UNIVERSITY OF CALIFORNIA SAN DIEGO

Neural Mesh Flow: 3D Manifold Mesh Generation via Diffeomorphic Flows

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science

in

Computer Science

by

Kunal Gupta

Committee in charge:

Professor Manmohan Chandraker, Chair
Professor David Kriegman
Professor Ravi Ramamoorthi

2020

The Thesis of Kunal Gupta is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2020

DEDICATION

To Lord Sri Krishna, the absolute truth and my teachers.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am grateful to Prof. Manmohan Chandraker, for his guidance during my Master's studies. His unique style of advising and ideation were especially helpful to me in developing confidence over my own research ideas and methodology. Without his unwavering support, this thesis would not have been possible.

I would also like to thank a number of professors at UC San Diego - Prof. Dinesh Bharadia, Prof. Francisco Contijoch, Prof. Falko Kuester and Prof. Nikolay Atanasov. I have been extremely fortunate to get the opportunity to work with them and learn from their experiences.

I owe special thanks to Krishna Murthy Jatavallabhula who has been an amazing mentor and a friend. It's hard to imagine getting through the uncertainties of research without his support. In the same spirit, I want to thank my collaborators Kshitiz Bansal, Brendan Colvert and other lab members who were always around to discuss research ideas and helped shape the course of my research.

I need to make a special mention of my MS Program advisors Wendy Yamamoto and Nadyne Nawar who helped me manage the stress and difficulties of adjusting to a new country and thrive in this rigorous program.

Most importantly, I would like to thank my parents and my sister, without whose sacrifices and immense emotional support, this work would not have been possible. Lastly, I would also like to thank my friends: Vamsidhar Reddy Kamanuru, Mayank Rajoria, Sai Hanisha Kilari, Pengcheng Cao, Zhaoliang Zheng, Ted Yu, Junchao Lin, Shreyam Natani, Shaurya Arya, who enriched my experience at UCSD both socially and academically.

This thesis, in full, has been submitted for publication of the material as it may appear in a conference, 2020, Kunal Gupta, Manmohan Chandraker. The thesis author was the primary investigator and author of this paper.

ABSTRACT OF THE THESIS

Neural Mesh Flow: 3D Manifold Mesh Generation via Diffeomorphic Flows

by

Kunal Gupta

Master of Science in Computer Science

University of California San Diego, 2020

Professor Manmohan Chandraker, Chair

Meshes are important representations of physical 3D entities in the virtual world. Applications like rendering, simulations and 3D printing require meshes to be *manifold* so that they can interact with the world like the real objects they represent. Prior methods generate meshes with great geometric accuracy but poor *manifoldness*. In this work we propose Neural Mesh Flow (NMF) to generate two-manifold meshes for genus-0 shapes. Specifically, NMF is a shape auto-encoder consisting of several Neural Ordinary Differential Equation (NODE)[5] blocks that learn accurate mesh geometry by progressively deforming a spherical mesh. Training NMF is simpler compared to state-of-the-art methods since it does not require any explicit mesh-based regularization. Our experiments demonstrate that NMF facilitates several applications such

as single-view mesh reconstruction, global shape parameterization, texture mapping, shape deformation and correspondence. Importantly, we demonstrate that manifold meshes generated using NMF are better-suited for physically-based rendering and simulation.

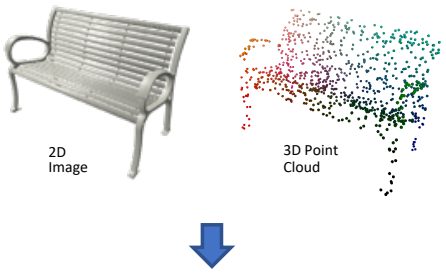
Chapter 1

Introduction

Mesh representations allow an efficient virtual representation of 3D objects, enabling applications in graphics rendering, simulations, modeling and manufacturing. Consequently, mesh generation or reconstruction from point clouds or images has received significant recent attention. While geometric accuracy has been a focus of prior works, we posit that physically-based applications require meshes to also satisfy *manifold* properties. Intuitively, a mesh is manifold if it can be physically realized, for example, by 3D printing. Typically, reconstructed meshes are post-processed with human inputs for manifoldness, in order to enable ray tracing, slicing or Boolean operations. In contrast, we propose a novel deep network that directly generates manifold meshes (Fig. 1.1).

A manifold is a topological space that locally resembles Euclidean space in the neighbourhood of each point. A manifold mesh is a discretization of the manifold using a disjoint set of simple 2D polygons, such as triangles, which allows designing simulations, rendering and other manifold calculations. While a mesh data structure can simply be defined as a set $(\mathscr{V}, \mathscr{E}, \mathscr{F})$ of vertices $\mathscr{V}$ and corresponding edges $\mathscr{E}$ or face $\mathscr{F}$, not every mesh $(\mathscr{V}, \mathscr{E}, \mathscr{F})$ is manifold. Mathematically, we list various constraints on a singly connected mesh with the set $(\mathscr{V}, \mathscr{E}, \mathscr{F})$ that enables *manifoldness*.

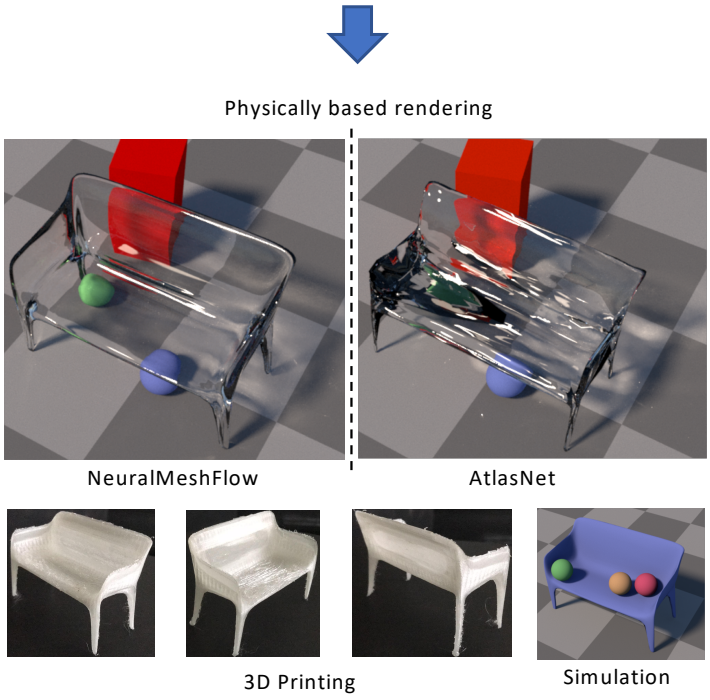
- Each edge $e \in \mathscr{E}$ is common to exactly 2 faces in $\mathscr{F}$ (Fig. 1.2a)
- Each vertex $v \in \mathscr{V}$ is shared by exactly one group of connected faces (Fig. 1.2b)



(a) Inputs

	Approach	Vertex	Edge	Face	Non-Int.
MeshRCNN[4]	explicit	✗	✗	✗	✗
AtlasNet[6]	explicit	✓	✗	✗	✗
AtlasNet-O[6]	explicit	✓	✓	✗	✗
Pixel2Mesh[3]	explicit	✓	✓	✗	✗
GEOMetrics[7]	explicit	✓	✓	✗	✗
3D-R2N2[8]	implicit	✗	✗	✓	✓
PSG[9]	implicit	✗	✗	✓	✓
OccNet[10]	implicit	✗	✗	✓	✓
NMF (Ours)	explicit	✓	✓	✓	✓

(b) Manifoldness of prior work



(c) Applications enabled by NMF Manifold Meshes

Figure 1.1. Given an input as either a 2D image or a 3D point cloud (a) Existing methods generate corresponding 3D mesh that fail one or more manifoldness conditions (b) yielding unsatisfactory results for various applications including physically based rendering (c). NeuralMeshFlow generates manifold meshes which can directly be used for high resolution rendering, physics simulations (see supplementary video) and be 3D printed without the need for any prepossessing or repair effort.

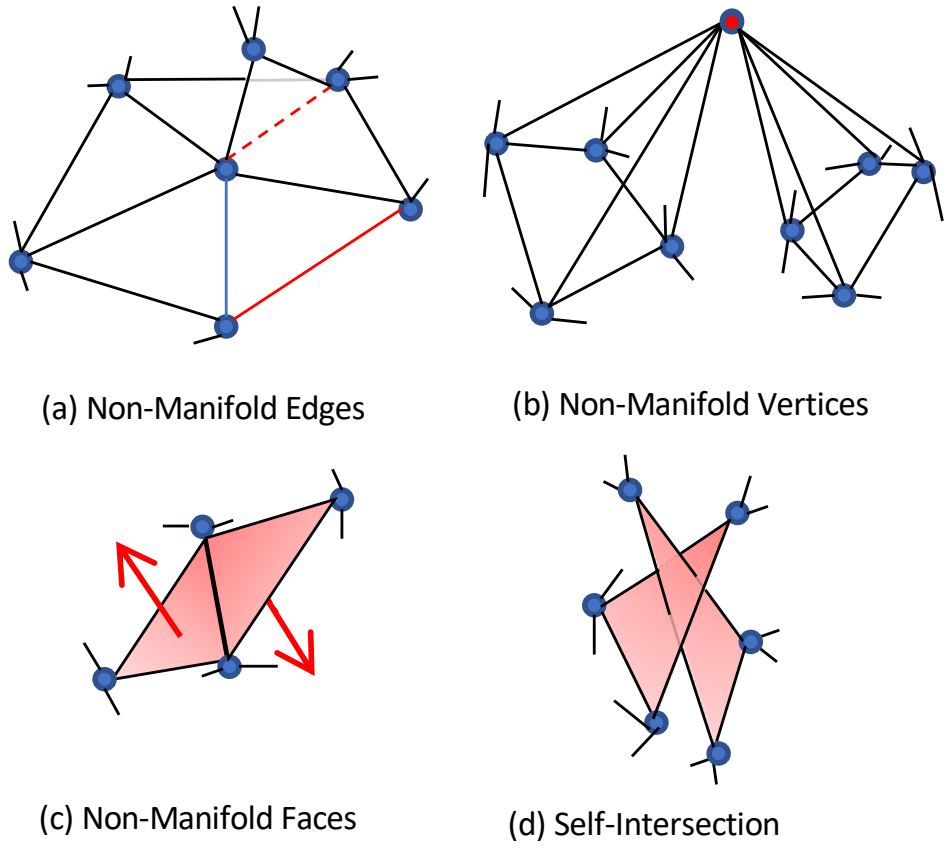


Figure 1.2. Non-manifold geometries for a part of singly connected mesh: (a) An edge that is shared by either exactly one (red) or more than two (red dashed) faces. (b) A vertex (red) shared by more than one group of connected faces. (c) Adjacent faces that have normals (red-arrow) oriented in opposite directions. (d) Faces intersecting other triangles of the same mesh.

- Adjacent faces F_i, F_j have normals oriented in same direction (Fig. 1.2c)

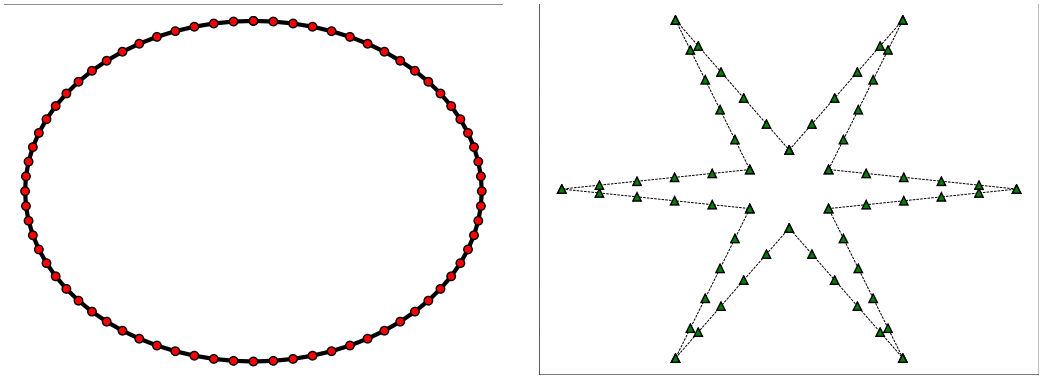
The above mentioned constraints on a mesh $(\mathscr{V}, \mathscr{E}, \mathscr{F})$ guarantee it to be a manifold in the limit of infinitesimally small discretization. That is not the case when dealing with practical meshes with large and non-uniformly distributed triangles. To ensure physical realizability, we tighten the definition with a fourth practical constraint that no two triangles may *intersect* (Fig. 1.2d).

While some simple defects like duplicate elements, isolated vertices, degenerate faces and inner surfaces can also cause a mesh to be *non-manifold*. Such deficiencies are quite simple to fix [11]. In the scope of this work, meshes do not exhibit such issues, we therefore do not include them in our definition of *manifoldness*.

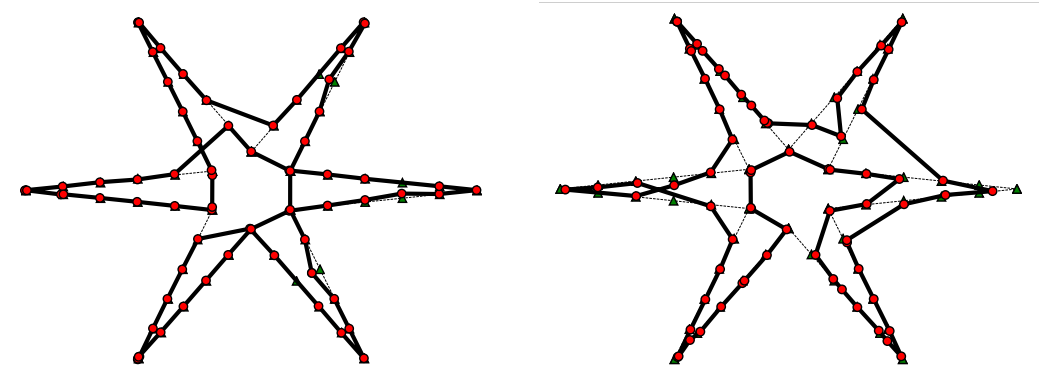
In this work, we pose the task of 3D shape generation as learning a diffeomorphic flow from a template genus-0 manifold mesh to a target mesh. Our key insight is that manifoldness is conserved under a diffeomorphic flow due to their uniqueness [12, 13] and orientation preserving property [14, 15]. In contrast to methods that learn "deformations" of a template manifold using an MLP or graph-based network [3, 7, 6], our approach ensures manifoldness of the generated mesh. We use Neural ODEs [5] to model the diffeomorphic flow, however, must overcome their limited capability to represent a wide variety of shapes [12, 13, 16], which has restricted prior works to single-category representations [17, 18]. We propose novel architectural features such as an instance normalization layer that enables generating 3D shapes across multiple categories and a series of diffeomorphic flows to gradually refine the generated mesh. We show quantitative comparisons to prior works and more importantly, compare resulting meshes on physically meaningful tasks such as rendering, simulation and 3D printing to highlight the importance of manifoldness.

Toy example: regularizer's dilemma

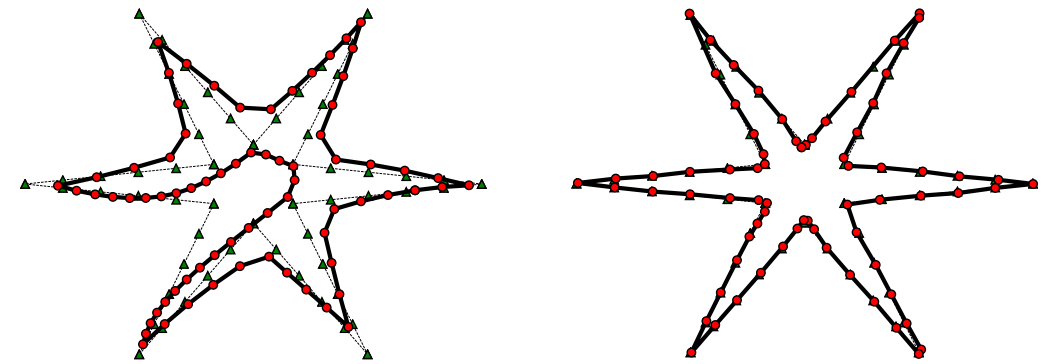
Consider the task of deforming a template unit spherical mesh S (Fig. 1.3a) into a target star mesh T (Fig. 1.3b). We approximate the deformation with a multi-layer perceptron (MLP) f_θ with a unit hidden layer of 256 neurons with $relu$ and output layer with $tanh$ activation. We train f_θ by minimizing various losses over the points sampled from S, T. A conventional approach involves minimizing the Chamfer Distance L_c between S, T, leading to accurate point predictions but several edge-intersections (Fig. 1.3c). By introducing edge length regularization L_e, we get fewer edge-intersections (Fig. 1.3d). but the solution is also geometrically sub-optimal. We can further reduce edge-intersections with Laplacian regularization (Fig. 1.3e), but this takes a bigger toll on geometric accuracy. Thus, attempting to reduce self-intersections by explicit regularization not only makes the optimization hard, but can also lead to predictions with lower geometric accuracy. In contrast, our proposed used of NODE (with dynamics f_θ) is designed by construction [13, 12] to prevent self-intersections without explicit regularization (Fig. 1.3f).



(a) Template (b) Target



(c) Only L_c (d) $L_c + L_e$



(e) $L_c + L_e + L_l$ (f) NODE

Figure 1.3. 2D Toy Example: For the task of deforming a manifold template mesh (a) to a target mesh (b) using explicit mesh regularization (c-e) trades edge-intersections for geometric accuracy. In contrast, a NODE[5] (f) is implicitly regularized preventing edge-intersections without loosing geometric accuracy.

In summary, we make the following contributions:

- A novel approach to 3D mesh generation, Neural Mesh Flow (NMF), with a series of NODEs that learn to deform a template mesh (ellipsoid) into a target mesh with greater *manifoldness*.
- Extensive comparisons to state-of-the-art mesh generation methods for physically based rendering and simulation (see supplementary video), highlighting the advantage of NMF's manifoldness.
- New metrics to evaluate manifoldness of 3D meshes and demonstration of applications to single-view reconstruction, 3D deformation, global parameterization and correspondence.

Chapter 2

Related Work

Existing learning based mesh generation methods, while yielding impressive geometric accuracy, do not satisfy one or more *manifoldness* conditions (Fig. 1.1b). While indirect approaches [8, 9, 19, 20, 21, 22] suffer from the non-manifoldness of the marching cube algorithm [2], direct methods [23, 3, 6, 7] are faced with the *regularizer's dilemma* on the trade-off between geometric accuracy and higher manifoldness, illustrated in Fig. 1.3 and discussed in Sec. 1.

2.1 Indirect Mesh Prediction

Indirect approaches predict the 3D geometry as either a distribution of voxels [24, 25, 26, 27, 28, 29, 30, 31], point clouds [9, 32] or an implicit function representing signed distance from the surface [20, 19, 22]. Both voxel and point set prediction methods struggle to generate high resolution outputs which later makes the iso-surface extraction tools ineffective or noisy [6]. While recent works [33, 34, 35] have been successful in using resolution up to 128^3, this is done by trading higher resolution for shallower networks and smaller batch sizes. Some works exploit the inherent sparsity of 3D data [31, 30] to obtain a higher resolution, but they are often too complicated to implement, require time consuming pre/post processing to get final 3D model. While point clouds provide a much sparser 3D representation, they are found to be memory intensive and harder to train when using resolution higher than 2500 points [6, 9]. Implicit representation based methods have recently been gaining popularity. They usually

involve feeding a neural network with a latent code and a query point, encoding the spatial coordinates [20, 19, 22] or local features [36], to predict the TSDF value [20] or the binary occupancy of the point [19, 22]. However, these approaches are computationally expensive since in order to get a surface from the implicit function representation, several thousands of points must be sampled. Moreover, for shapes such as chairs that have thin structures, implicit methods often fail to produce a single connected component.

All the above methods depend on the marching cube algorithm [2] for iso-surface extraction. While marching cubes can be applied directly to voxel grids, point clouds first regress the iso-surface using surface normals. Implicit function representations must regress TSDF values per voxel and then perform extensive query to generate iso-surface based on a threshold τ. This is used to classify grid vertices $v_i \in \mathcal{V}$ as 'inside' ($TSDF(v_i) \leq \tau$) and 'outside' ($TSDF(v_i) \geq \tau$). For each voxel, based on the arrangement of its grid vertices, marching cubes [2, 1, 37, 38] follows a lookup-table to find a triangle arrangement. Since this rasterization of iso-surface is a purely local operation, it often leads to ambiguities [38, 1, 37], resulting in meshes being *non-manifold*. An example of non-manifoldness is shown in figure 2.1 where for a particular set of scaler values, the marching cubes result in a non-manifold vertex.

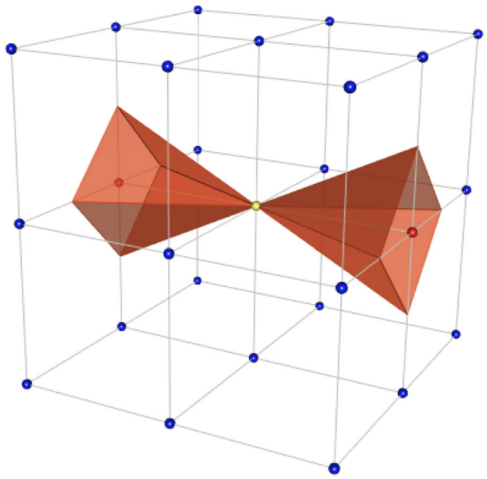


Figure 2.1. An example (taken from [1]) of a grid configuration which will result in a non-manifold triangulation. The blue, red and yellow grid vertices respectively represent the vertices with the scaler value smaller, greater and equal scaler to the isovalue. By running marching cubes [2] we end up with a non-manifold vertex.

2.2 Direct Mesh Prediction

A mesh based representation stores the surface information cheaply as list of vertices and faces that respectively define the geometric and topological information. Early methods of mesh generation relied on predicting the parameters of category based mesh models[39, 40, 41]. The templates are often based on SP models [42] which allows for learning low-dimensional parameters that can encode features of a mesh with much higher complexity. While these methods output manifold meshes, they work only when parameterized manifold meshes are available for the object category. This therefore limits their application to only selected object categories. Recently, meshes have been successfully generated for a wide class of categories using topological priors [6, 3]. Deep networks are used to update the vertices of initial mesh to match that of the final mesh. AtlasNet [6] uses point (or a ResNet-18 [43] for image input) encoder and uses a decoder consisting of a series of MLPs. Given points sampled from a *atlas* and the shape encoding, the decoder predicts the shape vertices. For obtaining the final mesh, topology of *atlas* mesh is used to connect the predicted vertices. The approach is simple to apply and uses Chamfer distance, applied on the vertices to train the network. Pixel2Mesh

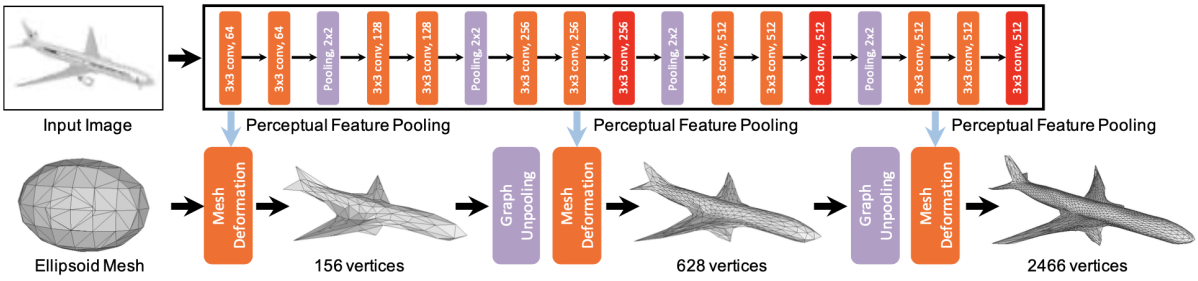


Figure 2.2. Pixel2Mesh[3] proposes a cascaded mesh deformation network. The full model contains three mesh deformation blocks in a row. Each block increases mesh resolution and estimates vertex locations, which are then used to extract perceptual image features from 2D CNN for the next block. Inspite of several mesh regularizers, the predicted mesh contains several non-manifold issues.

[3] (Fig 2.2) uses a similar approach where the template mesh is chosen as an icosphere and a coarse-to-fine deformation approach is proposed which is trained using vertex Chamfer loss. However, using a point set training scheme alone for meshes leads to severe topological issues and produced meshes are not manifold. Therefore, some recent works have proposed to use

mesh regularizers like Laplacian [3, 7, 4, 44], edge lengths [4, 7] and normal consistency [4] to constrain the flexibility of vertex predictions, but they suffer from the *regularizer's dilemma* discussed in Fig. 1.3, as better geometric accuracy comes at a cost of manifoldness. One of the limitations of using a template mesh is that the topology of predicted mesh is fixed. Therefore, approaches like Pixel2Mesh [3] cannot represent shapes with non-genus zero. MeshRCNN [4]

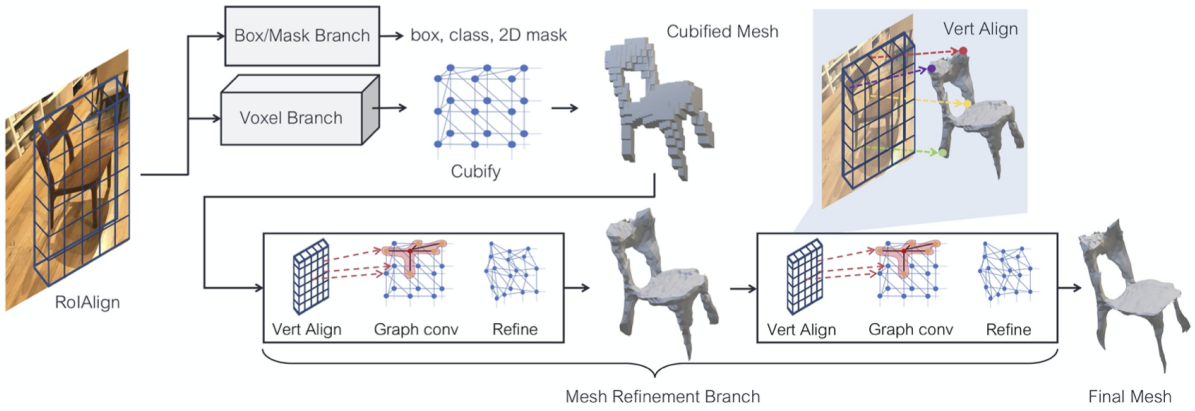


Figure 2.3. MeshRCNN[4] augments Mask R-CNN with 3D shape inference. The *voxel branch* predicts a coarse shape for each detected object which is further deformed with a sequence of refinement stages in the *mesh refinement branch*. While this avoids the limitation to genus-0 topology, the *cubify* operation results in non-manifold vertices and edges in the predicted mesh.

(Fig 2.3) solves this issue by first predicting a coarse voxel representation, and implementing a *cubify* step that converts this voxel representation to a mesh. This is done by replacing each occupied voxel with a cuboid triangle mesh with 8 vertices, 18 edges and 12 faces. This results in a watertight mesh whose topology depends on the voxel predictions. Later, mesh refinement is done using a deep graph-based CNN (GCNN) to obtain the final mesh. While this solves the limitation imposed by topology, it causes the mesh to have non-manifold vertices and edges. In contrast to the above approaches, the proposed NMF achieves high resolution meshes with a high degree of manifoldness across a wide variety of shape categories. Similar to previous approaches [6, 3, 7], an initial ellipsoid is deformed by updating its vertices. However, instead of using explicit mesh regularizers, NMF uses NODE blocks to learn the diffeomorphic flow to implicitly discourage self-intersections, maintain the topology and thereby achieve better manifoldness of generated shape. The method is end-to-end trainable without requiring any post-processing.

Chapter 3

Neural Mesh Flow

We now introduce NeuralMeshFlow (Fig 3.1), which learns to auto-encode 3D shapes. NMF broadly consists of four components. First, the target shape $\mathscr{M}_T$ is encoded by uniformly sampling N points from its surface and feeding them to a PointNet[45] encoder to get the global shape embedding z of size k. Second, NODE blocks diffeomorphically *flow* the vertices of template sphere towards target shape conditioned on shape embedding z. Third, the instance normalization layer performs non-uniform scaling of NODE output to ease cross-category training. Finally, refinement flows provide gradual improvement in quality. We start with a discussion of NODE and its regularizing property followed by details on each component.

3.1 NODE Overview.

Neural Ordinary Differential Equations (NODE) [5] are recently proposed class of neural networks where instead of learning directly learning a mapping from input to output domain, the transformation $\phi_T : \mathscr{X} \rightarrow \mathscr{X}$ is learned as solutions for initial value problem (IVP) of a parameterized ODE (eq 3.1) . Here $x_0, x_T \in \mathscr{X} \subset \mathbb{R}^n$ represent the input and output from the network respectively, while $T \in \mathbb{R}$ is a hyper parameter which represents the duration of the *flow* from x_0 to x_T. NODEs can be treated as the extension of residual neural networks [43] since a residual block is simply a discretization (eq 3.2) of a continuous time system of ordinary

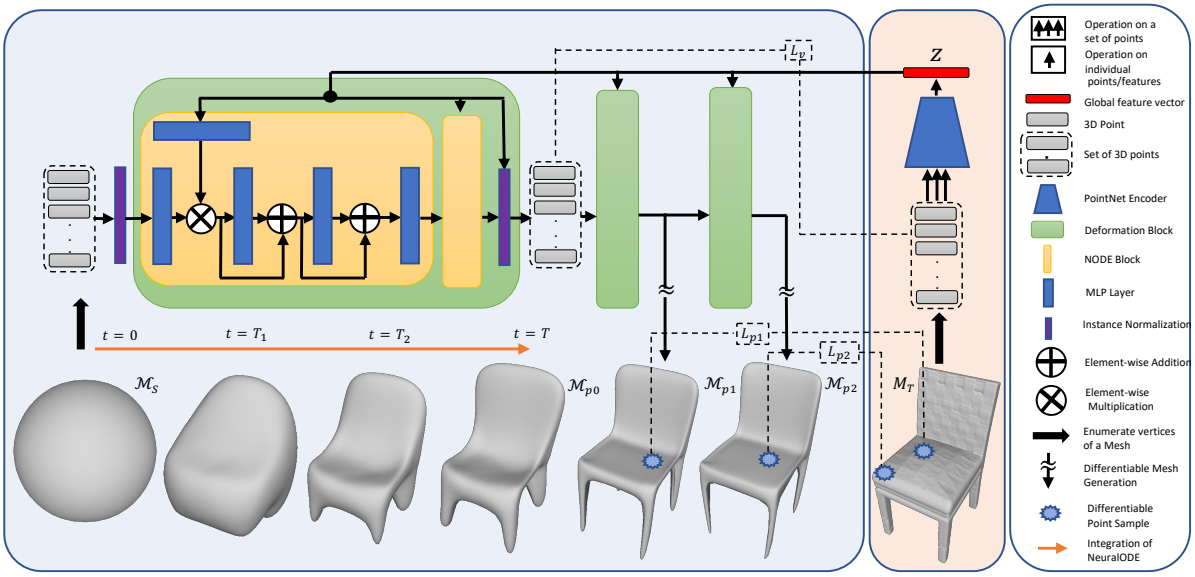


Figure 3.1. Neural Mesh Flow consists of three deformation blocks that perform point-wise flow on spherical mesh vertices based on the shape embedding z from target shape $\mathcal{M}_T$. The bottom row shows an actual chair being generated at various stages of NMF. Time instances $0 < T_1 < T_2 < T$ show the deformation of spherical mesh into a coarse chair representation $\mathcal{M}_{p0}$ by the first deformation block. Further deformation blocks perform refinements to yield refined meshes $\mathcal{M}_{p1}, \mathcal{M}_{p2}$.

differential equation (eq 3.3).

$$x_T = \phi_T(x_0) = x_0 + \int_0^T f_\Theta(x_t)dt \tag{3.1}$$

$$x_{t+1} - x_t = f_\Theta(x_t, t) \tag{3.2}$$

$$\frac{dx_t}{dt} = \lim_{\delta_t \to 0} \frac{x_{t+\delta_t} - x_t}{\delta_t} = f_\Theta(x_t, t) \tag{3.3}$$

Here, the neural networks form the *dynamics* f_Θ of the NODE. These can often be very deep layers of convolutional or multi-layer perceptrons. The adjoint sensitivity [46] method is employed to perform the reverse-mode differentiation through the ODE solver and therefore learn the network parameters Θ using the standard gradient descent approaches.

The Neural ODE formulation has several benefits. For a well behaved dynamics f_Θ

(Lipschitz continuous) any two distinct trajectories of NODE can never intersect due to the existence and uniqueness of IVP solutions [12, 13]. Moreover, NODE manifest the orientation preserving property of diffeomorphic flows[15, 14]. These properties lead to strong implicit regularization against self-intersection and non-manifold faces. Here, we will briefly discuss these two properties.

3.1.1 Absence of self-intersection

Distinct trajectories of a Neural ODE can never intersect due to the existence and uniqueness property of IVP solutions [12, 13]. We can prove it by simple contradiction. Refer to fig 3.2 for visualization of a 1-D case. Given two distinct initial values $x_0 = \{x_+, x_- | x_+ \neq x_-,\ x_+, x_- \in \mathbb{R}^n\}$ and the IVP (eq 3.1). For the range of values of $t \in [0, T]$, the IVP will create trajectories in space starting from $\{x_+, x_-\}$. Let for some time $t = t_*$ the two trajectories intersect i.e. $\phi_{t_*} = \phi_{t_*}(x_+) = \phi_{t_*}(x_-)$. Now, solving IVP (eq. 3.1) with initial value ϕ_{t_*} for the reverse time $-t_*$ will yield $\phi_{-t_*}(\phi_{t_*}(x_+)) = \phi_{-t_*}(\phi_{t_*}(x_-) \implies x_+ = x_-$ which is a contradiction. We therefore conclude that in a NODE, for the same time duration, distinct points lead to non-intersecting trajectories and this acts as a regularizer against self intersections. In fact, if forced to learn such a mapping, a NODE of arbitrary complexity converges to asymptotic solutions (Fig. 3.2). For our use case, this guarantee that distinct vertices of a mesh $\{x_+, x_- | x_+ \neq x_-,\ x_+, x_- \in \mathbb{R}^3\}$ will never intersect leading to a higher degree of *manifoldness*.

3.1.2 Preservation of Orientation

We will begin our discussion by first defining the *orientation* of a manifold. Let the ordered set $(b_1, b_2, ..., b_n)$ be the basis of $\mathbb{R}^n$. We can view them as a matrix B such that its i^{th} column is b_i. Depending upon the choice of the order of $b_i, i = 1, ..., n$ the determinant of B can either turn out to be positive (B^+) or negative (B^-). By an *orientation* for $\mathbb{R}^n$ we mean the choice of B^+ or B^- as the preferred source for picking a basis for $\mathbb{R}^n$. The default orientation is chosen to be B^+.

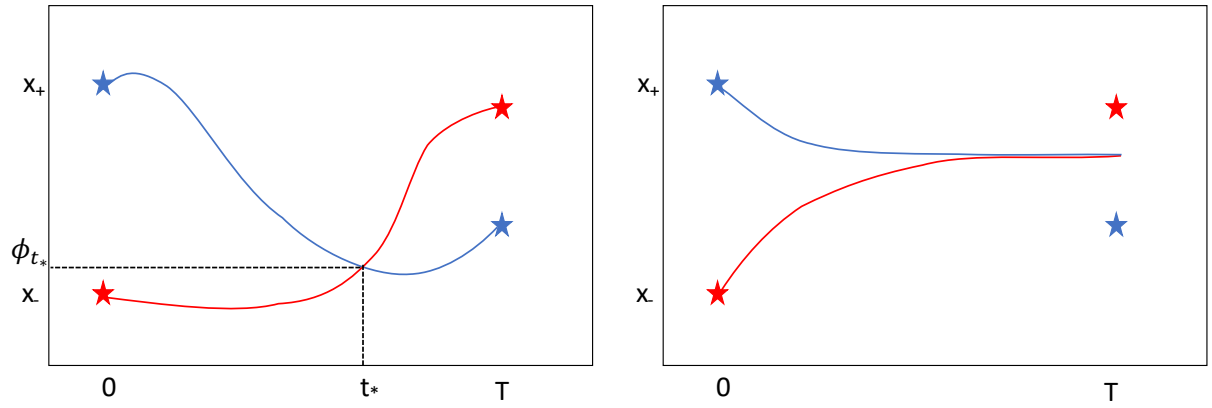


Figure 3.2. (Left) Continuous trajectories originating from distinct points x_- (blue) and x_+ (red) that intersect at time t_*. (Right) Since a NODE cannot have intersection of distinct trajectories, it fails to learn the required mapping and results in asymptotic curves.

Given a connected orientable manifold M (Fig 3.3(a)), we wish to map it to a desired manifold N via some mapping $\phi : M \to N$. The mapping ϕ is assumed to be a neural network which is trained by minimizing the distance between predicted and target points of manifold N. While any neural network of sufficient complexity can learn to predict accurate point locations (Fig 3.3(b)), the orientation of the predicted manifold is not consistent everywhere. Whether a mapping preserves or reverses the orientation at a point, depends on its jacobian. If the jacobian is positive definite, then the orientation is preserved, otherwise, we get opposite orientation. A non-diffeomorphic mapping can have a jacobian of arbitrary definiteness for each point of the manifold. We show this by observing a triangle on the input manifold M, whose orientation gets reversed under a non-diffeomorphic mapping (Fig 3.3(b)), whereas the rest of its neighbourhood retains the original orientation. A non-diffeomorphic mapping therefore, arbitrarily preserves/reverses the orientation at each point of the manifold.

If we restrict the mapping ϕ to be a diffeomorphic mapping, then the orientations of the input and predicted *local charts* $(U, \eta), (V, \psi)$ are either same or reversed everywhere over U. We prove this property in lemma 3.1.1.

Lemma 3.1.1 *Let M and N be orientable manifolds and $\phi : M \to N$ be a diffeomorphic. Consider local charts (U, η), (V, ψ) centered at p and $\phi(p)$ which are orientation preserving. Then ϕ is*

either orientation preserving or orientation reversing on U.

Proof: Given the local charts, let $\hat{\phi} = \psi \circ \phi \circ \eta^{-1}$. Since ϕ is diffeomorphic, this implies that $\hat{\phi} : \mathbb{R}^n \to \mathbb{R}^n$ is also diffeomorphic. Thus, the jacobian of $\hat{\phi}$ is either everywhere positive or everywhere negative (over U). Consequently, $\hat{\phi}$ will either be orientation preserving or reversing everywhere. Since η, ψ are both orientation preserving, this implies that ϕ also either preserves the orientation everywhere or reverses it everywhere over U. This completes the proof.

While this ensures uniformity of orientation over *local charts*, Lemma 3.1.2 extends this property to entire manifold given that it is *connected*. In figure 3.3(c) we observe that the orientation of the predicted manifold under a diffeomorphic mapping is reversed everywhere. This is because the jacobian of a diffeomorphic mapping can either be positive or negative definite everywhere.

Lemma 3.1.2 *Let M and N be connected orientable manifolds and $\phi : M \to N$ be a diffeomorphic. Then ϕ either everywhere preserves the orientation or reverses it for all M.*

Proof: From the above lemma we conclude that ϕ is either orientation preserving or reversing everywhere over the local chart U. This implies that U is open. Similarly, it can be shown that $M - A$ is open as well. But since M is connected, we either have A = M or A = $\{\}$. Thus, all local charts with non-zero elements, must have the same orientation. We therefore conclude that ϕ is either orientation preserving or reversing everywhere over M. This completes the proof.

To ensure that the predicted manifold indeed preserves the orientation, we further restrict the mapping ϕ to be a diffeomorphic flow, such as a Neural ODE [5]. Lemma 3.1.3 shows that the jacobian of a diffeomorphic flow is always positive, thereby proving the fact that the orientation is preserved under a diffeomorphic flow $\phi : M \to N$. We observe this in fig 3.3(d) where the orientation of predicted manifold is everywhere same as that of the input manifold M (Fig 3.3 (a)).

Lemma 3.1.3 *Let $\phi(t,x)$ be a diffeomorphic flow over some manifold M with the vector field f. Then the jacobian of $\phi(t,x)$ is always positive.*

Proof: Given the diffeomorphic flow ϕ with vector field f we have the following relationship.

$$\frac{\partial \phi(t,x)}{\partial t} = f(\phi(t,x)) \tag{3.4}$$

Let us denote the jacobian of $\phi(t,x)$ with $z(t)$ such that

$$z(t) := \frac{\partial \phi(t,x)}{\partial x} \tag{3.5}$$

By taking the partial derivative of eq 3.4 with respect to the flow $\phi(t,x)$ and applying chain rule we get

$$\frac{\partial}{\partial x}\left(\frac{\partial \phi(x,t)}{\partial t}\right) \cdot \frac{\partial x}{\partial \phi(x,t)} = \frac{df(\phi(x,t))}{d(\phi(x,t)} = Df(\phi(x,t)) \tag{3.6}$$

Substituting eq. 3.5 we get a familiar looking ordinary differential equation

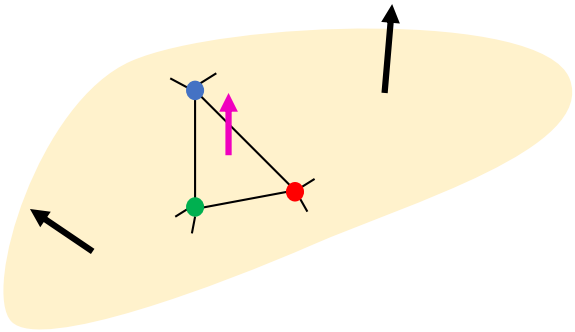
$$\frac{dz(t)}{dt} = Df(\phi(x,t))z(t) \tag{3.7}$$

We solve this using Wronskian formula to get the desired determinant of the jacobian (eq. 3.5)

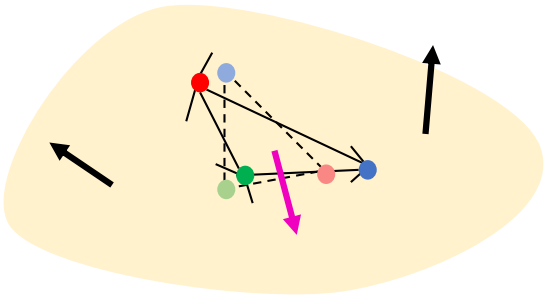
$$det\left(\frac{\partial}{\partial x}(\phi(x,t))\right) = exp\left(\int_0^t tr(Df(\phi(\zeta,x)))d\zeta\right) > 0 \tag{3.8}$$

We thus prove that the jacobian of a diffeomorphic flow over a manifold is always positive.

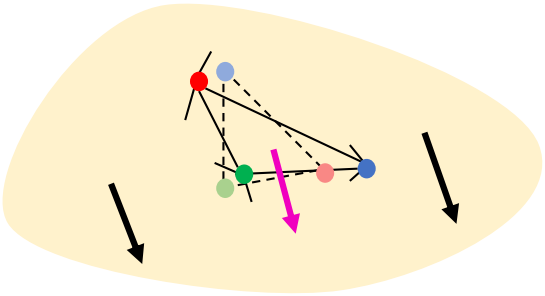
Based on above lemmas we therefore prove that a diffeomorphic flow such as a NeuralODE [5] over a connected manifold is orientation preserving. There are several other advantages to NODE compared to traditional MLPs such as improved robustness [47], parameter efficiency [5] and the ability to learn normalizing flows [48, 17, 18] and homeomorphism [13]. We refer readers to [12, 13, 16] for more details.



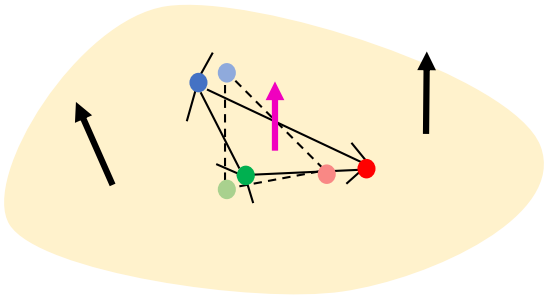
(a) Connected Manifold



(b) Image under non-diffeomorphic mapping



(c) Image under diffeomorphic mapping



(d) Image under diffeomorphic flow

Figure 3.3. Given a connected manifold (a) we compare the orientation of a triangle to its neighbourhood under the *pushforward* of various kind of mappings. We observe that the orientation is non-uniformly preserved/reversed under a non-diffeomorphic mapping (b) since the jacobian can be positive/negative definite anywhere. With a diffeomorphic mapping (c) the orientation is reversed everywhere if the jacobian is negative definite. However, when the mapping is a diffeomorphic flow (d) the jacobian is always positive definite (lemma 3.1.3) which results in the orientation being preserved everywhere.

3.2 Diffeomorphic Conditional Flow.

The standard NODE formulation cannot be used directly for the task of 3D mesh generation since they lack any means to feed in shape embedding and are therefore restricted to learning a few shape. A naive way would be to concatenate features to point coordinates like is done with traditional MLPs [3, 7] but this destroys the shape regularization properties due to several augmented dimensions[13, 12]. Our key insight is that instead of a fixed NODE dynamics f_Θ we can use a family of dynamics $f_{\Theta|z}$ parameterized by z while still retaining the uniqueness property as long as z is held constant for the purpose of solving IVP with initial conditions $\{x_0, x_T\}$.

The objective of conditional flow (NODE Block) therefore is to learn a mapping $F_{\Theta|z}$ (eq.3.9) given the shape embedding z and initial values $\{(p_I^i, p_O^i)|p_I^i \in \mathscr{M}_I, p_O^i \in \mathscr{M}_O\}$ where $\mathscr{M}_I, \mathscr{M}_O$ are respectively the input and output point clouds.

$$p_O^i = F_{\Theta|z}(p_I^i, z) = p_I^i + \int_0^T f_{\Theta|z}(p_I^i, z)dt \tag{3.9}$$

3.3 Instance Normalization.

Normalizing input and hidden features to zero mean and unit variance is important to reduce co-variate shift in deep networks [49, 50, 51, 52, 53, 54]. While trying to deform a template sphere to targets with different variances (like a firearm and chair) different parts of the template need to be *flown* by very different amounts to different locations (Fig. 3.4a). This is observed to causes significant *strain* on the NODE which ends up learning more complex dynamics resulting in meshes with poor geometric accuracy and manifoldness (Fig 3.4d and Table 3.1). Instance normalization separates the task of learning target variances from that of learning target attributes. As shown in Fig 3.5, when using instance normalization, NMF focuses

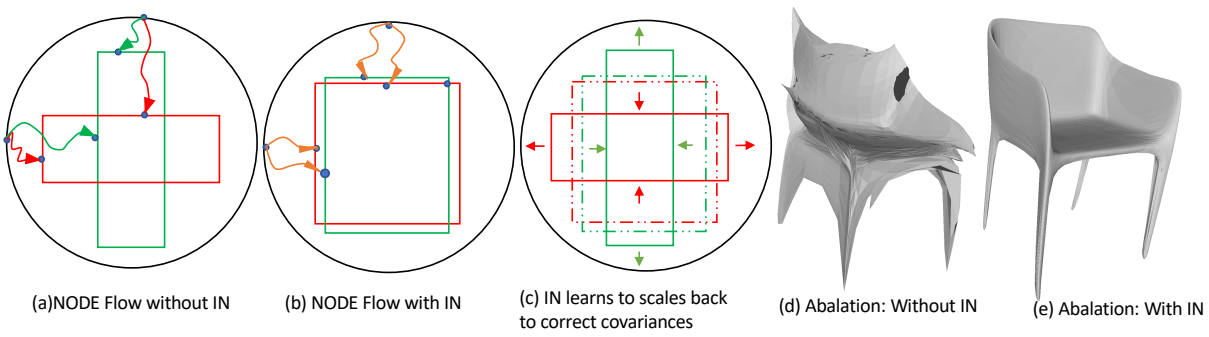
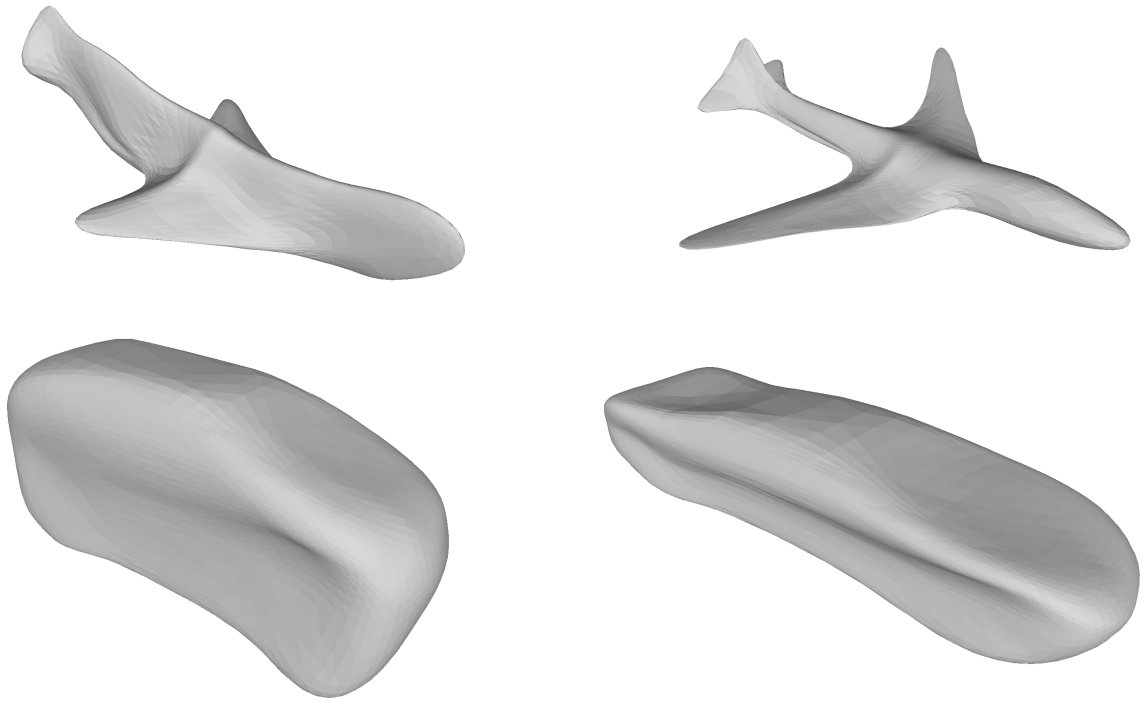


Figure 3.4. The impact of instance normalization (IN) and refinement flows in NMF. (a) Learning deformation of a template (black) to target shapes of different variances (red and green) require longer non-uniform NODE trajectories making learning difficult. (b) IN allows NODE to learn deformations to an arbitrary variance. (c) This leads to simpler dynamics and can later be scaled back to correct shape variance. (d) A model trained without IN leads to self-intersections and non-manifold faces due to very complex dynamics being learnt. (e) A model with IN is smoother and regularized.

on learning the geometry and other features (Fig 3.5(a)) whereas the instance normalization learns to scale it to match the target variance (Fig 3.5(b)). As we can see this greatly simplifies the task of learning 3D shapes across several categories. It gives NODE flexibility to deform the template to a target with arbitrary variance which yields better geometric accuracy(Fig. 3.4b). This is later scaled back to the correct variance by Instance normalization layer (Fig. 3.4c) Given an input point cloud $\mathscr{M} \in \mathbb{R}^{N\times 3}$ and its shape embedding z, the instance normalization calculates the point average $\mu \leftarrow \frac{1}{|\mathscr{M}|}\sum_i p^i, p^i \in \mathscr{M}$ and then applies non-uniform scaling $\mathscr{M} \leftarrow (\mathscr{M} - \mu) \odot \Delta(z)$ to arrive at correct target variances. Here $\Delta : \mathbb{R}^k \rightarrow \mathbb{R}^3$ is an MLP that regress the three variance coefficients based on shape embedding z. $\odot$ refers to the element wise multiplication.

3.4 Overall Architecture.

A single NODE block is often not sufficient to get desired quality of results. We therefore stack up two NODE blocks in a sequence followed by an instance normalization layer and call the collection a deformation block. While a single deformation block is capable of achieving reasonable results (as shown by $\mathscr{M}_{p0}$ in Fig. 3.1, 3.6) we get further refinement in quality by having two additional deformation blocks. Notice how the $\mathscr{M}_{p1}$ has a better geometric accuracy than $\mathscr{M}_{p0}$ and $\mathscr{M}_{p2}$ is *sharper* compared to $\mathscr{M}_{p1}$ with additional refinement. We report the



(a) NMF prediction before IN (b) NMF prediction after IN

Figure 3.5. Instance Normalization separates the task of learning shape attributes (a) and shape variances (b)

Table 3.1. The impact of instance normalization (IN) and refinement flows in NMF. (a) Learning deformation of a template (black) to target shapes of different variances (red and green) require longer non-uniform NODE trajectories making learning difficult. (b) IN allows NODE to learn deformations to an arbitrary variance. (c) This leads to simpler dynamics and can later be scaled back to correct shape variance. (d) A model trained without IN leads to self-intersections and non-manifold faces due to very complex dynamics being learnt. (e) A model with IN is smoother and regularized.

	Chamfer-L2 (↓)	Normal Consistency (↑)	NM-Faces (↓)	Self-Intersection (↓)	Time (↓)
No Instance Norm	6.48	**0.820**	2.94	3.28	183
0 refinement	5.00	0.818	0.39	**0.03**	**68**
1 refinement	4.93	0.819	**0.38**	**0.03**	124
2 refinement	**4.65**	0.818	0.73	0.09	189

geometric accuracy, manifoldness and inference time for different amounts of refinement in Table 3.1. The reported quantities are averaged over the 11 Shapnet categories (this excludes watercraft and lamp where NMF struggles with thin structures). It is important to note that we observe manifoldness metrics decrease by a small amount with more refinement modules. We believe this decrease in manifoldness is because in order to approximate sharp features, more complex dynamics need to be learned, which in turn demand a tighter tolerance value. In this work we did not choose an extremely tight tolerance so as to get a higher inference speed. For applications that require extremely high manifoldness, a tighter tolerance can be used. To summarize, the entire NMF pipeline can be seen as three successive diffeomorphic flows $\{F^0_{\Theta|z}, F^1_{\Theta|z}, F^2_{\Theta|z}\}$ of the initial spherical mesh to gradually approach the final shape.

3.5 Loss Function.

In order to learn the parameters Θ it is important to use a loss which meaningfully represents the difference between the predicted M_P and the target M_T meshes. To this end we use the bidirectional Chamfer Distance (eq.3.10) on the points sampled differentiably [55] from predicted $\tilde{M}_P$ and target $\tilde{M}_T$ meshes.

$$\mathscr{L}(\Theta) = \sum_{p \in \tilde{M}_P} \min_{q \in \tilde{M}_T} ||p-q||^2 + \sum_{q \in \tilde{M}_T} \min_{p \in \tilde{M}_P} ||p-q||^2 \tag{3.10}$$

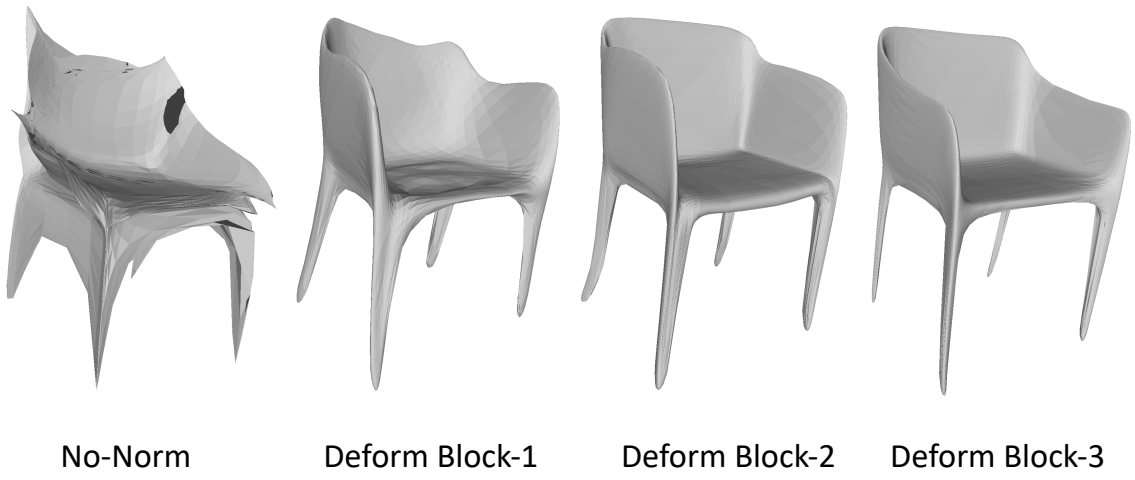


Figure 3.6. Effect of IN and refinement modules. With out any instance normalization (No-Norm) training NMF cross-category becomes difficult leading to severe non-manifoldness. With refinement modules (Deform Block -1,2,3) we successively get meshes that are sharper and more geometrically accurate.

We compute chamfer distances $\mathscr{L}_{p1}, \mathscr{L}_{p2}$ for meshes after deformation blocks $F^1_{\Theta|z}$ and $F^2_{\Theta|z}$. For meshes generated from $F^0_{\Theta|z}$ we found that computing chamfer distance $\mathscr{L}_v$ on the vertices gave better results since it encourages predicted vertices to be more uniformly distributed (like points sampled from target mesh). We thus arrive at the overall loss function to train NMF.

$$\mathscr{L} = w_0\mathscr{L}_v + w_1\mathscr{L}_{p1} + w_2\mathscr{L}_{p2} \tag{3.11}$$

Here we take $w_0 = 0.1, w_1 = 0.2, w_3 = 0.7$ so as to enhance mesh prediction after each deformation block. The adjoint sensitivity [46] method is employed to perform the reverse-mode differentiation through the ODE solver and therefore learn the network parameters Θ using the standard gradient descent approaches.

3.6 Dynamics Equation.

The Neural ODE $F_{\theta|z}$ is built around the dynamics equation $f_{\theta|z}$ which is learned by a deep network. Given a point $x \in \mathbb{R}^3$ we first get 512 length point features by applying a linear

Table 3.2. Effect of tolerance on geometric accuracy and manifoldness. A tighter choice of tolerance results in better performance.

Error Tolerance	Chamfer-L2 (↓)	Normal Consistency (↑)	NM-Faces (↓)	Self-Intersection (↓)	Time (↓)
1e-3	8.09	**0.832**	0.859	0.43	**187**
1e-4	6.09	0.828	0.75	0.24	184
1e-5	**5.54**	0.826	**0.71**	**0.10**	189

layer. To condition the NODE on shape embedding, we extract a 512 length shape feature from the shape embedding z and multiply it element wise with the obtained point features to get the *point-shape* features. Thus, *point-shape* features contains both the point features as well as the global instance information. Lastly, we feed the *point-shape* features into two residual MLP blocks each of width 512 and subsequent MLP of width 512 which outputs the predicted point location $y \in \mathbb{R}^3$. Based on the findings of [56, 16] we make use of the *tanh* activation after adding the residual predictions at each step. This ensures maximum flexibility in the dynamics learned by the deep network.

It is important to discuss *tolerance* value that is used to solve the dynamics. This determines the step size of the ODE solver. In Table 3.2 we show the trend in geometric accuracy as well as manifoldness at various tolerance values. Clearly, by taking a lower value of tolerance, we get higher geometric accuracy as well as manifoldness, but this comes at a slight cost of inference time. In practice, we did not find much improvement in results by taking tolerance lesser than $1e^{-5}$.

3.7 Implementation Details

For auto-encoding, we uniformly sample $N = 2520$ from the target mesh and using PointNet[45] encoder, get a shape embedding z of size $k = 1000$. During training, the Neural ODEs are solved with a tolerance of $1e^{-5}$ and interval of integration set to $t = 0.2$ for deforming an icosphere with 622 vertices. At test time, we use an icosphere of 2520 vertices and tolerance of $1e^{-5}$. We train NMF for 125 epochs using Adam [57] optimizer with a learning rate of 10^{-5} and a batch size of 250, on 5 NVIDIA 2080Ti GPUs for 2 days. For single view reconstruction,

we train an image to point cloud predictor network with pretrained ResNet encoder of latent code 1000 and a fully-connected decoder with size 1000,1000,3072 with relu non-linearities. The point predictor is trained for 125 epochs on the same split as NMF auto-encoder.

Chapter 4

Results

In this section we show qualitative and quantitative results on the task of auto-encoding and single view reconstruction of 3D shapes with comparison against several state of the art baselines. In addition to these tasks, we also demonstrate several additional features and applications of our approach including physics simulation, 3D printing, latent space interpolation texture mapping, consistent correspondence and shape deformations.

4.1 Data

We evaluate our approach on the ShapeNet Core dataset [58], which consists of 3D models across 13 object categories. We use the training, validation and testing splits provided by [8] to be comparable to other baselines. We use rendered views from [8] and sample 3D points using [59].

4.2 Evaluation criteria

We evaluate the predicted shape $\mathscr{M}_P$ for geometric accuracy to the ground truth $\mathscr{M}_T$ as well as for manifoldness. For geometric accuracy, we follow [4] and compute the bidirectional Chamfer distance according to (3.10) and normal consistency using (4.1) on 10000 points sampled from each mesh. Since Chamfer distance is sensitive to the size of meshes, we scale the meshes to lie within a unit radius sphere. Chamfer distances are report by multiplying with 10^3.

With $\tilde{M}_P, \tilde{M}_T$ the point sets sampled from $\mathscr{M}_p, \mathscr{M}_T$ and $\Lambda_{P,Q} = \{(p, argmin_q||p-q||) : p \in P\}$, we define

$$\mathscr{L}_n = |\tilde{M}_P|^{-1} \sum_{(p,q)\in\Lambda_{\tilde{M}_P,\tilde{M}_T}} |u_p \cdot u_q| + |\tilde{M}_T|^{-1} \sum_{(p,q)\in\Lambda_{\tilde{M}_T,\tilde{M}_P}} |u_q \cdot u_p| - 1 \tag{4.1}$$

We detect non-manifold vertices (Fig. 1.2(b)) and edges (Fig. 1.2(a)) using [60] and report the metrics 'NM-vertices', 'NM-edges' respectively as the ratio($\times 10^5$) of number of non-manifold vertices and edges to total number of vertices and edges in a mesh. To calculate non-manifold faces (Fig. 1.2(c)), we count number of times adjacent face normals have a negative inner product, then the metric 'NM-Faces' is reported as its ratio(%) to the number of edges in the mesh. To calculate the number of instances of self-intersection (Fig. 1.2(d)), we use [61] and report the ratio(%) of number of intersecting triangles to total number of triangles in a mesh.

4.3 Baselines

We compare with official implementations for Pixel2Mesh [3, 4], MeshRCNN [4] and AtlasNet [6]. We use pretrained models for all these baselines motioned in this paper since they share the same dataset split by [8]. We use the implementation of Pixel2Mesh provided by MeshRCNN, as it uses a deeper network that outperforms the original implementation. We also consider AtlasNet-O which is a baseline proposed in [6] that uses patches sampled from a spherical mesh, making it closer to our own choice of initial template mesh. To account for possible variation in manifoldness due to simple post processing techniques, we also report outputs of all mesh generation methods with 3 iterations of Laplacian smoothing. Further iterations of smoothing lead to loss of geometric accuracy without any substantial gain in manifoldness. We also compare with occupancy networks [19], a state-of-the-art indirect mesh generation method based on implicit surface representation. we compare with several variants of OccNet based on the resolution of Multi Iso-Surface Extraction algorithm [19]. To this end, we create OccNet baselines OccNet-1, OccNet-2 and OccNet-3 with MISE upsampling of 1, 2 and 3 times respectively. For fair comparison to other baselines, we use OccNet's refinement module

to output its meshes with 5200 faces.

4.4 Auto-encoding 3D shapes

We now evaluate NMF's ability to generate a shape given an input 3D point cloud and compare against AtlasNet and AtlasNet-O. We evaluate the geometric accuracy and manifoldness of generated meshes. Additionally, we show physically based renderings of generated meshes with dielectric and conductor materials to highlight artifacts due to non-manifoldness.

We report mean errors as well as per category results for shape generation from point clouds in Table 4.1.

Notice that AtlasNet-O with smoothing has 20 times the self-intersection compared to NMF without any smoothing. With Laplacian smoothing, NMF becomes practically intersection free. NMF also outperforms the baselines with and without smoothing in terms of non-manifold faces. Since AtlasNet uses 25 mesh non-manifold open templates to construct the final mesh it yields a constant value of 7.40 for its non-manifold edges while AtlasNet-O and NMF have manifold-edges. All the three methods have manifold vertices. Finally, NMF generates meshes with a higher normal consistency, leading to more realistic results in simulations and physically-based rendering. Visualizations in Fig. 4.1 show severe self-intersections and flipped normals for AtlasNet baselines which are absent for NMF. This leads to NMF giving more realistic physically based rendering results. Note the reflection of red box and green ball through NMF mesh, which are either distorted or absent for AtlasNet. The blue ball's reflection on conductor's surface is closer to ground truth for NMF due to higer manifoldness.

4.5 Single-view reconstruction

We evaluate NMF for single-view reconstruction and compare against state-of-the-art methods. Per category and Mean errors over ShapeNet categories are reported in Table 4.2 - 4.7. We note significantly lower self-intersections for NMF compared to the best baseline even after

Table 4.1. Per category performance for auto-encoding task. NMF clearly out-performs baselines in terms of manifoldness and normal consistency.

	Chamfer-L2 (↓)			Normal Consistency (↑)			NM-Faces ↓)			Self-Intersection (↓)		
	AtNet	AtNet-O	NMF	AtNet	AtNet-O	NMF	AtNet	AtNet-O	NMF	AtNet	AtNet-O	NMF
table	8.65	**4.38**	7.08	0.837	0.842	**0.853**	2.10	1.96	**1.18**	28.22	9.42	**0.20**
couch	2.84	**1.98**	3.24	0.679	0.668	**0.691**	1.02	0.44	**0.14**	26.96	0.79	**0.01**
speak.	**5.46**	5.50	7.30	0.680	0.679	**0.710**	0.40	0.14	**0.04**	22.74	0.28	**0.00**
firea.	**1.20**	1.88	2.19	0.975	0.975	**0.977**	2.17	1.51	**0.01**	24.86	6.46	**0.00**
plane	**1.11**	1.20	2.76	0.938	0.939	**0.957**	2.96	2.51	**0.91**	22.75	10.06	**0.04**
chair	**3.80**	5.19	5.87	0.682	0.697	**0.704**	2.01	1.89	**1.63**	22.94	7.85	**0.20**
monit.	1.76	**1.57**	2.27	0.734	**0.737**	0.699	0.76	0.80	**0.02**	25.94	2.83	**0.00**
phone	1.74	**1.40**	2.36	**0.910**	0.765	0.765	0.23	0.05	**0.02**	23.98	0.16	**0.00**
boat	**1.60**	2.36	4.66	0.835	0.838	**0.849**	0.86	0.47	**0.02**	20.90	2.73	**0.00**
lamp	**6.21**	7.00	19.23	0.917	**0.924**	0.923	1.11	2.18	**0.82**	16.81	9.01	**0.20**
bench	2.13	**1.81**	3.02	0.917	0.917	**0.926**	2.41	1.65	**0.60**	24.17	6.95	**0.06**
car	3.00	**2.74**	3.51	0.770	0.781	**0.805**	1.37	0.49	**0.08**	30.67	1.43	**0.01**
cabin	3.69	**3.53**	4.10	**0.900**	0.897	0.896	0.67	0.23	**0.15**	26.64	0.62	**0.01**
mean	4.15	**3.50**	5.54	0.815	0.816	**0.826**	1.72	1.43	**0.71**	24.80	6.03	**0.10**
mean (with Laplace)	4.59	**3.81**	5.25	0.807	0.811	**0.826**	0.47	0.56	**0.38**	13.26	2.02	**0.00**

Table 4.2. Single View Reconstruction: Chamfer Distances (↓). NMF is comparable to baseline methods in terms of chamfer distance.

Category	table	couch	speak.	firea.	plane	chair	monit.	phone	boat	lamp	bench	car	cabin.	mean	mean(w/Lap.)
MeshRCNN[4]	**5.34**	**3.73**	**8.27**	**2.07**	2.27	**5.56**	**4.17**	**3.00**	**3.55**	**13.67**	**3.21**	**3.33**	**5.11**	**4.73**	**5.96**
Pixel2Mesh[3]	6.64	4.48	9.76	2.42	2.71	6.66	5.03	3.57	3.78	16.55	3.80	3.41	5.86	5.48	10.79
AtlasNet-25[6]	8.67	4.97	10.38	2.08	**2.12**	5.77	5.08	3.50	3.62	15.73	3.32	4.06	5.14	5.48	7.76
AtlasNet-sph[6]	8.59	6.57	12.27	3.06	2.48	8.167	8.29	4.47	4.97	17.63	4.50	4.29	4.65	6.67	7.35
NMF (Ours)	10.95	6.20	12.95	4.67	3.70	8.94	7.94	4.88	7.15	26.49	4.85	4.566	5.139	7.82	8.64

Table 4.3. Single View Reconstruction: Normal Consistency (↑). NMF is comparable to best baselines and outperforms it after laplacian smoothing.

Category	table	couch	speak.	firea.	plane	chair	monit.	phone	boat	lamp	bench	car	cabin.	mean	mean(w/Lap.)
MeshRCNN[4]	0.743	0.723	0.717	0.623	0.693	0.708	0.782	**0.848**	0.648	0.655	0.655	0.649	0.730	0.698	0.758
Pixel2Mesh[3]	0.723	0.743	0.761	0.612	0.685	0.703	**0.805**	0.843	0.680	0.643	0.654	0.683	0.745	0.706	0.720
AtlasNet-25[6]	0.813	0.787	0.786	0.969	0.958	0.725	0.680	0.755	0.871	0.918	0.898	0.835	0.778	0.826	0.824
AtlasNet-sph[6]	0.808	**0.798**	0.790	**0.971**	0.962	**0.740**	0.695	0.759	**0.881**	0.923	0.901	**0.838**	0.777	**0.838**	0.836
NMF (Ours)	**0.844**	0.783	**0.792**	**0.971**	**0.963**	0.739	0.696	0.755	**0.881**	**0.925**	**0.932**	0.829	**0.778**	0.829	**0.837**

Table 4.4. Single View Reconstruction: Manifold Vertices (↓). NMF outperforms baselines by a large margin.

Category	table	couch	speak.	firea.	plane	chair	monit.	phone	boat	lamp	bench	car	cabin.	mean	mean(w/Lap.)
MeshRCNN[4]	15.869	3.052	3.055	2.332	2.630	16.292	6.733	0.468	3.503	23.496	21.710	1.586	10.281	9.319	9.319
Pixel2Mesh[3]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AtlasNet-25[6]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AtlasNet-sph[6]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NMF (Ours)	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

Table 4.5. Single View Reconstruction: Manifold Edges (↓). Both AtlasNet and MeshRCNN have severe non-manifold edges

Category	table	couch	speak.	firea.	plane	chair	monit.	phone	boat	lamp	bench	car	cabin.	mean	mean(w/Lap.)
MeshRCNN[4]	34.317	7.644	7.202	9.294	24.357	25.037	10.161	2.599	8.652	34.398	31.411	4.030	12.924	17.783	17.88
Pixel2Mesh[3]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
AtlasNet-25[6]	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3	7.40e3
AtlasNet-sph[6]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NMF (Ours)	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

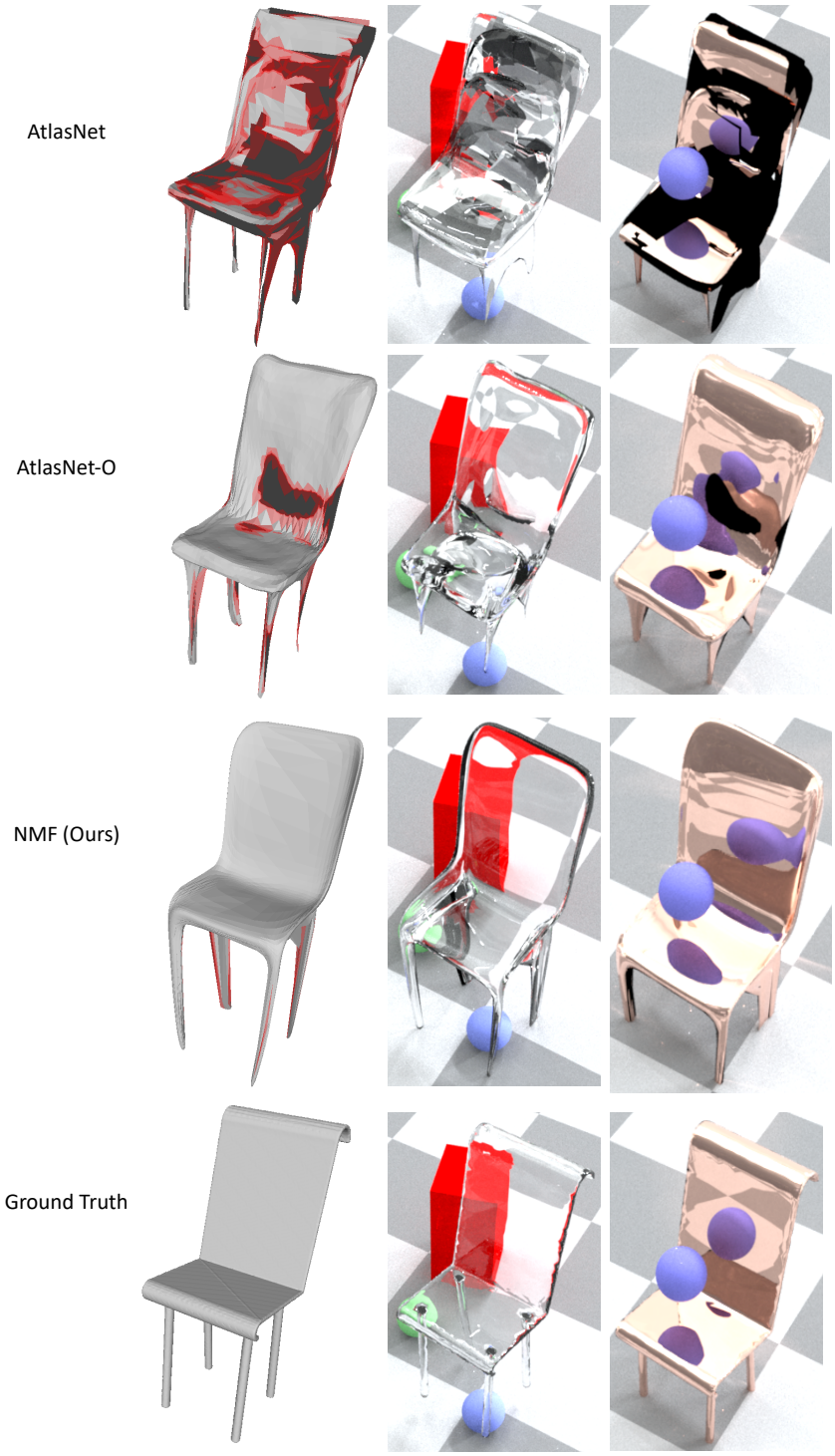


Figure 4.1. Auto-encoder: The first row shows mesh geometry along with self-intersections (red) and flipped normals (black). The bottom row shows results from physically based rendering with dielectric and conducting materials. The appearances of the red box, green ball and blue ball are more realistic for NMF than AtlasNet, since the latter suffers from severe self-intersections and flipped normals.

Table 4.6. Single View Reconstruction: Manifold Faces (↓). NMF is several times better than best baseline

Category	table	couch	speak.	firea.	plane	chair	monit.	phone	boat	lamp	bench	car	cabin.	mean	mean(w/Lap.)
MeshRCNN[4]	5.29	3.69	3.27	7.47	7.74	5.38	3.21	1.19	5.77	7.15	5.74	2.80	3.10	5.18	0.86
Pixel2Mesh[3]	2.74	2.37	2.50	5.80	5.36	3.53	3.04	2.61	4.21	5.39	3.18	2.49	2.75	3.33	0.88
AtlasNet-25[6]	1.77	0.92	0.40	4.14	3.52	1.84	0.756	0.31	1.40	1.72	1.91	1.17	0.50	1.76	0.48
AtlasNet-sph[6]	2.45	1.17	0.88	4.23	3.01	2.54	1.80	0.73	2.73	3.50	2.27	0.67	0.51	2.19	1.08
NMF (Ours)	**1.22**	**0.08**	**0.06**	**0.00**	**1.02**	**1.81**	**0.02**	**0.02**	**0.03**	**1.00**	**0.43**	**0.06**	**0.16**	**0.83**	**0.45**

Table 4.7. Single View Reconstruction: Self-Intersection (↓). NMF is several times better than best baseline

Category	table	couch	speak.	firea.	plane	chair	monit.	phone	boat	lamp	bench	car	cabin.	mean	mean(w/Lap.)
Pixel2Mesh[3]	10.18	10.41	9.81	17.27	18.26	12.07	12.24	10.87	15.16	16.57	11.21	11.34	10.06	12.29	6.52
AtlasNet-25[6]	29.94	28.11	25.46	32.10	29.82	24.69	26.69	27.23	26.30	19.48	27.83	30.14	27.60	26.94	17.57
AtlasNet-sph[6]	12.68	5.25	4.89	20.92	15.56	13.67	12.33	3.29	14.37	12.75	14.95	2.67	2.20	11.07	5.94
NMF (Ours)	**0.24**	**0.00**	**0.00**	**0.00**	**0.07**	**0.26**	**0.00**	**0.00**	**0.00**	**0.26**	**0.05**	**0.00**	**0.01**	**0.12**	**0.00**

smoothing. Our method again results in fewer than 50% non-manifold faces compared to the best baseline. NMF also gets the highest normal consistency performance post smoothing. Due to the *cubify* step as part of the MeshRCNN pipeline which converts a voxel grid into a mesh, the method has several non-manifold vertices and edges compared to deformation based methods Pixel2Mesh, AtlasNet-O and NMF. AtlasNet suffers from the most number of non manifold edges, almost 100 times that of MeshRCNN. We note that MeshRCNN[4] better performance in Chamfer Distance come at a cost of other metrics. We qualitatively show the effects of non-manifoldness in Figure 4.2. We observe that for dielectric material (second row), NMF is able to transmit background colors closest to the ground truth, whereas other baselines only reflect the white sky due to the presence of flipped normals. We also show the improvement in renderings due to smoothing on baselines in Fig. 4.3. Clearly, the rendering results are better after smoothing due to reduction in non-manifoldness, however, it is still not comparable to the quality from NMF. We some more rendering results from NMF in Figure 4.4.

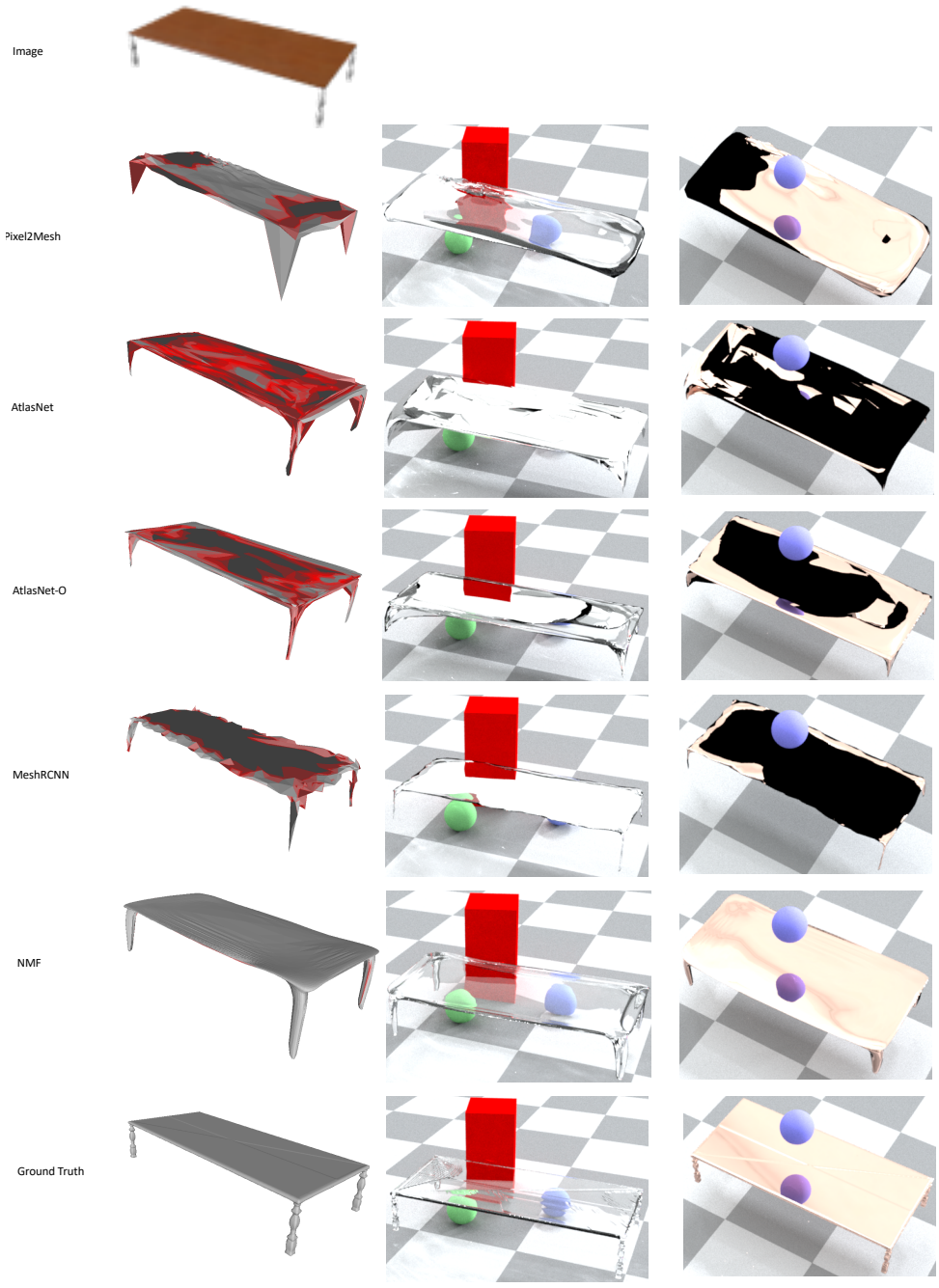


Figure 4.2. Single View Reconstruction: We compare NMF to other mesh generating baselines for SVR. Top row shows mesh geometry along with self-intersections (red) and flipped normals (black). Physically based renders for dielectric and conductor material are shown in rows 2 and 3 respectively. Notice the reflection of checkerboard floor, occluded part of red box and balls are all visible through NMF render but not with other baselines. This is due to the presence of severe self-intersection and flipped normals. The reflection of blue ball on metallic table is more realistic for NMF than other methods.

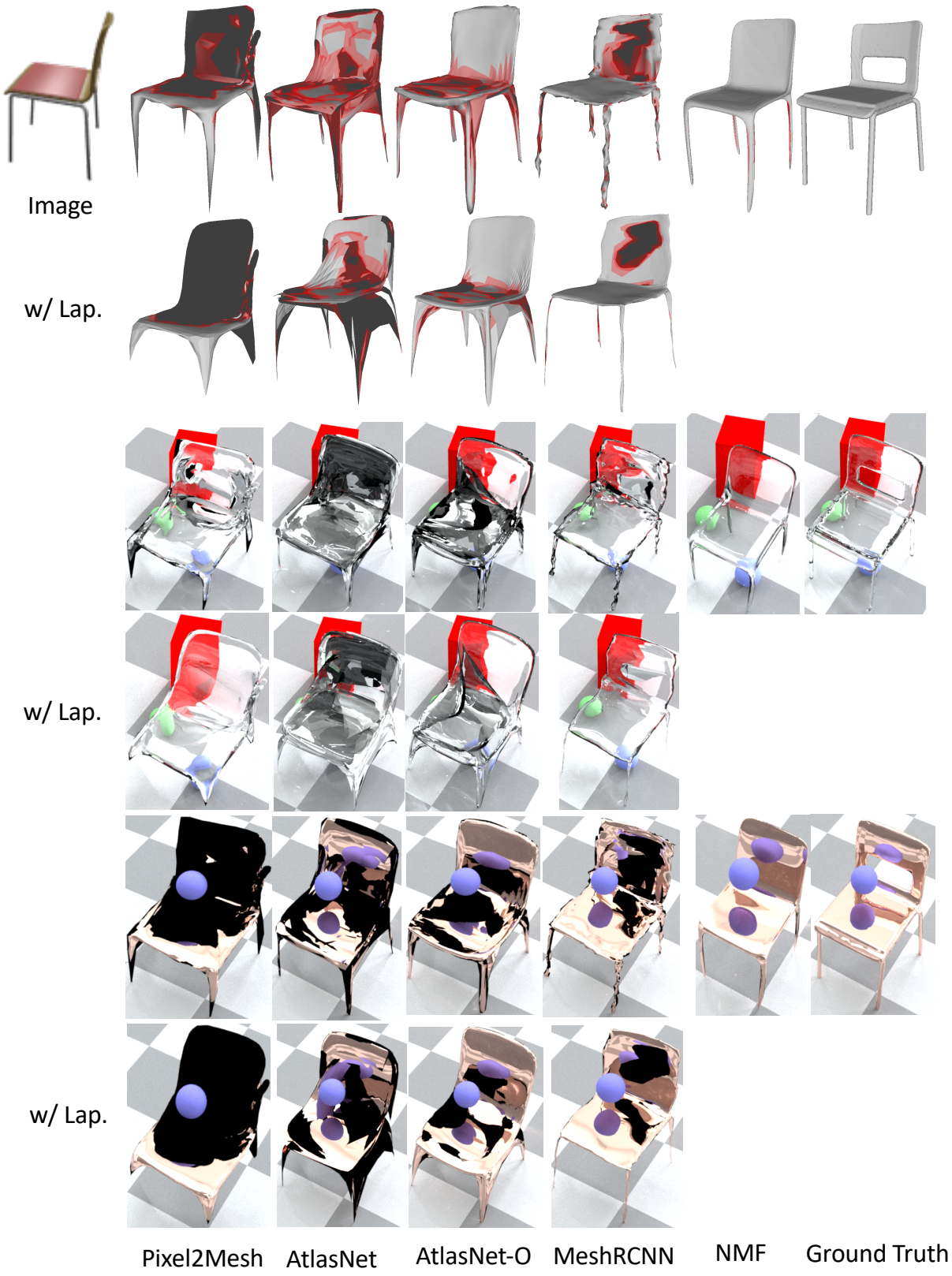


Figure 4.3. Effect of laplacian smoothing (referred by 'w/ Lap.') on rendering quality of baseline methods. Note that NMF without smoothing still gives superior results compared to the bset baseline after smoothing.

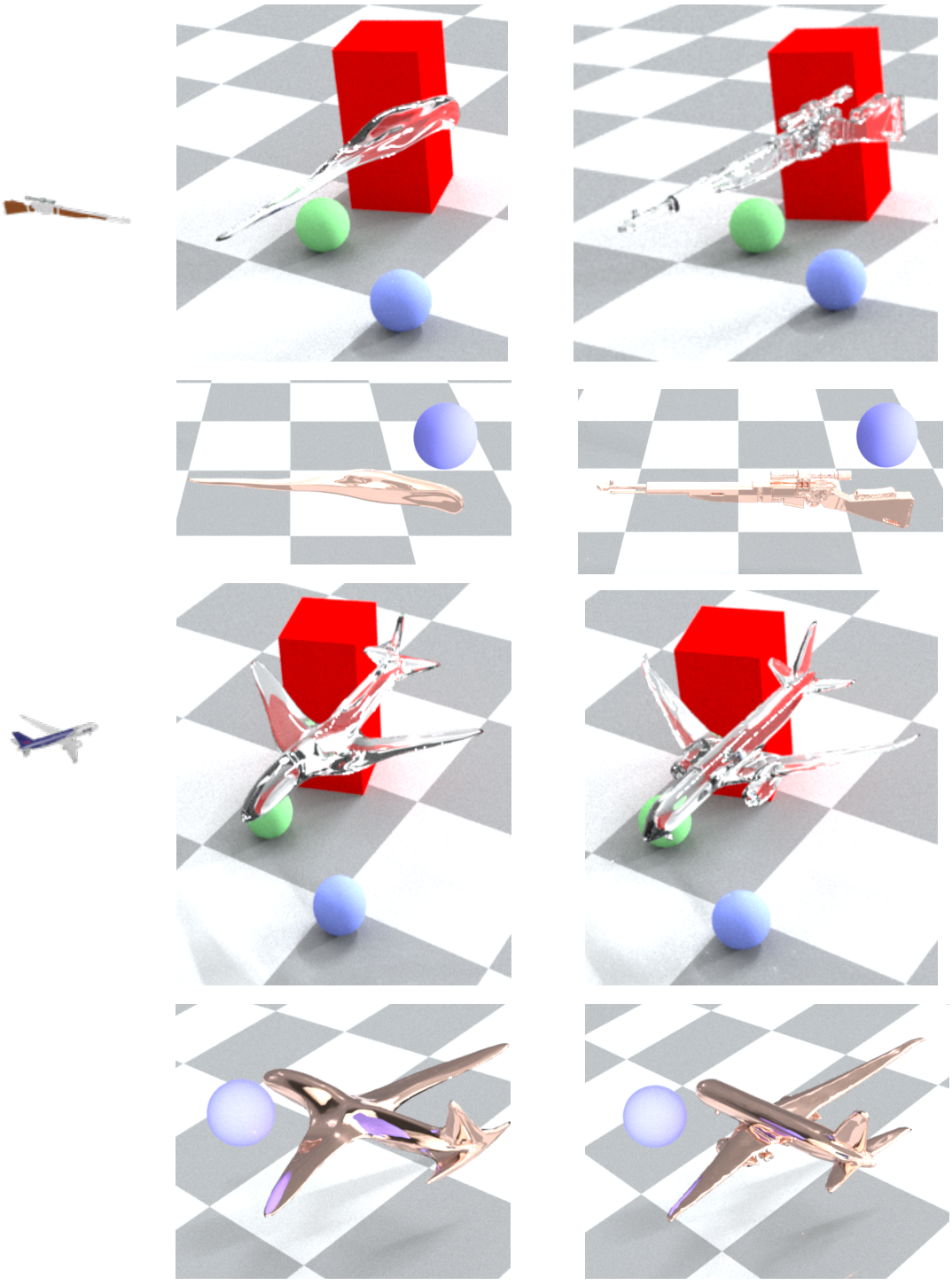


Figure 4.4. Additional Physically based renderings from NMF

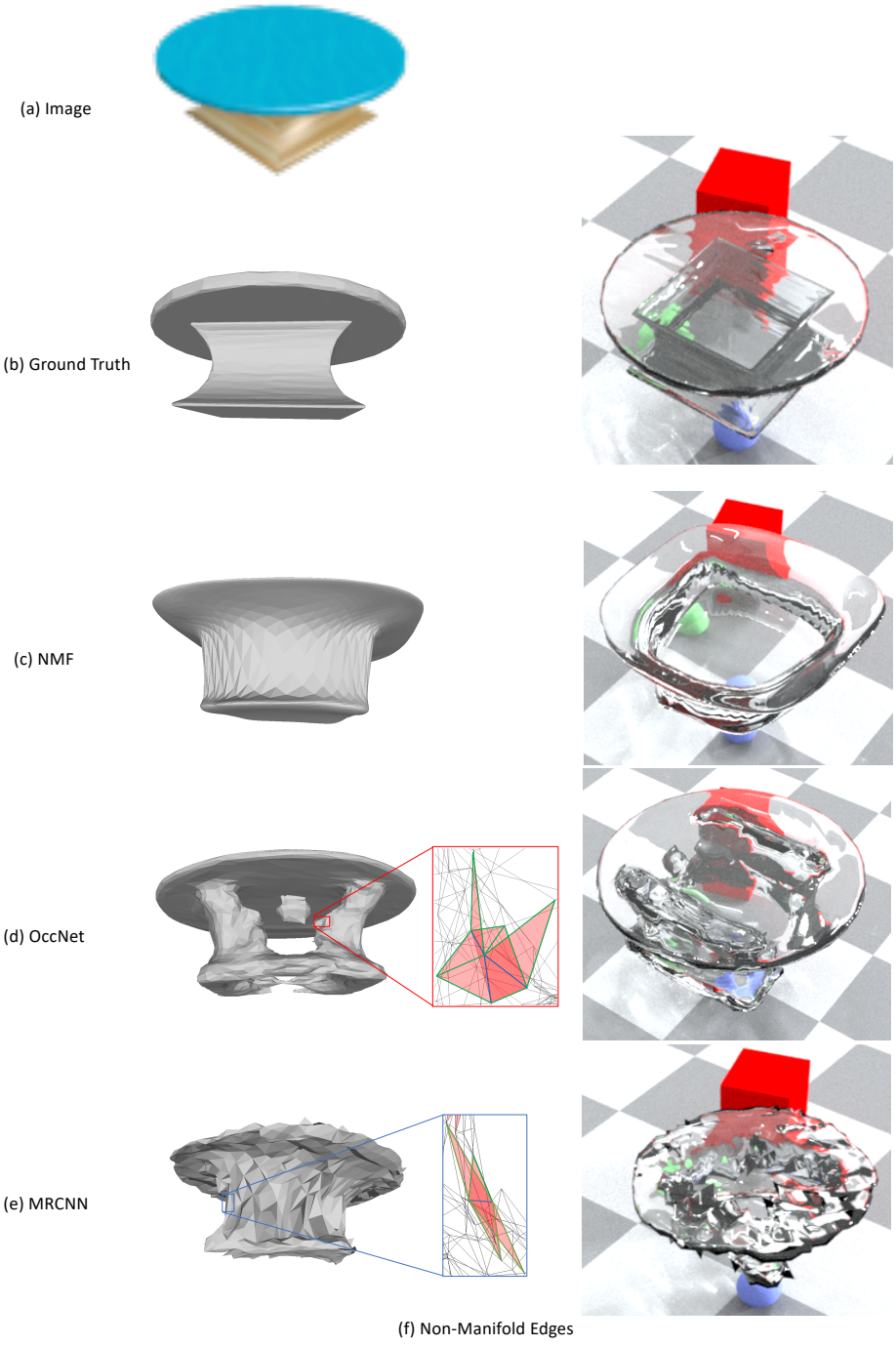


Figure 4.5. Implicit Methods: OccNet fails to give meshes that are singly connected and MeshRCNN has poor normal consistency along with severe self-intersections. Both OccNet and MeshRCNN have non-manifold edges (shown as zoomed out insets). NMF generates meshes that are visually appealing with higher manifoldness.

Table 4.8. Comparison with Implicit Representation method

Point Completion	Chamfer-L2 (↓)	Normal Consistency (↑)	NM-Vertices (↓)	NM-Edges (↓)	NM-Faces (↓)	Self-Intersection (↓)	Time (↓)
OccNet-1[10]	8.77	0.804	1.13	0.85	0.36	**0.00**	795
OccNet-2[10]	2.82	0.804	5.00	3.75	0.28	0.02	1622
OccNet-3[10]	**2.69**	0.805	6.74	3.74	**0.23**	0.08	7973
NMF	5.53	**0.826**	**0.00**	**0.00**	0.71	0.10	**189**
NMF w/ Laplace	5.25	0.825	0.00	0.00	0.38	**0.00**	294
Single View Recon.	Chamfer-L2 (↓)	Normal Consistency (↑)	NM-Vertices (↓)	NM-Edges (↓)	NM-Faces (↓)	Self-Intersection (↓)	Time (↓)
OccNet-1[10]	8.77	0.814	1.13	0.85	0.36	**0.00**	871
OccNet-2[10]	8.66	0.814	2.67	1.79	0.21	0.03	1637
OccNet-3[10]	8.33	0.814	2.79	1.90	**0.15**	0.09	6652
NMF	**7.82**	**0.829**	**0.00**	**0.00**	0.83	0.12	**187**
NMF w/ Laplace	8.64	0.837	0.00	0.00	0.45	**0.00**	292

4.6 Comparison with Implicit Representation

We evaluate NMF against state-of-the-art indirect mesh generation method OccNet for the task of single view reconstruction. Since Mean errors over ShapeNet categories are reported in Table 4.8 and qualitative results are shown in Fig 4.5. We observe that NMF outperforms the best baseline OccNet-3 in terms of geometric accuracy. This is primarily because NMF predicts a singly connected mesh object as opposed to OccNet which leads to several disconnected meshes. Moreover, due to the limitations imposed by the marching cubes algorithm discussed in section 2, OccNet-1,2,3 have several non-manifold vertices and edges where as by construction, NMF doesn't suffer from such limitation. An example of non-manifold edge is shown in figure 4.5. For sake of completeness, we also show the mesh generated by Mesh R-CNN that also suffers from non-manifold vertices and edges. NMF is also competitive with OccNet in terms of self-intersections since with Laplacian smoothing both methods practically become intersection free. While OccNet, outperforms NMF in terms of non-manifold faces, we argue that this comes at a cost of higher inference time. For reference, the fastest version of OccNet has comparable non-manifold faces and self-intersections but fares behind in terms of other metrics.

4.7 Soft Body Simulation

Physically based tasks like rendering, simulation and 3D printing require meshes to be manifold. Neural Mesh Flow learns to generate manifold meshes by construction since it

models a diffeomorphic flow and thereby maintains the uniqueness and orientation preserving properties[15, 14]. However the other mesh generation methods AtlasNet[6], Pixel2Mesh[3, 4], MeshRCNN[4] and OccNet[19] fail to generate meshes that satisfy these *manifoldness* properties. In our supplementary video, we perform qualitative comparisons amongst the mesh generation methods for these physically based tasks.

One of the advantages of a manifold mesh is that it allows us to do physically based simulations. In this experiment, we specifically take the challenging task of simulating the dropping of a mesh on a floor. Amongst other things, this is a challenging task because all the mesh components must support the stress and strain on the mesh as a whole and should result in the solutions to dynamic equations that best represent the reality. The simulations were performed using [62] with settings $pull = 0.9, push = 0.9, bending = 10$ to represent a rubber like material.

We show the 3D meshes (i) and their final form after hitting the ground (ii) in figure 4.6. It is interesting to note that AtlasNet[6] (Fig 4.6 (a)) consisting of 25 mesh patches, while giving good geometric accuracy, disintegrates into independent parts since the collision dynamics are solved for each individual meshlet and therefore the results are far from the ground truth (Fig 4.6(f)). On the contrary AtlasNet-O[6] is able to retain the mesh structure but due to severe self-intersections, the collision simulation is unrealistic and the amount of self-intersections increase after hitting the ground, which shows that merely having the correct mesh geometry is not enough for physically realizable meshes, instead it should also have *manifoldness*. While Pixel2Mesh[3, 23](Fig 4.6 (c)) also suffers simulation artifacts from self-intersections, we note that its mesh contains very few and sparse set of vertices to represent important shape features (like legs). Because of this *non-manifoldness* we encounter strange simulation behaviours such as the legs going through the floor (Fig 4.6 (c)) which is unrealistic. MeshRCNN[23] is found to suffer from *over-bounciness* of its meshes during simulation. We believe this is because of its poor normal consistency which causes issues when solving contact force equations. Neural Mesh flow, (Fig 4.6(e)) due to its high manifoldness gives realistic simulations that are close to

the ground truth (Fig 4.6(f)) which demonstrates its effectiveness and reinforces our hypothesis that *manifoldness* is key to physically realizable meshes. We also tested simulation for OccNet [19] but found that it crashed the simulator due to the presence of several non-manifold vertices and faces.

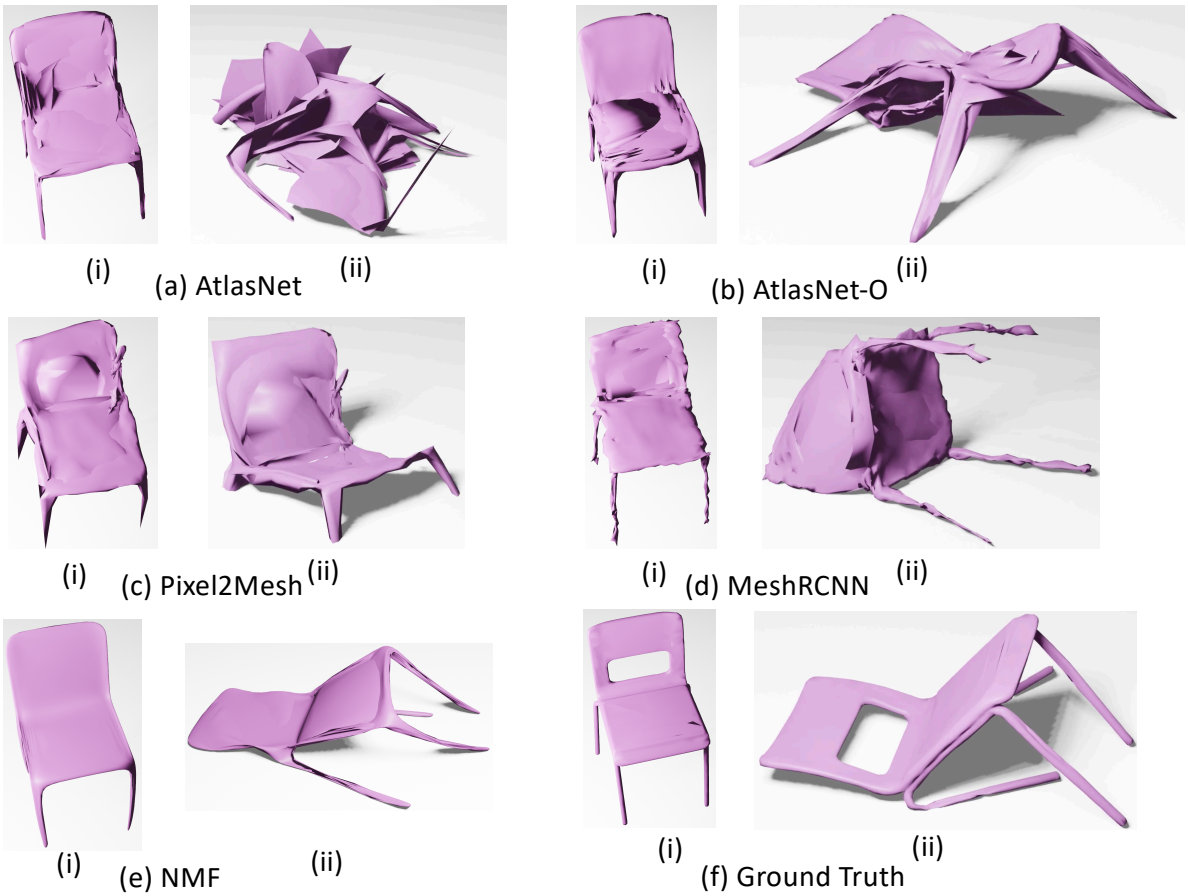


Figure 4.6. Qualitative results for soft body simulation. While AtlasNet (a) breaks down into 25 meshlets, AtlasNet-O (b) suffers from severe self-intersections leading to unrealistic simulations. Pixel2Mesh (c) owing to higher degree of non-manifold faces, leads to artifacts such has the chair going through the floor. MeshRCNN (d) has a high degree of non-manifoldness resulting into unrealistic simulation. NMF(e) due to being a manifold mesh, is close to the ground truth (f)

4.8 3D Printing

We show a few renders of a 3D printed shape. The shape was generated from NMF (using image from Fig. 3.1) without performing any post processing to the prediction. It is important to note that printing other methods require significant human inputs owing to high degrees of non-manifold issues. Fig 4.7 shows a 3D printed bench that was generated by NMF. Not only does it aids rapid 3D printing technology, the results thus obtained are very satisfactory.

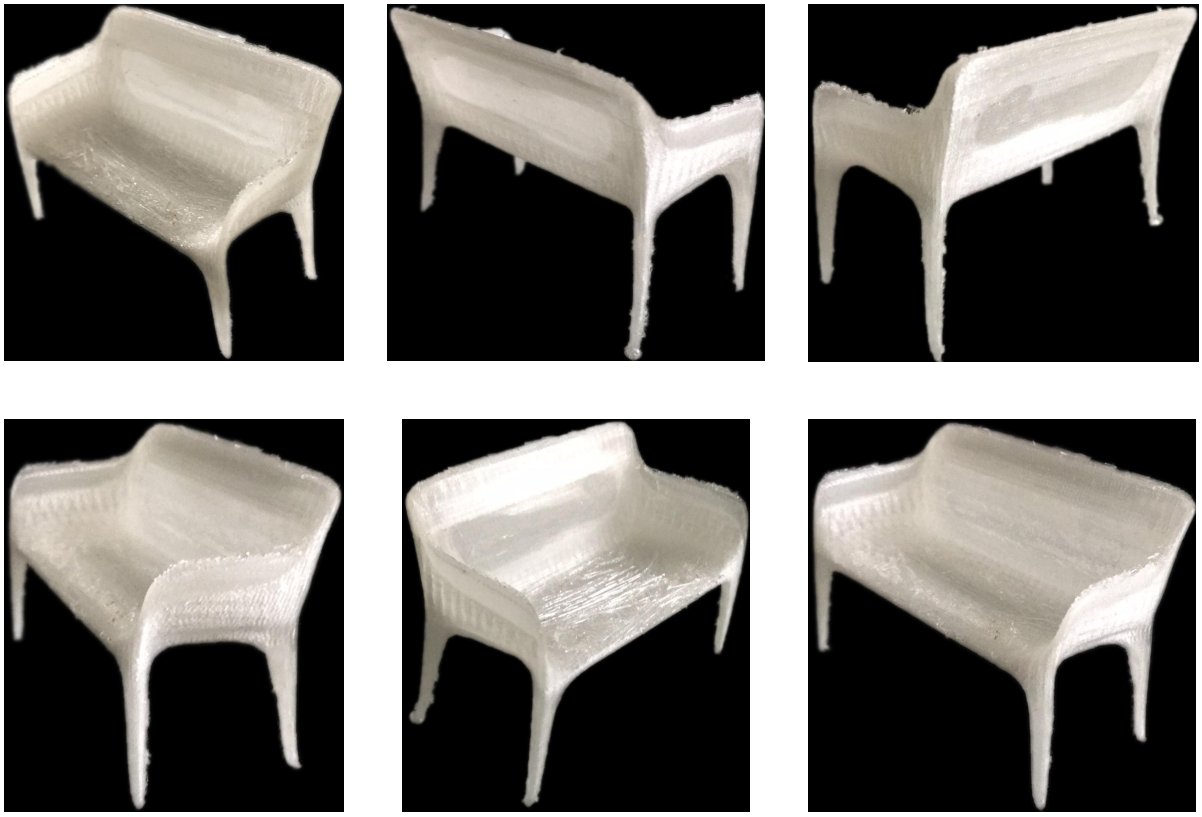


Figure 4.7. Example of a 3D printed shape generate by NMF. The printed models are geometrically accurate to the source image and were printed directly from NMF's mesh prediction without any post-processing/repair

4.9 More Qualitative Results

In this section we show additional applications that are enabled by NMF without any changes to its architecture. These mainly include texture mapping, global shape parameterization, shape deformation and correspondence.

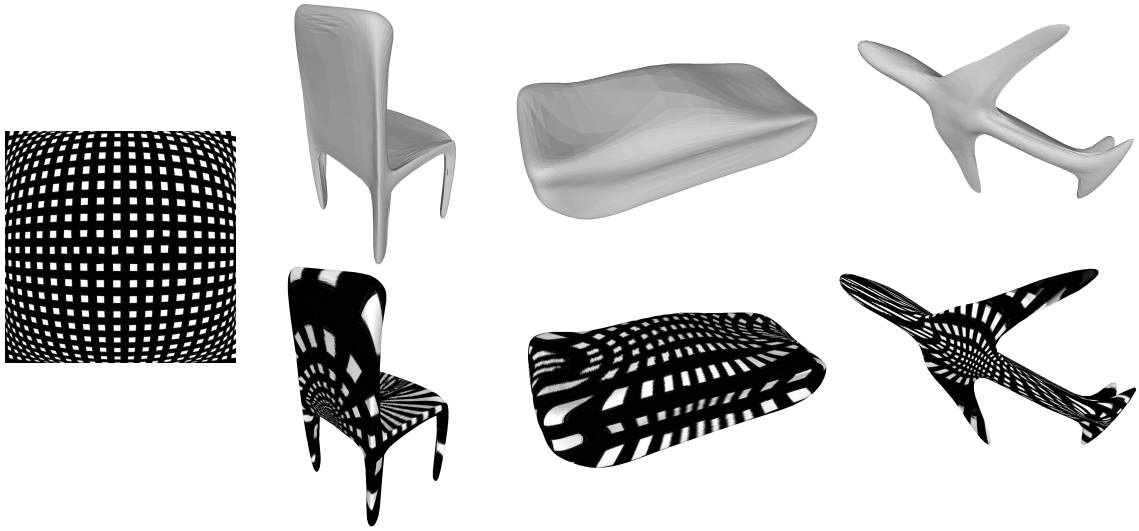


Figure 4.8. Texture Mapping using NMF. We observe minimal distortion while applying textures.

4.9.1 Texture Mapping and Parameterization

One of the important problems in graphics research is that of global shape parameterization which is often used to carry out texture mapping. Since NMF learns to diffeomorphically flow a spherical mesh to a target shape, it retains the local geodesics. This allows us to take a spherical texture (Fig 4.8(a)) and map it to generated shapes (Fig. 4.8(b-c)) without any human inputs. As we can qualitatively observe, the texture mapping is satisfactorily without any artifacts and distortions.

4.9.2 Shape Deformation, Interpolation

For any shape auto-encoder that strives to learn meaningful representations it is important to enable smooth latent space interpolations and have knowledge sharing across generated shapes. For the specific case of NMF that learns to diffeomorphically map a sphere to the target shape, given its shape embedding, the problems of shape deformation and latent space interpolation are identical. To this end, given two shape M_0, M_1 we feed them the point encoder to get their respective shape embeddings z_0, z_1. By linearly interpolating $z = \lambda z_0 + (1-\lambda) z_1, \lambda \in [0,1]$ we can get continuous and manifold intermediate shape deformations and interpolations. While

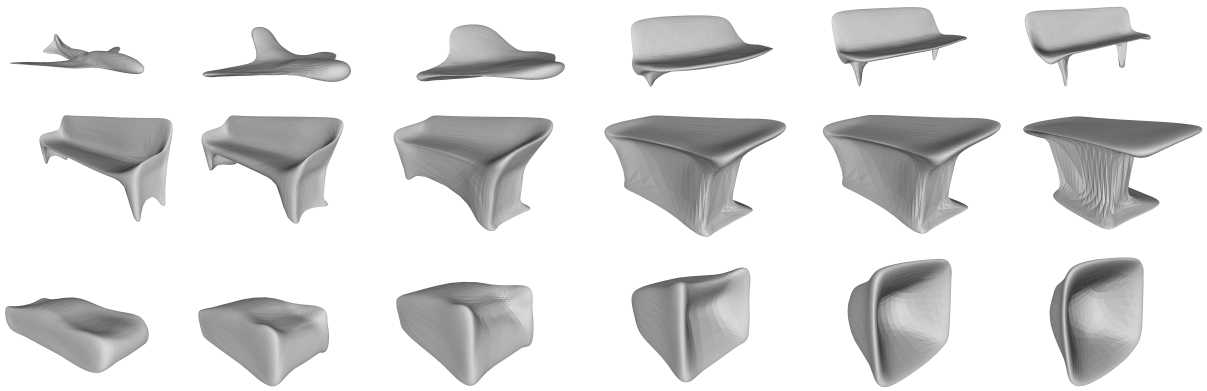


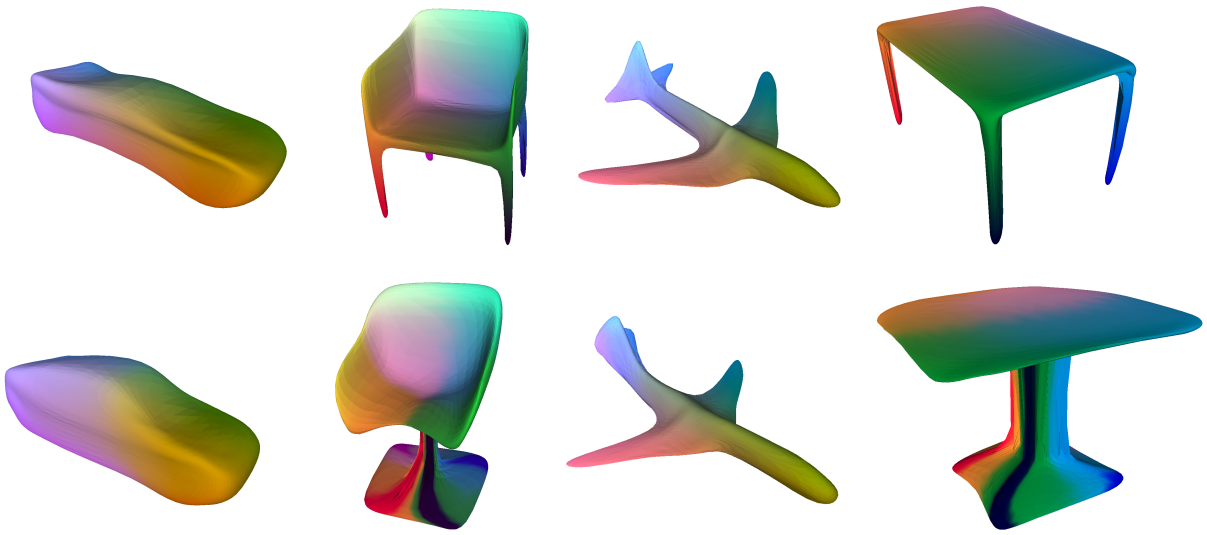
Figure 4.9. Latent space interpolation of NMF. Notice that the interpolated meshes are also manifold.

we show several such deformations in out **supplementary video**, we illustrate a few more interpolations in Fig 4.9 where we observe cross category interpolations that retain manifoldness property at each intermediate step.

4.9.3 Semantic Correspondence

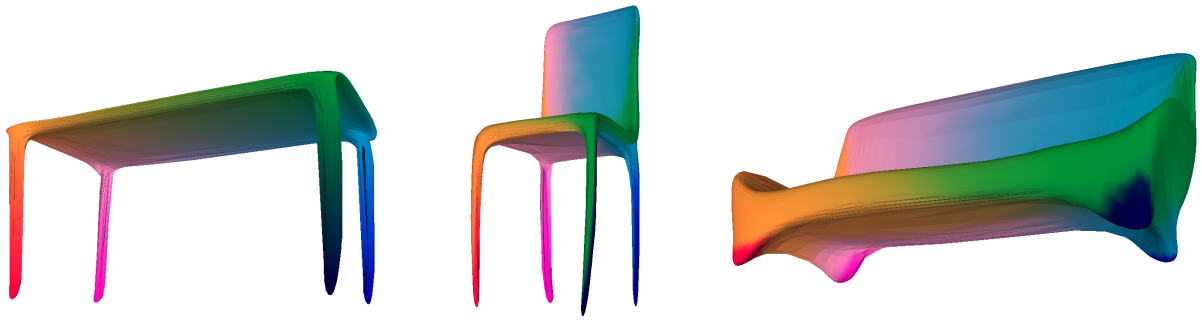
One of the consequence of having smooth interpolations is that NMF is able to learn part correspondence across instances in a category (Fig 4.10) as well as through instances belonging to different categories (Fig 4.11). It is important to mention that this is purely a consequence of NMF architecture and learning such semantic correspondences does not require any explicit training. In Fig. 4.10 (a) we note that the front and back part of the cars (including the wheels) have the same color which implies that they are semantically correlated. Similarly, the wings and tail of airplane (Fig4.10(c)) are semantically correlated among the two instances. Interestingly, for shapes where there is significant change in geometry (Fig 4.10(b,d)) such as a table/chair having four legs and not, we observe that NMF still maps the legs in the initial shape (top) to the base of the target shapes (bottom) that act as *pseudo-legs*. We observe NMF's ability to learn semantic correspondence even across categories (Fig 4.11). The legs of table (Fig 4.11(a)) are semantically mapped to the legs of a chair (Fig 4.11(b)) and even to a bench (Fig. 4.11(c)).

Thus, the above observations indicate that NMF learns really meaningful latent space for 3D manifold shapes.



(a) Car (b) Chair (c) Airplane (d) Table

Figure 4.10. Semantic correspondence learned by NMF without supervision. (a) the back and front of the cars are consistent. (b) even though the bottom chair does not have typical legs, we see that the coloring is semantically consistent and meaningful. The same follows for other categories like airplane (c) and table (d)



(a) Table (b) Chair (c) Bench

Figure 4.11. Cross-Category semantic correspondence learned by NMF without supervision. Note that the color of the 4 legs are consistent across shapes belonging to different categories.

Broader Impact

The broader positive impact of our work would be to inspire methods in computer graphics and associated industries such as gaming and animation, to generate meshes that require significantly less human intervention for rendering and simulation. The proposed NMF method addresses an important need that has not been adequately studied in a vast literature on 3D mesh generation. While NMF is a first step in addressing that need, it tends to produce meshes that are over-smooth (also reflected in other methods sometimes obtaining greater geometric accuracy), which might have potential negative impact in applications such as manufacturing. Our code, models and data will be publicly released to encourage further research in the community.

This thesis, in full, has been submitted for publication of the material as it may appear in a conference, 2020, Kunal Gupta, Manmohan Chandraker. The thesis author was the primary investigator and author of this paper.

Bibliography

[1] L. Custodio, S. Pesco, and C. Silva, "An extended triangulation to the marching cubes 33 algorithm," *Journal of the Brazilian Computer Society*, vol. 25, no. 1, pp. 1–18, 2019.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[3] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in *European Conf. on Computer Vision*, pp. 52–67, 2018.

[4] G. Gkioxari, J. Malik, and J. Johnson, "Mesh r-cnn," in *IEEE Int. Conf. on Computer Vision*, pp. 9785–9795, 2019.

[5] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Neural Information Processing Systems*, pp. 6571–6583, 2018.

[6] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "Atlasnet:a papier-mâché approach approach to learning 3d surface generation," *arXiv preprint arXiv:1802.05384*, 2018.

[7] E. J. Smith, S. Fujimoto, A. Romero, and D. Meger, "Geometrics: Exploiting geometric structure for graph-encoded objects," *arXiv preprint arXiv:1901.11461*, 2019.

[8] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," in *European Conf. on Computer Vision*, pp. 628–644, Springer, 2016.

[9] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 605–613, 2017.

[10] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

[11] B. Ma, H. Liu, L. Nan, and Y. Cong, "An end-to-end geometric deficiency elimination algorithm for 3d meshes," *arXiv preprint arXiv:2003.06535*, 2020.

[12] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural odes," *arXiv preprint arXiv:1904.01681*, 2019.

[13] H. Zhang, X. Gao, J. Unterman, and T. Arodz, "Approximation capabilities of neural ordinary differential equations," *arXiv preprint arXiv:1907.12998*, 2019.

[14] J. Arango and A. Gómez, "Flows and diffeomorphisms," *Revista Colombiana de Matemáticas*, vol. 32, no. 1, pp. 13–27, 1998.

[15] M. K. Fort, "The embedding of homeomorphisms in flows," *Proceedings of the American Mathematical Society*, vol. 6, no. 6, pp. 960–967, 1955.

[16] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama, "Dissecting neural odes," *arXiv preprint arXiv:2002.08071*, 2020.

[17] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan, "Pointflow: 3d point cloud generation with continuous normalizing flows," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 4541–4550, 2019.

[18] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Occupancy flow: 4d reconstruction by learning particle dynamics," in *IEEE Int. Conf. on Computer Vision*, pp. 5379–5389, 2019.

[19] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

[20] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

[21] M. Michalkiewicz, J. K. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson, "Deep level sets: Implicit surface representations for 3d shape inference," *arXiv preprint arXiv:1901.06802*, 2019.

[22] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 5939–5948, 2019.

[23] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "Meshcnn: a network with an edge," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[24] J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman, and J. Tenenbaum, "Marrnet: 3d shape reconstruction via 2.5 d sketches," in *Neural Information Processing Systems*, pp. 540–550, 2017.

[25] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum, "Learning shape priors for single-view 3d completion and reconstruction," in *European Conf. on Computer Vision*, pp. 646–662, 2018.

[26] X. Zhang, Z. Zhang, C. Zhang, J. Tenenbaum, B. Freeman, and J. Wu, "Learning to reconstruct shapes from unseen classes," in *Neural Information Processing Systems*, pp. 2257–2268, 2018.

[27] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta, "Learning a predictable and generative vector representation for objects," in *European Conf. on Computer Vision*, pp. 484–499, Springer, 2016.

[28] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Neural Information Processing Systems*, pp. 2074–2082, 2016.

[29] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.

[30] C. Häne, S. Tulsiani, and J. Malik, "Hierarchical surface prediction for 3d object reconstruction," in *2017 International Conference on 3D Vision (3DV)*, pp. 412–420, IEEE, 2017.

[31] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," in *IEEE Int. Conf. on Computer Vision*, pp. 2088–2096, 2017.

[32] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 206–215, 2018.

[33] J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman, and J. Tenenbaum, "Marrnet: 3d shape reconstruction via 2.5 d sketches," in *Neural Information Processing Systems*, pp. 540–550, 2017.

[34] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum, "Learning shape priors for single-view 3d completion and reconstruction," in *European Conf. on Computer Vision*, pp. 646–662, 2018.

[35] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *Neural Information Processing Systems*, pp. 82–90, 2016.

[36] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction," *arXiv preprint arXiv:2003.10983*, 2020.

[37] T. Rashid, S. Sultana, and M. A. Audette, "Watertight and 2-manifold surface meshes using dual contouring with tetrahedral decomposition of grid cubes," *Procedia engineering*, vol. 163, pp. 136–148, 2016.

[38] S. Raman and R. Wenger, "Quality isosurface mesh generation using an extended marching cubes lookup table," in *Computer Graphics Forum*, vol. 27, pp. 791–798, Wiley Online Library, 2008.

[39] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, "Keep it smpl: Automatic estimation of 3d human pose and shape from a single image," in *European Conf. on Computer Vision*, pp. 561–578, Springer, 2016.

[40] S. Zuffi, A. Kanazawa, D. W. Jacobs, and M. J. Black, "3d menagerie: Modeling the 3d shape and pose of animals," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 6365–6373, 2017.

[41] S. Zuffi, A. Kanazawa, and M. J. Black, "Lions and tigers and bears: Capturing non-rigid, 3d, articulated shape from images," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 3955–3963, 2018.

[42] S. Zuffi and M. J. Black, "The stitched puppet: A graphical model of 3d human shape and pose," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 3537–3546, 2015.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[44] W. Wang, D. Ceylan, R. Mech, and U. Neumann, "3dn: 3d deformation network," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1038–1046, 2019.

[45] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

[46] L. S. Pontryagin, *Mathematical theory of optimal processes*. Routledge, 2018.

[47] H. Yan, J. Du, V. Y. Tan, and J. Feng, "On robustness of neural ordinary differential equations," *arXiv preprint arXiv:1910.05513*, 2019.

[48] W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," *arXiv preprint arXiv:1810.01367*, 2018.

[49] H. Nam and H.-E. Kim, "Batch-instance normalization for adaptively style-invariant neural networks," in *Neural Information Processing Systems*, pp. 2558–2567, 2018.

[50] Y. Wu and K. He, "Group normalization," in *European Conf. on Computer Vision*, pp. 3–19, 2018.

[51] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[52] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Neural Information Processing Systems*, pp. 901–909, 2016.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

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

[55] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "Pytorch3d." https://github.com/facebookresearch/pytorch3d, 2020.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conf. on Computer Vision*, pp. 630–645, Springer, 2016.

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

[58] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[59] J. Huang, H. Su, and L. Guibas, "Robust watertight manifold surface generation method for shapenet models," *arXiv preprint arXiv:1802.01698*, 2018.

[60] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[61] T. Karras, "Maximizing parallelism in the construction of bvhs, octrees, and k-d trees," in *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, pp. 33–37, Eurographics Association, 2012.

[62] R. Hess, *Blender Foundations: The Essential Guide to Learning Blender 2.6*. Focal Press, 2010.