FESINC: Facial Expression Sculpturing with INterval Constraints

Zsófia Ruttkay
Centrum for Mathematics and Computer Science
POBox 94079
1090 GB Amsterdam, The Netherlands
Zsofia.Ruttkay@cwi.nl

Han Noot
Centrum for Mathematics and Computer Science
POBox 94079
1090 GB Amsterdam, The Netherlands
Han.Noot@cwi.nl

ABSTRACT

The animation of synthetic faces is still a low-level process requiring much human expertise and hw/sw resources. The constraint-based facial animation editor system, FESINC, provides two kinds of support: allows the a-priory, declarative definition of dynamical expressions and requirements, and assures that while making the animation, these requirements are fulfilled. The novelty of the approach is that it makes possible the intensional definition and manipulation of (facial) animations.

Keywords

Facial animation, constraints, interval propagation.

1. INTRODUCTION

Facial expressions are characteristic and effective ways of communication between humans. In man-to-man communication, a great variety of non-verbal facial signals are produced, which fulfill different functions [9]. Mouth shapes increase the eligibility of spoken text. Emotional and cognitive expressions help to understand the state of a situation and the speaker in a concise way. Other signals punctuate and structure spoken text or indicate state of discourse. There are facial signals - some of them with biological functions - which indicate the idle but alive/ awake state of a person. Last but not least, to a great extent we (unconsciously) identify individuals and enjoy their diversity on the basis of the look and dynamism of their faces.

During the 25 years history of computer facial animation, different models and deformation techniques have been proposed, see [8] for an overview. The importance of facial expressions is reflected in recent research on and applications of avatars [2, 6]. Avatars are human-like synthetic representatives of a complex computer systems or of a human in interactive or multi-user applications. Preceded by extensive study of analysis and coding of facial emotional expressions, we witness the first commercial applications of synthetic characters with affective talking heads, and software packages to design and animate synthetic faces [4,5].

Analyzing carefully the at first sight impressive applications, however, some common shortcomings can be identified:

- The facial expressions of a specific type are identical, due to the fact that single (tracked or synthetic) expressions are wired-in the system.
- The blending and concatenation of expressions is based on simple principles, often resulting in unnatural facial movement. While for visual speech the co-articulation and concatenation has been studies extensively, little attention has been paid on developing principles and methods to superimpose and blend facial expressions in time.
- The production of a subtle facial animation of a synthetic character is a tedious, low-level process which requires much professional skill and time, as there is neither enough available knowledge on the dynamism of human facial expressions, nor appropriate paradigms and tools to animate synthetic faces.
- The ‘look’ of a synthetic head is often non-photorealistic, but the expressions are realistic. That is, no facial animation framework is available to exploit the possibilities of the traditional animation, which uses exaggeration, special effects non-existing on realistic faces and some of the general ‘Disney-rules of animation’. Such ‘beyond-realism’ animation effects could increase the effectiveness and appeal of the non-verbal communication repertoire of a character.

We have designed and implemented a facial animation editing system which overcomes several of the above shortcomings. The system allows the animator to define and re-use animation building blocks, as well as express general (e.g. all the pair of features move symmetrically, except eyebrows) or time-related (e.g. synchronization) requirements the animation has to fulfill, all expressed in the form of constraints.

In Chapter 2 we introduce the concept of ‘expression sculpturing’. In Chapter 3 the issues of constraint processing are addressed. We explain how an interval constraint propagation mechanism is used in an interactive graphical editor setting. In Chapter 4, the currently implemented system is discussed. In the closing chapter, issues of further research are addressed.

2. THE DECLARATIVE ROLE OF CONSTRAINTS

2.1 Sculpturing dynamical facial expressions

We introduce the idea by an example. Let's assume that there is a perfect physically-based facial model [8] at our disposal. In order to simplify our discussion, we restrict the task to defining the contraction of the most important muscle pair involved (in the real face, and thus in the facial model as well), the Zygomatic major muscles, pulling up diagonally the corners of the mouth. We will use the common, but unrealistic [3] further simplifying hypothesis, namely that the muscle activation happens in three, linear stages: *application*, *release* and *relaxation* as given in Figure 1.

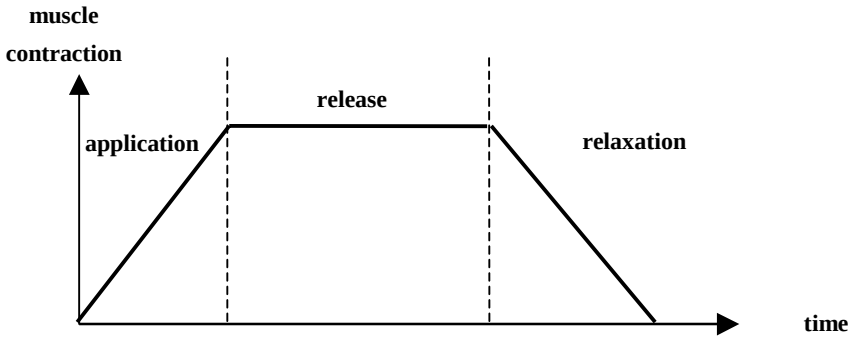

Figure 1. Stages of contraction of a muscle (based on simplifying assumption)

One can define infinitely many pairs of trapezoid-shaped muscle contraction functions — which ones produce an acceptable smile? How short or long a smile can be? How are the duration of application, release and relaxation related? What are the absolute and relative limits on the contraction at the start and end of the release? What is a typical generic smile like? In what ways and to what extent can a smile be specific? What is the total effect of co-existing expressions (e.g. smile and speech)?

The questions above, unfortunately, still cannot be answered by analyzing a huge sample of real smiles. In spite of the achievements of face tracking hw/sw, it is difficult to get enough real, spontaneous facial expression samples recorded under circumstances needed for analysis.

A tool which allows the animator to *declare* answers (albeit invented ones) for the above questions can be considered as a tool to sculpture facial motion. With such a tool, before starting to work on the detailed animation, the animator decides about the typical dynamical facial expressions of the character. When making a particular animation for the face in question, the animator inserts random or default examples of the character's smile. This he can modify either by directly adjusting parameter values, but only in such a way that the modification does not violate the declared characteristics of the facial expression, or by modifying some of the declared characteristics or adding new ones (e.g. a symmetrical smile can be made asymmetrical by lifting the general requirements of symmetry.)

When working on an animation, the system enforces that the animation meets the requirements, by re-adjusting certain parameters. In this way the animator's work becomes easier and faster, and is lifted to a higher, conceptual level.

The animator is supported in two (may be interwoven) stages of facial animation:

- to sculpture the dynamism and mimic repertoire of a face to be animated;
- to **make animations** for a face with a given mimic repertoire, meeting certain further requirements set for the particular animation.

2.2 Definition of facial expressions by constraints

We assume that the head to be animated can be *deformed* by specifying a fixed number N of facial deformation parameter values. Each *parameter* (e.g. muscle contraction, FAP,...) may take its value from a domain of a closed and finite interval of reals. One specific value of the domain is the *neutral value*, that is the value of the parameter in case of a neutral face. An animation is the evolution of the deformation along time. An *animation* is given by the vector of functions $\mathbf{(F_1(t),...F_N(t))}$, where $\mathbf{F_i: T\text{-}\text{-}>D_i}$ gives the value of the i-th parameter for each time moment in **T**, where **T** is the duration of the animation.

The parameter values are given explicitly for some time moments only, and for the rest of the time the value is computed on the basis of the given defining values. We use piece-wise *linear interpolation* on the intervals between the time moments with given parameter values, but other, smooth interpolation could be used too.

As introduced above, for the i-th parameter, a number of $\mathbf{P^i_j = (t^i_j, v^i_j)}$ *control points* (CPs) are given which define the parameter curve for the parameter. The number of control points may differ from parameter to parameter, and control points for different parameters need not be aligned along time. We assume that control points of one parameter are indexed according to increasing time, that is $\mathbf{t^i_j < t^i_{j+1}}$. If for a certain parameter no control points are given, then the value of the parameter is assumed to be the neutral value (which is the value of the parameter at neutral expression).

Usually the task of making/modifying a facial animation is given in terms of certain requirements. E.g. how long the animation should be, what expressions should the face show at certain time moments or intervals, blinks should be slow, etc. To specify an animation requires specifying a sufficient number of control points, at proper times with proper parameter values, namely so that the resulting $\mathbf{F_1(t),...F_N(t)}$ functions together produce an animation with the requested characteristics.

As opposed to the traditional computer animation tools where most (if not all) of the envisioned characteristics are only present in the head of the animator, our system allows to express these characteristics in terms of certain types of *constraints on co-ordinates of control points*. The animator can use a given set of types of constraints, the so-called basic types, to express desired characteristics. The constraints are listed and explained in Figure 2. Besides the listed types, the user may use any linear constraints. All the allowed types of constraints limit the value of a linear function of co-ordinates of certain control points. In order to get used to the notation and meaning of constraints, the reader is asked to define his/her own 'smile' with the help of constraints.

(Ia)	$\mathbf{t^i_j \in I^*}$	time range	(Ib)	$\mathbf{v^i_j \in I^*}$	value range
(IIa)	$\mathbf{t^i_j - t^n_m \in I}$	time duration	(IIb)	$\mathbf{v^i_j - v^n_m \in I^*}$	value change
(III)	$\mathbf{\frac{t^i_j - t^i_k}{t^i_r - t^i_s} \in I}$	relative time duration			
(IV)	$\mathbf{\frac{v^i_j - p}{v^n_m - q} \in I}$	relative parameter value, where $\mathbf{p}$ and $\mathbf{q}$ are reals (typically, neutral values), and either $\mathbf{I}$ does not contain 0 or is $\mathbf{v^i_j - q}$ always positive.			
(V)	$\mathbf{\frac{v^i_{j+1} - v^i_j}{t^i_{j+1} - t^i_j} \in I}$	parameter change speed			

Figure 2. The types of basic facial animation constraints. and stand for the time and value of the j-th CP in the i-th parameter. I denotes a finite or infinite, $\mathbf{I^*}$ a finite interval. Inequalities and equalities are expressed in the form of membership in an interval. E.g. x-y >= 20 would be expressed as x-y $\in$ [20, + ∞].

3. CONSTRAINT PROCESSING

3.1 Interaction between the user and the solver

Editing an animation takes place by a sequence of two kinds of primitive editing operations:. The user may change the time and value of a CP (or groups of CPs), and add/delete (groups of) CPs. These operations can be performed by directly manipulating a graphical representation of the control points of the parameter functions.

Interwoven with the manipulation of control points, the animator may also change (add/delete/modify) interactively the set of constraints which have to be satisfied. These operations may result in some violated constraints (e.g. he drags along time a CP, and a constraints prescribing that another parameter's certain CP should have the same time is then violated).

The basic engine behind the editor is the constraint solver, which assures that all the constraints get again satisfied. The solver achieves this by fulfilling two tasks:

- provides for the animator the **intervals for possible times/values** for each CP (and the editor then forces the animator to select only from these values)**;**
- **updates the animation** in such a way that the constraints remain satisfied.

Because of the first service, the solver can always accomplish the second task: as an answer for the animator modifying CPs, it can always adjust some part of the animation such that all the constraints are satisfied again. However, the user may add/modify a constraint in such a way that it contradicts the rest of the constraints, that is, the problem has no solution. If this is the case, the constraint addition/modification proposed by the animator is not accepted. Otherwise the animation is updated in such a way that the modified constraint gets also satisfied. The interplay of the animator and the constraint solver is illustrated in figure 3**.**

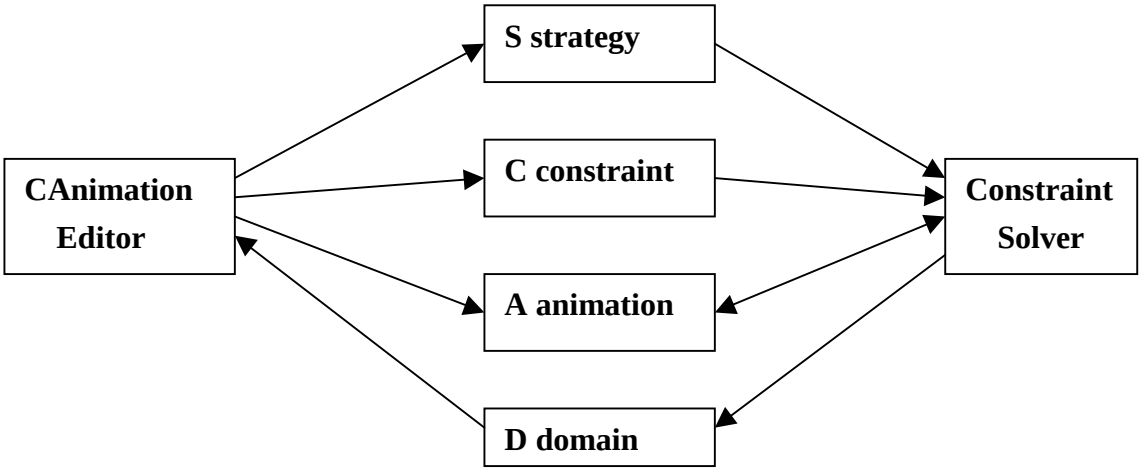

Figure 3. The interplay of animator and the constraint solver

3.2 Interval propagation

At each moment of editing an animation, the quadruple **<S, C, A, D>** is maintained, where:

C is the set of *constraints* to be fulfilled;

A is the *animation*, given in terms of value/time of CPs (some of these variables are constrained by elements of C);

D is the list of allowed *domains* for the value/time of CPs;

S is the *strategy* to be used to generate an animation;

The domains of values, **D** are such that whenever the user selects a value for a variable from its domain, it is possible to modify A in such a way that **C** gets fulfilled. Note that not only **A**, but **D** changes too: **D** is decided by **C** and the assignment done by the user. Moreover, usually there are many ways to update **A**, that is, there are many solutions of **C** to choose from. The user-defined strategy **S** defines an unique way to generate a solution.

In general, it is a difficult task to find a single solution for a CSP. It is an additional and comparably difficult task to compute the possible domains for each variable. In our case, because of the linearity of the used constraints and a few extra requirements, the solution set is always convex, and hence the domains are always closed intervals. Moreover, these intervals can be computed efficiently by interval propagation [10]. In a nutshell, interval propagation is an iterative process, which narrows an interval, if it can be done, based on the constraint referring to the variable having the interval in question as domain and the domains of the other constrained variables. Because of the special characteristics of the types of allowed constraints, the iterative process produces (finite or infinite) intervals which are exactly the projections of the solution set. If a variable gets a new or modified value (specified by the user or automatically, see below), then the same interval propagation mechanism is used to narrow further or update the domains.

Any solution can be generated by using the information gained on the domains by interval propagation, by iteratively selecting a not yet instantiated variable, assigning a value to it from its domain and updating the other domains. This scheme does not specify in which order variables should be considered, and which of the possible values should be selected for them. The user may specify different strategies to make these choices. E.g. if he wishes to keep the timing of the animation, then he should use the strategy ‘consider time variables first, and select a new value as close as possible to earlier value’. Further alternatives for value selection are: minimum/ maximum/middle of or a random value from the domain;

Note that the order of time variables to be dealt with is not specified yet. He may consider them ‘from left to right’, in order to keep the ‘beginning’ of the animation well-timed.

Often the animator wants to limit the effect of his move to a portion of the animation. He can achieve this by *freezing* the rest of the animation. If a part of the animation is frozen, then the domains, **D**, have to be recomputed, by propagating the ‘frozen’ variable values.

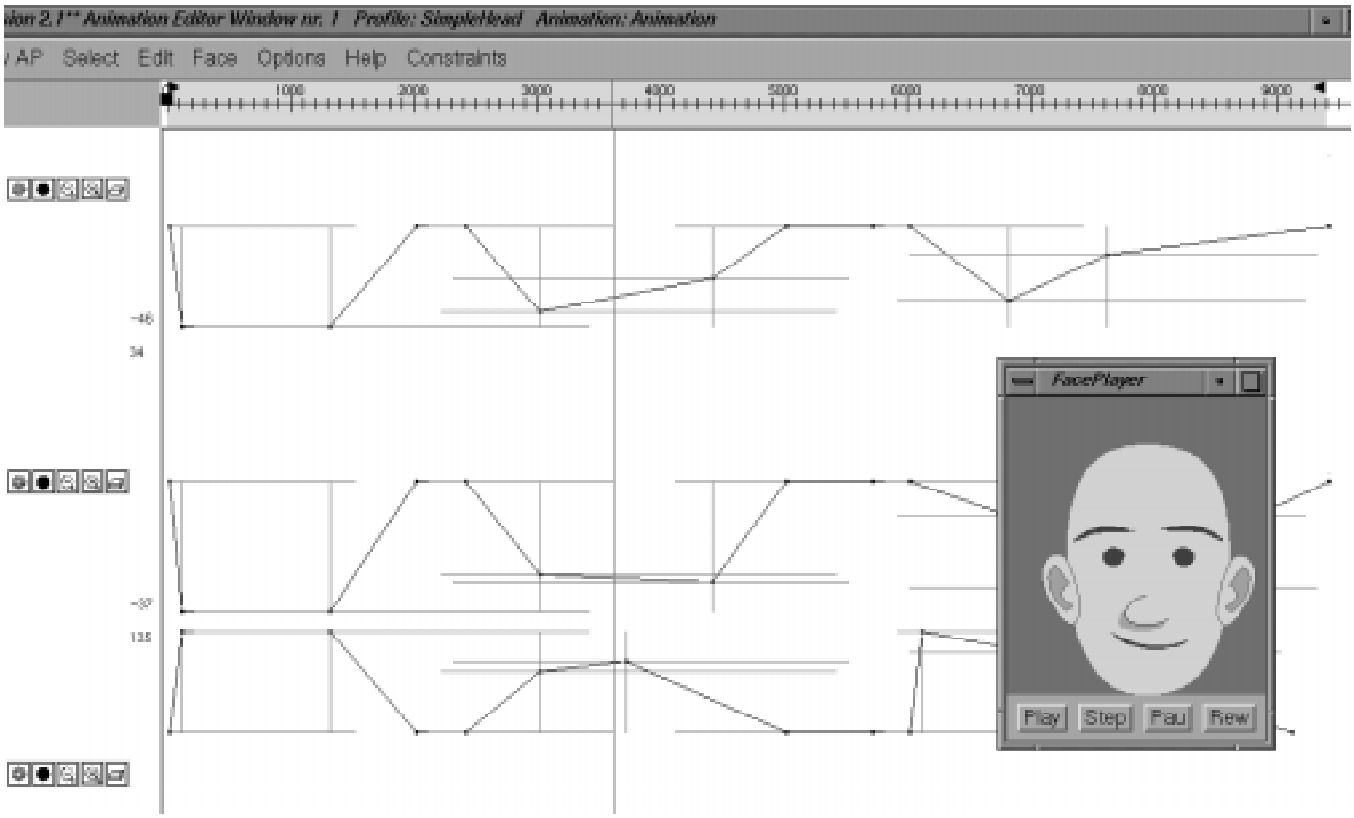

Figure 4. A snapshot of a session when different smiles were produced by re-using an earlier defined 'generic smile' (first instance) by modifying constraints and choosing different solution strategies. The three connected lines in black correspond to 3 of the 4 parameter curves which define the left and right mouth corner x/y positions, the black dots are the CPs which define the functions. The horizontal and vertical lines in grey indicate the possible values for the time and value of the corresponding CP.

4. IMPLEMENTATION

The first version of the constraint-based facial animation editing system has been implemented. It consists of 3 parts:

- Animation Editor: a graphical editor to load/save/edit animations (see [7] for details);
- the Constraint Manipulator to maintain strategies, store and visualize constraints and domains;
- the Solver, to generate domains and animations.

Animation Editor is our earlier made, non-constraint based tool, which was integrated with the Constraint Manipulator (taking care of all constraint-related features) under the name CAnimation Editor. The first two parts are written in Java 1.3 and integrated under a common user interface. The Constraint Manipulator supports the following functionalities (see Figure 4):

- add new constraints, delete/modify old ones;
- show the possible domains for each CP;
- freeze/free CPs;
- specify strategy;
- save and load animations with constraints and strategy;
- insert a piece of animation (with constraints) at a given time;
- specify new types of constraints (other than the provided basic ones).

The Solver consists of the OpAC solver, developed at Nancy [1] and an extra layer, both written in C++. The extra layer communicates with the Constraint Manipulator, implements the execution of the strategy and enforces all kinds of implicit requirements, all by generating constraints for OpAC and sending results back to the Constraint Manipulator. OpAC is a general interval constraint solver. It has been optimized to deal with the special set of basic animation constraints.

5. DISCUSSION

The novelty of our FESINC system lies in the application of constraints to define dynamical facial expressions in an abstract, declarative way, and provide support to generate and use different instances of those expressions. The current implementation already speeds up the process of making a facial animation, and allows the generation of different 'instances' of a dynamical expression defined in terms of constraints. (A session as shown in Figure 4 as well as the resulting animations will be

demonstrated during the talk.) However, several extensions can be thought of, both of conceptual and of technical nature.

The 'building blocks' are not treated in the editor as a unit, the CPs are connected only by constraints. Hence one may delete CPs from a 'smile', possibly destroying by this its intended characteristics. In the next version, the building blocks will be registered. The information will be used not only for improving the technicality of editing, but also as a basis for higher-level, script-based but (but not necessarily deterministic) animation.

Once building blocks can be selected, it would be very useful and interesting to change (several) constraints by intuitive, user-friendly parameters like exaggerate; increase intensity/length/deviation from 'default'. A research issue is to find out how to handle blending and concatenation in the constraint-based framework.

Constraints can be used to express requirements on the general motion characteristics of the face (e.g. symmetrical motion, slow movements). In a next version it will be possible to express such requirements, and the ConstraintManager will generate the appropriate constraints automatically.

It should be tested if real animators find the current set expressive enough. It is also a question if the idea of abstract, constraint-based animation is a maintainable paradigm for animators.

Though facial animation is very different from body animation, some experiments with hand gesture and body animation makes us believe that our framework could be used for them too. Further experiments should be conducted.

6. ACKNOWLEDGMENT

We thank Frederic Goualard for making the OpAC solver available for us, and also for his insightful remarks on our way of using it.

7. REFERENCES

[1] Benhamou, F. Goualard, F. The OpACSolver, University of Nantes, personal communication, 1999-2001.

[2] Cassell, J., Sullivan, J., Prevost, S., Churchill, E. Embodied Conversational Agents, MIT Press, Cambridge, MA, 2000.

[3] Essa, I. Analysis, Interpretationa nd Synthesis of Facial Expressions, MIT Media Lab Perc. Comp. tech. Rep. 272, 1994.

[4] FaceWorks http://www.interface.digital.com/overview/default.html

[5] famous3D http://www.famous3d.com/solutions/production/producer.html

[6] Noma, T., Zhao, L., Badler, N. Design of a virtual human presenter, IEEE Computer Graphics and Applications, Vol. 20. No. 4. 2000, pp. 79-85.

[7] Noot, H. Ruttkay, Zs. CharToon 2.0 Manual, CWI Report INS-R0004, Amsterdam, 2000. Available from http:// www.cwi.nl/FASE/

[8] Parke, F., Waters, K. Computer Facial Animation, AK Peters, Wellesley, MA, 1996.

[9] Poggi, I., Pelachaud, C., De Rosis, F. Eye communication in a conversational 3D synthetic agent, AI Communications, 2000.

[10] Ruttkay, Zs. Constraint-based facial animation, Int. Journal of Constraints, Vol. 6. pp. 85-113, 2001.