Rig Inversion by Training a Differentiable Rig Function

Mathieu Marquis Bolduc
mmarquisbolduc@ea.com
SEED Electronic Arts
Austin, Texas, United States of America

Hau Nghiep Phan
hphan@ea.com
SEED Electronic Arts
Montréal, Québec, Canada

ABSTRACT

Rig inversion is the problem of creating a method that can find the rig parameter vector that best approximates a given input mesh. In this paper we propose to solve this problem by first obtaining a differentiable rig function by training a multi layer perceptron to approximate the rig function. This differentiable rig function can then be used to train a deep learning model of rig inversion.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Animation**.

KEYWORDS

rig inversion, neural networks, computer animation

ACM Reference Format:
Mathieu Marquis Bolduc and Hau Nghiep Phan. 2022. Rig Inversion by Training a Differentiable Rig Function. In *SIGGRAPH Asia 2022 Technical Communications (SA '22 Technical Communications), December 6–9, 2022, Daegu, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3550340.3564218

1 INTRODUCTION

It is possible to directly obtain 3D and 4D mesh data from various capture systems (e.g.[Fyffe et al. 2015; Ghosh et al. 2011]) as an alternative to traditional motion capture that tracks joints positions. One motivation for such systems is to capture soft body deformations at a high fidelity. It is desirable for this 3D data to be rigged, i.e. to obtain the set of best-corresponding animation parameters (such as animation tool GUI parameters). This allows 3D animation artists to edit the pose using their own tools, as well as to use the captured data in various tools and software built for any particular rig. If the rig logic is a function that outputs a 3D mesh from animation parameters inputs [Hahn et al. 2012], or rig parameters, then obtaining the rig parameters corresponding to a 3D mesh is the problem of inverting that function. It should be noted that the rig logic that we wish to inverse may be neither injective nor surjective; it may be possible for the exact same mesh to be obtained from different sets of rig parameters, and it is also extremely unlikely for the rig function to be able to perfectly output any given mesh that can be captured. The consequence is that in practice the rig function should not be considered bijective, i.e. it cannot be perfectly inverted. Thus the problem of rig inversion becomes the problem of finding a function whose output are the rig parameters that, as input to the rig logic, minimize the difference between the input mesh and the rigged mesh.

SA '22 Technical Communications, December 6–9, 2022, Daegu, Republic of Korea

ACM ISBN 978-1-4503-9465-9/22/12…$15.00
https://doi.org/10.1145/3550340.3564218

2 PREVIOUS WORK

Rig inversion has been a subject of interest in recent years. Noting that the performance costs of Jacobian-based methods grows to the square of the number of training samples, [Holden et al. 2015] proposed a more memory-efficient method based on training a multi-layer perceptron to minimize the difference in output rig parameters given an input mesh, using data from animators. [Holden et al. 2017] also proposed a method based on Gaussian process regression. [Racković et al. 2021] clusters vertices together with corresponding rig control parameters such that each cluster can be inverted using gaussian processes regression. [Rackovic et al. 2021] improves on linear approximations of the rig by modeling the rig using a second-order polynomial which is then inverted using Levenberg-Marquardt optimization. [Gustafson et al. 2020] analytically create an inference-optimized version of their rig as to speed up jacobian-based methods of rig inversion.

3 PROPOSED METHOD

In this work we assume that the rig logic $L(r)$ (Eq. 1), which returns the mesh x from rig parameters values r, is complex and non-linear. This is the case for the facial animation rigs we use, and thus the problem of rig inversion cannot be solved by linear methods. We also assume rigs that have too many parameters to be solved in a practical manner by Jacobian-based methods. We note that our rigs cannot be completely clustered as proposed in [Racković et al. 2021] because they don't respect that work's clustering hypothesis. When we attempted to invert our rigs with meshes from capture sessions using previously published methods such as [Holden et al. 2015], we were faced with a number of difficulties, which we will now detail.

$$L(r) = x \tag{1}$$

$$L^{-1}(L(r)) = r \tag{2}$$

$$L(\hat{L}^{-1}(x)) \approx x \tag{3}$$

3.1 Issues in inverting the rig function

3.1.1 Non-Surjectivity. The limited data from rigged animations does not always generalize well to new poses that are not observed during training. Even with a comprehensive training set, because our rig functions are not surjective, there are captured meshes that do not exist in the space of rigged meshed, and thus cannot be represented in a training dataset built from rigged animations.

As discussed in section 3, in these cases where an exact solution L^{-1} does not exist (Eq. 2), we wish to instead find an approximate inverse rig function $\hat{L}^{-1}(x)$ that will provide the rig parameter vector r that best approximates the input mesh x when applied to the rig logic L (Eq. 3)

3.1.2 Non-Injectivity. Our rig functions are not injective, i.e. there may exist a set of different input rig parameters that will output the same mesh. This means that training a deep learning model to invert such rigs using a regression loss on the rig parameters is noisy, as the training dataset will contain different ground truths for identical or very similar inputs. In practice this means gradients from that loss will make the model learn to output an average of such ground truths, with no guarantees that these averages will be good solutions.

3.1.3 Distribution of rig parameters. In practice, not every value in the rig parameter vector r has the same importance. Some rig parameters have a more pronounced effect on the resulting mesh than others. While it is desirable to learn all of them, some can tolerate a higher error rate when we wish to minimize the perceptual difference between a mesh x and $L(\hat{L}^{-1}(x))$. However, the importance of each parameter is difficult to weigh and prone to human bias.

3.2 Obtaining a differentiable rig function

Our proposed method is based on the idea that all three of these difficulties can be solved by the same change to the training procedure. We propose to train the deep learning model inverting the rig function by replacing the loss on the output rig parameter vector r by a loss on the mesh $L(\hat{L}^{-1}(x))$ resulting from the rig parameter vector L^{-1} when evaluated by the rig function L. This can be made practical by obtaining a differentiable approximation $\hat{L}_d$ of the rig function L. This approximation can then be used after the inverted rig model during training in an Encoder-Decoder fashion (Fig. 2) to obtain an output mesh. A loss can then be applied on this resulting mesh and back propagated to the inverted rig model.

The rig function is most often treated as a black box [Holden et al. 2017; Seol and Lewis 2014] that can be evaluated but not analytically manipulated. It may also be that the rig function L cannot be derived for the entire input domain $\bar{r}$. We can however train a fully differentiable approximation $\hat{L}_d$ of the rig function. We propose to train a decoder-shaped MLP using a dataset of randomly selected rig parameter vectors that have been evaluated by the rig logic in the animation software (Fig. 1). Unlike its inverse L^{-1}, the rig logic L is assumed to be a true function, and thus training a deep learning model to approximate it is likely to pose less difficulties than approximating its inverse. Recent work provides more details on how to best train an approximation of the rig function [Song et al. 2020].

3.3 Training the inverse rig model

Having a differentiable approximation of the rig, we can train the rig inversion model using only mesh data with no corresponding rig parameter vector ground truth (Fig. 2), which solves the three issues we presented at the beginning of this section. To obtain a good generalization, this training data should be randomly augmented (Section 3.1.3) to provide noisy but recognizable meshes in the largest possible variety of plausible poses. Regarding the last issue we presented (Section 3.1.3), we assume that mesh topologies will be oversampled in regard to the number of vertices in the most perceptually critical areas. The fortunate consequence of this hypothesis is that errors on the most critical rig parameters will result in higher losses on the mesh.

There are a few additional properties to consider. As [Holden et al. 2015] observes, it is desirable to obtain rig parameters that the artists would find sensible. As such, sparse rig parameter are a desirable property as artists will want to minimize the number of parameters to work with. Using Relu activations in the rig inverse model is a good way to encourage sparsity [Goodfellow et al. 2016]. Finally, it is also desirable to artists for subsequent frames in an animation to have temporarily coherent rig parameters, something that is not guaranteed if the rig function is not bijective. To this end, we may need to enforce a local Lipschitz continuity in the inverse rig function network. Noise augmentation is an effective way of doing so [Usama and Chang 2018].

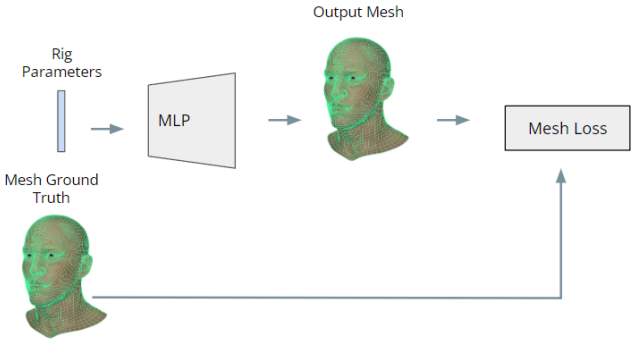

Figure 1: Learning an approximation of the rig function.

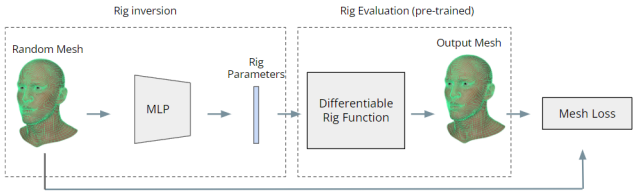

Figure 2: Training the rig inversion model using a differentiable rig approximation.

4 EXPERIMENTAL RESULTS

All of the quantitative experimental results we present are from a facial animation rig used by a content creation team at EA. The rig in question has 137 input parameters and outputs 628 weights, which are linearly composed to create a 8447 vertices mesh in a matrix multiplication operation. Additional, qualitative tests using animations instead of capture data confirmed the proposed method is applicable to other kinds of animation rigs as well (Fig. 4).

4.1 Differentiable rig function

From the experimental rig we generated a set of 100000 random rig parameter values. As stated in Section 3, sparsity is desirable, and so we select both the number of non-zero values as well as their respective values following a uniform distribution. Those random rig parameter activations are run in Maya™'s python scripting system in order to obtain the corresponding weights and thus the mesh output. 10% of this dataset is reserved as a test set and the remaining 90% as a training set. Previous work such as [Holden et al. 2017] have stated that the size of such a training dataset needs to grow in an impossibly large fashion. This is true only if all inputs and outputs are dependent on every other one. In practice this is rarely the case and not the case with the rig used in these experiments, and as such it is possible for the dataset to have a size that is practical. As the dataset is randomly generated in a uniform distribution, no validation set is used during training.

The architecture used to model $\hat{L}_d$ is a MLP with two hidden layers of 1024 parameters which outputs 628 factors. Hidden layers have leaky ReLu activations [Nair and Hinton 2010] while the output has sigmoid activations to constrain the output between [0,1] which is the weights range. We use no normalization as it severely degrades our experimental results. The model is trained by an Adam optimizer with a learning rate starting at 1e-4 and being halved every time the training loss reaches a plateau for 20 epochs, until the learning rate is smaller than 1e-6. Blendshape factors are multiplied by the matrix, and the loss used is a MSE loss on the vertex positions of the resulting mesh. All results are consistent across different random weight initialisations.

As the rig we used for experiments is uncharacteristically not a black box, we also compare results with a differentiable implementation of that rig function in Pytorch™.

Table 1: Rig logic approximation results presenting vertex error in mm.

Model	Mean Vertex Error	Maximum Vertex Error
Programmatic	**0.22**	**1.4**
MLP Model	0.78	5.9

Table 1 compares the test result for the trained and programmatic rig logic approximations. The learned approximation is fairly precise with less than a millimeter error on average. The non-zero error in the programmatic method is explained by slight differences in curve evaluation between the Pytorch™ and Maya™ implementations.

4.2 Inverse rig approximation results

Both differentiable rig functions (trained and programmatic) are used to train an inverse rig function approximation model. In both cases the architecture used is an encoder-shaped MLP with 4 hidden layers of [2048, 1024, 512, 256] parameters with leaky ReLu activations, while the output has tanh activations corresponding to the domain of the rig parameters. It is critical to avoid having the inverse rig model feeding out-of-manifold values to the rig model approximation as this will lead to unpredictable behavior. As with the rig function model we use no normalization layers.

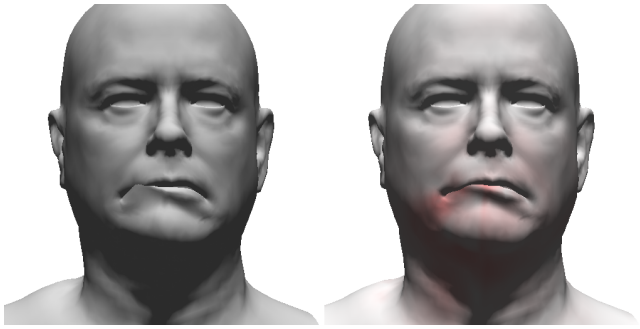

Figure 3: Left: Randomly combining blendshapes with no regard for rig logic produces a recognizable but noisy mesh, similar to imperfect meshes coming from the capture process or content-generation models. Right: Using a mesh loss trains the inverse rig logic to find the rig parameters that best approximate the input mesh when applied to the rig logic, which doubles as a denoising process.

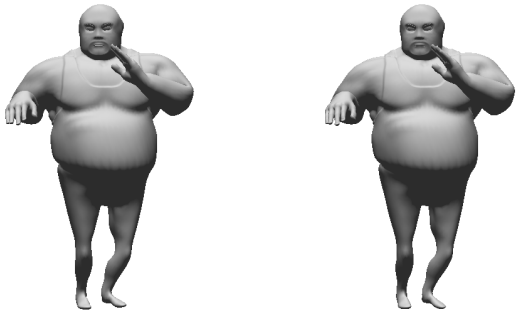

Figure 4: Qualitative result on a full body, non-blendshape rig. The difference between the original animation (left) and the mesh animated with recovered rig parameters (right) is almost imperceptible.

This model is trained with meshes generated by randomly combining blendshapes with no respect to the rig logic - Since the rig logic is highly non-surjective even in regard to the more limited space of factors, it is highly unlikely for the rig to be able to exactly represent a random combination of blendshapes (Fig. 3). Combining all blendshapes usually results in non-plausible meshes, and so both the number of blendshapes used in each training sample as well as their respective value follows a uniform distribution. Each epoch comprises 10000 iterations using mini batches of 64 samples. The model is trained with the same optimizer and schedule as the rig function approximation. All results are consistent across different random weight initialisations.

Table 2 compares the result of training the rig inverse approximation using either differentiable rig approximation method and testing the resulting model on a test set of 10000 poses randomly generated using the same method as the training set, as well as a set of captured data. To obtain unbiased test results, we do not use the rig function approximations $\hat{L}_d$ for testing purposes. Instead, we obtain the rig parameter vector r for each test sample from

Table 2: Rig inversion results presenting vertex error in mm.

	Mean Vertex Error	Max Vertex Error
Random Poses		
Programmatic $\hat{L}_d$	**2.5**	**15.0**
Trained $\hat{L}_d$	3.0	17.2
[Holden et al. 2015]	9.2	100.0
Captured Data		
Programmatic $\hat{L}_d$	**4.8**	**49.0**
Trained $\hat{L}_d$	4.9	49.1
[Holden et al. 2015]	10.0	99.0

the inverse rig approximation model and use them as input to the original rig logic L using the Maya(™) animation software.

Training with either programmatic or trained rig approximations yield a significant improvement over the previous deep learning method of Holden [Holden et al. 2015] which is trained using rig parameters ground truth. Captured data is highly unlikely to be perfectly expressible by the rig, and so achieving a zero error on it is not possible. The capture data is also noisy, which explains the large maximum error for all methods. This is favorable as it allows inverse rig techniques to filter noise in captured data.

4.3 Parameter Smoothness Results

Finally we evaluate the usefulness of the model in giving out smooth rig parameters for a temporarily coherent input sequence. We use the squared second-order differences as a measure of roughness, and we compare the result with directly optimizing the rig parameter vector without training an inverse rig model. The results are presented in Table 3. We see that using an encoder-shaped inverse rig model outputs rig parameters that are an order or magnitude smoother, without need for noise augmentation. This is despite each rig parameter vector being initialized with the previous frame value prior to being optimized in order to help with temporal coherency. Qualitatively, the result from directly optimizing the rig parameter vector is too unstable for practical use.

Table 3: Temporal coherency results presenting rig parameters roughness

	Roughness
Proposed method with Programmatic $\hat{L}_d$	3.8e-3
Proposed method with trained $\hat{L}_d$	**3.4e-3**
Directly optimizing with Programmatic $\hat{L}_d$	2.3e-2
Directly optimizing with trained $\hat{L}_d$	2.7e-2

5 CONCLUSION AND FUTURE WORK

We demonstrated the effectiveness of using a differentiable rig function approximation to train a rig function approximation. Our proposed method presents several improvements over the prior art. First and foremost, the proposed method fully addresses the non-bijective nature of complex rig functions while also generalizing well to plausible meshes that cannot be exactly produced by the rig function. The proposed method also requires no dataset of artist-produced rigged poses or animations. As future work we hope to improve accuracy and reduce the number of training epochs by using architectures that are tailored for mesh data instead of MLPs.

ACKNOWLEDGMENTS

We want to acknowledge Mattias Teye from SEED Electronic Arts for his valuable contribution to this project. We also want to thank Adobe™ for the luchador character.

REFERENCES

Graham Fyffe, Andrew Jones, Oleg Alexander, Ryosuke Ichikari, and Paul Debevec. 2015. Driving High-Resolution Facial Scans with Video Performance Capture. *ACM Trans. Graph.* 34, 1, Article 8 (dec 2015), 14 pages. https://doi.org/10.1145/2638549

Abhijeet Ghosh, Graham Fyffe, Borom Tunwattanapong, Jay Busch, Xueming Yu, and Paul Debevec. 2011. Multiview Face Capture Using Polarized Spherical Gradient Illumination. *ACM Trans. Graph.* 30, 6 (dec 2011), 1–10. https://doi.org/10.1145/2070781.2024163

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* MIT Press, Cambridge, MA, USA, 507. http://www.deeplearningbook.org.

Stephen Gustafson, Aaron Lo, and Paul Kanyuk. 2020. Analytically Learning an Inverse Rig Mapping. In *ACM SIGGRAPH 2020 Talks* (Virtual Event, USA) *(SIGGRAPH '20)*. Association for Computing Machinery, New York, NY, USA, Article 27, 2 pages. https://doi.org/10.1145/3388767.3407316

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-Space Physics. *ACM Transactions on Graphics - TOG* 31 (07 2012). https://doi.org/10.1145/2185520.2185568

Daniel Holden, Jun Saito, and Taku Komura. 2015. Learning an Inverse Rig Mapping for Character Animation. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '15)*. Association for Computing Machinery, New York, NY, USA, 165–173. https://doi.org/10.1145/2786784.2786788

Daniel Holden, Jun Saito, and Taku Komura. 2017. Learning Inverse Rig Mappings by Nonlinear Regression. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (1 March 2017), 1167 – 1178. https://doi.org/10.1109/TVCG.2016.2628036

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Haifa, Israel) *(ICML'10)*. Omnipress, Madison, WI, USA, 807–814.

Stevo Rackovic, Claudia Soares, Dusan Jakovetic, and Zoranka Desnica. 2021. Accurate, Interpretable, and Fast Animation: An Iterative, Sparse, and Nonconvex Approach. https://doi.org/10.48550/ARXIV.2109.08356

Stevo Racković, Cláudia Soares, Dušan Jakovetić, Zoranka Desnica, and Relja Ljubobratović. 2021. Clustering of the Blendshape Facial Model. https://doi.org/10.48550/ARXIV.2110.15313

Yeongho Seol and J. P. Lewis. 2014. Tuning Facial Animation in a Mocap Pipeline. In *ACM SIGGRAPH 2014 Talks* (Vancouver, Canada) *(SIGGRAPH '14)*. Association for Computing Machinery, New York, NY, USA, Article 13, 1 pages. https://doi.org/10.1145/2614106.2614108

Steven L. Song, Weiqi Shi, and Michael Reed. 2020. Accurate Face Rig Approximation with Deep Differential Subspace Reconstruction. *ACM Trans. Graph.* 39, 4, Article 34 (jul 2020), 12 pages. https://doi.org/10.1145/3386569.3392491

Muhammad Usama and Dong Eui Chang. 2018. Towards Robust Neural Networks with Lipschitz Continuity. https://doi.org/10.48550/ARXIV.1811.09008