NIGnets and Neural ODEs for Representing Non-Self-Intersecting Geometry

Atharva Aalok
Stanford University
Stanford, California
atharva1@stanford.edu

Abstract

We develop a new shape representation scheme that gives us a hard guarantee on representing only non-self-intersecting geometry. We achieve this by representing our shape as the solution of an Initial Value Problem Ordinary Differential Equation. We start with an initial shape, a circle, which is modified by a NeuralODE to represent a final target shape. In particular this is a diffeomorphic flow and therefore preserves the non-self-intersecting property.
We discuss how the need for such a representation scheme guaranteeing non-self-intersection arises quite naturally in many engineering domains especially in shape optimization. We then discuss another geometry representation, NIGnets, we developed that also gives such a guarantee. Next, a through discussion of the NeuralODE based shape representation is provided and compared with NIGnets. We perform a wide range of experiments comparing the performance of NeuralODE and NIGnets based on training time, inference speed, accuracy of representation and ability to work with a variety of different geometric loss functions. We perform these experiments on a small dataset that we curated consisting of 2D shapes. Finally, ideas to further increase the representation power of these methods are discussed.

1. Introduction

Shape optimization is the study of designing shapes that minimize or maximize some quantity of interest. An example would be designing the wing of an airplane that maximizes the lift-to-drag ratio. A typical shape optimization process loop consists of three steps:

Shape Representation Using a parameterization method to represent the shape. The parameters ϕ are the design variables. E.g. splines with their control points as parameters.

Shape Evaluation Evaluating some characteristic of the shape that is to be minimized or maximized. E.g. lift-to-drag ratio of a wing.

Shape Improvement Changing the parameters ϕ to design shapes that achieve better characteristics. E.g. gradient descent to optimize ϕ.

We will focus on shape representation.

In a large class of shape optimization problems the shapes of interest are simple closed curves. Simple closed curves, also called Jordan curves are curves that are closed, i.e. they form loops and are simple, i.e. they do not self-intersect.

To understand why an optimization problem might be concerned only with simple closed curves consider the following problem: Imagine that there is a flow going left $\rightarrow$ right around the body shown in Figure 1, and consider the lift-to-drag ratio of this shape. The flow only sees the boundary of this shape, the complicated internal representation does not have any effect whatsoever on the properties of the body. All the representation used to describe the inner curves is wasteful.

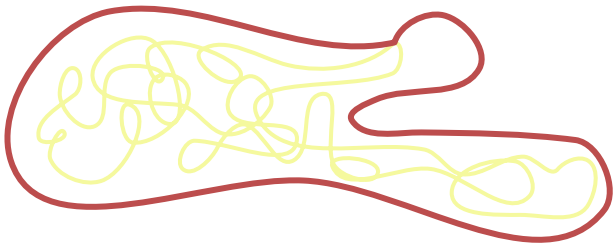

Figure 1. An incoming flow would only interact with the boundary. The complicated internal mess is wasteful representation.

This is especially crucial when doing optimization. Shape representation methods that can potentially represent self-intersecting shapes would cause the optimization algorithm to search a bigger design space than needed. Also, self-intersecting shapes might be physically undesirable or problematic to deal with in downstream tasks such as shape evaluation. Thus, such a shape parameterization would require manual tuning during optimization. As an e.g. con-

sider the optimization happening in Figure 2. We represent the shape using 12 spline control points which are then fed into a neural network that predicts the shape's lift-to-drag ratio. Essentially, we have trained a surrogate model to mimic a Computational Fluid Dynamics (CFD) solver. During training we used non-self-intersecting shapes as they are the ones of interest, but during optimization, if after a gradient step the spline starts intersecting the network struggles to predict its lift-to-drag ratio and steers the optimization in an even worse direction. This is the classical distributional shift problem.

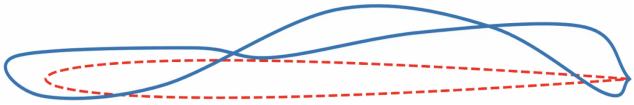

Figure 2. Optimization starts from the dashed red initial airfoil shape which is iteratively modified. We see that the optimization process suffers from distributional shift. Once a self-intersecting shape is reached it is iteratively made even worse.

When using shape parameterization that can represent self-intersecting shapes the optimization algorithm's search has to be made limited, or someone has to sit and manually tune and check for self-intersection. Sometimes additional losses are added to the objective to prevent self-intersection. This is a hassle. In effect prevents an automated and aggressive shape space exploration. In an ideal setting, one would like to leave gradient descent running and go to sleep and wake up to find the optimal shape. Thus we need a shape parameterization that has the quality of non-self-intersection in-built and obeys this constraint for any set of parameters.

We are in the process of developing a new neural architecture for shape parameterization, Neural Injective Geometry or NIGnets [2], that can describe general simple closed curves. This method ensures that only non-self-intersecting curves are generated for any combination of the parameters.

Until now the Neural Injective Geometry network architecture has the following components that give it a lot of representation power:

- Injective Networks (The core)
- Monotonic Networks
- Pre and Post Auxilliary Networks

The exact details can be found on the NIGnets website `https://atharvaaalok.github.io/NIGnets/`.

The shape fitting procedure consists of defining a target shape through a point cloud as shown in Figure 3. Points on the curve that our network represents are also obtained as a point cloud, we call this the candidate curve. Then we train our network with an appropriate geometric loss function to measure the dissimilarity between the candidate curve and the target shape. Using this loss we can then tune the parameters ϕ of our network to try and fit the target curve. An example fit is shown in Figure 4.

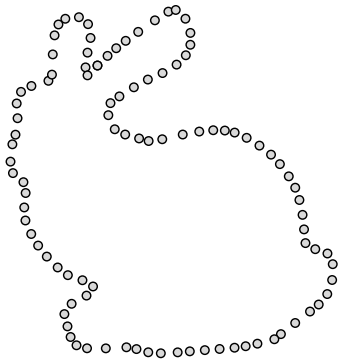

Figure 3. Point cloud representation of the target shape.

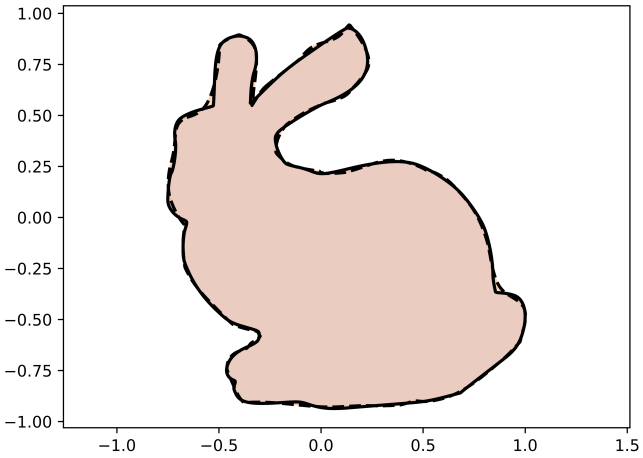

Figure 4. Fitting a target curve using NIGnets

In this study we design and compare another very powerful technique that can either be used independently or add a lot of representational power to the existing NIGnet architecture. It transforms shape representation into the solution of an Initial Value Problem (IVP) for an Ordinary Differential Equation (ODE). We will derive a form of the representation that will make use of Neural Ordinary Differential Equations, allowing us to make use of existing autograd based ODE solvers that use adjoint for backpropagating [6].

2. Related Work

Neural geometry representation has attracted a lot of attention in recent years [11, 10, 7, 15]. Several approaches to deep geometry representations have been studied including point clouds [12], voxel based representations [14] and implicit neural geometry representation using level-sets [11, 10]. Work on mesh deformation based methods has also been done [7] but these methods essentially patch together different meshes, each individually transformed to represent their final geometry. Recent work on representing highly detailed geometry using Lagrangian approaches has produced great results but these methods use regular-

ization to represent non-self-intersecting, making it a soft constraint [8]. Cheng *et al*. [4] have developed a new approach to representing geometry by using text prompts and transforming them to a set of valid CAD commands. But there is no guarantee on the resulting CAD commands being valid and may not compile. Also, another problem with this approach is that the resulting CAD commands are a non-differentiable result in the pipeline. This makes it hard to use them in optimization. Work on representing only non-self-intersecting geometry, especially fitting shapes based on only a boundary point cloud is quite new. One recent method was developed by us, NIGnets [2]. NIGnets are an explicit geometry representation method. That transforms the unit interval $[0, 1]$ to the boundary of a simple closed curve, also called a Jordan curve. While NIGnets are very stable during training and extremely fast at inference, these advantages are only well realised when working in 2D. In 3D they start to suffer from the curse of dimensionality and require not only a lot of manual training to fit target shapes but also become very compute and memory intensive. Another method is to use diffeomorphic flows. In this type of representation we start with a simple initial non-self-intersecting shape that is transformed in time according to an ordinary differential equation to produce a final shape. The time interval of integration is kept fixed in $[0, 1]$ and different shapes correspond to different ODEs that the initial shape is evolved according to. This has been taken advantage of in [16], but there the authors focussed on a latent space representation and not on capturing the fine-grained details of the geometry. They also did not use only boundary point clouds for shape fitting. Also, Yang *et al*. [16] considered only 3D point clouds and the performance of NeuralODEs in 2D in terms of representation power, training stability and inference speed has not been studied before. We work on a similar method of using NeuralODEs to fit single target shapes where our focus and performance criteria will be the ability to fit a single shape in great detail. We focus on all the performance criteria and also perform a thorough comparison with NIGnets.

3. Methods

We look at two representation methods that make use of Neural ODEs for shape representation by continuous deformation of an initial shape.

- **Bump and Rotate Strategy**: It's a constrained NeuralODE representation that adds deformations (bumps) while continuously rotating our shape. We first formulate it in a finite deformation setting and then take the limit as the number of deformations got to ∞ to obtain the continuous deformation formulation. This formulation produces only non-self-intersecting geometry in both the finite deformation and continuous deformation settings.
- **Unconstrained NeuralODE**: We then describe how any Neural ODE flow is diffeomorphic and will guarantee only non-self-intersecting geometry provided that our neural network describing the evolution function of the ODE is Lipschitz continuous. This is the full general setting and encompasses the previous bump and rotate strategy. This formulation only works in the continuous deformations setting and does not provide a hard guarantee when using finite number of deformations.

3.1. Bump and Rotate Neural ODE

This shape representation works by adding bumps to an initial simple closed curve. The basic idea is to add the same Δy to the curve points at the same x that ensures non-self-intersection and keep rotating the shape continuously to add deviations in different directions.

The procedure is roughly as follows:

1. Start with an initial simple closed curve which is the circle.
2. Feed that into a neural network that given an input x value outputs y that is how much the points at the curve points at x should be moved in y.
3. Rotate the shape by some angle $\Delta\theta$ and feed it back into the neural network.
4. Repeat for θ till 2π.

Mathematically, Consider $X = [x, y]$. Say we transform X from $X(0)$ repeatedly and we are currently at some θ with $X(\theta)$. Then our operations will transform $X(\theta)$ to $X(\theta + \Delta\theta)$ as follows:

$$X(\theta + \Delta\theta) = R(\Delta\theta)\left[X(\theta) + NN(X(\theta), \theta)\Delta\theta\right]$$

where the following conditions have to apply:

- the x coordinate is not changed, the equation makes changes only to the y value.
- $NN(X(\theta), \theta)$ takes in as input only the x coordinate and θ and does not make use of y.

R is a rotation matrix that rotates $\Delta\theta$. It is given by:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$X(\theta + \Delta\theta) = R(\Delta\theta)X(\theta) + R(\Delta\theta)NN(X(\theta), \theta)\Delta\theta$$

subtract $X(\theta)$ from both sides,

$$X(\theta+\Delta\theta)-X(\theta) = R(\Delta\theta)X(\theta)-X(\theta)+R(\Delta\theta)NN(X(\theta), \theta)\Delta\theta$$

divide by $\Delta\theta$,

$$\frac{X(\theta+\Delta\theta)-X(\theta)}{\Delta\theta} = \frac{(R(\Delta\theta)-I)}{\Delta\theta}X(\theta)+R(\Delta\theta)NN(X(\theta),\theta)$$

take $\lim_{\Delta\theta\to 0}$ on both sides, and also note that $R(0) = I$, hence

$$\frac{R(\Delta\theta)-I}{\Delta\theta} = \frac{R(\Delta\theta)-R(0)}{\Delta\theta} = \dot{R}(0)$$

we get the final equation:

$$\dot{X}(\theta) = \dot{R}(0)X(\theta) + NN(X(\theta),\theta)$$

we see that this fits the form of a NeuralODE [5]. We will use pre-existing NeuralODE frameworks to perform the integration. In particular, we will make use of torchdiffeq [6].

In integration form we have, $X(2\pi) = X(0) + \int_0^{2\pi} \dot{X}(\theta)d\theta$
Also we have,

$$\dot{R}(\theta) = \begin{bmatrix} -\sin\theta & -\cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix}$$

We see that this formulation has non-self-intersection inbuilt both for the finite deformation setting as well as the continuous deformation setting. The downside is that we have a constraint on the

3.2. Unconstrained Neural ODE deformations

We discussed how we can produce continuous deformations on top of an initial shape to produce a final shape. We now discuss a complete generalization of the Neural ODE framework for producing deformations that preserve simple closedness of our curves. In particular we use the following theorem 3.1:

Theorem 3.1 (Picard–Lindelöf Existence–Uniqueness) *Let $D \subseteq \mathbb{R}\times\mathbb{R}^n$ be a closed rectangle with $(t_0, y_0) \in \operatorname{int} D$. Let $f : D \to \mathbb{R}^n$ be a function that is* continuous in t *and* Lipschitz-continuous in y *(with Lipschitz constant independent of t). Then there exists some $\varepsilon > 0$ such that the initial-value problem*

$$y'(t) = f\big(t,\, y(t)\big), \qquad y(t_0) = y_0$$

has a unique solution $y(t)$ on the interval $[t_0-\varepsilon,\ t_0+\varepsilon]$.

The Picard-Lindelöf theorem tells us that any Neural ODE flow where the driving function is Lipschitz continuous (globally) produces globally unique evolutions of different initial conditions. The Lipschitz continuity arises if neural network has finite weights and uses Lipshitz nonlinearities, such as tanh or relu. In our subsequent formulations therefore, we make use of tanh as our non-linearity.

In total, we have the same setup as before where we start from $X(0)$ and evolve it according to a 2/3 dimensional ordinary differential equation of the form:

$$\dot{X}(\theta) = NN(X(\theta),\theta)$$

where, now we no longer have any restriction on the neural network input and outputs as we had in the bump and rotate formulation. We can use any neural network architecture as long as we use non-linearities that are Lipschitz continuous.

It is worth noting not just the unconstrained nature of this representation but also the inherently dimension free nature. In particular, θ no longer represents an angle. It can be any arbitrary variable that is integrated over in an arbitrary interval. In particular, to make these things explicit we replace θ with t and we integrate over t in the interval $[0, 1]$. We will use the torchdiffeq [6] package to integrate our Neural ODEs using a 5^{th} order Runge-Kutta method.

3.3. Bump and Rotate vs Unconstrained Neural ODE

As we discovered that the Unconstrained Neural ODE method encompasses the Bump and Rotate method and its performance is strictly superior (as shown in Figure 5 we will in this study compare NIGnets only with unconstrained Neural ODE representation scheme. Henceforth, references to Neural ODE will mean the unconstrained Neural ODE representation method as described above.

4. Dataset and Features

We have developed a dataset of target shapes [2] that we would like to test our representation method against. The dataset consists of the following breakdown in terms of shape complexity:

- **Low complexity**: like circle, airfoil, heart etc. to test our implementation and check that things work.
- **Intermediate complexity**: like puzzle piece, aircraft, stanford bunny etc.
- **High complexity**: fractal shapes like Minkowski fractal, star fractal, snowflake fractal etc.

A few sample target shapes are shown in Figure 6.

In particular the shapes are created as SVG images, which can be sampled at an arbitrary number of points. For more precise fitting of the shapes we can sample more points while trading off training time due to more compute. The shapes are available here: `https://github.com/atharvaaalok/NIGnets/tree/main/docs/assets/shape_svg` and a preview for fitting using NIGnets to these shapes is available here: `https://atharvaaalok.github.io/NIGnets/showcase`.

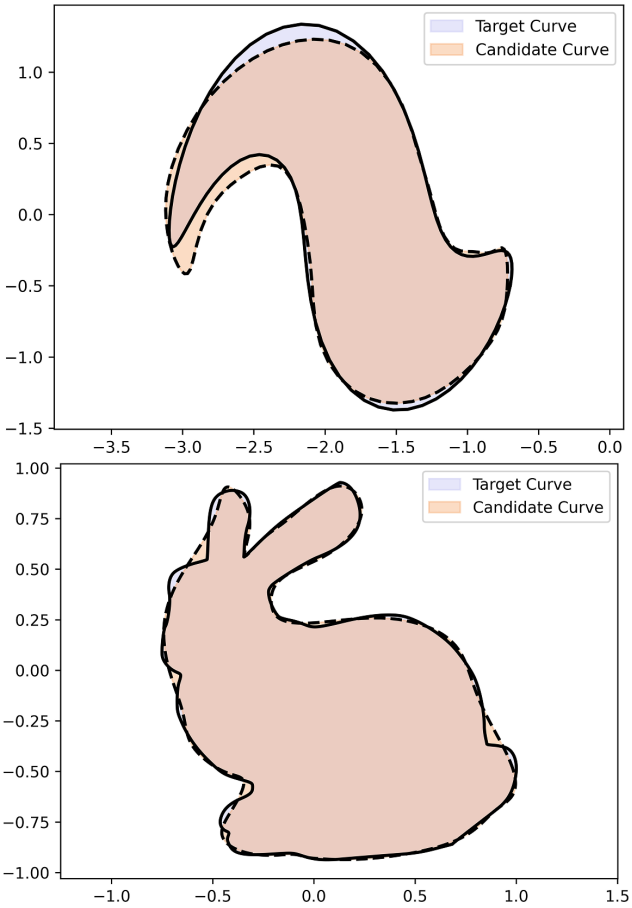

Figure 5. a) Top figure shows Bump and Rotate struggles with even a simple shape. b) Bottom figure shows Neural ODE's strong performance on a higher complexity target shape the the stanford bunny.

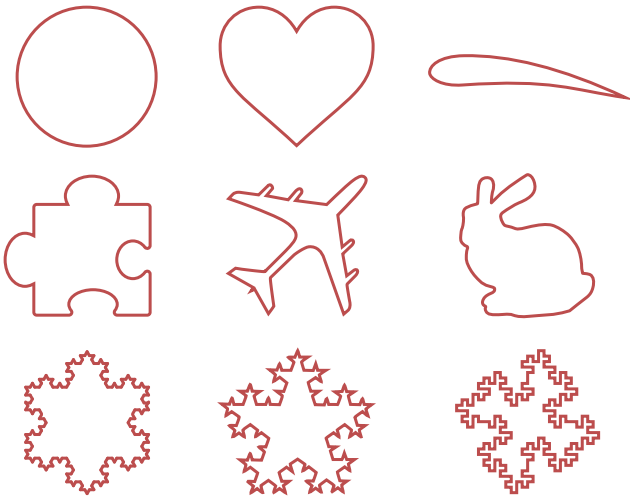

Figure 6. Sample target shapes from the dataset arranged to show low, intermediate and high complexity shapes.

In terms of data-preprocessing we perform centering and uniform scaling in each dimension of the point cloud generated from our images. In particular after we sample a given number of discrete points from our SVG we move its centroid to the origin and scale the image to lie between the unit hypercube, i.e. $[-1, 1]$ in the longest dimension. We perform the same scaling in each dimension so as to not change the aspect ratio of the shape. This is because the goal of our shape networks is to fit the shapes in their undeformed form and be able to handle different length scales in different dimensions.

In this study we compare the performance of NIGnets and Neural ODEs on low and intermediate complexity shapes. Our experiments show that the methods need architectural improvements better they are able to meaningfully tackle the high complexity shapes.

5. Experiments and Results

We perform a wide range of experiments comparing the performance of NeuralODE and NIGnets based on:

- Accuracy of Representation
- Training Time
- Inference Speed
- Ability to work with different Geometric Loss functions.

In particular, we perform experiments on shapes subsampled from our dataset. We discuss each of these performance metric in thorough detail next.

Code for all our experiments is provided here: `https://github.com/atharvaaalok/neural_ode_shape_representation`.

Throughout our training we use Adam as our optimizer with a learning rate of 0.1 for NIGnets and 0.01 for Neural ODEs. We perform training for 10000 epochs for NIGnets and 1000 epochs for Neural ODEs. This is because as we show later NIGnets are roughly 10x faster than Neural ODEs and therefore allows us to get through training epochs faster.

5.1. Accuracy of Representation

The accuracy of representation is a direct metric that tells us how well a particular representation scheme is able to capture the fine-grained details of a target shape. To have a fair comparison, we use the same total number of parameters in our NIGnet and Neural ODE representations and allow them a large enough training time to fit to their maximum capacity. In particular we use a total of 2800 parameters in each of NIGnet and Neural ODE networks. Note that in this comparison we do not consider the same total

Shape	NIGnet	Neural ODE
circle	1.41e-5	1.51e-5
square	4.55e-5	5.2e-5
airfoil	4.60e-6	2.17e-5
heart	5.07e-5	2.61e-5
stanford bunny	3.69e-4	5.35e-4
hand	2.23e-3	1.1e-2
puzzle piece	6.87e-4	3.59e-4
airplane	1.38e-3	1.13e-3

Table 1. A comparison of NIGnets and Neural ODEs based on accuracy of representation measured by the final Chamfer Loss after training.

epochs but rather the total time of training that it takes for the network to fit the target shape. In particular, we perform training for 10000 epochs for NIGnets and 1000 epochs for Neural ODEs. This is because as we show later NIGnets are roughly 10x faster than Neural ODEs and therefore allows us to get through training epochs faster. Since training stability is a major factor in accuracy of representation, we also consider that as a factor in our comparison of representation accuracy. We do this as follows: we initialize randomly each of NIGnets and Neural ODE nets 10 times and fit them to our target shapes and average out the performance when reporting. Also, it must be noted that the reported performance is on Chamfer Loss. Since, Neural ODEs fail to work with MSE Loss (a disadvantage of the approach) we use Chamfer Loss to fit shapes and report that. Results are shown in Table 1.

We observe that NIGnets work better for simpler shapes but the performance on more complex shapes is split. On some of the intermediate complexity shapes like the stanford bunny and hand, NIGnets perform better while on the puzzle piece and airplane Neural ODEs have better accuracy.

Example fits to the stanford bunny by NIGnet and Neural ODE is shown in Figure 7.

5.2. Training Time

To make more sense of the accuracy of representation we provide below the total training time to fit each target shape once in Table 2.

We observe that NIGnets are extremely consistent in their training times, this is because they are an explicity representation method which directly transform an input to an output using an analytically defined function. Whereas, Neural ODEs show a high variance in training time. This is because of the adaptive step size used in numerical integration. The integration time depends on the exact function represented by the ODE network. We observe higher training times for more complex shapes, which aligns with the intuition that these would require more complex ODE networks that need smaller steps during integration to achieve low integration error.

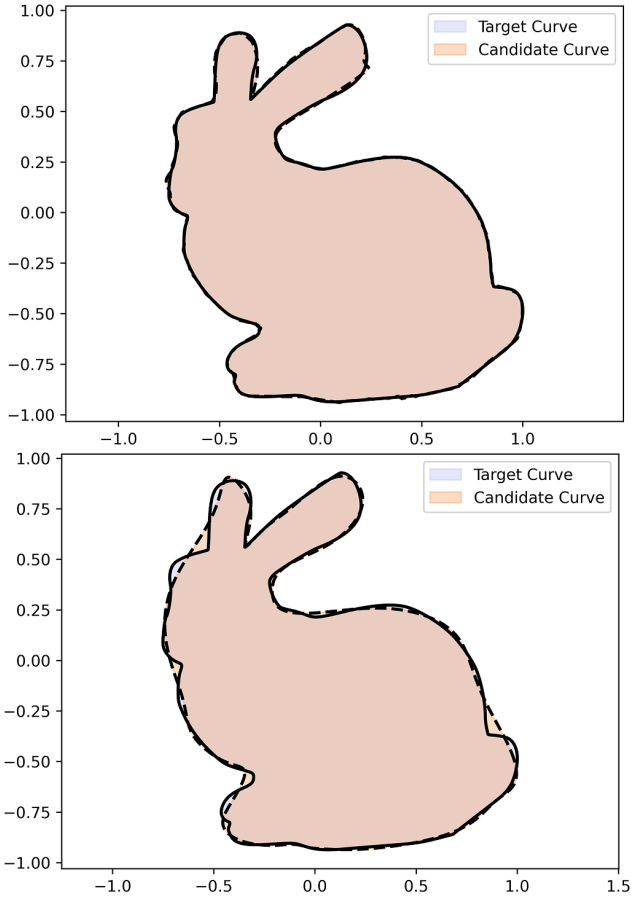

Figure 7. a) Top figure shows NIGnet fit to the stanford bunny b) Bottom figure shows Neural ODEs fit to the stanford bunny. Both networks used the same number of parameters 2800.

Shape	NIGnet	Neural ODE
circle	102.5	40.1s
square	99.2s	57.5s
airfoil	100.3s	130.2s
heart	100.9s	52.3s
stanford bunny	105.5s	88.1s
hand	108.3s	125.0s
puzzle piece	105.6s	99.4s
airplane	105.1s	127.2s

Table 2. A comparison of NIGnets and Neural ODEs based on training time.

5.3. Inference Time

Inference time is the time taken to transform an input to the shape represented by our geometry representation method. Since the inference time depends on the discretization resolution (a finer discretization requires more compute

Num Points on Shape	NIGnet	Neural ODE
10	0.00085s	0.01552s
100	0.00142s	0.02059s
1000	0.00336s	0.03674s
10000	0.01705s	0.13668s
100000	0.11853s	0.78579s
1000000	1.13743s	10.23218s

Table 3. A comparison of NIGnets and Neural ODEs based on inference time averaged over 100 runs.

and time) we compare how the inference time compares and scales for both NIGnets and Neural ODEs. This comparison is shown in Table 3.

These results were obtained by first fitting the same sized (2800 parameter) NIGnet and Neural ODE models to the stanford bunny as the target shape. Then the model was queried 100 times on different discretization resolutions and the total time reported is the average time over these 100 runs. We observed empirically that at 100 runs the variance of the average time was low enough to provide a reasonable estimate.

We observe that both NIGnets and Neural ODEs scale almost linearly with the discretization resolution though the variation for NIGnets is more regular. We also observe that NIGnets consistently offer a 10x speed boost compared to Neural ODEs making them superior for optimization tasks.

An important consideration for this comparison for Neural ODEs is the integration scheme used. We use an adaptive 5^{th} order Runge-Kutta method. A different integration scheme would lead to a different set of step sizes being used and consequently different inference times and also different final representations. Since NIGnets are an explicit representation defined analytically, they do not suffer this issue.

5.4. Ability to work with different Geometric Loss functions

The quality of a shape representation scheme can also be judged based on its ability to work with different geometric loss functions. We study the robustness of NIGnets and Neural ODEs by analysing their ability to work with different geometric losses. In particular we use the geometric loss functions from geosimilarity [1]. If the geometry representation captures the shape features we say that it works and show that with a ✓and an inability to drive down the loss is shown with a ✗. Results of our experiments are shown in Table 4.

We observe that NIGnets work with 6 out of 9 geometric loss functions while Neural ODEs only work with 3 out of 9 available loss functions. NIGnets are therefore a more robust representation scheme w.r.t. the metric used for measuring shape similarity/dissimilarity.

Geometric Loss Function	NIGnet	Neural ODE
MSELoss	✓	✗
ChamferLoss	✓	✓
USDFLoss	✓	✓
MAELoss	✓	✗
MaxAbsErrorLoss	✗	✗
MaxSquaredErrorLoss	✗	✗
SmoothHausdorffLoss	✓	✓
SmoothMaxSquaredErrorLoss	✓	✗
HausdorffLoss	✗	✗

Table 4. A comparison of NIGnets and Neural ODEs based on their ability to work with different loss functions.

6. Conclusion and Future Work

The following are lucrative research directions:

- Invertible Residual Networks: Using iResNets [3] to perform the discrete deformation version of Neural ODEs. This will allow consistent compute times in training and also speed up inference times.
- Extend the comparison to 3D. To compare how the curse of dimensionality affects NIGnets and NeuralODEs. In particular, theoretical predictions for NIGnets predict that compute would grow by a factor of 9/4 as all the 2×2 matrices will now become 3×3. Since matrix multiplication is $O(N^2)$ we expect a factor of increase of $(3/2)^2$. For Neural ODEs the factor of increase is (3/2) as the only thing changing is the dimension of the data tensors from $(N, 2)$ to $(N, 3)$. Therefore, Neural ODEs should be able to close the gap in terms of compute times when moving to 3D.
- Compare performance of NIGnets and Neural ODEs on real world shape optimization tasks such as aerodynamic shape optimization to compare their performance in terms of their ability to morph shapes quickly and stably while maintaining the guarantee on non-self-intersection.

7. Acknowledgements

We thank Patrick Kidger for pointing out the crucial detail that any NeuralODE would describe a non-self-intersecting geometry. We also thank Ricky T. Chen for his torchdiffeq `https://github.com/rtqichen/torchdiffeq` library that allowed a seamless execution from idea to code. We also thank Prof. David Duvenaud for pointing us to the Pointflow paper [16].

References

[1] A. Aalok. Geosimilarity, 2025.

[2] A. Aalok. Nignets, 2025.

[3] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *International conference on machine learning*, pages 573–582. PMLR, 2019.

[4] C. Chen, J. Wei, T. Chen, C. Zhang, X. Yang, S. Zhang, B. Yang, C.-S. Foo, G. Lin, Q. Huang, et al. Cadcrafter: Generating computer-aided design models from unconstrained images. *arXiv preprint arXiv:2504.04753*, 2025.

[5] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[6] R. T. Q. Chen. torchdiffeq, 2018.

[7] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.

[8] M. Guo, B. Wang, K. He, and W. Matusik. Tetsphere splatting: Representing high-quality geometry with lagrangian volumetric meshes. *arXiv preprint arXiv:2405.20283*, 2024.

[9] C. Igel. Smooth min-max monotonic networks. *arXiv preprint arXiv:2306.01147*, 2023.

[10] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[11] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[12] 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*, pages 652–660, 2017.

[13] J. Sill. Monotonic networks. *Advances in neural information processing systems*, 10, 1997.

[14] 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*, pages 1912–1920, 2015.

[15] Y.-P. Xiao, Y.-K. Lai, F.-L. Zhang, C. Li, and L. Gao. A survey on deep geometry learning: From a representation perspective. *Computational Visual Media*, 6(2):113–133, 2020.

[16] G. Yang, X. Huang, Z. Hao, M.-Y. Liu, S. Belongie, and B. Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019.