Speech Driven to 3D Facial Animation in Maya

Yessica Daniela Hernandez
MSc Computer Animation and Visual Effects
Bournemouth University
2025

Abstract

Retargeting is a highly relevant topic in the field of computer graphics currently, as it enables the transfer of animation data across different models, significantly accelerating the animation process and providing a reusable boilerplate for animation production. This project presents the implementation of a pipeline that connects the output of a machine learning model developed by PixelMux, which generates speech-driven videos from a portrait and an audio file, with Autodesk Maya. This is achieved through the development of a retargeting pipeline that integrates the generated data into Maya's environment via a custom plugin.

Contents

Figures

Code listings

Chapter 1 Introduction

Facial motion retargeting is the process of transferring motion from a source to a target face mesh. The source motion can originate from an already animated 3D face, a marker-based or markerless system, or from speech-driven models and other generative sources. On the target side, facial models can range from hyper-realistic humans to highly stylized fantasy characters.

One of the main reasons motion capture (MOCAP) techniques remain widely used in retargeting pipelines is their high accuracy in capturing facial motion. However, there is a growing trend toward the use of machine learning-based systems, which have been steadily improving in accuracy thanks to recent advancements in Artificial Intelligence (AI) and more robust landmarks system detection.

1.1 Problem Statement

Facial animation in 3D environments is typically achieved through two primary methods: keyframe animation and performance-driven animation [1]. In traditional keyframe-based workflows, animators manually craft and refine facial poses frame by frame. This process is highly time-consuming, often requiring between 12 to 16 working days for a single animator to produce just 60 seconds of animation [2].

In contrast, performance-driven animation captures facial expressions and body movements using camera-based systems, which are then mapped onto digital characters. A prominent example of this approach is motion capture (MOCAP) technology. Historically, high-quality MOCAP systems were prohibitively expensive and required extensive calibration and specialized technical expertise. However, recent developments have introduced mobile-based MO-

CAP solutions, which offer a more accessible and cost-effective alternative —albeit with certain limitations in precision and flexibility.

Implementing performance-driven animation typically requires a retargeting pipeline, in which captured motion data is adapted to fit the digital character's rig. In both academic and industry contexts, these pipelines predominantly rely on motion capture (MOCAP) as the primary data source. However, this research introduces an alternative approach: a speech-driven retargeting pipeline. This decision is not a reflection of limitations in MOCAP technology, which remains highly effective; rather, it is a strategic extension of a personal project called PixelMux, a system designed to animate AI-generated videos using speech input. The objective is to establish a direct connection between the PixelMux machine learning model and the animation generation process, enabling a novel and scalable method for producing facial animation from audio.

1.2 Objectives and Contributions

This thesis focuses on enhancing the integration of the PixelMux machine learning model into professional 3D animation workflows. The key contributions are:

- **Maya Plugin Development:** A custom plugin was created to import and manipulate PixelMux-generated facial animation directly within Autodesk Maya.
- **Improved Facial Landmark Detection:** PixelMux initially relied on a sparse set of 68 facial landmarks for video generation. This work introduces a denser detection system comprising 478 landmarks, enabling more detailed and expressive facial mapping. For implementation, a manually selected subset of 51 landmarks was adopted to ensure compatibility with the retargeting pipeline developed in this project.
- **Speech-Driven Retargeting Pipeline:** A new pipeline was developed to transfer facial motion from PixelMux to 3D characters. By tracking the movement of 51 facial landmarks over time, the system activates facial action units, which drive muscle-based deformations and enable animation directly from speech input. The pipeline design is inspired by the paper **A Facial Motion Retargeting Pipeline for Appearance-Agnostic 3D Characters**.

Chapter 2
Related Work

Zhu and Joslin (2024) propose two approaches to formulate the 3D facial motion retargeting problem: cross-mapping, when the source and target models differ anatomically, and parallel mapping, when the data is generated using an identical reference model. Additionally, they present a comprehensive taxonomy that divides the facial retargeting pipeline into two fundamental components: Facial Motion Interpretation and 3D Face Parameterization. The following chapter will examine in detail the various techniques used within these two components, as well as provide a deeper exploration of the concepts of cross-mapping and parallel mapping.

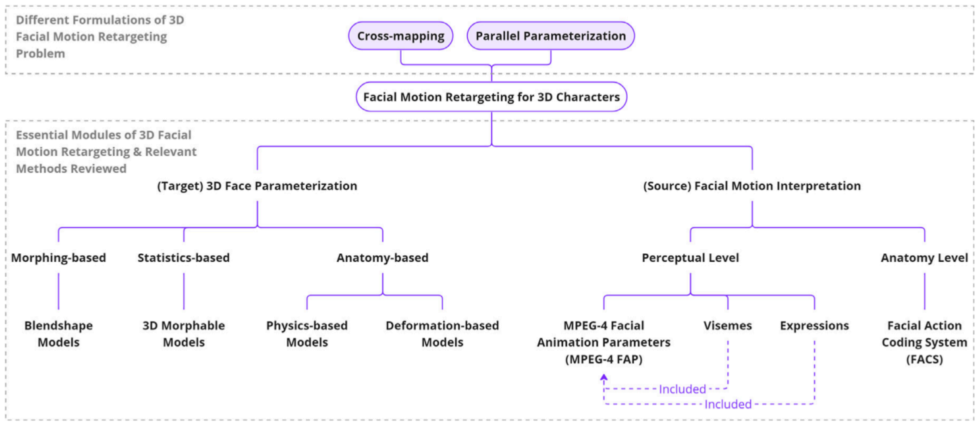

Different Formulations of 3D Facial Motion Retargeting Problem
Cross-mapping
Parallel Parameterization
Facial Motion Retargeting for 3D Characters
Essential Modules of 3D Facial Motion Retargeting & Relevant Methods Reviewed
(Target) 3D Face Parameterization
(Source) Facial Motion Interpretation
Morphing-based
Statistics-based
Anatomy-based
Perceptual Level
Anatomy Level
Blendshape Models
3D Morphable Models
Physics-based Models
Deformation-based Models
MPEG-4 Facial Animation Parameters (MPEG-4 FAP)
Visemes
Expressions
Facial Action Coding System (FACS)
Included
Included

Figure 2.1: Core Modules of Facial Retargeting Frameworks According to Zhu & Joslin.

2.1 3D Face Parametrization

During data transfer, it is essential to explicitly define both; the source and target models; to ensure accurate mapping between the source space and the 3D model space. This parametrization not only clarifies how and where the data should be aligned; but also guarantees that the process remains reproducible and transferable across different models.

The parametrization of a 3D mesh typically involves specifying the number of vertices, defining the mesh topology, and identifying key feature points or regions, such as areas corresponding to facial muscle geometry, which are essential for accurate deformation and animation.

2.1.1 Morphing-based Parametrization (Blendshape models)

The Blendshape model is one of the most intuitive 3D face parametrizations and has been widely used in industrial practices [2]. They are built by morphing a neutral 3D face into multiple predefined expressions or poses, all of which share the same mesh topology. Each blendshape acts as a basis vector representing a specific facial action, and the final expression is typically obtained through linear interpolation.

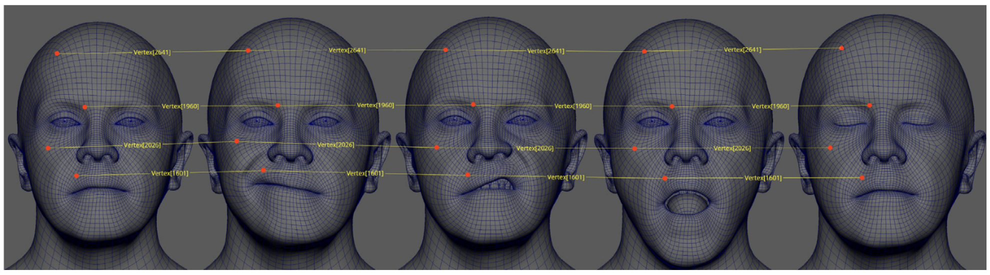


Figure 2.2: Vertex-to-Vertex Correspondence between the Original Face shape and A Blendshape.

Creating blendshapes typically involves extensive manual labour and must be customised for each individual 3D model. This makes the process difficult to replicate using conventional methods. However, several techniques have been proposed to automate and optimise blendshape generation. For instance, Zhang et al. [3] explored the use of video sequences to generate editable and controllable face models for 3D facial animation. In 2013, Cao et al. [4] employed an RGB-D camera to capture facial expressions from 150 subjects, creating a comprehensive database from which subject-specific blendshapes were generated by morphing a generic face model to match individual facial geometry and expression depth. More recently, in 2020, Li et al. [5] introduced a self-supervised model capable of predicting personalised blendshapes from a single neutral face scan, significantly reducing the need for manual intervention.

2.1.2 Statistic-based parametrization - 3D Morphable Models

3D Morphable Models (3DMM) capture the statistical variation of human facial geometry using existing 3D face datasets. Their versatility extends to improving face recognition accuracy, reenacting facial expressions in images and videos, and enabling realistic motion retargeting between 3D faces. Unlike blendshape models—whose blend units often exhibit statistical dependencies —the basis vectors in a 3DMM are orthogonal, ensuring that generated face shapes and expressions remain within anatomically plausible ranges [1].

Despite their strengths, 3DMM have notable limitations. The extracted variations are often not perceptually intuitive, and the resulting parameterized faces can lack expressiveness. Additionally, 3DMMs rely heavily on well-constrained and regularised training datasets, which are limited in number and typically contain data only from human 3D face models, restricting their generalisability to other character types or stylized designs.

2.1.3 Anatomy-Based Models

Anatomy-based face parameterization replicates some or all of the anatomical components of the human face, such as bones, muscles, and soft tissue. These components are then driven by physics simulations or geometric deformations, allowing facial expressions to be generated automatically through realistic biomechanical behavior. [1]

2.1.3.1 Physically-Simulated Face Anatomy

This approach replicate the actual muscle system of the human face, which demands significant computational power and can be challenging to control. In 2016, Cong et al.[6] introduced a system that generates muscle-specific blendshapes, which drive the deformation of a tetrahedral flesh volume and, subsequently, the face mesh. That same year, Ichim et al. [7] proposed a volumetric blendshape model that nonlinearly interpolates deformation gradients across tetrahedralized face volumes. This model allows direct control over mesh deformation—similar to traditional blendshape systems—while also incorporating physics-based features such as collisions and inertia.

More recently, in 2022, researchers at Weta Digital [8] proposed a technique using a nonlinear jaw rig and a set of muscle abstractions, including muscle fiber curves, to drive facial animation. This method was employed in the production of *Avatar 2: The Way of Water*, demonstrating that anatomy-based face parameterization can deliver high-fidelity speech animation with impressive realism.

2.1.3.2 Deformation Inspired By Face Anatomy

The approach involves replicating anatomical components of the face and directly defining their influence on the mesh. In this method, the vertices of the face mesh deform according to the movement of these anatomical replicas. Most 3D software supports joint-based deformation, where joints exert weighted influence over mesh vertices, allowing artists to manipulate these joints to achieve expressive and controllable mesh deformation.

2.2 Facial Motion Interpretation

Facial motion interpretation translates raw facial motion data into parameters that represent its semantic meaning and intensity, enabling the transfer of expressive information across different systems [9]. This process is valuable for both encoding real human facial movements and interpreting animations generated by 3D face models. In the following section, the researcher review various techniques used to interpret facial motion.

2.2.1 The Facial Action Coding System (FACS)

The Facial Action Coding System (FACS) [10], proposed by Ekman and Friesen in 1978, is a comprehensive framework grounded in facial anatomy. It encodes observable movements of individual or groups of facial muscles into discrete units known as Action Units (AUs). When combined, these AUs represent a wide range of facial expressions. FACS is widely used for interpreting facial motion data, and many face rigging workflows adopt it as a reference for generating blendshapes [1].

While FACS classifies facial motion at the anatomical level, other techniques —such as expressions and visemes—interpret motion at the perceptual level, focusing on emotion-related or speech-related cues. However, none of these systems explicitly define how to quantify the intensity of facial motion, which remains a challenge in both analysis and animation.

2.2.2 Perceptual Level Interpretation

2.2.2.1 Expressions & Visemes

Facial expressions provide a perceptual-level description of facial motion, typically involving the coordinated movement of multiple facial regions. Recent research has leveraged expression recognition to enhance the expressiveness of facial motion retargeting results [24]. However, because expressions represent the face as a whole, they lack the granularity to describe how specific facial regions deform during a motion sequence.

Similar to expressions, visemes [11] describe perceptual-level movements, but are specific to the mouth. The term viseme—short for visual phoneme—refers to the observable visual patterns associated with one or more phonemes [12]. While phonemes are the smallest units of sound in a language [13], a single viseme can correspond to multiple phonemes, making it a useful abstraction for speech-driven facial animation.

2.2.2.2 MPEG-4 Facial Animation Parameters (FAP)

The MPEG-4 standard defines 68 Facial Animation Parameters (FAPs), organised into 10 groups. The first group includes two perceptual-level descriptors: visemes and expressions. The remaining groups specify movements of individual facial features, such as raising the left eyebrow or stretching the left lip corner [14].

A key distinction between MPEG-4 FAPs and FACS lies in their foundational principles. FACS defines Action Units (AUs) based on muscle actions, whereas MPEG-4 defines Facial Animation Parameter Units (FAPUs) using fractional distances between key facial landmarks [15]. This structural difference influences how each system interprets and drives facial animation.

Chapter 3
Design and Implementation

In their published paper, Zhu and Joslin propose a 3D facial motion retargeting pipeline designed at the anatomical level to ensure cross-character compatibility. The pipeline consists of two main modules:

- The source motion interpretation module processes marker-based MOCAP data, converting it into FACS Action Unit (AU) labels and measuring their intensities by analysing marker geometries on a per-frame basis.
- The target face parameterization module takes a 3D face mesh and automatically generates a rig composed of simple polygon meshes that represent facial muscle abstractions. Each muscle abstraction has its own local deformation space, capturing both active contractions and passive movements, and is directly driven by the interpreted source motion.

In this thesis, the researcher reuse their target face parameterization module, but instead of using marker-based MOCAP, the researcher adapt the system to work with video-based motion generation.

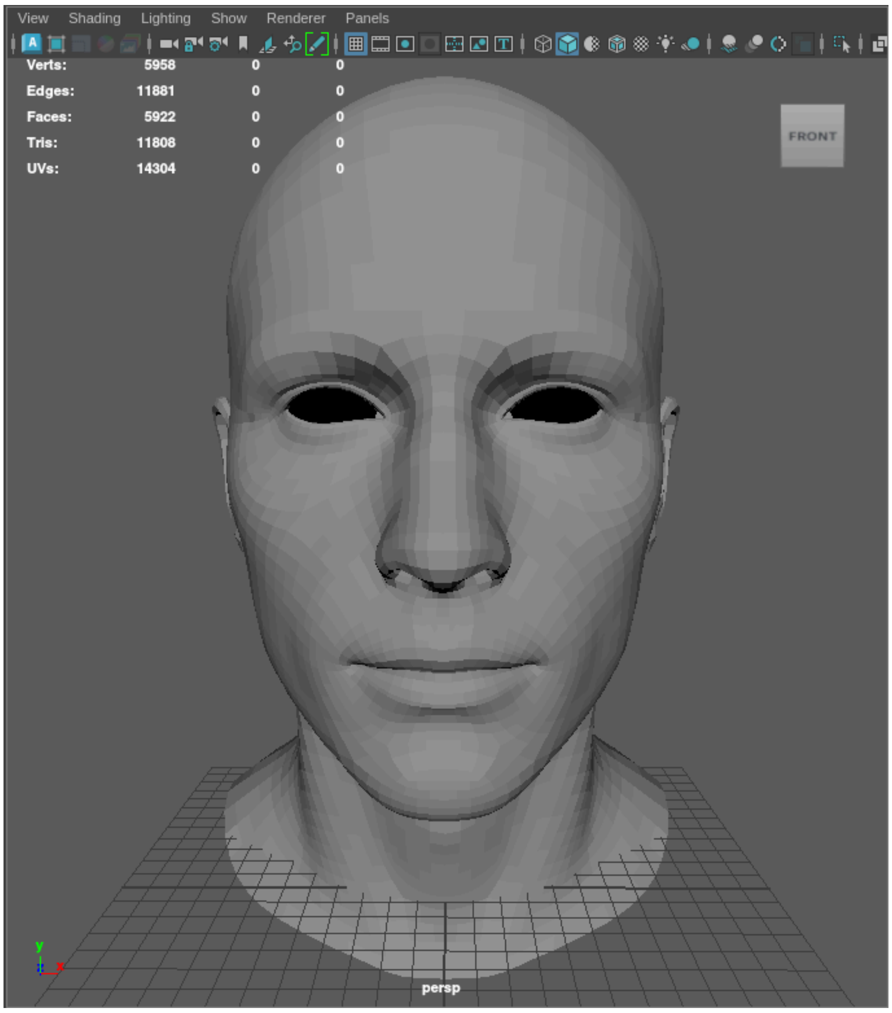
3.1 Facial Parameterization

3.1.1 Base Mesh Template Selection

Selecting an appropriate base mesh is a foundational step in designing a facial retargeting system. Beyond its geometry, the mesh must serve as a universal reference structure, capable of adapting to a wide range of characters while preserving both expressiveness and computational efficiency.

In this project, several 3D facial models were evaluated to identify one that offered a clean, symmetrical, and uniform topology. These characteristics are fundamental not only for controlled mesh deformation but also for enabling automated assignment of muscle regions and precise motion transfer.

The chosen mesh was manually optimised to reduce its vertex count to a level that balances visual fidelity with performance. This simplification did not compromise the mesh's ability to represent complex facial expressions; instead, it made it more suitable for integration into data-driven animation pipelines.



```
Verts:     5958    0    0
Edges:    11881    0    0
Faces:     5922    0    0
Tris:     11808    0    0
UVs:      14304    0    0
```

Figure 3.3: The final base mesh template.

3.1.2 Muscle Abstraction and Semantic Mapping

In the context of the present pipeline, muscles are not represented as physical anatomical structures, but rather as functional regions within a 3D mesh. This abstraction enables the retargeting system to interpret facial movement, from landmarks displacement to action unit to muscles.

To achieve this, 28 muscle regions were manually defined on the base mesh, each associated with a set of vertices that represent its influence over the facial surface. These regions were conceptualized based on anatomical references but adapted to facilitate integration into a deformation system driven by AUs.

Each muscle region was labeled and numbered to maintain consistency in rigging and data mapping. This nomenclature enables the system to activate specific deformations in response to detected movement in facial landmarks.

Beyond its technical function, this muscle segmentation serves as a bridge between the semantics of facial motion and its digital representation, allowing the pipeline to remain compatible with stylized or non-human characters, as long as they adhere to the deformation logic defined by these regions.

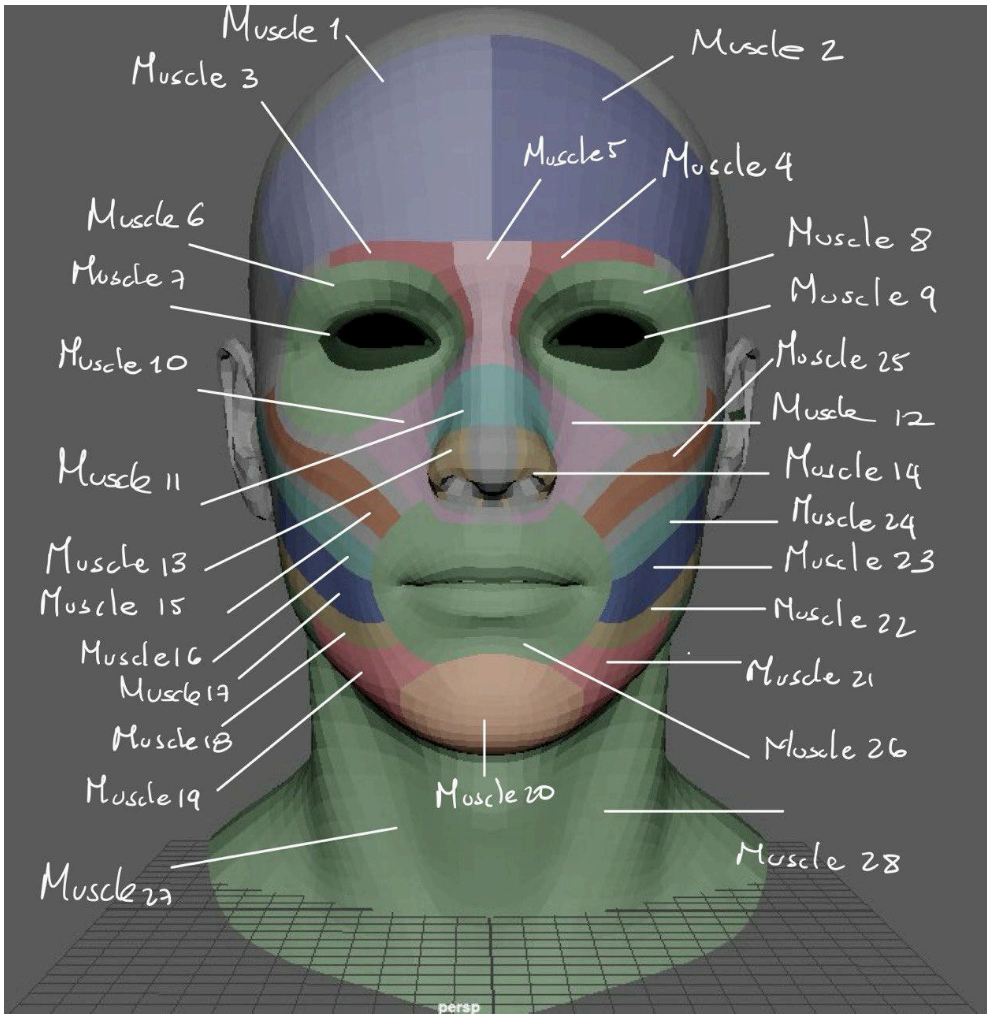
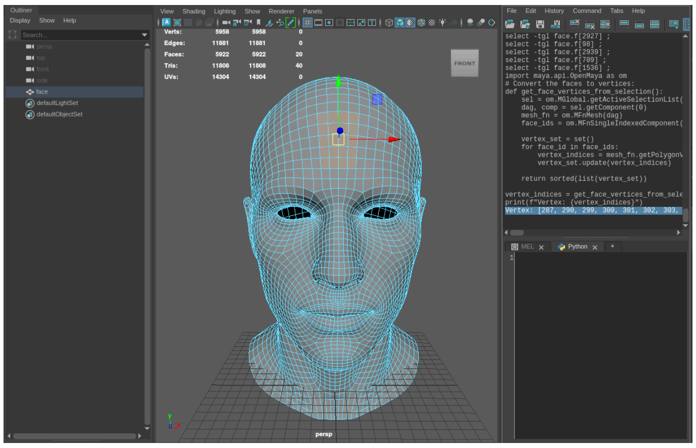


Figure 3.4: Muscles Regions

3.1.2.1 Muscle Identification

Each muscle region was manually selected based on visual references, and its corresponding vertex indices were extracted to define localised deformation zones. To automate this process, a custom Python script was developed using the Maya API. This script converts selected polygonal faces into a sorted list of vertex indices.



```
Verts:     5958    5958     0
Edges:    11881   11881     0
Faces:     5922    5922    20
Tris:     11808   11808    40
UVs:      14304   14304     0
```

```
select -tgl face.f[2927] ;
select -tgl face.f[98] ;
select -tgl face.f[2939] ;
select -tgl face.f[709] ;
select -tgl face.f[1536] ;
import maya.api.OpenMaya as om
# Convert the faces to vertices:
def get_face_vertices_from_selection():
    sel = om.MGlobal.getActiveSelectionList(
    dag, comp = sel.getComponent(0)
    mesh_fn = om.MFnMesh(dag)
    face_ids = om.MFnSingleIndexedComponent(

    vertex_set = set()
    for face_id in face_ids:
        vertex_indices = mesh_fn.getPolygonV
        vertex_set.update(vertex_indices)

    return sorted(list(vertex_set))

vertex_indices = get_face_vertices_from_sele
print(f"Vertex: {vertex_indices}")
Vertex: [287, 290, 299, 300, 301, 302, 303,
```

Figure 3.5: Example of Muscle Selection Regions

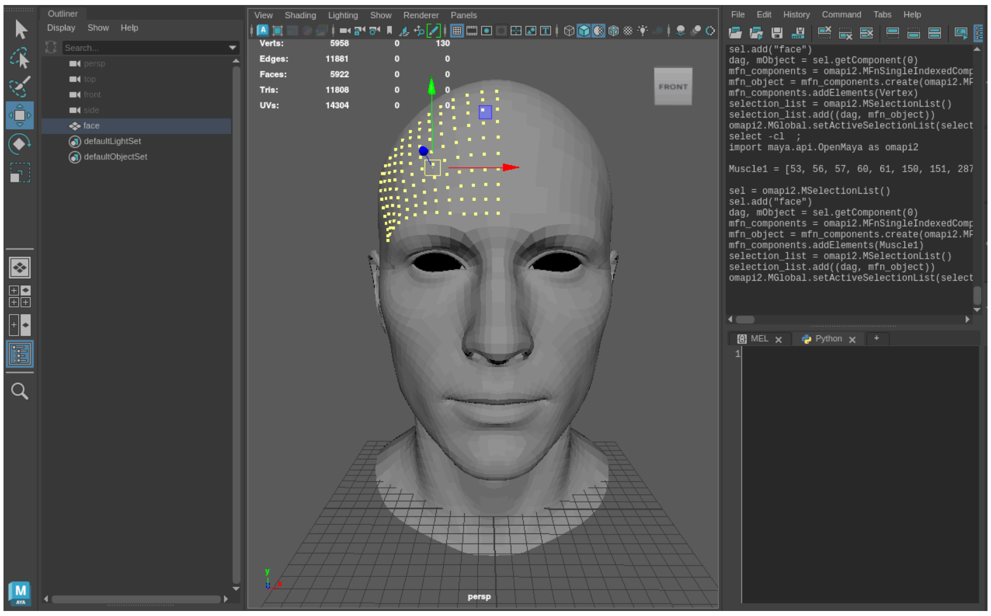
Listing 3.1: "Python script to extract vertex indices from selected faces in Maya"

```python
# To extract the vertices list of the faces selected
import maya.api.OpenMaya as om

# Convert the selected faces to vertex indices
def get_face_vertices_from_selection():
    sel = om.MGlobal.getActiveSelectionList()
    dag, comp = sel.getComponent(0)
    mesh_fn = om.MFnMesh(dag)
    face_ids = om.MFnSingleIndexedComponent(comp).getElements()

    vertex_set = set()
    for face_id in face_ids:
        vertex_indices = mesh_fn.getPolygonVertices(face_id)
        vertex_set.update(vertex_indices)

    return sorted(list(vertex_set))
```

To validate the accuracy of the extracted regions, a second script was used to highlight the vertices of each muscle patch within the Maya viewport. The example below shows the selection of Muscle 1 (Frontalis_R_Ftls).

Listing 3.2: "Python script for selecting vertices of Muscle 1 (Frontalis_R_Ftls) in Maya"

```python
import maya.api.OpenMaya as omapi2

# Taxonomy: Muscle 1 – Frontalis_R_Ftls
Muscle1 = [53, 56, 57, 60, 61, 150, 151, 287, 288, 289, 290, 295,
298, 299, 300, 301, 302, 303, 304, 305, 306, 324, 422, 423, 497,
498, 630, 632, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692,
693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 1048,
1049, 1050, 1051, 1052, 1053, 1056, 1057, 1058, 1059, 1343, 1344,
1345, 1346, 1583, 1660, 1661, 1662, 1691, 1702, 1703, 1704, 1705,
2415, 3026, 3027, 3028, 3089, 3090, 3091, 3095, 3496, 3497, 3498,
3501, 3502, 3504, 3505, 3506, 3507, 3508, 3509, 3523, 3598, 3643,
3976, 3977, 3978, 3979, 3980, 3981, 3982, 3983, 4257, 4442, 4443,
4444, 4445, 4586, 4710, 4749, 4750, 4751, 4752, 4753, 4754, 4755,
4756, 4757, 4936, 4937, 5387, 5456, 5457, 5458, 5459, 5460, 5461,
5462, 5463]

sel = omapi2.MSelectionList()
sel.add("face") # Name of the mesh
dag, mObject = sel.getComponent(0)
mfn_components = omapi2.MFnSingleIndexedComponent(mObject)
mfn_object = mfn_components.create(omapi2.MFn.kMeshVertComponent)
mfn_components.addElements(Muscle1) # Name the of the muscles list
selection_list = omapi2.MSelectionList()
selection_list.add((dag, mfn_object))
omapi2.MGlobal.setActiveSelectionList(selection_list)
```



```
Verts:   5958    0    130
Edges:   11881   0    0
Faces:   5922    0    0
Tris:    11808   0    0
UVs:     14304   0
```

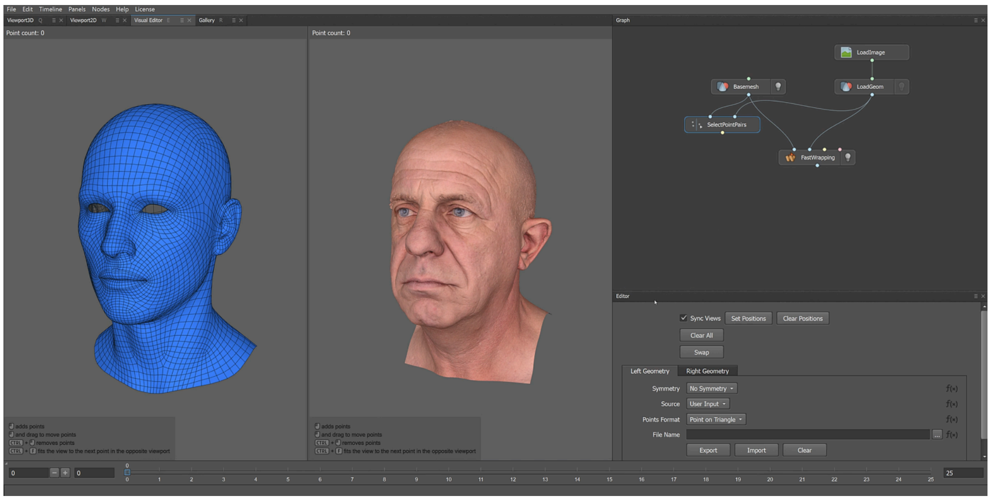
```
sel.add("face")
dag, mObject = sel.getComponent(0)
mfn_components = omapi2.MFnSingleIndexedComp
mfn_object = mfn_components.create(omapi2.MF
mfn_components.addElements(Vertex)
selection_list = omapi2.MSelectionList()
selection_list.add((dag, mfn_object))
omapi2.MGlobal.setActiveSelectionList(select
select -cl  ;
import maya.api.OpenMaya as omapi2

Muscle1 = [53, 56, 57, 60, 61, 150, 151, 287

sel = omapi2.MSelectionList()
sel.add("face")
dag, mObject = sel.getComponent(0)
mfn_components = omapi2.MFnSingleIndexedComp
mfn_object = mfn_components.create(omapi2.MF
mfn_components.addElements(Muscle1)
selection_list = omapi2.MSelectionList()
selection_list.add((dag, mfn_object))
omapi2.MGlobal.setActiveSelectionList(select
```

Figure 3.6: Validation of the muscles array in the Maya viewport

3.1.3 Retopology Strategy

A critical requirement for the success of the retargeting pipeline was ensuring structural consistency across different 3D face meshes. Specifically, the system relies on stable vertex indexing to accurately map muscle regions and apply deformation data. However, conventional modeling tools like Maya often alter vertex order during mesh operations, which compromises the transferability of deformation logic.

To address this, the project employed a topology transfer strategy using FaceForm Wrap, a tool that enables the projection of a base template mesh's topology onto new geometries. This approach ensures that all meshes used in the pipeline share a consistent vertex structure, allowing for the reliable application of precomputed muscle patches and blendshape deformations.

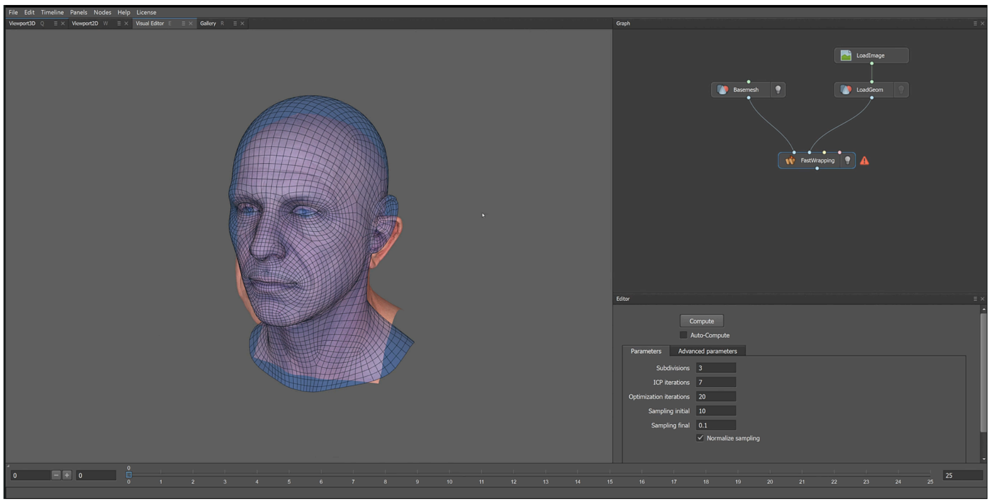


Sync Views [✓] · Set Positions · Clear Positions · Clear All · Swap

Left Geometry | Right Geometry

Symmetry: No Symmetry
Source: User Input
Points Format: Point on Triangle
File Name:

Figure 3.7: FaceForm Wrap Retopology [Example 1]



Auto-Compute [ ]

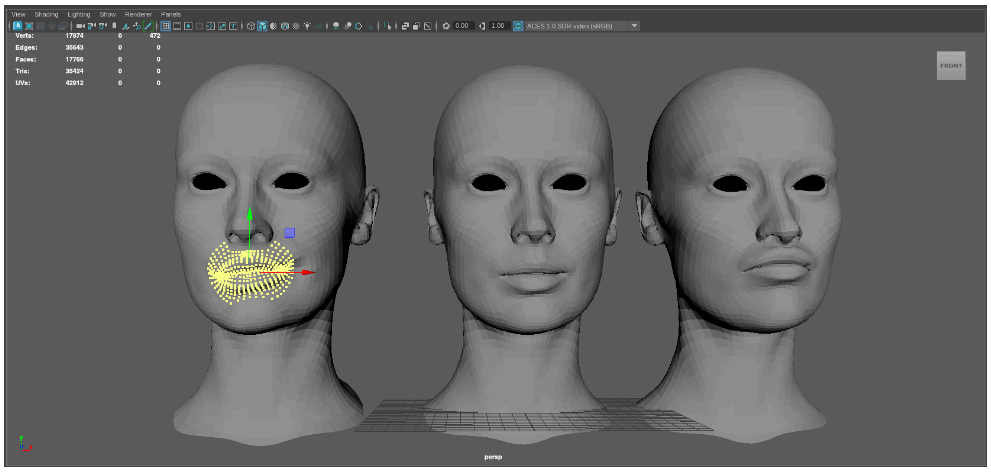
Parameters | Advanced parameters

Subdivisions: 3
ICP iterations: 7
Optimization iterations: 20
Sampling initial: 10
Sampling final: 0.1
Normalize sampling [✓]

Figure 3.8: FaceForm Wrap Retopology [Example 2]

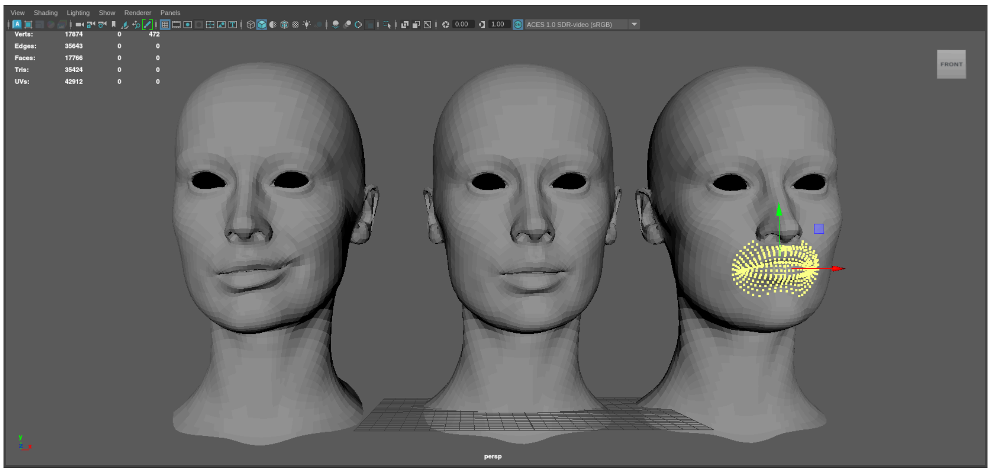
3.1.3.1 Topology Validation

To confirm the integrity of the transferred topology, several meshes were retopologized and tested against the base template. Validation was performed by reapplying the muscle selection scripts and verifying that the spatial distribution and vertex counts matched expected values. This step was essential to guarantee that the pipeline could generalise across different characters while maintaining anatomical fidelity.



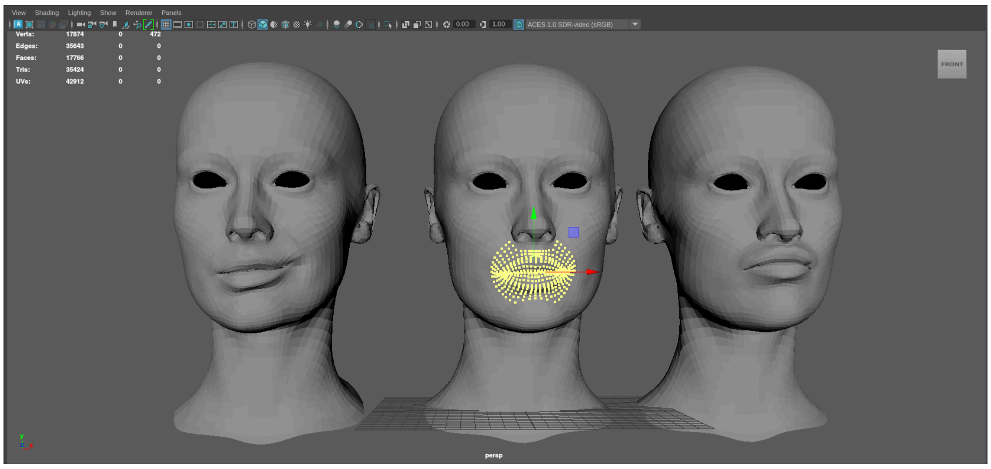
```
Verts:    17874    0    472
Edges:    35643    0      0
Faces:    17766    0      0
Tris:     35424    0      0
UVs:      42912    0      0
```

Figure 3.9: Topology Validation of the Muscle26_OrbicularisOris_OOrs [Example1]



```
Verts:    17874    0    472
Edges:    35643    0      0
Faces:    17766    0      0
Tris:     35424    0      0
UVs:      42912    0      0
```

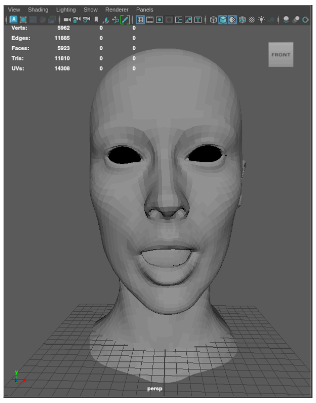
Figure 3.10: Topology Validation of the Muscle26_OrbicularisOris_OOrs [Example 2]



```
Verts:   17874   0   472
Edges:   35643   0   0
Faces:   17766   0   0
Tris:    35424   0   0
UVs:     42912   0   0
```

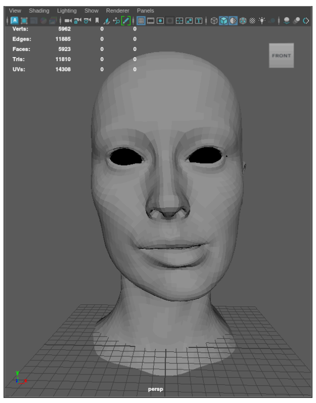
Figure 3.11: Topology Validation of the Muscle26_OrbicularisOris_OOrs [Example 3]

3.1.4 Blendshape for Expressive Deformation



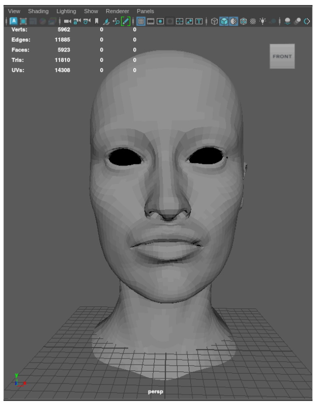
```
Verts:   5962    0   0
Edges:   11885   0   0
Faces:   5923    0   0
Tris:    11810   0   0
UVs:     14308   0   0
```

(a) AU25



```
Verts:   5962    0   0
Edges:   11885   0   0
Faces:   5923    0   0
Tris:    11810   0   0
UVs:     14308   0   0
```

(b) AU30 - Left



```
Verts:   5962    0   0
Edges:   11885   0   0
Faces:   5923    0   0
Tris:    11810   0   0
UVs:     14308   0   0
```

(c) AU17

The present project adopts a blendshape-based strategy. Each AU is represented by a localised deformation space. These spaces are defined by sets of active and passive muscle regions, each associated with specific vertex indices on the mesh. The deformation of these regions is governed by a delta transfer vector, which encodes the directional displacement of each vertex when a given AU is activated.

The blendshapes serve as templates that encapsulate the expressive potential of each AU. To construct them, the researcher used the Eisko model, an open-source facial rig with built-in controls. After analysing the visual manifestation of each AU, the researcher, manually replicated the corresponding expressions for both the left and right sides of the face. Each mesh was exported as an .obj file. TO ensure compability, each mesh was retopologized using the FaceForm Wrap to ensure compatibility with the base mesh topology.

This process was repeated for 50 Action Units, resulting in a comprehensive set of blendshapes. These were then processed to compute the delta transfer vectors and stored in a structured format (deltaTransfer.json). The implementation of this system is encapsulated in the ActionUnit.cpp and ActionUnit.h classes, which handle the deformation logic during runtime. All calculations were precomputed to optimise performance and ensure real-time responsiveness.

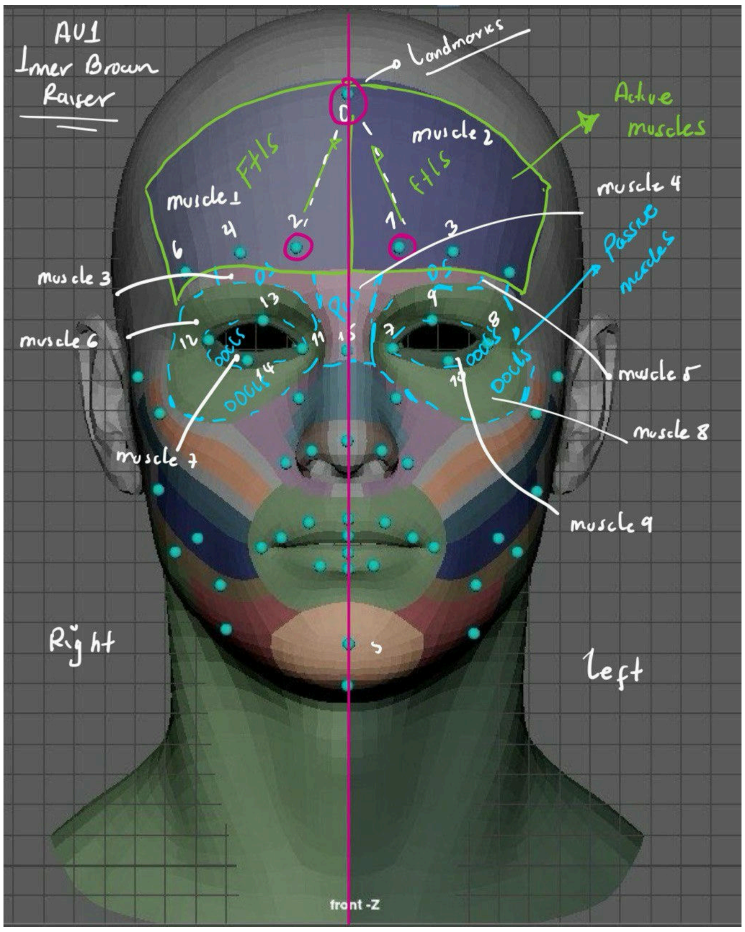


Figure 3.13: Example of Action Units, Landmarks and Muscles Relation

3.2 Precomputed Data Assets

To ensure real-time performance and scalability, the system relies on a set of precomputed data assets stored in structured JSON files. These assets encapsulate the core semantic relationships between facial landmarks, muscle patches, and AUs, allowing the plugin to operate efficiently without recalculating deformation logic at runtime.

Upon initialisation, the plugin loads these files into memory and maps them to custom data structures tailored to the pipeline's requirements. This design choice reflects a deliberate trade-off between computational cost and responsiveness, enabling expressive animation without compromising speed.

3.2.1 Delta Transfer

The deltaTransfer.json file encodes the deformation vectors for each vertex within every muscle patch, across all AUs. These vectors define how each region of the mesh should respond when a specific AU is activated. With approximately 750 vertices per AU and 50 AUs in total, the dataset exceeds 400,000 entries—making real-time computation impractical.

To address this, the deformation data was precomputed using the modelpaths.json file, which provides relative paths to all required assets, including the plugin binary and the muscle patch definitions. The resulting delta vectors are stored in a map structure optimized for fast lookup during animation playback.

3.2.2 Landmarks–Action Unit Mapping

The landmarksActionUnits.json file defines the semantic relationship between facial landmark pairs and the Action Units they influence. This mapping ensures that landmark-driven deformations align with the expressive intent of each AU, as defined by the blendshape templates.

During runtime, the system calculates Euclidean distances between landmarks in the input portrait and the generated frames. The landmark with the greatest displacement is used to infer the most likely active AU, enabling dynamic and context-aware facial animation.

3.2.3 Landmark Indexing for Mesh and Pixel Space

To maintain consistency across spatial representations, the files landmarksMeshIndex.json and landmarksPixelIndex.json define the index positions of facial landmarks in both 3D mesh space and 2D pixel space. These mappings were manually curated to ensure accurate tracking and alignment, forming the foundation of the landmark-based deformation system.

3.3 Source Motion Analysis

The motion source in this system is speech-driven. The model takes a neutral portrait image and an audio file as input, and predicts the displacement of facial landmarks over time based on the semantic and acoustic features of the audio. This approach enables expressive animation without requiring traditional motion capture setups.

3.3.1 Landmark Detection Model

The system for facial tracking uses Dense Facial Landmarks; to capture facial geometry, focusing on the density and distribution of key regions. Instead of sparse landmark sets, it targets anatomical intersections where facial muscles meet and affect expression.

From the full set of 478 detected landmarks, a curated subset of 51 key points was selected for their relevance to muscle-based deformation and compatibility with the retargeting pipeline. This subset strikes a balance between expressive detail and computational efficiency, enabling the accurate interpretation of facial motion while supporting real-time performance.

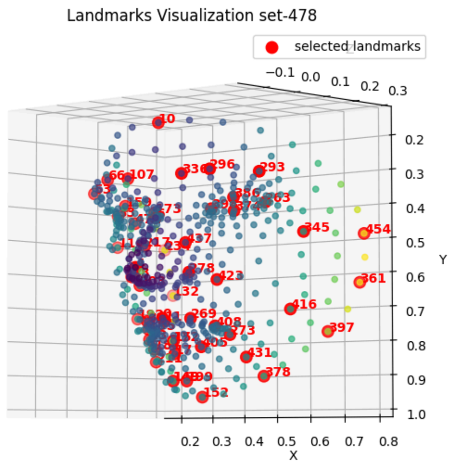


Figure 3.14: A matplotlib representation of a Face Input with 51 Landmarks

3.4 Conceptual Workflow of the Retargeting Pipeline

Once the input video is generated, the system proceeds through a series of deformation stages. First, vertex and muscle region data are extracted from the input mesh. Facial landmarks are then detected in both the neutral and current frames, and their displacements are analysed to compute activation levels for each AU. These activations drive localised skinning and muscle-based deformations, which are transferred to the final mesh using proximity mapping techniques.

3.4.1 Mesh Vertex Extraction

To initialise the deformation process, the input mesh is parsed using the TinyObjLoader library, which efficiently reads .obj files and extracts geometric data. The vertex positions are stored in a vector of type glm::vec3, representing the spatial coordinates of each point in the mesh.

3.4.2 Muscle Region Extraction

Once the input mesh has been loaded and its vertices stored, the system proceeds to identify and extract the muscle regions. This is achieved by cross-referencing the mesh vertex data with the predefined muscle patches, which are stored as indexed regions corresponding to anatomical zones.

3.4.3 Landmark Extraction (Neutral & Current Frame)

With the input mesh loaded and the facial landmark system initialized, the pipeline extracts landmark positions from both the neutral portrait image and the current animation frame. These landmarks serve as reference points for measuring facial motion over time.

3.4.4 Landmark Displacement and Distance Computation

Once the landmarks are extracted from both the neutral portrait and the current animation frame, the system computes their spatial displacement to quantify facial motion. This is achieved by calculating the Euclidean distance between corresponding landmark pairs across frames.

3.4.5 Action Unit Activation

The displacement value reflect the dynamic changes caused by speech input and serve as the basis for Action Unit (AU) activation inference. By identifying which landmarks exhibit the greatest movement, the system can determine which muscle regions are most likely engaged, enabling anatomically accurate deformation.

This analysis transforms raw landmark data into semantically meaningful motion, bridging the gap between audio-driven input and expressive facial animation via Action Unit.

3.4.6 Landmark-Based Skinning

To translate landmark motion into mesh deformation, the plugin generates one MObject instance within the Maya viewport: the muscle mesh. This is derived from the input .obj file and serve as the foundation for the deformation system.

Using the 3D landmark data extracted from the mesh, the system automatically positions joints at key anatomical locations. These joints were thinking to be bound to the muscle mesh and deform another mesh which represetn sking via proximity wrap.

This approach allows the system to simulate muscle-based motion in a way that is both anatomically informed and computationally efficient. By leveraging Maya's native rigging tools, the pipeline integrates seamlessly into professional animation workflows while maintaining control over expressive detail.

3.4.7 Muscle Mesh Deformation

The muscle deformation method applies AUs displacements to a Maya mesh by accumulating per-vertex deltas and then applying only the incremental change (delta-of-delta) scaled by the AU's intensity. Given an AU-to-deltas map and an active AU in the frame.

3.5 Plugin Integration

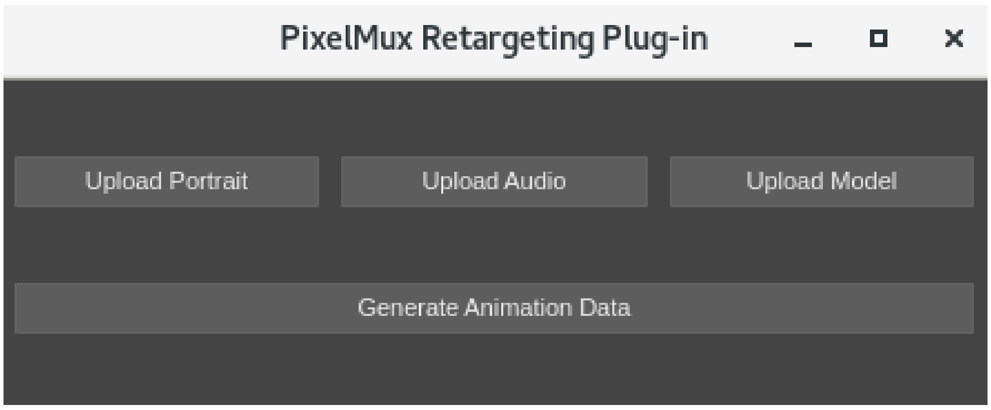


Figure 3.15: Plugin UI

The plugin was developed in Qt and presents a clean, three-step workflow:

- Portrait Input: Load a facial image (JPG or PNG).
- Audio Input: Load the corresponding audio file (WAV).
- Mesh Input: Select the base 3D model (OBJ).

Once all inputs are provided, the user clicks Generate to run the deformation pipeline. Progress is reported via a status bar, and the resulting frames appear directly in the Maya viewport.

Chapter 4
System and Technical Requirements

This project was developed as an extension of the original PixelMux repository, introducing a new feature focused on facial motion retargeting. While the full PixelMux system remains private, this submission includes only the implementation of the retargeting pipeline.

To maintain consistency with the original repository and promote modularity, the system architecture follows a separation between two core directories: **cmd** and **pkg**.

pkg directory contains all reusable components of the system. It includes classes and utilities for facial landmark processing, AU activation, Mesh extraction, and math utils. The code in this directory is designed to be platform-agnostic and can be reused across different Digital Content Creation (DCC) tools such as Blender, Unreal Engine, or Houdini.

cmd directory contains the environment-specific implementation that integrates the reusable logic from **pkg** into **Autodesk Maya**. It includes plugin classes, UI components, and mesh manipulation routines that rely on Maya's DCC interface. The code in **cmd** is not reusable outside of Maya, but can be adapted to other DCCs by replacing the interface layer.

4.1 System Design

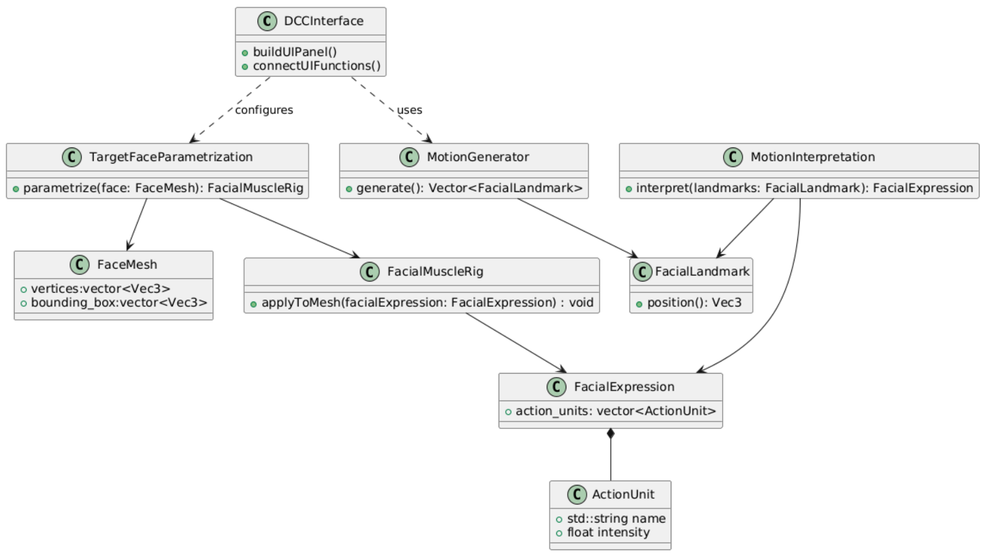

DCCInterface
buildUIPanel()
connectUIFunctions()
configures
uses
TargetFaceParametrization
parametrize(face: FaceMesh): FacialMuscleRig
MotionGenerator
generate(): Vector<FacialLandmark>
MotionInterpretation
interpret(landmarks: FacialLandmark): FacialExpression
FaceMesh
vertices:vector<Vec3>
bounding_box:vector<Vec3>
FacialMuscleRig
applyToMesh(facialExpression: FacialExpression) : void
FacialLandmark
position(): Vec3
FacialExpression
action_units: vector<ActionUnit>
ActionUnit
std::string name
float intensity

Figure 4.16:

4.2 Requirements and Dependencies

The development environment for this project was configured to support cross-platform compilation and integration with Autodesk Maya. The build system is based on CMake, and external dependencies are managed using vcpkg.

4.2.1 System Requirements

- CMake ≥ 3.25
- vcpkg (for dependency management)
- Autodesk Maya 2023.3
- DevKit Maya 2023.3 version
- C++17

4.2.2 Global Dependencies

The project relies on the following external libraries:
- gtest – Unit testing framework for validating core logic.

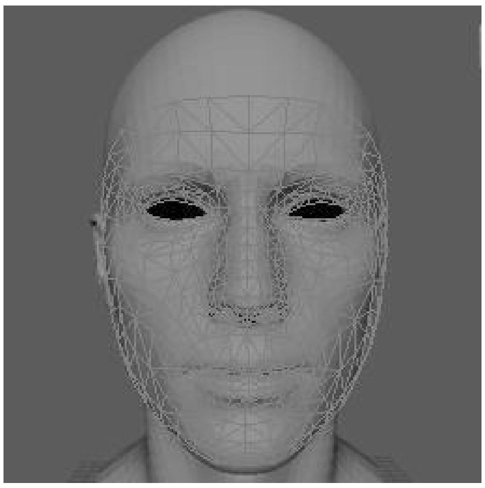
- glm – OpenGL Mathematics library used for vector and matrix operations.
- tinyobjloader – Lightweight .obj file parser for mesh loading.
- nlohmann-json – Modern C++ JSON library used for configuration and data serialization.

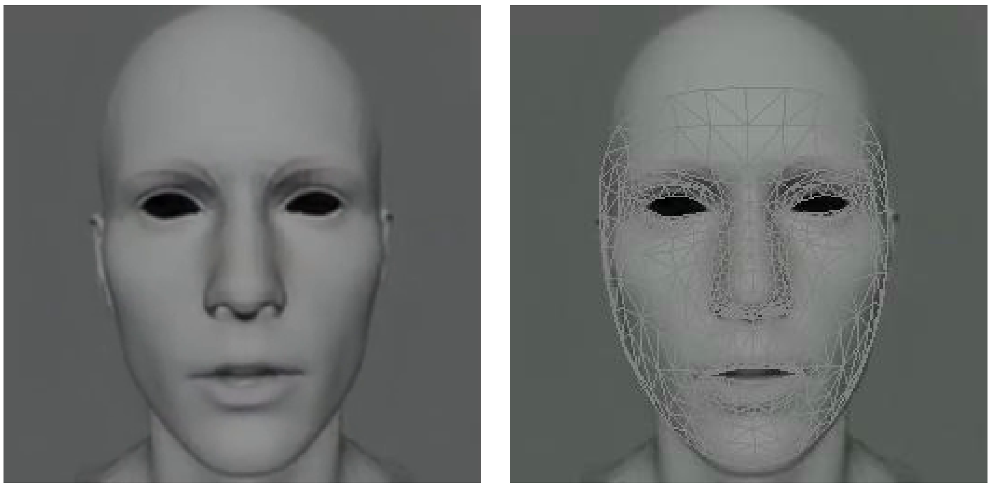
These dependencies are installed and managed via vcpkg, ensuring consistent builds across platforms and simplifying future integration with other DCC tools.

Chapter 5
Results and Improvements

5.1 Neutral Face Input and Pose Face Input

(a) Neutral Face


(b) Neutral Face Landmarks



(a) AI Generated Face (b) AI Generated Face Landmarks

5.2 Output 3D Mesh and Expected Pose Analysis

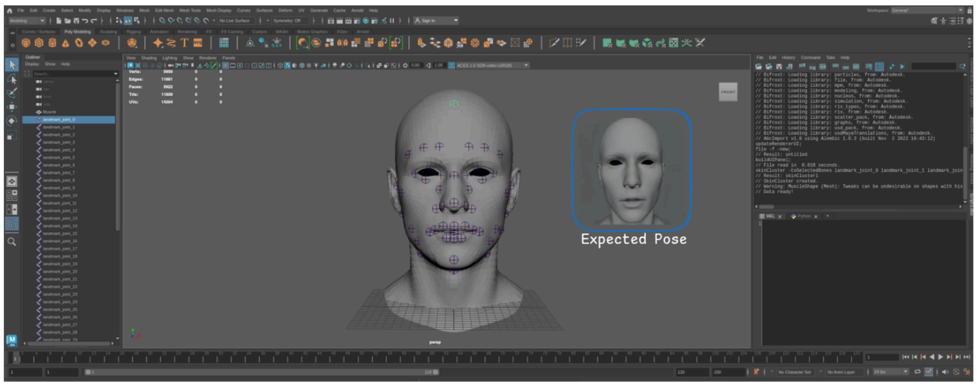


Figure 5.19: Expected Pose and Output 3D Mesh

Visually, the mesh and the expected frame look very similar. Since the generated pose is focused on the mouth, the detected action unit AU16 successfully deformed the model's muscle mesh around the mouth. However, it's important to remember that more accurate facial expression detection requires multiple AUs to be present simultaneously on the face, so the system definitely needs improvement in this area.

On the other hand, the mesh shown corresponds exclusively to the muscles. According to the paper's recommendation, there should be an intermediate mesh that drives the deformation onto a second mesh called the "skin," which would serve as the final output of the plugin. This test has not yet implemented that skin mesh, so it should be added in future iterations.

5.3 Improvements

1. Standardize all AU blendshapes on a single reference mesh. Although most were created from the same base, a few were built on a different yet topologically identical mesh. With additional time, these outliers can be remapped to ensure full consistency.
2. Replace the custom map-based AU to delta transfer with a single contiguous vector. Maps were originally chosen to simplify debugging and mirror Zhu & Joslin's organization, but their non continuous memory layout can hinder real-time performance. Consolidating into one vector will be harder to test but will yield faster, more predictable access.
3. Deepen Maya integration. The current plugin provides basic import and deformation, but it needs further GUI controls, live playback, and error handling to serve as a complete, real-time speech-driven animation tool.
4. Automate PixelMux API integration. Due to time constraints, data loading was handled manually. This process must be refactored to use PixelMux's API directly, streamlining data ingestion and reducing manual steps.

Chapter 6
Conclusion

This thesis set out to implement and validate a facial deformation pipeline driven by Action Units (AUs) within a Maya plugin. Despite time and experience constraints, the research successfully demonstrated that high-intensity AU detection can drive mesh deformations following the principles of Zhu and Joslin. This proof-of-concept confirms the viability of landmark-based deformation in an interactive environment.

The work holds significance for both real-time facial animation and academic exploration of muscle-driven rigs. By integrating the pipeline into Maya, we provided artists with immediate visual feedback and a foundation for more complex controllers. However, the current prototype handles only the most prominent AU per frame, limiting expressiveness and blending fidelity.

Looking ahead, extending the system to process multiple simultaneous AUs will more faithfully reproduce full expressions. Incorporating temporal smoothing and automating landmark selection could further streamline production workflows. A comprehensive user study would also quantify perceptual improvements and guide refinement.

In closing, this project bridges theoretical AU frameworks and practical animation tools. It lays the groundwork for richer, more intuitive expression rigs and opens new directions for research in speech-driven facial animation.

Bibliography

[1] C. Zhu and C. Joslin, "A review of motion retargeting techniques for 3D," *Elsevier*, 2024.

[2] Kucharska, "How long does it take to produce an animation," *Studio Pigeon Blog*, 2023.

[3] L. Zhang, N. Snavely, B. Curless, and S. M. Seitz, "Spacetime Faces: High-Resolution Capture for Modeling and Animation," in *ACM SIGGRAPH 2004 Papers*, 2004, pp. 548–558.

[4] C. Cao, Y. Weng, S. Zhou, Y. Tong, and K. Zhou, "FaceWarehouse: A 3D Facial Expression Database for Visual Computing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 3, pp. 413–425, 2013.

[5] S. Racković, C. Soares, D. Jakovetić, Z. Desnica, and R. Ljubobratović, "Clustering of the blendshape facial model," *European Signal Processing Conference*, 2021.

[6] C. Matthew, K. Bhat, and R. Fedkiw, "Art-Directed Muscle Simulation for High-End Facial Animation," pp. 457–465, 2016.

[7] A. E. Ichim, L. Kavan, M. Nimier-David, and M. Pauly, "Building and Animating User-Specific Volumetric Face Rigs," pp. 107–117, 2016.

[8] B. Choi *et al.*, "Animatomy: An Animator-Centric, Anatomically Inspired System for 3D Facial Modeling, Animation and Transfer," pp. 1–9, 2022.

[9] J. Hamm, C. G. Kohler, R. C. Gur, and R. Verma, "Automated Facial Action Coding System for Dynamic Analysis of Facial Expressions in Neuropsychiatric Disorders," *Journal of Neuroscience Methods*, vol. 200, no. 2, pp. 237–256, 2011.

[10] P. Ekman and W. V. Friesen, "Facial Action Coding System," *Environmental Psychology and Nonverbal Behavior*, 1978.

[11] R. Daněček, M. J. Black, and T. Bolkart, "EMOCA: Emotion Driven Monocular Face Capture and Animation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 20311–20322.

[12] C. G. Fisher, "Confusions Among Visually Perceived Consonants," *Journal of Speech and Hearing Research*, vol. 11, no. 4, pp. 796–804, 1968.

[13] J. M. De Martino, L. P. Magalhães, and F. Violaro, “Facial Animation Based on Context-Dependent Visemes,” *Computers & Graphics*, 2006.

[14] ISO/IEC JTC1/SC29/WG11, “Coding of Moving Pictures and Videos,” 1998. [Online]. Available: http://home.mit.bme.hu/~szanto/education/mpeg/14496-2.pdf

[15] J. Ostermann, “Animation of Synthetic Faces in MPEG-4,” in *Proceedings of Computer Animation'98 (Cat. No. 98EX169)*, IEEE, 1998, pp. 49–55.