Metahuman AU Mapping Procedure:

By Zoe Lynch

To integrate the AU mapping script using Unreal Engine and MetaHumans, follow the detailed steps below. This process encompasses downloading and setting up necessary software, preparing the scene, executing the Python script for AU mapping, and finally processing the output.

Download and Initial Setup

Download the Custom Python Script: Access the provided URL here to download our custom Python script for AU (Action Unit) mapping.

Install Unreal Engine and UE Quixel Bridge: Download and install Unreal Engine along with the Quixel Bridge plugin. These tools are essential for creating and managing realistic scenes and assets.

Preparing the Scene

Set Up a Typical Scene: Within Unreal Engine, configure a standard scene by setting up the environment, lighting, and camera angle to suit your project's needs.

Import and Place a MetaHuman: Use the Quixel Bridge to import a MetaHuman asset into your scene. Ensure the MetaHuman's face is well-positioned in front of the camera, adjusting orientation and position according to your facial display preferences.

Assign a Built-in Control Rig: Attach the built-in control rig to your MetaHuman for detailed facial animation control.

Sequence Configuration: Assign the sequencer to the camera and add the MetaHuman to the Sequencer for animation.

Running the AU Mapping Script

Execute the Python Script: In Unreal Engine, navigate to `File > Execute Python Script` and select the previously downloaded custom Python script. This script imports necessary modules, defines functions for initializing control rig parameters, and assigns AU values for facial animation.

Sync Controls in Sequencer: Post script execution, the desired MetaHuman controls will now be synchronized within the sequencer, ready for animation based on the inputted AU data.

Output and Finalization

Animate and Save Frames: Run the sequence within the Unreal Engine to animate the MetaHuman according to the specified AU values. Each frame should sequentially represent the corresponding inputted AU data.

Export Frames as PNG Files: Save the animated frames as PNG files for further analysis or use in your research.

Script Description

The Python script serves multiple functions crucial for automating facial animation in Unreal Engine:

- **Module Importation**: It begins by importing necessary modules for Unreal Engine operations, CSV file handling, and OS operations.
- **Initialization and Resetting**: Defines a function to reset or initialize control rig parameters for each frame to default or specified values.
- **AU Value Assignment**: Includes a function to assign specific AU values to the MetaHuman's control rig, animating the facial expressions based on data read from CSV files.
- **CSV AU Value Processing**: Contains a function to read AU values from specified CSV files and populate a dictionary with these values for animation purposes.
- **Animation Logic**: Demonstrates conditional animation logic, applying different facial expressions over sequential frames.

AU Prediction code of custom emotions beyond standardized emotions:

https://colab.research.google.com/drive/1-dkLKZPZkT7ZeCYThNFFEIS9EYLcKmcN?usp=sharing

The process described in the provided code involves the utilization of OpenFace for AU facial feature extraction and analysis of non standardized emotions used in our study (Disappointed, Gratitude, Relief, and Resigned) . Here's a structured approach detailing the functionality and procedure:

Introduction to OpenFace in Colab

- The code begins by introducing the purpose of the notebook, which is to demonstrate how to use OpenFace for facial feature extraction from images of a specific emotion.
- OpenFace is cloned from its GitHub repository to ensure the latest version is utilized.

Installation and Configuration

- Essential dependencies, including CMake, GCC, and Python packages, are installed to prepare the environment for OpenFace.
- OpenFace is compiled and built from the source within the Colab environment, which involves downloading models and executing installation scripts.

Processing Images with OpenFace

- Directories for storing frames and processed images (we created 4 datasets with 10 different facial expressions from the internet associated with the target emotions) are created and managed using shell commands within the notebook.
- OpenFace's `FaceLandmarkImg` tool is used to process images stored in a specified directory, generating output that includes facial landmarks, pose estimation, and Action Unit (AU) intensities.

Displaying Processed Images

- Processed images, possibly with facial landmarks and AU intensity annotations, are displayed directly within the Colab notebook using IPython display functionalities.

Docker Usage

- The notebook demonstrates how to use Docker to run OpenFace, showing commands to install Docker, run the OpenFace container, and execute OpenFace commands within the container.

Dlib Installation and Compilation

- Instructions for cloning, building, and installing dlib are provided. Dlib is a toolkit containing machine learning algorithms and tools for developing complex software in C++ to solve real-world problems.

Data Aggregation and Analysis

- The script includes steps for aggregating facial feature data from processed images, displaying the overlay of facial landmarks and AU intensities, and analyzing AU data for different emotions or facial expressions.
- Custom functions are defined to facilitate the overlay of analysis data on images, conversion of image sequences to video, and detailed analysis of AU intensities across different frames or emotions.

Advanced Analysis and Visualization

- Pandas and Matplotlib are used for data handling and visualization, including calculating the percentage of appearance for each AU, identifying highly correlated AUs, and aggregating AU data from multiple subjects. It also returns the common AUs throughout each face which can be used to generate an expression corresponding to the target emotion in the metahuman.

Resources:

Unreal Engine

- **Unreal Engine Documentation: The official documentation for Unreal Engine offers comprehensive guides and tutorials.**
    - **Link: Unreal Engine Documentation**
    - **https://docs.unrealengine.com/**
- **Unreal Engine Forums: A community forum where users can ask questions, share experiences, and find answers to common issues.**
    - **Link: Unreal Engine Forums**
    - **https://forums.unrealengine.com/**
- **YouTube - Unreal Engine: The official YouTube channel provides video tutorials, developer interviews, and project showcases.**
    - **Link: Unreal Engine on YouTube**
    - **https://www.youtube.com/user/UnrealDevelopmentKit**

Quixel Bridge

- **Quixel Tutorials: Quixel offers tutorials on how to use Bridge, including integration with Unreal Engine.**
    - **Link: Quixel Tutorials**
    - **https://quixel.com/tutorials**
- **Quixel Documentation: The official documentation provides a deep dive into using Quixel Bridge effectively.**
    - **Link: Quixel Documentation**
    - **https://help.quixel.com/hc/en-us**

OpenFace

- **OpenFace GitHub Repository: The GitHub page for OpenFace includes installation instructions, usage examples, and code documentation.**
    - **Link: OpenFace on GitHub**
    - **https://github.com/TadasBaltrusaitis/OpenFace**
- **OpenFace Wiki: The wiki associated with the GitHub repository offers additional insights into the tool's capabilities and applications.**
    - **Link: OpenFace Wiki**

- **https://github.com/TadasBaltrusaitis/OpenFace/wiki**