



RIG LOGIC

RUNTIME EVALUATION OF
METAHUMAN FACE RIGS

WHITE PAPER



UNREAL
ENGINE



TABLE OF CONTENTS

Introduction	3
Terminology	3
Background	5
The challenge of real-time facial rig evaluation	6
Solution	7
Requirements	7
Reduction of joint deformations	7
Level of Detail (LOD)	9
MetaHuman DNA file format	11
Corrective expressions	12
Runtime Strategy	15
Joints	15
Blend shapes	16
Animated maps	16
Performance	16
Conclusion	17

Introduction

Rig Logic is a runtime facial rig evaluation solver system developed by 3Lateral. Rig Logic is the infrastructural foundation for advanced character customization and player digitization systems that offer high-fidelity rigs, and is also a versatile characterization tool.



Figure 1: Rig Logic user interface

Rig Logic relies on a universal set of rules for defining the muscular system of a human face, and on a proprietary file format from 3Lateral called MetaHuman DNA, which is designed to store the complete description of a 3D object's rig and geometry.

Rig Logic was developed by **3Lateral**, a research and development company that has focused on facial rigging and animation since its founding in 2012. The company's projects include numerous films and games such as *Senua's Saga: Hellblade II* and *Marvel's Spider-Man*. 3Lateral became part of Epic Games in 2019. Rig Logic is the rig solver currently used by Epic Games' **MetaHuman Creator** tool.

In this paper, we will explain the reasoning behind the MetaHuman DNA file format and other decisions we made in building the Rig Logic system, and why it works for real-time rendering of facial poses and animation.

Terminology

To explain how Rig Logic evolved to a full real-time solution for facial rigging, a few terms must be defined.

Expression - A specific facial representation, such as a smile, frown, or phoneme. In facial rigging and animation, an expression is defined by a set of joint transforms (translation, rotation, or scaling), blend shapes (also known as morph targets), and animated maps.



Figure 2: Facial expression on a MetaHuman character

Joints - A facial rig consists of virtual bones and joints that mimic, to some degree, the actual bones, joints, and muscles in the face. Because real-life muscles are difficult to mimic exactly, virtual joints work in their place to drive the animation of vertices in a facial model. Each joint is assigned a set of vertices that it animates.

Animation curve - A representation of the XYZ movement of a vertex over time.

Control - A mechanism that holds the weight for a single basic expression, which is used to compute joints, blend shapes, and animated maps that participate in the expression. Ideally, there is one control per **FACS Action Unit**.

Rig control interface - Any interface that combines controls or makes them easier to use. This could be any user interface, from a graphical user interface (GUI) with sliders, to an internal interface that gathers and uses controls to cause another action.

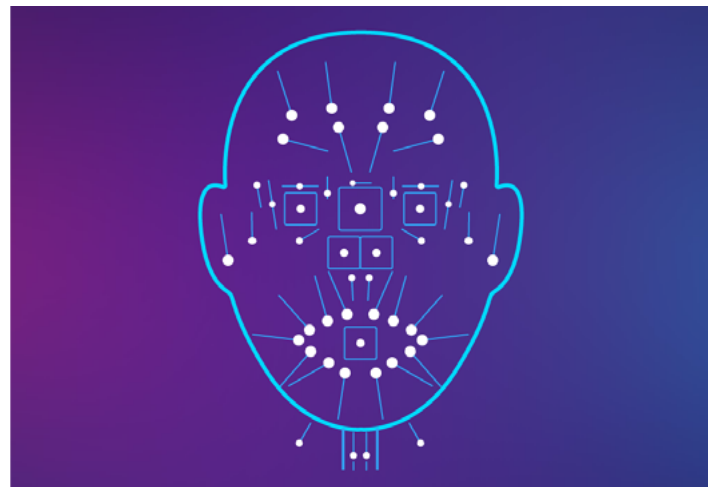


Figure 3: Rig control interface as GUI with sliders

Complexity - A general term referring to the number of rig controls a facial rig has. A rig with 200 controls is said to be more complex than a rig with 50 controls. The more controls there are, the more potential exists for combinations of expressions that will require corrective expressions, resulting in more computation and higher memory requirements.

Corrective expressions - Internal tweaks that refine complex expressions. Corrective expressions are discussed in more detail later in this paper.

Background

Over the course of its evolution, Rig Logic has gone through several major iterations. The latest generation represents the collective work and refinements accumulated through years of research and development.

In a successful facial rig, input values are controls that each represent a single facial **FACS** expression. From such input, calculations occur to produce the following output values:

- Joint transformation
- Blend shape channel weights
- Shader multipliers for animated maps

These three output values are the core of most successful facial rig systems in use today. Our research has been largely aimed toward making the calculation and output of these three values more accurate and efficient.

When 3Lateral first started to work with real-time rigs, they were built entirely using the toolkit provided by Maya, a digital content creation package from Autodesk and a popular choice among 3D artists in games, film and television, and other fields. The Maya toolkit includes Driven Keys (DKs) complemented with corrective expressions and a shader network. Deformations were computed using these abstractions only, which provided us with the three output values needed to successfully animate the face.

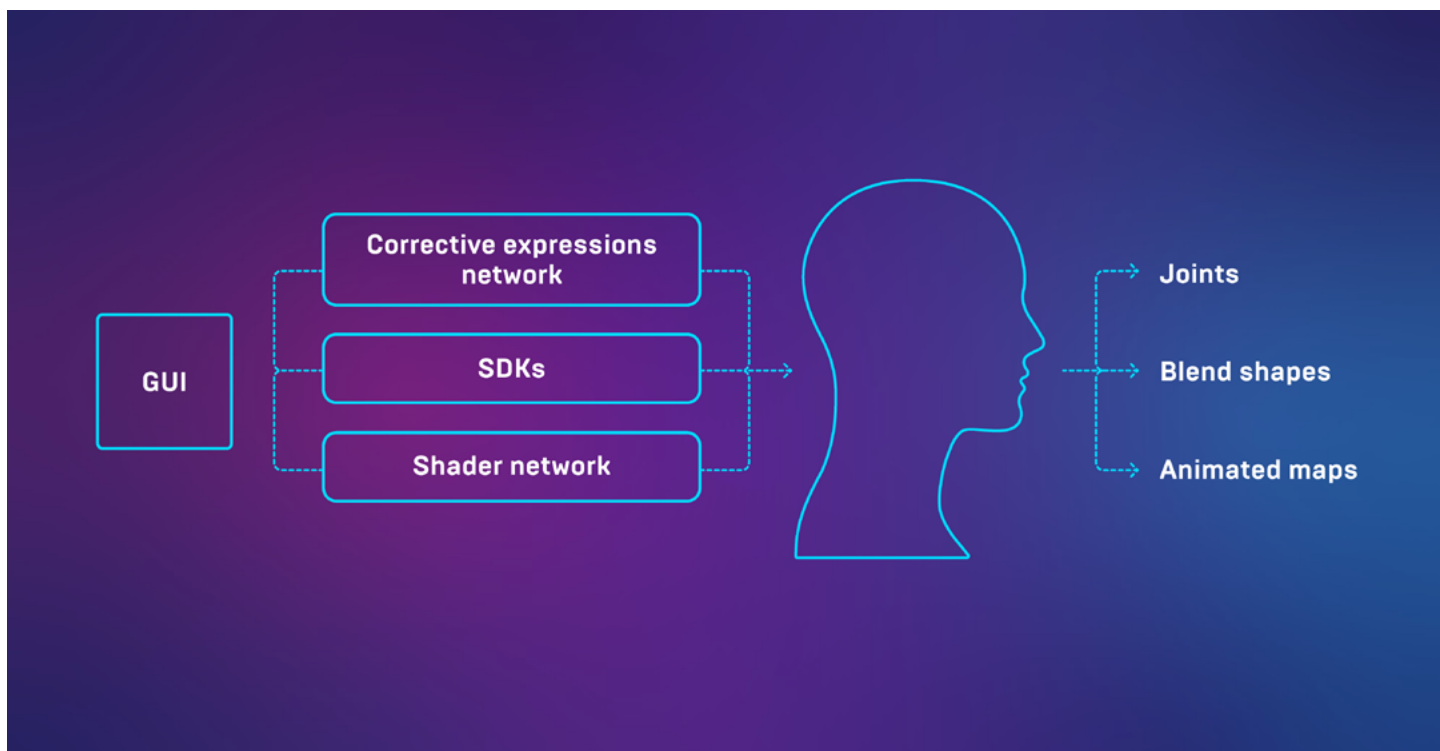


Figure 4: Flow of data in original 3Lateral real-time facial rig evaluation process

While this solution worked well and provided a high level of fidelity, rig evaluation was very slow, and integration with other technologies was complicated.

The challenge of real-time facial rig evaluation

The challenge we faced was to develop a facial rigging system that produced the same or similar level of fidelity while computing in real time at 30 fps or higher. We also wanted it to be possible to integrate the system with other technologies.

At its core, Rig Logic was tasked with computing output values representing deformations based on the same types of input values that Maya uses, but running much faster. We also added the goal of making the rig reusable across characters regardless of facial shape, making the rig solution highly exportable and much more useful to various technologies.

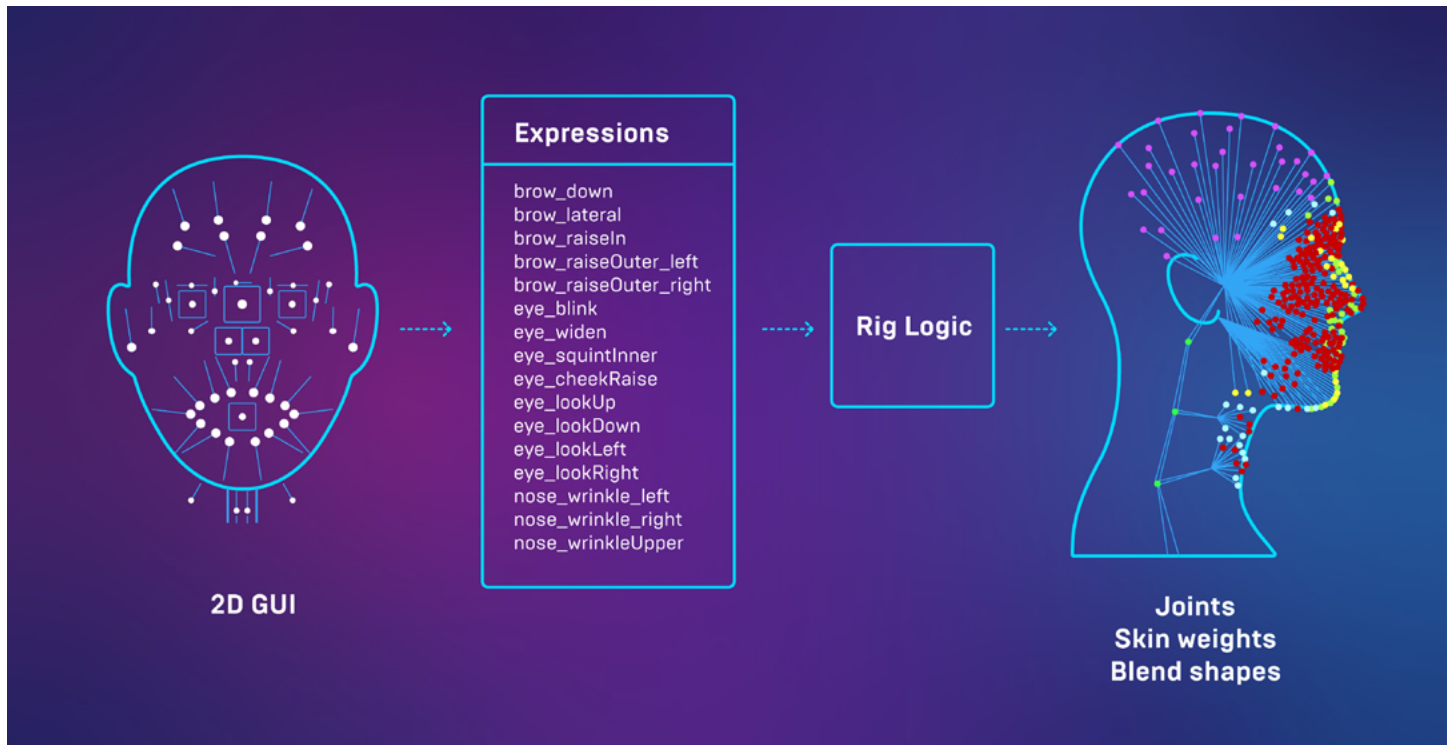


Figure 5: Data flow from rig controls to expressions, to Rig Logic computation, and finally to facial joints to pose the face into the desired expression.

A state-of-the-art facial rig for producing high-fidelity facial animation contains around 800 joints and more than 200 controls for expressions, complemented with 1,000+ corrective expressions, meaning we had our work cut out in developing a runtime solution that could accommodate real-time output.

The most important elements we use for MetaHuman Creator facial rigs are:

- Joints and skin weights deformer
- Blend shape deformer
- Base and animated textures setup
- Rig Logic plugin and MetaHuman DNA file
- User interface
- Built-in LODs

Solution

Requirements

To achieve a viable runtime solution for facial rigging and animation, Rig Logic set out to address the following requirements for the rig and its operation:

- **Runtime evaluation.** Ideally, runtime evaluation of the facial rig would mean real-time playback of animation at a reasonable speed such as 30 fps or more.
- **Reduction of input and output parameters.** Reducing the number of parameters naturally makes the system run faster.
- **Lossless animation compression.** Compressed data is faster to process due to its smaller footprint. Data compression usually provides a reduction of 10–15x reduction in data size at minimal CPU cost, but this reduction is viable only if such compression is lossless.
- **Reusability of rigs.** The system should be designed such that there is no rig dependency on facial shape, proportion, demography, etc., making it possible to easily share animation data between different characters.
- **Flexible LOD usage.** The same animation should be able to drive all LOD specifications.
- **More efficient pipeline.** Creating a rig that evaluates at runtime naturally leads to certain pipeline efficiency improvements. For example, exporting of animation would be much lighter due to much smaller data sizes, and if the system is designed correctly, rig updates do not require re-exporting of all animations.
- **Links easily with any facial animation software.** Because the solution relies on a muscle-based organization of the rig, it's straightforward to integrate with audio-based facial animation solutions such as [SpeechGraphics](#).
- **Support non-linear animation mixing.** The system would need to support dynamic responses at runtime. For example, a character could carry on a performance while having an interactive response to an in-game event, such as a flashlight shining in the character's eyes. The solution should also support emotion layering in animation.

Reduction of joint deformations

To decouple rigs from Maya, Rig Logic was developed as an attempt to capture the necessary data that represents facial rigs, and to provide a runtime component for evaluating it in real time. The first generations of Rig Logic just did that: extracted the data from Maya and represented Driven Keys with the linear function:

$$y = k * x$$

The y value increases linearly as x increases, and the degree to which y increases (the slope) is determined by k .

- x is the input control value, restricted to the range $[0.0, 1.0]$
- y is the output deformation value (e.g. value of joint.rotation.x parameter)
- k is the rig parameter that defines how the function behaves

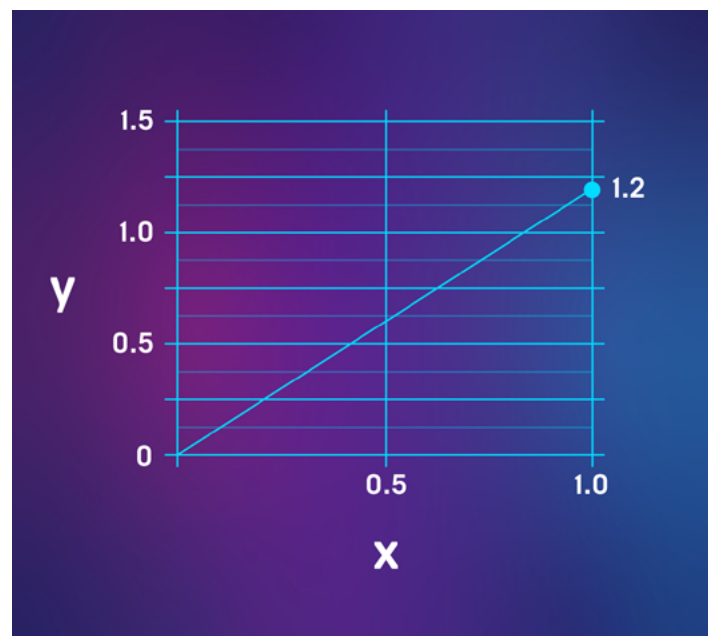


Figure 6: Representation of slope function $y=k*x$ for determining x (input), y (output), and k (rig parameter) values.



Figure 7: Facial expression on a MetaHuman character

Treating the x and y values in this way made it possible to encode the parameters of these slope functions in a matrix.

A sparse **Compressed row storage (CRS) matrix** was chosen as the matrix type, with the goal of minimizing storage space requirements and giving a decent computational performance. Geometric primitives were not addressed in this version of Rig Logic; the system still relied on geometric primitives (meshes) in Maya.

Later analysis of this matrix revealed that dense groups of these parameters were forming in the otherwise mostly sparse matrix. It was also observed that these same dense submatrices happened to contain parameters for deformations targeting specific regions on the face, thus the concept of *joint groups* was introduced. Joint groups represent these dense submatrices and also provide a logical separation of joints into groups that target specific regions of the face.

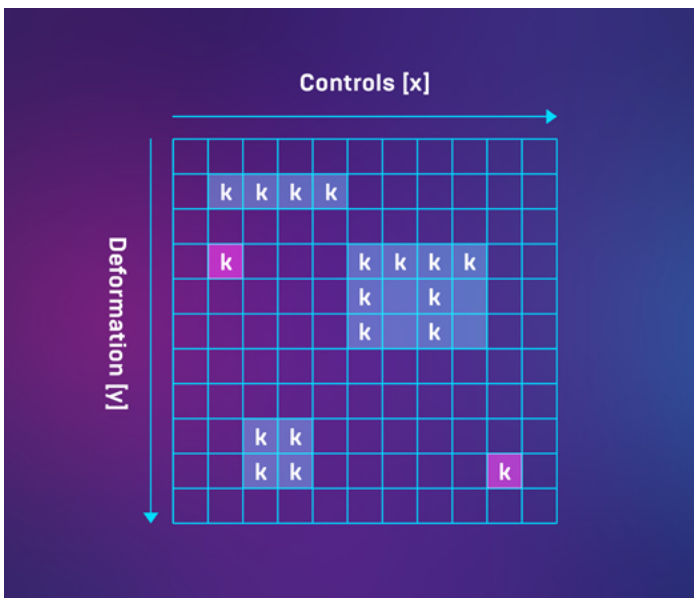


Figure 8: Representative sample matrix of joint transformation deltas [k] as calculated with the linear function $y = k*x$.

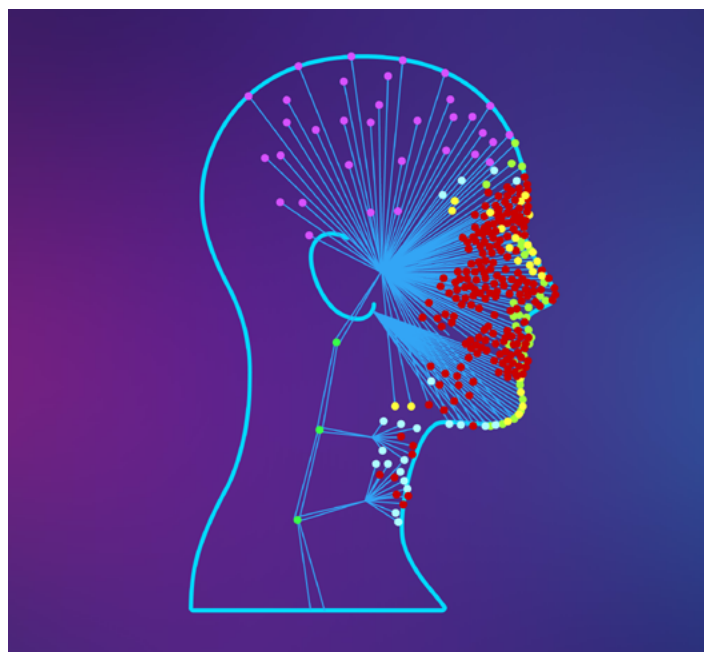


Figure 9: Joint groups

As can be seen in Figure 8, there were also outliers—parameters that were not part of a dense submatrix. (Outliers are represented as pink in the diagram.) These outliers turned out to represent deformations on regions of the face where the joints in question shouldn't have any influence at all. As a result of this observation, a smart pruning process was implemented to get rid of these outliers, leaving the matrix with only the densely populated regions.

This pruning process effectively reduced the data that needed to be stored and evaluated at runtime and got rid of incorrect deformations, and the resulting dense regions provided a good opportunity to implement a very fast algorithm for evaluating joints.

Later iterations also separated distinct data types—joints, blend shapes, and animated maps—into separate matrices, each of them representing a conditional linear function.

Level of Detail (LOD)

It is common for a facial expression to consist of two or more basic expressions. An example of a complex expression would be a combination of three basic expressions: (1) raising the eyebrows while (2) opening the eyes wide and (3) parting the lips slightly.

To provide the highest possible efficiency for these complex expressions, Rig Logic was designed with LOD support in mind. Recall that every expression is defined by transformations of specific joints, but not all joints participate in any given expression. At LOD 0, the most detailed level, all the referenced joint transforms for that expression are used to form the expression. However, lower LODs (LOD 1, LOD 2, etc.) use a strict subset of the joints referenced in LOD 0.

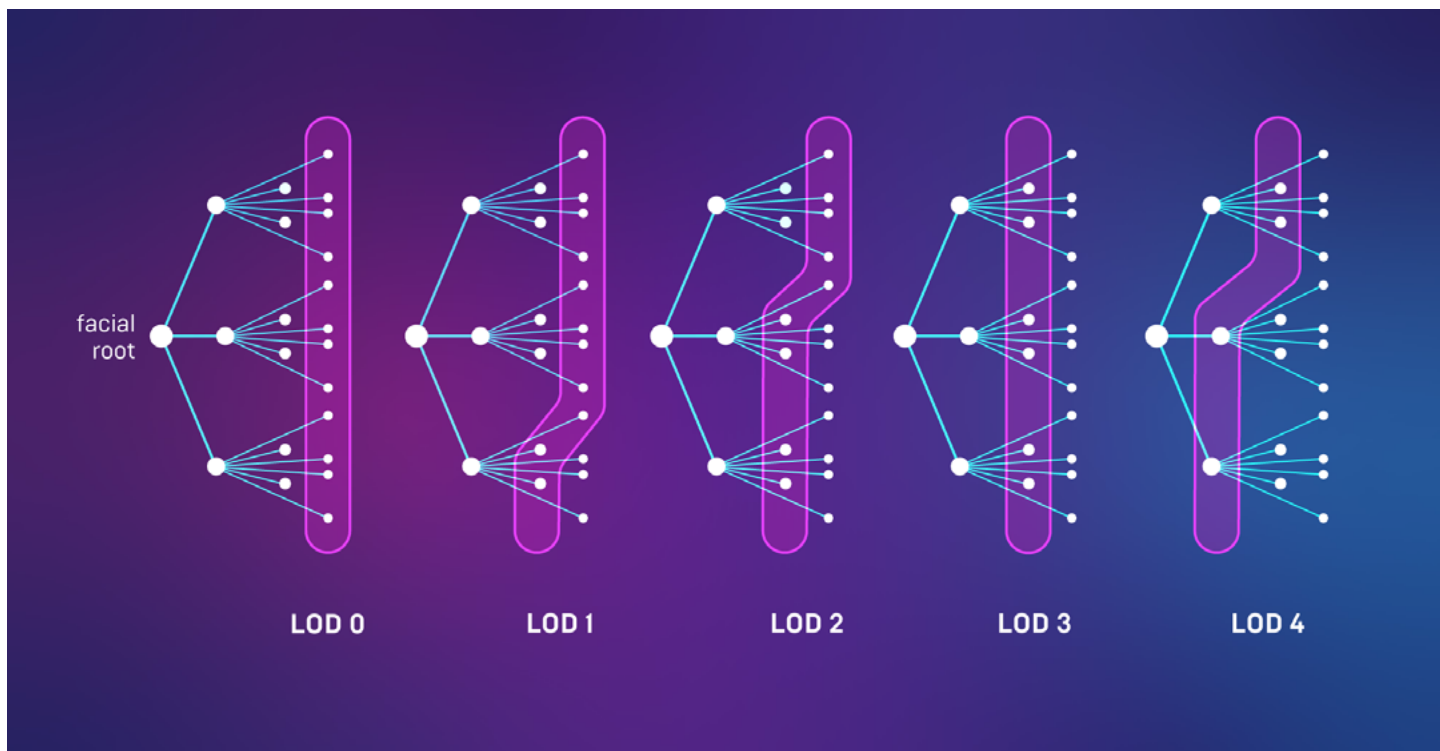


Figure 10: Diagram showing the change in the number of associated joints from LOD 0 to LOD 4, from the densest topology and maximum number of joints (LOD 0) to least dense topology and minimum number of joints (LOD 4).

Limiting the joints referenced in lower LODs to the same joints referenced in LOD 0 provides a zero-cost mechanism for switching between LODs, with minimum data waste. Moving from a lower to higher LOD simply adds the specified joints to the data already loaded, and moving from a higher to lower LOD simply disregards the unreferenced joints.

While there are eight LODs built into the system, Rig Logic can be customized to add more if needed. Depending on the joint count constraint and quality of deformations required for different parts of the face, the system can easily combine joints and skin weights with different topologies to form different LODs.

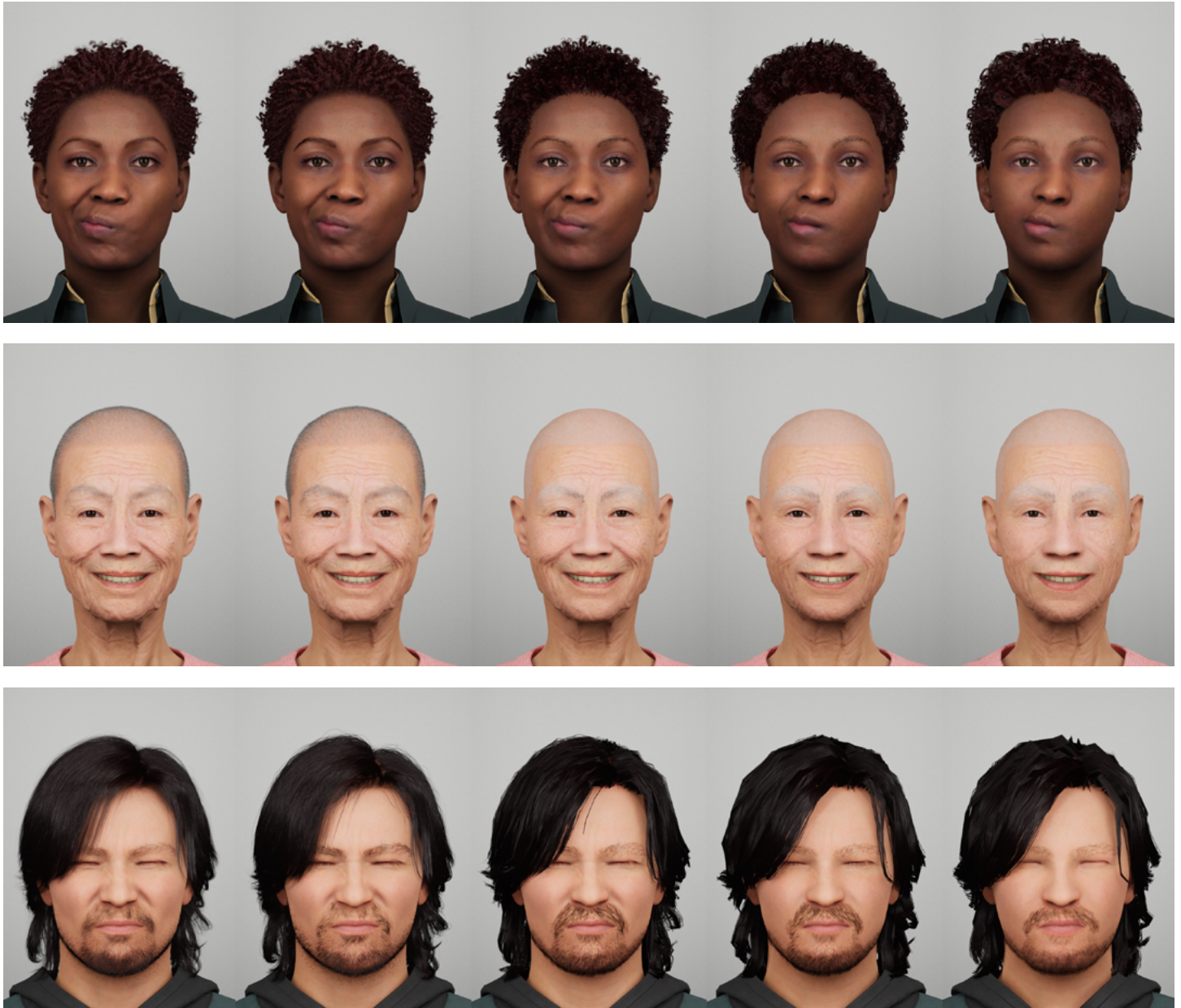


Figure 11: Examples of expressions represented as higher to lower LODs, with zero-cost switching between LODs.

To further reduce memory usage, it is also possible to specify the expected LODs that will be needed at runtime (or specify a maximum and/or minimum LOD). Any data that is not referenced by the chosen LODs will not be loaded.

MetaHuman DNA File Format

The 3Lateral MetaHuman DNA file format is designed to store the complete description of a 3D object's rig and geometry. Relying only on a MetaHuman DNA file, it is possible to reconstruct the complete mesh of an object and have it fully rigged, ready to be animated. In practice, MetaHuman DNA files are used to store only the faces of human characters, and not their bodies, props, or other objects.

The contents of a MetaHuman DNA file are encoded using a proprietary binary format, although the format specification, as well as the code that reads it, is open source.

Layers

Data contained within MetaHuman DNA files is separated into several logical layers. Layers are connected into a loose hierarchy, where each subsequent layer in a MetaHuman DNA file relies on data stored in the layers below it.

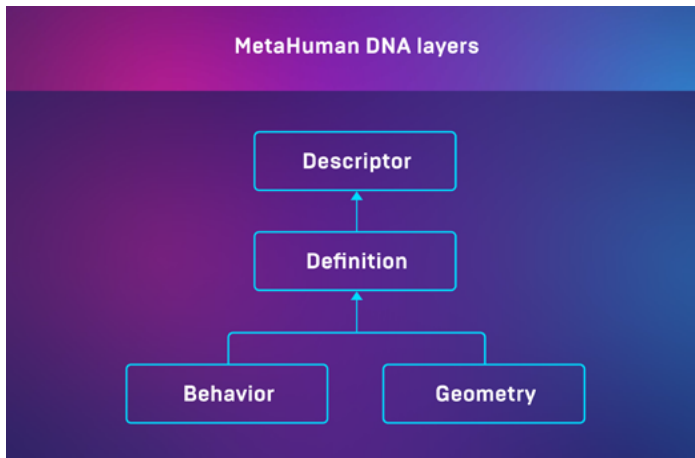


Figure 12: Logical layers in MetaHuman DNA files

It is possible to selectively load MetaHuman DNA data only up to a specified layer. As can be seen in Figure 12, the Behavior and Geometry layers are not dependent upon each other. This independence is crucial for use cases where the MetaHuman DNA file is needed only to drive a rig [runtime evaluation using the Behavior layer] without the need to access the Geometry data.

This independence makes the solution portable, so it doesn't have to be applied to Maya geometry or any other specific geometry, and can be used with other applications where facial geometry resides. The file format was carefully designed to produce an efficient yet complete format for real-time facial animation.

Descriptor layer

The Descriptor layer contains basic metadata about the rig, such as:

- Name of the character
- Age
- Facial archetype
- Arbitrary string metadata in the form of key/value pairs
- Compatibility parameters as needed (relevant for higher-level systems, e.g. for mixing MetaHuman DNA files)

Definition layer

The Definition layer contains the static data of the rig, such as:

- Names of controls, joints, blend shapes, animated maps, and meshes
- Mappings of joints, blend shapes, animated maps, and meshes to individual LODs
- Joint hierarchy
- Joint transformations of the binding pose (such as T-pose)

This layer contains necessary information to perform filtering in the subsequent layers based on the chosen LODs.

Behavior layer

The Behavior layer contains the dynamic data of the rig, which is used to:

- Map GUI controls to raw control values
- Compute corrective expressions
- Compute joint transformations
- Compute blend shape channel weights
- Compute animated map weights

Geometry layer

The Geometry layer contains all the data needed to reconstruct the mesh of the character, along with its skin weights and blend shape target deltas. The mesh information itself is structured in a format resembling the OBJ file format.



Figure 13: Facial expression on a MetaHuman character

Corrective expressions

For complex expressions, there will be some overlap, where some joints are affected by more than one basic expression. In such a case, simply activating all the specified deformations is likely to lead to 'multiplied' deformations, which are usually detrimental to the visible result.

Corrective expressions represent the solution to correct joint overlap in complex expressions. For each complex expression, a corrective expression must be defined that essentially undoes the damage caused on these problematic regions. Such complex expressions must be defined for each combination of basic expressions (or at least the combinations that are considered important).

Therefore, when the number of supported basic expressions rises, the number of complex expressions exponentially rises as well. Due to memory and computation costs, it is not feasible to enable a corrective solution for every possible combination, but the most important combinations can be served.

Figure 14 shows an example of facial deformations to represent a facial pose for angry shouting. The creation of such a pose might require a combination of several basic expressions:

- jawOpenExtreme
- mouthCornerPull
- mouthStretch
- neckStretch
- noseWrinkler
- squintInner
- browsLower

There are likely to be several joint overlaps for these expressions. For example, the mouth and nose expressions, when summed together without correction, are likely to cause extreme stretching around that area of the face.

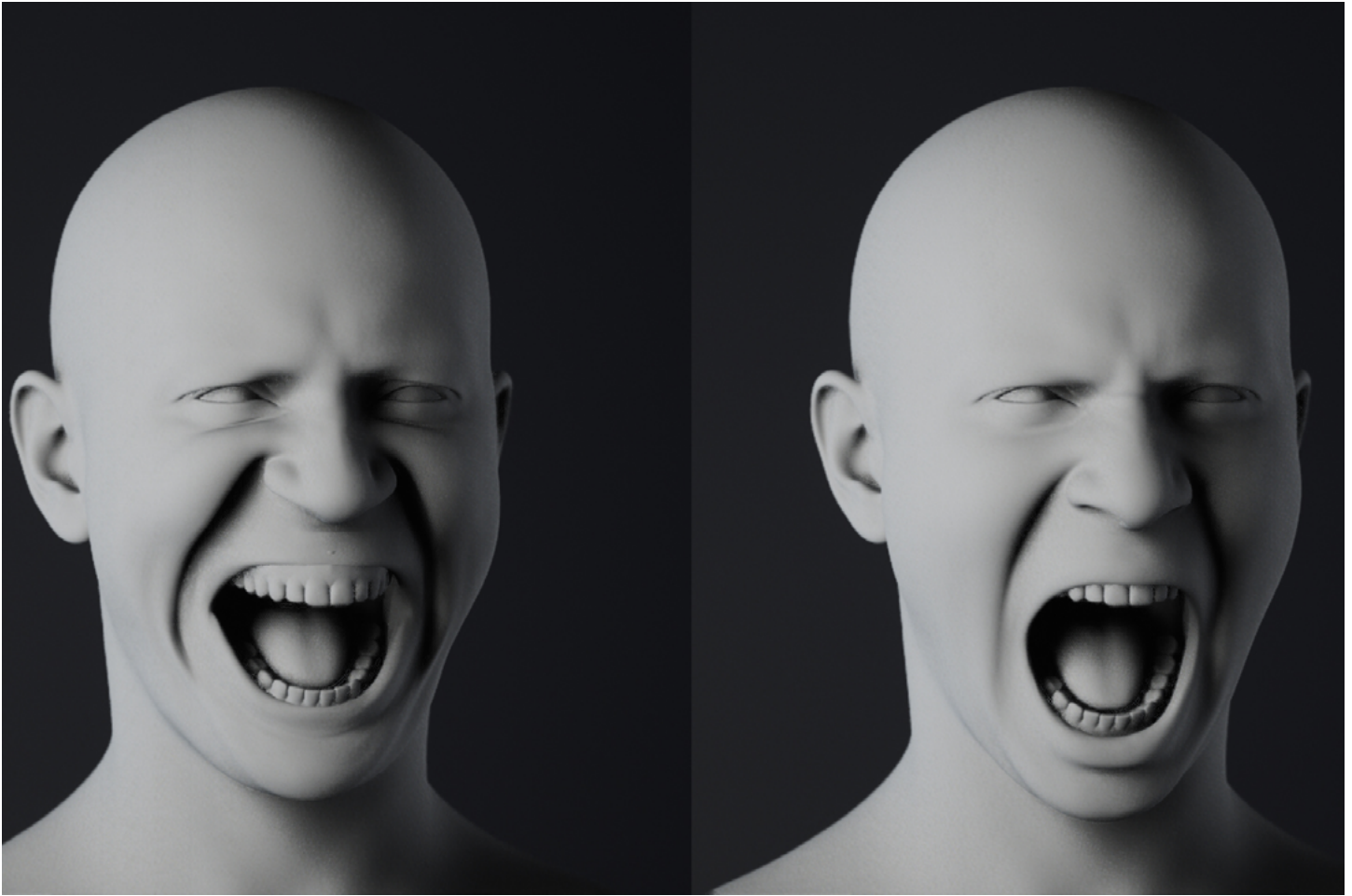


Figure 14: Facial poses before (left) and after (right) application of corrective expressions to a specific combination of basic expressions.

The head on the left shows a straightforward sum of the expressions listed above, which results in an unnatural pose. The solution is to adjust the pose with specific corrective expressions every time this specific combination of expressions is activated.

The head on the right shows the facial pose after corrective expressions have been applied for this combination of expressions, resulting in a much more natural and convincing pose.

Additional examples of corrective expressions in action are shown in Figure 15.

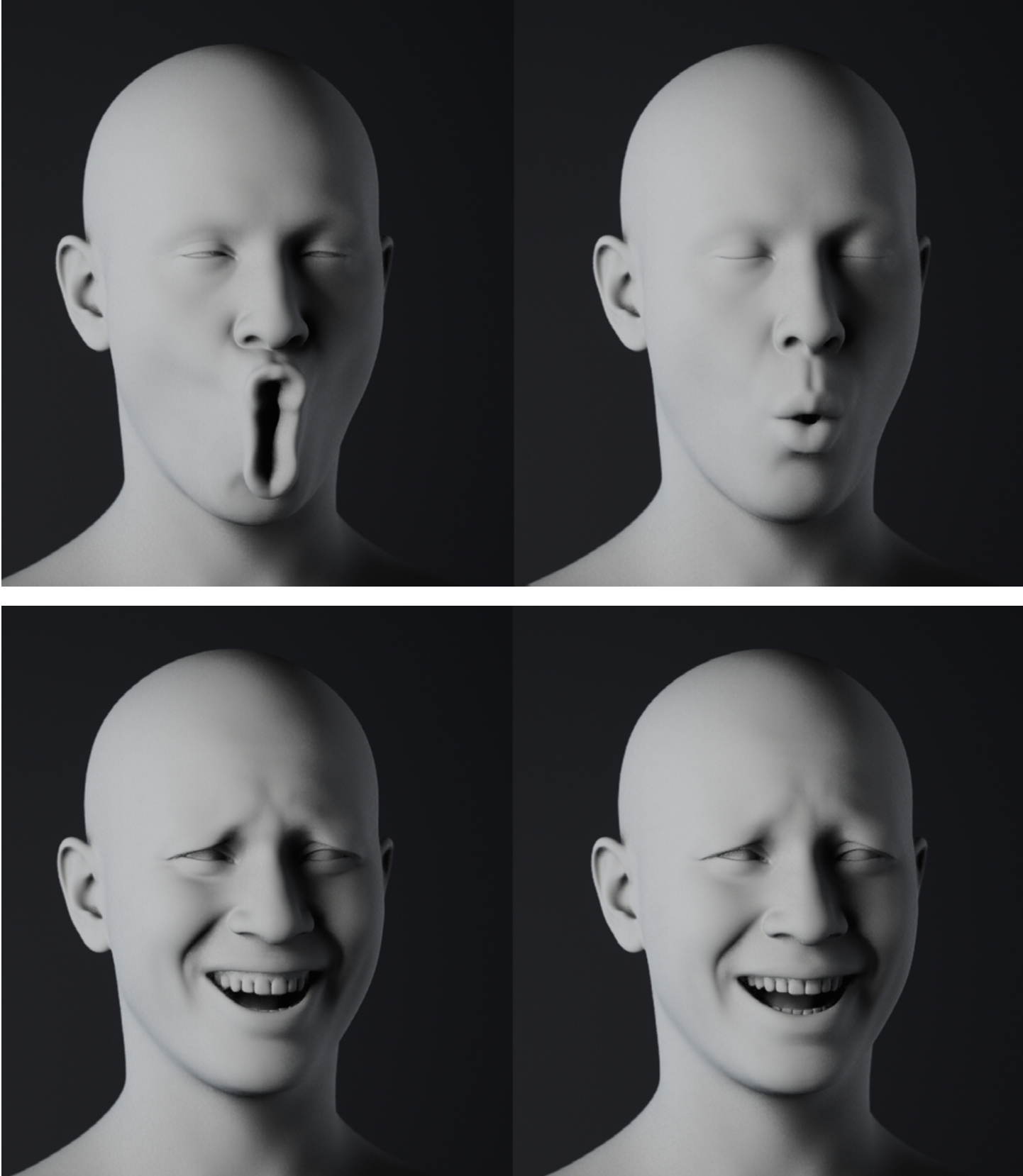


Figure 15: Corrective expressions applied (right) to correct "multiplied" deformations (left).

Runtime strategy

Joints

In Rig Logic, joints are separated into joint groups through a novel pruning strategy. During the preparation of a MetaHuman DNA file, the data that represents joint transformations is pruned by getting rid of undesirable data that would otherwise adversely affect the results (although its damage would be insignificant). This has a positive effect of reducing the amount of data that needs to be stored and also reduces the amount of computation that will have to be performed at runtime.

The outline of the pruning process is as follows:

- The face is separated into a number of logical regions (forehead, jaw, chin, etc.).
- Joints are separated into groups based on the facial regions they affect.
- A mapping is established between each expression and the joint groups affected by the expression.
- A constraint is defined to enforce that no joint groups are allowed to contain data that would affect regions they are not expected to control.

This essentially means that when a joint group is assembled to be exported into a MetaHuman DNA file, it will contain only the data of those joints that belong to the region of the face to which the joint group is assigned, and the data will also be filtered to contain transformations of only the specific expressions that may affect the joint group being processed.

The aforementioned LOD mechanism for joints is actually defined over each joint group individually, per each LOD.



Figure 16: Facial regions defined by color

Blend shapes

Blend shape channel weights computed by Rig Logic represent the intensity by which blend shape target deltas need to be activated. Blend shape target deltas are not produced by Rig Logic themselves—instead, they are stored in the Geometry section of MetaHuman DNA files. The computational process is a simple remapping of the input control weights that Rig Logic receives.

Just as with joints, the data to compute blend shape channel weights is also ordered by LODs. Thus, a simple slicing operation is enough to limit computation for the selected LOD.

Animated maps

The outputs produced by animated maps are multipliers, which are directly fed into shaders that control wrinkles, effects of blood flow under the skin, etc.

Following the same principles as the other solutions, animated maps are ordered by LODs and enjoy the same benefit of zero-cost LOD switching.

Performance

Through the process of separating joints into joint groups, it was possible to utilize dense submatrices instead of a sparse matrix for joint storage, which proved to be beneficial in implementing a very efficient algorithm for computing joint transformations. We used a custom matrix layout that partitions data into small blocks in a cache- and **SIMD**-friendly manner, minimally padded such that no scalar remainder loop is necessary. Thus, the whole code path is fully vectorized (while the padding memory overhead is less than 2%).

When compared to earlier generations of Rig Logic that used a simple CSR matrix for the same operation, this new version achieved a ~6x improvement in evaluation time. At the same time, the number of joints being evaluated was significantly increased. The resulting code achieves a near-ideal CPU pipeline utilization.



Conclusion

Rig Logic is a vast improvement over earlier real-time facial animation solutions. With its many efficiencies, portability, and other features, Rig Logic has proven to be a workable solution for MetaHuman Creator and other implementations.

In the future, we plan to improve Rig Logic even further by continuing to develop the technology, which will accompany the project needs of Epic Games. We will also look to emerging technologies to inform development and enhance future versions of Rig Logic.

About this document

Technology development

3Lateral

Author

Andreas Franc

Contributor


Pavel Kovac

Editor

Michele Bousquet

Graphics and layout

Jung Kwak



Note: Rig Logic is incorporated into the following products: MetaHuman Creator, MetaHuman character assets, and Unreal Engine; its use is covered under the applicable licenses for these products.