Building a Scalable UGC Auto-Rigging Pipeline with OpenUSD

Wayne Wu
Genies Inc.
Los Angeles, California, USA

Davide Munisso
Genies Inc.
Los Angeles, California, USA

Figure 1: Examples of auto-rigged UGC avatars (left) and UGC wearables (right) that can be equipped to the avatars, all composed and rendered with OpenUSD.

ACM Reference Format:
Wayne Wu and Davide Munisso. 2025. Building a Scalable UGC Auto-Rigging Pipeline with OpenUSD. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '25 Talks), August 09-13, 2021.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3450623.3464663

1 INTRODUCTION

User-generated content (UGC), including content produced by artificial intelligence (AIGC), plays a pivotal role in the 3D avatar ecosystem, facilitating limitless customization possibilities. The system developed by Genies concentrates primarily on humanoid avatars and their associated customization assets, such as wearables, skins, and emotes. These UGC avatar assets form the foundation of our real-time social and gaming experiences. A scalable and efficient pipeline is essential to process and conform uploaded assets into an interoperable 3D representation. A critical step within this pipeline is auto-rigging, which must possess the robustness to manage assets generated by potentially millions of users. Our strategy for scalability is grounded in OpenUSD, aimed at optimizing computational performance, reducing processing costs, and maintaining the flexibility necessary for the continuous refinement of our system.

SIGGRAPH '25 Talks, August 10-14, 2025, Vancouver, BC

ACM ISBN 978-1-4503-8373-8/21/08.
https://doi.org/10.1145/3450623.3464663

2 EFFICIENT AND LOSSLESS PROCEDURAL COMPUTATION

The integration of UGC assets can be enabled with auto-rigging [Shi et al. 2024]. With the combination of geometric computation, procedural rigging, and machine learning, we achieve efficient and high-fidelity processing of these assets. While traditional 3D software tools such as Maya and Houdini offer the necessary functionality, they come with significant licensing costs and software complexity. Alternatively, OpenUSD and NumPy present a lightweight yet robust solution. The *usd-core* Python package is compact and straightforward to install. Its seamless data interchange with NumPy enables vectorized computations that can be further accelerated with GPU-based packages such as PyTorch and Warp [Macklin 2024]. We employ a lossless representation of 3D data utilizing OpenUSD, and develop a thin wrapper to convert USD primitives, such as Mesh and Skeleton, into NumPy-based objects for subsequent processing. This significantly reduces the overhead introduced by data conversion and maintains data homogeneity throughout the pipeline.

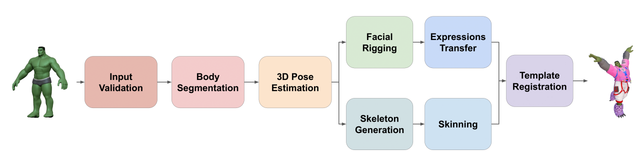

Figure 2: A high-level overview of our auto-rigging pipeline for custom avatars. Wearables follow a similar but more simplified pipeline.

3 CUSTOM SCHEMAS FOR AVATAR ASSETS

To support diverse UGC assets, a standardized representation within OpenUSD is crucial. We introduce custom schemas, such as *UsdAvatar* and *UsdWearable*, to define the scene composition, attributes, and behavior of avatar assets. Each asset type is constructed with a collection of well-structured USD primitives. Additional attributes are added to define asset-specific properties and customization parameters. Additionally, we implement behaviors that enable asset configuration. For example, *UsdAvatar* assets can equip wearables with automatic skeleton retargeting; *UsdWearable* assets can be refitted according to the avatar's shape using radial basis function interpolation [de Boer et al. 2007]. By conforming UGC assets to predefined templates that adhere to our schema, they enable interoperability within our ecosystem. This structured representation streamlines the ingestion process, maintains asset uniformity, and ultimately enables avatar customization using OpenUSD.

4 LAYER-DRIVEN TASK GRAPH

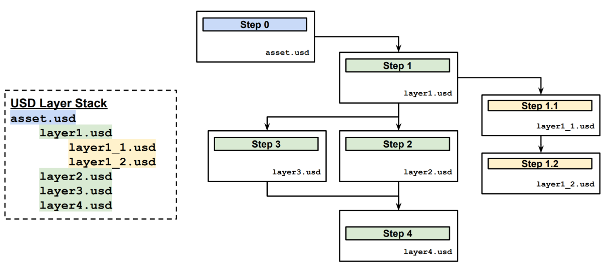

Figure 3: A generalized example of a task dependency graph that maps to the USD layer stack of an asset.

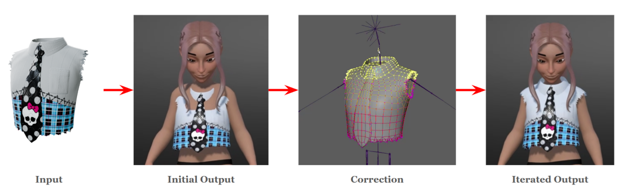

Figure 4: Any layer can be edited manually and re-ingested back into the pipeline for downstream processes.

A scalable UGC pipeline necessitates an iterative improvement framework, particularly for refining erroneous outputs. We structure individual auto-rigging steps as discrete USD layers, ensuring the preservation of intermediate outputs within the final asset representation. This layered methodology offers two principal benefits: (1) Enhanced debugging capabilities—developers can examine the final USD file, typically packaged in the USDZ format, to assess intermediate data at each processing step. (2) Corrective flexibility—each intermediate layer can be manually adjusted to rectify inaccuracies, thereby facilitating iterative refinement. The revised data is reintegrated into the pipeline and leveraged to improve machine learning model performance. Task orchestration is enabled through a custom task dependency graph, inspired by Apache Airflow, that is tailored to align with the USD layer stack of an asset. The dependency graph optimizes for parallel processing and caching while preserving the structural integrity of the USD file hierarchy.

5 RESULTS AND DEPLOYMENT

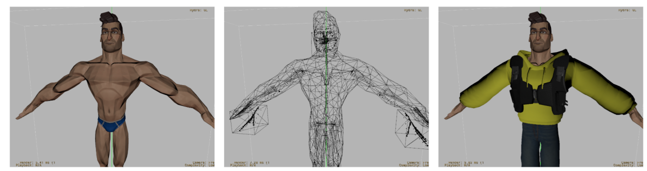

Figure 5: The final processed avatar (left) with all its intermediate data for debugging (middle), and with wearables refitted directly inside our custom UsdViewer (right).

By establishing a structured OpenUSD representation through schema-based definitions, we enable interactive avatar visualization and composition with UGC assets without game engine dependencies. Avatars can be configured programmatically or using our internal UsdView-based asset configurator. Our workflow seamlessly integrates with OpenUSD's ecosystem, such as Hydra-based rendering. The adoption of OpenUSD as the foundation of our procedural system has drastically reduced our end-to-end processing time compared to our previous pipeline that relied on multiple 3D software packages. The lightweight Python-based environment ensures seamless and cost-effective deployment on cloud infrastructure, providing a scalable solution. Our UGC pipeline has been systematically tested on more than 5,000 assets, with about 15% undergoing correction and re-ingestion through our iterative editing framework, further enhancing data quality and pipeline robustness.

6 CHALLENGES AND FUTURE WORK

While OpenUSD provides a scalable foundation for our UGC pipeline, its limited visualization tools and inconsistent level of DCC adoption necessitate additional tooling for improved usability. This aligns with our long-term objective of building a comprehensive UGC avatar configuration system natively in OpenUSD. We also aim to experiment with OpenExec and Hydra Scene Delegate for evaluating schema-defined avatar behaviors. Finally, optimizing data and behavior redundancy between the UGC pipeline and our runtime avatar system could unify the pipeline, enhancing OpenUSD's applicability to real-time experiences.

REFERENCES

A. de Boer, M.S. van der Schoot, and H. Bijl. 2007. Mesh deformation based on radial basis function interpolation. *Computers Structures* 85, 11 (2007), 784–795. https://doi.org/10.1016/j.compstruc.2007.01.013 Fourth MIT Conference on Computational Fluid and Solid Mechanics.

Miles Macklin. 2024. Warp: Differentiable Spatial Computing for Python. In *ACM SIGGRAPH 2024 Courses* (Denver, CO, USA) *(SIGGRAPH Courses '24)*. Association for Computing Machinery, New York, NY, USA, Article 24, 147 pages. https://doi.org/10.1145/3664475.3664543

Weiqi Shi, Maurice Chu, Dario Kneubuhler, Ervin Teng, Adam Burr, Tallon Hodge, Ian Sachs, and Andrew Kunz. 2024. End-to-end Automatic Body and Face Setup for Generative or User Created 3D Avatar. In *ACM SIGGRAPH 2024 Talks* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 13, 2 pages. https://doi.org/10.1145/3641233.3664345