ANIMA - AI-Driven Natural Interpolation for Mesh ARAP-ing

Jeng Wen Joshua Lean
National Tsing Hua University
Taiwan

Aurick Daniel Franciskus Setiadi
National Tsing Hua University
Taiwan

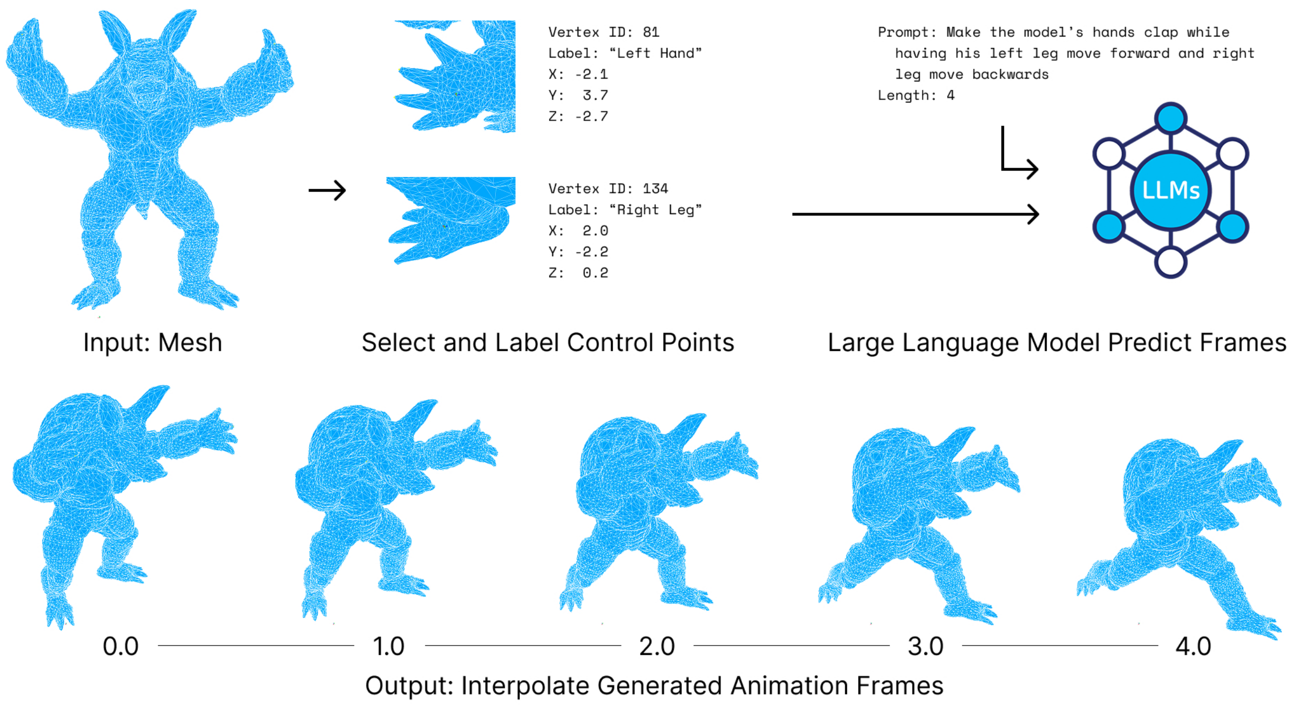

Vertex ID: 81
Label: "Left Hand"
X: -2.1
Y: 3.7
Z: -2.7
Vertex ID: 134
Label: "Right Leg"
X: 2.0
Y: -2.2
Z: 0.2
Prompt: Make the model's hands clap while having his left leg move forward and right leg move backwards
Length: 4
LLMs
Input: Mesh
Select and Label Control Points
Large Language Model Predict Frames
0.0
1.0
2.0
3.0
4.0
Output: Interpolate Generated Animation Frames

Figure 1: ANIMA in Action: Transforming a 3D Mesh into a Dynamic Clapping Animation Using AI-Driven Deformation

ABSTRACT

This paper introduces a 3D modeling system designed for interactive mesh manipulation and AI-driven animation generation. The system allows users to manipulate 3D meshes through intuitive vertex selection and As-Rigid-As-Possible deformation algorithms. A novel pipeline extracts selected vertices as control points with semantic roles and processes them with a text prompt to generate animation frames using a large language model. The resulting vertex deformations are applied to the mesh for real-time visualization. By integrating efficient local mesh processing with AI-powered deformation generation, this system demonstrates a scalable approach to enhancing 3D content creation workflows in computer graphics, blending traditional geometric modeling with generative AI techniques.

1 INTRODUCTION

Creating and animating 3D models traditionally demands expertise in complex interfaces, manual rigging, and labor-intensive keyframe animation, posing significant barriers for novices and inefficiencies for experienced users. Conventional tools often require meticulous manual adjustments, limiting the speed and accessibility of 3D content creation. This paper presents an innovative 3D modeling system that streamlines mesh manipulation and animation through a seamless integration of interactive editing and AI-driven deformation generation. The proposed pipeline enables users to intuitively select and modify mesh vertices using As-Rigid-As-Possible (ARAP) [1] [8] deformation algorithms for precise surface editing. Concurrently, it leverages a large language model (LLM) to interpret user-defined control points [6] with semantic roles (e.g., "left arm", "head") [3] and text prompts, generating animation frames with precise vertex positions. This approach minimizes manual effort while producing realistic poses and animations. Designed for accessibility and efficiency, the system holds promise for applications in animation, game design, and rapid prototyping, offering a scalable framework that blends traditional geometric modeling with generative AI techniques.

2 METHODOLOGY

This section outlines the methodology for developing and implementing the proposed 3D modeling system, which integrates interactive mesh manipulation with AI-driven animation generation.

The system is designed to streamline 3D content creation through a pipeline that combines user-driven geometric editing with automated, AI-powered deformation generation. The methodology is divided into three key components: mesh manipulation, AI-driven animation generation, and real-time visualization.

2.1 Mesh Manipulation Framework

The system employs a C++-based application for interactive 3D mesh editing. Users load triangular mesh data (e.g., in PLY format) and interact with the mesh through a trackball camera interface, enabling rotation and zoom. Vertex selection is facilitated via mouse-based controls, allowing users to pick specific vertices or triangles in 3D space. Selected vertices serve as control points, which can be manually adjusted to deform the mesh using As-Rigid-As-Possible (ARAP) algorithms. ARAP ensures that deformations preserve local rigidity, maintaining the mesh's structural integrity while enabling flexible shape modifications. The system supports storing multiple mesh states to facilitate animation frame creation.

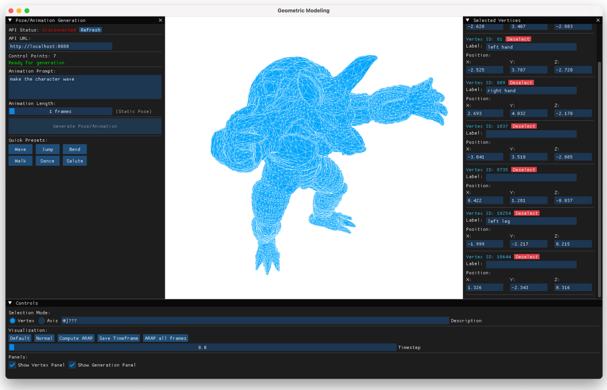
2.2 AI-Driven Animation Generation

To automate animation, the system extracts user-selected control points, each annotated with semantic roles (e.g., "left arm," "head"), and pairs them with a user-provided text prompt describing the desired animation. These inputs are processed by a large language model (LLM), which generates a sequence of animation frames as absolute 3D vertex positions. The LLM is modified to interpret geometric and semantic data, ensuring that the generated deformations align with the user's intent. The resulting vertex positions are converted into relative deformation deltas to maintain compatibility with the original mesh, enabling smooth integration into the animation pipeline. The system renders the manipulated and animated mesh in real time, providing immediate feedback to users. Visualization modes allow users to toggle between rendering styles, such as surface normals, vertex weights, or wireframes, to inspect the mesh's properties. Manipulation gizmos are overlaid to assist with precise vertex adjustments. The rendering loop updates continuously to reflect changes from manual edits or AI-generated deformations, ensuring a responsive and interactive experience. The user interface includes panels for vertex editing and animation control, designed to be intuitive for both novice and experienced users.

2.3 Implementation Detail

2.3.1 Frameworks. We develop the application using OpenGL version 3.3, leveraging its programmable pipeline capabilities for rendering custom visuals. GLFW is utilized as the windowing and context creation library due to its lightweight API and strong support for handling input devices such as keyboard and mouse. Together, OpenGL and GLFW provide a robust foundation for rendering real-time 3D graphics and user interaction in an efficient and cross-platform manner [2, 7].

2.3.2 Mesh Loader. For mesh import functionality, we employ the Assimp (Open Asset Import Library), which offers support for numerous 3D model formats including PLY, OBJ, and STL. Assimp abstracts away the file parsing complexities and provides a consistent API to access vertex positions, normals, and faces, thus



API Status: Disconnected Refresh
API URL: http://localhost:8080
Control Points: 7
Ready for generation
Animation Prompt: make the character wave
Animation Length: 1 frames (Static Pose)
Generate Pose/Animation
Quick Presets: Wave Jump Bend Walk Dance Salute

-2.628	3.407	-2.883

Vertex ID: 81 Deselect
Label: left hand
Position: X: -2.525 Y: 3.787 Z: -2.720

Vertex ID: 889 Deselect
Label: right hand
Position: X: 2.693 Y: 4.032 Z: -2.170

Vertex ID: 1037 Deselect
Label:
Position: X: -3.041 Y: 3.510 Z: -2.085

Vertex ID: 8735 Deselect
Label:
Position: X: 0.422 Y: 1.281 Z: -0.037

Vertex ID: 10254 Deselect
Label: left leg
Position: X: -1.999 Y: -2.217 Z: 0.215

Vertex ID: 10644 Deselect
Label:
Position: X: 1.326 Y: -2.343 Z: 0.316

Selection Mode: [✓ Vertex] Axis @]??? Description
Visualization: Default Normal Compute ARAP Save Timeframe ARAP all frames
0.0 Timestep
Panels: [✓ Show Vertex Panel] [✓ Show Generation Panel]

Figure 2: A peek at the interactive application

simplifying mesh initialization [9]. Upon loading, vertex data is converted into our custom data structure for efficient processing and rendering.

2.3.3 Mesh Data Structure. We implement the Half-Edge data structure to represent mesh topology. This structure is well-suited for traversal and manipulation operations commonly required in mesh editing tasks. Each face (triangle), edge, and vertex maintains a pointer to one of its half-edges, and each half-edge stores references to its origin vertex, adjacent face, edge, and its opposite half-edge. This setup allows constant-time access to neighboring vertices and faces, which is essential for algorithms like ARAP that rely heavily on local mesh connectivity [5]. Additional properties such as vertex positions, edge weights, and triangle areas are stored as attributes to support geometry processing.

2.3.4 ARAP Implementation. We integrate the As-Rigid-As-Possible (ARAP) surface deformation algorithm using the libigl geometry processing library. libigl provides a high-level interface for computing the ARAP energy and solving the associated optimization problem efficiently using pre-factorized sparse linear solvers [4].

The ARAP method minimizes a deformation energy that measures how closely the local deformations preserve rigidity. Specifically, the energy is defined as:

$$E(\mathcal{S}') = \sum_{i=1}^{n} w_i \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}'_i - \mathbf{p}'_j) - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j) \right\|^2 \tag{1}$$

where $\mathbf{p}_i, \mathbf{p}_j$ are the original vertex positions, $\mathbf{p}'_i, \mathbf{p}'_j$ are the deformed positions, $\mathbf{R}_i$ is the optimal local rotation at vertex i, and w_{ij} are cotangent weights. The energy term encourages local neighborhoods to undergo near-rigid transformations.

The minimization is performed in an iterative two-step process:

2.3.5 Rotation Estimation: At each iteration, the optimal rotation matrix $\mathbf{R}_i$ for each vertex is computed using Singular Value Decomposition (SVD) on the covariance matrix:

$$\mathbf{S}_i = \sum_{j \in \mathcal{N}(i)} w_{ij} (\mathbf{p}_i - \mathbf{p}_j)(\mathbf{p}'_i - \mathbf{p}'_j)^\top \tag{2}$$

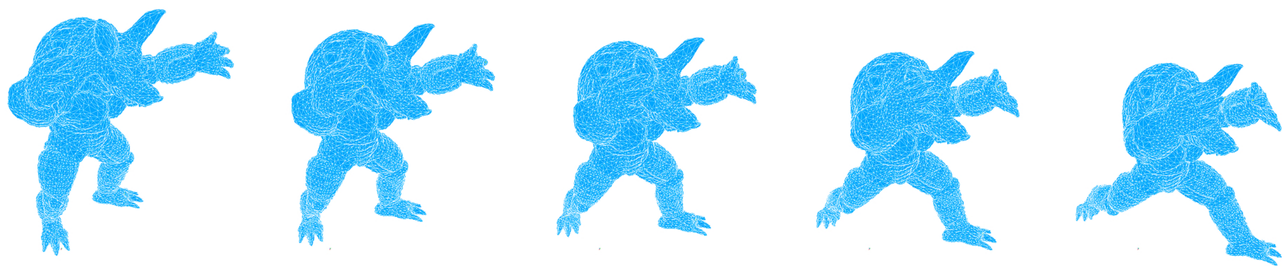


Figure 3: Four animation frames generated from the prompt: "make the model's hands clap while having his left leg move forward and right leg move backwards." Both legs, hands, and torso were selected as control points. The animation was synthesized using OpenAI's GPT-4.1 model.

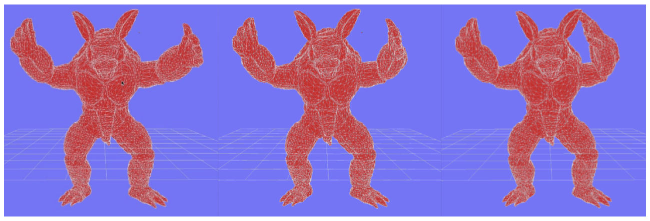


Figure 4: An example of interpolated mesh given by 2 KeyFrames

$$\mathbf{R}_i = \arg\min_{\mathbf{R}\in SO(3)} \|\mathbf{S}_i - \mathbf{R}\|_F \tag{3}$$

This gives the best-fit rotation $\mathbf{R}_i \in SO(3)$ that aligns the original edge vectors to the deformed ones in a least-squares sense.

2.3.6 Position Update: Once the rotations are estimated, the new vertex positions $\mathbf{p}'_i$ are computed by solving a sparse linear system derived from the minimization of the energy:

$$\sum_{j\in\mathcal{N}(i)} w_{ij}(\mathbf{p}'_i - \mathbf{p}'_j) = \sum_{j\in\mathcal{N}(i)} w_{ij} \cdot \frac{1}{2}(\mathbf{R}_i + \mathbf{R}_j)(\mathbf{p}_i - \mathbf{p}_j) \tag{4}$$

This step solves for all $\mathbf{p}'_i$ simultaneously, typically using a pre-factorized sparse linear solver.

In our system, selected vertices act as constraints or anchors, and the surrounding mesh deforms accordingly while attempting to preserve local rigidity. This results in smooth and visually plausible deformations, even under significant user-defined transformations. Users may switch between linear (single-pass) and iterative (rotation-updating) ARAP modes to trade off between computational speed and deformation accuracy.

2.3.7 Editor Tools. To facilitate intuitive mesh editing and animation, the system provides a suite of interactive editor tools. Users can select vertices directly on the mesh using mouse-based picking, which highlights selected control points for further manipulation. A 3D transformation gizmo is overlaid on the selected vertices, allowing users to translate them within the scene using axis-constrained movement.

2.3.8 Animation Tools. For temporal animation control, a timeline slider is provided. This enables users to specify keyframes by assigning specific deformed vertex positions at selected time intervals. Between keyframes, the system employs linear interpolation (LERP) to generate smooth transitions of vertex positions, producing continuous animation sequences. This keyframe-based workflow allows users to construct animations incrementally while maintaining precise control over individual vertex trajectories and overall deformation behavior.

2.3.9 Prompt Processing Pipeline. Upon receiving user input, the system constructs a structured prompt that includes:

- Control point labels and 3D coordinates
- User prompt (e.g., "make the character raise its left hand")

This information is tokenized and formatted into a JSON-like structure fed into the LLM. The model then infers plausible motion trajectories per control point, returning frame-wise absolute 3D positions. These outputs are post-processed into local deltas to ensure smooth transitions and avoid discontinuities.

3 RESULTS AND EVALUATION

The developed 3D modeling system was evaluated through practical testing on a variety of triangular mesh inputs, including the Armadillo model. The system successfully enabled users to select vertices, apply ARAP deformations, and generate animations using text prompts. The ARAP algorithm preserved local rigidity, ensuring smooth and realistic deformations. In terms of performance, the ARAP computation—when executed without parallelization—required approximately one second for mid-sized meshes (Armadillo). Meanwhile, the large language model (LLM), used as the reasoning component for interpreting prompts and generating animation frames, required approximately one minute to process each prompt and return vertex positions. These results demonstrate that while ARAP provides efficient deformation updates, the overall pipeline is bottlenecked by LLM inference time. Nonetheless, the integration of intuitive manual editing with AI-driven animation confirms the system's viability for prototyping and creative tasks in animation workflows.

4 CONCLUSIONS

This work demonstrates a novel 3D modeling system that blends interactive mesh manipulation with AI-powered animation generation. By combining ARAP deformation with LLM-driven frame generation, the system simplifies complex animation workflows, making them accessible and efficient. Future work could focus on automation of vertex selection and enhancing AI prompt expressiveness for broader applications in computer graphics.

5 CONTRIBUTION

The system was developed collaboratively from the ground up. Aurick was primarily responsible for implementing the geometric framework, including mesh data structures and deformation algorithms. Joshua focused on integrating the large language model (LLM) into the system, enabling prompt-based animation generation and seamless communication between the AI component and the mesh manipulation framework.

REFERENCES

[1] Marco Colaianni, Javier Ovide, and Jürgen Schulze. 2016. Heterogeneous Deformable Modeling of Surfaces with ARAP. Computer-Aided Design 80 (2016), 1–10. https://doi.org/10.1016/j.cad.2016.07.005

[2] GLFW Developers. 2024. GLFW: An OpenGL library. https://www.glfw.org/.

[3] Jianan Guo, Hao Wu, Yixin Liu, and Hao Zhang. 2024. Move as You Say, Interact as You Can: Language-Guided Human Motion Generation with Scene Affordance. arXiv preprint arXiv:2407.01426 (2024). https://doi.org/10.48550/arXiv.2407.01426

[4] Alec Jacobson, Daniele Panozzo, et al. 2024. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

[5] Lutz Kettner. 1999. Using generic programming for designing a data structure for polyhedral surfaces. Computational Geometry 13, 1 (1999), 65–90.

[6] Zeyu Liu, Jikai Wu, Xin Wang, and Chi Zhang. 2022. MotionScript: Natural Language Descriptions for Expressive 3D Human Motions. arXiv preprint arXiv:2212.01568 (2022). https://doi.org/10.48550/arXiv.2212.01568

[7] Dave Shreiner, Graham Sellers, John Kessenich, and Barthold Licea-Kane. 2013. OpenGL Programming Guide: The Official Guide to Learning OpenGL. Addison-Wesley.

[8] Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. Computer Graphics Forum 26, 3 (2007), 417–426. https://doi.org/10.1111/j.1467-8659.2007.01062.x

[9] Assimp Development Team. 2024. Open Asset Import Library (Assimp). https://www.assimp.org/.