Max Patch: *Hello World!*

-for Facial Movements and Max/MSP

YouTube Link: https://youtu.be/be65zQkX1t4?si=zVCl6eHdJckuG-1d

Hello World! is a Max/MSP piece of improvised interactive electronic music that responds to facial movements. Named after the classic "Hello, World!" program in computer programming, this Max/MSP piece explores interactive music using facial movements. By leveraging FaceOSC (Fig. 1), my facial movements, including positions of features and jaw, are converted into digital signals, which are then transmitted to Max for real-time interaction (Fig. 2).

The interactive music piece uses a performance filled with tension and gradually building emotion to mimic and express a newborn's first arrival to the world.

Fig. 1

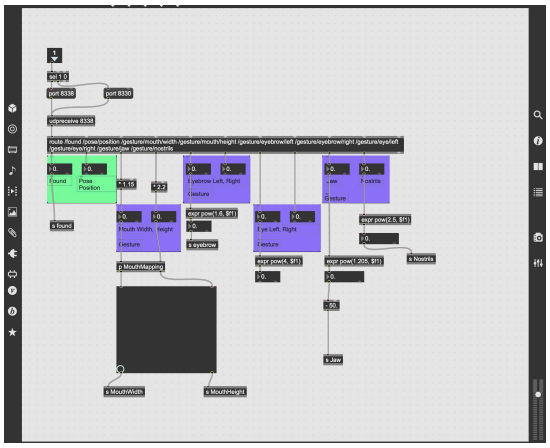


Fig. 2

1. Technology Documentation:

I designed two types of sounds in Max: a looped sound and a background drone sound (Fig. 3). These sounds are processed mainly using the “groove~” sample playback object, which manipulates loops of the audio samples stored in “buffer~” objects. The sounds are controlled by signals from FaceOSC, which adjust the start and end times of the audio loops processed by the “groove~” object.

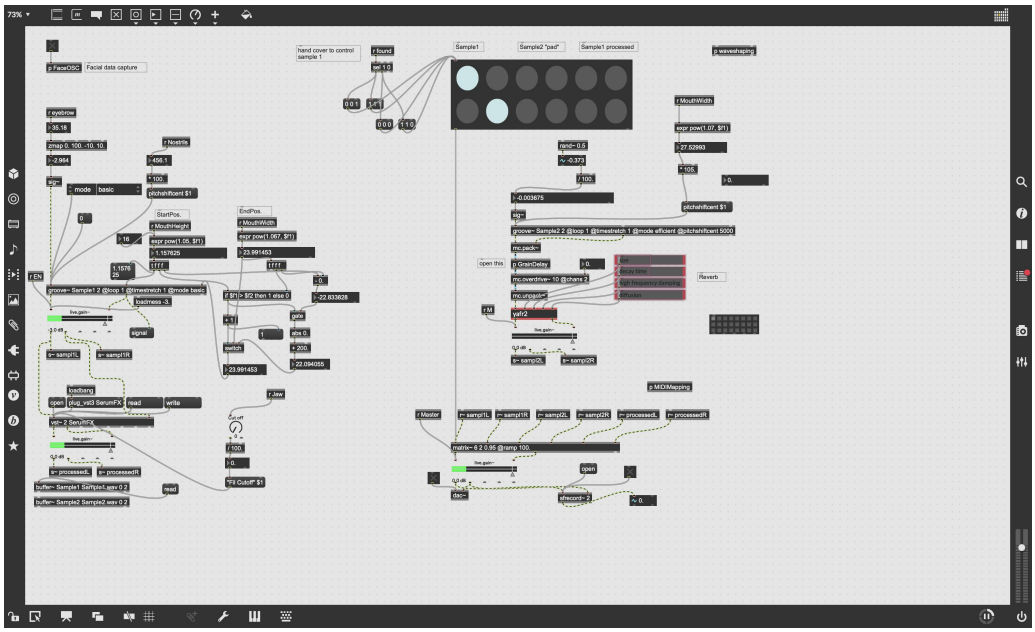


Fig. 3

1.1 Looped Sound:

As shown in the left part of Fig. 3, I designed an algorithmic system to create tension within the looped sound. First, FaceOSC generates signals based on my mouth height and width, which are projected onto a “pictslider” object to make sure that the numerical signals are not too large. The signals are limited to 0-127 (Fig. 2).

Next, I connected the mouth height and width signals to an “expr” object to perform an exponential function calculation. The base values of the functions are both greater than 1, while the exponents are dynamic values generated by the mouth’s

height and width. Smaller values change less, whereas larger values change more, thus enhancing the tension effect in the real-time interactive music.

Then, I connected the mouth height and width signals to the Loop Min and Loop Max inputs of the “groove~” object respectively to control the looped sound's start and end times in real time. Since the mouth width is easier to maintain, it is connected to Loop Max, with the base for the exponential function set to 1.067. This results in a maximum output of approximately 3774, meaning that the 4-second audio stored in the “buffer” can have a loop endpoint up to around 3.7 seconds. For the mouth height (connected to Loop Min), I needed to reduce the time value’s dynamic range, so I set the base of its exponential function to 1.05, making the maximum output approximately 490, which corresponds to starting the loop at most 0.49 seconds.

To prevent audio discontinuities when the Loop Min value (controlled by mouth height) exceeds Loop Max (controlled by mouth width), I designed an algorithmic system centered around the “if”, “gate”, and “switch” objects to analyze and process the values produced by the mouth width, thereby ensuring the end time not to exceed the start time in the looped audio:

“if” object: Compares values. When the height value exceeds the width value, it outputs a signal of 1; Otherwise, it outputs 0.

“gate” object: Receives the signal from the “if” object. The gate only outputs a value when the “if” object outputs a signal of 1.

“switch” object: Receives the signal from the if object after adding 1 to it, resulting in a value of 1 or 2. When the received value is 1, it outputs the mouth width value after exponential processing. When the received value is 2, it outputs the absolute difference between width and height.

As shown in Fig. 4, the value of the mouth height after exponential calculation always determines the start time of the looped audio.

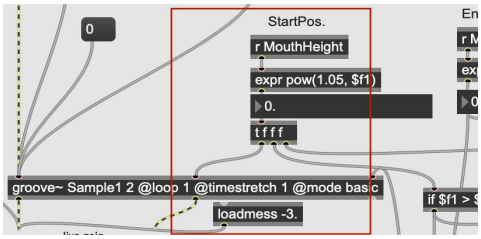


Fig. 4

In Fig. 5, when the mouth is smiling (Width > Height), the “if” object outputs a signal of 0. The “gate” object does not output, and the “switch” object receives a signal of 1, directly outputting the mouth width value after exponential processing. This is sent to Loop Max of the “groove” object, defining the end time of the looped audio.

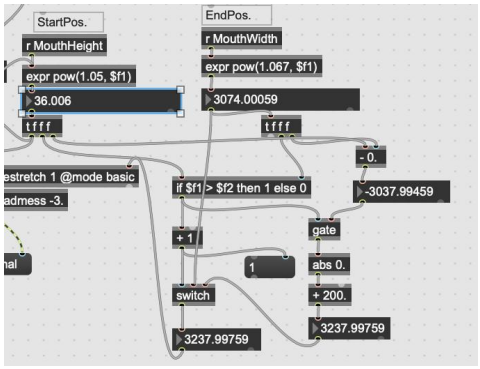


Fig. 5

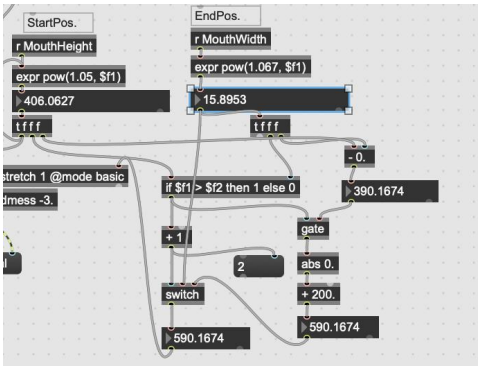


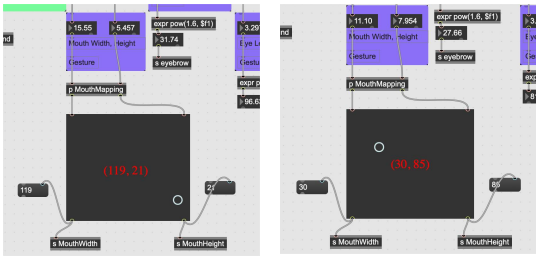
Fig. 6

When the width value is less than the height value (Fig. 6), the sound effect is designed so that the audio stops when the mouth just starts to form an oval shape (i.e., when Height slightly exceeds Width, causing Loop Min to exceed Loop Max). As the mouth continues to tighten (Height increasingly exceeds Width), the looped audio speed becomes faster and more intense. Here is how I designed this behavior:

As shown in Fig. 6, when the mouth continues to tighten after forming an oval shape (Width < Height), to ensure Loop Max > Loop Min, the “if” object outputs 1,

allowing the “switch” object to receive a signal from the “gate”. The signal output from the “gate” is processed as |Width - Height| and then added to a constant *a*, ensuring that Loop Max is always greater than Loop Min. Therefore, I need to determine the value of this constant *a* to ensure continuous looping of the audio. Below is the method to determine constant *a*:

The numerical values of mouth height and width in the “pictslider” object (Fig. 2) generally approximate a decreasing linear function. By capturing the values of height and width at two different moments, assume Width = *y* and Height = *x*. As shown in Fig. 7, the relationship between *y* and *x* is approximately *y* = -89/64 * *x* + 9485/64. Then, after exponential function processing, assuming Height output is *x′* and Width output is *y′*, we have *x′* = 1.067^*x* and *y′* = 1.05^*y*, which translates to 1.05^(-89/64 * *x* + 9485/64).



$$\begin{cases} 119 = 21\,a + b \\ 30 = 85\,a + b \end{cases}$$

Fig. 7

To ensure Loop Max is greater than Loop Min, we set the equation as: |*x′* - *y′*| + a > *x′*, which expands to:

$$1.067^{x} - 1.05^{-89/64*x+9485/64} + a > 1.067^{x}$$

Thus:

$$a > 1.05^{-89/64*x+9485/64}$$

Given that *x′* > *y′*, we can infer that:

$$x \in \left(-\infty;\ \frac{64\ln\left(\frac{1}{2^{\frac{9485}{32}}5^{\frac{9485}{64}}}\right)+9485\ln(21)}{89\ln(21)-89\ln(20)-64\ln(10)}\right)$$

Since x is greater than 0, the range of x is approximately between 0-59.49, as Width output y = 1.05^{-89/64 * x + 9485/64} decreases with increasing x, a must always be greater than the maximum value of y. The maximum value occurs when x = 0, with a's minimum value approximately equal to 1381, where Loop Max could always be greater than Loop Min. However, to design the condition where audio stops when the mouth starts to form an oval shape (Fig. 8), we also need to determine the minimum value of a that allows Loop Min to exceed Loop Max. In this scenario, x reaches its maximum in the range, yielding $a \approx 24.40$. Therefore, for Width < Height, the constant a must be between 24.40 and 1381 to ensure that the audio stops when the mouth just starts to form an oval shape and the looped audio speed becomes faster and more intense as the mouth continues to tighten. To make it easier for the mouth to reach a position where the audio stops, I set a = 200.

Fig. 8

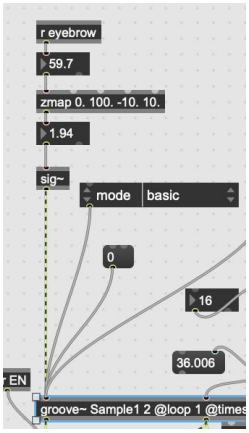


Fig. 9 **Fig. 10**

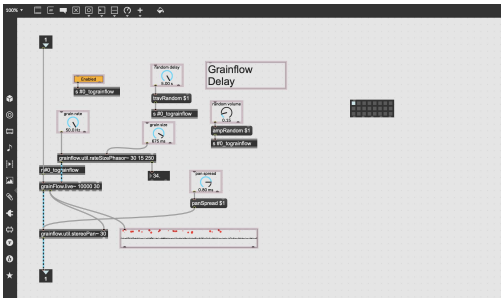
Next, I connected the groove~ object to a sig~ object controlled by the eyebrow (Fig. 9), allowing slight adjustments to playback speed. Finally, I connected the entire groove~ object to Serum FX (Fig. 10) to modify the timbre. With this setup, a smiling mouth results in slow loop playback, and as the mouth tightens, the speed increases until it forms an oval shape, at which point the audio stops. If the mouth continues to tighten, the audio speed increases further, and the tension of the interaction grows, creating a dynamic and interactive looped sound system.

1.2 Background Drone Sound:

As shown in the right part of Fig. 3, the background drone sound is also generated using the “groove~” object, which loops audio stored in the “buffer~” object. To create noticeable variations in the background drone sound, I connected the “groove~” object to a “pitchshiftcent” object controlled by my mouth width. When I smile, the mouth width increases, causing an increase in pitch shift and raising the pitch of the sound. Conversely, as the mouth tightens, the pitch drops, resulting in a lower-frequency sound. This allows the mouth width to effectively control the pitch of the sound. I then connected the “groove~” object to a grin delay sub-patch (Fig. 11) and a

"yafr2" reverb object and adjusted the parameters of "yafr2", which could add granular texture and spatial depth to the sound.

This system results in a dynamic background drone sound, with the pitch rising as the mouth opens into a smile and lowering when the mouth tightens. The reverb adds atmosphere, making the drone sound affluent and immersive.



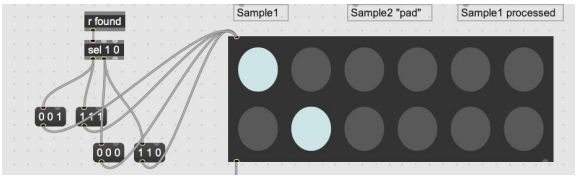


Fig. 11 **Fig. 12**

Finally, I routed the background drone sound, the processed looped sound (after Serum FX), and the unprocessed looped sound to a "matrix~" object. The "matrixctrl" was then connected to a "sel" object that could detect whether FaceOSC was sending a signal (Fig. 12). This allowed for further interactive control—when I covered my face with my hands, FaceOSC would stop transmitting signals, resulting in the "sel" object controlling the unprocessed looped sound output. This enabled additional interaction using hand movements, enhancing the musical expressivity by adding another layer of performance control.

This piece utilizes the "sfrecord" object for internal recording, and I connected an AKAI Midimix controller to manage the looped sound and the background drone sound.

2. Music Documentation:

This piece begins with a representation of a newborn's first arrival to the world.

During the first 20 seconds, my head is lowered, preventing FaceOSC from detecting my face. Due to the modulation by the "sel" object, the unprocessed looped sound remains silent, while the looped sound (after Serum FX processing) is set to loop an extremely short clip (0.001 seconds), creating a high-frequency, tension-filled sound (Fig. 13). This sound gradually comes out in using the Midimix, symbolizing the newborn's imminent arrival into the world.

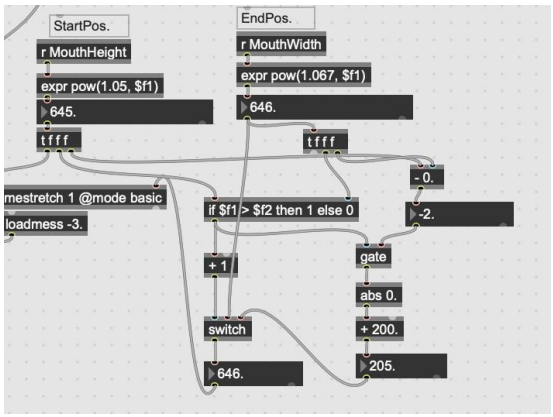


Fig. 13

After 20 seconds, I raise my head to let FaceOSC detect my face, initiating interactive control. My facial expressions and hand movements alternate, representing the newborn's curiosity about the new world. The music swells and relaxes accordingly, embodying moments of curiosity interspersed with caution.

After a short transition, around 1 minute and 23 seconds, the background drone sound gradually enters, symbolizing a sense of danger. The newborn, represented by my facial gestures, begins to exhibit fearful expressions, and the movements become more rapid, increasing the tension in the looped sound. The music grows more intense, reaching a peak at 2 minutes and 14 seconds, where the music stops, transitioning into the final section.

After 2 minutes and 24 seconds, my head repeatedly lowers and raises, symbolizing a cautious release from fear. The tentative lowering of the head represents gradually letting go of caution, and eventually, continuous interaction between my face and Max signifies the danger has passed. The newborn becomes satisfied, and the music ends with several intermittent looped sounds, symbolizing the newborn's renewed curiosity and longing for the world.

3. Summary:

Overall, *Hello World!*, this piece of improvised interactive music embodies the experience of a newborn exploring the world for the first time. Through expressive facial movements and interactions with Max/MSP, it conveys the newborn's curiosity, fear of encountering danger, and the renewed longing for the world once the danger has passed. The performance strikes a balance between tension and release, designed to convey these three states in a flexible and dynamic way.