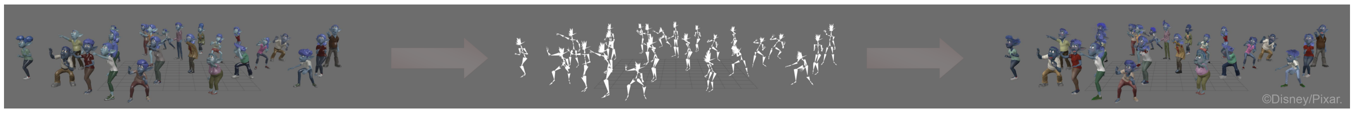
Analytically Learning an Inverse Rig Mapping

Stephen Gustafson
Pixar Animation Studios
sgustafson@pixar.com

Aaron Lo
Pixar Animation Studios
alo@pixar.com

Paul Kanyuk
Pixar Animation Studios
pkanyuk@pixar.com



ABSTRACT

One of the main obstacles to applying the latest advances in motion synthesis to feature film character animation is that these methods operate directly on skeletons instead of high-level rig parameters. Loosely known as the "rig inversion problem", this hurdle has prevented the crowd department at Pixar from procedurally modifying character skeletons close to camera, knowing that these procedural edits would not be reliably transferred to the equivalent motion on the full character for polish.

Prior attempts at solving this problem have tended to involve hard-coded heuristics, which are cumbersome for production to debug and maintain. To alleviate this overhead, we have adopted an approach of solving the inversion problem with an iterative least-squares algorithm. However, although there are numerous existing methods for solving this problem, the computation of the rig Jacobian is a frequent requirement, which in practice is prohibitively expensive. To accelerate this process, we propose a method wherein an approximation of the rig is derived analytically, through an offline learning process. Using this approximation, we invert full character rigs at real-time rates.

CCS CONCEPTS

• **Computing methodologies → Procedural animation**; **Machine learning approaches**; • **Theory of computation → Numeric approximation algorithms**.

KEYWORDS

rig inversion, machine learning

ACM Reference Format:
Stephen Gustafson, Aaron Lo, and Paul Kanyuk. 2020. Analytically Learning an Inverse Rig Mapping. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '20 Talks), August 17, 2020.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3388767.3407316

1 INTRODUCTION

Production rigs are arbitrarily complex, and have no well-defined mapping for computing rig parameters from joint transformations. Past attempts at solving the inversion problem have involved hard-coded heuristics, both in the form of rig-specific code as well rigged networks that describe the mapping, requiring significant time to setup and maintain. This has previously caused film productions to forego motion synthesis techniques that operation in terms of joint transforms, since animation polish requires inversion to parameters.

To automate this process, we developed a work-flow in which the only input required from the user is the list of parameters to use for the inversion process. Using those parameters, the inversion is solved using the well-known Gauss-Newton (GN) iterative least squares algorithm. Using GN, we are able to invert from joint transforms to rig parameters within an acceptable threshold in most cases. In cases where GN fails to converge, the Levenberg-Marquardt (LM) [Levenberg 1944] algorithm is used instead.

Between GN and LM, we are able to solve all of our inversion problems. However, both algorithms require the computation of the Jacobian matrix per iteration. Althought it is possible to compute the Jacobian using analytical methods [Orin and Schrader 1984], the arbitrary complexity of our rigs requires us to to evaluate the rig Jacobian using methods that are applicable to "black box" rigs: Namely, finite-differencing (FD). In practice, computing the Jacobian using FD is prohibitely expensive [1], requiring n^2 rig executions [Hahn et al. 2012]. This problem is compounded by the statefulness of our rigs: Modification of rig parameters is not thread-safe, so it is not possible to parallelize the evaluation of the Jacobian by computing different columns on different threads.

To accelerate the Jacobian, we learn an approximation of a rig through an offline training process. The training procedure exercises rig parameters to obtain a new rig function, which approximates the original. Unlike the original rigs, the approximation function is stateless, allowing the computation of the Jacobian to be performed in parallel. Further, the approximation provides sufficient rig knowledge to allow the problem to be partitioned into sub-problems, enabling additional parallelization.

2 RELATED WORK

seen a variety of solutions. In [Holden et al. 2017], the Jacobian is approximated using Gaussian processes, leading to a real-time inversion procedure. This was taken one step further in [Holden et al. 2015], which performed the complete inversion using neural

[1] Average of 3.5s to compute the full rig Jacobian on a production rig, requiring over minute to iteratively solve just the first frame. Inverting animations may take hours.

networks. However, both methods utilize a pool of pre-generated animations to serve as training data. fIn the early stages of production, our animation inventory is inadequate to feed such training processes. Given uncertainty at the effectiveness of these techniques on purely synthetic data, and concerns over the requisite training time as character rigs undergo frequent changes early in production, we opted against these approaches.

3 RIG APPROXIMATION

We define the rig functions as $F(\beta)$, taking β rig parameters, and producing an array of 4x4 matrix transformations. A classification procedure exercises a source rig over a range of values to decompose $F(\beta)$ into a set of operators $P = \{P_0(\beta_0), P_1(\beta_1), \ldots P_n(\beta_n)\}$, where $P_i : \mathbb{R} \rightarrow \mathbb{M}_4$, and $P_i(0) \coloneqq I$. The result of each P_i contributes to the approximation by multiplying against a subset of the transformations resulting from $F(0)$. To construct an approximation, we must obtain both the set P and its composition order.

Classification begins by analytically determining a suitable function for each P_i such that for each affected output k of $F(\beta)$,

$$\left|\frac{F_k(0)P_i(\alpha) - F_k(\delta_i\alpha)}{\alpha}\right| < \epsilon$$

where $\delta_i\alpha$ denotes the set of rig parameters fixed at 0, with the i'th entry equal to α, and ϵ specifies an error threshold for validation. Each P_i is determined by analytically factoring $F_k(\delta_i\alpha)F_k(0)^{-1}$ into a set of pre-defined operators, and selecting the best fit, if any. The pre-defined set of operators includes translation and scaling along an arbitrary axis, rotation about an arbitrary axis and pivot, and uniform scaling about an arbitrary pivot. Once a potential P_i is selected, the choice is validated against other affected outputs of $F(\beta_i)$. If a suitable P_i cannot be determined, the parameter is discarded and will not be used for inversion.

The set of pre-defined operators do not always accurately model the original function. In production rigs, we encounter some deformations which are *rotation-like*, but not true rotations. A generous value of ϵ allows slightly inaccurate approximations to be adopted.

4 DERIVING COMPOSITION ORDER

Having identified a set of operators, their composition order must be determined. The relative composition order of the i'th and j'th rig parameters can be determined by sampling the rig function at $F(0)$, $F(\delta_i\alpha)$, $F(\delta_j\alpha)$ and $F(\delta_i\alpha + \delta_j\alpha)$. If the effect of parameter i precedes that of j, then we expect that for each affected output k,

$$F_k(\delta_i\alpha)_k F(0)^{-1} = F_k(\delta_i\alpha + \delta_j\alpha)F_k(\delta_j\alpha)^{-1} \tag{1}$$

Conversely, if the effect i applies after j, then we would expect:

$$F_k(\delta_j\alpha)F_k(0)^{-1} = F_k(\delta_i\alpha + \delta_j\alpha)F_k(\delta_i\alpha)^{-1} \tag{2}$$

If both (1) and (2) are true, that implies that the rig parameters are linearly independent and may be sorted relative to each other in any order we choose. If neither equation holds, that suggests that the parameters have a complex inter-dependency that can't be approximated through our pre-defined function set.

The final composition order is derived by sorting the rig parameters, based on (1) and (2). Since some pairs of parameters are unsortable, items are sorted using an insertion sort algorithm, utilizing a modified binary search. If (1) and (2) are both false while comparing parameters, both parameters are discarded.

It is possible that some operator classifications remain invalid following the sorting procedure, such as if the effect of a rig parameter changes when two other parameters are placed in a certain configuration. But sorting parameters have proven sufficient for our production rigs, given the types of parameters they provide.

5 INVERSION

To invert joint transformations, we solve for rig parameters using GN in the normal way, substituting the original rig function with the learned approximation. At each iteration, the Jacobian is still computed with FD. However, since the rig function is stateless, the columns of the Jacobian are able to be computed in parallel. If GN fails to converge after a fixed number of iterations (typically 30), an LM solve is attempted instead.

An additional benefit of the approximation is that, whereas the original rig was arbitrarily complex, the approximation uses a fixed set of operators with a trivially introspectable structure. By analyzing the structure, we are able to partition the rig into smaller problems, each of which can be solved in parallel. For instance, the set of all joints that make up an arm must be solved together, but the left arm can be solved independent from the right arm.

6 RESULTS AND APPLICATIONS

By simplifying the rig function and parallelizing the Jacobian, we are able to reduce the average time taken to compute the full rig Jacobian at each iteration from 3.5 sec. to $6e^{-4}$ sec. – a reduction by more than 5000x. By combining this with additional parallelism enabled by rig partitioning, we have been able to invert joint transforms to rig parameters with target convergance thresholds at a rate of 2ms/frame, placing the inversion procedure firmly in the real-time realm.

Although our approach required a training procedure, a relative small number of evaluations have been required in practice. Common Common production rigs tend to $\sim 3k$ rig evaluations during training, a process taking around 2 minutes.

In application, our rig inversion implementation allowed the crowds department of the Pixar film Onward, to use procedural lookats and virtual production techniques much closer to camera than previously possible. Rig inversion is now a standard tool in Pixar's crowd pipeline and is facilitating further use of motion synthesis and modification for feature film animation.

REFERENCES

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-Space Physics. *ACM Trans. Graph.* 31, 4, Article 72 (July 2012), 8 pages. 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 (March 2017), 1167–1178. https://doi.org/10.1109/TVCG.2016.2628036

Kenneth Levenberg. 1944. A method for the solution of certain non-linear problems in least squares. *Quart. Appl. Math.* 2, 2 (July 1944), 164–168. https://doi.org/10.1090/qam/10666

David E. Orin and William W. Schrader. 1984. Efficient Computation of the Jacobian for Robot Manipulators. *The International Journal of Robotics Research* 3, 4 (1984), 66–75. https://doi.org/10.1177/027836498400300404 arXiv:https://doi.org/10.1177/027836498400300404