[2] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra, "PCPNet: Learning local shape properties from raw point clouds," in *Computer Graphics Forum*, vol. 37, pp. 75–85, 2018.

[3] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5099–5108, 2017.

[4] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "PU-Net: Point cloud upsampling network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2790–2799, June 2018.

[5] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "EC-Net: an edge-aware point set consolidation network," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 386–402, September 2018.

[6] Y. Wang, S. Wu, H. Huang, D. Cohen-Or, and O. Sorkine-Hornung, "Patch-based progressive 3D point set upsampling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5958–5967, 2019.

[7] P. Raina, S. Mudur, and T. Popa, "MLS^2: Sharpness field extraction using CNN for surface reconstruction," in *Proceedings of Graphics Interface 2018*, GI 2018, pp. 66–75, Canadian Human-Computer Communications Society, 2018.

[8] C. Weber, S. Hahmann, and H. Hagen, "Sharp feature detection in point clouds," in *Shape Modeling International Conference (SMI), 2010*, pp. 175–186, IEEE, 2010.

[9] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, "Harmonic networks: Deep translation and rotation equivariance," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2017.

[10] P. Raina, S. Mudur, and T. Popa, "Sharpness fields in point clouds using deep learning," *Computers & Graphics*, vol. 78, pp. 37–53, 2019.

[11] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D reconstructions of indoor scenes," in *Proc. CVPR 2017*, pp. 5828–5839.

[12] P. Raina, T. Popa, and S. Mudur, "Fine feature reconstruction in point clouds by adversarial domain translation," in *Proceedings of Graphics Interface 2020*, GI 2020, p. To appear, Canadian Human-Computer Communications Society, 2020.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[15] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," in *Advances in neural information processing systems*, pp. 2017–2025, 2015.

[16] A. C. Öztireli, G. Guennebaud, and M. Gross, "Feature preserving point set surfaces based on non-linear kernel regression," in *Computer Graphics Forum*, vol. 28, 2009.

[17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1125–1134, 2017.

[18] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *International Conference on Learning Representations*, 2019.

[19] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[20] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9446–9454, 2018.

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, vol. 9, pp. 249–256, 2010.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[23] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," in *Proc. International Conference on Learning Representations (ICLR), Workshop track*, 2018.

[24] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *International Conference on Learning Representations (ICLR), workshop track*, 2015.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[26] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[27] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, pp. 2223–2232, 2017.

[28] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "StarGAN: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8789–8797, June 2018.

[29] Z. Yang, Z. Hu, C. Dyer, E. P. Xing, and T. Berg-Kirkpatrick, "Unsupervised text style transfer using language models as discriminators," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 7298–7309, December 2018.

[30] S. Huang, Q. Li, C. Anil, X. Bao, S. Oore, and R. B. Grosse, "TimbreTron: A WaveNet (CycleGAN (CQT (audio))) pipeline for musical timbre transfer," in *International Conference on Learning Representations (ICLR)*, p. To appear., May 2019.

[31] N. Dehmamy, L. Stornaiuolo, and M. Martino, "Vox2Net: From 3D shapes to network sculptures," in *NeurIPS Workshop on Machine Learning for Creativity and Design*, December 2018.

[32] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.

[33] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view CNNs for object classification on 3D data," in *Proc. Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2016.

[34] 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 *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2017.

[35] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. IEEE ICCV*, pp. 945–953, 2015.

[36] H. Huang, E. Kalogerakis, S. Chaudhuri, D. Ceylan, V. G. Kim, and E. Yumer, "Learning local shape descriptors from part correspondences with multiview convolutional networks," *ACM Trans. Graph.*, vol. 37, pp. 6:1–6:14, Nov. 2017.

[37] Y. Liao, S. Donné, and A. Geiger, "Deep marching cubes: Learning explicit surface representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2916–2925, 2018.

[38] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7652–7660, 2018.

[39] R. Roveri, A. C. Öztireli, I. Pandele, and M. Gross, "PointProNets: Consolidation of point clouds with convolutional neural networks," in *Computer Graphics Forum*, vol. 37, pp. 87–99, Wiley Online Library, 2018.

[40] R. Roveri, L. Rahmann, A. C. Oztireli, and M. Gross, "A network architecture for point cloud classification via automatic depth images generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4176–4184, 2018.

[41] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE international conference on computer vision workshops*, pp. 37–45, 2015.

[42] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 3189–3197, 2016.

[43] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. CVPR*, vol. 1, p. 3, 2017.

[44] N. Verma, E. Boyer, and J. Verbeek, "FeaStNet: Feature-steered graph convolutions for 3D shape analysis," in *CVPR 2018-IEEE Conference on Computer Vision & Pattern Recognition*, 2018.

[45] O. Litany, A. Bronstein, M. Bronstein, and A. Makadia, "Deformable shape completion with graph convolutional autoencoders," in *CVPR 2018-IEEE Conference on Computer Vision & Pattern Recognition*, 2018.

[46] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "A papier-mâché approach to learning 3D surface generation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 216–224, June 2018.

[47] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas, "Learning representations and generative models for 3D point clouds," in *Proc. International Conference on Learning Representations (ICLR)*, IEEE, 2018.

[48] Y. Yang, C. Feng, Y. Shen, and D. Tian, "FoldingNet: Point cloud auto-encoder via deep grid deformation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 206–215, June 2018.

[49] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, J. Levine, A. Sharf, and C. Silva, "State of the art in surface reconstruction from point clouds," in *Eurographics STAR reports*, vol. 1, pp. 161–185, 2014.

[50] F. Cazals and J. Giesen, "Delaunay triangulation based surface reconstruction," in *Effective computational geometry for curves and surfaces*, pp. 231–276, Springer, 2006.

[51] 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*, pp. 67–76, ACM, 2001.

[52] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *SGP*, vol. 7, 2006.

[53] M. Kazhdan and H. Hoppe, "Screened poisson surface reconstruction," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 3, p. 29, 2013.

[54] R. Campos, R. Garcia, and T. Nicosevici, "Surface reconstruction methods for the recovery of 3D models from underwater interest areas," in *OCEANS 2011 IEEE - Spain*, pp. 1–10, June 2011.

[55] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Point set surfaces," in *Proceedings of the Conference on Visualization'01*, pp. 21–28, IEEE Computer Society, 2001.

[56] D. Levin, "The approximation power of moving least-squares," *Mathematics of Computation of the American Mathematical Society*, vol. 67, no. 224, pp. 1517–1531, 1998.

[57] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Computing and rendering point set surfaces," *IEEE TVCG*, vol. 9, no. 1, pp. 3–15, 2003.

[58] M. Alexa and A. Adamson, "On normals and projection operators for surfaces defined by point sets," in *Proceedings of the First Eurographics conference on Point-Based Graphics*, pp. 149–155, 2004.

[59] D. Levin, "Mesh-independent surface interpolation," in *Geometric modeling for scientific visualization*, pp. 37–49, Springer, 2004.

[60] R. Kolluri, "Provably good moving least squares," *ACM Transactions on Algorithms (TALG)*, vol. 4, no. 2, p. 18, 2008.

[61] Z.-Q. Cheng, Y.-Z. Wang, B. Li, K. Xu, G. Dang, and S.-Y. Jin, "A survey of methods for moving least squares surfaces.," in *Volume Graphics*, pp. 9–23, 2008.

[62] G. Guennebaud and M. Gross, "Algebraic point set surfaces," in *ACM TOG*, vol. 26, p. 23, ACM, 2007.

[63] Y. Lipman, D. Cohen-Or, and D. Levin, "Data-dependent MLS for faithful surface approximation," in *SGP '07*, (Aire-la-Ville, Switzerland), pp. 59–67, Eurographics Association, 2007.

[64] S. Fleishman, D. Cohen-Or, and C. T. Silva, "Robust moving least-squares fitting with sharp features," *ACM TOG*, vol. 24, no. 3, pp. 544–552, 2005.

[65] T. R. Jones, F. Durand, and M. Desbrun, "Non-iterative, feature-preserving mesh smoothing," in *ACM Transactions on Graphics (TOG)*, vol. 22, pp. 943–949, ACM, 2003.

[66] P. J. Huber, *Robust statistics*. Springer, 2011.

[67] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an open-source mesh processing tool," in *Eurographics Italian Chapter Conference* (V. Scarano, R. D. Chiara, and U. Erra, eds.), The Eurographics Association, 2008.

[68] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher, "Geometric surface smoothing via anisotropic diffusion of normals," in *Proceedings of the conference on Visualization'02*, pp. 125–132, IEEE Computer Society, 2002.

[69] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[70] S. Fleishman, I. Drori, and D. Cohen-Or, "Bilateral mesh denoising," in *ACM transactions on graphics (TOG)*, vol. 22, pp. 950–953, ACM, 2003.

[71] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *ICCV*, vol. 98, p. 2, 1998.

[72] X. Sun, P. L. Rosin, R. Martin, and F. Langbein, "Fast and effective feature-preserving mesh denoising," *IEEE transactions on visualization and computer graphics*, vol. 13, no. 5, pp. 925–938, 2007.

[73] Y. Zheng, H. Fu, O. K.-C. Au, and C.-L. Tai, "Bilateral normal filtering for mesh denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 10, pp. 1521–1530, 2010.

[74] L. He and S. Schaefer, "Mesh denoising via L_0 minimization," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 64, 2013.

[75] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l 0 gradient minimization," in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 174, ACM, 2011.

[76] U. Pinkall and K. Polthier, "Computing discrete minimal surfaces and their conjugates," *Experimental mathematics*, vol. 2, no. 1, pp. 15–36, 1993.

[77] S. Gumhold, X. Wang, and R. MacLeod, "Feature extraction from point clouds," in *Proceedings of 10th international meshing roundtable*, vol. 2001, 2001.

[78] M. Pauly, R. Keiser, and M. Gross, "Multi-scale feature extraction on point-sampled surfaces," in *CGF*, vol. 22, pp. 281–289, 2003.

[79] J. Cao, S. Wushour, X. Yao, N. Li, J. Liang, and X. Liang, "Sharp feature extraction in point clouds," *IET image processing*, vol. 6, no. 7, pp. 863–869, 2012.

[80] T.-T. Tran, V.-T. Cao, S. Ali, D. Laurendeau, *et al.*, "Automatic method for sharp feature extraction from 3D data of man-made objects," in *Computer Graphics Theory and Applications (GRAPP), 2014 International Conference on*, pp. 1–8, IEEE, 2014.

[81] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[82] J. Nie, "Extracting feature lines from point clouds based on smooth shrink and iterative thinning," *Graphical Models*, vol. 84, pp. 38–49, 2016.

[83] D. Bazazian, J. R. Casas, and J. Ruiz-Hidalgo, "Fast and robust edge extraction in unorganized point clouds," in *Digital Image Computing: Techniques and Applications (DICTA), 2015*, pp. 1–8, IEEE, 2015.

[84] S. Fu and L. Wu, "Feature line extraction from point clouds based on geometric structure of point space," *3D Research*, vol. 10, no. 2, p. 16, 2019.

[85] C. Weber, S. Hahmann, H. Hagen, and G.-P. Bonneau, "Sharp feature preserving MLS surface reconstruction based on local feature line approximations," *Graphical Models*, vol. 74, no. 6, pp. 335–345, 2012.

[86] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, April 2017.

[87] H. Narayanan and S. Mitter, "Sample complexity of testing the manifold hypothesis," in *Advances in Neural Information Processing Systems*, pp. 1786–1794, 2010.

[88] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár, "Amortised map inference for image super-resolution," in *International Conference on Learning Representations (ICLR)*, April 2017.

[89] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651, JMLR. org, 2017.

[90] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, "StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5907–5915, 2017.

[91] S. Reed, Z. Akata, H. Lee, and B. Schiele, "Learning deep representations of fine-grained visual descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 49–58, 2016.

[92] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv preprint arXiv:1701.07875*, 2017.

[93] H. Zhang, V. Sindagi, and V. M. Patel, "Image de-raining using a conditional generative adversarial network," *IEEE transactions on circuits and systems for video technology*, 2019.

[94] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[95] T.-W. Hui, C. C. Loy, and X. Tang, "Depth map super-resolution by deep multi-scale guidance," in *European conference on computer vision (ECCV)*, pp. 353–369, Springer, 2016.

[96] Y. Wen, B. Sheng, P. Li, W. Lin, and D. D. Feng, "Deep color guided coarse-to-fine convolutional network cascade for depth image super-resolution," *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 994–1006, 2019.

[97] H. Huang, D. Li, H. Zhang, U. Ascher, and D. Cohen-Or, "Consolidation of unorganized point clouds for surface reconstruction," *ACM transactions on graphics (TOG)*, vol. 28, no. 5, p. 176, 2009.

[98] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer, "Parameterization-free projection for geometry reconstruction," in *ACM Transactions on Graphics (TOG)*, vol. 26, p. 22, ACM, 2007.

[99] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, and H. R. Zhang, "Edge-aware point set resampling," *ACM transactions on graphics (TOG)*, vol. 32, no. 1, p. 9, 2013.

[100] S. Wu, H. Huang, M. Gong, M. Zwicker, and D. Cohen-Or, "Deep points consolidation," *ACM Transactions on Graphics (ToG)*, vol. 34, no. 6, p. 176, 2015.

[101] W. Zhang, H. Jiang, Z. Yang, S. Yamakawa, K. Shimada, and L. B. Kara, "Data-driven upsampling of point clouds," *Computer-Aided Design*, vol. 112, pp. 1–13, 2019.

[102] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[103] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 605–613, 2017.

[104] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "ShapeNet: An information-rich 3D model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[105] A. Boulch and R. Marlet, "Deep learning for robust normal estimation in unstructured point clouds," in *Computer Graphics Forum*, vol. 35, pp. 281–290, Wiley Online Library, 2016.

[106] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Surface reconstruction from unorganized points*, vol. 26. ACM, 1992.

[107] V. Surazhsky and C. Gotsman, "Explicit surface remeshing," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pp. 20–30, Eurographics Association, 2003.

[108] S. Torbert, *Applied computer science*. Springer, 2016.

[109] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, "Embree: A kernel framework for efficient CPU ray tracing," *ACM Trans. Graph.*, vol. 33, pp. 143:1–143:8, July 2014.

[110] "VCG library." `http://vcg.isti.cnr.it/vcglib/index.html`, 2004. Accessed: 2019-03-08.

[111] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[112] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015.

[113] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[114] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[115] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *European Conference on Computer Vision*, pp. 702–716, Springer, 2016.

[116] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[117] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "Modeling point clouds with self-attention and gumbel subset sampling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3323–3332, 2019.