FACEJAM.APP: JAVASCRIPT FACIAL EXPRESSIONS FOR MUSICAL EXPRESSION

Christopher J. Tralie
Ursinus College Math And Computer Science
ctralie@alumni.princeton.edu

Parker Fairchild
Ursinus College Computer Science
pdfairchild8@gmail.com

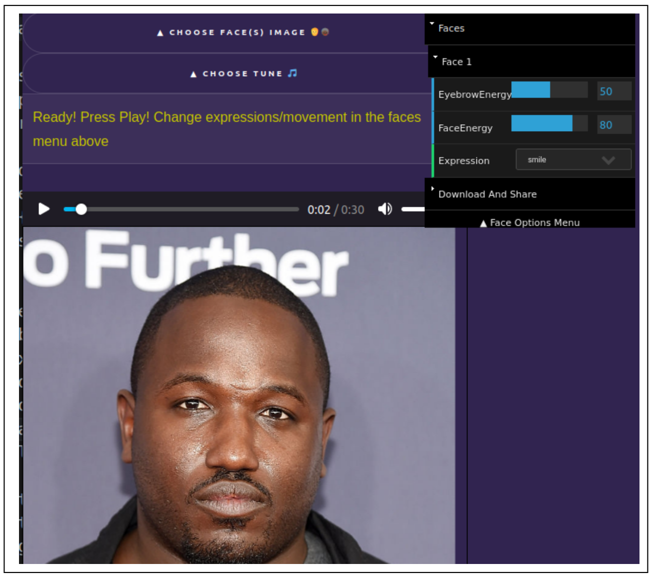
ABSTRACT

We present www.facejam.app: a system that combines computer vision, computer graphics, and MIR to automatically animate facial expressions to music. This work started off as an offline Python script that won "best code" at the HAMR Hackathon at Deezer in 2018, and we have extended it to work live in the browser using Javascript and WebGL. The system automatically detects facial landmarks in an image using face-api.js [1], and it uses dynamic programming beat tracking to move detected eyebrows up and down to the beat, while also mapping instantaneous power to a "smile" expression. These audio aspects drive a model face, which is used to warp an arbitrary face real time using piecewise affine warps via GLSL shaders. The system supports animating multiple faces in an image, and audio can be sourced from uploaded files, player recordings from a microphone, or 30 second preview clips from the Apple Music. Players can also save their favorite results automatically to files from the browser to share on social media. Our fully client side prototype system is currently live at https://www.facejam.app.

1. INTRODUCTION

The use of facial expressions to visualize data dates back at least to Chernoff faces in the '70s [2]. In our modern rendition of face expression-based data visualization, we specifically use facial expressions to visualize rhythm and energy features over time in musical audio, which we synchronize live to the audio. This makes for a fun music visualizer in which faces come alive and move in sync with the music. This whimsical application fits in well with current social media trends with face filters.

In what follows, we explain a basic prototype which is similar to the one we developed at HAMR in 2018, and then we follow with some improvements we added in our new web-based version.



Ready! Press Play! Change expressions/movement in the faces menu above

Figure 1. A screenshot of www.facejam.app

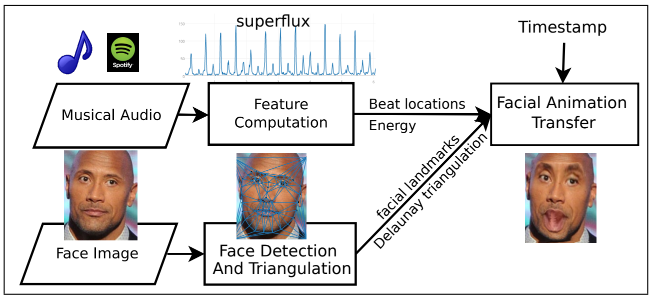


Figure 2. A block diagram of the system.

2. TECHNICAL DETAILS OF THE PIPELINE

Figure 2 shows an overview of our system, which combines computer vision techniques and MIR techniques, as well as asynchronous programming in Javascript, to achieve our goal. Our system is implemented fully on the client side for simplicity and sustainability.

2.1 Audio Features

On the audio side, we compute the instantaneous signal energy within a sliding over time, which is later mapped to a smile expression; that is, faces smile more when there's more going on in the audio. We also use beat onset times to separately animate the eyebrows, moving the eyebrows up closer to an onset. To this end, we use the offline Viterbi

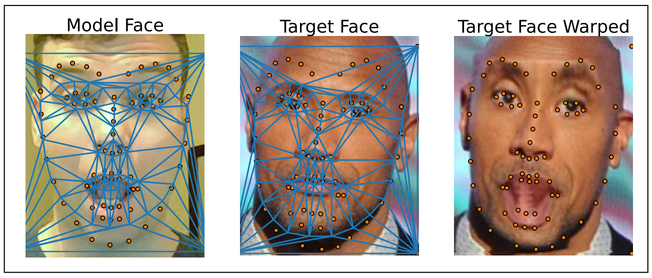


Figure 3. A model face is animated based on music, and a target face in a user specified image is warped accordingly. This is done via a Deluanay triangulation of facial landmarks, which is used to transfer facial landmark motion from a model face to a target face, whereby we setup a piecewise affine warp of the target face to change its expression. Note how the landmarks on the target face are positioned relative to the same triangles as the corresponding landmarks in the model, which is how we achieve "shape analogies" from a neutral face to a different expression between the model and the target.

beat tracking algorithm of Ellis [3] on top of a superflux audio novelty function [4]. This technique requires a tempo bias as input, which we compute using the combined ACF-DFT technique of Peeters [5]. We use the results of this beat tracking to move the eyebrows of a target face up and down.

2.2 Computer Vision Techniques

On the computer vision side, to enable such facial animation, we rely on facial landmark detection software to localize finer geometry of user specified face images [1] (as seen in Figure 3). As a preprocessing step outside of the system, we model different expressions, such as the smile that is currently in our prototype, as a sequence of these landmarks on a model face. Since the landmarks are noisy, we use median filtering and Procrustes alignment [6] to clean up these landmarks before saving them to our model.

Next, we need to to transfer a changing expression on the model (including both beat-based eyebrow movement and energy-based partial expressions) to a face detected in a user supplied image [1]. We compare the moving facial landmarks in the model with respect to its neutral expression, and we perform point location inside of a Delaunay triangulation of the facial landmarks. We then express the landmarks using barycentric coordinates, which are relative to the locations of the triangle. To move the landmarks in the target face in the same relative direction, we apply these same barycentric coordinates to the triangles in the target face using the facial landmark coordinates of the target. Finally, now that we have new landmark locations describing the motion of the target face, we perform a triangle by triangle piecewise affine warp to the pixels in the target face to warp the entire face. Figure 3 shows an example.

[1] See [7] for another simple approach to this.

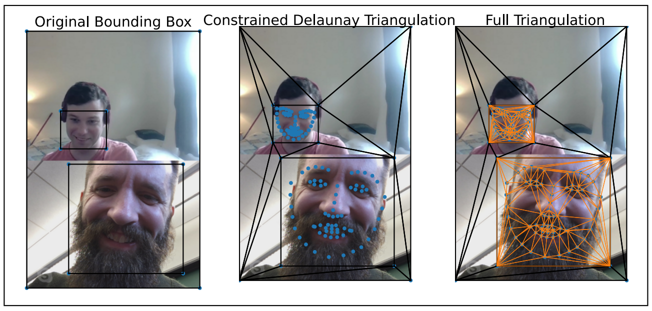


Figure 4. A triangulation is created for multiple faces using a constrained Delaunay triangulation to connect the bounding boxes between the faces

3. IMPROVEMENTS FOR THE WEB

The initial prototype for this system was developed for the Hacking for Audio And Music Research (HAMR) Hackathon in 2018. We have completely redone the pipeline from scratch in Javascript and made numerous improvements along the way.

Most noticeably, barycentric warps were implemented in software before, which caused the rendering stage to be a 3-4x factor slower than real time. In our new system, we use a WebGL shader to perform the warp on the GPU, which is far faster than real time. This means that it is now possible to interactively tune parameters, such as “eyebrow energy,” or the vertical extent of eyebrow movement as a function of the beat.

The next bottleneck after the image warping is the triangle point location in pure Javascript. We accelerate this by assuming that the landmarks stay within the adjacent triangles, and we search those first before reverting to a brute force search through all triangles [2].

We also now support multiple faces which are animated in sync (Figure 4), and we allow flexible inputs for both images and audio. On a mobile device, a person can easily take their picture and load it directly into the app.

Furthermore, we now support audio recording, and we have a built in Apple Music searcher + 30 second song preview integration to make it easy to choose from a huge variety of clips. Fortuitously, we find that this 30 second length is about right to convey a particular tune + face concept.

To move towards settings in which this may be used at a concert venue for people to synchronize their phones to live audio, we provide a real time option for beat tracking by implementing a Bayes filter version (as opposed to an offline Viterbi method) of the efficient bar pointer model of [8] (original full state space devised in [9]). As in [10], we use a state space model restricted to a single beat interval.

Finally, to make it easy to share fun results on social media, we allow the user to save results to a video. This is performed client side in the browser using an Emscripten port of ffmpeg [11].

[2] A further acceleration could be to to a breadth-first search through a half-edge data structure representing the triangulation if there are no hits in this neighborhood.

4. REFERENCES

[1] V. Mühler, “face-api.js,” https://justadudewhohacks.github.io/face-api.js/docs/index.html, accessed 2019-09-09.

[2] H. Chernoff, “The use of faces to represent points in k-dimensional space graphically,” *Journal of the American statistical Association*, vol. 68, no. 342, pp. 361–368, 1973.

[3] D. P. Ellis, “Beat tracking by dynamic programming,” *Journal of New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.

[4] S. Böck and G. Widmer, “Maximum filter vibrato suppression for onset detection,” in *Proc. of the 16th Int. Conf. on Digital Audio Effects (DAFx). Maynooth, Ireland*, vol. 7, 2013.

[5] G. Peeters, “Template-based estimation of time-varying tempo,” *EURASIP Journal on Advances in Signal Processing*, vol. 2007, pp. 1–14, 2006.

[6] P. H. Schönemann, “A generalized solution of the orthogonal procrustes problem,” *Psychometrika*, vol. 31, no. 1, pp. 1–10, 1966.

[7] J.-y. Noh and U. Neumann, “Expression cloning,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001, pp. 277–288.

[8] F. Krebs, S. Böck, and G. Widmer, “An efficient state-space model for joint tempo and meter tracking.” in *Proc. of the 16th Int. Conf. on Music Information Retrieval (ISMIR). Malaga, Spain*, 2015, pp. 72–78.

[9] N. Whiteley, A. T. Cemgil, and S. J. Godsill, “Bayesian modelling of temporal structure in musical audio.” in *Proc. of the 5th Int. Conf. on Music Information Retrieval (ISMIR). Victoria, Canada*. Citeseer, 2006, pp. 29–34.

[10] M. Heydari and Z. Duan, “Don’t look back: An online beat tracking method using rnn and enhanced particle filtering,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 236–240.

[11] K. Hiiragi, “ffmpeg.js,” https://github.com/Kagami/ffmpeg.js/, accessed: 2021-08-27.